Министерство цифрового развития и массовых коммуникаций

Российской Федерации Ордена Трудового Красного Знамени

федеральное государственное образовательное бюджетное учреждение высшего профессионального образования

«Московский технический университет связи и информатики»

**Кафедра электроники**

**Отчет по лабораторной работе №3**

**По дисциплине «Кроссплатформенные технологии программирования» по теме:** **«Алгоритм A\* (“A star”)»**

Выполнил: студент

группы БСТ2001

Ибодуллоев У.Х.

Вариант 7

Проверил:

Ст. преп. Полянцева К. А.

Москва

1. Цель работы:

Реализовать поиск кратчайшего пути на основе реализованного алгоритма A\* (“A star”).

1. Задание:

В Location.java:

1. Обеспечить реализацию метода equals ().
2. Обеспечить реализацию метода hashcode().

В AStarState.java:

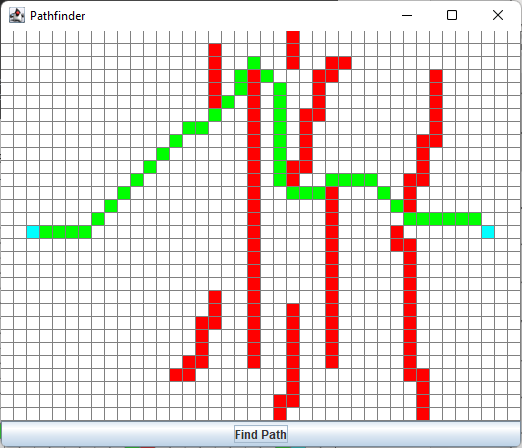
1. Добавить два (нестатических) поля в класс AstarState с типом HashMap<Location,Waypoint> одно для "открытых вершин" и другой для "закрытых вершин"
2. Реализовать метод public int numOpenWaypoints() (возвращает количество точек в наборе открытых вершин)
3. Реализовать метод public Waypoint getMinOpenWaypoint() (должна проверить все вершины в наборе открытых вершин, и после этого она должна вернуть ссылку на вершину с наименьшей общей стоимостью)
4. Реализовать метод public boolean addOpenWaypoint(Waypoint newWP) (Если в наборе «открытых вершин» в настоящее время нет вершины для данного местоположения, то необходимо просто добавить новую вершину, если в наборе «открытых вершин» уже есть вершина для этой локации, добавьте новую вершину только в том случае, если стоимость пути до новой вершины меньше стоимости пути до текущей)
5. Реализовать метод public boolean isLocationClosed(Location loc) (возвращать значение true, если указанное местоположение встречается в наборе закрытых вершин, и false в противном случае)
6. Реализовать метод public void closeWaypoint(Location loc) (перемещает вершину из набора «открытых вершин» в набор «закрытых вершин»)
7. Код программы:

Ссылка на GitHub: <https://github.com/1Double/MTUCI/tree/main/Term_4/KTP/La>b3

1. Листинг файла Location.java
2. import java.util.Objects;  
     
   */\*\*  
    \* 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);  
    }  
     
    @Override  
    public boolean equals(Object obj) {  
    if (obj == this)  
    return true;  
    if (obj == null || obj.getClass() != this.getClass())  
    return false;  
     
    Location location = (Location) obj;  
    return xCoord == location.xCoord && yCoord == location.yCoord;  
    }  
     
    @Override  
    public int hashCode() {  
    int result = 17; // Простое число  
     
    // используется другое простое число для перемножения  
    result = 37 \* result + xCoord;  
    result = 37 \* result + yCoord;  
    return result;  
    }  
   }
3. Листинг файла Point3d.java

import static java.lang.Math.\*;  
  
import java.util.HashMap;  
import java.util.Iterator;  
import java.util.Map;  
import java.util.Set;  
import java.util.concurrent.ForkJoinWorkerThread;  
  
*/\*\*  
 \* 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> openWaypoints = new HashMap<>();  
 private HashMap<Location, Waypoint> closeWaypoints = 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 (numOpenWaypoints() == 0) {  
 return null;  
 }  
  
 Waypoint minWaypoint = null;  
 float min = Float.*MAX\_VALUE*;  
  
 for (Waypoint waypoint : openWaypoints.values()) {  
 float cost = waypoint.getTotalCost();  
 if (cost < min) {  
 min = cost;  
 minWaypoint = waypoint;  
 }  
 }  
 return minWaypoint;  
 }  
  
 */\*\*  
 \* 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 openWP = openWaypoints.get(newWP.loc);  
  
 if (openWP == null || newWP.getPreviousCost() < openWP.getPreviousCost()) {  
 openWaypoints.put(newWP.loc, 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)  
 {  
 Waypoint waypoint = openWaypoints.remove(loc);  
 if (openWaypoints != null) {  
 closeWaypoints.put(loc, waypoint);  
 }  
 }  
  
 */\*\*  
 \* Returns true if the collection of closed waypoints contains a waypoint  
 \* for the specified location.  
 \*\*/* public boolean isLocationClosed(Location loc)  
 {  
 return closeWaypoints.containsKey(loc);  
 }  
}

1. Выполнение:



1. Выводы

В ходе работы я научился реализовывать поиск кратчайшего пути на основе алгоритма A\*.