ФЕДЕРАЛЬНОЕ АГЕНСТВО СВЯЗИ

Ордена Трудового Красного Знамени

федеральное государственное бюджетное

образовательное учреждение высшего образования

«Московский Технический Университет Связи и Информатики» (МТУСИ)

Кафедра математической кибернетики и информационных технологий

Лабораторная работа №3

«Java-Я-звезда!»

Выполнила:

Студентка 2 курса

Группы БСТ1602

Костромина Любовь

Москва 2018

Оглавление

[Цель лабораторной работы: 3](#_Toc514181765)

[Задачи: 3](#_Toc514181766)

[Анализ предметной области и выбор инструментария: 3](#_Toc514181767)

[Ход решения задач: 3](#_Toc514181768)

[Вывод: 4](#_Toc514181769)

# Цель лабораторной работы:

Объединение данных преподавателем исходных файлов для возможности выполнения алгоритма А\*, позволяющего найти лучший путь из одной точки в другую.

# Задачи:

1. Прежде чем начать, нужно скачать исходные файлы:

• Map2D.java – карта, по которой двигается алгоритм.

• Location.java - координаты конкретной ячейки на карте.

• Waypoint.java - представляет отдельные точки в созданный путь.

• AStarPathfinder.java - реализует алгоритм поиска пути как статический метод.

• AStarState.java - хранит набор открытых точек и закрытых точек и обеспечивает базовые операции для функционирования алгоритма поиска.

• AStarApp.java - обеспечивает редактируемый вид карты и запускает поиск пути по запросу.

• JMapCell.java - используется для отображения состояния клеток на карте.

1. Подготовка типа для использования с наборами типов Java.

• обеспечение реализации метода equals ();

• обеспечение реализации метода hashcode().

Добавить реализацию каждого из этих методов в тип Location.

1. Добавить две (нестатические) области класса AStarState с типом HashMap<Location, Waypoint>: одну для "открытых точек" и другую для "закрытых точек."

4. Наконец, верните действительный метод, если новый пункт был добавлен в набор открытых точек, или ошибочный, если новые точки не добавляются.

# 

# Анализ предметной области и выбор инструментария:

Я использовала IntelliJ IDEA 2017.3.4.

# Ход решения задач:

Метод ‘public int numOpenWaypoints()’ возвращает количество точек в набор открытых точек.

Функция ‘public Waypoint getMinOpenWaypoint()’ проверяет все точки в наборе открытых точек и возвращает ссылку на точку с наименьшей общей стоимостью. Если нет точки в "открытых" наборах, возвращает NULL.

Метод ‘public boolean addOpenWaypoint(Waypoint newWP)’ добавляет указанную точку при существующей путевой точке.

• Если в настоящее время нет точек для этого места в "открытых точках" набора, то просто добавляет новую точку.

• Если точка уже в этом месте в "открытой точке" набора, то потом добавляет новый пункт, только если "старая цена" за новую точку меньше "старой цены" за текущую точку. Мы возвращаем существующую маршрутную точку с «открытого» набора, если он есть и действуем в соответствии с правилом «открытого» набора - заменяем предыдущую точку на новую, просто используя hashmap.put (), как обычно (он заменяет старый код, а значения сопоставляет с новым кодом).

Возвращаю действительный метод, если новый пункт был добавлен в набор открытых точек, или ошибочный, если новые точки не добавляются.

Функция ‘public boolean isLocationClosed(Location loc)’ возвращает действительный метод, если указанное расположение появляется в наборе закрытых точек или в случае неверного метода.

Функция ‘public void closeWaypoint(Location loc)’ берет точку и перемещает её от набора «открытых точек» к набору «закрытых точек». Так как точки закодированы их расположением, метод берет расположение точек.

Процесс простой:

• удаляем соответствие точки указанному расположению из набора «открытых точек»;

• добавляем точку, которую удалили из набора закрытых точек.

# Вывод:

Я познакомилась с реализацией такого интересного алгоритма, как алгоритм А\*, который позволяет найти кратчайший путь из пункта А в пункт Б. В данной лабораторной работе я не писала этот алгоритм с нуля, а использовала готовые исходные файлы, нуждающиеся лишь в доработке и соединении в один проект.