**Федеральное агентство связи**

**Ордена трудового Красного Знамени федеральное государственное бюджетное**

**образовательное учреждение высшего образования**

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

Кафедра Математическая кибернетика и информационные технологии

Отчет по лабораторной работе №3

«Алгоритм A\* («A star»)»

по дисциплине «Объектно-ориентированное программирование»

Выполнила студентка

группы БУТ 1952

Рюмина К.С.

Москва, 2020

Для выполнения данной лабораторной работы, необходимо было скачать исходные файлы такие как, например, класс представления координат перемещения и поиска пути алгоритма A\*, приложения, которое обеспечивает редактируемый вид 2D-карты и т.д.

**Задания**

1. Обеспечить реализацию методов equals() и hashCode() для класса Location.
2. Добавить поля для «открытых» и «закрытых» вершин в класс AStarState и реализовать методы numOpenWaypoints, getMinOpenWaypoint, addOpenWaypoint, isLocationClosed,closeWaypoint.

Для выполнения задания был подготовлен класс Location (рис.1) для совместного использования с классами Java.

Также были внесены изменения в класс AStarState (рис.2).

public class Lab1 {  
 public static void main(String[] args) {  
 Point3d[] points = new Point3d[3];  
 for (int i = 0; i < args.length; i = i + 3){  
 double x = Double.parseDouble(args[i]);  
 double y = Double.parseDouble(args[i+1]);  
 double z = Double.parseDouble(args[i+2]);  
 points[i / 3] = new Point3d(x, y, z);  
 }  
 if ( (points[0].equals(points[1])) | (points[0].equals(points[2])) | (points[1].equals(points[2])) ){  
 System.out.println("Some points are equal, there's no triangle");  
 }  
 else{  
 System.out.printf("Area of triangle: %.2f \n", computeArea(points[0], points[1], points[2]));  
 }  
  
 }  
 public static double computeArea(Point3d first, Point3d second, Point3d third) {  
 // Возвращение треугольника  
 double a = first.distanseTo(second);  
 double b = first.distanseTo(third);  
 double c = second.distanseTo(third);  
 double p = (a + b + c) / 2.0;  
 double s = p \* (p - a) \* (p - b) \* (p - c);  
 return Math.sqrt(s);  
 }  
  
}

Рисунок 1 – Исходный код Location

import java.util.HashMap;  
  
*/\*\*  
 \* This class stores the basic state necessary for the A\* algorithm to compute a  
 \* path across a map. This state includes a collection of "open waypoints" and  
 \* another collection of "closed waypoints." In addition, this class provides  
 \* the basic operations that the A\* pathfinding algorithm needs to perform its  
 \* processing.  
 \*\*/*public class AStarState  
{  
 */\*\* This is a reference to the map that the A\* algorithm is navigating. \*\*/* private Map2D map;  
  
 public HashMap<Location, Waypoint> openPoint = new HashMap<Location, Waypoint>();  
 public HashMap<Location, Waypoint> closedPoint = new HashMap<Location, Waypoint>();  
  
  
 */\*\*  
 \* Initialize a new state object for the A\* pathfinding algorithm to use.  
 \*\*/* public AStarState(Map2D map)  
 {  
 if (map == null)  
 throw new NullPointerException("map cannot be null");  
  
 this.map = map;  
 }  
  
 */\*\* Returns the map that the A\* pathfinder is navigating. \*\*/* public Map2D getMap()  
 {  
 return map;  
 }  
  
 */\*\*  
 \* This method scans through all open waypoints, and returns the waypoint  
 \* with the minimum total cost. If there are no open waypoints, this method  
 \* returns <code>null</code>.  
 \*\*/* public Waypoint getMinOpenWaypoint()  
 {  
 System.out.println("Getting min open waypoint");  
 Waypoint minOpenWaypoint = null;  
 float min = Float.MAX\_VALUE;  
 if (openPoint.size() != 0){  
 for (Waypoint value : openPoint.values()){  
 float cost = value.getTotalCost();  
 if (cost < min){  
 min = cost;  
 minOpenWaypoint = value;  
 }  
 }  
 System.out.println("Got min open waypoint: " + minOpenWaypoint);  
 return minOpenWaypoint;  
 }  
 else  
 return null;  
 }  
  
 */\*\*  
 \* This method adds a waypoint to (or potentially updates a waypoint already  
 \* in) the "open waypoints" collection. If there is not already an open  
 \* waypoint at the new waypoint's location then the new waypoint is simply  
 \* added to the collection. However, if there is already a waypoint at the  
 \* new waypoint's location, the new waypoint replaces the old one <em>only  
 \* if</em> the new waypoint's "previous cost" value is less than the current  
 \* waypoint's "previous cost" value.  
 \*\*/* public boolean addOpenWaypoint(Waypoint newWP)  
 {  
 boolean isInSet = openPoint.containsKey(newWP.getLocation());  
 if (!isInSet){  
 System.out.println("New open waypoint");  
 openPoint.put(newWP.getLocation(), newWP);  
 return true;  
 }  
 else{  
 Waypoint prevWaypoint = openPoint.get(newWP.getLocation());  
 if (prevWaypoint.getPreviousCost() > newWP.getPreviousCost()){  
 System.out.println("Better open waypoint");  
 openPoint.put(newWP.getLocation(), newWP);  
 return true;  
 }  
 }  
 System.out.println("Open waypoint wasn't added");  
 return false;  
 }  
  
  
 */\*\* Returns the current number of open waypoints. \*\*/* public int numOpenWaypoints()  
 {  
 System.out.println("Size of openPoint: " + openPoint.size());  
 return openPoint.size();  
 }  
  
  
 */\*\*  
 \* This method moves the waypoint at the specified location from the  
 \* open list to the closed list.  
 \*\*/* public void closeWaypoint(Location loc)  
 {  
 System.out.println("Closing waypoint. Size of closedPoint: " + closedPoint.size());  
 Waypoint p = openPoint.remove(loc);  
 closedPoint.put(loc, p);  
 }  
  
 */\*\*  
 \* Returns true if the collection of closed waypoints contains a waypoint  
 \* for the specified location.  
 \*\*/* public boolean isLocationClosed(Location loc)  
 {  
 System.out.println("Is waypoint closed? " + closedPoint.containsKey(loc));  
 return closedPoint.containsKey(loc);  
 }  
}

Рисунок 2 – Исходный код AStarState

При запуске программы (рис.3) открывается окно с полями и двумя вершинами для поиска пути между ними (рис.4).



Рисунок 3 – Запуск программы

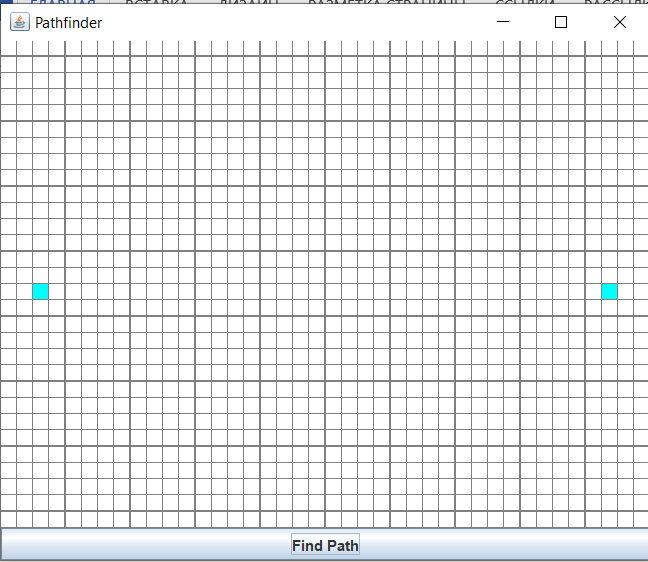


Рисунок 4 - окно программы

После нажатия на кнопку «Find Path» производится поиск пути между вершинами без препятсткий(рис.5), а после, также нажав ту же кнопку – с препятствиями, заранее выбрав какие-либо ячейки для появления этих препятствий (рис.6).

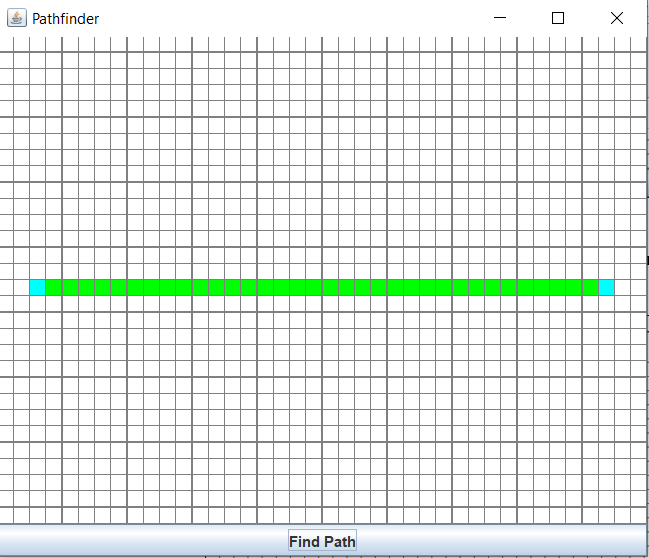


Рисунок 5 – поиск пути без препятствий

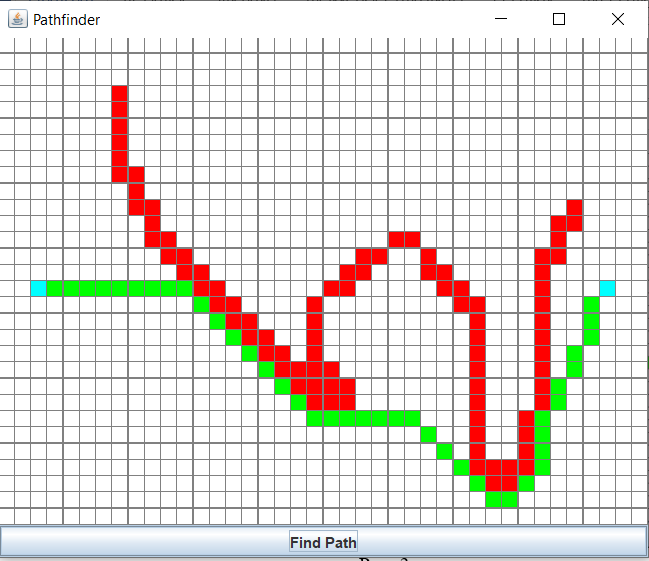


Рисунок 6 – поиск пути с препятствиями

**Заключение.**

В ходе выполнения данной работы я изучила работу HashMap и узнала необходимость изменения методов hashCode и equals для классов.