**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**

Факультет РиТ

**ЛАБОРАТОРНАЯ РАБОТА №3**

по дисциплине «Java-программирование»

на тему:

«Основы объектно-ориентированного программирования»

Выполнил: студ. гр. БПЗ1901

Неживлева Ксения

Проверил: Харрасов Камиль Раисович

Москва 2021

Цель работы:

Java позволяет использовать объекты. В данной лабораторной работе необходимо использовать классы по одному на файл, чтобы описать, как эти объекты работают.

Выполнение:

Для начала необходимо подготовить класс Location для совместного использования с классами коллекции Java. Поскольку мы будем использовать контейнеры для хеширования для выполнения данного задания, то для этого необходимо:

• Обеспечить реализацию метода equals ().

• Обеспечить реализацию метода hashcode().

package com.lab3;  
  
import java.util.Objects;  
  
*/\*\*  
 \* Этот класс представляет конкретное место на 2D-карте.  
 \* Координаты - это целые числа.  
 \*\*/*public class Location  
{  
 */\*\* X координата этого места. \*\*/* public int xCoord;  
  
 */\*\* Y координата этого места. \*\*/* public int yCoord;  
  
 */\*\* Создает новое местоположение с указанными целыми координатами. \*\*/* public Location(int x, int y)  
 {  
 xCoord = x;  
 yCoord = y;  
 }  
  
 */\*\* Создает новое местоположение с координатами (0, 0). \*\*/* public Location()  
 {  
 this(0, 0);  
 }  
  
 */\*\* Реализацию метода equals ()\*\*/* @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 Location location = (Location) o;  
 return xCoord == location.xCoord && yCoord == location.yCoord;  
 }  
*/\*\* реализация метода hashcode().\*\*/* @Override  
 public int hashCode() {  
 return Objects.*hash*(xCoord, yCoord);  
 }  
}

Рис.1-класс Location

Добавим два (нестатических) поля в класс AStarState с таким типом, одно для "открытых вершин" и другой для "закрытых вершин". После создания и инициализации полей, реализуем следующие методы в классе AStarState:

1. public int numOpenWaypoints()

Этот метод возвращает количество точек в наборе открытых вершин.

1. 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. Если в "открытом" наборе нет вершин, функция возвращает NULL.

1. public boolean addOpenWaypoint(Waypoint newWP)

Данный метод усложняет то, что он должен добавлять указанную вершину только в том случае, если существующая вершина хуже новой. Вот что должен делать этот метод:

• 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, добавим новую вершину только в том случае, если стоимость пути до новой вершины меньше стоимости пути до текущей.

Другими словами, если путь через новую вершину короче, чем путь через текущую вершину, замените текущую вершину на новую. Данный шаг довольно прост - заменим предыдущую точку на новую, используя метод HashMap.put(), который заменит старое значение на новое. Пусть данный метод вернет значение true, если новая вершина была успешно добавлена в набор, и false в противном случае.

1. public boolean isLocationClosed(Location loc)

Эта функция должна возвращать значение true, если указанное местоположение встречается в наборе закрытых вершин, и false в противном случае. Так как закрытые вершины хранятся в хэш-карте с расположениями в качестве ключевых значений, данный метод достаточно просто в реализации.

1. public void closeWaypoint(Location loc)

Эта функция перемещает вершину из набора «открытых вершин» в набор «закрытых вершин». Так как вершины обозначены местоположением, метод принимает местоположение вершины.

Процесс:

• Удалите вершину, соответствующую указанному местоположению из набора «открытых вершин».

• Добавьте вершину, которую вы удалили, в набор закрытых вершин. Ключом должно являться местоположение точки.

import java.util.HashMap;  
import java.util.Iterator;  
  
*/\*\*  
 \* 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;  
 */\*\* два (нестатических) поля в класс AStarState с таким типом, одно  
 для "открытых вершин" и другой для "закрытых вершин"\*\*/* public HashMap<Location, Waypoint> opened;  
 public HashMap <Location, Waypoint> closed;  
 */\*\*  
 \* 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;  
 opened = new HashMap<>();  
 closed = new HashMap<>();  
 }  
  
 */\*\* Returns the map that the A\* pathfinder is navigating. \*\*/* public Map2D getMap()  
 {  
 return map;  
 }  
  
 */\*\*  
 \* Эта функция должна проверить все вершины в наборе открытых вершин,  
 \* и после этого она должна вернуть ссылку на вершину с наименьшей общей  
 \* стоимостью. Если в "открытом" наборе нет вершин, функция возвращает  
 \* NULL.  
 \*\*/* public Waypoint getMinOpenWaypoint()  
 {  
 if (numOpenWaypoints() == 0)  
 return null;  
 Iterator iter = opened.values().iterator();  
 Waypoint min = (Waypoint)iter.next();  
 for (Waypoint w = min; iter.hasNext(); w = (Waypoint)iter.next())  
 if (w.getTotalCost() < min.getTotalCost())  
 min = w;  
 return min;  
 }  
 */\*\*  
 \* • 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.  
 \*\*/* public boolean addOpenWaypoint(Waypoint newWP)  
 {  
 Location newLoc = newWP.getLocation();  
 Waypoint w = opened.get(newLoc);  
 if (w == null || newWP.getPreviousCost() < w.getPreviousCost())  
 {  
 opened.put(newLoc, newWP);  
 return true;  
 }  
 return false;  
 }  
 */\*\* Этот метод возвращает количество точек в наборе открытых вершин. \*\*/* public int numOpenWaypoints()  
 {  
 return opened.size();  
 }  
 */\*\*  
 \* Эта функция перемещает вершину из набора «открытых вершин» в набор  
 \* «закрытых вершин». Так как вершины обозначены местоположением, метод  
 \* принимает местоположение вершины.  
 \*\*/* public void closeWaypoint(Location loc)  
 {  
 Waypoint v = opened.remove(loc);  
 closed.put(loc,v);  
 }  
  
 */\*\*  
 \* Эта функция должна возвращать значение true, если указанное  
 \* местоположение встречается в наборе закрытых вершин, и false в противном  
 \* случае.  
 \*\*/* public boolean isLocationClosed(Location loc)  
 {  
 return closed.containsKey(loc);  
 }  
}

Рис.2-класс AStarState

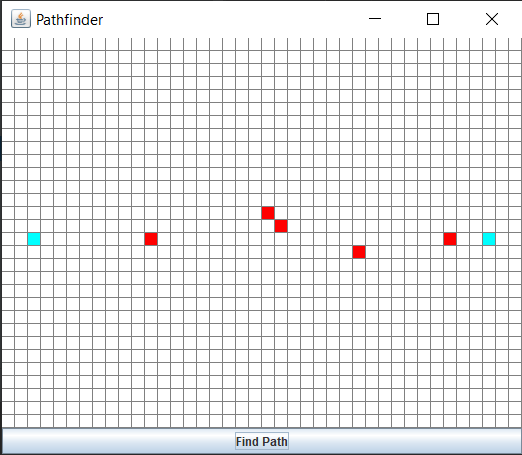


Рис.3-Результат работы программы до нажатия кнопки

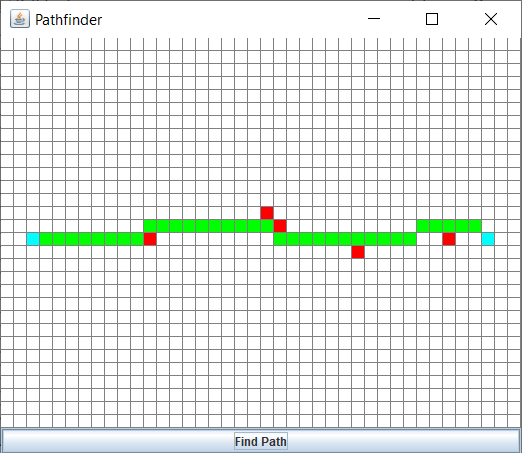


Рис.3-Результат работы программы после нажатия кнопки

Вы можете щелкнуть по различным квадратам, чтобы сделать из них в барьеры (красные) или проходимые клетки (белые). Синие клетки обозначают начало и конец пути. Нажав на кнопку "Find Path", программа вычислит путь, используя алгоритм А\*, и затем отобразит его зеленым цветом. В случае если путь не будет найден, то программа просто не отобразит путь.