**A\*算法求解迷宫问题**

1. **实验目的**
2. **熟悉和掌握启发式搜索的定义，估价函数和算法过程，并且利用A\*算法求解迷宫问题，理解求解流程和搜索顺序。**
3. **实验环境**
4. **硬件环境：微型计算机**
5. **软件环境：Windows操作系统，Java语言，eclipse开发工具**
6. **实验内容与分析**

迷宫问题可以表述为：一个存储每一个点的二维数组（isReachable=true表示点可走，反之不可走，点用（x，y）表示），寻找从某一个给定的其实单元格出发，经由行相邻或列相邻的可通过的单元格，最终可以到达目标单元格的、所走过的单元格序列。在人一个单元格中，都只能看到与它相邻的4个单元格。

A\*算法是一种启发式搜索算法，它不需遍历所有节点，只是利用包含问题启发式信息的评价函数对节点进行排序，使搜索方向朝着最有可能找到目标并产生最优解的方向。它的独特之处是检查最短路径中每个可能的节点时引入了全局信息，对当前节点距终点的距离做出估计，并作为评价节点处于最短路径上的可能性度量。

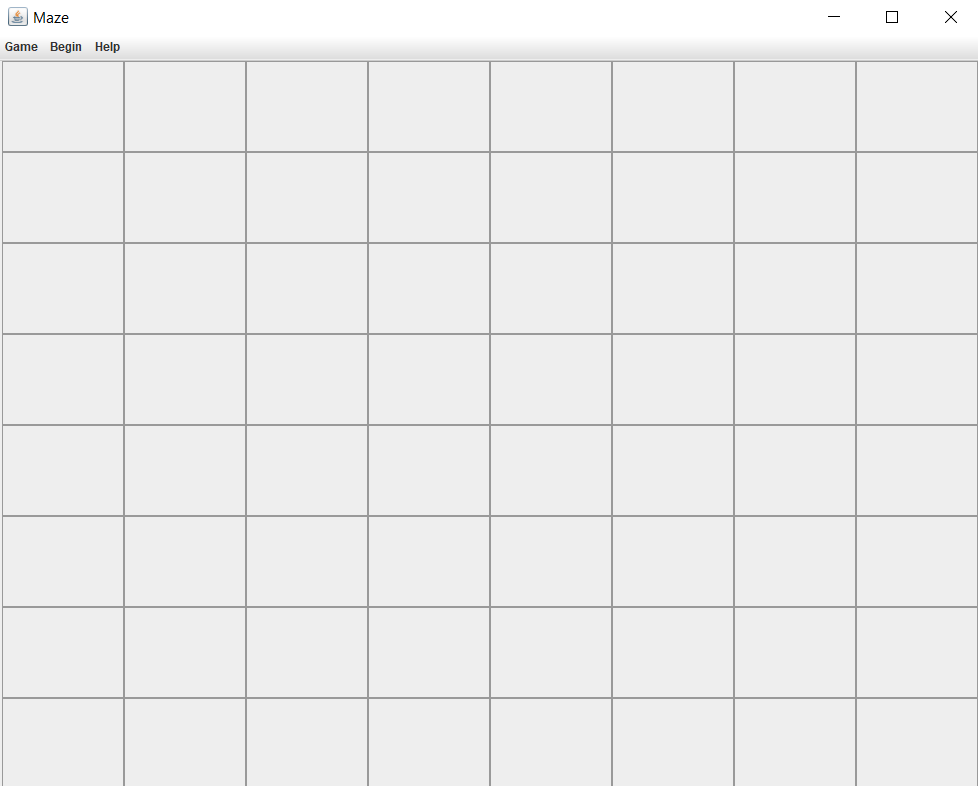
A\*算法中引入了评估函数，评估函数为：f（n）=g（n）+h（n） 其中：n是搜索中遇到的任意状态。g(n)是沿路径从起点到n点的移动耗费。h(n)是对n到目标状态代价的启发式估计。即评估函数f ( n) 是从初始节点到达节点n 处已经付出的代价与节点n 到达目标节点的接近程度估价值的总和。

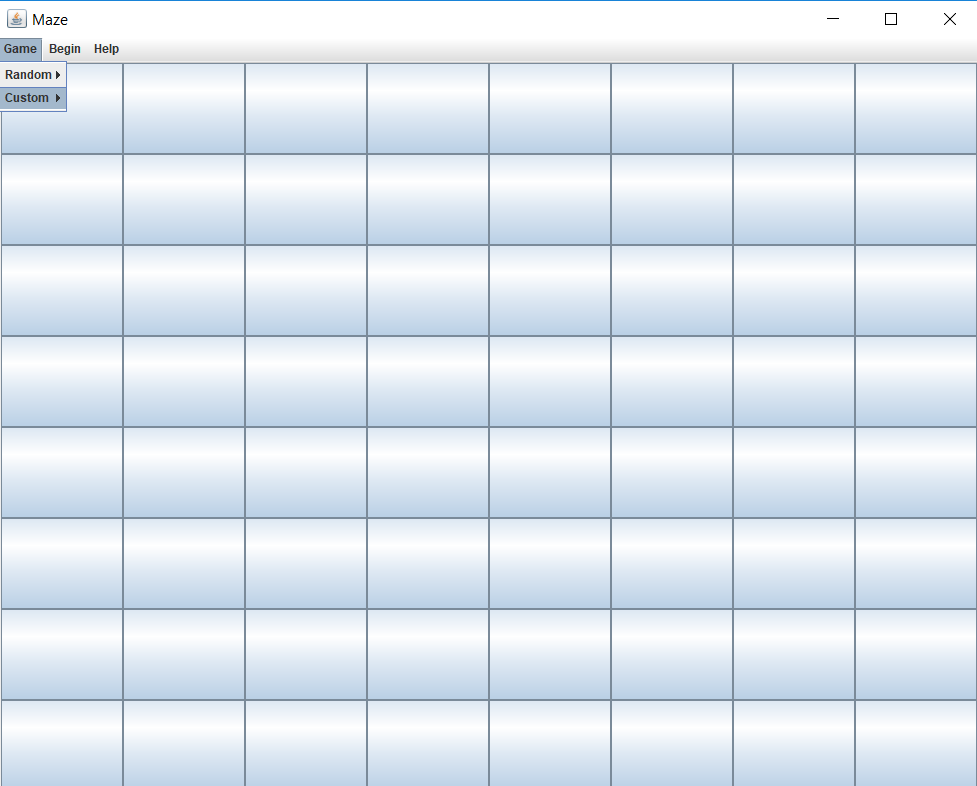
这里我们定义n点到目标点的最小实际距离为h（n）\*，A\*算法要满足的条件为：h（n）<=h（n）\* 。迷宫走的时候只能往上下左右走，每走一步，代价为1，H值可以用不同的方法估算。我们这里使用的方法被称为曼哈顿方法，它计算从当前格到目的格之间水平和垂直的方格的数量总和，即： h（n）=|endPoint.x – n.x|+ |endpoint – n.y|这里endPoint表示迷宫的目标点，n表示当前点，很明显这里h（n）<=h（n）\*。 所以利用f（n）=g（n）+h（n）这种策略，我们可以不断地逼近目标点，从而找到问题的解。

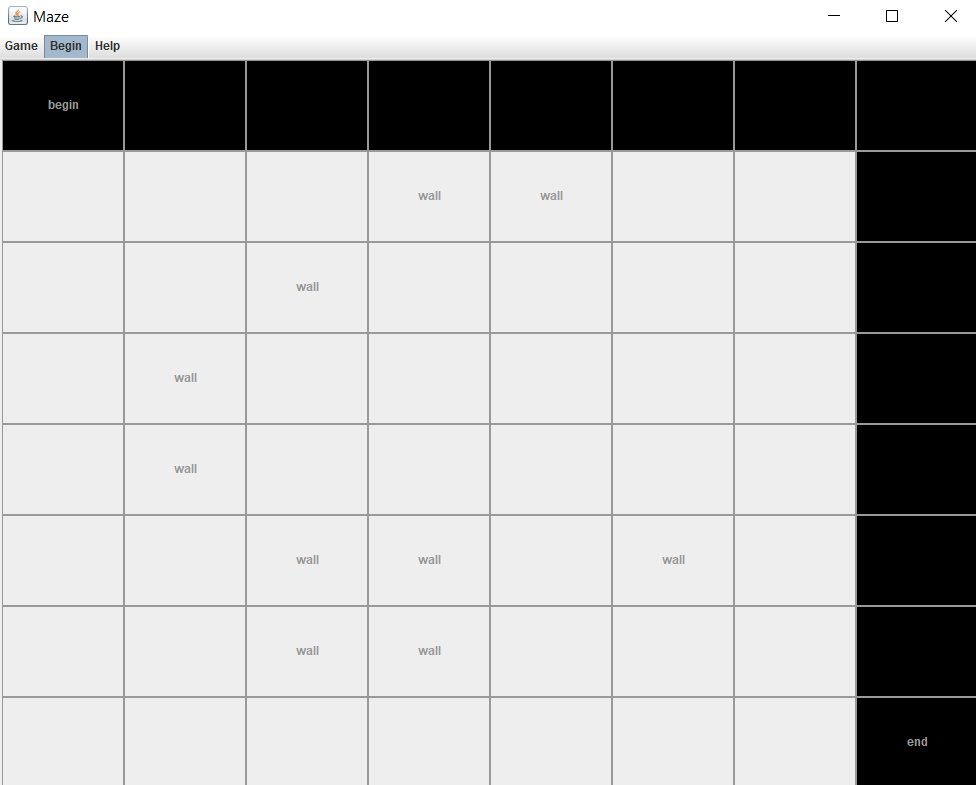
1. **设计实现**

**1类定义：**

1. **MazePoint:** 该类封装的是点信息，包括坐标、是否可以通过以及一些相关操作等等。
2. **GoMaze：该类为开始迷宫的创建以及设定起始点、终点、地图等**
3. **AxSearch：该类封装的是A\*算法的函数，对open表、closed表以及启发函数值得计算操作。**
4. **BoardMaze：该类为迷宫的窗口界面创建**
5. **运行与测试**







**运行程序之后，点击Game->custom，然后就可以启动迷宫，接着可以点击迷宫棋格进行起始点、终点、墙的设置，设置完之后Begin便可以得出寻找终点路线。**

1. **心得体会**

**本实验使用启发式搜索算法的A\*算法，这个过程中，不断对该算法的理解程度纵向加深，对open表、closed表的使用也更加清晰。而在这个迷宫搜索的启发函数，使用的是f**（n）= g（n）+ h（n）.

1. **源代码**

**package wen.maze;**

**import java.awt.EventQueue;**

**public class StartMaze {**

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

**EventQueue.invokeLater(new Runnable() {**

**@Override**

**public void run() {**

**//start the maze**

**GoMaze go = new GoMaze();**

**}**

**});**

**}**

**}**

package wen.maze;

import java.awt.Button;

import java.awt.Color;

import java.awt.GridLayout;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.MouseAdapter;

import java.awt.event.MouseEvent;

import java.util.ArrayList;

import java.util.LinkedList;

import javax.swing.JButton;

import javax.swing.JMenu;

import javax.swing.JMenuBar;

import javax.swing.JOptionPane;

import javax.swing.JPanel;

public class GoMaze {

public MazePoint[][] mazePoint;

public JButton[][] mazeButton;

public LinkedList<MazePoint> close = new LinkedList();

public BoardMaze boardMaze;

JMenu menuHome;

JMenu menuBegin;

JMenu menuHelp;

JMenu menu1;

JMenu menu2;

JPanel buttonPanel;

int size = 8;

int choose = 0;

public GoMaze() {

action();

}

public void action() {

boardMaze = new BoardMaze();

createButton();

createMenu();

menu1.addMouseListener(new MouseAdapter() {

public void mouseClicked(MouseEvent e) {

choose = 1;

System.out.println("Random");

}

});

menu2.addMouseListener(new MouseAdapter() {

public void mouseClicked(MouseEvent e) {

choose = 2;

System.out.println("Custom");

CustomMaze customMaze = new CustomMaze();

mazePoint = customMaze.createMaze(size);

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

for(int j = 0; j < size; j++) {

mazeButton[i][j].setEnabled(true);

mazeButton[i][j].setText("");

}

}

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

for(int j = 0; j < size; j++) {

mazeButton[i][j].addActionListener(new ButtonAction());

}

}

}

});

menuBegin.addMouseListener(new MouseAdapter() {

public void mouseClicked(MouseEvent e) {

int begin = 0;

int end = 0;

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

for(int j = 0; j < size; j++) {

if("begin".equals(mazeButton[i][j].getText())) {

begin++;

}

if("end".equals(mazeButton[i][j].getText())) {

end++;

}

}

}

if(begin != 1 || end != 1)JOptionPane.showMessageDialog(null, "Only can one begin and one end", "error", JOptionPane.INFORMATION\_MESSAGE);

else {

System.out.println("Begin");

switch(choose) {

case 0:

JOptionPane.showMessageDialog(null, "Please choose the 'Random' or 'Custom'!", "warning", JOptionPane.INFORMATION\_MESSAGE);

break;

case 1:

break;

case 2:

AxSearch axSearch = new AxSearch();

close = axSearch.search(mazePoint, size);

System.out.print("\nthe close is: ");

for(MazePoint a : close) {

System.out.print(a.getX() + "" + a.getY() + "" + a.getF() + " ");

}

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

for(int j = 0; j < size; j++) {

mazeButton[i][j].setEnabled(false);

for(MazePoint a : close) {

Color c = new Color(1);

if(i == a.getX() && j == a.getY()) {

mazeButton[i][j].setBackground(c);

break;

}

}

}

}

break;

}

}

}

});

}

public void createButton() {

buttonPanel = new JPanel();

buttonPanel.setLayout(new GridLayout(size, size));

mazeButton = new JButton[size][size];

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

for(int j = 0; j < size; j++) {

mazeButton[i][j] = new JButton();

mazeButton[i][j].setText("");

mazeButton[i][j].setEnabled(false);

buttonPanel.add(mazeButton[i][j]);

}

}

boardMaze.add(buttonPanel);

}

public void createMenu() {

JMenuBar menu = new JMenuBar();

menuHome = new JMenu("Game",true);

menuBegin = new JMenu("Begin", true);

menuHelp = new JMenu("Help", true);

menu1 = new JMenu("Random");

menu2 = new JMenu("Custom");

menuHome.add(menu1);

menuHome.add(menu2);

menu.add(menuHome);

menu.add(menuBegin);

menu.add(menuHelp);

boardMaze.setJMenuBar(menu);

}

class ButtonAction implements ActionListener {

@Override

public void actionPerformed(ActionEvent e) {

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

for(int j = 0; j < size; j++) {

if(e.getSource() == mazeButton[i][j]) {

if("".equals(mazeButton[i][j].getText())) {

mazeButton[i][j].setText("wall");

mazePoint[i][j].setZ(0);

}else if("wall".equals(mazeButton[i][j].getText())) {

mazeButton[i][j].setText("begin");

mazePoint[i][j].setZ(8);

}else if("begin".equals(mazeButton[i][j].getText())) {

mazeButton[i][j].setText("end");

mazePoint[i][j].setZ(4);

}else if("end".equals(mazeButton[i][j].getText())) {

mazeButton[i][j].setText("");

mazePoint[i][j].setZ(1);

}

}

}

}

}

}

}

package wen.maze;

import javax.swing.JFrame;

public class BoardMaze extends JFrame {

public BoardMaze() {

//---set the property of the mazeBoard

setSize(1000, 800);

setTitle("Maze");

setLocationRelativeTo(null);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setVisible(true);

//---

}

}

package wen.maze;

public class CustomMaze {

public MazePoint[][] mazePoint;

public MazePoint[][] createMaze(int size) {

mazePoint = new MazePoint[size][size];

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

for(int j = 0; j < size; j++) {

mazePoint[i][j] = new MazePoint(i, j , 1);

mazePoint[i][j].setF(0);

mazePoint[i][j].setZZ(1);

}

}

return mazePoint;

}

}

package wen.maze;

import java.awt.List;

import java.util.ArrayList;

import java.util.LinkedList;

public class AxSearch {

// public ArrayList<MazePoint> open = new ArrayList();

// public ArrayList<MazePoint> close = new ArrayList();

public MazePoint nodeNow;

public MazePoint nodeTemp;

public MazePoint nodeNext;

public MazePoint begin;

public MazePoint end;

public OpenClose oc;

public LinkedList<MazePoint> open = new LinkedList();

public LinkedList<MazePoint> close = new LinkedList();

public int g;

public int h;

public int f;

public LinkedList<MazePoint> search(MazePoint[][] mazePoint, int size) {

begin = new MazePoint();

end = new MazePoint();

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

for(int j = 0; j < size; j++) {

if(mazePoint[i][j].getZ() == 8)begin = mazePoint[i][j];

if(mazePoint[i][j].getZ() == 4)end = mazePoint[i][j];

System.out.print(mazePoint[i][j].getX() + "" + mazePoint[i][j].getY() + " " + mazePoint[i][j].getZ() + " ");

}

System.out.println();

}

System.out.println("begin : " + begin.getX() + "" + begin.getY());

System.out.println("end : " + end.getX() + "" + end.getY());

g = 0;

h = Math.abs(end.getX() - begin.getX()) + Math.abs(end.getY() - begin.getY());

f = g + h;

begin.setG(g);

begin.setH(h);

begin.setF(f);

nodeNow = begin;

open.add(begin);

int index = 0;

oc = new OpenClose();

while(true) {

if(nodeNow.getX() - 1 >= 0 && mazePoint[nodeNow.getX() - 1][nodeNow.getY()].getZ() != 0 && mazePoint[nodeNow.getX() - 1][nodeNow.getY()].getZZ() != 0 && !oc.isExist(mazePoint[nodeNow.getX() - 1][nodeNow.getY()], open)) {

g = nodeNow.getG() + 1;

h = Math.abs(end.getX() - (nodeNow.getX() - 1)) + Math.abs(end.getY() - nodeNow.getY());

f = g + h;

mazePoint[nodeNow.getX() - 1][nodeNow.getY()].setG(g);

mazePoint[nodeNow.getX() - 1][nodeNow.getY()].setH(h);

mazePoint[nodeNow.getX() - 1][nodeNow.getY()].setF(f);

open.addFirst(mazePoint[nodeNow.getX() - 1][nodeNow.getY()]);

}

if(nodeNow.getX() + 1 < size && mazePoint[nodeNow.getX() + 1][nodeNow.getY()].getZ() != 0 && mazePoint[nodeNow.getX() + 1][nodeNow.getY()].getZZ() != 0 && !oc.isExist(mazePoint[nodeNow.getX() + 1][nodeNow.getY()], open)) {

g = nodeNow.getG() + 1;

h = Math.abs(end.getX() - (nodeNow.getX() + 1)) + Math.abs(end.getY() - nodeNow.getY());

f = g + h;

mazePoint[nodeNow.getX() + 1][nodeNow.getY()].setG(g);

mazePoint[nodeNow.getX() + 1][nodeNow.getY()].setH(h);

mazePoint[nodeNow.getX() + 1][nodeNow.getY()].setF(f);

open.addFirst(mazePoint[nodeNow.getX() + 1][nodeNow.getY()]);

}

if(nodeNow.getY() - 1 >= 0 && mazePoint[nodeNow.getX()][nodeNow.getY() - 1].getZ() != 0 && mazePoint[nodeNow.getX()][nodeNow.getY() - 1].getZZ() != 0 && !oc.isExist(mazePoint[nodeNow.getX()][nodeNow.getY() - 1], open)) {

g = nodeNow.getG() + 1;

h = Math.abs(end.getX() - (nodeNow.getX())) + Math.abs(end.getY() - (nodeNow.getY() - 1));

f = g + h;

mazePoint[nodeNow.getX()][nodeNow.getY() - 1].setG(g);

mazePoint[nodeNow.getX()][nodeNow.getY() - 1].setH(h);

mazePoint[nodeNow.getX()][nodeNow.getY() - 1].setF(f);

open.addFirst(mazePoint[nodeNow.getX()][nodeNow.getY() - 1]);

}

if(nodeNow.getY() + 1 < size && mazePoint[nodeNow.getX()][nodeNow.getY() + 1].getZ() != 0 && mazePoint[nodeNow.getX()][nodeNow.getY() + 1].getZZ() != 0 && !oc.isExist(mazePoint[nodeNow.getX()][nodeNow.getY() + 1], open)) {

g = nodeNow.getG() + 1;

h = Math.abs(end.getX() - (nodeNow.getX())) + Math.abs(end.getY() - (nodeNow.getY() + 1));

f = g + h;

mazePoint[nodeNow.getX()][nodeNow.getY() + 1].setG(g);

mazePoint[nodeNow.getX()][nodeNow.getY() + 1].setH(h);

mazePoint[nodeNow.getX()][nodeNow.getY() + 1].setF(f);

open.addFirst(mazePoint[nodeNow.getX()][nodeNow.getY() + 1]);

}

for(MazePoint a : open) {

if(a.getX() == nodeNow.getX() && a.getY() == nodeNow.getY()) {

mazePoint[nodeNow.getX()][nodeNow.getY()].setZZ(0);

open.remove(open.indexOf(a));

if(!oc.isExist(a, close))close.add(a);

break;

}

}

System.out.println("nodeNow: " + nodeNow.getX() + "" + nodeNow.getY());

int maxIndex = 0;

int maxF = 100;

for(MazePoint e : open) {

if(e.getF() < maxF) {

maxF = e.getF();

maxIndex = open.indexOf(e);

}

}

if(nodeNow.getX() == end.getX() && nodeNow.getY() == end.getY()) {

System.out.println("Find the endPoint");

break;

}

if(open.isEmpty() && nodeNow.getX() != end.getX() && nodeNow.getY() != end.getY()) {

System.out.println("The maze is closed");

break;

}

nodeNow = open.get(maxIndex);

System.out.println("nodeNext: " + nodeNow.getX() + "" + nodeNow.getY());

if(index == 1000)break;

}

System.out.print("\nthe open is: ");

for(MazePoint e : open) {

System.out.print(e.getX() + "" + e.getY() + "" + e.getF() + " ");

}

System.out.println();

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

for(int j = 0; j < size; j++) {

System.out.print(mazePoint[i][j].getF() + " ");

}

System.out.println();

}

return close;

}

}

package wen.maze;

public class MazePoint {

private int x;

//co-ordinate x

private int y;

//co-ordinate y

private int z;

//right of way the x,y

private int g;

//on the distance of the begining point

private int h;

private int f;

private int zz;

public MazePoint() {}

public MazePoint(int x, int y, int z) {

this.x = x;

this.y = y;

this.z = z;

}

public void setX(int x) {

this.x = x;

}

public int getX() {

return x;

}

public void setY(int y) {

this.y = y;

}

public int getY() {

return y;

}

public void setZ(int z) {

this.z = z;

}

public int getZ() {

return z;

}

public void setG(int g) {

this.g = g;

}

public int getG() {

return g;

}

public void setH(int h) {

this.h= h;

}

public int getH() {

return h;

}

public void setF(int f) {

this.f = f;

}

public int getF() {

return f;

}

public void setZZ(int zz) {

this.zz = zz;

}

public int getZZ() {

return zz;

}

}