**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**ОТЧЕТ**

**по учебной практике**

**Тема: Реализация алгоритма А\* на языке Java с графическим интерфейсом.**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 0832 |  | Кривенцова Л.С. |
| Студентка гр. 0832 |  | Деткова А.С. |
| Студентка гр. 0832 |  | Здобнова К.Д. |
| Руководитель |  | Фирсов М.А. |

Санкт-Петербург

2022

**ЗАДАНИЕ**

**НА УЧЕБНУЮ ПРАКТИКУ**

|  |
| --- |
| Студентка Кривенцова Л.С. группы 0382 |
| Студентка Деткова А.С. группы 0382 |
| Студентка Здобнова К.Д. группы 0382  Тема практики: Реализация алгоритма А\* на языке Java с графическим интерфейсом. |
| Задание на практику:  Командная итеративная разработка визуализатора алгоритма на Java с графическим интерфейсом.  Алгоритм: А\*. |
| Сроки прохождения практики: 29.06.2022 – 12.07.2022 |
| Дата сдачи отчета: 11.07.2022 |
| Дата защиты отчета: 11.07.2022 |
|  |

|  |  |  |
| --- | --- | --- |
| Студентка гр. 0832 |  | Кривенцова Л.С. |
| Студентка гр. 0832 |  | Деткова А.С. |
| Студентка гр. 0832 |  | Здобнова К.Д. |
| Руководитель |  | Фирсов М.А. |

**АННОТАЦИЯ**

Основной целью работы является получение навыков программирования на языке Java и освоение парадигмы объектно-ориентированного программирования. Цель достигается командной работой путём разработки программы с графическим интерфейсом, использующей указанные в задании алгоритмы. В данной работе необходимо реализовать алгоритм А\*, находящий во взвешенном графе маршрут наименьшей стоимости от выбранной начальной вершины до выбранной конечной.

**SUMMARY**

The main purpose of the work is to gain programming skills in the Java language and master the paradigm of object-oriented programming. The goal is achieved by teamwork by developing a program with a graphical interface that uses the algorithms specified in the task. In this work, it is necessary to implement the A\* algorithm, which finds the least cost route from the selected initial vertex to the selected final vertex in the weighted graph.

**СОДЕРЖАНИЕ**

|  |  |  |
| --- | --- | --- |
|  | Введение | 5 |
| 1. | Требования к программе | 6 |
| 1.1.  1.1.1  1.1.2  1.1.3 | Исходные требования к программе  Общие исходные требования  Требования к вводу исходных данных  Требования к выводу данных | 6  6  7  7 |
| 2. | План разработки и распределение ролей в бригаде | 8 |
| 2.1. | План разработки | 8 |
| 2.2. | Распределение ролей в бригаде | 8 |
| 3. | Особенности реализации | 10 |
| 3.1. | Возможности приложения | 10 |
| 3.2. | Структура данных для реализация алгоритма А\* | 11 |
| 3.3  3.4  3.5 | Представление графа  Визуализация графа  Визуализация алгоритма | 14  15  16 |
| 4. | Тестирование | 18 |
| 4.1 | Тестирование алгоритма А\* | 18 |
| 4.2 | Тестирование приложения  Заключение  Список использованных источников  Приложение А. UML-диаграмма | 23  27  28  29 |
|  | Приложение B. Исходный код | 30 |
|  |  |  |
|  |  |  |
|  |  |  |

**ВВЕДЕНИЕ**

Задача практики состоит в разработке приложения, позволяющего находить минимальный маршрут в взвешенном графе. Программа визуализирует пошаговое нахождение такого пути на графе. Для его поиска используется алгоритм А\*.

Поиск A\* (произносится «А звезда» или «А стар», от англ.Astar) — в информатике и математике, алгоритм поиска по первому наилучшему совпадению на графе, который находит маршрут с наименьшей стоимостью от одной вершины (начальной) к другой (целевой, конечной).

Порядок обхода вершин определяется эвристической функцией «расстояние + стоимость» (обычно обозначаемой как f(x)). Эта функция—сумма двух других: функции стоимости достижения рассматриваемой вершины (x) из начальной (обычно обозначается как g(x) и может быть как эвристической, так и нет), и функции эвристической оценки расстояния от рассматриваемой вершины к конечной (обозначается как h(x)).

Функция h(x) должна быть допустимой эвристической оценкой, то есть не должна переоценивать расстояния к целевой вершине. Например, для задачи маршрутизации h(x) может представлять собой расстояние до цели по прямой линии, так как это физически наименьшее возможное расстояние между двумя точками.

**1. ТРЕБОВАНИЯ К ПРОГРАММЕ**

**1.1. Исходные требования к программе**

**1.1.1 Общие исходные требования:**

- Приложение должно быть с графическим интерфейсом;

- Приложение должно быть ясным и удобным для пользователя;

- Помимо визуализации алгоритма, должны выводиться текстовые пояснения происходящего для пользователя;

- Визуализация алгоритма должна быть пошаговой, шаги не должны быть крупными;

- Должна быть возможность задать входные данные как из файла, так и при работе в самом приложении;

- Следует предусмотреть возможности взаимодействия с графическими элементами с помощью мыши (простейший пример: создание вершины графа в заданном месте на холсте по щелчку).

Макет приложения:

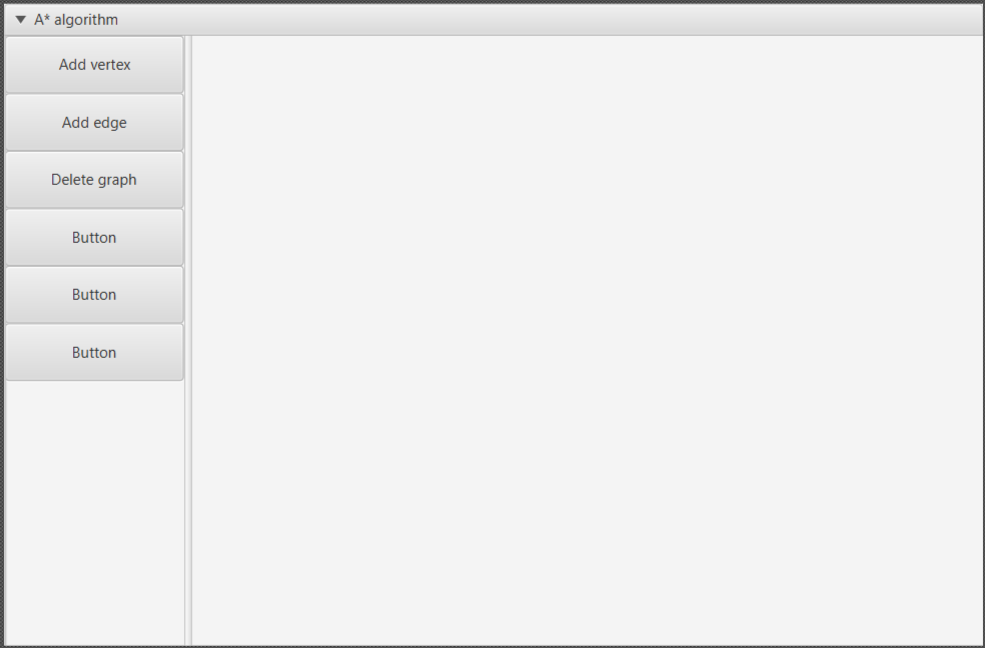


Рисунок 1. Макет приложения

**1.1.2 Требования к вводу исходных данных:**

Пользователь может задать ввод данных из файла и из специального текстового окна внутри приложения. Также должна быть возможность создать граф в самом приложении с помощью специальных инструментов.

**1.1.3. Требования к выводу данных**

Программа должна показывать результат выполнения работы алгоритма А\*: графическое отображение полученного пути на графе и вывод результата (путь и вес) в специальное окно.

**2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ**

**2.1. План разработки**

1. Завершение прохождения курса на платформе Stepik – **30 июня**.

2. Создание общего репозитория. Выбор графического интерфейса и создание прототипа для GUI. Сдача вводного задания и согласование спецификации и плана разработки. Написание класса для реализации алгоритма А\*. – **1 июля.**

3. Создание прототипа - приложение, демонстрирующее интерфейс, но (почти) не реализующее основные функции. Прототип включает в себя: стартовое окно приложения, окно редактора графа; создание кнопок для редактирования, поля для ввода данных; написание классов для хранения и обработки графа; дизайн интерфейса – **4 июля**.

4. Добавление возможности чтения данных из файла, написание методов для отрисовки редактирования графа. Добавление возможности запустить алгоритм А\* для введенного графа. Сдача первой версии программы – **8 июля**.

5. Добавление тестирования и исключений для крайних случаев. Добавление пошаговой визуализации алгоритма А\*. Сдача второй версии программы – **11 июля**.

6. Завершить разработку проекта – **11 июля.**

**2.2. Распределение ролей в бригаде**

1. Разработка прототипа:

Кривенцова Любовь – написание алгоритма А\*.

Деткова Анна – написание классов для работы с графом и его отрисовки.

Здобнова Ксения – написание отчета, создание макета графического интерфейса и его дизайн, написание классов для работы с графом и его отрисовки.

2. Разработка 1-ой версии программы:

Кривенцова Любовь – написание тестов для алгоритма А\*. Добавление алгоритма А\* в приложение. Написание отчета.

Деткова Анна – добавление в приложение функции для запуска алгоритма А\*. Визуализация алгоритма в приложении. Добавление редактирования графа с помощью нажатия мыши.

Здобнова Ксения – визуализация в приложении методов редактирования графа. Визуализация алгоритма в приложении. Добавление редактирования графа с помощью нажатия мыши. Написание отчета.

3. Разработка 2-ой версии программы:

Кривенцова Любовь – написание тестов для алгоритма А\*. Проверка работы алгоритма на крайних случаях. Написание отчета.

Деткова Анна – добавление пошаговой визуализации работы с графом. Запуск приложения через консоль, написание соответствующей инструкции по запуску. Написание отчета.

Здобнова Ксения – добавление исключений, возникающих при взаимодействии пользователя с инструментами редактирования графа. Написание отчета.

**3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ**

**3.1 Возможности приложения**

Приложение запускается со стартового окна, функционал работы с графом предоставляется после нажатия на кнопку «*Start*»:

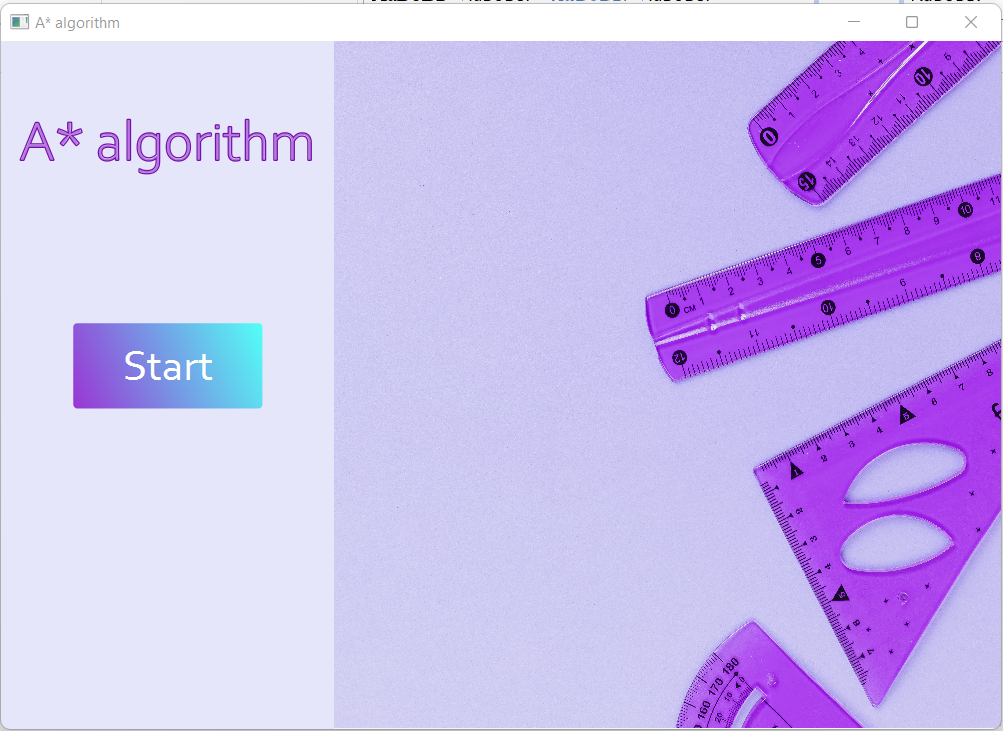


Рисунок 2. Стартовое окно

Далее пользователю открывается редактор, в котором можно работать с графом.

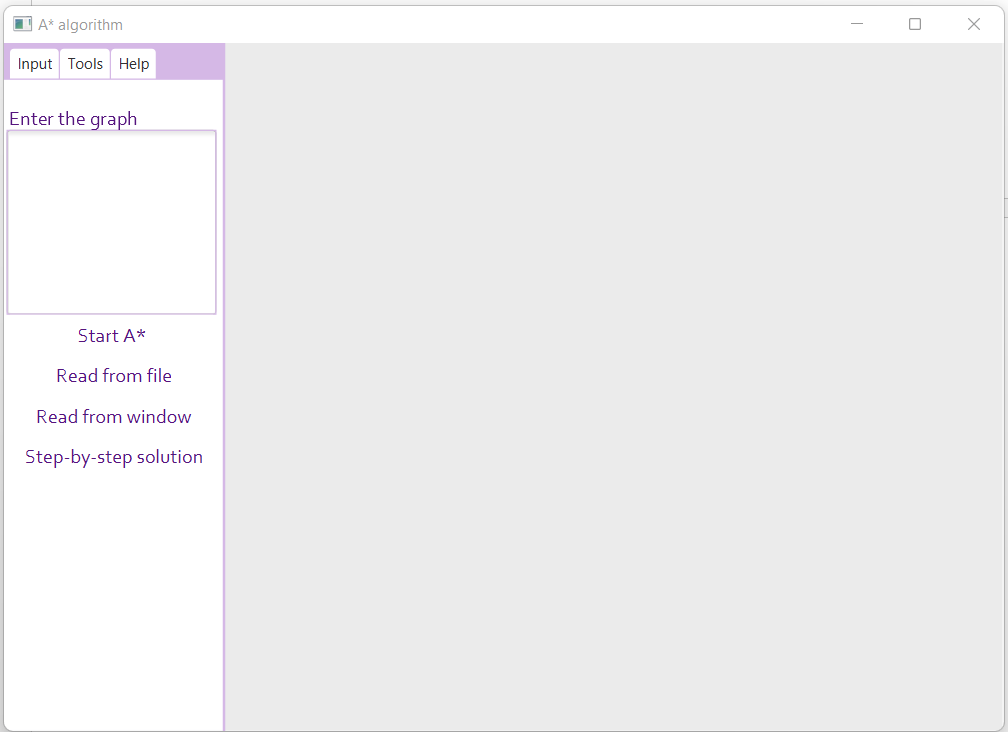


Рисунок 3. Окно редактора

Во вкладке «*Input*» пользователь может ввести граф для последующей работы с ним. Кнопка «*Start A\**» запускает алгоритм А\*. «*Read from file*» дает возможность пользователю считать данные о графе из файла – название файла вбивается пользователем в специальном окне, которое появляется после нажатия соответствующей кнопки. «*Read from window*» запускает алгоритм по данным, введенным в окно, расположенном на панели. «*Step-by-step-solution*» запускает пошаговое выполнение алгоритма на графе – пользователь может идти пошагово посмотреть работу алгоритма, будут доступны кнопки возвращения на шаг назад, на шаг вперед и к концу алгоритма. Во время режима пошагового выполнения кнопки редактирования графа недоступны.

Во вкладке «*Tools*» расположены инструменты редактирования графа. Кнопка «*Add edge*» добавляет ребро - пользователь кликает мышкой по двум вершинам, между которыми нужно провести ребро, затем открывается окно для ввода веса ребра. «*Delete vertex*» удаляет вершину – пользователь кликает по вершине, которую следует удалить. «*Delete edge*» удаляет ребро – пользователь кликает мышкой по двум вершинам, между которыми нужно удалить ребро. «*Clear*» - очищает поле.

Во вкладке «*Help*» описаны требования к формату ввода данных и вывод.

Двойным кликом правой кнопки мыши по полю пользователь может добавить вершину, при добавлении ей задается дефолтное имя. При одинарном нажатии на вершину правой кнопки мыши открывается контекстное меню с двумя доступными пользователю функциями – переименовать вершину и удалить вершину.

**3.2. Структура данных для реализация алгоритма А\***

Для работы алгоритма был написан класс AStar, имеющий следующую структуру:

public class AStar {

private HashMap<Vertex, Double> f; - хэш-таблица, соотносящая вершину и её сумму пройденного расстояния и эвристического веса.

private HashMap<Vertex, Double> g; - хэш-таблица, соотносящая вершину и её сумму пройденного расстояния.

private ArrayList<Vertex> in\_open; - массив, реализующий очередь вершин, из которой в алгоритме вершины берутся для рассмотрения.

private ArrayList<Vertex> in\_closed; - массив пройденных (рассмотренных) вершин.

private HashMap<Vertex, Vertex> from; - хэш-таблица, где ключом является вершина, а значением – вершина, путём из которой мы пришли в вершину, указанную в ключе.

private ArrayList<HashMap<Vertex, Double>> f\_steps; - массив хэш-таблиц, соотносящих вершину и её значение (вес + значение эвристической функции) на каждом витке алгоритма. Реализован для вывода пошагового решения.

private List<ArrayList<Vertex>> paths; - двумерный массив. Вложенный массив представляет собой сбор вершин, образующих путь, который рассматривается на витке алгоритма. Реализован для вывода пошагового решения.

private ArrayList<Vertex> solution; - массив вершин, составляющий итоговый, найденный в алгоритме, кратчайший путь.

private StringBuilder path; - строка, в которую записывается итоговый путь.

public AStar();- конструктор, инициализирует поля пустыми списками и хеш-таблицами.

public static double h(Vertex a, Vertex b); — эвристика двух вершин — рассчитывает евклидово расстояние. Принимает на вход две вершины, между которыми рассчитывается расстояние, и возвращает вещественное число – рассчитанное евклидово расстояние.

public Vertex min\_f(); - метод не принимает ничего в качестве аргументов, возвращает наиболее подходящую вершину, куда следует сделать следующий шаг алгоритма (выбирает из очереди вершину, с наименьшим показателем f).

private Vertex a\_star(Vertex start, Vertex finish, ArrayList<Vertex> graph); — алгоритм А\*, в качестве аргументов принимает начальную и конечную вершины (ссылки) и массив вершин, представляющий собой граф; возвращает finish вершину, или null, если путь построить не удалось.

public ArrayList<Vertex> a\_star\_public(Vertex start, Vertex finish, ArrayList<Vertex> graph); - «разворачивает» цепочку вершин. Продвигаясь по полям from проходит от конечной вершины (полученной в методе a\_star) до начальной, и восстанавливает путь, который возвращает как список вершин — кратчайший путь от start до finish.

public String getPath(); - метод, проходясь по вершинам пути, заполняет поле-строку path и возвращает значения поля.

public ArrayList<Vertex> getStepPath(Vertex n); - метод, принимающий в качестве аргумента ссылку на вершину графа и возвращающий массив вершин – путь, которым алгоритм пришёл к поданной вершине.

public double getWeight(); - метод проходится по вершинам итогового пути (solution) и считает суммарный вес, который возвращает в форме вещественного числа.

public ArrayList<HashMap<Vertex, Double>> getF\_steps(); - геттер для получения значения поля f\_steps.

public List<ArrayList<Vertex>> getPaths(); – геттер для получения значения приватного поля paths.

Подробнее про алгоритм А\* и реализующий его метод a\_star.

В очередь заносится стартовая вершина, её расстояние от начала равно нулю, а вес высчитывается как значение эвристической функции.

Пока в очереди есть вершины, продолжается цикл while (с условием, пока очередь не пуста). Если очередь оказывается пустой, значит пути нет и функция, не заходя в цикл, возвращает значение null.

В цикле выбирается текущая вершина, методом определения вершины из с наименьшим весом (находит такую вершину метод min\_f). Так, в current хранится текущая вершина.

Так как логика алгоритма предполагает, что начальная и конечная вершины не совпадают, это отражено в функционале приложения (нельзя выбрать одну вершину и в качестве начала, и в качестве конца). Таким образом, обрабатывать такой случай не требуется.

Такая вершина удаляется из очереди и заносится в массив пройденных вершин. Соответственно меняются поля in\_open и in\_closed.

Запускается подцикл for, проходящий по соседям текущей вершины. Для рассматриваемой соседней вершины рассчитывается (но не записывается в данные поля g) расстояние от начала до неё. Если вершина не находится в очереди, или рассчитанное расстояние меньше расстояния, записанного в данных поля g, то заполняются соответствующие зависимости соседа в полях from, f, g. Если рассматриваемая соседняя вершина не находится в очереди (и она не пройдена), встаёт в очередь. Итерация цикла завершается.

**3.3. Представление графа**

Для представления графа были написаны следующие классы: *Graph*, *Vertex*, *Coordinates<T>.*

Класс *Coordinates<T>* - класс для представления координат вершины, имеет два поля, для *х* и *y* координаты вершины соответственно. Методы: геттеры и сеттеры координат.

Класс *Vertex* — вершина. Хранит символьное имя вершины, ее координаты и хеш-таблицу ребер, исходящих из данной вершины: ключ — соседняя с данной вершина, значение — вес ребра. Методы: добавление соседа вершине, удаление соседа, геттер имени вершины, геттер соседей вершины, геттер и сеттер координат вершины.

Класс *Graph* — класс графа. Хранит список вершин (*Vertex*) графа, а также хеш-таблицу допустимых имен вершин в графе. Ключ — допустимое имя, значение — булева переменная, если *true* имя свободно, иначе — занято. Таким образом, все имена вершин графа разные. Методы: получение первого допустимого имени, добавление/удаление вершины по имени и ссылке, добавление/удаление ребра, поиск вершины с заданным именем, очистка графа.

**3.4. Визуализация графа**

Для графического отображения графа в приложении были написаны следующие классы: *VertexDrawable*, *EdgeDrawable*, *GraphController*.

Класс *VerteDrawable* — класс графического отображения вершины графа, содержит имя вершины, ссылку на отображаемую вершину, графическое отображение (кружок и подпись имени вершины), координаты, радиус для графического отображения (для круга). Методы: геттер и сеттер имени вершины, получение графического отображения вершины (круга), получение графического отображения имени вершины (подпись с именем), получение ссылки на отрисовываемую вершину, движение вершины (т. е. изменение координат вершины и ее перерисовка).

Класс *EdgeDrawable* — класс графического отображения ребра. Содержит ссылки на отрисовываемые вершины, направление ребра, веса ребер, линию для отрисовки (графический объект), подписи весов для отрисовки. Методы: установить противоположное направление для ребра, удалить одно направление, узнать является ребро двунаправленным или однонаправленным, сравнение вершин переданных в метод со ссылками в ребре, геттер графического отображения веса ребра, самого ребра (линии), движение линии (т. е. изменение ее координат), геттер ссылок на вершины, между которыми задано это ребро.

Все графические объекты хранятся и отрисовываются в классе *GraphController*, класс, который задает основную логику взаимодействия графики и логики приложения, содержит два списка: для ребер и вершин графа (*EdgeDrawable* и *VertexDrawable*), объект класса *Astar*, две выбранные вершины для запуска А\*, удаления/добавления ребра, объект класса *Graph* — граф. Позволяет считать граф, создавая граф и его графическое отображение. Добавить/удалить ребро/вершину. Двигать вершину на холсте. Запускает и отображает алгоритм А\*.

**3.5. Визуализация алгоритма**

Визуализация алгоритма возможна двумя способами: в пошаговом режиме и с мгновенным выводом результата работы программы.

Мгновенный вывод результата работы алгоритма.

Перед началом работы пользователю необходимо выбрать две вершины: начало и конец пути на графе, это делается с помощью двойного клика ЛКМ по двум вершинам, выбранные вершины меняют цвет, порядок выбора вершин важен.

В результате работы алгоритма в окно результатов выводится символьный путь между выбранными вершинами, а также его вес. Сам же найденный путь на графе подсвечивается зеленым.

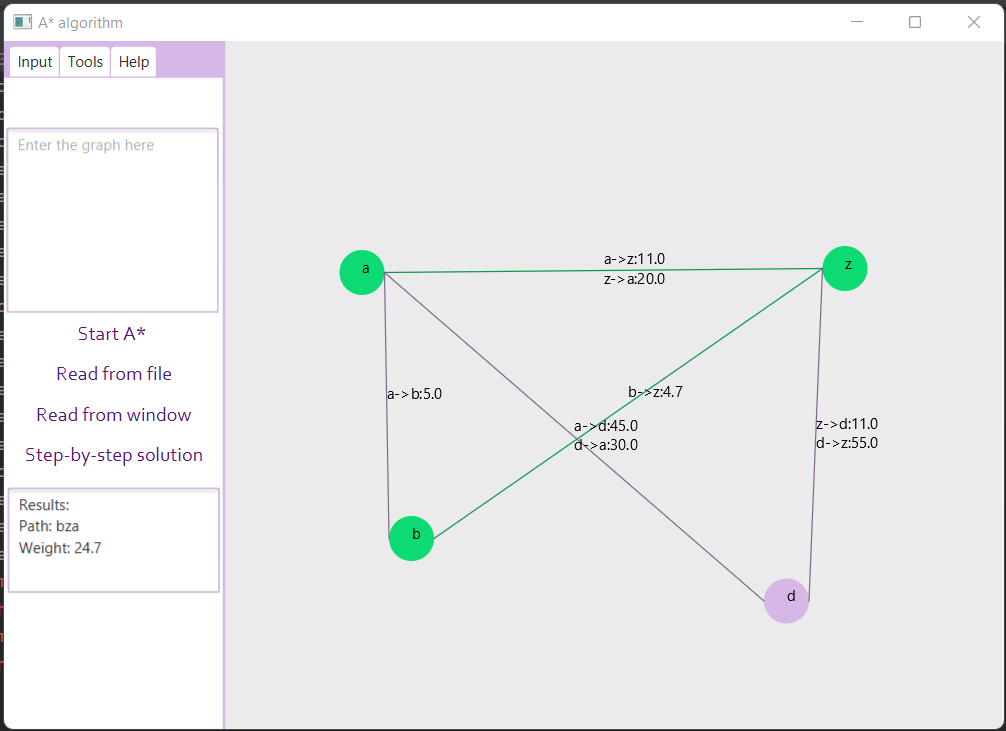


Рисунок 4. Построенный путь алгоритмом А\*.

Пошаговый алгоритм.

В начале работы также выбирается две вершины, но пошаговый режим позволяет двигаться по алгоритму пошагово, на каждом шаге выводится текущий найденный путь и эвристики вершин (f), шаги можно делать как вперед так и назад, а также перейти сразу к концу алгоритма. После того как пользователь переходит в пошаговый режим, возможность редактировать граф (в т. ч. двигать, удалять и считывать снова) блокируется до тех пор, пока пользователь не выйдет из пошагового режима, нажав кнопку «*End Step-by-step A\**».

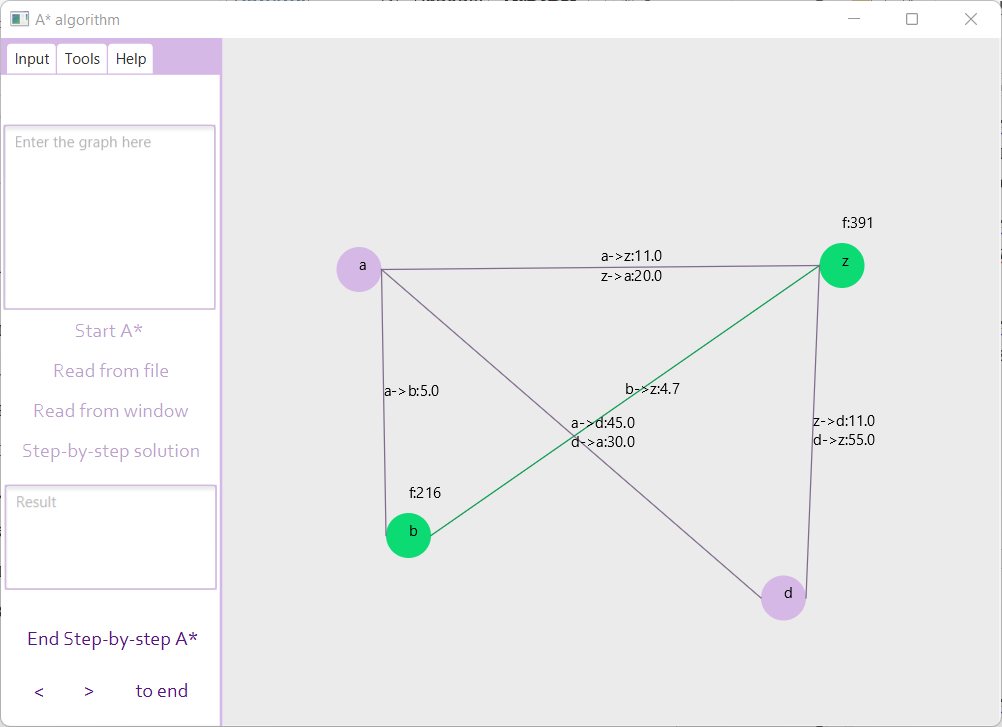


Рисунок 5. Режим пошаговой работы алгоритма.

**4. ТЕСТИРОВАНИЕ**

**4.1. Тестирование алгоритма А\***

В начале работы приложения происходит тестирование алгоритма А\*.

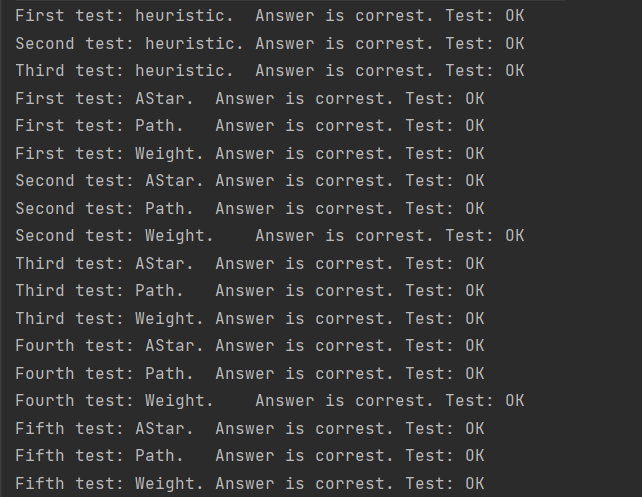


Рисунок 6. Вывод результата тестирования в консоль.

Для тестирования был написан общедоступный класс - *public class Tests* {

*public static final double DOUBLE\_CONST= 0.0000000001*; - константа, выступающая в роли эпсилон в сравнениях вещественных чисел.

*public static final int SLEEP\_CONST= 500;* - константа, задающая задержку вывода.

Каждый последующий метод возвращает строку, содержащую информацию о конкретном тесте и его результате. Тест проходит следующим образом: задаются тестовые данные. На них прогоняются методы алгоритма, результаты которых сравниваются с правильными выходными данными. В зависимости от того, совпали ли тестовые и готовые выходные данные, возвращается нужная строка.

*public static void RunTests() throws InterruptedException;* - главный и единственный *public* метод, который вызывает все реализованные тесты и выводит их результат в консоль с временной задержкой (с помощью *InterruptedException - Thread.sleep()).*

*private static String FirstTestH();* - тест, проверяющий работу статического метода *AStar*, вычисляющего эвристику на первом примере (описан в табл.1).

*private static String SecondTestH() и private static String ThirdTestH()* работают аналогично, но проверяют метод на втором и третьем примере.

*private static String FirstTestAStar(AStar n), private static String FirstTestPath(AStar n) и private static String FirstTestWeight(AStar n)* - перед вызовом этих методов, *в RunTests()* создаётся объект класса *AStar*, на котором будет проводиться проверка в данных методах. В каждый метод объект передаётся в качестве аргумента.

В методе *FirstTestAStar(AStar n)* создаётся конкретный граф для первого примера и проверяется основной метод *AStar a\_star\_public*, реализующий алгоритм.

В методе *FirstTestPath(AStar n)* на уже созданном графе проверяется метод *AStar – getPath(),* возвращающий найденный в ходе работы алгоритма кратчайший путь в виде строки.

В методе *FirstTestWeight(AStar n)* на уже созданном графе проверяется метод *AStar – getWeight(),* возвращающий найденный в ходе работы алгоритма вес кратчайшего пути.

*private static String SecondTestAStar(AStar n) ,*

*private static String SecondTestPath(AStar n) ,*

*private static String SecondTestWeight(AStar n),*

*private static String ThirdTestAStar(AStar n),*

*private static String ThirdTestPath(AStar n),*

*private static String ThirdTestWeight(AStar n),*

*private static String FourthTestAStar(AStar n),*

*private static String FourthTestPath(AStar n),*

*private static String FourthTestWeight(AStar n),*

*private static String FifthTestAStar(AStar n),*

*private static String FifthTestPath(AStar n),*

*private static String FifthTestWeight(AStar n);* - группа методов, работающих аналогично и проверяющая те же методы класса *AStar*, но на примере уже других графов.

}

Таблица 1 – Результаты тестирования алгоритма.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Входные данные | Комментарий | Результат | Описание |
| Тест 1. Эвристика | Vertex('a', 0, 0)  Vertex('b', 0, 0) | Программа  работает верно. | First test: heuristic.  Answer is correst. Test: OK | Случай, когда вершины находятся в одной точке с нулевыми координатами, и их эвристика равна нулю. |
| Тест 2. Эвристика | Vertex('a', 100.234, 2)  Vertex('b', 189.2, 0) | Программа  работает верно. | Second test: heuristic.  Answer is correst. Test: OK | Случай, проверяющий точность эвристики с вещественными числами. |
| Тест 3. Эвристика | Vertex('a', 100, 300)  Vertex('b', 100 + Math.sqrt(600), 305)) | Программа  работает верно. | Third test: heuristic.  Answer is correst. Test: OK | Общий случай. |
| Проверка методов на графе – пример 1. | Начальная вершина: a  Конечная: b  Информация о вершинах и ребрах:  2  a 0 0  b 0 0  a b 0 | Программа  работает верно. | First test: AStar.  Answer is correst. Test: OK  First test: Path.  Answer is correst. Test: OK  First test: Weight.  Answer is correst. Test: OK | Случай, когда обе точки находятся на одном месте: координаты и вес ребра нулевые. |
| Проверка методов на графе – пример 2. | Начальная вершина: a  Конечная: e  Информация о вершинах и ребрах:  5  a 100.234 2  b 189.2 0  c 200 300  d 100 300  e 500 500  a b 3  b c 1  c d 1  a d 5  d e 1 | Программа  работает верно. | Second test: AStar.  Answer is correst. Test: OK  Second test: Path.  Answer is correst. Test: OK  Second test: Weight.  Answer is correst. Test: OK | Общий случай. |
| Проверка методов на графе – пример 3. | Начальная вершина: a  Конечная: d  Информация о вершинах и ребрах:  8  a 100 300  b 400 500  c 350 250  d 80 90  e 330 220  x 105 375  y 296 100  z 500 150  a b 3  a c 5  a d 7  a e 1  x z 1  y z 4 | Программа  работает верно. | Third test: AStar.  Answer is correst. Test: OK  Third test: Path.  Answer is correst. Test: OK  Third test: Weight.  Answer is correst. Test: OK | Случай, при котором граф  имеет различные ребра с  различным весом, но при  этом только одно ребро  является путём от начальной  до конечной. |
| Проверка методов на графе – пример 4. | Начальная вершина: a  Конечная: f  Информация о вершинах и ребрах:  6  a 1 1  b 1 2  c 2 3  d 3 4  e 4 5  f 5 6  a b 1  b c 2  c d 3  d e 4  e f 5  a f 34 | Программа  работает верно. | Fourth test: AStar.  Answer is correst. Test: OK  Fourth test: Path.  Answer is correst. Test: OK  Fourth test: Weight.  Answer is correst. Test: OK | Случай, при котором  имеется прямой путь от  начальной вершины до  конечной, но он значительно  превышает суммарный вес  всех промежуточных рёбер. |
| Проверка методов на графе – пример 5. | Начальная вершина: a  Конечная: e  Информация о вершинах и ребрах:  4  a 300 300  b 200 200  d 400 200  e 300 100  a b 4  a d 5  b e 1  d e 0.5 | Программа  работает верно. | Fifth test: AStar.  Answer is correst. Test: OK  Fifth test: Path.  Answer is correst. Test: OK  Fifth test: Weight.  Answer is correst. Test: OK | Случай, при котором  существует ровно 2, равных  по числу вершин, пути до  конечной вершины. |

Крайние случаи с отрицательными значениями тестировать не требуется, так как формат ввода контролируется исключениями.

**4.2. Тестирование приложения**

Была написана обработка исключений для корректной работы приложения. Требования к правильному вводу данных проверяются в классе *CheckRules*, бросающий исключения типа *DataFormatException*. В случае, когда пользователь ввел некорректные данные в файл или в окно, появляется сообщение об описании ошибки. Ошибки продемонстрированы на рисунках 7 -12.

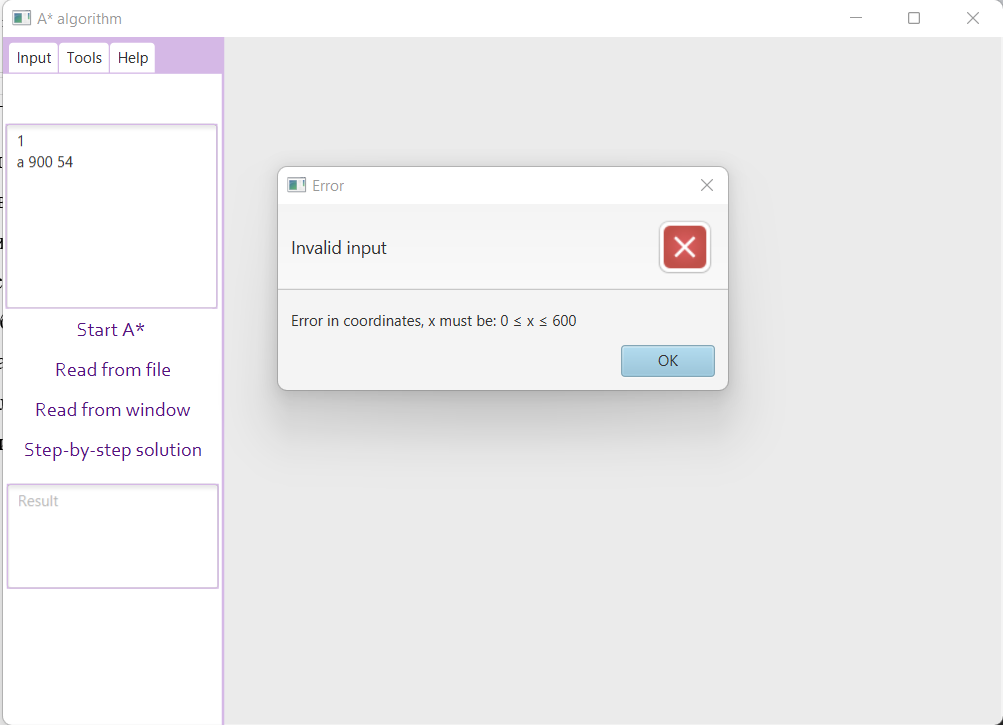


Рисунок 7. Попытка ввода неправильных координат.

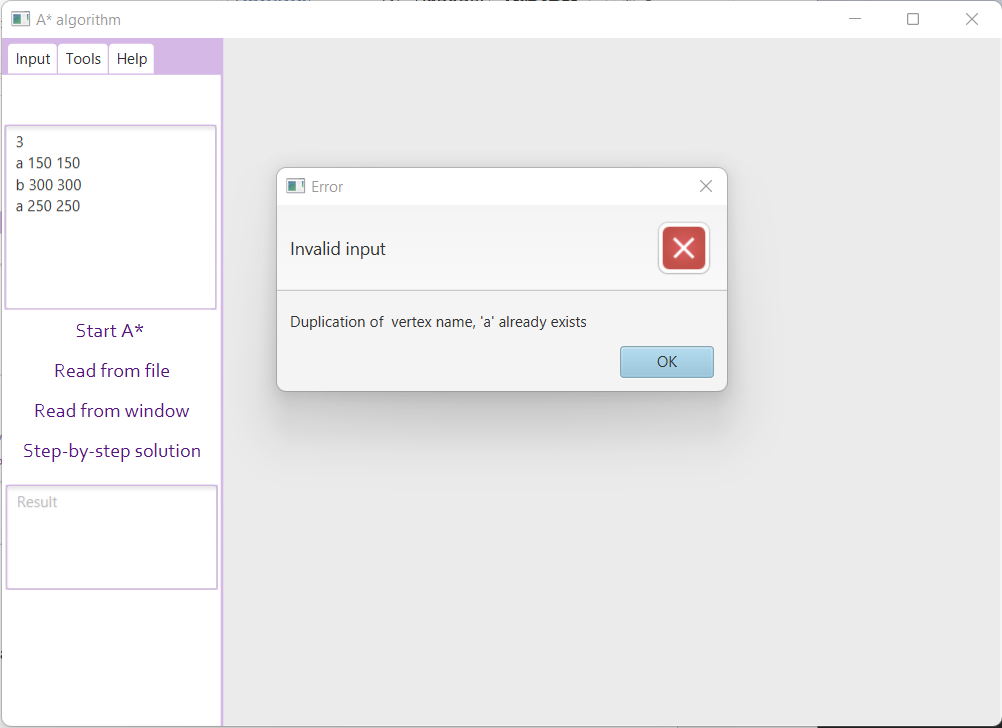


Рисунок 8. Попытка повторно создать вершину.

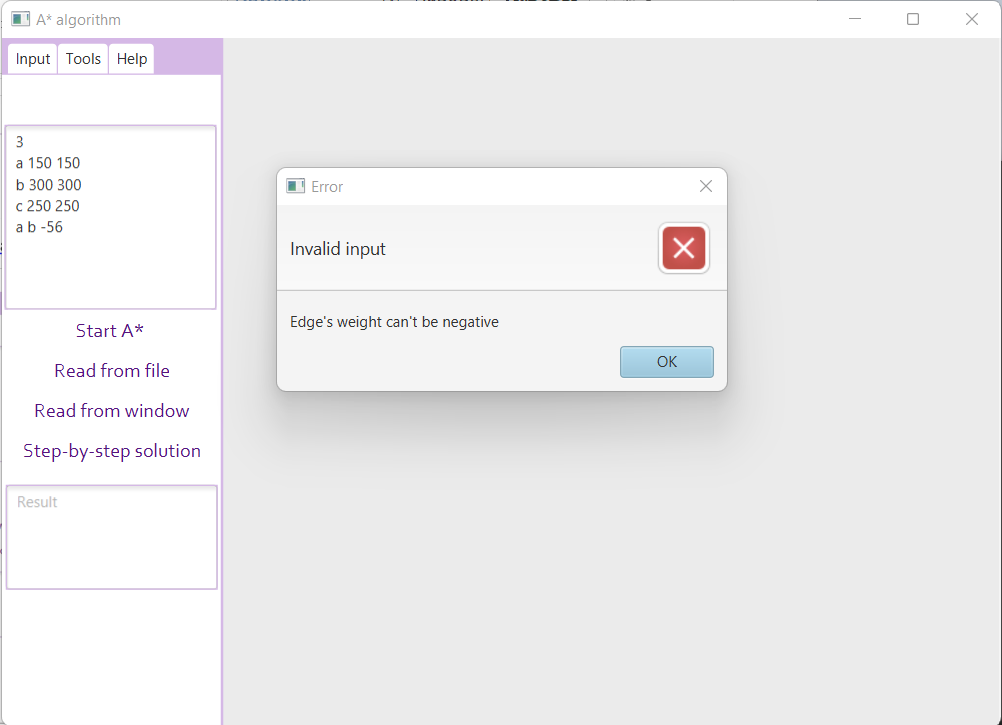


Рисунок 9. Попытка присвоить ребру отрицательный вес.

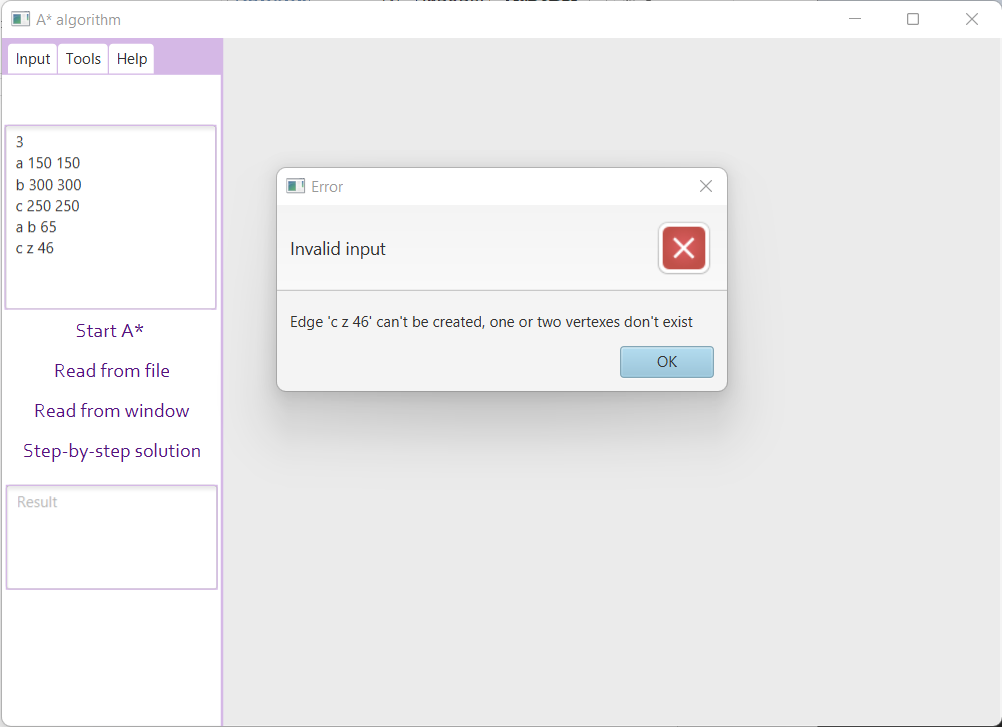


Рисунок 10. Попытка создать ребро с несуществующей вершиной.

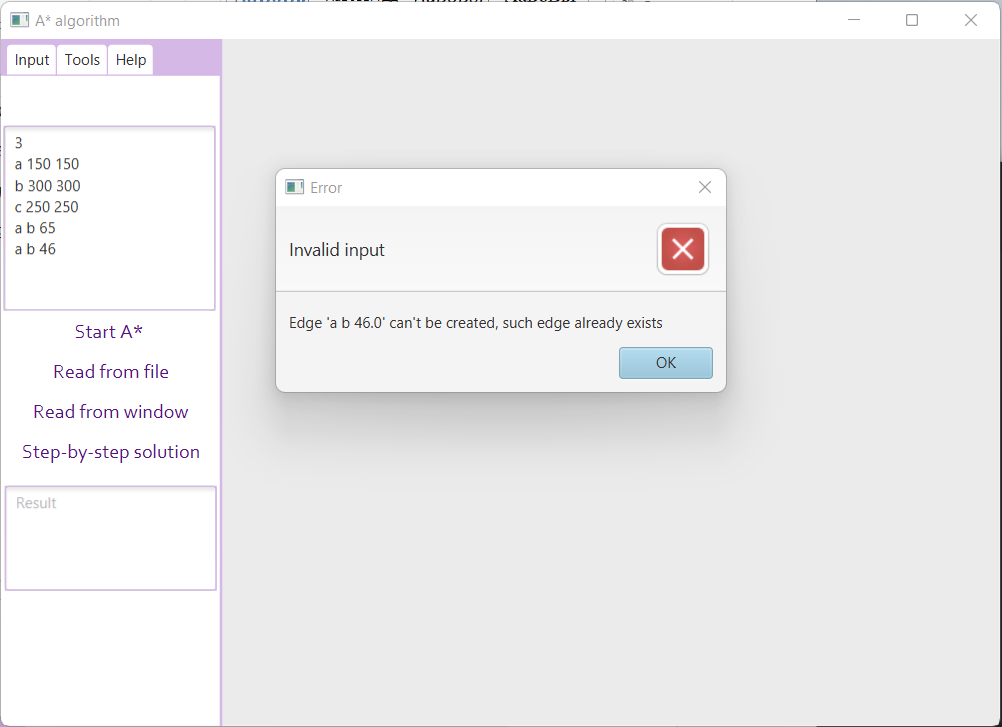


Рисунок 11. Попытка ввести второй раз уже существующее ребро.

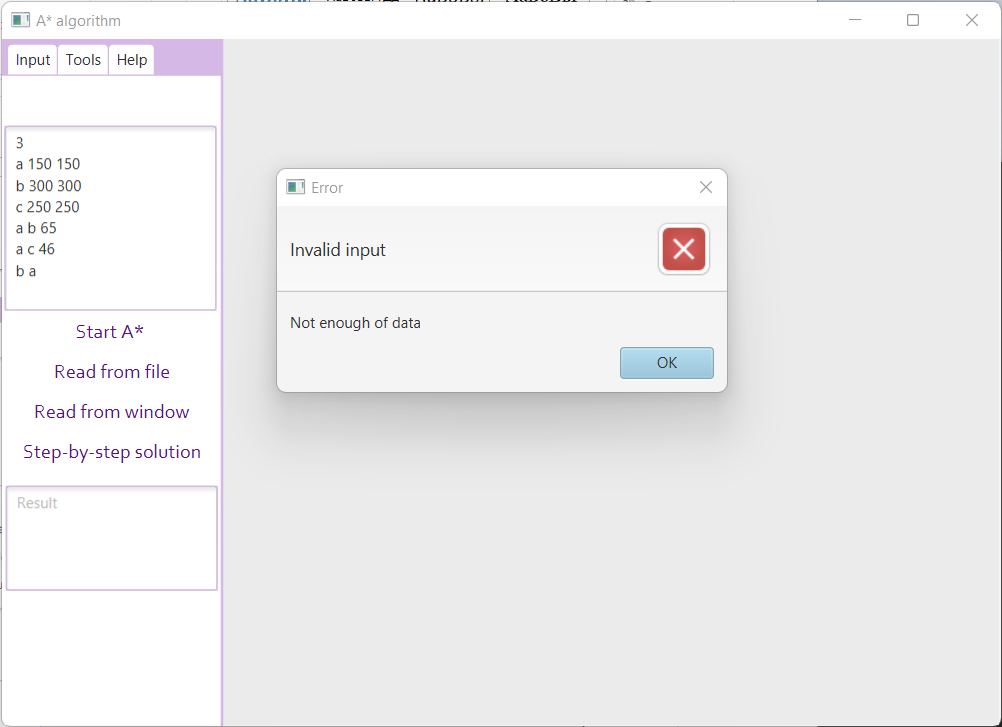


Рисунок 12. Попытка ввести не все необходимые данные для создания графа.

**ЗАКЛЮЧЕНИЕ**

В ходе выполнения практической работы, выл реализован алгоритм на графах А\*. Был получен навык создания Maven файла, для включения зависимостей и параметров сборки. Был написан GUI интерфейс для взаимодействия с программой, в котором можно создавать и редактировать граф, так же он обладает функционалом, который наглядно показывает работу алгоритма в пошаговом режиме. Для написания GUI была изучена библиотека JavaFX.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Документация JDK 11 [Электронный ресурс]. URL: <https://docs.oracle.com/en/java/javase/11/> (Дата обращения: 01.07.2022).

2. Пакет JavaFX [Электронный ресурс]. URL: <https://docs.oracle.com/javase/8/javafx/api/toc.htm> (Дата обращения: 02.07.2022).

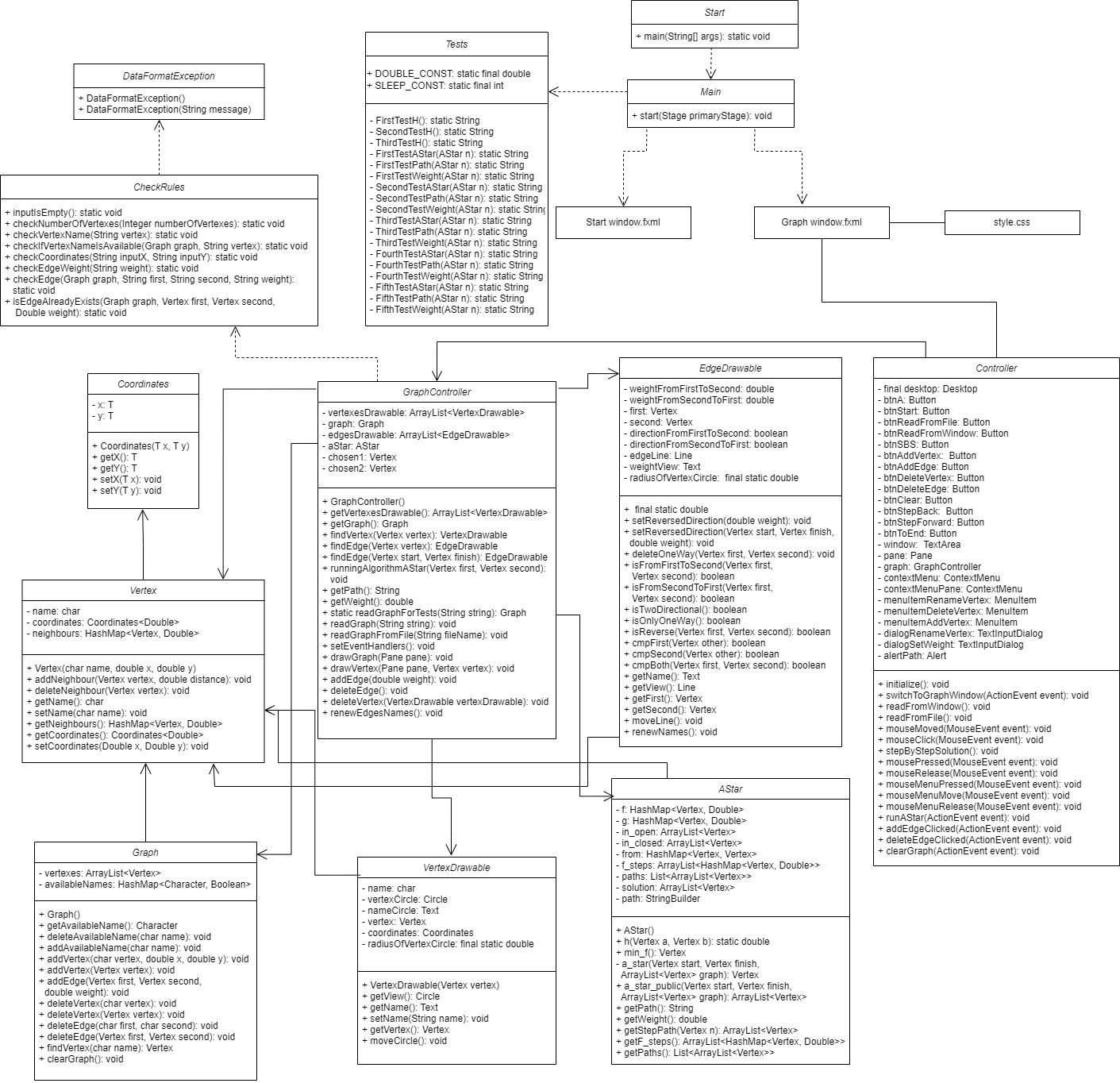
3. Документация Maven [Электронный ресурс]. URL: <https://maven.apache.org/guides/index.html> (Дата обращения: 04.07.2022).

4. Информация об алгоритме A\* [Электронный ресурс]. URL: [https://ru.wikipedia.org/wiki/A\*](https://ru.wikipedia.org/wiki/A*) (Дата обращения: 30.06.2022).

5. Герберт Шилдт" Java. Руководство для начинающих: учебник. - СПб.: Диалектика, 2019. - 816 с.

**ПРИЛОЖЕНИЕ А**

**UML-диаграмма**



**ПРИЛОЖЕНИЕ B**

**Исходный код**

Название файла: *AStar.java*

*package application.app;*

*import java.util.ArrayList;*

*import java.util.Arrays;*

*import java.util.HashMap;*

*import java.util.List;*

*public class AStar {*

*private HashMap<Vertex, Double> f;*

*private HashMap<Vertex, Double> g;*

*private ArrayList<Vertex> in\_open;*

*private ArrayList<Vertex> in\_closed;*

*private HashMap<Vertex, Vertex> from;*

*private ArrayList<HashMap<Vertex, Double>> f\_steps; // список значений эвристической функции на каждом шаге*

*private List<ArrayList<Vertex>> paths; // список путей на каждом шаге*

*private ArrayList<Vertex> solution;*

*private StringBuilder path;*

*public AStar(){*

*f = new HashMap<Vertex, Double>();*

*g = new HashMap<Vertex, Double>();*

*in\_open = new ArrayList<Vertex>();*

*in\_closed = new ArrayList<Vertex>();*

*from = new HashMap<Vertex, Vertex>();*

*f\_steps = new ArrayList();*

*paths = new ArrayList();*

*solution = new ArrayList<Vertex>();*

*path = new StringBuilder();*

*}*

*public static double h(Vertex a, Vertex b){ // Р­РІСЂРёСЃС‚РёС‡РµСЃРєР°СЏ С„СѓРЅРєС†РёСЏ*

*return Math.sqrt( Math.pow(a.getCoordinates().getX() - b.getCoordinates().getX(), 2) +*

*Math.pow(a.getCoordinates().getY() - b.getCoordinates().getY(), 2));*

*}*

*public Vertex min\_f(){*

*Double min = f.get(in\_open.get(0));*

*Vertex min\_v = in\_open.get(0);*

*for(int i = 0; i < in\_open.size(); i++){*

*if (f.get(in\_open.get(i)) < min) {*

*min = f.get(in\_open.get(i));*

*min\_v = in\_open.get(i);*

*}*

*}*

*return min\_v;*

*}*

*private Vertex a\_star(Vertex start, Vertex finish){*

*Vertex current;*

*in\_open.add(start);*

*g.put(start, (double)0); f.put(start, g.get(start) + h(start, finish));*

*while (in\_open.size() > 0){*

*current = min\_f();*

*paths.add(getStepPath(current));*

*HashMap<Vertex,Double> f\_step = new HashMap<>(f);*

*f\_steps.add(f\_step);*

*if (current == finish) return finish;*

*in\_open.remove(current);*

*in\_closed.add(current);*

*for (HashMap.Entry<Vertex, Double> neighbour : current.getNeighbours().entrySet()) {*

*double temp\_g = g.get(current) + neighbour.getValue();*

*if ( !in\_open.contains(neighbour.getKey())*

*&& !in\_closed.contains(neighbour.getKey())*

*|| (g.containsKey(neighbour.getKey()) && temp\_g < g.get(neighbour.getKey()))){*

*from.put(neighbour.getKey(), current);*

*g.put(neighbour.getKey(), temp\_g);*

*f.put(neighbour.getKey(), g.get(neighbour.getKey()) + h(neighbour.getKey(), finish));*

*}*

*if (!in\_open.contains(neighbour.getKey())*

*&& !in\_closed.contains(neighbour.getKey())){*

*in\_open.add(neighbour.getKey());*

*}*

*}*

*}*

*return null;*

*}*

*public ArrayList<Vertex> a\_star\_public(Vertex start, Vertex finish){*

*Vertex goal = a\_star(start, finish);*

*if (goal == null) return solution;*

*solution.add(goal);*

*while(from.containsKey(goal)){*

*solution.add(0, from.get(goal));*

*goal = from.get(goal);*

*}*

*return solution;*

*}*

*public String getPath(){*

*path = new StringBuilder();*

*for (Vertex i : solution)*

*path.append(i.getName());*

*return path.toString();*

*}*

*public ArrayList<Vertex> getStepPath(Vertex n){*

*ArrayList<Vertex> step\_path = new ArrayList<>();*

*step\_path.add(n);*

*while(from.containsKey(n)){*

*step\_path.add(0, from.get(n));*

*n = from.get(n);*

*}*

*return step\_path;*

*}*

*public double getWeight(){*

*double weight = 0;*

*for (int i = 0; i < solution.size() - 1; i++)*

*weight += solution.get(i).getNeighbours().get(solution.get(i + 1));*

*return weight;*

*}*

*public ArrayList<HashMap<Vertex, Double>> getF\_steps(){*

*return f\_steps;}*

*public List<ArrayList<Vertex>> getPaths(){*

*return paths;*

*}*

*}*  
Название файла: *CheckRules.java*

*package application.app;*

*public class CheckRules {*

*public static void inputIsEmpty() throws DataFormatException {*

*throw new DataFormatException("You haven't entered any data");*

*}*

*public static void checkNumberOfVertexes(Integer numberOfVertexes) throws NumberFormatException, DataFormatException {*

*if (numberOfVertexes < 0){*

*throw new DataFormatException("Number of vertex can't be negative");*

*}*

*}*

*public static void checkVertexName(String vertex) throws DataFormatException{*

*if (vertex.length() > 1)*

*throw new DataFormatException("Vertex name must be one character, name '" + vertex + "' is invalid");*

*if (!('a' <= vertex.charAt(0) && vertex.charAt(0) <= 'z'))*

*throw new DataFormatException("Name of the vertex must be the English letter, '" + vertex + "' is invalid");*

*}*

*public static void checkIfVertexNameIsAvailable(Graph graph, String vertex) throws DataFormatException{*

*if (!graph.isNameAvailable(vertex.charAt(0)))*

*throw new DataFormatException("Duplication of vertex name, '" + vertex.charAt(0) + "' already exists");*

*}*

*public static void checkCoordinates(String inputX, String inputY) throws NumberFormatException, DataFormatException{*

*try {*

*double x = Double.parseDouble(inputX);*

*double y = Double.parseDouble(inputY);*

*} catch (NumberFormatException e) {*

*throw new NumberFormatException();*

*}*

*if (Double.parseDouble(inputX) < 0 || Double.parseDouble(inputX) > 600)*

*throw new DataFormatException("Error in coordinates, x must be: 0 ≤ x ≤ 600");*

*if (Double.parseDouble(inputY) < 0 || Double.parseDouble(inputY) > 530)*

*throw new DataFormatException("Error in coordinates, y must be: 0 ≤ y ≤ 530");*

*}*

*public static void checkEdgeWeight(String weight) throws NumberFormatException, DataFormatException {*

*double weightNumber = Double.parseDouble(weight);*

*if (weightNumber < 0) throw new DataFormatException("Edge's weight can't be negative");*

*if (weightNumber > Double.MAX\_VALUE) throw new DataFormatException("Invalid edge value");*

*}*

*public static void checkEdge(Graph graph, String first, String second, String weight)*

*throws NumberFormatException, DataFormatException {*

*checkVertexName(first);*

*checkVertexName(second);*

*Vertex firstVertex = graph.findVertex(first.charAt(0));*

*Vertex secondVertex = graph.findVertex(second.charAt(0));*

*if (firstVertex == null || secondVertex == null){*

*StringBuilder edge = new StringBuilder();*

*edge.append(("Edge '" + first + " " + second + " " + weight + "'").toString());*

*throw new DataFormatException(edge.toString() + " can't be created, one or two vertexes don't exist");*

*} else if (firstVertex == secondVertex) {*

*StringBuilder edge = new StringBuilder();*

*edge.append(("Edge '" + first + " " + second + " " + weight + "'").toString());*

*throw new DataFormatException(edge.toString() + " can't be created, vertexes must be different");*

*}*

*checkEdgeWeight(weight);*

*}*

*public static void isEdgeAlreadyExists(Graph graph, Vertex first, Vertex second, Double weight) throws DataFormatException{*

*if (graph.isEdgeAlreadyExists(first, second)) {*

*StringBuilder edge = new StringBuilder();*

*edge.append(("Edge '" + first.getName() + " " + second.getName() + " " + String.valueOf(weight) + "'").toString());*

*throw new DataFormatException(edge.toString() + " can't be created, such edge already exists");*

*}*

*}*

*}*

Название файла: *Controller.java*

*package application.app;*

*import javafx.event.ActionEvent;*

*import javafx.event.EventHandler;*

*import javafx.fxml.FXML;*

*import javafx.fxml.FXMLLoader;*

*import javafx.scene.Parent;*

*import javafx.scene.Scene;*

*import javafx.scene.control.\*;*

*import javafx.scene.control.Button;*

*import javafx.scene.control.MenuItem;*

*import javafx.scene.control.TextArea;*

*import javafx.scene.input.MouseButton;*

*import javafx.scene.input.MouseEvent;*

*import javafx.scene.layout.Pane;*

*import javafx.stage.FileChooser;*

*import javafx.stage.Stage;*

*import javafx.scene.control.Alert;*

*import javafx.scene.control.Alert.AlertType;*

*import java.awt.\*;*

*import java.io.File;*

*import java.io.IOException;*

*import java.util.Objects;*

*import java.util.Optional;*

*public class Controller {*

*private final Desktop desktop = Desktop.getDesktop(); // информация об устройстве для считывания файла*

*public Button endStepByStepButton;*

*@FXML*

*private Button btnA, btnStart, btnReadFromFile, btnReadFromWindow, btnSBS;*

*@FXML*

*private Button btnAddVertex, btnAddEdge, btnDeleteVertex, btnDeleteEdge, btnClear;*

*@FXML*

*private Button btnStepBack, btnStepForward, btnToEnd;*

*@FXML*

*private TextArea window, resultWindow;*

*@FXML*

*private Pane pane;*

*private GraphController graph;*

*@FXML*

*private ContextMenu contextMenu, contextMenuPane;*

*@FXML*

*private MenuItem menuItemRenameVertex, menuItemDeleteVertex;*

*@FXML*

*private MenuItem menuItemAddVertex;*

*@FXML*

*private TextInputDialog dialogRenameVertex, dialogSetWeight;*

*@FXML*

*private Alert alertPath, alertError;*

*private boolean eventFlag;*

*@FXML*

*public void initialize(){*

*graph = new GraphController();*

*contextMenu = new ContextMenu();*

*contextMenuPane = new ContextMenu();*

*menuItemRenameVertex = new MenuItem("Rename vertex");*

*menuItemDeleteVertex = new MenuItem("Delete vertex");*

*contextMenu.getItems().addAll(menuItemRenameVertex, menuItemDeleteVertex);*

*dialogRenameVertex = new TextInputDialog();*

*dialogSetWeight = new TextInputDialog();*

*menuItemAddVertex = new MenuItem("Add vertex");*

*contextMenuPane.getItems().add(menuItemAddVertex);*

*alertPath = new Alert(AlertType.INFORMATION);*

*eventFlag = true;*

*alertError = new Alert(AlertType.ERROR);*

*alertError.setTitle("Error");*

*}*

*@FXML*

*public void switchToGraphWindow(ActionEvent event) throws IOException {*

*Parent root = FXMLLoader.load(Objects.requireNonNull(getClass().getResource("Graph window.fxml")));*

*Stage stage = (Stage) btnStart.getScene().getWindow();*

*stage.setScene(new Scene(root, 800, 550));*

*}*

*@FXML*

*public void readFromWindow() throws NumberFormatException{*

*try {*

*graph = new GraphController();*

*graph.readGraph(window.getText());*

*graph.drawGraph(pane);*

*} catch (ArrayIndexOutOfBoundsException e){*

*alertError.setHeaderText("Invalid input");*

*alertError.setContentText("Not enough of data");*

*alertError.showAndWait();*

*graph = new GraphController();*

*graph.drawGraph(pane);*

*} catch (NumberFormatException e){*

*alertError.setHeaderText("Invalid input");*

*alertError.setContentText("Error in numerical values");*

*alertError.showAndWait();*

*graph = new GraphController();*

*graph.drawGraph(pane);*

*} catch (DataFormatException e){*

*alertError.setHeaderText("Invalid input");*

*alertError.setContentText(e.getMessage());*

*alertError.showAndWait();*

*graph.getGraph().clearGraph();*

*graph = new GraphController();*

*graph.drawGraph(pane);*

*}*

*}*

*@FXML*

*public void readFromFile() throws NumberFormatException, IOException{*

*FileChooser fileChooser = new FileChooser();*

*FileChooser.ExtensionFilter extensionFilter = new FileChooser.ExtensionFilter("TXT files (\*.txt)", "\*.txt");*

*fileChooser.getExtensionFilters().add(extensionFilter);*

*File file = fileChooser.showOpenDialog(pane.getScene().getWindow());*

*String path;*

*if(file != null) {*

*path = file.getPath();*

*} else {*

*return;*

*}*

*try {*

*graph = new GraphController();*

*graph.readGraphFromFile(path);*

*graph.drawGraph(pane);*

*} catch (ArrayIndexOutOfBoundsException e){*

*alertError.setHeaderText("Invalid input");*

*alertError.setContentText("Not enough of data");*

*alertError.showAndWait();*

*graph = new GraphController();*

*graph.drawGraph(pane);*

*} catch (NumberFormatException e){*

*alertError.setHeaderText("Invalid input");*

*alertError.setContentText("Error in numerical values");*

*alertError.showAndWait();*

*graph = new GraphController();*

*graph.drawGraph(pane);*

*} catch (DataFormatException e){*

*alertError.setHeaderText("Invalid input");*

*alertError.setContentText(e.getMessage());*

*alertError.showAndWait();*

*graph.getGraph().clearGraph();*

*graph = new GraphController();*

*graph.drawGraph(pane);*

*} catch (IOException e) {*

*alertError.setHeaderText("Error occurred in opening this file");*

*alertError.setContentText(e.getMessage());*

*alertError.showAndWait();*

*graph.getGraph().clearGraph();*

*graph = new GraphController();*

*graph.drawGraph(pane);*

*}*

*}*

*@FXML*

*public void mouseMoved(MouseEvent event){*

*graph.drawGraphAndLabels(pane);*

*}*

*@FXML*

*public void mouseClick(MouseEvent event){*

*if (eventFlag) {*

*if (event.getButton() == MouseButton.PRIMARY){*

*if (contextMenu.isShowing()) {*

*contextMenu.hide();*

*}*

*if (event.getClickCount() == 2){*

*graph.setEventHandlers();*

*}*

*}*

*if (event.getButton() == MouseButton.SECONDARY){*

*if (event.getClickCount() == 1) {*

*setHandlers();*

*} if (event.getClickCount() == 2) {*

*if (contextMenu.isShowing()) {*

*contextMenu.hide();*

*}*

*char name = graph.getGraph().getAvailableName();*

*if (name == '\*') {*

*alertError.setHeaderText("Maximum number of vertexes per field");*

*alertError.setContentText("You can't add new vertex");*

*alertError.showAndWait();*

*} else {*

*Vertex vertex = new Vertex(name, event.getX(), event.getY());*

*graph.drawVertex(pane, vertex);*

*graph.drawGraph(pane);*

*}*

*}*

*}*

*}*

*}*

*private void setHandlers() {*

*for (VertexDrawable vertexDrawable : graph.getVertexesDrawable()) {*

*vertexDrawable.getView().setOnMouseClicked(new EventHandler<MouseEvent>() {*

*@Override*

*public void handle(MouseEvent mouseEvent) {*

*contextMenu.show(pane, mouseEvent.getScreenX(), mouseEvent.getScreenY());*

*menuItemRenameVertex.setOnAction((ActionEvent actionEvent) -> {*

*if (contextMenu.isShowing()) {*

*contextMenu.hide();*

*}*

*Vertex vertex = vertexDrawable.getVertex();*

*char oldName = vertex.getName();*

*//добавить проверку, что такое имя не занято*

*dialogRenameVertex.setTitle("Rename vertex");*

*dialogRenameVertex.setHeaderText("Enter vertex name:");*

*dialogRenameVertex.setContentText("Name:");*

*Optional<String> newName = dialogRenameVertex.showAndWait();*

*newName.ifPresent(name -> {*

*try {*

*CheckRules.checkVertexName(name);*

*CheckRules.checkIfVertexNameIsAvailable(graph.getGraph(), name);*

*graph.getGraph().addAvailableName(oldName);*

*vertexDrawable.setName(name);*

*graph.getGraph().deleteAvailableName(name.charAt(0));*

*graph.renewEdgesNames();*

*graph.drawGraph(pane);*

*} catch (DataFormatException e) {*

*alertError.setHeaderText("Invalid name");*

*alertError.setContentText(e.getMessage());*

*alertError.showAndWait();*

*}*

*});*

*});*

*menuItemDeleteVertex.setOnAction((ActionEvent actionEvent) -> {*

*if (contextMenu.isShowing()) {*

*contextMenu.hide();*

*}*

*graph.deleteVertex(vertexDrawable);*

*graph.drawGraph(pane);*

*});*

*}*

*});*

*}*

*}*

*@FXML*

*public void stepByStepSolution(){*

*resultWindow.clear();*

*eventFlag = false;*

*btnClear.setDisable(true);*

*btnAddEdge.setDisable(true);*

*btnDeleteEdge.setDisable(true);*

*btnA.setDisable(true);*

*btnReadFromFile.setDisable(true);*

*btnReadFromWindow.setDisable(true);*

*btnSBS.setDisable(true);*

*graph.startStepByStep();*

*btnStepBack.setVisible(true);*

*btnStepForward.setVisible(true);*

*btnToEnd.setVisible(true);*

*endStepByStepButton.setVisible(true);*

*graph.drawGraphAndLabels(pane);*

*}*

*@FXML*

*public void stepBack() {*

*graph.doStep(-1);*

*graph.drawGraphAndLabels(pane);*

*}*

*@FXML*

*public void stepForward() {*

*graph.doStep(1);*

*graph.drawGraphAndLabels(pane);*

*if (graph.isFinalInStepByStep()) {*

*String path = graph.getPath();*

*if (path == null || path.equals("")){*

*resultWindow.setText("Path doesn't exist");*

*} else {*

*resultWindow.setText("Results:" + "\n" + "Path: " + path + "\n" + "Weight: " + Double.toString(graph.getWeight()));*

*}*

*}*

*}*

*@FXML*

*public void toEnd() {*

*graph.toEndStepByStep();*

*graph.drawGraphAndLabels(pane);*

*String path = graph.getPath();*

*if (path == null || path.equals("")){*

*resultWindow.setText("Path doesn't exist");*

*} else {*

*resultWindow.setText("Results:" + "\n" + "Path: " + path + "\n" + "Weight: " + Double.toString(graph.getWeight()));*

*}*

*}*

*@FXML*

*public void endStepByStep() {*

*eventFlag = true;*

*btnClear.setDisable(false);*

*btnAddEdge.setDisable(false);*

*btnDeleteEdge.setDisable(false);*

*btnA.setDisable(false);*

*btnReadFromFile.setDisable(false);*

*btnReadFromWindow.setDisable(false);*

*btnSBS.setDisable(false);*

*btnStepBack.setVisible(false);*

*btnStepForward.setVisible(false);*

*btnToEnd.setVisible(false);*

*endStepByStepButton.setVisible(false);*

*graph.normalGraphColor();*

*graph.allowEvents();*

*graph.endSBS();*

*graph.drawGraph(pane);*

*}*

*@FXML*

*public void mousePressed(MouseEvent event) {*

*btnStart.setStyle("-fx-background-color: linear-gradient(to bottom left, #3fbab4, #410b61)");*

*}*

*@FXML*

*public void mouseRelease(MouseEvent event) {*

*btnStart.setStyle("-fx-background-color: linear-gradient(to bottom left, #52fff6, #7012a6)");*

*}*

*@FXML*

*public void mouseMenuPressed(MouseEvent event) {*

*Button button = (Button) event.getSource();*

*button.setStyle("-fx-background-color: linear-gradient(to bottom left, #52fff6, #9a35d4)");*

*}*

*@FXML*

*public void mouseMenuMove(MouseEvent event) {*

*Button button = (Button) event.getSource();*

*button.setStyle("-fx-background-color: linear-gradient(to bottom left, #a2fcf8, #d5b8e6)");*

*}*

*@FXML*

*public void mouseMenuRelease(MouseEvent event) {*

*Button button = (Button) event.getSource();*

*button.setStyle("-fx-background-color: white");*

*}*

*@FXML*

*public void runAStar(ActionEvent event) throws IOException {*

*Vertex vertexA = graph.getGraph().findVertex('a');*

*Vertex vertexB = graph.getGraph().findVertex('z');*

*graph.runningAlgorithmAStar(vertexA, vertexB);*

*alertPath.setTitle("Astar solution");*

*alertPath.setHeaderText("Results:");*

*String path = graph.getPath();*

*if (path == null || path.equals("")){*

*resultWindow.setText("Path doesn't exist");*

*alertPath.setContentText("Path doesn't exist");*

*alertPath.showAndWait();*

*} else {*

*resultWindow.setText("Results:" + "\n" + "Path: " + path + "\n" + "Weight: " + Double.toString(graph.getWeight()));*

*alertPath.setContentText("Path: " + path + "\n" + "Weight: " + Double.toString(graph.getWeight()));*

*alertPath.showAndWait();*

*}*

*}*

*@FXML*

*public void addEdgeClicked(ActionEvent event){*

*dialogSetWeight.setTitle("Set weight");*

*dialogSetWeight.setHeaderText("Enter weight:");*

*dialogSetWeight.setContentText("Weight:");*

*Optional <String> newWeight = dialogSetWeight.showAndWait();*

*newWeight.ifPresent(weight -> {*

*try {*

*CheckRules.checkEdgeWeight(weight);*

*graph.addEdge(Double.valueOf(weight));*

*graph.drawGraph(pane);*

*} catch (DataFormatException e) {*

*alertError.setHeaderText("Invalid weight");*

*alertError.setContentText(e.getMessage());*

*alertError.showAndWait();*

*} catch (NumberFormatException e){*

*alertError.setHeaderText("Invalid weight");*

*alertError.setContentText("Weight must be a number");*

*alertError.showAndWait();*

*}*

*});*

*}*

*@FXML*

*public void deleteEdgeClicked(ActionEvent event) {*

*graph.deleteEdge();*

*graph.drawGraph(pane);*

*}*

*@FXML*

*public void clearGraph(ActionEvent event){*

*graph = new GraphController();*

*graph.drawGraph(pane);*

*}*

*}*

Название файла: *Coordinates.java*

*package application.app;*

*public class Coordinates <T>{*

*private T x;*

*private T y;*

*Coordinates (T x, T y) {*

*this.x = x;*

*this.y = y;*

*}*

*public T getX(){*

*return x;*

*}*

*public T getY(){*

*return y;*

*}*

*public void setX(T x) {*

*this.x = x;*

*}*

*public void setY(T y) {*

*this.y = y;*

*}*

*}*

Название файла: *DataFormatException.java*

*package application.app;*

*public class DataFormatException extends Exception{*

*public DataFormatException(){*

*super();*

*}*

*public DataFormatException(String message){*

*super(message);*

*}*

*}*

Название файла: *EdgeDrawable.java*

*package application.app;*

*//import project.application.model.Vertex;*

*import javafx.scene.paint.Paint;*

*import javafx.scene.shape.Line;*

*import javafx.scene.text.Text;*

*public class EdgeDrawable {*

*private double weightFromFirstToSecond, weightFromSecondToFirst;*

*private Vertex first, second; // first - source, second - destination or versa*

*private boolean directionFromFirstToSecond, directionFromSecondToFirst;*

*private Line edgeLine;*

*private Text weightView;*

*private final static double radiusOfVertexCircle = 18.0;*

*public EdgeDrawable(Vertex first, Vertex second, double weight){*

*this.first = first;*

*this.second = second;*

*weightFromFirstToSecond = weight;*

*directionFromFirstToSecond = true;*

*edgeLine = new Line();*

*weightView = new Text(first.getName() + "->" + second.getName() + ":" + Double.toString(weightFromFirstToSecond));*

*double x1, x2, y1, y2;*

*x1 = first.getCoordinates().getX();*

*x2 = second.getCoordinates().getX();*

*y1 = first.getCoordinates().getY();*

*y2 = second.getCoordinates().getY();*

*if (x1 > x2) {*

*x1 -= radiusOfVertexCircle;*

*x2 += radiusOfVertexCircle;*

*} else {*

*x2 -= radiusOfVertexCircle;*

*x1 += radiusOfVertexCircle;*

*}*

*edgeLine.setStartX(x1);*

*edgeLine.setEndX(x2);*

*edgeLine.setStartY(y1);*

*edgeLine.setEndY(y2);*

*if (x1 > x2) {*

*weightView.setX(x1 - Math.abs(x2 - x1) / 2);*

*} else {*

*weightView.setX(x2 - Math.abs(x2 - x1) / 2);*

*}*

*weightView.setY((y1 + y2) / 2 - 5);*

*edgeLine.setStroke(Paint.valueOf("#806b8d"));*

*}*

*public void setReversedDirection(double weight){*

*weightFromSecondToFirst = weight;*

*directionFromSecondToFirst = true;*

*weightView.setText(second.getName() + "->" + first.getName() + ":" + Double.toString(weightFromSecondToFirst) +*

*"\n" + first.getName() + "->" + second.getName() + ":" + Double.toString(weightFromFirstToSecond));*

*}*

*public void setReversedDirection(Vertex start, Vertex finish, double weight){*

*if (start == first && finish == second) {*

*return;*

*}*

*weightFromSecondToFirst = weight;*

*directionFromSecondToFirst = true;*

*weightView.setText(second.getName() + "->" + first.getName() + ":" + Double.toString(weightFromSecondToFirst) +*

*"\n" + first.getName() + "->" + second.getName() + ":" + Double.toString(weightFromFirstToSecond));*

*}*

*public void deleteOneWay(Vertex first, Vertex second){*

*if (first == this.first & second == this.second) {*

*directionFromFirstToSecond = false;*

*weightFromFirstToSecond = 0;*

*weightView.setText(second.getName() + "->" + first.getName() + ":" + Double.toString(weightFromSecondToFirst));*

*} else if(first == this.second & second == this.first) {*

*directionFromSecondToFirst = false;*

*weightFromSecondToFirst = 0;*

*weightView.setText(second.getName() + "->" + first.getName() + ":" + Double.toString(weightFromFirstToSecond));*

*}*

*}*

*public boolean isFromFirstToSecond(Vertex first, Vertex second) {*

*return this.first == first && this.second == second && directionFromFirstToSecond;*

*}*

*public boolean isFromSecondToFirst(Vertex first, Vertex second) {*

*return this.second == first && this.first == second && directionFromSecondToFirst;*

*}*

*public boolean isTwoDirectional() {*

*return directionFromFirstToSecond && directionFromSecondToFirst;*

*}*

*public boolean isOnlyOneWay() {*

*return directionFromFirstToSecond ^ directionFromSecondToFirst;*

*}*

*public boolean isReverse(Vertex first, Vertex second) {*

*return (cmpFirst(second) && cmpSecond(first));*

*}*

*public boolean cmpFirst(Vertex other) {*

*return first == other;*

*}*

*public boolean cmpSecond(Vertex other) {*

*return second == other;*

*}*

*public boolean cmpBoth(Vertex first, Vertex second) {*

*return cmpFirst(first) && cmpSecond(second) || isReverse(first, second);*

*}*

*public Text getName(){*

*return weightView;*

*}*

*public Line getView() { return edgeLine; }*

*public void moveLine() {*

*double x1, x2, y1, y2;*

*x1 = first.getCoordinates().getX();*

*x2 = second.getCoordinates().getX();*

*y1 = first.getCoordinates().getY();*

*y2 = second.getCoordinates().getY();*

*if (x1 > x2) {*

*x1 -= radiusOfVertexCircle;*

*x2 += radiusOfVertexCircle;*

*} else {*

*x2 -= radiusOfVertexCircle;*

*x1 += radiusOfVertexCircle;*

*}*

*edgeLine.setStartX(x1);*

*edgeLine.setEndX(x2);*

*edgeLine.setStartY(y1);*

*edgeLine.setEndY(y2);*

*if (x1 > x2) {*

*weightView.setX(x1 - Math.abs(x2 - x1) / 2);*

*} else {*

*weightView.setX(x2 - Math.abs(x2 - x1) / 2);*

*}*

*weightView.setY((y1 + y2) / 2 - 5);*

*}*

*public void renewNames() {*

*if (isTwoDirectional()) {*

*weightView.setText(second.getName() + "->" + first.getName() + ":" + Double.toString(weightFromSecondToFirst) +*

*"\n" + first.getName() + "->" + second.getName() + ":" + Double.toString(weightFromFirstToSecond));*

*} else if (directionFromFirstToSecond) {*

*weightView.setText(first.getName() + "->" + second.getName() + ":" + Double.toString(weightFromFirstToSecond));*

*} else {*

*weightView.setText(second.getName() + "->" + first.getName() + ":" + Double.toString(weightFromSecondToFirst));*

*}*

*}*

*}*

Название файла: *AStar.java*

*package application.app;*

*import java.util.ArrayList;*

*import java.util.Arrays;*

*import java.util.HashMap;*

*import java.util.List;*

Название файла: *Graph.java*

*package application.app;*

*import java.util.ArrayList;*

*import java.util.HashMap;*

*public class Graph {*

*private ArrayList<Vertex> vertexes;*

*private HashMap<Character, Boolean> availableNames;*

*public ArrayList<Vertex> getVertexes() {*

*return vertexes;*

*}*

*public Graph(){*

*vertexes = new ArrayList<Vertex>();*

*availableNames = new HashMap<Character, Boolean>();*

*for(char letter = 'a'; letter <= 'z'; letter++){*

*availableNames.put(letter, true);*

*}*

*}*

*public Character getAvailableName() {*

*for(char letter = 'a'; letter <= 'z'; letter++){*

*if (availableNames.get(letter).equals(true)) {*

*return letter;*

*}*

*}*

*return '\*';*

*}*

*public void deleteAvailableName(char name){*

*availableNames.put(name, false);*

*}*

*public void addAvailableName(char name){*

*availableNames.put(name, true);*

*}*

*public void addVertex(char vertex, double x, double y){*

*vertexes.add(new Vertex(vertex, x, y));*

*availableNames.put(vertex, false);*

*}*

*public void addVertex(Vertex vertex){*

*vertexes.add(vertex);*

*availableNames.put(vertex.getName(), false);*

*}*

*public void addEdge(Vertex first, Vertex second, double weight){*

*first.addNeighbour(second, weight);*

*}*

*public void deleteVertex(char vertex){*

*for(Vertex v : vertexes){*

*if (v.getName() == vertex){*

*for(Vertex subV : vertexes){*

*subV.getNeighbours().remove(v);*

*}*

*availableNames.put(vertex, true);*

*vertexes.remove(v);*

*return;*

*}*

*}*

*}*

*public void deleteVertex(Vertex vertex){*

*for(Vertex subV : vertexes){*

*subV.getNeighbours().remove(vertex);*

*}*

*availableNames.put(vertex.getName(), true);*

*vertexes.remove(vertex);*

*}*

*public void deleteEdge(char first, char second){*

*Vertex firstVertex = null, secondVertex = null;*

*for(Vertex v : vertexes){*

*if (v.getName() == first){*

*firstVertex = v;*

*}*

*if (v.getName() == second){*

*secondVertex = v;*

*}*

*}*

*if (firstVertex != null && secondVertex != null){*

*firstVertex.deleteNeighbour(secondVertex);*

*}*

*}*

*public void deleteEdge(Vertex first, Vertex second){*

*if (first != null && second != null){*

*first.deleteNeighbour(second);*

*}*

*}*

*public Vertex findVertex(char name) {*

*for (Vertex v : vertexes) {*

*if (v.getName() == name) {*

*return v;*

*}*

*}*

*return null;*

*}*

*public boolean isNameAvailable(char name){*

*return availableNames.get(name);*

*}*

*public boolean isEdgeAlreadyExists(Vertex first, Vertex second){*

*return first.getNeighbours().containsKey(second);*

*}*

*public void clearGraph(){*

*vertexes.clear();*

*}*

*}*

Название файла: *GraphController.java*

*package application.app;*

*import javafx.event.EventHandler;*

*import javafx.scene.input.MouseButton;*

*import javafx.scene.input.MouseEvent;*

*import javafx.scene.layout.Pane;*

*import javafx.scene.paint.Paint;*

*import javafx.scene.text.Text;*

*// files*

*import java.io.BufferedReader;*

*import java.io.IOException;*

*import java.nio.charset.StandardCharsets;*

*import java.nio.file.Files;*

*import java.nio.file.Path;*

*import java.nio.file.Paths;*

*import java.util.ArrayList;*

*import java.util.HashMap;*

*public class GraphController { //для считывания графа*

*private ArrayList<VertexDrawable> vertexesDrawable;*

*private Graph graph;*

*private ArrayList<EdgeDrawable> edgesDrawable;*

*private AStar aStar;*

*private Vertex chosen1, chosen2;*

*public GraphController(){*

*vertexesDrawable = new ArrayList<VertexDrawable>();*

*graph = new Graph();*

*edgesDrawable = new ArrayList<EdgeDrawable>();*

*eventFlag = true;*

*}*

*public ArrayList<VertexDrawable> getVertexesDrawable(){*

*return vertexesDrawable;*

*}*

*public Graph getGraph(){*

*return graph;*

*}*

*private VertexDrawable findVertex(Vertex vertex) {*

*for(VertexDrawable v: vertexesDrawable) {*

*if (v.getVertex() == vertex) {*

*return v;*

*}*

*}*

*return null;*

*}*

*private EdgeDrawable findEdge(Vertex vertex) {*

*for(EdgeDrawable e: edgesDrawable) {*

*if(e.cmpFirst(vertex) || e.cmpSecond(vertex)){*

*return e;*

*}*

*}*

*return null;*

*}*

*private EdgeDrawable findEdge(Vertex start, Vertex finish) {*

*for(EdgeDrawable e: edgesDrawable){*

*if(e.cmpBoth(start, finish)){*

*return e;*

*}*

*}*

*return null;*

*}*

*public void normalGraphColor() {*

*for (VertexDrawable v : vertexesDrawable){*

*v.getView().setFill(Paint.valueOf("#d5b8e6"));*

*}*

*for (EdgeDrawable e : edgesDrawable) {*

*e.getView().setStroke(Paint.valueOf("#806b8d"));*

*}*

*}*

*public void runningAlgorithmAStar(Vertex first, Vertex second) {*

*normalGraphColor();*

*ArrayList<Vertex> solution;*

*aStar = new AStar();*

*if (chosen1 != null && chosen2 != null) {*

*solution = aStar.a\_star\_public(chosen1, chosen2);*

*VertexDrawable vertex1 = findVertex(chosen1);*

*VertexDrawable vertex2 = findVertex(chosen2);*

*if(vertex1 != null){*

*vertex1.getView().setEffect(null);*

*}*

*if (vertex2 != null) {*

*vertex2.getView().setEffect(null);*

*}*

*chosen1 = null;*

*chosen2 = null;*

*} else {*

*return;*

*}*

*for (int i = 0; i < solution.size() - 1; i++) {*

*VertexDrawable cur = findVertex(solution.get(i));*

*VertexDrawable next = findVertex(solution.get(i + 1));*

*if (cur != null && next != null) {*

*cur.getView().setFill(Paint.valueOf("#0cda73"));*

*next.getView().setFill(Paint.valueOf("#0cda73"));*

*}*

*EdgeDrawable edge = findEdge(solution.get(i), solution.get(i + 1));*

*if (edge != null){*

*edge.getView().setStroke(Paint.valueOf("#0a9a50"));*

*}*

*}*

*}*

*private int stepNumber;*

*private boolean eventFlag;*

*private ArrayList<Text> heuristics;*

*public void startStepByStep() {*

*eventFlag = false;*

*aStar = new AStar();*

*if (chosen1 != null && chosen2 != null) {*

*aStar.a\_star\_public(chosen1, chosen2);*

*chosen1 = null;*

*chosen2 = null;*

*stepNumber = 0;*

*ArrayList<Vertex> result = aStar.getPaths().get(stepNumber);*

*HashMap<Vertex, Double> fResults = aStar.getF\_steps().get(stepNumber);*

*VertexDrawable cur = findVertex(result.get(0));*

*normalGraphColor();*

*if (cur != null) {*

*heuristics = new ArrayList<>();*

*heuristics.add(new Text(cur.getVertex().getCoordinates().getX(),*

*cur.getVertex().getCoordinates().getY() - 30,*

*"f:" + Math.round(fResults.get(result.get(0)))));*

*cur.getView().setFill(Paint.valueOf("#0cda73"));*

*}*

*}*

*}*

*public void drawGraphAndLabels(Pane pane) {*

*drawGraph(pane);*

*if (heuristics != null) {*

*for (Text text: heuristics) {*

*pane.getChildren().add(text);*

*}*

*}*

*}*

*public void toEndStepByStep() {*

*int stepSize = aStar.getPaths().size() - 1 - stepNumber;*

*doStep(stepSize);*

*}*

*public void allowEvents() {*

*eventFlag = true;*

*}*

*public void endSBS() {*

*heuristics = null;*

*}*

*public void doStep(int step) {*

*if (stepNumber == 0 && step <= -1 || stepNumber == aStar.getPaths().size() - 1 && step >= 1) {*

*return;*

*} else {*

*stepNumber += step;*

*}*

*if (stepNumber >= 0 && stepNumber < aStar.getPaths().size()) {*

*ArrayList<Vertex> result = aStar.getPaths().get(stepNumber);*

*HashMap<Vertex, Double> fResults = aStar.getF\_steps().get(stepNumber);*

*heuristics = new ArrayList<>();*

*normalGraphColor();*

*if (stepNumber == 0) {*

*VertexDrawable cur = findVertex(result.get(0));*

*if (cur != null) {*

*cur.getView().setFill(Paint.valueOf("#0cda73"));*

*heuristics.add(new Text(cur.getVertex().getCoordinates().getX(),*

*cur.getVertex().getCoordinates().getY() - 30,*

*"f:" + Math.round(fResults.get(result.get(0)))));*

*}*

*return;*

*}*

*for (int i = 0; i < result.size() - 1; i++) {*

*VertexDrawable cur = findVertex(result.get(i));*

*VertexDrawable next = findVertex(result.get(i + 1));*

*if (cur != null && next != null) {*

*cur.getView().setFill(Paint.valueOf("#0cda73"));*

*next.getView().setFill(Paint.valueOf("#0cda73"));*

*heuristics.add(new Text(cur.getVertex().getCoordinates().getX(),*

*cur.getVertex().getCoordinates().getY() - 30,*

*"f:" + Math.round(fResults.get(result.get(i)))));*

*if (i == result.size() - 2) {*

*heuristics.add(new Text(next.getVertex().getCoordinates().getX(),*

*next.getVertex().getCoordinates().getY() - 30,*

*"f:" + Math.round(fResults.get(result.get(i + 1)))));*

*}*

*}*

*EdgeDrawable edge = findEdge(result.get(i), result.get(i + 1));*

*if (edge != null){*

*edge.getView().setStroke(Paint.valueOf("#0a9a50"));*

*}*

*}*

*}*

*}*

*public boolean isFinalInStepByStep() {*

*return (stepNumber == aStar.getPaths().size() - 1);*

*}*

*public String getPath(){*

*if(aStar != null){*

*return aStar.getPath();*

*}*

*return null;*

*}*

*public double getWeight(){*

*if(aStar != null){*

*return aStar.getWeight();*

*}*

*return -1;*

*}*

*public static Graph readGraphForTests(String string){*

*Graph graphTest = new Graph();*

*String[] tokens = string.split("[\n ]");*

*int numberOfVertexes = Integer.parseInt(tokens[0]);*

*for (int i = 1; i < numberOfVertexes \* 3; i += 3){*

*Vertex vertex = new Vertex(tokens[i].charAt(0), Double.parseDouble(tokens[i + 1]), Double.parseDouble(tokens[i + 2]));*

*graphTest.addVertex(vertex);*

*}*

*for (int i = numberOfVertexes \* 3 + 1; i < tokens.length; i+= 3){*

*char start, finish;*

*double weight;*

*start = tokens[i].charAt(0);*

*finish = tokens[i + 1].charAt(0);*

*weight = Double.parseDouble(tokens[i + 2]);*

*Vertex sVertex, fVertex;*

*sVertex = graphTest.findVertex(start);*

*fVertex = graphTest.findVertex(finish);*

*graphTest.addEdge(sVertex, fVertex, weight);*

*}*

*return graphTest;*

*}*

*//в readGraphFromWindow, readGraphFromFile добавить исключения, если вводимые данные пустые, плюс для*

*// файлов исключение на то, что файл не существует*

*public void readGraph(String string) throws NumberFormatException, DataFormatException, ArrayIndexOutOfBoundsException{*

*if (string.isEmpty()) CheckRules.inputIsEmpty();*

*vertexesDrawable = new ArrayList<VertexDrawable>();*

*edgesDrawable = new ArrayList<EdgeDrawable>();*

*if (graph != null) {*

*graph.clearGraph();*

*} else {*

*graph = new Graph();*

*}*

*String[] tokens = string.split("[\\r\\n ]+");*

*int numberOfVertexes = Integer.parseInt(tokens[0]);*

*CheckRules.checkNumberOfVertexes(numberOfVertexes);*

*for (int i = 1; i < numberOfVertexes \* 3; i += 3){*

*CheckRules.checkVertexName(tokens[i]);*

*CheckRules.checkIfVertexNameIsAvailable(graph, tokens[i]);*

*CheckRules.checkCoordinates(tokens[i + 1], tokens[i + 2]);*

*Vertex vertex = new Vertex(tokens[i].charAt(0), Double.parseDouble(tokens[i + 1]), Double.parseDouble(tokens[i + 2]));*

*graph.addVertex(vertex);*

*vertexesDrawable.add(new VertexDrawable(vertex));*

*}*

*for (int i = numberOfVertexes \* 3 + 1; i < tokens.length; i+= 3){*

*CheckRules.checkEdge(graph, tokens[i], tokens[i + 1], tokens[i + 2]);*

*char start = tokens[i].charAt(0);*

*char finish = tokens[i + 1].charAt(0);*

*double weight = Double.parseDouble(tokens[i + 2]);*

*Vertex startVertex = graph.findVertex(start);*

*Vertex finishVertex = graph.findVertex(finish);*

*CheckRules.isEdgeAlreadyExists(graph, startVertex, finishVertex, weight);*

*graph.addEdge(startVertex, finishVertex, weight);*

*boolean flag = false;*

*for (EdgeDrawable e: edgesDrawable) {*

*if (e.isReverse(startVertex, finishVertex)) {*

*e.setReversedDirection(weight);*

*flag = true;*

*}*

*}*

*if (!flag) {*

*edgesDrawable.add(new EdgeDrawable(startVertex, finishVertex, weight));*

*}*

*}*

*setEventHandlers();*

*}*

*public void readGraphFromFile(String fileName) throws IOException, DataFormatException, ArrayIndexOutOfBoundsException {*

*try {*

*String data = new String(Files.readAllBytes(Paths.get(fileName)));*

*readGraph(data);*

*} catch (IOException e){*

*throw new DataFormatException("Error occurred in reading this file");*

*}*

*}*

*protected void setEventHandlers() {*

*// добавлены функции реакции круга на перетаскивание мышью, меняются координаты вершины в окне,*

*// а также меняются координаты круга*

*for(VertexDrawable v: vertexesDrawable){*

*final Double[] x = new Double[1];*

*final Double[] y = new Double[1];*

*v.getView().setOnMousePressed(new EventHandler<MouseEvent>() {*

*@Override*

*public void handle(MouseEvent mouseEvent) {*

*if (eventFlag) {*

*x[0] = v.getView().getLayoutX() - mouseEvent.getSceneX();*

*y[0] = v.getView().getLayoutY() - mouseEvent.getSceneY();*

*}*

*}*

*});*

*v.getView().setOnMouseDragged(new EventHandler<MouseEvent>() {*

*@Override*

*public void handle(MouseEvent mouseEvent) {*

*if (eventFlag) {*

*double newX, newY;*

*newX = mouseEvent.getSceneX() + x[0];*

*newY = mouseEvent.getSceneY() + y[0];*

*if (newX <= 600 && newY <= 530 && newX >= 0 && newY >= 0){*

*v.getVertex().setCoordinates(newX, newY);*

*}*

*v.moveCircle();*

*for(EdgeDrawable e: edgesDrawable) {*

*e.moveLine();*

*}*

*}*

*}*

*});*

*v.getView().setOnMouseClicked(new EventHandler<MouseEvent>() {*

*@Override*

*public void handle(MouseEvent mouseEvent) {*

*if (eventFlag) {*

*if(mouseEvent.getButton() == MouseButton.PRIMARY && mouseEvent.getClickCount() == 2){*

*if (chosen1 == v.getVertex()) {*

*VertexDrawable vertex1 = findVertex(chosen1);*

*if(vertex1 != null){*

*vertex1.getView().setFill(Paint.valueOf("#d5b8e6"));*

*}*

*chosen1 = null;*

*return;*

*} else if (chosen2 == v.getVertex()) {*

*VertexDrawable vertex2 = findVertex(chosen2);*

*if(vertex2 != null){*

*vertex2.getView().setFill(Paint.valueOf("#d5b8e6"));*

*}*

*chosen2 = null;*

*return;*

*}*

*if(chosen1 == null) {*

*chosen1 = v.getVertex();*

*v.getView().setFill(Paint.valueOf("#5e64d7"));*

*} else if (chosen2 == null) {*

*v.getView().setFill(Paint.valueOf("#a34acd"));*

*chosen2 = v.getVertex();*

*} else {*

*VertexDrawable vertex1 = findVertex(chosen1);*

*VertexDrawable vertex2 = findVertex(chosen2);*

*if(vertex1 != null){*

*vertex1.getView().setFill(Paint.valueOf("#d5b8e6"));*

*}*

*if (vertex2 != null) {*

*vertex2.getView().setFill(Paint.valueOf("#d5b8e6"));*

*}*

*chosen1 = v.getVertex();*

*v.getView().setFill(Paint.valueOf("#5e64d7"));*

*chosen2 = null;*

*}*

*}*

*}*

*}*

*});*

*}*

*}*

*public void drawGraph(Pane pane){*

*pane.getChildren().clear();*

*for (VertexDrawable vertex: vertexesDrawable){*

*pane.getChildren().add(vertex.getView());*

*pane.getChildren().add(vertex.getName());*

*}*

*for (EdgeDrawable edge: edgesDrawable){*

*pane.getChildren().add(edge.getView());*

*pane.getChildren().add(edge.getName());*

*}*

*}*

*public void drawVertex(Pane pane, Vertex vertex){*

*graph.addVertex(vertex);*

*VertexDrawable drawableVertex = new VertexDrawable(vertex);*

*vertexesDrawable.add(drawableVertex);*

*drawGraph(pane);*

*setEventHandlers();*

*}*

*public void addEdge(double weight) throws DataFormatException {*

*if (chosen1 != null && chosen2 != null) {*

*if (graph.isEdgeAlreadyExists(chosen1, chosen2)){*

*throw new DataFormatException("Edge already exists");*

*}*

*graph.addEdge(chosen1, chosen2, weight);*

*EdgeDrawable edgeDrawable = findEdge(chosen1, chosen2);*

*if (edgeDrawable != null && edgeDrawable.isOnlyOneWay()) {*

*edgeDrawable.setReversedDirection(chosen1, chosen2, weight);*

*} else if (edgeDrawable == null) {*

*edgesDrawable.add(new EdgeDrawable(chosen1, chosen2, weight));*

*}*

*}*

*}*

*public void deleteEdge() {*

*if (chosen1 != null && chosen2 != null) {*

*EdgeDrawable edgeDrawable = findEdge(chosen1, chosen2);*

*if (edgeDrawable != null && edgeDrawable.isOnlyOneWay()) {*

*chosen1.deleteNeighbour(chosen2);*

*if (edgeDrawable.isFromFirstToSecond(chosen1, chosen2)) {*

*edgesDrawable.remove(edgeDrawable);*

*} else if (edgeDrawable.isFromSecondToFirst(chosen1, chosen2)) {*

*edgesDrawable.remove(edgeDrawable);*

*}*

*} else if (edgeDrawable != null && edgeDrawable.isTwoDirectional()) {*

*edgeDrawable.deleteOneWay(chosen1, chosen2);*

*chosen1.deleteNeighbour(chosen2);*

*}*

*}*

*}*

*public void deleteVertex(VertexDrawable vertexDrawable) {*

*Vertex vertex = vertexDrawable.getVertex();*

*for (Vertex v : graph.getVertexes()) {*

*EdgeDrawable edgeDrawable = findEdge(vertex, v);*

*if(edgeDrawable != null) {*

*edgesDrawable.remove(edgeDrawable);*

*vertex.deleteNeighbour(v);*

*}*

*}*

*vertexesDrawable.remove(vertexDrawable);*

*graph.deleteVertex(vertex);*

*}*

*public void renewEdgesNames() {*

*for (EdgeDrawable e : edgesDrawable) {*

*e.renewNames();*

*}*

*}*

*}*

Название файла: *Main.java*

*package application.app;*

*import javafx.application.Application;*

*import javafx.fxml.FXMLLoader;*

*import javafx.scene.Parent;*

*import javafx.scene.Scene;*

*import javafx.stage.Stage;*

*import java.net.URL;*

*public class Main extends Application {*

*@Override*

*public void start(Stage primaryStage) throws Exception {*

*Tests.RunTests();*

*URL cl;*

*Parent root;*

*if ((cl = getClass().getResource("Start window.fxml")) != null){*

*root = FXMLLoader.load(cl);*

*} else {*

*return;*

*}*

*primaryStage.setTitle("A\* algorithm");*

*primaryStage.setScene(new Scene(root, 800, 550));*

*primaryStage.show();*

*}*

*public static void main(String[] args) {*

*launch();*

*}*

*}*

Название файла: *Start.java*

*package application.app;*

*public class Start {*

*public static void main(String[] args) {*

*Main.main(args);*

*}*

*}*

Название файла: *Tests.java*

*package application.app;*

*import java.util.ArrayList;*

*import java.util.HashMap;*

*public class Tests {*

*public static final double DOUBLE\_CONST= 0.0000000001;*

*public static final int SLEEP\_CONST= 500;*

*public static void RunTests() throws InterruptedException {*

*System.out.println(FirstTestH());*

*Thread.sleep(SLEEP\_CONST);*

*System.out.println(SecondTestH());*

*Thread.sleep(SLEEP\_CONST);*

*System.out.println(ThirdTestH());*

*Thread.sleep(SLEEP\_CONST);*

*AStar atest = new AStar();*

*System.out.println(FirstTestAStar(atest));*

*Thread.sleep(SLEEP\_CONST);*

*System.out.println(FirstTestPath(atest));*

*Thread.sleep(SLEEP\_CONST);*

*System.out.println(FirstTestWeight(atest));*

*Thread.sleep(SLEEP\_CONST);*

*atest = new AStar();*

*System.out.println(SecondTestAStar(atest));*

*Thread.sleep(SLEEP\_CONST);*

*System.out.println(SecondTestPath(atest));*

*Thread.sleep(SLEEP\_CONST);*

*System.out.println(SecondTestWeight(atest));*

*Thread.sleep(SLEEP\_CONST);*

*atest = new AStar();*

*System.out.println(ThirdTestAStar(atest));*

*Thread.sleep(SLEEP\_CONST);*

*System.out.println(ThirdTestPath(atest));*

*Thread.sleep(SLEEP\_CONST);*

*System.out.println(ThirdTestWeight(atest));*

*Thread.sleep(SLEEP\_CONST);*

*atest = new AStar();*

*System.out.println(FourthTestAStar(atest));*

*Thread.sleep(SLEEP\_CONST);*

*System.out.println(FourthTestPath(atest));*

*Thread.sleep(SLEEP\_CONST);*

*System.out.println(FourthTestWeight(atest));*

*Thread.sleep(SLEEP\_CONST);*

*atest = new AStar();*

*System.out.println(FifthTestAStar(atest));*

*Thread.sleep(SLEEP\_CONST);*

*System.out.println(FifthTestPath(atest));*

*Thread.sleep(SLEEP\_CONST);*

*System.out.println(FifthTestWeight(atest));*

*Thread.sleep(SLEEP\_CONST);*

*}*

*private static String FirstTestH() {*

*if (AStar.h(new Vertex('a', 0, 0), new Vertex('b', 0, 0)) == 0.0)*

*return "First test: heuristic." + "\tAnswer is correst. Test: OK";*

*else return "First test: heuristic." + "\tAnswer is incorrest. Test failed";*

*}*

*private static String SecondTestH() {*

*if (Math.abs(AStar.h(new Vertex('a', 100.234, 2), new Vertex('b', 189.2, 0)) - 88.98847765862723) < DOUBLE\_CONST)*

*return "Second test: heuristic." + "\tAnswer is correst. Test: OK";*

*else return "Second test: heuristic." + "\tAnswer is incorrest. Test failed";*

*}*

*private static String ThirdTestH() {*

*if (Math.abs(AStar.h(new Vertex('a', 100, 300), (new Vertex('b', 100 + Math.sqrt(600), 305))) - 25) < DOUBLE\_CONST)*

*return "Third test: heuristic." + "\tAnswer is correst. Test: OK";*

*else return "Third test: heuristic." + "\tAnswer is incorrest. Test failed";*

*}*

*private static String FirstTestAStar(AStar n) {*

*Graph graph = GraphController.readGraphForTests("2\na 0 0\nb 0 0\na b 0");*

*ArrayList<Vertex> test = n.a\_star\_public(graph.findVertex('a'), graph.findVertex('b'));*

*ArrayList<Vertex> array = new ArrayList<>();*

*array.add(graph.findVertex('a'));*

*array.add(graph.findVertex('b'));*

*if (test.equals(array)) return "First test: AStar." + "\tAnswer is correst. Test: OK";*

*else return "First test: AStar." + "\tAnswer is incorrest. Test failed";*

*}*

*private static String FirstTestPath(AStar n) {*

*if (n.getPath().equals("ab")) return "First test: Path." + "\tAnswer is correst. Test: OK";*

*else return "First test: Path." + "\tAnswer is incorrest. Test failed";*

*}*

*private static String FirstTestWeight(AStar n) {*

*if (Math.abs(n.getWeight() - 0) < DOUBLE\_CONST) return "First test: Weight." + "\tAnswer is correst. Test: OK";*

*else return "First test: Weight." + "\tAnswer is incorrest. Test failed";*

*}*

*private static String SecondTestAStar(AStar n) {*

*Graph graph = GraphController.readGraphForTests("5\na 100.234 2\nb 189.2 0\nc 200 300\n" +*

*"d 100 300\ne 500 500\na b 3\nb c 1\nc d 1\na d 5\nd e 1");*

*ArrayList<Vertex> test = n.a\_star\_public(graph.findVertex('a'), graph.findVertex('e'));*

*ArrayList<Vertex> array = new ArrayList<>();*

*array.add(graph.findVertex('a'));*

*array.add(graph.findVertex('d'));*

*array.add(graph.findVertex('e'));*

*if (test.equals(array)) return "Second test: AStar." + "\tAnswer is correst. Test: OK";*

*else return "Second test: AStar." + "\tAnswer is incorrest. Test failed";*

*}*

*private static String SecondTestPath(AStar n) {*

*if (n.getPath().equals("ade")) return "Second test: Path." + "\tAnswer is correst. Test: OK";*

*else return "Second test: Path." + "\tAnswer is incorrest. Test failed";*

*}*

*private static String SecondTestWeight(AStar n) {*

*if (Math.abs(n.getWeight() - 6) < DOUBLE\_CONST) return "Second test: Weight." + "\tAnswer is correst. Test: OK";*

*else return "Second test: Weight." + "\tAnswer is incorrest. Test failed";*

*}*

*private static String ThirdTestAStar(AStar n) {*

*Graph graph = GraphController.readGraphForTests("8\na 100 300\nb 400 500\nc 350 250\n" +*

*"d 80 90\ne 330 220\nx 105 375\ny 296 100\nz 500 150\na b 3\na c 5\na d 7\na e 1\nx z 1\ny z 4");*

*ArrayList<Vertex> test = n.a\_star\_public(graph.findVertex('a'), graph.findVertex('d'));*

*ArrayList<Vertex> array = new ArrayList<>();*

*array.add(graph.findVertex('a'));*

*array.add(graph.findVertex('d'));*

*if (test.equals(array)) return "Third test: AStar." + "\tAnswer is correst. Test: OK";*

*else return "Third test: AStar." + "\tAnswer is incorrest. Test failed";*

*}*

*private static String ThirdTestPath(AStar n) {*

*if (n.getPath().equals("ad")) return "Third test: Path." + "\tAnswer is correst. Test: OK";*

*else return "Third test: Path." + "\tAnswer is incorrest. Test failed";*

*}*

*private static String ThirdTestWeight(AStar n) {*

*if (Math.abs(n.getWeight() - 7) < DOUBLE\_CONST) return "Third test: Weight." + "\tAnswer is correst. Test: OK";*

*else return "Third test: Weight." + "\tAnswer is incorrest. Test failed";*

*}*

*private static String FourthTestAStar(AStar n) {*

*Graph graph = GraphController.readGraphForTests("6\na 1 1\nb 1 2\nc 2 3\n" +*

*"d 3 4\ne 4 5\nf 5 6\na b 1\nb c 2\nc d 3\nd e 4\ne f 5\na f 34");*

*ArrayList<Vertex> test = n.a\_star\_public(graph.findVertex('a'), graph.findVertex('f'));*

*ArrayList<Vertex> array = new ArrayList<>();*

*array.add(graph.findVertex('a'));*

*array.add(graph.findVertex('b'));*

*array.add(graph.findVertex('c'));*

*array.add(graph.findVertex('d'));*

*array.add(graph.findVertex('e'));*

*array.add(graph.findVertex('f'));*

*if (test.equals(array)) return "Fourth test: AStar." + "\tAnswer is correst. Test: OK";*

*else return "Fourth test: AStar." + "\tAnswer is incorrest. Test failed";*

*}*

*private static String FourthTestPath(AStar n) {*

*if (n.getPath().equals("abcdef")) return "Fourth test: Path." + "\tAnswer is correst. Test: OK";*

*else return "Fourth test: Path." + "\tAnswer is incorrest. Test failed";*

*}*

*private static String FourthTestWeight(AStar n) {*

*if (Math.abs(n.getWeight() - 15) < DOUBLE\_CONST) return "Fourth test: Weight." + "\tAnswer is correst. Test: OK";*

*else return "Fourth test: Weight." + "\tAnswer is incorrest. Test failed";*

*}*

*private static String FifthTestAStar(AStar n) {*

*Graph graph = GraphController.readGraphForTests("4\na 300 300\nb 200 200\nd 400 200\ne 300 100" +*

*"\na b 4\na d 5\nb e 1\nd e 0.5");*

*ArrayList<Vertex> test = n.a\_star\_public(graph.findVertex('a'), graph.findVertex('e'));*

*ArrayList<Vertex> array = new ArrayList<>();*

*array.add(graph.findVertex('a'));*

*array.add(graph.findVertex('b'));*

*array.add(graph.findVertex('e'));*

*if (test.equals(array)) return "Fifth test: AStar." + "\tAnswer is correst. Test: OK";*

*else return "Fifth test: AStar." + "\tAnswer is incorrest. Test failed";*

*}*

*private static String FifthTestPath(AStar n) {*

*if (n.getPath().equals("abe")) return "Fifth test: Path." + "\tAnswer is correst. Test: OK";*

*else return "Fifth test: Path." + "\tAnswer is incorrest. Test failed";*

*}*

*private static String FifthTestWeight(AStar n) {*

*if (Math.abs(n.getWeight() - 5) < DOUBLE\_CONST) return "Fifth test: Weight." + "\tAnswer is correst. Test: OK";*

*else return "Fifth test: Weight." + "\tAnswer is incorrest. Test failed";*

*}*

*}*

Название файла: *Vertex.java*

*package application.app;*

*import java.util.HashMap;*

*public class Vertex {*

*private char name;*

*private Coordinates<Double> coordinates;*

*private HashMap<Vertex, Double> neighbours;*

*public Vertex(char name, double x, double y) {*

*this.name = name;*

*coordinates = new Coordinates<>(x, y);*

*neighbours = new HashMap<Vertex, Double>();*

*}*

*public void addNeighbour(Vertex vertex, double distance){*

*neighbours.put(vertex, distance);*

*}*

*public void deleteNeighbour(Vertex vertex){*

*neighbours.remove(vertex);*

*}*

*public char getName(){*

*return name;*

*}*

*public void setName(char name){*

*this.name = name;*

*}*

*public HashMap<Vertex, Double> getNeighbours(){*

*return neighbours;*

*}*

*public Coordinates<Double> getCoordinates(){*

*return coordinates;*

*}*

*public void setCoordinates(Double x, Double y) {*

*this.coordinates.setX(x);*

*this.coordinates.setY(y);*

*}*

*}*

Название файла: *VertexDrawable.java*

*package application.app;*

*import javafx.event.EventHandler;*

*import javafx.scene.Cursor;*

*import javafx.scene.input.MouseEvent;*

*import javafx.scene.paint.Paint;*

*import javafx.scene.shape.Circle;*

*//import project.application.model.\*;*

*import javafx.scene.text.Text;*

*import java.util.Arrays;*

*import java.util.HashMap;*

*public class VertexDrawable {*

*private char name;*

*private Circle vertexCircle;*

*private Text nameCircle;*

*private Vertex vertex;*

*private Coordinates coordinates;*

*private final static double radiusOfVertexCircle = 18.0;*

*public VertexDrawable(Vertex vertex){*

*this.vertex = vertex;*

*name = vertex.getName();*

*coordinates = vertex.getCoordinates();*

*vertexCircle = new Circle();*

*vertexCircle.setLayoutX((Double) coordinates.getX());*

*vertexCircle.setLayoutY((Double) coordinates.getY());*

*vertexCircle.setRadius(radiusOfVertexCircle);*

*vertexCircle.setFill(Paint.valueOf("#d5b8e6"));*

*nameCircle = new Text(String.valueOf(name));*

*nameCircle.setX((Double) coordinates.getX());*

*nameCircle.setY((Double) coordinates.getY());*

*vertexCircle.setOnMouseEntered(new EventHandler<MouseEvent>() {*

*@Override*

*public void handle(MouseEvent mouseEvent) {*

*vertexCircle.setCursor(Cursor.HAND);*

*}*

*});*

*}*

*public Circle getView(){*

*return vertexCircle;*

*}*

*public Text getName(){*

*return nameCircle;*

*}*

*public void setName(String name){*

*nameCircle.setText(name);*

*vertex.setName(name.charAt(0));*

*}*

*public Vertex getVertex() {*

*return vertex;*

*}*

*public void moveCircle() {*

*coordinates = vertex.getCoordinates();*

*vertexCircle.setLayoutX((Double) coordinates.getX());*

*vertexCircle.setLayoutY((Double) coordinates.getY());*

*nameCircle.setX((Double) coordinates.getX());*

*nameCircle.setY((Double) coordinates.getY());*

*}*

*}*

Название файла: *Graph window.fxml*

*<?xml version="1.0" encoding="UTF-8"?>*

*<?import javafx.scene.Cursor?>*

*<?import javafx.scene.control.Button?>*

*<?import javafx.scene.control.ContextMenu?>*

*<?import javafx.scene.control.MenuItem?>*

*<?import javafx.scene.control.Tab?>*

*<?import javafx.scene.control.TabPane?>*

*<?import javafx.scene.control.TextArea?>*

*<?import javafx.scene.layout.AnchorPane?>*

*<?import javafx.scene.layout.Pane?>*

*<?import javafx.scene.shape.Line?>*

*<?import javafx.scene.text.Font?>*

*<?import javafx.scene.text.Text?>*

*<AnchorPane prefHeight="550.0" prefWidth="800.0" xmlns="http://javafx.com/javafx/18" xmlns:fx="http://javafx.com/fxml/1" fx:controller="application.app.Controller">*

*<children>*

*<Pane fx:id="pane" layoutX="176.0" layoutY="-2.0" onMouseClicked="#mouseClick" onMouseMoved="#mouseMoved" prefHeight="552.0" prefWidth="622.0" style="-fx-background-color: #ebebeb;" />*

*<TabPane layoutY="-1.0" prefHeight="550.0" prefWidth="175.0" style="-fx-background-color: #E6E6FA;" stylesheets="@style.css" tabClosingPolicy="UNAVAILABLE">*

*<tabs>*

*<Tab style="-fx-background-color: white;" text="Input">*

*<content>*

*<AnchorPane minHeight="0.0" minWidth="0.0" prefHeight="522.0" prefWidth="168.0" style="-fx-background-color: white; -fx-border-color: white;">*

*<children>*

*<TextArea fx:id="window" layoutX="2.0" layoutY="40.0" prefHeight="148.0" prefWidth="169.0" promptText="Enter the graph here" style="-fx-border-color: #d5b8e6;">*

*<contextMenu>*

*<ContextMenu>*

*<items>*

*<MenuItem mnemonicParsing="false" text="Unspecified Action" />*

*</items>*

*</ContextMenu>*

*</contextMenu>*

*</TextArea>*

*<Text fill="#55147e" layoutX="4.0" layoutY="36.0" strokeType="OUTSIDE" strokeWidth="0.0">*

*<font>*

*<Font name="Corbel" size="16.0" />*

*</font>*

*</Text>*

*<Button fx:id="btnReadFromFile" layoutX="4.0" layoutY="220.0" mnemonicParsing="false" onAction="#readFromFile" onMouseExited="#mouseMenuRelease" onMouseMoved="#mouseMenuMove" onMousePressed="#mouseMenuPressed" onMouseReleased="#mouseMenuRelease" prefHeight="32.0" prefWidth="168.0" style="-fx-border-radius: 2; -fx-border-color: white; -fx-background-color: white;" text="Read from file" textFill="#55147e">*

*<cursor>*

*<Cursor fx:constant="TEXT" />*

*</cursor>*

*<font>*

*<Font name="Corbel" size="16.0" />*

*</font>*

*</Button>*

*<Button fx:id="btnSBS" layoutX="4.0" layoutY="284.0" mnemonicParsing="false" onAction="#stepByStepSolution" onMouseExited="#mouseMenuRelease" onMouseMoved="#mouseMenuMove" onMousePressed="#mouseMenuPressed" onMouseReleased="#mouseMenuRelease" prefHeight="32.0" prefWidth="168.0" style="-fx-border-radius: 2; -fx-border-color: white; -fx-background-color: white;" text="Step-by-step solution" textFill="#55147e">*

*<cursor>*

*<Cursor fx:constant="TEXT" />*

*</cursor>*

*<font>*

*<Font name="Corbel" size="16.0" />*

*</font>*

*</Button>*

*<Button fx:id="btnA" layoutX="2.0" layoutY="188.0" mnemonicParsing="false" onAction="#runAStar" onMouseExited="#mouseMenuRelease" onMouseMoved="#mouseMenuMove" onMousePressed="#mouseMenuPressed" onMouseReleased="#mouseMenuRelease" prefHeight="32.0" prefWidth="168.0" style="-fx-border-radius: 2; -fx-border-color: white; -fx-background-color: white;" text="Start A\*" textFill="#55147e">*

*<cursor>*

*<Cursor fx:constant="TEXT" />*

*</cursor>*

*<font>*

*<Font name="Corbel" size="16.0" />*

*</font>*

*</Button>*

*<Button fx:id="btnReadFromWindow" layoutX="4.0" layoutY="252.0" mnemonicParsing="false" onAction="#readFromWindow" onMouseExited="#mouseMenuRelease" onMouseMoved="#mouseMenuMove" onMousePressed="#mouseMenuPressed" onMouseReleased="#mouseMenuRelease" prefHeight="32.0" prefWidth="168.0" style="-fx-border-radius: 2; -fx-border-color: white; -fx-background-color: white;" text="Read from window" textFill="#55147e">*

*<cursor>*

*<Cursor fx:constant="TEXT" />*

*</cursor>*

*<font>*

*<Font name="Corbel" size="16.0" />*

*</font>*

*</Button>*

*<Button fx:id="btnStepBack" layoutX="14.0" layoutY="476.0" mnemonicParsing="false" onAction="#stepBack" onMouseExited="#mouseMenuRelease" onMouseMoved="#mouseMenuMove" onMousePressed="#mouseMenuPressed" onMouseReleased="#mouseMenuRelease" prefHeight="32.0" prefWidth="29.0" style="-fx-border-radius: 2; -fx-border-color: white; -fx-background-color: white;" text="&lt;" textFill="#55147e" visible="false">*

*<cursor>*

*<Cursor fx:constant="TEXT" />*

*</cursor>*

*<font>*

*<Font name="Corbel" size="16.0" />*

*</font>*

*</Button>*

*<Button fx:id="btnStepForward" layoutX="56.0" layoutY="476.0" mnemonicParsing="false" onAction="#stepForward" onMouseExited="#mouseMenuRelease" onMouseMoved="#mouseMenuMove" onMousePressed="#mouseMenuPressed" onMouseReleased="#mouseMenuRelease" prefHeight="32.0" prefWidth="29.0" style="-fx-border-radius: 2; -fx-border-color: white; -fx-background-color: white;" text="&gt;" textFill="#55147e" visible="false">*

*<cursor>*

*<Cursor fx:constant="TEXT" />*

*</cursor>*

*<font>*

*<Font name="Corbel" size="16.0" />*

*</font>*

*</Button>*

*<Button fx:id="btnToEnd" layoutX="94.0" layoutY="476.0" mnemonicParsing="false" onAction="#toEnd" onMouseExited="#mouseMenuRelease" onMouseMoved="#mouseMenuMove" onMousePressed="#mouseMenuPressed" onMouseReleased="#mouseMenuRelease" prefHeight="18.0" prefWidth="69.0" style="-fx-border-radius: 2; -fx-border-color: white; -fx-background-color: white;" text="to end" textFill="#55147e" visible="false">*

*<cursor>*

*<Cursor fx:constant="TEXT" />*

*</cursor>*

*<font>*

*<Font name="Corbel" size="16.0" />*

*</font>*

*</Button>*

*<Button fx:id="endStepByStepButton" layoutX="8.0" layoutY="435.0" mnemonicParsing="false" onAction="#endStepByStep" onMouseExited="#mouseMenuRelease" onMouseMoved="#mouseMenuMove" onMousePressed="#mouseMenuPressed" onMouseReleased="#mouseMenuRelease" prefHeight="32.0" prefWidth="163.0" style="-fx-border-radius: 2; -fx-border-color: white; -fx-background-color: white;" text="End Step-by-step A\*" textFill="#55147e" visible="false">*

*<font>*

*<Font name="Corbel" size="16.0" />*

*</font></Button>*

*<TextArea fx:id="resultWindow" editable="false" layoutX="3.0" layoutY="328.0" prefHeight="84.0" prefWidth="169.0" promptText="Result" style="-fx-border-color: #d5b8e6;" />*

*</children>*

*</AnchorPane>*

*</content>*

*</Tab>*

*<Tab style="-fx-background-color: white;" text="Tools">*

*<content>*

*<AnchorPane minHeight="0.0" minWidth="0.0" prefHeight="490.0" prefWidth="154.0" style="-fx-background-color: white;">*

*<children>*

*<Button fx:id="btnAddEdge" layoutX="4.0" layoutY="44.0" mnemonicParsing="false" onAction="#addEdgeClicked" onMouseExited="#mouseMenuRelease" onMouseMoved="#mouseMenuMove" onMousePressed="#mouseMenuPressed" onMouseReleased="#mouseMenuRelease" prefHeight="32.0" prefWidth="168.0" style="-fx-border-radius: 2; -fx-background-color: white;" text="Add edge" textFill="#55147e">*

*<cursor>*

*<Cursor fx:constant="TEXT" />*

*</cursor>*

*<font>*

*<Font name="Corbel" size="16.0" />*

*</font>*

*</Button>*

*<Button fx:id="btnDeleteEdge" layoutX="4.0" layoutY="76.0" mnemonicParsing="false" onAction="#deleteEdgeClicked" onMouseExited="#mouseMenuRelease" onMouseMoved="#mouseMenuMove" onMousePressed="#mouseMenuPressed" onMouseReleased="#mouseMenuRelease" prefHeight="32.0" prefWidth="168.0" style="-fx-border-radius: 2; -fx-background-color: white;" text="Delete edge" textFill="#55147e">*

*<cursor>*

*<Cursor fx:constant="TEXT" />*

*</cursor>*

*<font>*

*<Font name="Corbel" size="16.0" />*

*</font>*

*</Button>*

*<Button fx:id="btnClear" layoutX="4.0" layoutY="108.0" mnemonicParsing="false" onAction="#clearGraph" onMouseExited="#mouseMenuRelease" onMouseMoved="#mouseMenuMove" onMousePressed="#mouseMenuPressed" onMouseReleased="#mouseMenuRelease" prefHeight="32.0" prefWidth="168.0" style="-fx-border-radius: 2; -fx-background-color: white;" text="Clear" textFill="#55147e">*

*<cursor>*

*<Cursor fx:constant="TEXT" />*

*</cursor>*

*<font>*

*<Font name="Corbel" size="16.0" />*

*</font>*

*</Button>*

*</children>*

*</AnchorPane>*

*</content>*

*</Tab>*

*<Tab style="-fx-background-color: white;" text="Help">*

*<content>*

*<AnchorPane minHeight="0.0" minWidth="0.0" prefHeight="180.0" prefWidth="200.0" style="-fx-background-color: white;">*

*<children>*

*<TextArea editable="false" layoutX="-3.0" layoutY="-2.0" prefHeight="525.0" prefWidth="182.0" text="Correct input:&#10;n - number of vertexes&#10;1 ≤ n ≤ 30&#10;Next n vertexes &#10;and its coordinates&#10;Then the weight of&#10; the edge between &#10;the vertexes in the format:&#10;vertex1, vertex2, weight&#10;&#10;Double right-click button:&#10;add new vertex&#10;with default name&#10;&#10;Right-click button on vertex&#10;open a context menu&#10;where you can rename&#10;or delete vertex &#10;&#10;You can choose two vertexes&#10;by double left-click button&#10;The first selected vertex&#10;is the source&#10;the second is the destination &#10;&#10;By selectiong two vertexes&#10;you can:&#10;run A\* algorithm&#10;delete edge&#10;add edge&#10;&#10;Example of input:&#10;3&#10;a 150 150&#10;b 550 250&#10;c 250 350&#10;b a 5&#10;a c 6&#10;c b 10&#10;b c 3&#10;&#10;If you select &quot;a&quot; and &quot;b&quot;&#10;the correct output will be:&#10;Path: acb&#10;Weight: 16.0">*

*<font>*

*<Font name="Corbel" size="16.0" />*

*</font>*

*</TextArea>*

*</children></AnchorPane>*

*</content>*

*</Tab>*

*</tabs>*

*<cursor>*

*<Cursor fx:constant="TEXT" />*

*</cursor>*

*</TabPane>*

*<Line endX="-78.0" endY="369.39996337890625" layoutX="254.0" layoutY="181.0" startX="-78.0" startY="-181.0" stroke="#d5b8e6" strokeWidth="2.0" />*

*</children>*

*</AnchorPane>*

Название файла: *Start window.fxml*

*<?xml version="1.0" encoding="UTF-8"?>*

*<?import javafx.scene.control.Button?>*

*<?import javafx.scene.image.Image?>*

*<?import javafx.scene.image.ImageView?>*

*<?import javafx.scene.layout.AnchorPane?>*

*<?import javafx.scene.layout.Pane?>*

*<?import javafx.scene.text.Font?>*

*<?import javafx.scene.text.Text?>*

*<AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="550.0" prefWidth="800.0" xmlns="http://javafx.com/javafx/18" xmlns:fx="http://javafx.com/fxml/1" fx:controller="application.app.Controller">*

*<children>*

*<ImageView fitHeight="550.0" fitWidth="800.0">*

*<image>*

*<Image url="@photo1.jpg" />*

*</image>*

*</ImageView>*

*<Pane prefHeight="550.0" prefWidth="266.0" style="-fx-background-color: #E6E6FA;">*

*<children>*

*<Text fill="#bb75e4" layoutX="15.0" layoutY="95.0" stroke="#7012a6" strokeLineCap="ROUND" strokeLineJoin="ROUND" strokeType="OUTSIDE" text="A\* algorithm">*

*<font>*

*<Font name="Corbel Light" size="45.0" />*

*</font>*

*</Text>*

*<Button fx:id="btnStart" layoutX="58.0" layoutY="226.0" mnemonicParsing="false" onAction="#switchToGraphWindow" onMousePressed="#mousePressed" onMouseReleased="#mouseRelease" prefHeight="63.0" prefWidth="151.0" style="-fx-background-color: linear-gradient(to bottom left, #52fff6, #9a35d4); -fx-border-color: linear-gradient(to bottom left, #52fff6, #9a35d4); -fx-border-radius: 2;" text="Start" textFill="WHITE">*

*<font>*

*<Font name="Corbel" size="35.0" />*

*</font>*

*</Button>*

*</children>*

*</Pane>*

*</children>*

*</AnchorPane>*

Название файла: *style.css*

.tab-pane .tab-header-area .tab-header-background {

-fx-background-color: #d5b8e6;

}