**Федеральное агентство связи**

**Ордена трудового Красного Знамени федеральное государственное бюджетное**

**образовательное учреждение высшего образования**

**«Московский технический университет связи и информатики»**

Кафедра Математическая кибернетика и информационные технологии

Лабораторная работа № 3

Выполнил: студент группы БФИ2002

Ковачев В.Е.

Проверила:

Москва, 2021

**Задание:**

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

• Обеспечить реализацию метода equals ().

• Обеспечить реализацию метода hashcode().

2) Состояния А\*

После того, как класс Location готов к использованию, вы можете завершить реализацию класса AStarState. Это класс, который поддерживает наборы открытых и закрытых вершин, поэтому он действительно обеспечивает основную функциональность для реализации алгоритма А\*.

Как упоминалось ранее, состояние А\* состоит из двух наборов вершин, один из открытых вершин и другой из закрытых. Чтобы упростить алгоритм, вершины будут храниться в хэш-карте, где местоположение вершин является ключом, а сами вершины являются значениями. Таким образом, у вас будет такой тип:

HashMap

Добавьте два (нестатических) поля в класс AStarState с таким типом, одно для "открытых вершин" и другой для "закрытых вершин". Кроме того, не забудьте инициализировать каждое из этих полей для ссылки на новую пустую коллекцию.

После создания и инициализации полей, вы должны реализовать следующие методы в классе AStarState:

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

Не удаляйте вершину из набора после того, как вы вернули ее; просто верните ссылку на точку с наименьшей общей стоимостью.

3) 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, добавьте 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

Как вы могли заметить, что в таком случае вам потребуется извлечь существующую вершину из «открытого набора», если таковая имеется. Данный шаг довольно прост - замените 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.

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». Так как вершины обозначены местоположением, метод принимает местоположение вершины. Процесс должен быть простым:

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

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

**Ход работы:**

**Задание №1**

**Для выполнения первого задания был подготовлен класс Location для совместного использования с классами коллекции Java:**

importjava.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 = 15; **// Простое число**

**// используется другое простое число для перемножения**

result = 30 \* result + xCoord;

result = 30 \* result + yCoord;

return result;

}

}

**Задание №2,3,4,5**

Для выполнения второго задания был написан метод public Waypoint getMinOpenWaypoint() представленный ниже в классе AStarState.

Для выполнения третьего задания был написан метод public boolean addOpenWaypoint(Waypoint newWP) представленный ниже в классе AStarState.

Для выполнения четвёртого задания был написан метод public boolean isLocationClosed(Location loc) представленный ниже в классе AStarState.

Для выполнения пятого задания был написан метод public void closeWaypoint(Location loc) представленный ниже в классе AStarState.

import java.util.HashMap;

import java.util.Iterator;

import java.util.Map;

import java.util.Set;

import java.util.concurrent.ForkJoinWorkerThread;

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;

}

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();

}

public void closeWaypoint(Location loc)

{

Waypoint waypoint = openWaypoints.remove(loc);

if (openWaypoints != null) {

closeWaypoints.put(loc, waypoint);

}

}

public boolean isLocationClosed(Location loc)

{

return closeWaypoints.containsKey(loc);

}

}

**Результат работы программы**

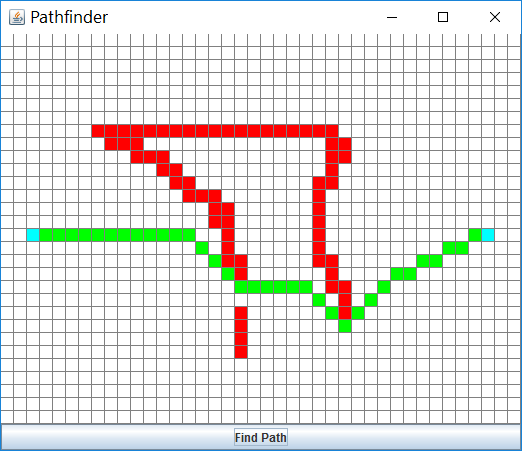


Рисунок №1- Результат работы программы

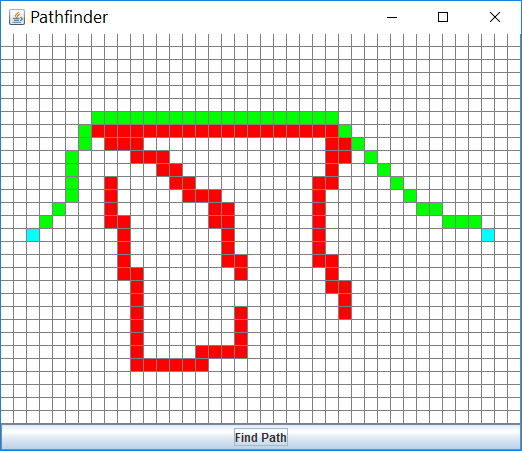


Рисунок №2- Результат работы программы

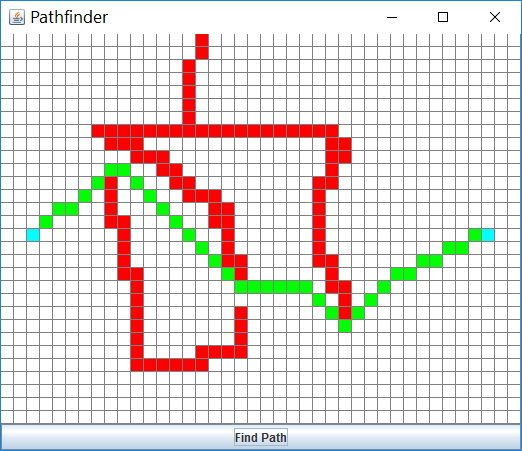


Рисунок №3- Результат работы программы

**Вывод**

Мы научились использовать ассоциативные массивы на языке Java. В результате выполнения работы был создан алгоритм, который позволил добавлять информацию в ассоциативный массив, а также считывать информацию из него и обрабатывать, по итогу выполнения программы был создан алгоритм А\* для поиска пути.

**Список использованной литературы**

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

2006.

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