Ордена Трудового Красного Знамени

федеральное государственное бюджетное

образовательное учреждение высшего

образования «Московский технический

университет связи и информатики»

Кафедра информатики

Лабораторная работа № 2 «I'm A Star!»

Выполнил:

Студент 2 курса

Факультета «ОТФ-2»

Группы БСТ-1602

Марченко Дмитрий

Вариант № 11

Москва, 2018

**Цель работы:** Один очень широко используемый алгоритм для этого вида проблемы называют А\*, и это - очень эффективный алгоритм для новаторства в компьютерной программе. Алгоритм концептуально довольно прост. Начиная с начального расположения, алгоритм постепенно создает путь от источника до места назначения, всегда используя "до сих пор лучший путь ", чтобы сделать следующий шаг. Это гарантирует, что завершающий путь также будет оптимальным.

**Задание:**

1) Первое, что должно быть сделано – тип должен быть подготовлен для использования с наборами типов Java. Поскольку вы будете использовать кеширование хранилищ для выполнения данного задания, то это предполагает: обеспечение реализации метода equals (); обеспечение реализации метода hashcode().

2) закончить реализацию типа AStarState. Это тип, который хранит наборы открытых и закрытых точек, так что это действительно обеспечивает базовую функциональность для А\*.

3) реализовать следующие методы типа AStarState:

public int numOpenWaypoints()

public Waypoint getMinOpenWaypoint()

public boolean addOpenWaypoint(Waypoint newWP)

public boolean isLocationClosed(Location loc)

public void closeWaypoint(Location loc)

**Анализ предметной области и инструментарий:** для разработки программы обработки информации, поступающей от пользователя в циклах и ее вывода взят пакетjava development kit-9.0.4 с классом Scanner для организации ввода и стандартный редактор тхт.

**Код программы:**

**Описание класса 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;  //Метод сравнивает текущую точку с полученной |
|  | public boolean equals (Location p){ |
|  | return (xCoord==p.xCoord && yCoord==p.yCoord); |
|  | } |
|  | public int hashcode(Location p){ |
|  | final int Prime=31; |
|  | int result = 1; |
|  | result = result \* Prime + xCoord; |
|  | result = result \* Prime + yCoord; |
|  | return result; |
|  | } |
|  | /\*\* 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); |
|  | } |
|  | } |

**Описание класса AStarState:**

|  |
| --- |
| /\*\* |
|  | \* 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. |
|  | \*\*/ |
|  | import java.util.ArrayList; |
|  | import java.util.List; |
|  | public class AStarState |
|  | { |
|  | /\*\* This is a reference to the map that the A\* algorithm is navigating. \*\*/ |
|  | private Map2D map; |
|  | private List<Waypoint> openPoints; |
|  | private List<Waypoint> closedPoints; |
|  | /\*\* |
|  | \* 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 ArrayList<Waypoint>(); |
|  | closedPoints = new ArrayList<Waypoint>(); |
|  | } |
|  | /\*\* 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()//возвращает «дешевейшую» точку |
|  | { |
|  | // TODO: Implement. |
|  | if(numOpenWaypoints()==0) проверка на существование открытых точек |
|  | return null; |
|  | Waypoint minPoint = openPoints.get(0); получение первой открытой |
|  | for (int i=1; i<numOpenWaypoints();i++) сравнение со всеми //открытыми точками при помощи метода getTotalCost() |
|  | { |
|  | Waypoint p = openPoints.get(i); |
|  | if (p.getTotalCost()<minPoint.getTotalCost()) |
|  | minPoint=p; |
|  | } |
|  | return minPoint; |
|  | } |
|  |
|  |
|  | /\*\* |
|  | \* 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) |
|  | { |
|  | // TODO: Implement. |
|  | for (int i=0; i<numOpenWaypoints();i++) |
|  | { |
|  | Waypoint previousPoint=openPoints.get(i); |
|  | Location previousLocation=previousPoint.getLocation(); |
|  | if (previousLocation==newWP.getLocation()) |
|  | if(previousPoint.getPreviousCost()<newWP.getPreviousCost()) |
|  | { |
|  | openPoints.remove(previousPoint); |
|  | openPoints.add(newWP); |
|  | return true; |
|  | } |
|  | else |
|  | return false; |
|  | } |
|  | openPoints.add(newWP); |
|  | return false; |
|  | } |
|  | /\*\* Returns the current number of open waypoints. \*\*/ |
|  | public int numOpenWaypoints() |
|  | { |
|  | // TODO: Implement. |
|  | return openPoints.size();//метод подсчитывает число открытых //точек при помощи метода size() |
|  | } |
|  | /\*\* |
|  | \* This method moves the waypoint at the specified location from the |
|  | \* open list to the closed list. |
|  | \*\*/ |
|  | public void closeWaypoint(Location loc)//перемещает точку из списка //открытых в закрытых |
|  | { |
|  | // TODO: Implement. |
|  | for(int i=0;i<numOpenWaypoints();i++)//поиск среди открытых |
|  | { |
|  | Waypoint currentpoint=openPoints.get(i);// получение iй //точки |
|  | if(loc.equals(currentpoint.getLocation())) |
|  | { |
|  | openPoints.remove (currentpoint);//удаление из откр. |
|  | closedPoints.add(currentpoint);//помещение в закр. |
|  | } |
|  | } |
|  | } |
|  | /\*\* |
|  | \* Returns true if the collection of closed waypoints contains a waypoint |
|  | \* for the specified location. |
|  | \*\*/ |
|  | public boolean isLocationClosed(Location loc)//проверка на закрытую |
|  | { |
|  | // TODO: Implement. |
|  | for (int i=0;i<closedPoints.size();i++) |
|  | {  //сравнивает точку со всеми точками в списке закрытых при помощи методов equals и //getLocation |
|  | if (loc.equals(closedPoints.get(i).getLocation())) |
|  | return true; |
|  | } |
|  | return false; |
|  | } |
|  | } |

**Классы не подвергшиеся изменениям:**

AStarApp.java - ообеспечивает работу интерфейса

AStarPathfinder.java – алгоритмы поиска кратчайших путей.

JMapCell.java – работа и визуализация «ячеек» путей

Map2D.java – обеспечивает работу двумерной карты из ячеек

WayPoint.java –представляет собой точку на карте и методы обработки координат

**Выводы:** построены алгоритмы работы с массивами объектов, сравнения, обработки их значений.