**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

по дисциплине «Объектно-ориентированное программирование»

Выполнил: студент группы БУТ1952

Обухов А.Д.

Проверил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва, 2020

Цель работы:

Изучить использование типа HashMap<>.

Задание.

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.

В ходе выполнения задания были внесены изменения в классы Location и AStarState.  
Исходный код Location:

/\*\*

\* 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 o) {

if (o == this) return true;

if (o == null) return false;

if (!(o instanceof Location)) return false;

return xCoord == ((Location)o).xCoord && yCoord == ((Location)o).yCoord;

}

public int hashCode() {

return 31 \* (xCoord + yCoord);

}

}

Исходный код AStarState:

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> closedWaypoints = new HashMap();

private HashMap<Location, Waypoint> openWaypoints = new HashMap();

/\*\*

\* 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()

{

if (openWaypoints.size() == 0) {

return null;

}

Waypoint result = null;

for (Waypoint w : openWaypoints.values()) {

if (result == null || result.getTotalCost() > w.getTotalCost()) {

result = w;

}

}

return result;

}

/\*\*

\* 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 oldWP = openWaypoints.get(newWP.getLocation());

if (oldWP == null) {

openWaypoints.put(newWP.getLocation(), newWP);

return true;

}

if (oldWP.getRemainingCost() > newWP.getRemainingCost()) {

openWaypoints.put(newWP.getLocation(), newWP);

return true;

}

return false;

}

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

public int numOpenWaypoints()

{

return openWaypoints.size();

}

/\*\*

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

\* open list to the closed list.

\*\*/

public void closeWaypoint(Location loc)

{

closedWaypoints.put(loc, openWaypoints.remove(loc));

}

/\*\*

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

\* for the specified location.

\*\*/

public boolean isLocationClosed(Location loc)

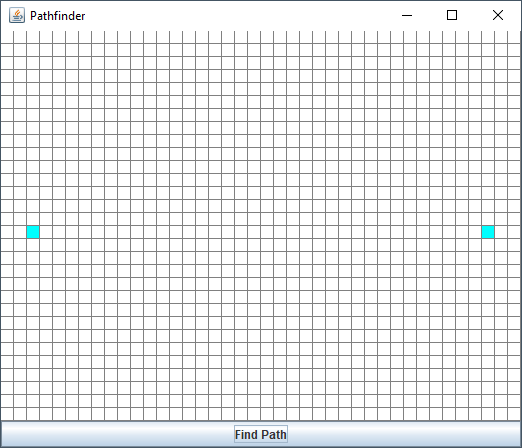
{

return closedWaypoints.containsKey(loc);

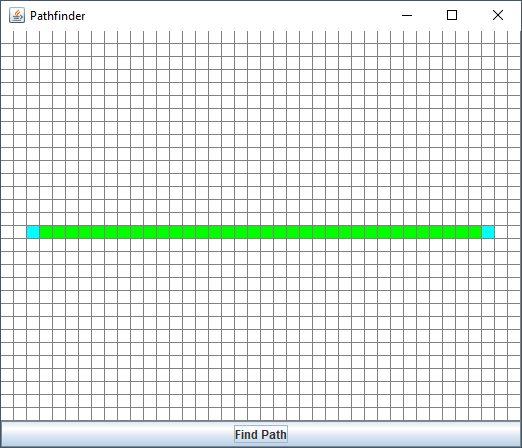
}

}

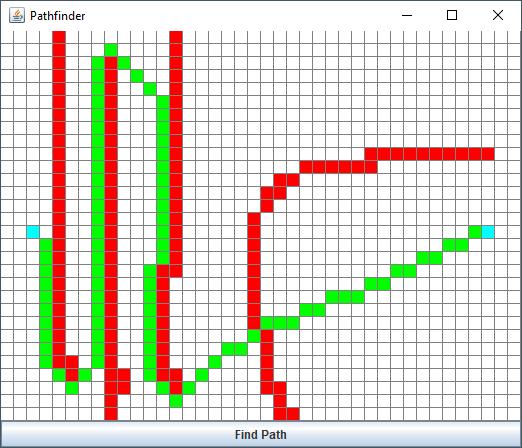
При запуске программы открывается окно, содержащее поле и две вершины, между которыми ищется путь.

  
Рис. 1 – окно программы.

При нажатии на кнопку “Find Path” программа начинает искать путь между вершинами.

  
Рис. 2 – путь без препятствий.

При нажатии кнопкой мыши на какую-либо из ячеек появляется препятствие, которое алгоритм пытается обойти.

  
Рис. 3 – путь с препятствиями.

Заключение.

В ходе выполнения этой лабораторной была изучена работа HashMap и необходимость переопределения методов hashCode и equals для своих классов.