**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 Математическая кибернетика и информационные технологии

Отчет по лабораторной работе № 3

по дисциплине «Информационные технологии и программирование»

Выполнил: студент группы БФИ2102

Голубева Елена Александровна

Проверила:

Мосева Марина Сергеевна

Москва, 2022

Цель работы

Изучение работы алгоритма А\* - поиск пути от начального местоположения до пункта назначения с успешным преодолением препятствий.

Задание

Прежде чем начать, вам необходимо скачать исходные файлы для данной лабораторной работы:

• Map2D.java - представляет карту, по которой перемещается алгоритм А\*, включая в себя информацию о проходимости ячеек

• Location.java - этот класс представляет координаты конкретной ячейки на карте

• Waypoint.java - представляет отдельные вершины в сгенерированном пути

• AStarPathfinder.java - этот класс реализует алгоритм поиска пути А\* в виде статического метода.

• AStarState.java - этот класс хранит набор открытых и закрытых вершин, и предоставляет основные операции, необходимые для функционирования алгоритма поиска А\*.

• AStarApp.java - простое Swing-приложение, которое обеспечивает редактируемый вид 2D-карты, и запускает поиск пути по запросу

• JMapCell.java - это Swing -компонент, который используется для отображения состояния ячеек на карте

Изменить классы Location и AStarState.

В Location:

• Обеспечить реализацию метода equals ().

• Обеспечить реализацию метода hashcode().

В AStarState:

Добавьте два (нестатических) поля с таким типом, одно для "открытых вершин" и другой для "закрытых вершин".

Реализовать следующие методы:

1) public int numOpenWaypoints() Этот метод возвращает количество точек в наборе открытых вершин.

2) public Waypoint getMinOpenWaypoint() Эта функция должна проверить все вершины в наборе открытых вершин, и после этого она должна вернуть ссылку на вершину с наименьшей общей стоимостью

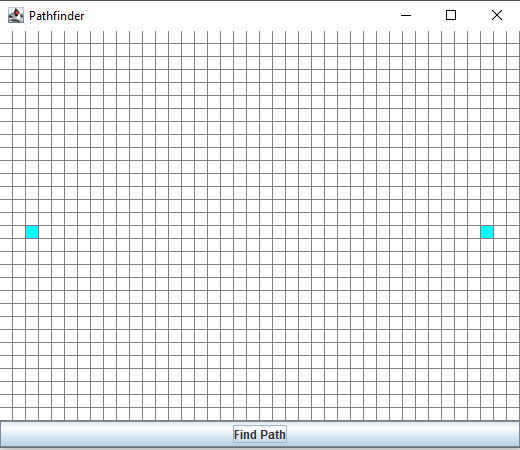
3) public boolean addOpenWaypoint(Waypoint newWP)

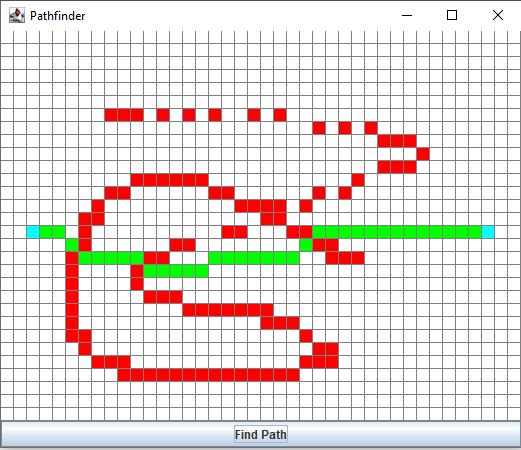
4) public boolean isLocationClosed(Location loc) Эта функция должна возвращать значение true, если указанное местоположение встречается в наборе закрытых вершин, и false в противном случае.

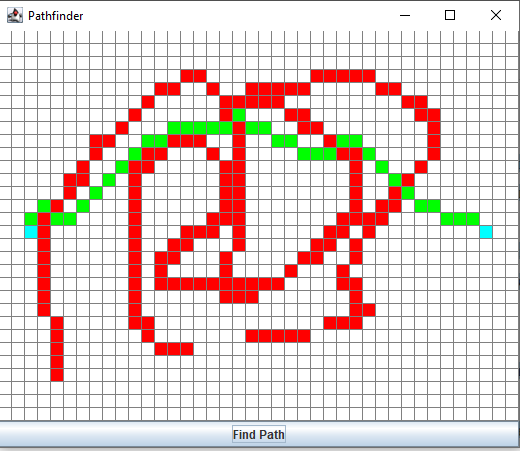
5) public void closeWaypoint(Location loc) Эта функция перемещает вершину из набора «открытых вершин» в набор «закрытых вершин».

Ход работы

1. Результат работы кода.







Добавленный код в класс Location

public boolean equals(Object obj) { //проверка на равенство объектов  
  
 if (obj instanceof Location) {  
  
 Location other = (Location) obj;  
 if (xCoord == other.xCoord && yCoord == other.yCoord) {  
 return true;  
 }  
 }  
 return false;  
}  
  
public int hashCode() { //хэш-код для каждого местоположения  
 int result =3;  
  
 result = 31 \* result + xCoord;  
 result = 31 \* result + yCoord;  
 return result;  
}

Добавленный код в класс AStarState

package com.company;  
import java.util.\*;  
  
//хранит набор открытых и закрытых точек и представляет основные операции для алгоритма А\*  
public class AStarState  
{  
 //ссылка на карту, по которой перемещается алгоритм A\*  
 private Map2D map;  
 //Инициализация всех открытых/закрытых путевых точек и их местоположений.  
 private Map<Location, Waypoint> open\_waypoints  
 = new HashMap<Location, Waypoint> ();  
  
 private Map<Location, Waypoint> closed\_waypoints  
 = new HashMap<Location, Waypoint> ();  
  
 public Waypoint getMinOpenWaypoint() //метод сканирует все открытые путевые точки и возвращает путевую точку с минимальной общей стоимостью  
 {  
 if (numOpenWaypoints() == 0)  
 return null;  
  
 // Инициализируйте ключевой набор всех открытых путевых точек, итератор для перебора набора и переменную  
 // для хранения наилучшей путевой точки и стоимости для этой путевой точки.  
 Set open\_waypoint\_keys = open\_waypoints.keySet();  
 Iterator i = open\_waypoint\_keys.iterator();  
 Waypoint best = null;  
 float best\_cost = Float.*MAX\_VALUE*;  
  
 // Проверка всех открытых путевых точек  
 while (i.hasNext())  
 {  
 // сохраняет текущее местоположение  
 Location location = (Location)i.next();  
 // Сохраняет текущую путевую точку  
 Waypoint waypoint = open\_waypoints.get(location);  
 // Сохраняет общую стоимость для текущей путевой точки  
 float waypoint\_total\_cost = waypoint.getTotalCost();  
 // Проверка стоимости для текущей путевой точки  
 if (waypoint\_total\_cost < best\_cost)  
 {  
 best = open\_waypoints.get(location);  
 best\_cost = waypoint\_total\_cost;  
 }  
 }  
 return best;  
 }  
  
 //добавляет указанную вершину только в том  
 //случае, если существующая вершина хуже новой  
 public boolean addOpenWaypoint(Waypoint newWP)  
 {  
 // Местоположение новой точки  
 Location location = newWP.getLocation();  
  
 // Проверяет, есть ли уже открытая путевая точка в новом местоположении путевых точек  
 if (open\_waypoints.containsKey(location))  
 {  
 Waypoint current\_waypoint = open\_waypoints.get(location);  
 if (newWP.getPreviousCost() < current\_waypoint.getPreviousCost())  
 {  
 open\_waypoints.put(location, newWP);  
 return true;  
 }  
 return false;  
 }  
 // Если еще нет открытой путевой точки, то добавить ее в коллекцию открытых путевых точек  
 open\_waypoints.put(location, newWP);  
 return true;  
 }  
 //Возвращает текущий номер открытой путевой точки  
 public int numOpenWaypoints()  
 {  
 return open\_waypoints.size();  
 }  
  
//Этот метод перемещает путевую точку в  
//указанном местоположении из открытого списка в закрытый список.  
 public void closeWaypoint(Location loc)  
 {  
 Waypoint waypoint = open\_waypoints.remove(loc);  
 closed\_waypoints.put(loc, waypoint);  
 }  
//Возвращает значение true, если коллекция закрытых путевых  
// точек содержит путевую точку для указанного местоположения.  
 public boolean isLocationClosed(Location loc)  
 {  
 return closed\_waypoints.containsKey(loc);  
 }  
  
}

Вывод

В процессе выполнения данной лабораторной работы мы изучили работу алгоритма А\* - поиск пути от начального местоположения до пункта назначения с успешным преодолением препятствий.

Ссылка на Github: https://github.com/ElenaGolubeva/Laboratory-3-Java

Список использованной литературы

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

2006.

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