**Федеральное агентство связи**

**Ордена Трудового Красного Знамени**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский технический университет связи и информатики»**

Кафедра Математической кибернетики и информационных технологий

**Отчет по лабораторной работе**

по предмету «СИАОД»

на тему:

«Сетевые алгоритмы. Эвристические алгоритмы поиска путей»

Выполнил: студент группы

Клюшкин Дмитрий Алексеевич

Руководитель:

Кутейников Иван Александрович

Москва 2020

*Цель работы:* Реализовать алгоритм поиска кратчайшего расстояния между двумя клетками в соответствии с вариантом и алгоритм A\*. Сравнить результаты работы данного алгоритма с алгоритмом A\*.

*Вариант:* 14. Ортогональный лучевой алгоритм

*Ход работы:*

*Код программы:*

**Ортогональный лучевой алгоритм**

import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.awt.event.MouseEvent;  
import java.awt.event.MouseListener;  
  
public class RayFind {  
 private JMapCell[][] mapCells;  
private int width;  
private int height;  
private Location startLoc;  
private Location finishLoc;  
  
 private class MapCellHandler implements MouseListener  
 {private boolean modifying;  
private boolean makePassable;  
public void mousePressed(MouseEvent e)  
 {  
 modifying = true;  
  
 JMapCell cell = (JMapCell) e.getSource();  
  
 makePassable = !cell.isPassable();  
  
 cell.setPassable(makePassable);  
 }  
public void mouseReleased(MouseEvent e)  
 {  
 modifying = false;  
 }  
public void mouseEntered(MouseEvent e)  
 {  
 if (modifying)  
 {  
 JMapCell cell = (JMapCell) e.getSource();  
 cell.setPassable(makePassable);  
 }  
 }  
public void mouseExited(MouseEvent e)  
 {  
 // This one we ignore.  
 }  
public void mouseClicked(MouseEvent e)  
 {  
 // And this one too.  
 }  
 }  
  
 public RayFind(int w, int h) {  
 if (w <= 0)  
 throw new IllegalArgumentException("w must be > 0; got " + w);  
  
 if (h <= 0)  
 throw new IllegalArgumentException("h must be > 0; got " + h);  
  
 width = w;  
 height = h;  
  
 startLoc = new Location(2, h / 2);  
 finishLoc = new Location(w - 3, h / 2);  
  
 initGUI();  
 }  
  
 private void initGUI()  
 {  
 JFrame frame = new JFrame("Pathfinder");  
 frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 Container contentPane = frame.getContentPane();  
  
 contentPane.setLayout(new BorderLayout());  
  
 // Use GridBagLayout because it actually respects the preferred size  
 // specified by the components it lays out.  
  
 GridBagLayout gbLayout = new GridBagLayout();  
 GridBagConstraints gbConstraints = new GridBagConstraints();  
 gbConstraints.fill = GridBagConstraints.*BOTH*;  
 gbConstraints.weightx = 1;  
 gbConstraints.weighty = 1;  
 gbConstraints.insets.set(0, 0, 1, 1);  
  
 JPanel mapPanel = new JPanel(gbLayout);  
 mapPanel.setBackground(Color.*GRAY*);  
  
 mapCells = new JMapCell[width][height];  
  
 MapCellHandler cellHandler = new MapCellHandler();  
  
 for (int y = 0; y < height; y++)  
 {  
 for (int x = 0; x < width; x++)  
 {  
 mapCells[x][y] = new JMapCell();  
  
 gbConstraints.gridx = x;  
 gbConstraints.gridy = y;  
  
 gbLayout.setConstraints(mapCells[x][y], gbConstraints);  
  
 mapPanel.add(mapCells[x][y]);  
 mapCells[x][y].addMouseListener(cellHandler);  
 }  
 }  
  
 contentPane.add(mapPanel, BorderLayout.*CENTER*);  
  
 JButton findPathButton = new JButton("Find Path");  
 findPathButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) { findAndShowPath(); }  
 });  
  
 contentPane.add(findPathButton, BorderLayout.*SOUTH*);  
  
 frame.pack();  
 frame.setVisible(true);  
  
 mapCells[startLoc.xCoord][startLoc.yCoord].setEndpoint(true);  
 mapCells[finishLoc.xCoord][finishLoc.yCoord].setEndpoint(true);  
 }  
  
 private void findAndShowPath() {  
 Location pos = startLoc;  
 boolean activ=true;  
  
 System.*out*.println(mapCells[0][0].isPassable());  
  
 while (activ){  
 double minDist = Double.*MAX\_VALUE*;  
 Location newLoc=pos;  
 for (int i=-1;i<=1;i+=2){  
 if (mapCells[newLoc.xCoord+i][newLoc.yCoord].isPassable() && !mapCells[newLoc.xCoord+i][newLoc.yCoord].path) {  
 double dist = newLoc.addCoord(i, 0).gipotenuza(finishLoc.xCoord, finishLoc.yCoord);  
 if (dist < minDist) {  
 minDist = dist;  
 pos = newLoc.addCoord(i, 0);  
 }  
 }  
 if (mapCells[newLoc.xCoord][newLoc.yCoord+i].isPassable() && !mapCells[newLoc.xCoord][newLoc.yCoord+i].path) {  
 double dist = newLoc.addCoord(0, i).gipotenuza(finishLoc.xCoord, finishLoc.yCoord);  
 if (dist < minDist) {  
 minDist = dist;  
 pos = newLoc.addCoord(0, i);  
 }  
 }  
 }  
  
 mapCells[pos.xCoord][pos.yCoord].setPath(true);  
 if (minDist==Double.*MAX\_VALUE*) activ=false;  
 if (pos.equals(finishLoc)) activ=false;  
 }  
  
 }  
  
 public static void main(String[] args) {  
 RayFind rayFind = new RayFind(40,30);  
 }  
  
  
}

**A\***

private void findAndShowPath()  
{  
 // Create a Map2D object containing the current state of the user input.  
  
 Map2D map = new Map2D(width, height);  
 map.setStart(startLoc);  
 map.setFinish(finishLoc);  
   
 for (int y = 0; y < height; y++)  
 {  
 for (int x = 0; x < width; x++)  
 {  
 mapCells[x][y].setPath(false);  
  
 if (mapCells[x][y].isPassable())  
 map.setCellValue(x, y, 0);  
 else  
 map.setCellValue(x, y, Integer.*MAX\_VALUE*);  
 }  
 }  
   
 // Try to compute a path. If one can be computed, mark all cells in the  
 // path.  
   
 Waypoint wp = AStarPathfinder.*computePath*(map);  
   
 while (wp != null)  
 {  
 Location loc = wp.getLocation();  
 mapCells[loc.xCoord][loc.yCoord].setPath(true);  
   
 wp = wp.getPrevious();  
 }  
}

public static Waypoint computePath(Map2D map)  
{  
 AStarState state = new AStarState(map);  
 Location finishLoc = map.getFinish();  
  
 Waypoint start = new Waypoint(map.getStart(), null);  
 start.setCosts(0, *estimateTravelCost*(start.getLocation(), finishLoc));  
 state.addOpenWaypoint(start);  
  
 Waypoint finalWaypoint = null;  
 boolean foundPath = false;  
   
 while (!foundPath && state.numOpenWaypoints() > 0)  
 {  
 Waypoint best = state.getMinOpenWaypoint();  
 if (best.getLocation().equals(finishLoc))  
 {  
 finalWaypoint = best;  
 foundPath = true;  
 }  
 *takeNextStep*(best, state);  
 state.closeWaypoint(best.getLocation());  
 }  
   
 return finalWaypoint;  
}

|  |  |  |
| --- | --- | --- |
|  | **Ортогональный лучевой алгоритм** | **A\*** |
| Рисунок |  |  |
| Время в мс | **0.2081** | **1.8412** |
| Кол-во клеток | **10х10** | **10х10** |

|  |  |  |
| --- | --- | --- |
|  | **Ортогональный лучевой алгоритм** | **A\*** |
| Рисунок |  |  |
| Время в мс | **0.3263** | **5.8181** |
| Кол-во клеток | **20х20** | **20х20** |

|  |  |  |
| --- | --- | --- |
|  | **Ортогональный лучевой алгоритм** | **A\*** |
| Рисунок |  |  |
| Время в мс | **0.2652** | **6.3785** |
| Кол-во клеток | **30х30** | **30х30** |

|  |  |  |
| --- | --- | --- |
|  | **Ортогональный лучевой алгоритм** | **A\*** |
| Рисунок |  |  |
| Время в мс | **0.1847** | **2.584** |
| Кол-во клеток | **10х10** | **10х10** |

Вывод: построил регулярную сеть в виде клеток с заданной стоимостью прохождения. Реализовал алгоритм поиска кратчайшего расстояния между двумя клетками в соответствии с вариантом и алгоритм A\*. При сравнении скорости работы алгоритмов, можно сделать вывод, что алгоритм A\* лучше справляется со своей задачей, но затрачивает больше времени, а ортогональный лучевой алгоритм быстрее, но менее точный.