**Федеральное агентство связи**

Ордена Трудового Красного Знамени

федеральное государственное бюджетное образовательное учреждение высшего образования   
«Московский технический университет связи и информатики»

Кафедра «Информатика»

Лабораторная работа №3: Алгоритм A\* («A star»)

по дисциплине «Кроссплатформенные технологии программирования»

Если вы когда-нибудь играли в какую-либо игру на компьютере на основе карт, то вы, вероятно, сталкивались с органами компьютерного управления, которые умеют самостоятельно рассчитывать путь из пункта А в пункт Б. На самом деле это обычная распространенная проблема как в играх, так и в других видах программного обеспечения - поиск пути от начального местоположения до пункта назначения с успешным преодолением препятствий.

Выполнил: студент

группы БСТ1803

Корнеев Александр

Вариант 12

Москва 2020

**Содержание**

[1 Задание на разработку программы 3](#_Toc41583110)

[2 Разработанные функции и классы 6](#_Toc41583111)

[2.1 Класс Location 6](#_Toc41583112)

[2.1.1 Метод int hashCode 6](#_Toc41583113)

[2.1.2 Метод boolean equals 6](#_Toc41583114)

[2.2. Класс AStarState 6](#_Toc41583115)

[2.2.1 Метод int numOpenWaypoints 6](#_Toc41583116)

[2.2.2 Метод Waypoint getMinOpenWaypoint 6](#_Toc41583117)

[2.2.3 Метод boolean addOpenWaypoint 7](#_Toc41583118)

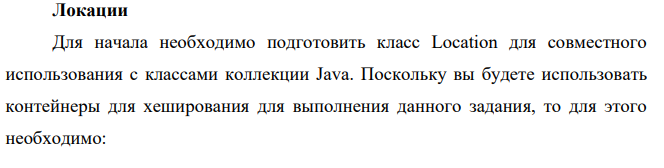
[2.2.4 Метод boolean isLocationClosed 7](#_Toc41583119)

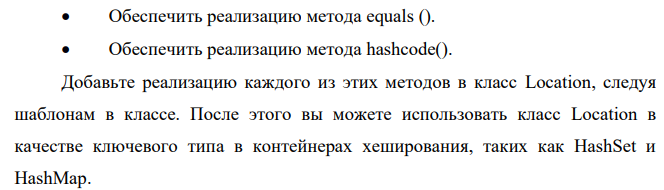
[2.2.5 Метод void closeWaypoint 7](#_Toc41583120)

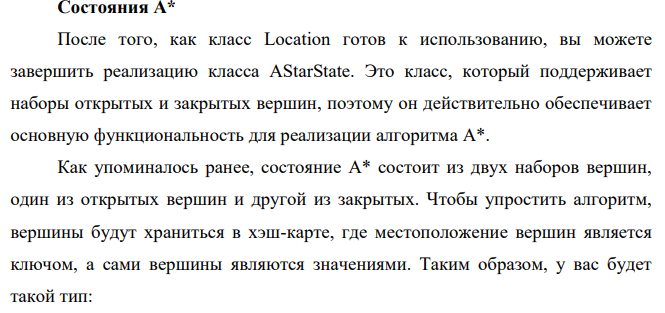
[3 Код программы 7](#_Toc41583121)

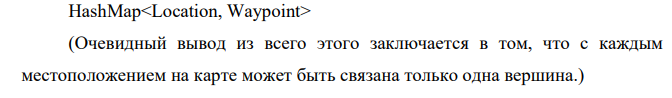
[4 Результат работы программы 9](#_Toc41583122)

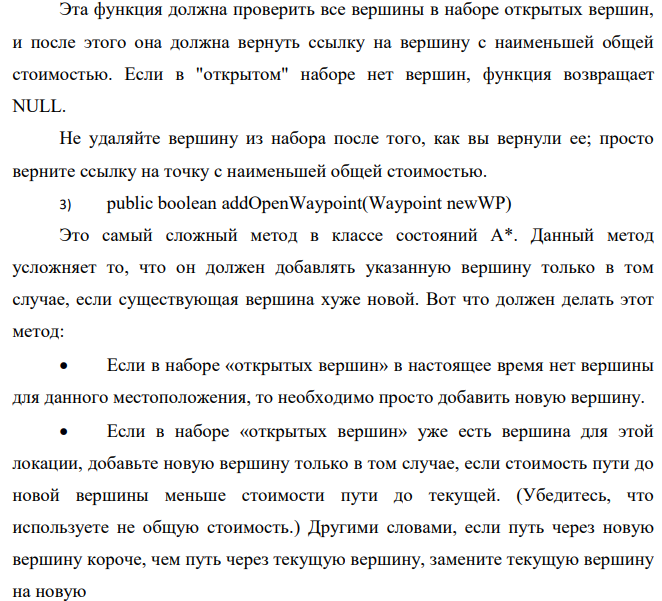
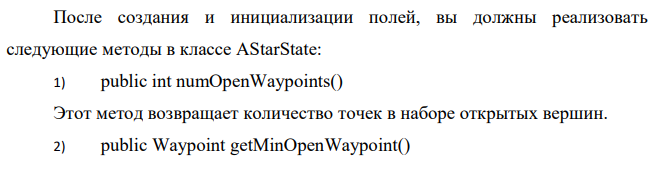
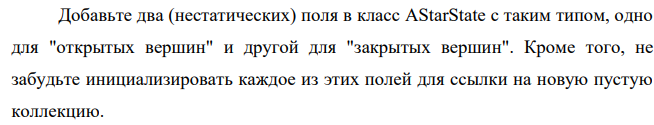
# Задание на разработку программы

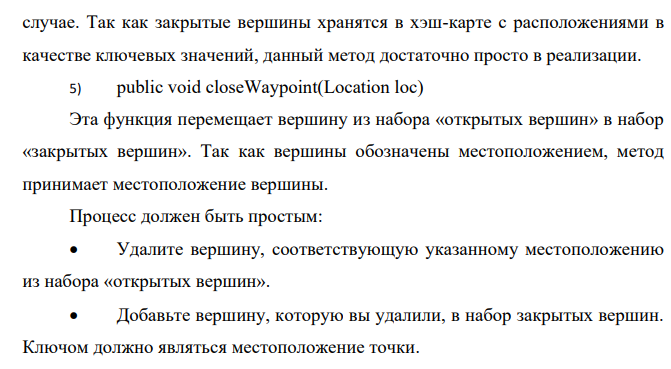
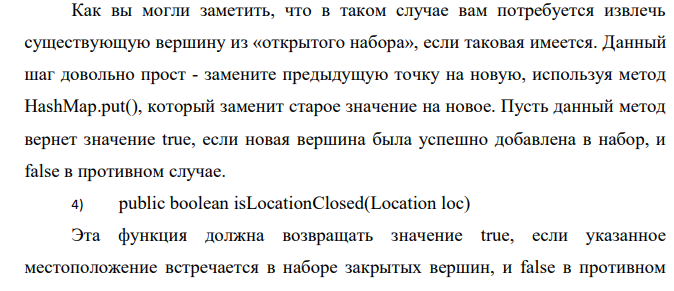


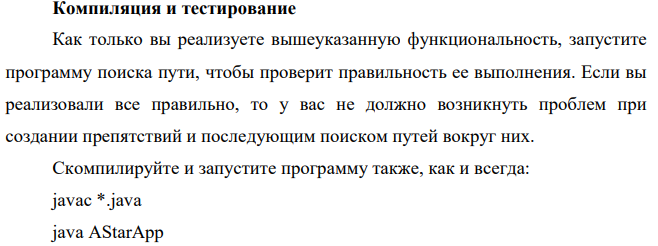












# Разработанные функции и классы

## Класс Location

Класс описывающий координатную точку на 2д карте.

* + 1. Метод int hashCode

Возвращает хэш данной точки

* + 1. Метод boolean equals

Сравнивает классы Location, возвращает **true** если точки равны, иначе **false**

## 2.2. Класс AStarState

* + 1. Метод int numOpenWaypoints

Этот метод возвращает количество точек в наборе открытых вершин

* + 1. Метод Waypoint getMinOpenWaypoint

Возвращает ссылку на вершину с наименьшей общей стоимостью, из набора “открытых” вершин. Если набор вершин пуст, возвращает **NULL**

* + 1. Метод boolean addOpenWaypoint

Метод добавляющий вершины. Возвращает **true,** если вершина успешно добавлена, иначе **false**

* + 1. Метод boolean isLocationClosed

Метод возвращает **true**, если местоположение встречается в наборе закрытых вершин, иначе **false**

* + 1. Метод void closeWaypoint

Перемещает вершину из набора “открытых” в набор “закрытых”

# Код программы

**public** **class** Location

{

@Override

**public** **int** hashCode() {

**final** **int** prime = 31;

**int** result = 1;

result = prime \* result + xCoord;

result = prime \* result + yCoord;

**return** result;

}

@Override

**public** **boolean** equals(Object obj) {

**if** (**this** == obj)

**return** **true**;

**if** (obj == **null**)

**return** **false**;

**if** (getClass() != obj.getClass())

**return** **false**;

Location other = (Location) obj;

**if** (xCoord != other.xCoord)

**return** **false**;

**if** (yCoord != other.yCoord)

**return** **false**;

**return** **true**;

}

}

/\*\*

\* 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** (**this**.numOpenWaypoints() == 0) **return** **null**;

Waypoint minCostWay = **null**;

**for** (Waypoint element : oW.values())

{

**if** (minCostWay == **null**) minCostWay = element;

**if** (element.getTotalCost()<minCostWay.getTotalCost()) minCostWay = element;

}

**return** minCostWay;

}

/\*\*

\* 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)

{

Location loc = newWP.getLocation();

**if** (oW.containsKey(loc))

{

**if** (oW.get(loc).getPreviousCost()>newWP.getPreviousCost())

{

oW.put(loc, newWP);

**return** **true**;

}

**else** **return** **false**;

}

**else**

{

oW.put(loc,newWP);

**return** **true**;

}

}

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

**public** **int** numOpenWaypoints()

{

**return** **this**.oW.size();

}

/\*\*

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

\* open list to the closed list.

\*\*/

**public** **void** closeWaypoint(Location loc)

{

cW.put(loc, oW.get(loc));

oW.remove(loc);

}

/\*\*

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

\* for the specified location.

\*\*/

**public** **boolean** isLocationClosed(Location loc)

{

**if** (cW.containsKey(loc)) **return** **true**;

**else** **return** **false**;

}

# Результат работы программы

