ОРДЕНА ТРУДОВОГО КРАСНОГО ЗНАМЕНИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ “МОСКОВСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ СВЯЗИ И ИНФОРМАТИКИ”.

Кафедра МКиИТ

Лабораторная работа №3 по дисциплине: ”Технологии программирования ”

“ Задача No3: Java-Я-звезда!”.

Выполнила: студентка группы БСТ1602

Лушина Ольга

Проверил: М. Г. Городничев

Москва 2018

**Содержание:**

1. Цели и задачи. 3
2. Анализ предметной области и выбор инструментария. 3
3. Функции и их объяснение. 3
4. Выводы. 6
5. Цели и задачи.

Изменить расположение и AStarState. Всё остальное-это код платформы, которая позволяет редактировать карту и показывать путь, по которому алгоритм генерирует.

1. Анализ предметной области и выбор инструментария:

В данной лабораторной работе я использовала бесплатный пакет Jdk, также стандартный редактор txt.

1. Функции и их объяснение:

|  |
| --- |
| Тип 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> openPoints; |
|  |

|  |
| --- |
| private HashMap<Location, Waypoint> closePoints; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\*\* |
|  |

|  |
| --- |
| \* 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; |
|  |

|  |
| --- |
| openPoints = new HashMap<Location, Waypoint>(); |
|  |

|  |
| --- |
| closePoints = new HashMap<Location, Waypoint>(); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\*\* Returns the map that the A\* pathfinder is navigating. \*\*/ |
|  |

|  |
| --- |
| public Map2D getMap() |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| return map; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\*\* |
| Эта функция проверяет все точки в наборе открытых точек и возвращает ссылку на точку с наименьшей общей стоимостью иначе возвращают NULL. |
|  |

|  |
| --- |
|  |

|  |
| --- |
| \*\*/ |
|  |

|  |
| --- |
| public Waypoint getMinOpenWaypoint() |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| Waypoint[] p=new Waypoint[]{}; |
|  |

|  |
| --- |
| p=openPoints.values().toArray(p); |
|  |

|  |
| --- |
| Waypoint minPoint=p[0]; |
|  |

|  |
| --- |
| double min=minPoint.getTotalCost(); |
|  |

|  |
| --- |
| for(int i=1;i<p.length; i++){ |
|  |

|  |
| --- |
| double cost = p[i].getTotalCost(); |
|  |

|  |
| --- |
| if (cost<min) { |
|  |

|  |
| --- |
| minPoint=p[i]; |
|  |

|  |
| --- |
| min=cost; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| return minPoint; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\*\* |
|  |

|  |
| --- |
| Добавляет указанную точку при существующей путевой. |

|  |
| --- |
| \*\*/ |
|  |

|  |
| --- |
| public boolean addOpenWaypoint(Waypoint newWP) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| Waypoint p=openPoints.get(newWP.getLocation()); |
|  |

|  |
| --- |
| if (p==null){ |
|  |

|  |
| --- |
| openPoints.put(newWP.getLocation(),newWP); |
|  |

|  |
| --- |
| return true; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| if(newWP.getPreviousCost()<p.getPreviousCost()){ |
|  |

|  |
| --- |
| openPoints.replace(newWP.getLocation(),newWP); |
|  |

|  |
| --- |
| return true; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| return false; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\*\* Этот метод просто возвращает количество точек в набор отрытых точек.\*\*/ |
|  |

|  |
| --- |
| public int numOpenWaypoints() |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| return openPoints.size(); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\*\* |
| Эта функция берет точку и перемещает её от набора  «открытых точек» к набору «закрытых точек». |
|  |

|  |
| --- |
| \*\*/ |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| public void closeWaypoint(Location loc) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| Waypoint p = openPoints.remove(loc); |
|  |

|  |
| --- |
| if(p!=null) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| closePoints.put(loc, p); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\*\* |
| Возвращает действительный метод если указанное расположение появляется в наборе закрытых точек или в случае неверного метода. |

|  |
| --- |
|  |

|  |
| --- |
| public boolean isLocationClosed(Location loc) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| return closePoints.containsKey(loc); |
|  |

|  |
| --- |
| } |
|  |

}

1. Выводы: получила вышеупомянутую функциональность, реализовала нужные по заданию методы.