ФЕДЕРАЛЬНОЕ АГЕНСТВО СВЯЗИ

Ордена Трудового Красного Знамени

федеральное государственное бюджетное

образовательное учреждение высшего образования

«Московский Технический Университет Связи и Информатики» (МТУСИ)

Кафедра МКиИТ

Лабораторная работа №2

«Java-Я-звезда»

Выполнил:

Студент 2 курса

Группы БСТ-1602

Смышляев Владислав

Вариант №15

Москва, 2018

Содержание

[1. Цель и задачи 3](#_Toc512777514)

[2. Анализ предметной области, выбор инструментария 3](#_Toc512777515)

[3. Объяснение функций 3](#_Toc512777516)

[3.1. Класс AStarState 3](#_Toc512777517)

[3.2. Класс Location 6](#_Toc512777518)

[4. Выводы 7](#_Toc512777519)

1. Цель и задачи

Цель работы: Изучить работу структур данных таких как “словари” и использовать их методы.

Задачи: Дописать программу позволяющую найти путь из токи А в точку Б, обходя препятствия, то есть проходить своеобразный лабиринт, нарисованный пользователем. При этом составленный путь должен быть самым коротким.

1. Анализ предметной области, выбор инструментария

Для написания данной программы использоваться редактор кода Visual Studio Code, из-за удобства редактирования, подцветки кода, а также возможности работать с контролем версий Гит. Для компиляции кода использовалась командная консоль Windows. В качестве компилятора кода использовался Java Development Kit от компании Oracle.

1. Объяснение функций
   1. Класс 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;

/\*\*

\* Use <code>open\_waypoints and closed\_waypoints</code> as hash maps

\*/

private HashMap<Location,Waypoint> open\_waypoints;

/\*\*

\* Use <code>open\_waypoints and closed\_waypoints</code> as hash maps

\*/

private HashMap<Location,Waypoint> closed\_waypoints;

/\*\*

\* 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;

open\_waypoints = new HashMap<>();

closed\_waypoints = new HashMap<>();

}

/\*\* 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()

{

Waypoint[] points = new Waypoint[]{};

points = open\_waypoints.values().toArray(points);

Waypoint min\_point = points[0];

//Start iterating through the list

for (int i=1;i<points.length;i++)

{

Waypoint point = points[i];

if (min\_point.getTotalCost() > point.getTotalCost()) min\_point = point;

}

return min\_point;

}

/\*\*

\* 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)

{

//Compare locations to see if received waypoint is in the list

if (open\_waypoints.containsKey(newWP.getLocation())) {

//Next check if its cost is less than the existing one

if (newWP.getPreviousCost() < open\_waypoints.get(newWP.getLocation()).getPreviousCost()) {

open\_waypoints.replace(newWP.getLocation(), newWP);

//Return true that the point was added

return true;

} else {

//Return false if the point was not satisfying the requirments

return false;

}

}

//Add the point to the list if it was not found in the list

open\_waypoints.put(newWP.getLocation(),newWP);

return true;

}

/\*\* Returns the current number of open waypoints. \*\*/

public int numOpenWaypoints()

{

return open\_waypoints.size();

}

/\*\*

\* This method moves the waypoint at the specified location from the

\* open list to the closed list.

\*\*/

public void closeWaypoint(Location loc)

{

//Iterate through the list

Waypoint point = open\_waypoints.get(loc);

closed\_waypoints.put(point.getLocation(),point);

open\_waypoints.remove(point.getLocation());

}

/\*\*

\* Returns true if the collection of closed waypoints contains a waypoint

\* for the specified location.

\*\*/

public boolean isLocationClosed(Location loc)

{

Waypoint point = closed\_waypoints.get(loc);

return point != null;

}

}

Класс AStarState является ключевым для работы программы. Здесь определены функции, для занесения точек в список исследованных, проверка на исследованность, нахождение точки с минимальным расстоянием до неё, а также проверка на исследованность точки и нахождения количества неисследованных точек.

Для обеспечения работы данного класса у него существует два основных свойства, а точнее два “словаря”, для исследованных и неисследованных точек на карте. С помощью “словарей” можно с лёгкостью найти количество каких-либо точек, просто узнав его размер. Также очень полезным здесь является метод проверки на принадлежность к “словарю”.

В качестве пары ключ/значение были использованы классы Location/Waypoint соответственно.

Работа каждой функции полностью описана в комментариях к коду.

* 1. Класс Location

/\*\*

\* 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;

}

/\*\*

\* Finds whether two locations are in the same position

\* @param loc - Object that can be casted to Location class

\*/

@Override

public boolean equals(Object loc)

{

Location loc\_cast = ((Location) loc);

return xCoord == loc\_cast.xCoord && yCoord == loc\_cast.yCoord;

}

/\*\*

\* Used with hashmaps to get the hash of the location

\* Calculated using the position of the location

\*/

@Override

public int hashCode()

{

int hash = 1;

final int PRIME = 31;

hash += PRIME\*hash+xCoord;

hash += PRIME\*hash+yCoord;

return hash;

}

/\*\* Creates a new location with coordinates (0, 0). \*\*/

public Location()

{

this(0, 0);

}

}

От части класс Location уже был написан, но его требовалось дописать, добавив функции hashCode() и equals(), для обеспечения исправной работы данного класса как ключа для “словарей”. В остальном модификации данного класса не производились.

1. Выводы

Таким образом, были изучены одна из самых основных структур данных “словарь”. Дописана программа и выполнена поставленная задача.