**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

Митрохин Ярослав Игоревич

Руководитель:

Кутейников Иван Александрович

Москва 2020

*Цель работы:* построить регулярную сеть в виде клеток с заданной стоимостью прохождения. Реализовать алгоритм поиска кратчайшего расстояния между двумя клетками в соответствии с вариантом и алгоритм А\*. Сравнить результаты работы данного алгоритма с алгоритмом А\*. Разработать графический интерфейс пользователя с визуализацией поля и отображением кратчайшего расстояния между задаваемыми пользователем клетками и всеми клетками, которые были рассмотрены в рамках выбора кратчайшего маршрута.

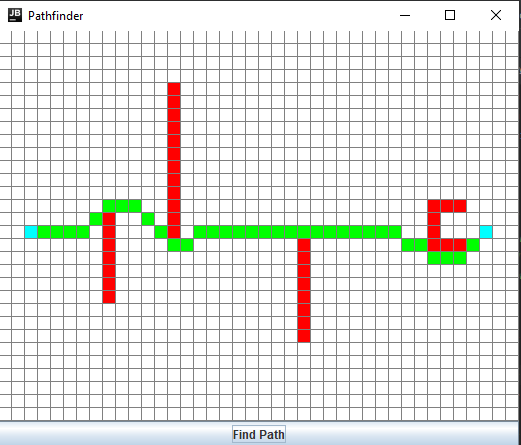
По результатам работы проанализировать временную сложность работы заданного алгоритма в зависимости от числа клеток.

*Ход работы:*

*Код программы(А\*):*

private void findAndShowPath()  
{  
 // Create a Map2D object containing the current state of the user input.  
 double time = System.*nanoTime*();  
 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();  
 }  
 System.*out*.println("Time: "+(System.*nanoTime*()-time)/1000000);  
}

*Результат программы:*

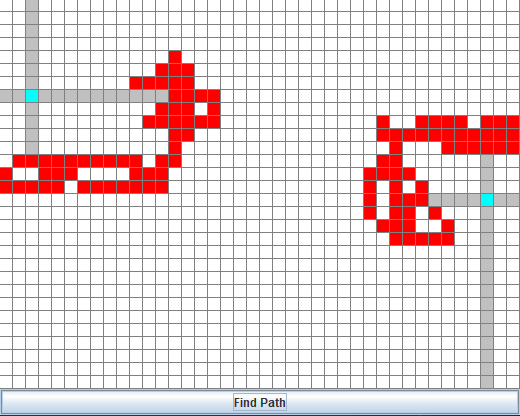
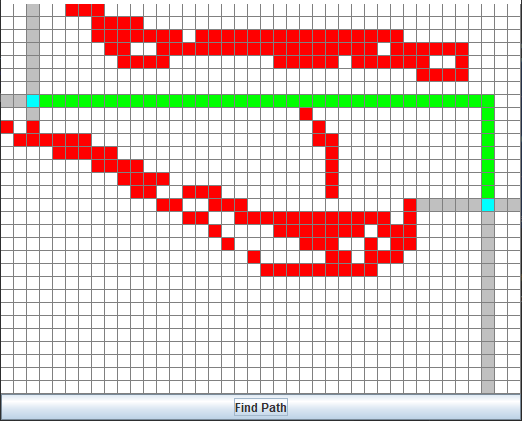
**

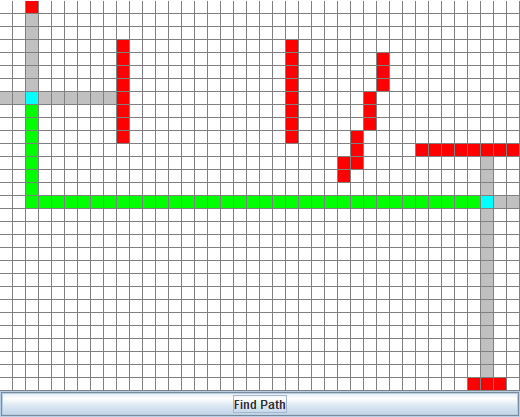
*Код программы(Четырехлучевой алгоритм):*

double time = System.*nanoTime*();  
 /\*startLoc = new Location(2, 7);  
 finishLoc = new Location(w - 3, h / 2);\*/  
 for (int x = 0; x<finishLoc.xCoord-1;x++) {  
 if (mapCells[startLoc.xCoord + x][startLoc.yCoord].isPassable()) {  
 mapCells[startLoc.xCoord + x][startLoc.yCoord].setPath(true);  
 }  
 else  
 {  
 for (int x2 = 0; x2<finishLoc.xCoord-1;x2++) {  
 if (mapCells[startLoc.xCoord + x2][startLoc.yCoord].isPassable()) {  
 mapCells[startLoc.xCoord + x2][startLoc.yCoord].setPath(false);  
 mapCells[startLoc.xCoord + x2][startLoc.yCoord].setNope(true);  
 }  
 else break;  
 }  
 break;  
 }  
 }  
 for (int y = 0;y<Math.*abs*(startLoc.yCoord-finishLoc.yCoord);y++) {  
 if (mapCells[finishLoc.xCoord][finishLoc.yCoord - y].isPassable())  
 mapCells[finishLoc.xCoord][finishLoc.yCoord - y].setPath(true);  
 else  
 {  
 for (int y2 = 0;y2<Math.*abs*(startLoc.yCoord-finishLoc.yCoord);y2++) {  
 if (mapCells[finishLoc.xCoord][finishLoc.yCoord - y2].isPassable()){  
 mapCells[finishLoc.xCoord][finishLoc.yCoord - y2].setPath(false);  
 mapCells[finishLoc.xCoord][finishLoc.yCoord - y2].setNope(true);  
 }  
 else break;  
 }  
 break;  
 }  
 }  
 for (int y1 = 0; y1<Math.*abs*(startLoc.yCoord-finishLoc.yCoord)+1;y1++) {  
 if (mapCells[startLoc.xCoord][startLoc.yCoord + y1].isPassable())  
 mapCells[startLoc.xCoord][startLoc.yCoord + y1].setPath(true);  
 else  
 {  
 for (int y3 = 0; y3<Math.*abs*(startLoc.yCoord-finishLoc.yCoord)+1;y3++) {  
 if (mapCells[startLoc.xCoord][startLoc.yCoord + y3].isPassable()){  
 mapCells[startLoc.xCoord][startLoc.yCoord + y3].setPath(false);  
 mapCells[startLoc.xCoord][startLoc.yCoord + y3].setNope(true);  
 }  
 else break;  
 }  
 break;  
 }  
 }  
  
 for (int x1 =0; x1<finishLoc.xCoord-startLoc.xCoord;x1++) {  
 if (mapCells[finishLoc.xCoord - x1][finishLoc.yCoord].isPassable())  
 mapCells[finishLoc.xCoord - x1][finishLoc.yCoord].setPath(true);  
 else  
 {  
 for (int x3 =0; x3<finishLoc.xCoord-startLoc.xCoord;x3++) {  
 if (mapCells[finishLoc.xCoord - x3][finishLoc.yCoord].isPassable()){  
 mapCells[finishLoc.xCoord - x3][finishLoc.yCoord].setPath(false);  
 mapCells[finishLoc.xCoord - x3][finishLoc.yCoord].setNope(true);  
 }  
 else break;  
 }  
 break;  
 }  
 }  
 for (int y4 = 0;y4<startLoc.yCoord+1;y4++)  
 {  
 mapCells[startLoc.xCoord][startLoc.yCoord - y4].setNope(true);  
 }  
 for (int x4 = 0;x4<startLoc.xCoord+1;x4++)  
 {  
 mapCells[startLoc.xCoord-x4][startLoc.yCoord].setNope(true);  
 }  
 for (int y5 = 0; y5<height-finishLoc.yCoord;y5++)  
 {  
 mapCells[finishLoc.xCoord][finishLoc.yCoord+y5].setNope(true);  
 }  
 for (int x5 = 0; x5<width-finishLoc.xCoord;x5++)  
 {  
 mapCells[finishLoc.xCoord+x5][finishLoc.yCoord].setNope(true);  
 }  
 System.*out*.println("Time: "+(System.*nanoTime*()-time)/1000000);  
}

*Результат программы:*

*При невозможности встречи двух разноименных лучей, лучи выходящие из точек входа и выхода окрашиваются в серый цвет. Наикротчайший путь из начальной точки в конечную выделяется зеленым цветом. Из каждой точки выходят по 4 луча, при том если луч достигает края координатной сетки, то он тоже не рассчитывается и окрашивается в серый цвет. Алгоритм основан на соединении двух лучей выходящих из начальной и конечной точки. Алгоритм А\* рассматривает стоимость всех близлижайших клеток и выбирает «лучшую» из них, тем самым проводя цветную линию и переходя на следующую точку. При невозможности прохождения луча от начальной точки к другой происходит ошибка. Примеры выполнения четырехлучевого алгоритма:*

**

**

*Таблица сравнения времени работы алгоритмов:*

|  |  |  |  |
| --- | --- | --- | --- |
| Сложность нахождения кратчайшего пути | Простая | Средняя | Сложная |
| А\* | 80.49 | 83.52 | 92.21 |
| Четырехлучевой алгоритм | 2.64 | 9.23 | 14.93 |

*Выводы:* построил регулярную сеть в виде клеток с заданной стоимостью прохождения. Реализовал алгоритм поиска кратчайшего расстояния между двумя клетками в соответствии с вариантом и алгоритм А\*. Сравнил результаты работы данного алгоритма с алгоритмом А\*. Разработал графический интерфейс пользователя с визуализацией поля и отображением кратчайшего расстояния между задаваемыми пользователем клетками и всеми клетками, которые были рассмотрены в рамках выбора кратчайшего маршрута. По результатам работы проанализировал временную сложность работы заданного алгоритма в зависимости от числа клеток. Сравнил время работы алгоритма А\* и четырехлучевого алгоритма, сделал выводы, что А\* работает медленнее в силу того, что он более нагруженный, чем четырехлучевой алгоритм. Был реализован интерфейс, где пользователь сам может выбрать, где расставить «препятствия». Начальная и конечная точка расставляются вручную в программе. Программа начинает выполнение алгоритмов по нажатию на кнопку «Find Path».