ФЕДЕРАЛЬНОЕ АГЕНСТВО СВЯЗИ

Ордена Трудового Красного Знамени

федеральное государственное бюджетное

образовательное учреждение высшего образования

«Московский Технический Университет Связи и Информатики» (МТУСИ)

Кафедра Информатики

Лабораторная работа по технологиям программирования №3

«Java-Я-звезда!»»

Выполнил:

Студент 2 курса

Группы БСТ-1602

Тагунов Виталий

Вариант №16

Москва, 2018

**Цель работы:** дописать программу, которая находит кратчайший путь из заданной точки А в точку Б, обходя при этом заблокированные заранее клетки.

**Анализ предметной области:** в этой работе я использовал пакет разработчика Java Development Kit. Для написания кода была использована программа Блокнот, для того, чтобы можно было без проблем использовать виртуальную среду разработки Java и его компилятор.

**Программа:**

**Location.java:**

/\*\*

\* This class represents a specific location in a 2D map. Coordinates are

\* integer values.

\*\*/

public class Location

{

/\*\* X coordinate of this location. \*\*/

public int xCoord;

/\*\* Y coordinate of this location. \*\*/

public int yCoord;

/\*\* Creates a new location with the specified integer coordinates. \*\*/

public Location(int x, int y)

{

xCoord = x;

yCoord = y;

}

/\*\* Creates a new location with coordinates (0, 0). \*\*/

public Location()

{

this(0, 0);

}

public boolean equals(Object obj)

{

Location loc = (Location) obj;

return (xCoord == loc.xCoord && yCoord == loc.yCoord);

}

public int hashCode()

{

int result = 0;

result = xCoord;

result = result\*31+ yCoord;

return result;

}

}

Методы:

**Equals –** равенство 2-х элементов.

**hashCode()** – получение хэш-кода для установки размерности поля.

**AStarState.java:**

**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;**

**private HashMap<Location, Waypoint> open;**

**private HashMap<Location, Waypoint> close;**

**/\*\***

**\* 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;**

**open = new HashMap<Location, Waypoint>();**

**close = new HashMap<Location, Waypoint>();**

**}**

**/\*\* 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()**

**{**

**Waypoint[] a=new Waypoint[]{};**

**a=openPoints.values().toArray(a);**

**Waypoint minP=a[0];**

**double min=minP.getTotalCost();**

**for(int i=1;i<a.length; i++)**

**{**

**double sum = a[i].getTotalCost();**

**if (sum<min)**

**{**

**minP=a[i];**

**min=sum;**

**}**

**}**

**return minP;**

**}**

**/\*\***

**\* 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)**

**{**

**Waypoint a = open.get(newWP.getLocation());**

**if (a==null)**

**{**

**open.put(newWP.getLocation(),newWP);**

**return true;**

**}**

**if (newWP.getPreviousCost()<a.getPreviousCost())**

**{**

**open.replace(newWP.getLocation(),newWP);**

**return true;**

**}**

**return false;**

**}**

**/\*\* Returns the current number of open waypoints. \*\*/**

**public int numOpenWaypoints()**

**{**

**return open.size();**

**}**

**/\*\***

**\* This method moves the waypoint at the specified location from the**

**\* open list to the closed list.**

**\*\*/**

**public void closeWaypoint(Location loc)**

**{**

**Waypoint a = open.remove(loc);**

**if(a!=null)**

**{**

**close.put(loc, a);**

**}**

**}**

**/\*\***

**\* Returns true if the collection of closed waypoints contains a waypoint**

**\* for the specified location.**

**\*\*/**

**public boolean isLocationClosed(Location loc)**

**{**

**return close.containsKey(loc);**

**}**

**}**

В классе **AStarState** содержаться все главные методы для реализации программы.

В самом начале объявлены два списка точек: открытые и закрытые клетки.

Дальше идут разные методы:

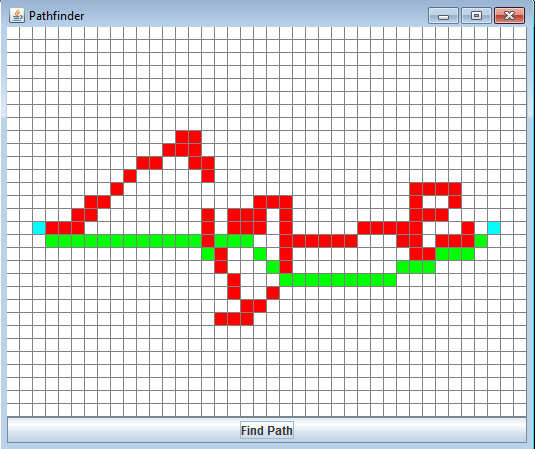
Нахождение минимальной точки - **getMinOpenWaypoint();**

Добавление точки в список открытых – **addOpenWaypoint();**

Перенесение точки из списка открытых в список закрытых – **closeWaypoint();**

Проверка точки на закрытость – **isLocationClosed();**

**Результат работы:**

****

**Вывод:** задача по нахождению кратчайшего пути выполнена с помощью изменения и добавления методов в 2 класса программы.