Министерство цифрового развития, связи и

Массовых коммуникаций Российской Федерации

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 №3

по дисциплине «Кроссплатформенные технологии программирования»

Выполнил: студент группы БСТ2001

Багдасарян Артём

Москва 2022

Оглавление

[**1.** **Цель и задание** 3](#_Toc97797644)

[**2.** **Ход выполнения лабораторной работы** 3](#_Toc97797645)

[**2.1** **Код программы** 3](#_Toc97797646)

[**2.2** **Результат работы программы** 4](#_Toc97797647)

[**3.** **Вывод** 5](#_Toc97797648)

[**Список использованных источников** 6](#_Toc97797649)

1. **Цель и задание**

В данной лабораторной работе мы изучим алгоритм поиск самого короткого пути A\*.

1. **Ход выполнения лабораторной работы** 
   1. **Код программы**

Ниже представлен код Location.java

*/\*\* Сравнивает определенную локацию с текущей. \*\*/*public boolean equals(Object obj) {  
  
 // Проверка приналежности объекта obj к классу Location  
 if (obj instanceof Location) {  
  
 // Каст другого объекта к классу Location,  
 // потом сравнение. Возвращает true если равны.  
 Location other = (Location) obj;  
 if (xCoord == other.xCoord && yCoord == other.yCoord) {  
 return true;  
 }  
 }  
 // Если мы попали сюда - координаты не равны. Возвращает false.  
 return false;  
}  
  
*/\*\* Обеспечивает hashCode для каждой локации. \*\*/*public int hashCode() {  
 int result = 20; // Какое-то число  
 // Используем другое число для умножения  
 result = 30 \* result + xCoord;  
 result = 30 \* result + yCoord;  
 return result;  
}

Ниже представлен код AStarState.java

private Map<Location, Waypoint> open\_waypoints  
 = new HashMap<Location, Waypoint>();  
  
private Map<Location, Waypoint> closed\_waypoints  
 = new HashMap<Location, Waypoint>();

public Waypoint getMinOpenWaypoint()  
{  
 // Если нет открытых вершин, возвращаем null.  
 if (numOpenWaypoints() == 0)  
 return null;  
 Set open\_waypoint\_keys = open\_waypoints.keySet();  
 Iterator i = open\_waypoint\_keys.iterator();  
 Waypoint best = null;  
 float best\_cost = Float.*MAX\_VALUE*;  
 // Проверяет все вершины.  
 while (i.hasNext()) {  
 // Сохраняет текущую локацию.  
 Location location = (Location) i.next();  
 // Сохранет текущую вершину.  
 Waypoint waypoint = open\_waypoints.get(location);  
 // Сохраняет текущую "стоимость" вершины.  
 float waypoint\_total\_cost = waypoint.getTotalCost();  
  
 // Если общая "стоимость" текущей вершины лучше (меньше)  
 // тогда сохраненная "стоимость" сохраненной лучшей вершины заменяет  
 // сохраненную вершину текущей вершины и сохраненную общую "стоимость"  
 // текущей общей "стоимостью".  
 if (waypoint\_total\_cost < best\_cost) {  
 best = open\_waypoints.get(location);  
 best\_cost = waypoint\_total\_cost;  
 }  
 }  
 // Возвращает вершину с минимальной общей "стоимостью".  
 return best;  
}

public boolean addOpenWaypoint(Waypoint newWP)  
{  
 // Находит локацию новой вершины.  
 Location location = newWP.getLocation();  
  
 // Проверяет есть ли уже открытая вершина на месте новой вершины.  
 if (open\_waypoints.containsKey(location)) {  
 // Если на этом месте уже есть открытая вершина,  
 // проверяет меньше ли "предыдущая стоимость" новой вершины  
 // "предыдущей стоимости" текущей.  
 Waypoint current\_waypoint = open\_waypoints.get(location);  
 if (newWP.getPreviousCost() < current\_waypoint.getPreviousCost()) {  
 // Если "предыдущая стоимость" новой вершины меньше  
 // "предыдущей стоимости" текущей вершины, новая вершина  
 // заменяет старую и возвращает true.  
 open\_waypoints.put(location, newWP);  
 return true;  
 }  
 // Если "предыдущая стоимость" новой вершины не меньше "предыдущей стоимости"  
 // текущей вершины, return false.  
 return false;  
 }  
 // Если в наборе «открытых вершин» в настоящее время нет вершины для данного места,  
 // то необходимо просто добавить новую вершину и return true.  
 open\_waypoints.put(location, newWP);  
 return true;  
}

public int numOpenWaypoints()  
{  
 return open\_waypoints.size();  
}

public void closeWaypoint(Location loc)  
{  
 Waypoint waypoint = open\_waypoints.remove(loc);  
 closed\_waypoints.put(loc, waypoint);  
}

public boolean isLocationClosed(Location loc)  
{  
 return closed\_waypoints.containsKey(loc);  
}

* 1. **Результат работы программы**

Результат работы программы показан на рисунке 1

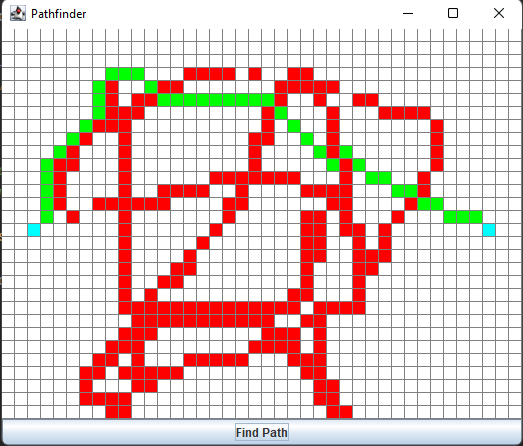


Рисунок 1 – Результат работы программы

* 1. **Программа на Git hub**

Ссылка на репозиторий на Git hub: https://github.com/Artem2406/KTP

1. **Вывод**

В данной лабораторной работе мы изучили алгоритм поиск самого короткого пути A\*.

# **Список использованных источников**

1. Камаев В.А., Костерин В.В. Технологии программирования. М.: Высшая школа, 2006.

2. Жоголев Е.А.Технология программирования. – М.: Научный мир, 2004.