

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ
по учебной практике
Тема: Разработка визуализатора алгоритма A* на языке Java с
графическим интерфейсом.

| | | |
|--------------------|-------|----------------|
| Студент гр. 0382 | _____ | Афанасьев Н.С. |
| Студент гр. 0382 | _____ | Крючков А.М. |
| Студентка гр. 0382 | _____ | Рубежова Н.А. |
| Руководитель | _____ | Фирсов М.А. |

Санкт-Петербург
2022

ЗАДАНИЕ НА УЧЕБНУЮ ПРАКТИКУ

Студент Афанасьев Н.С. группы 0382

Студент Крючков А.М. группы 0382

Студентка Рубежова Н.А. группы 0382

Тема практики: Разработка визуализатора алгоритма A* на языке Java с графическим интерфейсом.

Задание на практику:

Командная итеративная разработка визуализатора алгоритма на Java с графическим интерфейсом.

Алгоритм: A* (нахождение кратчайшего пути в графе).

Сроки прохождения практики: 29.06.2020 – 12.07.2020

Дата сдачи отчета: 08.07.2020

Дата защиты отчета: 08.07.2020

| | | |
|--------------|-------|----------------|
| Студент | _____ | Афанасьев Н.С. |
| Студент | _____ | Крючков А.М. |
| Студентка | _____ | Рубежова Н.А. |
| Руководитель | _____ | Фирсов М.А. |

АННОТАЦИЯ

Цель практики заключается в командной итеративной разработке визуализатора алгоритма A* (A-star) с графическим интерфейсом. Работа подразумевает выполнение задач в команде для достижения поставленной цели. Результатом работы должна стать программа, которая визуализирует алгоритм A* на графе, введенном пользователем. Также целью практики является изучение нового языка программирования Java и отработка полученных знаний на практике.

SUMMARY

The purpose of the practice is the team iterative development of the A* algorithm visualizer (A-star) with a graphical interface. Work involves performing tasks in a team to achieve a goal. The result of the work should be a program that visualizes the A * algorithm on a graph entered by the user. Also, the purpose of the practice is to learn a new Java programming language and practice the acquired knowledge in practice.

СОДЕРЖАНИЕ

| | | |
|------|---|----|
| | Введение | 5 |
| 1. | Требования к программе | 6 |
| 1.1. | Исходные требования к программе | 6 |
| 2. | План разработки и распределение ролей в бригаде | 8 |
| 2.1. | План разработки | 8 |
| 2.2. | Распределение ролей в бригаде | 9 |
| 3. | Особенности реализации | 10 |
| 3.1. | Представление графа | 10 |
| 3.2. | Реализация алгоритма | 10 |
| 3.3 | Графический интерфейс | 11 |
| 3.3 | Визуализация алгоритма | 13 |
| 4. | Тестирование | 15 |
| 4.1 | Тестирование структуры представления графа | 15 |
| 4.2 | Тестирование кода алгоритма A* | 16 |
| 4.3 | Тестирование через Maven | 18 |
| | Заключение | 19 |
| | Список использованных источников | 20 |
| | Приложение А. UML-диаграмма | 21 |

ВВЕДЕНИЕ

Цель практики: итеративно разработать визуализатор алгоритма A^* на языке Java с графическим интерфейсом.

Задачи практики:

1. Изучить новый язык программирования *Java* и его основные средства.
2. Научиться итеративной разработке в команде с использованием системы контроля версий *Git*.
3. Реализовать выбранный алгоритм на языке *Java* с визуализацией и графическим интерфейсом.
4. Защитить разработанный проект.

Реализуемый алгоритм:

Алгоритм A^* . Позволяет найти кратчайший путь в ориентированном графе от начальной точки до конечной. Вместо равномерного исследования всех возможных путей он отдаёт предпочтение путям с низкой стоимостью.

В начале работы просматриваются узлы, смежные с начальным; выбирается тот из них, который имеет минимальное значение эвристики $f(x)$, после чего этот узел раскрывается.

Примечание: $f(x) = g(x) + h(x)$, где $g(x)$ — функция стоимости достижения рассматриваемой вершины x из начальной, $h(x)$ — функция эвристической оценки расстояния от рассматриваемой вершины к конечной.

На каждом этапе алгоритм оперирует с множеством путей из начальной точки до всех ещё не раскрытых (листовых) вершин графа — множеством частных решений, — которое размещается в очереди с приоритетом. Приоритет пути определяется по значению $f(x)$. Алгоритм продолжает свою работу до тех пор, пока значение $f(x)$ целевой вершины не окажется меньшим, чем любое значение в очереди, либо пока всё дерево не будет просмотрено. Из множества решений выбирается решение с наименьшей стоимостью.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1 Исходные Требования к программе

1.1.1 Общие Исходные Требования

- а. Приложение должно быть с графическим интерфейсом.

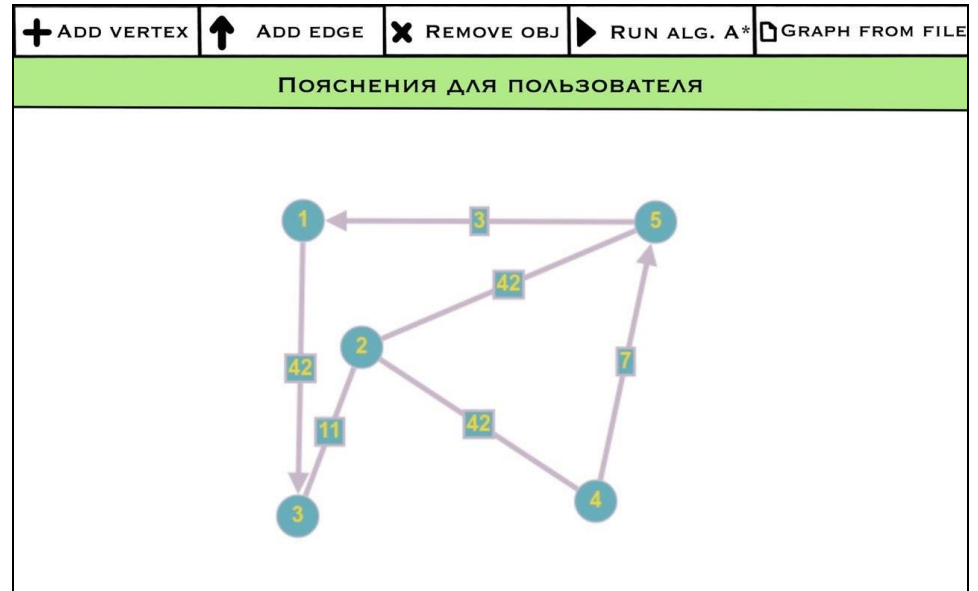


Рисунок 1 – Эскиз графического интерфейса

- б. Приложение должно быть ясным и удобным для пользователя.

1.1.2 Исходные Требования к вводу исходных данных

Пользователь должен иметь возможность задать граф двумя способами: создавать/удалять вершины и ребра щелчком мыши или с помощью текстового файла (указывая матрицу весов и координаты вершин).

1.1.2. Формат выходных данных

- а. Визуализация процесса работы алгоритма (пройденные ребра, рассмотренные вершины с их эвристиками).
- б. Графическое отображение найденного кратчайшего пути, а также вывод его длины.

1.1.3 Исходные требования к визуализации алгоритма

- a. Помимо визуализации алгоритма, должны выводиться текстовые пояснения происходящего для пользователя.
- b. Визуализация алгоритма должна быть пошаговой, шаги не должны быть крупными, полное описание пошаговой визуализации алгоритма см. в разделе 3.4.
- c. Пользователь должен иметь возможность перейти к предыдущему или следующему шагу и поставить на паузу процесс выполнения алгоритма.
- d. Соблюдать единый стиль приложения.

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

1. Изучение нового ЯП Java на платформе Stepik – **до 30 июня**;
2. Согласование спецификации и плана разработки – **30 июня**;
3. Разработка прототипа – **до 1 июля**:
 - a. Подготовка среды разработки, создание репозитория на GitHub;
 - b. Создание макета графического интерфейса без основных функций
 - c. Реализация ввода входных данных: через файл и через интеракцию с окном - и корректного их отображения в виде графа.
4. Утверждение / сдача прототипа – **1 июля**.
5. Разработка 1ой версии приложения – **до 6 июля**:
 - a. Реализация алгоритма A*
 - b. Обеспечение взаимодействия с пользователем и вывод результатов алгоритма – найденный кратчайший путь и процесс его нахождения
 - c. Обработка исключений
 - d. Создание тестов
6. Утверждение / сдача 1 версии приложения – **6 июля**
7. Разработка 2ой версии приложения – **до 8 июля**:
 - a. Добавление стилей в графический интерфейс, работа над внешним видом приложения
 - b. Доработка программы и тестов
8. Сдача / защита финальной версии – **8 июля**

2.2. Распределение ролей в бригаде

Написание отчёта – общие, основные пункты – Рубежова / UML-диаграмма – Афанасьев / в особенностях реализации – каждый описывает свою часть.

Разработка прототипа:

Крючков – Создание макета графического интерфейса без основных функций.

Афанасьев – Реализация ввода входных данных: через файл и через интеракцию с окном - и корректного их отображения в виде графа.

Рубежова – написание текста для стартового окна About, подготовка файла README.md.

Разработка 1ой версии приложения:

Рубежова - Реализация алгоритма A^* и создание тестов.

Крючков – Обеспечение взаимодействия пользователя с интерфейсом, сохранение графа в файл, добавление возможности вывода результатов непрерывно по таймеру.

Афанасьев – Вывод результатов алгоритма – найденный кратчайший путь и процесс его нахождения по шагам, а также обработка исключений, вывод сообщений об ошибках.

Разработка 2ой версии приложения:

Крючков – Добавление стилей в графический интерфейс, работа над внешним видом приложения.

Рубежова – Обеспечение запуска тестов из консоли.

Афанасьев – Исправление ошибок.

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Представление графа

В качестве основной структуры данных, используемой в программе, выступает ориентированный взвешенный граф, который описан в классе *Graph*. Граф описывается двумя полями. Первое поле – словарь вершин с порядковым номером вершины в качестве ключа и позицией вершины на плоскости в качестве значения. Второе поле – матрица весов – словарь, где ключ – это начальная вершина ребра, а значение – коллекция пар, состоящих из конечной вершины и соответствующего ребру веса. Для построения графа реализованы методы для добавления в граф или удаления ребра или вершины. Помимо этого, были реализованы геттеры как для обоих словарей в целом, так и для позиции конкретной вершины или веса конкретного ребра, а также созданы методы для проверки графа на наличие определённых рёбер или вершин.

3.2. Реализация алгоритма

При реализации алгоритма A^* необходимо было учитывать тот факт, что для реализации этого алгоритма необходимо иметь доступ к промежуточным данным. В такой ситуации возможно два подхода: либо на каждом этапе алгоритма выводить изменения на экран, либо сохранять промежуточные данные и выводить их уже после выполнения алгоритма. Был выбран второй способ по двум основным причинам: во-первых, сохранение промежуточных результатов и их использование после завершения алгоритма позволяет просматривать шаги не только по прямому порядку их выполнения, но и в обратном порядке (возможность прокручивать шаги назад); во-вторых, данная реализация позволяет уже при начале визуализации понимать, успешно ли выполнится алгоритм или же нет. Выполнение и хранения результатов происходит в классе *AStar*. Выполнение самого алгоритма происходит в конструкторе при передаче начальной и конечной вершины, выбранной эвристики и самого графа. Всего имеется 4 варианта эвристики: расстояние

Чебышева, Манхэттенская метрика, Евклидово расстояние и нулевая эвристика (в этом случае алгоритм представляет из себя алгоритм *Дейкстры*).

Суть самого алгоритма – поиск минимального пути в графе при помощи оценок пути от конкретной точки до требуемой точки. Иными словами, при построении пути рассматриваются не все точки подряд, а те для которых эвристическая оценка пути минимальна на данном шаге. В результате выполнения алгоритма мы получаем кратчайший путь, который возможно и не полностью достоверный (так как используется эвристика), но который получен за более быстрое время. Хранение эвристических оценок происходит в мин-куче, что уменьшает время доступа к вершине с наименьшей эвристической оценкой.

На каждом шаге алгоритм запоминает эвристику и эвристическую оценку пути каждой рассмотренной вершины, а также ребро, по которому прошёл алгоритм на данном шаге. Также алгоритм содержит следующие поля: кол-во шагов работы алгоритма, стартовая и финальная вершины, список рёбер в полученном пути и его длина (под длинной пути подразумевается сумма весов рёбер, из которых он состоит). Для каждого поля реализованы соответствующие геттеры.

3.3. Графический интерфейс

Графический интерфейс реализован с помощью библиотеки *JavaFX*. В начале программа загружает макет формата *.fxml*, затем функционал дополняется специальным контроллером, реализованным в классе *MainViewController*, содержащем обработчики событий окна. В результате пользователь получает рабочий графический интерфейс (см. рис 2). Всё рабочее окно можно логически разделить на три части: панель инструментов, полотно для графа и панель для вывода сообщений.

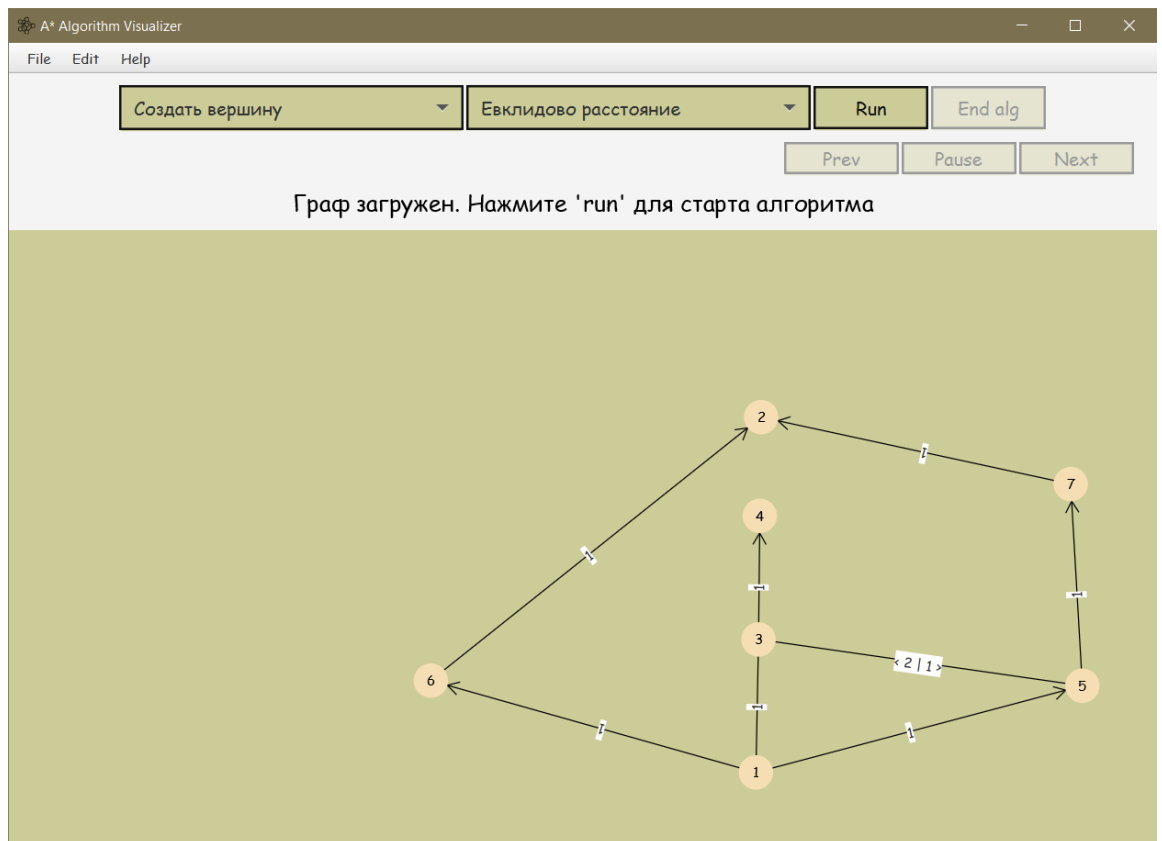


Рисунок 2 – Графический интерфейс пользователя.

Панель инструментов включает себя возможность выбрать действие над полотном (создать вершину, соединить вершины, удалить вершину/ребро), выбрать эвристику, загрузить граф из файла или сохранить его, начать визуализацию алгоритма. При этом некоторые клавиши будут недоступны, пока не будут выполнены соответствующие для них требования. Например, кнопка запуска визуализации не будет доступна, пока не будет выбрана эвристика. Также существует возможность открыть окно с подробным описанием алгоритма.

Панель с сообщениями выводит подсказки для пользователя касательно интерфейса и хода работы, а также информацию о выполнении алгоритма, после начала его визуализации. Методы для вывода сообщений на экран находятся в статическом классе *Message*. Статичность методов обусловлена тем, что возможность послать сообщение необходима из разных уровней визуализации интерфейса.

Работа с полотном реализована в классе *Canvas*, который хранит граф и позволяет интерактивно его изменять. Если в классе графа существуют методы для добавления в модель рёбер и вершин, то в классе полотна реализованы методы для добавления рёбер и вершин на экран и удаления их оттуда. После каждого изменения происходит отрисовка полотна с начала (для этого созданы отдельные методы) согласно хранящемуся в памяти графу, чтобы избежать возможных ошибок в согласовании графа в памяти и графа на экране. Рёбра и вершины на полотне имеют более сложную структуру, нежели в модели, (ввиду особенностей графического представления), поэтому для каждого из них реализован свой класс – *Edge* и *Node*. Так как класс непосредственно связан с отрисовкой графа, в нём также реализованы методы для чтения и вывода на экран графа из файла, а также сохранения графа на экране в файл. При чтении графа из файла координаты вершин масштабируются, чтобы граф поместился на полотно. Также класс содержит информацию для визуализации результатов алгоритма: рёбра, которые необходимо покрасить, вершины, которые нужно пометить и т.д. – подробнее это будет описано далее.

3.4 Визуализация алгоритма

При запуске алгоритма пользователем, сначала полностью выполняется алгоритм A^* , результаты которого затем передаются в отдельный класс – *GraphVisuals*, который содержит сведения о том, какую информацию необходимо выводить на экран (на полотно) на конкретном шаге. Более конкретно, выводятся следующие промежуточные данные (см. рис. 3): непосредственно сам граф; рёбра, пройденные к моменту данного шага (выделяются зелёным); информация об эвристиках рассмотренных вершин (в виде текста рядом с вершинами) и на последнем шаге найденный путь (выделяется жёлтым).

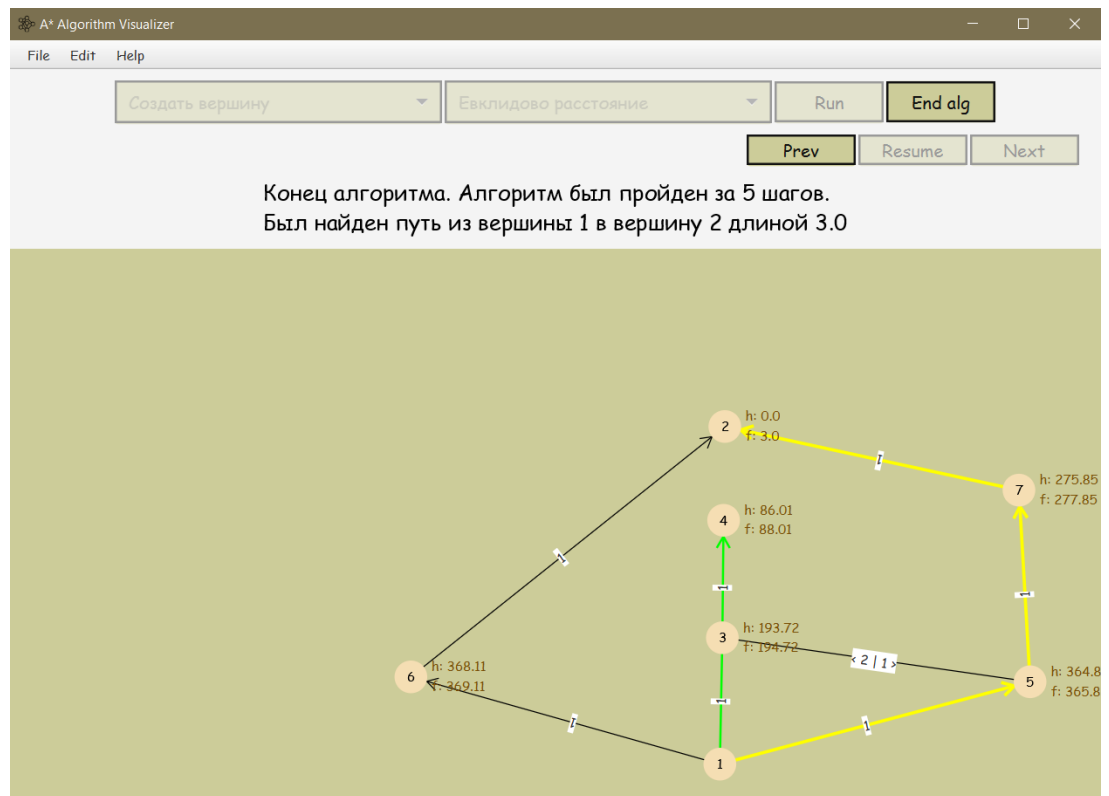


Рисунок 3 – Визуализация алгоритма

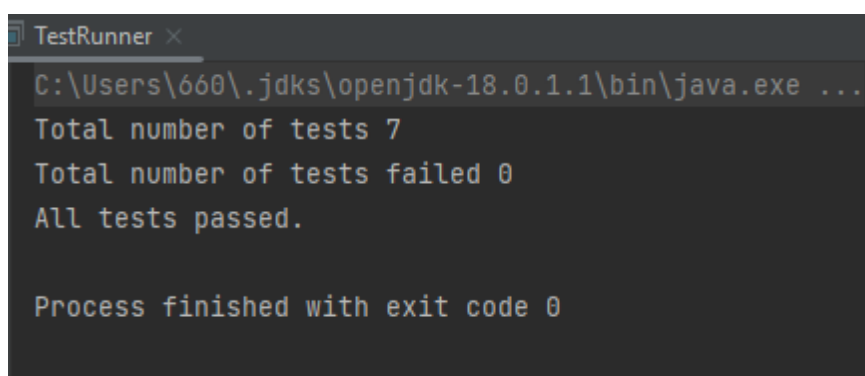
Возможны два режима просмотра алгоритма: переход к следующему шагу происходит автоматически со временем или переход вперёд/назад происходит по интеракции пользователя с окном. Для первого режима был добавлен асинхронный процесс с таймером, который вызывает смену шага каждую секунды после начала визуализации алгоритма. Для второго режима в интерфейсе были реализованы кнопки для остановки алгоритма и пошагово перехода вперёд и назад.

Полную *UML*-диаграмму классов и взаимосвязей см. в приложении А.

4. ТЕСТИРОВАНИЕ

В проекте было реализовано модульное тестирование с использованием библиотеки JUnit.

Тестирование проводилось для работы алгоритма A^* и для структуры представления графа. Для каждого случая был реализован соответствующий тест-класс, наследованный от *TestCase*, со своими тестовыми методами. Для запуска всех тестов был реализован класс *TestRunner*, который с помощью метода *runClasses()* запускает тестовый набор, составленный из предоставленных классов-наследников *TestCase*.



```
TestRunner x
C:\Users\660\.jdk\openjdk-18.0.1.1\bin\java.exe ...
Total number of tests 7
Total number of tests failed 0
All tests passed.

Process finished with exit code 0
```

Рисунок 4 – Вывод тестов класса *TestRunner*.

4.1. Тестирование структуры представления графа

Для тестирования структуры представления графа был реализован тест-класс *GraphTest*, наследованный от *TestCase*, который с помощью аннотированных(*@Test*) методов *testEdgesInfo()* и *testVerticeInfo()* проверяет, правильно ли обрабатываются и создаются графы, заданные в файлах-тестах.

Перед запуском всех тестовых методов вызывается аннотированный *@BeforeAll* метод *readGraphs()*, который считывает графы из файлов-тестов, создает из этих графов *ArrayList* для дальнейшей обработки и сравнения результатов и запоминает этот список в static переменную *graphList*.

Метод *testEdgesInfo()* проверяет для каждого тестового графа значение поля *edgesInfo* после считывания и обработки, т.е. правильно ли создались ребра графа. Для сравнения ожидаемого результата с фактическим результатом теста используется *Assertions.assertEquals()*. В случае, если результаты не совпадают, генерируется исключение *AssertionFailedError*.

Аналогично, метод *testVerticesInfo()* проверяет для каждого тестового графа значение поля *verticesInfo* после считывания и обработки, т.е. правильно ли создались вершины графа.

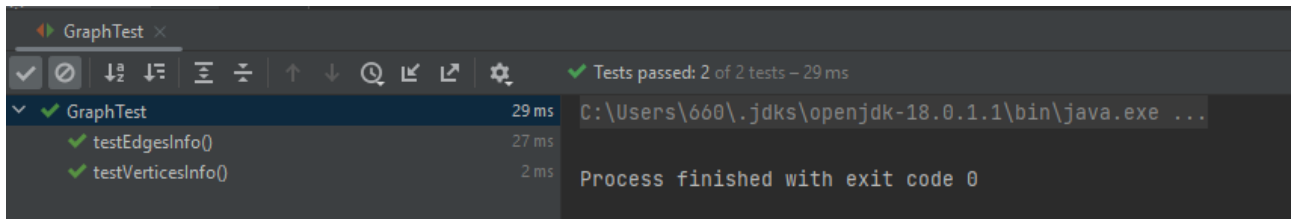


Рисунок 5 – Вывод тестов класса *GraphTest*. Без ошибки.

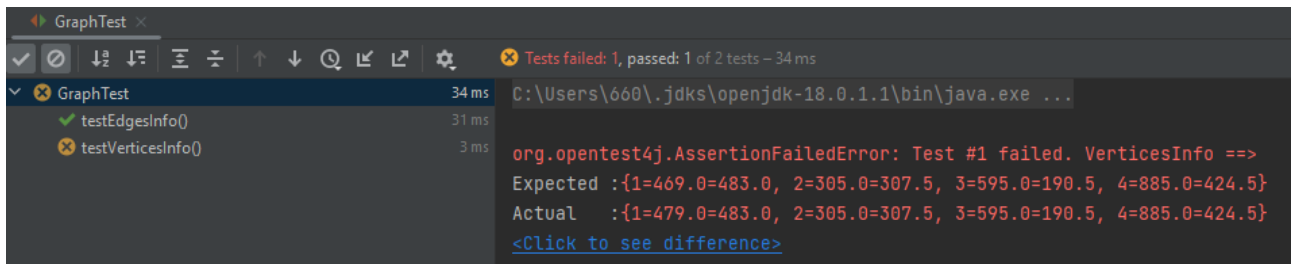


Рисунок 6 – Вывод тестов класса *GraphTest*. В случае ошибки.

4.2. Тестирование кода алгоритма A^* .

Для тестирования работы алгоритма A^* был реализован тест-класс *AStarTest*, наследованный от *TestCase*, который с помощью аннотированных (*@Test*) методов *testEdgesSteps()*, *testFinalPath()*, *testCountSteps()*, *testPathLen()*, *testHeuristics()* соответственно проверяет, корректно ли:

- запоминаются промежуточные ребра, по которым «ходил» алгоритм;
- находится итоговый кратчайший путь;
- считается количество шагов, за которое кратчайший путь был найден;
- вычисляется длина кратчайшего пути;
- вычисляются эвристики смежных вершин на каждом шаге

в процессе выполнения алгоритма A^* к графам, заданным в файлах-тестах.

Перед запуском всех тестовых методов вызывается аннотированный *@BeforeAll* метод *readGraphs()*, который считывает графы из файлов-тестов, создает из этих графов *ArrayList* для дальнейшей обработки и сравнения результатов и запоминает этот список в static переменную *graphList*. Также

создается *ArrayList* *<AStar> resList*, который хранит сгенерированные конструктором объекты *AStar*, результаты выполнения алгоритма по каждому графу из *graphlist*.

Метод *testEdgesSteps()* проверяет для каждого тестового графа значение поля *edgesSteps* объекта *AStar* после выполнения алгоритма, т.е. правильно ли запомнились промежуточные ребра, по которым «ходил» алгоритм. Для сравнения ожидаемого результата с фактическим результатом теста используется *Assertions.assertEquals()*. В случае, если результаты не совпадают, генерируется исключение *AssertionFailedError*.

Метод *testFinalPath()* проверяет для каждого тестового графа значение поля *FinalPath* объекта *AStar* после выполнения алгоритма, т.е. правильно ли найден итоговый кратчайший путь. Ожидаемые результаты сравниваются с полученными с помощью *Assertions.assertEquals()*.

Метод *testCountSteps()* проверяет для каждого тестового графа значение поля *CountSteps* объекта *AStar* после выполнения алгоритма, т.е. правильно ли подсчитано количество шагов, за которое кратчайший путь был найден. Ожидаемые результаты сравниваются с полученными с помощью *Assertions.assertEquals()*.

Метод *testPathLen()* проверяет для каждого тестового графа значение поля *PathLen* объекта *AStar* после выполнения алгоритма, т.е. правильно ли вычислена длина кратчайшего пути. Ожидаемые результаты сравниваются с полученными с помощью *Assertions.assertEquals()*.

Метод *testHeuristics()* проверяет для каждого тестового графа значение поля *PathLen* объекта *AStar* после выполнения алгоритма, т.е. правильно ли вычислены эвристики смежных вершин на каждом шаге. Ожидаемые результаты сравниваются с полученными с помощью *Assertions.assertEquals()*.

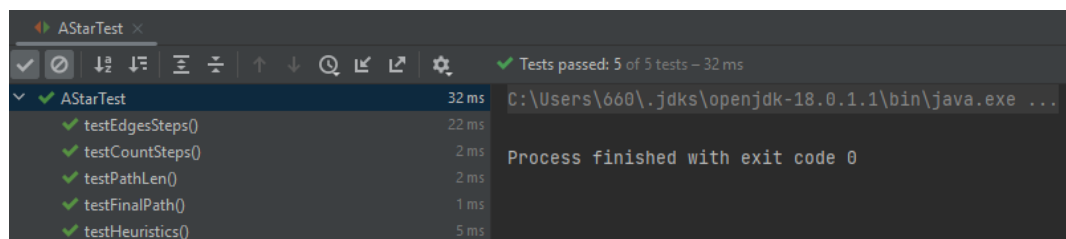


Рисунок 7 – Вывод тестов класса *AStarTest*. Без ошибки.

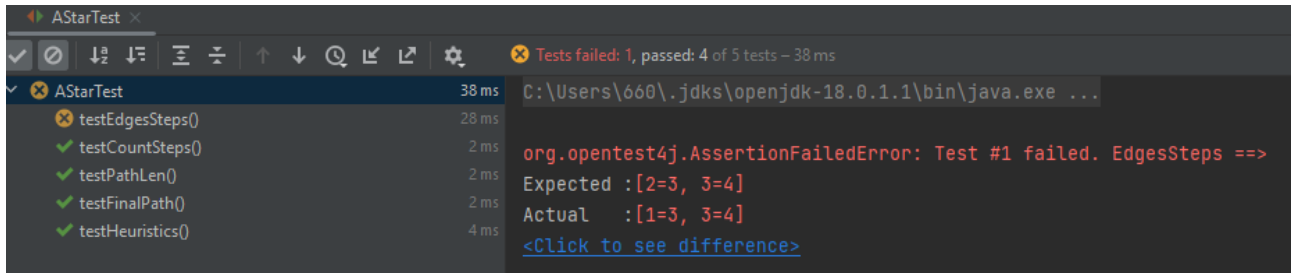


Рисунок 8 – Вывод тестов класса *AStarTest*. В случае ошибки.

4.3. Тестирование через Maven.

Тестирование можно проводить при помощи *Maven*. Для этого необходимо запустить скрипт через команду `mvn clean test`. При запуске будут проверяться все возможные тесты (см. рис. 9 и 10).

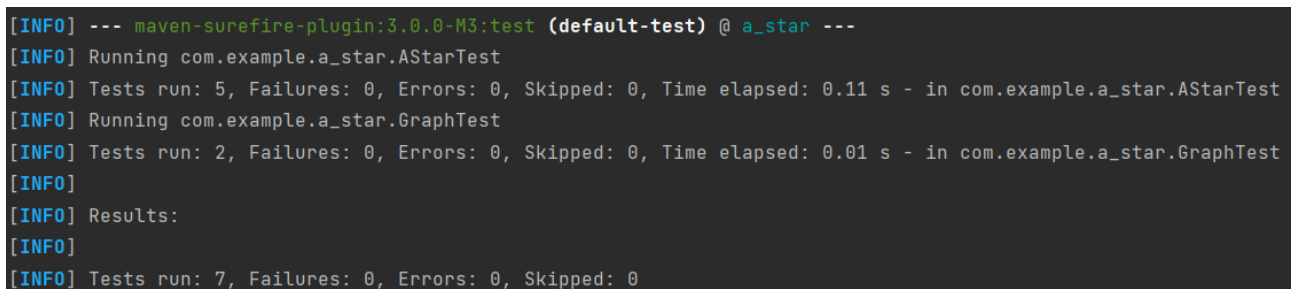


Рисунок 9 – Вывод тестов через Maven. Без ошибки.

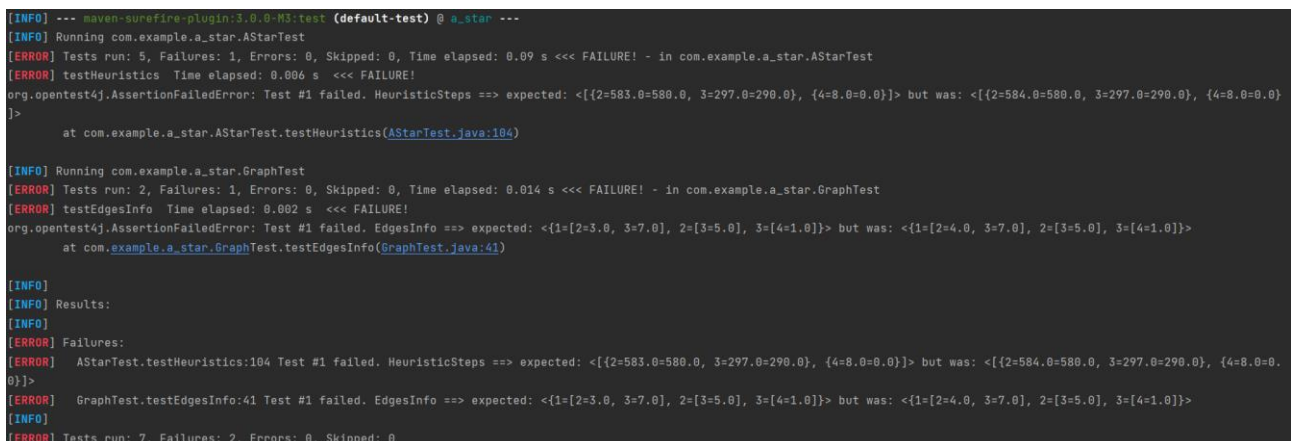


Рисунок 10 – Вывод тестов через Maven. В случае ошибки.

ЗАКЛЮЧЕНИЕ

В ходе практики были изучены основные разделы языка программирования — *Java*, а именно: базовый синтаксис, объекты, классы и пакеты, обработка ошибок, ввод-вывод в файл.

Главной задачей практики была разработка графического интерфейса для наглядного демонстрирования работы алгоритма A^* . Для этого был итеративно разработан визуализатор алгоритма A^* на основе языка *Java*, с использованием библиотеки *JavaFX*. Для ускорения времени разработки графического интерфейса был использован *FXML* файл.

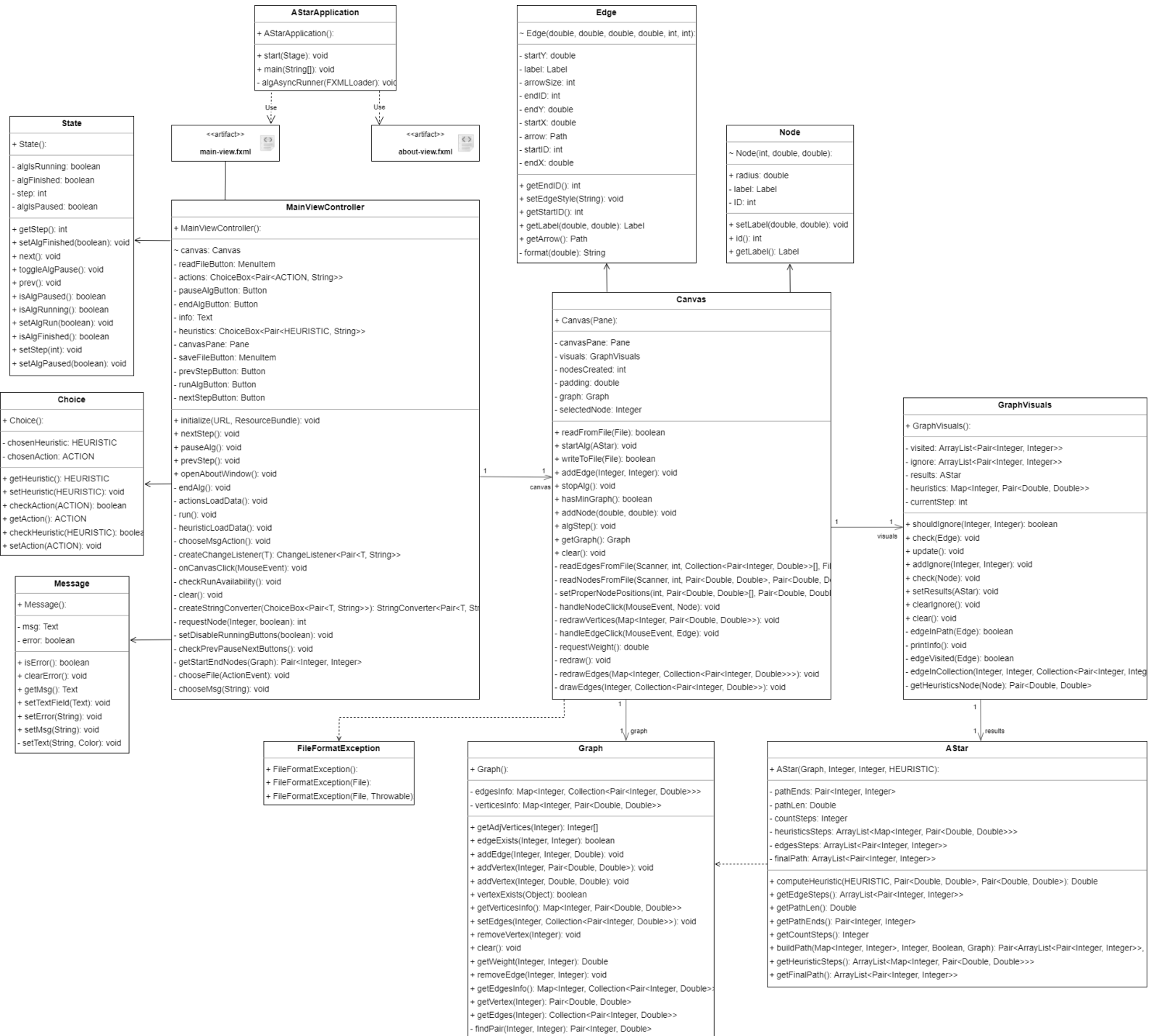
Для совместной работы была использована система контроля версий *git*, что позволило каждому участнику наиболее быстро работать над своей частью программы. Реализованный визуализатор позволяет в полной мере решать поставленную задачу, обладая при этом удобным и понятным пользователю графическим интерфейсом.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Документация к JavaFX - FXML. URL:
https://openjfx.io/javadoc/18/javafx.fxml/javafx/fxml/doc-files/introduction_to_fxml.html (дата обращения: 01.06.2022).
2. Документация к JavaFX - запуск приложения при помощи maven. URL:
<https://openjfx.io/openjfx-docs/> (дата обращения: 05.06.2022).
3. Документация к Junit. URL: <https://junit.org/junit5/docs/current/user-guide/> (дата обращения: 05.06.2022).
4. Описание алгоритма A*. URL:
https://en.wikipedia.org/wiki/A*_search_algorithm

ПРИЛОЖЕНИЕ А

UML-ДИАГРАММА



ПРИЛОЖЕНИЕ В

ИСХОДНЫЙ КОД

Название файла: *AStarApplication.java*

```
package com.example.a_star;

import javafx.animation.KeyFrame;
import javafx.animation.Timeline;
import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.stage.Stage;
import javafx.util.Duration;

import java.io.IOException;
import java.util.Objects;

public class AStarApplication extends Application {
    @Override
    public void start(Stage stage) throws IOException {
        FXMLLoader fxmlLoader = new
FXMLLoader(AStarApplication.class.getResource("main-view.fxml"));
        algAsyncRunner(fxmlLoader);
        Scene scene = new Scene(fxmlLoader.load(), 1000, 700);
        stage.setTitle("A* Algorithm Visualizer");
        stage.getIcons().add(new
Image(Objects.requireNonNull(getClass().getResourceAsStream("icon.png"))));
        stage.setScene(scene);
        stage.show();
        ((MainViewController) fxmlLoader.getController()).openAboutWindow();
    }

    private void algAsyncRunner(FXMLLoader fxmlLoader) {
        Timeline timer = new Timeline(
            new KeyFrame(Duration.seconds(1), event -> {
                MainViewController mainViewController =
fxmlLoader.getController();
                if (!State.isAlgPaused()) mainViewController.nextStep();
            }));
        timer.setCycleCount(Timeline.INDEFINITE);
        timer.play();
    }

    public static void main(String[] args) { launch(); }
}
```

Название файла: *Canvas.java*

```
package com.example.a_star;

import javafx.scene.control.TextInputDialog;
import javafx.scene.input.MouseButton;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.Pane;
import javafx.util.Pair;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.*;
```

```

import static com.example.a_star.Choice.*;

public class Canvas {
    private final Pane canvasPane;
    private final Graph graph;
    private final GraphVisuals visuals;
    private int nodesCreated;
    private Integer selectedNode;
    private final double padding = 50;

    public Canvas(Pane pane) {
        this.canvasPane = pane;
        this.graph = new Graph();
        this.visuals = new GraphVisuals();
        nodesCreated = 0;
    }

    public Graph getGraph() {
        return graph;
    }

    public void addNode(double x, double y) {
        if(x > Node.radius && x < canvasPane.getWidth()-Node.radius &&
            y > Node.radius && y < canvasPane.getHeight()-Node.radius) {
            graph.addVertex(++nodesCreated, x, y);
            redraw();
        }
    }

    public void addEdge(Integer node1, Integer node2) {
        if(!graph.edgeExists(node1, node2)) {
            double weight = requestWeight();
            if(weight > 0) {
                graph.addEdge(node1, node2, weight);
                redraw();
            } else Message.setError("Вес должен быть числом больше нуля");
        }
    }

    private void handleNodeClick(MouseEvent e, Node node) {
        if (e.getButton() == MouseButton.PRIMARY && checkAction(ACTION.CONNECT))
        {
            if(selectedNode != null && selectedNode != node.id()){
                addEdge(selectedNode, node.id());
                selectedNode = null;
            } else {
                selectedNode = node.id();
            }
        } else if(e.getButton() == MouseButton.PRIMARY &&
checkAction(ACTION.DELETE)) {
            graph.removeVertex(node.id());
            redraw();
        }
    }

    private void handleEdgeClick(MouseEvent e, Edge edge) {
        if(e.getButton() == MouseButton.PRIMARY && checkAction(ACTION.DELETE)) {
            graph.removeEdge(edge.getStartID(), edge.getEndID());
            graph.removeEdge(edge.getEndID(), edge.getStartID());
            redraw();
        }
    }
}

```

```

private double requestWeight(){
    TextInputDialog dialog = new TextInputDialog("0");
    dialog.setTitle(null);
    dialog.setContentText(null);
    dialog.setHeaderText("Введите вес ребра:");

    try{ return dialog.showAndWait().map(Double::parseDouble).orElse(0.0); }
    catch (NumberFormatException ignored){ }
    return 0;
}

private void redraw(){
    canvasPane.getChildren().clear();
    visuals.clearIgnore();

    Map<Integer, Pair<Double, Double>> verticesInfo =
graph.getVerticesInfo();
    Map<Integer, Collection<Pair<Integer, Double>>> edgesInfo =
graph.getEdgesInfo();

    redrawEdges(edgesInfo);
    redrawVertices(verticesInfo);
}

private void redrawVertices(Map<Integer, Pair<Double, Double>> verticesInfo)
{
    for(Integer id: verticesInfo.keySet()){
        Pair<Double, Double> pair = verticesInfo.get(id);
        Node node = new Node(id, pair.getKey(), pair.getValue());
        if(State.isAlgRunning())
            visuals.check(node);
        canvasPane.getChildren().add(node);
        canvasPane.getChildren().add(node.getLabel());
        node.setOnMouseClicked(e -> handleNodeClick(e, node));
    }
}

private void redrawEdges(Map<Integer, Collection<Pair<Integer, Double>>>
edgesInfo) {
    for(Integer start: edgesInfo.keySet()){
        Collection<Pair<Integer, Double>> collection;
        if((collection = edgesInfo.get(start)) != null){
            drawEdges(start, collection);
        }
    }
}

private void drawEdges(Integer start, Collection<Pair<Integer, Double>>
collection) {
    Pair<Double, Double> startPoint = graph.getVertex(start);
    for (Pair<Integer, Double> pair : collection) {
        Integer end = pair.getKey();
        Double weight = pair.getValue();
        if(visuals.shouldIgnore(start, end)) continue;

        Pair<Double, Double> endPoint = graph.getVertex(end);
        Edge edge = new Edge(
            startPoint.getKey(), startPoint.getValue(),
endPoint.getKey(), endPoint.getValue(),
            start, end);
        if(State.isAlgRunning())
            visuals.check(edge);
        canvasPane.getChildren().add(edge);
    }
}

```



```

        if(!graph.edgeExists(end, start)){
            canvasPane.getChildren().add(edge.getArrow());
            canvasPane.getChildren().add(edge.getLabel(weight, 0));
        }else{
            canvasPane.getChildren().add(edge.getLabel(weight,
graph.getWeight(end, start)));
            visuals.addIgnore(end, start);
        }
        edge.setOnMouseClicked(e -> handleEdgeClick(e, edge));
    }
}

public void clear(){
    graph.clear();
    nodesCreated = 0;
    redraw();
}

@SuppressWarnings("unchecked")
public boolean readFromFile(File file) {
    try(Scanner sc = new Scanner(file)){
        if(!sc.hasNextLine()) throw new FileFormatException(file);
        String data = sc.nextLine();
        int N;
        try{
            if((N = Integer.parseInt(data)) < 0) throw new Exception();
        }catch (Exception e){ throw new FileFormatException(file, e); }

        Pair<Double, Double>[] nodes = new Pair[N];
        Collection<Pair<Integer, Double>>[] edges = new Collection[N];
        Pair<Double, Double> maxXY = new Pair<>(1.0, 1.0);

        maxXY = readNodesFromFile(sc, N, maxXY, nodes, file);
        readEdgesFromFile(sc, N, edges, file);
        setProperNodePositions(N, nodes, maxXY);

        graph.clear();
        nodesCreated = N;
        for(int i = 0; i < N; i++){
            graph.addVertex(i+1, nodes[i]);
        }
        for(int i = 0; i < N; i++){
            graph.setEdges(i+1, edges[i]);
        }
        redraw();
        return true;
    } catch (IOException e){
        e.printStackTrace();
    } catch (FileFormatException e) {
        Message.setError(e.getMessage());
    }
    redraw();
    return false;
}

public boolean writeToFile(File file){
    try (FileWriter fw = new FileWriter(file)){
        fw.write((graph.getVerticesInfo().keySet().size()) + "\n");

        for (Integer i: graph.getVerticesInfo().keySet()) {
            if (graph.vertexExists(i)){
                fw.write((graph.getVertex(i).getKey() - padding) + " " +
(graph.getVertex(i).getValue() - padding)+ "\n");
            }
        }
    }
}

```

```

        for (Integer i: graph.getVerticesInfo().keySet()) {
            for (Integer j: graph.getVerticesInfo().keySet()) {
                fw.write(graph.getWeight(i, j) + " ");
            }
            fw.write('\n');
        }
        return true;
    } catch (IOException e) { System.out.println(e.getMessage()); }
    return false;
}

private void setProperNodePositions(int N, Pair<Double, Double>[] nodes,
Pair<Double, Double> maxXY) {
    double width = canvasPane.getWidth() - 2*Node.radius - 2*padding;
    double height = canvasPane.getHeight() - 2*Node.radius - 2*padding;
    double scaleX = width/ maxXY.getKey();
    double scaleY = height/ maxXY.getValue();

    for(int i = 0; i < N; i++) {
        double x = nodes[i].getKey(), y = nodes[i].getValue();
        nodes[i] = new Pair<>(x*scaleX+Node.radius+padding,
y*scaleY+Node.radius+padding);
    }
}

private void readEdgesFromFile(Scanner sc, int N, Collection<Pair<Integer,
Double>>[] edges, File file) throws FileFormatException {
    String[] tmp;
    for(int i = 0; i < N; i++){
        if(!sc.hasNextLine() ||
            (tmp = sc.nextLine().split(" ")).length != N) throw new
FileFormatException(file);
        Collection<Pair<Integer, Double>> collection = new ArrayList<>();
        for(int j = 0; j < N; j++) {
            double weight;
            try {
                if((weight = Double.parseDouble(tmp[j])) < 0) throw new
Exception();
                if (i != j && weight != 0) collection.add(new Pair<>(j + 1,
weight));
            } catch (Exception e) { throw new FileFormatException(file); }
        }
        if(collection.size()>0)
            edges[i] = collection;
    }
}

private Pair<Double, Double> readNodesFromFile(Scanner sc, int N,
Pair<Double, Double> maxXY, Pair<Double, Double>[] nodes, File file) throws
FileFormatException {
    String[] tmp;
    for(int i = 0; i < N; i++){
        if(!sc.hasNextLine() ||
            (tmp = sc.nextLine().split(" ")).length != 2) throw new
FileFormatException(file);
        double x, y;
        try{
            if((x = Double.parseDouble(tmp[0])) < 0 || (y =
Double.parseDouble(tmp[1])) < 0) throw new Exception();
        } catch (Exception e) { throw new FileFormatException(file, e); }

        maxXY = new Pair<>(Math.max(x, maxXY.getKey()), Math.max(y,

```

```

maxXY.getValue());
        nodes[i] = new Pair<>(x, y);
    }
    return maxXY;
}

public void startAlg(AStar results){
    visuals.setResults(results);
    visuals.update();
    redraw();
}

public void algStep(){
    if(State.isAlgRunning()){
        visuals.update();
        redraw();
    }
}

public boolean hasMinGraph() {
    return graph.getVerticesInfo().size() > 1;
}

public void stopAlg() {
    visuals.clear();
    redraw();
}
}

```

Название файла: *Choice.java*

```
package com.example.a_star;
```

```

public class Choice {
    public enum ACTION { NONE, ADD, CONNECT, DELETE }
    public enum HEURISTIC { NONE, CHEBYSHEV, MANHATTAN, EUCLID, DIJKSTRA}

    private static ACTION chosenAction = ACTION.NONE;
    private static HEURISTIC chosenHeuristic = HEURISTIC.NONE;

    public static ACTION getAction() {
        return chosenAction;
    }

    public static void setAction(ACTION action) {
        chosenAction = action;
    }

    public static boolean checkAction(ACTION action){
        return chosenAction == action;
    }

    public static HEURISTIC getHeuristic() {
        return chosenHeuristic;
    }

    public static void setHeuristic(HEURISTIC heuristic) {
        chosenHeuristic = heuristic;
    }

    public static boolean checkHeuristic(HEURISTIC heuristic){
        return chosenHeuristic == heuristic;
    }
}

```

Название файла: *AStar.java*

```
package com.example.a_star;

import java.util.*;
import javafx.util.Pair;
import static java.lang.Math.*;

public class AStar {
    private final ArrayList<Map<Integer, Pair<Double, Double>>> heuristicsSteps;
    //список h(x) и f(x) рассматриваемых вершин на каждом шаге
    private final ArrayList<Pair<Integer, Integer>> edgesSteps;           //ребра,
    которые должны подкрашиваться на каждом шаге
    private Double pathLen;
    //длина найденного кратчайшего пути
    private Integer countSteps;
    //количество шагов
    private final Pair<Integer, Integer> pathEnds;
    private final ArrayList<Pair<Integer, Integer>> finalPath;
    //итоговый кратч.путь - н-р, список ребер [(1,2), (2,3), (3,4)]

    public Double computeHeuristic(Choice.HEURISTIC heur, Pair<Double, Double>
    coordsCur, Pair<Double, Double> coordsEnd) {
        try {
            return switch (heur) {
                case EUCLID ->
                    Math.sqrt(pow((coordsCur.getKey() - coordsEnd.getKey()),
2) + pow((coordsCur.getValue() - coordsEnd.getValue()), 2));
                case CHEBYSHEV ->
                    max(abs(coordsCur.getKey() - coordsEnd.getKey()),
abs(coordsCur.getValue() - coordsEnd.getValue()));
                case MANHATTAN ->
                    abs(coordsCur.getKey() - coordsEnd.getKey()) +
abs(coordsCur.getValue() - coordsEnd.getValue());
                case DIJKSTRA -> 0.0;
                default -> null;
            };
        } catch (IllegalArgumentException e){
            System.out.println(e.getMessage());
            return 0.0;
        }
    }

    public Pair<ArrayList<Pair<Integer, Integer>>, Double>
    buildPath(Map<Integer, Integer> map, Integer end, Boolean deadlock, Graph graph)
    {
        ArrayList<Pair<Integer, Integer>> path = new ArrayList<>();
        Double finalWeight = 0.0;
        if (deadlock) return new Pair<>(path, 0.0);
        while (!end.equals(-1)) {
            Integer tmp = end;
            end = map.get(end);
            if (!end.equals(-1))
                path.add(new Pair<>(end, tmp));
            finalWeight += graph.getWeight(end, tmp);
        }
        ArrayList<Pair<Integer, Integer>> reversePath = new ArrayList<>();
        for (int i = path.size() - 1; i >= 0; i--)
            reversePath.add(path.get(i)); // Append the elements in reverse
order
        return new Pair<>(reversePath, finalWeight);
    }
}
```

```

    }

    public AStar(Graph graph, Integer start, Integer end, Choice.HEURISTIC heur)
    {
        pathEnds = new Pair<>(start, end);
        heuristicsSteps = new ArrayList<>();
        edgesSteps = new ArrayList<>();
        pathLen = 0.0;
        countSteps = -1; //т.к. в цикле while извлечение стартовой вершины из
        кучи тоже считается за шаг, а нам нужно учитывать только переходы по ребру

        //prepare to alg
        Map<Integer, Double> startDists = new HashMap<>(); //хранит
        g(x) для вершин
        startDists.put(start, 0.0);

        PriorityQueue<Pair<Pair<Integer, Integer>, Double>> heap = new
        PriorityQueue<>(Comparator.comparingDouble(Pair::getValue));
        heap.add(new Pair<>(new Pair<>(start, start), 0.0));

        Map<Integer, Integer> path = new HashMap<>(); //map хранит пары: вершина
        - откуда в нее пришли
        path.put(start, -1);
        boolean deadlock = false;

        //start the alg
        while (!heap.isEmpty()) {
            Pair<Integer, Integer> pairCurPrev = heap.poll().getKey();
            Integer current = pairCurPrev.getKey();
            Integer prev = pairCurPrev.getValue();
            edgesSteps.add(new Pair<>(prev, current)); //запоминаем ребра, по
            которым гуляем
            countSteps++;
            deadlock = false;

            if (current.equals(end)) break;
            if (!graph.vertexExists(current)) continue;
            Map<Integer, Pair<Double, Double>> adjacentHeur = new HashMap<>();
            //создаем Map эвристик смежн вершин: (смежная вершина-Pair(f(x), h(x))), где
            f(x)=g(x)+h(x)

            Collection<Pair<Integer, Double>> listAdj =
            graph.getEdgesInfo().get(current); //коллекция смежных вершин
            boolean check = false;
            if(listAdj != null){
                Collection<Integer> listAdjVertices = new ArrayList<>();
                for (Pair<Integer, Double> pairVertexWeight : listAdj) {
                    listAdjVertices.add(pairVertexWeight.getKey());
                }
                check = !startDists.keySet().containsAll(listAdjVertices);
            }

            if (check){
                for (Pair<Integer, Double> pairNextWeight : listAdj) {
                    //проходимся по смежным
                    Integer next = pairNextWeight.getKey();
                    //смежная вершина
                    Double weight = pairNextWeight.getValue();
                    //вес ребра до нее
                    Double g = startDists.get(current) + weight;
                    //находим g(x) для смежной
                    if (!startDists.containsKey(next) || g <

```

```

startDists.get(next)) {
    path.put(next, current);
    startDists.put(next, g); //запомнили g(x) для next
    Double h = computeHeuristic(heur,
graph.getVerticesInfo().get(next), graph.getVerticesInfo().get(end)); //найдем
h(x)

    Double f = g + h;
    adjacentHeur.put(next, new Pair<>(f, h));
    heap.add(new Pair<>(new Pair<>(next, current), f));
}
}
}
else deadlock = true;
heuristicsSteps.add(adjacentHeur);
}
edgesSteps.remove(0);
if (countSteps != 0) {
    Pair<ArrayList<Pair<Integer, Integer>>, Double> resultsAboutPath =
buildPath(path, end, deadlock, graph);
    finalPath = resultsAboutPath.getKey();
    pathLen = resultsAboutPath.getValue();
}
else {
    finalPath = new ArrayList<>();
    pathLen = 0.0;
}
}

public Double getPathLen() { return pathLen; }
public Integer getCountSteps() { return countSteps; }
public ArrayList<Pair<Integer, Integer>> getFinalPath() { return finalPath;}

public ArrayList<Map<Integer, Pair<Double,Double>>> getHeuristicSteps()
{return heuristicsSteps; }
public ArrayList<Pair<Integer, Integer>> getEdgeSteps() {return edgesSteps;
}
public Pair<Integer, Integer> getPathEnds() { return pathEnds; }
}

```

Название файла: *Edge.java*

```
package com.example.a_star;
```

```

import javafx.scene.control.Label;
import javafx.scene.shape.Line;
import javafx.scene.shape.LineTo;
import javafx.scene.shape.MoveTo;
import javafx.scene.shape.Path;
import javafx.scene.text.Text;

```

```

public class Edge extends Line {
    private final static int arrowSize = 11;
    private final int startID;
    private final int endID;
    private double startX, startY, endX, endY;
    private final Path arrow;
    private final Label label;

```

```

    Edge(double startX, double startY, double endX, double endY, int start, int
end){
        super(startX, startY, endX, endY);
        this.startX = startX;

```

```

        this.startY = startY;
        this.endX = endX;
        this.endY = endY;
        this.startID = start;
        this.endID = end;
        this.arrow = new Path();
        this.label = new Label();
    }

    public Path getArrow(){
        double angle = Math.atan2((endY - startY), (endX - startX)) - Math.PI /
2.0;

        double sin = Math.sin(angle);
        double cos = Math.cos(angle);
        endX += sin*Node.radius;
        endY -= cos*Node.radius;
        double x1 = (- 1.0 / 2.0 * cos + Math.sqrt(3) / 2 * sin) * arrowSize +
endX;
        double y1 = (- 1.0 / 2.0 * sin - Math.sqrt(3) / 2 * cos) * arrowSize +
endY;
        double x2 = (1.0 / 2.0 * cos + Math.sqrt(3) / 2 * sin) * arrowSize +
endX;
        double y2 = (1.0 / 2.0 * sin - Math.sqrt(3) / 2 * cos) * arrowSize +
endY;

        arrow.getElements().add(new MoveTo(endX, endY));
        arrow.getElements().add(new LineTo(x1, y1));
        arrow.getElements().add(new MoveTo(endX, endY));
        arrow.getElements().add(new LineTo(x2, y2));

        return arrow;
    }

    public Label getLabel(double weightForward, double weightBackward){
        double angle = Math.toDegrees(Math.atan2((endY - startY), (endX -
startX)));

        String weight = (weightBackward != 0 && weightForward != weightBackward)
?
            "< " + format(weightBackward) + " | " + format(weightForward) +
" >" : format(weightForward);
        Text text = new Text(weight);

        label.setText(weight);
        label.setLayoutX(Math.min(startX, endX)+Math.abs(startX-endX)/2-
text.getLayoutBounds().getWidth()/2);
        label.setLayoutY(Math.min(startY, endY)+Math.abs(startY-endY)/2-
text.getLayoutBounds().getHeight()/2);
        label.setStyle("-fx-background-color: #fff; -fx-rotate: " + angle);
        return label;
    }

    public int getStartID() {
        return startID;
    }

    public int getEndID() {
        return endID;
    }

    private static String format(double val) {
        if(val == (long)val)
            return String.format("%d", (long) val);
    }

```

```

        else
            return String.format("%s", val);
    }

    public void setEdgeStyle(String style){
        this.setStyle(style);
        if (arrow != null) arrow.setStyle(style);
    }
}

```

Название файла: *FileFormatException.java*

```

package com.example.a_star;

import java.io.File;

public class FileFormatException extends Exception {
    public FileFormatException(){
        super();
    }

    public FileFormatException(File file){
        super("Неверный формат файла: " + file.getName());
    }

    public FileFormatException(File file, Throwable t){
        super("Неверный формат файла: " + file.getName(), t);
    }
}

```

Название файла: *Graph.java*

```

package com.example.a_star;

import javafx.util.Pair;
import java.util.ArrayList;
import java.util.Collection;
import java.util.HashMap;
import java.util.Map;

public class Graph {
    private final Map<Integer, Collection<Pair<Integer, Double>>> edgesInfo;
    private final Map<Integer, Pair<Double, Double>> verticesInfo;

    public Graph(){
        edgesInfo = new HashMap<>();
        verticesInfo = new HashMap<>();
    }

    public void addEdge(Integer start, Integer end, Double weight){
        edgesInfo.putIfAbsent(start, new ArrayList<>());
        edgesInfo.get(start).add(new Pair<>(end, weight));
    }

    public void setEdges(Integer start, Collection<Pair<Integer, Double>> collection){
        edgesInfo.put(start, collection);
    }

    public void addVertex(Integer id, Double x, Double y){
        verticesInfo.putIfAbsent(id, new Pair<>(x, y));
    }

    public void addVertex(Integer id, Pair<Double, Double> pair){
        verticesInfo.putIfAbsent(id, pair);
    }
}

```



```

public Pair<Double, Double> getVertex(Integer id){
    return verticesInfo.get(id);
}

public void removeEdge(Integer start, Integer end){
    Collection<Pair<Integer, Double>> collection;
    if((collection = edgesInfo.get(start)) != null){
        collection.removeIf(pair -> pair.getKey().equals(end));
    }
}

public void removeVertex(Integer id){
    verticesInfo.remove(id);
    edgesInfo.remove(id);
    for(Integer key: edgesInfo.keySet())
        removeEdge(key, id);
}

public boolean edgeExists(Integer start, Integer end){
    return findPair(start, end) != null;
}

public boolean vertexExists(Object v){
    return verticesInfo.containsKey(v);
}

public Double getWeight(Integer start, Integer end){
    Pair<Integer, Double> pair;
    return (pair = findPair(start, end)) != null? pair.getValue() : 0;
}

public Integer[] getAdjVertices(Integer id){
    Collection<Pair<Integer, Double>> collection;
    if((collection = edgesInfo.get(id)) != null) {
        Integer[] vertices = new Integer[collection.size()];
        int i = 0;
        for (Pair<Integer, Double> pair : collection)
            vertices[i++] = pair.getKey();
        return vertices;
    }
    return null;
}

public Map<Integer, Pair<Double, Double>> getVerticesInfo() { return
verticesInfo; }
public Map<Integer, Collection<Pair<Integer, Double>>> getEdgesInfo() {
return edgesInfo; }
public Collection<Pair<Integer, Double>> getEdges(Integer key){
    return edgesInfo.get(key);
}

private Pair<Integer, Double> findPair(Integer start, Integer end){
    Collection<Pair<Integer, Double>> collection;
    if((collection = edgesInfo.get(start)) != null){
        for (Pair<Integer, Double> pair : collection) {
            if (pair.getKey().equals(end)) return pair;
        }
    }
    return null;
}

public void clear(){

```

```

        edgesInfo.clear();
        verticesInfo.clear();
    }
}

```

Название файла: *GraphVisuals.java*

```

package com.example.a_star;

import javafx.util.Pair;

import java.util.ArrayList;
import java.util.Collection;
import java.util.HashMap;
import java.util.Map;

/**
 * Contains information about algorithm on current step
 * This information is used to visualize algorithm on graph
 */
public class GraphVisuals{
    private AStar results;
    private final ArrayList<Pair<Integer, Integer>> ignore;
    private final ArrayList<Pair<Integer, Integer>> visited;
    private final Map<Integer, Pair<Double, Double>> heuristics;
    private int currentStep;

    public GraphVisuals() {
        this.ignore = new ArrayList<>();
        this.visited = new ArrayList<>();
        this.heuristics = new HashMap<>();
        this.currentStep = 0;
    }

    public void setResults(AStar results){
        this.results = results;
    }

    public void addIgnore(Integer end, Integer start) {
        ignore.add(new Pair<>(end, start));
    }

    public boolean shouldIgnore(Integer start, Integer end) {
        return edgeInCollection(start, end, ignore);
    }

    public void clearIgnore() { ignore.clear(); }

    public void update(){
        int step = Math.min(State.getStep(), results.getCountSteps());
        if(step > this.currentStep){
            visited.addAll(results.getEdgeSteps().subList(currentStep, step));
            for(int i = currentStep; i < step; i++)
                heuristics.putAll(results.getHeuristicSteps().get(i));
        }else if(step < this.currentStep){
            visited.subList(step, currentStep).clear();
            for(int i = step; i < currentStep; i++) {
                heuristics.keySet().removeAll(results.getHeuristicSteps().get(i).keySet());
            }
        }
        currentStep = step;
        if(step == results.getCountSteps()){
            State.setAlgFinished(true);
            State.setAlgPaused(true);
        }
    }
}

```

```

    }

    printInfo();

    State.setAlgFinished(step == results.getCountSteps());
    State.setAlgPaused(State.isAlgPaused() || State.isAlgFinished());
}

private boolean edgeVisited(Edge edge){
    return edgeInCollection(edge.getStartID(), edge.getEndID(), visited)
        || edgeInCollection(edge.getEndID(), edge.getStartID(),
visited);
}

private boolean edgeInPath(Edge edge){
    return (currentStep == results.getCountSteps())
        && (edgeInCollection(edge.getStartID(), edge.getEndID(),
results.getFinalPath())
        || edgeInCollection(edge.getEndID(), edge.getStartID(),
results.getFinalPath()));
}

private Pair<Double, Double> getHeuristicsNode(Node node){
    return heuristics.get(node.id());
}

private boolean edgeInCollection(Integer start, Integer end,
Collection<Pair<Integer, Integer>> collection){
    for (Pair<Integer, Integer> pair : collection)
        if(pair.getKey().equals(start) && pair.getValue().equals(end))
            return true;
    return false;
}

public void check(Edge edge){
    if(edgeInPath(edge))
        edge.setEdgeStyle("-fx-stroke-width: 3; -fx-stroke: #FFFF00;");
    else if (edgeVisited(edge))
        edge.setEdgeStyle("-fx-stroke-width: 2; -fx-stroke: #00FF00;");
}

public void check(Node node){
    Pair<Double, Double> pair = getHeuristicsNode(node);
    if(pair != null)
        node.setLabel(pair.getKey(), pair.getValue());
}

public void clear(){
    results = null;
    currentStep = 0;
    visited.clear();
    heuristics.clear();
}

private void printInfo() {
    String res;
    if(currentStep == 0){
        res = "Начало алгоритма. Находимся в вершине " +
results.getPathEnds().getKey()
        + ".\nНеобходимо дойти до вершины " +
results.getPathEnds().getValue();
    }else if(currentStep == results.getCountSteps()){
        String path = "из вершины " + results.getPathEnds().getKey() + " в

```

```

вершину " + results.getPathEnds().getValue();
        res = "Конец алгоритма. Алгоритм был пройден за " +
results.getCountSteps() + " шагов.\n"
        + (results.getPathLen() > 0 ? "Был найден путь " + path + "
длиной " + results.getPathLen()
        : "Путь " + path + " не был найден" );
    }else{
        res = "Рассматриваем ещё неучтённые соседние вершины: "
        + results.getHeuristicSteps().get(currentStep-1).keySet()
        + "\nПереходим в вершину с минимальной эвристической
оценкой: "
        + results.getEdgeSteps().get(currentStep-1).getValue();
    }
    Message.setMsg(res);
}
}

```

Название файла: *MainViewController.java*

```

package com.example.a_star;

import javafx.beans.value.ChangeListener;
import javafx.collections.FXCollections;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.ChoiceBox;
import javafx.scene.control.MenuItem;
import javafx.scene.control.TextInputDialog;
import javafx.scene.image.Image;
import javafx.scene.input.MouseButton;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.Pane;
import javafx.scene.text.Text;
import javafx.stage.FileChooser;
import javafx.stage.Stage;
import javafx.util.Pair;
import javafx.util.StringConverter;

import java.io.File;
import java.net.URL;
import java.util.Objects;
import java.util.ResourceBundle;

import static com.example.a_star.Choice.*;

public class MainViewController implements Initializable {
    Canvas canvas;

    @FXML
    private Pane canvasPane;
    @FXML
    private ChoiceBox<Pair<ACTION, String>> actions;
    @FXML
    private ChoiceBox<Pair<HEURISTIC, String>> heuristics;
    @FXML
    private MenuItem readFileButton;
    @FXML
    private MenuItem saveFileButton;
    @FXML

```

```

private Button runAlgButton;
@FXML
private Button endAlgButton;
@FXML
private Button prevStepButton;
@FXML
private Button pauseAlgButton;
@FXML
private Button nextStepButton;
@FXML
private Text info;

@Override
public void initialize(URL url, ResourceBundle rb) {
    canvas = new Canvas(canvasPane);
    Message.setTextField(info);
    actions.setConverter(createStringConverter(actions));

    actions.getSelectionModel().selectedItemProperty().addListener(createChangeListener(ACTION.NONE));
    actions.loadData();
    heuristics.setConverter(createStringConverter(heuristics));

    heuristics.getSelectionModel().selectedItemProperty().addListener(createChangeListener(HEURISTIC.NONE));
    heuristicLoadData();
    Message.setMsg("Создайте граф при помощи инструментов или импортируйте из файла");
}

@FXML
private void run() {
    Graph graph = canvas.getGraph();
    Pair<Integer, Integer> pair = getStartEndNodes(graph);
    if(pair != null){
        setDisableRunningButtons(false);
        AStar alg = new AStar(graph, pair.getKey(), pair.getValue(),
getHeuristic());
        if(alg.getCountSteps() > 0){
            canvas.startAlg(alg);
            Message.setMsg("Начало алгоритма");
        }else{
            endAlg();
            Message.setMsg("Пути из начальной вершины не существует");
        }
    }
}

@FXML
private void endAlg() {
    setDisableRunningButtons(true);
    State.setStep(0);
    State.setAlgFinished(true);
    canvas.stopAlg();
    Message.setMsg("Алгоритм остановлен");
}

private void setDisableRunningButtons(boolean disableOrNot) {
    State.setAlgRun(!disableOrNot);
    State.setAlgPaused(disableOrNot);
    State.setAlgFinished(disableOrNot);
    runAlgButton.setDisable(!disableOrNot);
    endAlgButton.setDisable(disableOrNot);
}

```

```

        actions.setDisable(!disableOrNot);
        heuristics.setDisable(!disableOrNot);
        checkPrevPauseNextButtons();
    }

    private void checkPrevPauseNextButtons(){
        nextStepButton.setDisable(State.isAlgFinished() || !State.isAlgPaused()
|| !State.isAlgRunning());
        prevStepButton.setDisable(State.getStep() == 0 || !State.isAlgPaused()
|| !State.isAlgRunning());
        pauseAlgButton.setDisable(State.isAlgFinished() ||
!State.isAlgRunning());
        pauseAlgButton.setText(State.isAlgPaused() ? "Resume" : "Pause");
    }

    @FXML
    public void nextStep(){
        State.next();
        canvas.algStep();
        checkPrevPauseNextButtons();
    }

    @FXML
    private void prevStep(){
        State.prev();
        canvas.algStep();
        checkPrevPauseNextButtons();
    }

    @FXML
    private void pauseAlg(){
        State.toggleAlgPause();
        checkPrevPauseNextButtons();
    }

    @FXML
    private void chooseFile(ActionEvent e) {
        if(!State.isAlgRunning()) {
            FileChooser fileChooser = new FileChooser();
            FileChooser.ExtensionFilter extFilter = new
FileChooser.ExtensionFilter("TXT files (*.txt)", "*.txt");
            fileChooser.getExtensionFilters().add(extFilter);
            MenuItem node = (MenuItem) e.getSource();
            File file; boolean res;
            if (node.equals(readFileButton)
                && (file = fileChooser.showOpenDialog(new Stage())) != null)
            {
                res = canvas.readFromFile(file);
                checkRunAvailability();
                if(res) chooseMsg("Граф загружен.");
            } else if (node.equals(saveFileButton)
                && (file = fileChooser.showSaveDialog(new Stage())) != null)
            {
                res = canvas.writeToFile(file);
                if(res) Message.setMsg("Граф сохранён");
            }
        }
    }

    @SuppressWarnings("unchecked")
    private void heuristicLoadData() {
        heuristics.setValue(new Pair<>(HEURISTIC.NONE, "Выберите
эвристику..."));
    }

```

```

        heuristics.setItems(FXCollections.observableArrayList(
            new Pair<>(HEURISTIC.CHEBYSHEV, "Расстояние Чебышева"),
            new Pair<>(HEURISTIC.MANHATTAN, "Манхэттенская метрика"),
            new Pair<>(HEURISTIC.EUCLID, "Евклидово расстояние"),
            new Pair<>(HEURISTIC.DIJKSTRA, "Нулевая эвристика (Дейкстры)")
        ));
    }
    @SuppressWarnings("unchecked")
    private void actionsLoadData() {
        actions.setValue(new Pair<>(ACTION.NONE, "Выберите действие..."));
        actions.setItems(FXCollections.observableArrayList(
            new Pair<>(ACTION.ADD, "Создать вершину"),
            new Pair<>(ACTION.CONNECT, "Соединить вершины"),
            new Pair<>(ACTION.DELETE, "Удалить вершину/ребро")
        ));
    }

    @FXML
    public void openAboutWindow() {
        try {
            FXMLLoader fxmlLoader = new FXMLLoader();
            fxmlLoader.setLocation(getClass().getResource("about-view.fxml"));
            Scene scene = new Scene(fxmlLoader.load(), 560, 530);
            Stage stage = new Stage();
            stage.setTitle("About");
            stage.getIcons().add(new
Image(Objects.requireNonNull(getClass().getResourceAsStream("icon.png"))));
            stage.setScene(scene);
            stage.show();
        } catch (Exception e) {
            System.out.println("Error while opening the 'about' window");
            e.printStackTrace();
        }
    }

    private <T> StringConverter<Pair<T, String>>
createStringConverter(ChoiceBox<Pair<T, String>> choiceBox){
        return new StringConverter<>() {
            @Override
            public String toString(Pair<T, String> pair) { return
pair.getValue(); }
            @Override
            public Pair<T, String> fromString(String s) {
                return choiceBox.getItems().stream().filter(item ->
                    item.getValue().equals(s)).findFirst().orElse(null);
            }
        };
    }

    private <T> ChangeListener<Pair<T, String>> createChangeListener(T t){
        return (selected, t1, t2) -> {
            if(t instanceof ACTION) {
                setAction((ACTION) selected.getValue().getKey());
                chooseMsgAction();
            }
            else if(t instanceof HEURISTIC) {
                setHeuristic((HEURISTIC) selected.getValue().getKey());
                checkRunAvailability();
                chooseMsg(null);
            }
        };
    }
}

```

```

@FXML
private void onCanvasClick(MouseEvent e) {
    if (e.getSource().equals(canvasPane)) {
        if (e.getButton() == MouseButton.PRIMARY && getAction() ==
ACTION.ADD) {
            canvas.addNode(e.getX(), e.getY());
        }
    }
    checkRunAvailability();

    if(!Message.isError()){
        if(getAction()!=ACTION.ADD) chooseMsgAction();
        else chooseMsg(null);
    }
    else Message.clearError();
}

private void checkRunAvailability(){
    runAlgButton.setDisable(Choice.checkHeuristic(HEURISTIC.NONE) ||
!canvas.hasMinGraph() || State.isAlgRunning());
}

private Pair<Integer, Integer> getStartEndNodes(Graph graph) {
    Integer start = requestNode(1, true);
    Integer end = requestNode(2, false);
    if(graph.getVertex(start) == null || graph.getVertex(end) == null) {
        Message.setError("Паpa указанных вершин не существует");
        return null;
    }
    if (start.equals(end)) {
        Message.setError("Начало не может совпадать с концом");
        return null;
    }
    return new Pair<>(start, end);
}

private int requestNode(Integer defaultNode, boolean startNode){
    TextInputDialog dialog = new
TextInputDialog(String.valueOf(defaultNode));
    dialog.setTitle(null);
    dialog.setContentText(null);
    dialog.setHeaderText("Введите " + (startNode?"стартовую":"конечную") + "
вершину:");

    try{ return
dialog.showAndWait().map(Integer::parseUnsignedInt).orElse(0); }
    catch (NumberFormatException ignored){ }
    return 0;
}

private void chooseMsg(String prev) {
    if(prev != null && !prev.equals("")) prev += " ";
    else prev = "";

    if(canvas.hasMinGraph()){
        Message.setMsg(prev + (
!runAlgButton.isDisabled() ? "Нажмите 'run' для старта
алгоритма" : "Выберете эвристику"));
    }
    else Message.setMsg(prev + "В графе должно быть минимум две вершины");
}

private void chooseMsgAction(){

```



```

        switch (getAction()){
            case ADD -> Message.setMsg("Кликните по полотну, чтобы создать
вершину");
            case CONNECT -> Message.setMsg("Кликните по двум вершинам, чтобы
соединить их");
            case DELETE -> Message.setMsg("Кликните по вершине или ребру, чтобы
удалить объект");
        }
    }

    @FXML
    private void clear(){
        if(!State.isAlgRunning()) {
            canvas.clear();
            checkRunAvailability();
            Message.setMsg("Создайте граф при помощи инструментов или
импортируйте из файла");
        }
    }
}

```

Название файла: *Message.java*

```

package com.example.a_star;

import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.scene.text.Text;

public class Message {
    private static Text msg;
    private static boolean error = false;

    public static Text getMsg() {
        return msg;
    }

    public static void setMsg(String msg){
        setText(msg, Color.BLACK);
        Message.error = false;
    }

    public static void setError(String msg){
        setText(msg, Color.DARKRED);
        Message.error = true;
    }

    public static boolean isError(){ return error; }
    public static void clearError(){ error = false; }

    private static void setText(String msg, Color color) {
        Message.msg.setText(msg);
        Message.msg.setFill(color);
    }

    public static void setTextField(Text info) {
        Message.msg = info;
    }
}

```

Название файла: *Node.java*

```

package com.example.a_star;

```

```

import javafx.scene.control.ContentDisplay;
import javafx.scene.control.Label;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.scene.text.Text;
import javafx.scene.text.TextBoundsType;

public class Node extends StackPane {
    public final static double radius = 15;
    private final int ID;
    private final Label label;

    Node(int id, double x, double y){
        super();
        this.setLayoutX(x-radius);
        this.setLayoutY(y-radius);
        Circle circle = new Circle(x, y, radius);
        circle.setFill(Color.WHEAT);
        Text text = new Text(String.valueOf(id));
        text.setTextBoundsType(TextBoundsType.VISUAL);
        this.getChildren().addAll(circle, text);
        label = new Label();
        label.setLayoutX(x+radius+4);
        label.setLayoutY(y-radius-4);
        label.setStyle("-fx-text-fill: #7a5109;");
        this.ID = id;
    }

    public void setLabel(double f, double h){
        label.setText("h: " + Math.round(h * 100.0) / 100.0 + "\nf: " +
Math.round(f * 100.0) / 100.0);
    }

    public Label getLabel(){ return label; }

    public int id(){ return ID; }
}

```

Название файла: *State.java*

```

package com.example.a_star;

public class State {
    private static boolean algIsRunning = false;
    private static boolean algIsPaused = true;
    private static boolean algFinished = false;
    private static int step = 0;

    public static boolean isAlgRunning() {
        return algIsRunning;
    }

    public static boolean isAlgPaused() {
        return algIsPaused;
    }

    public static int getStep() {
        return step;
    }

    public static void setAlgRun(boolean algIsRunning) {
        State.algIsRunning = algIsRunning;
    }
}

```

```

    public static void toggleAlgPause() {
        algIsPaused = !algIsPaused;
    }

    public static void setStep(int step) {
        State.step = step;
    }

    public static void next(){
        step++;
    }

    public static void prev(){
        step = Math.max(0, --step);
    }

    public static boolean isAlgFinished() {
        return algFinished;
    }

    public static void setAlgFinished(boolean algFinished) {
        State.algFinished = algFinished;
    }

    public static void setAlgPaused(boolean paused) {
        algIsPaused = paused;
    }
}

```

Название файла: *AStarTest.java*

```

package com.example.a_star;

import junit.framework.TestCase;
import org.junit.jupiter.api.*;

import java.io.File;
import java.util.*;

public class AStarTest extends TestCase {

    static ArrayList<GraphForTest> graphList = new ArrayList<>();
    static ArrayList<AStar> resList = new ArrayList<>();
    static int COUNT_TEST = 4;

    @BeforeAll
    public static void readGraphs() {
        GraphForTest graph;
        for (int i = 1; i <= COUNT_TEST; i++) {
            graph = new GraphForTest(new
File(String.format("src/test/java/com/example/a_star/graphs/%s.txt", i));
            graphList.add(graph);
        }
        AStar alg;
        for (GraphForTest gr : graphList) {
            alg = new AStar(gr, gr.getStart(), gr.getEnd(), gr.getHeur());
            resList.add(alg);
        }
    }

    @AfterAll
    public static void clean() {
        graphList = new ArrayList<>();
        resList = new ArrayList<>();
    }
}

```

```

    }
    @Test
    public void testEdgesSteps() {
        int i = 0;
        ArrayList<String> expectedEdgeSteps = new ArrayList<>();
        expectedEdgeSteps.add("[1=3, 3=4]");
        expectedEdgeSteps.add("[1=2, 2=3, 1=5, 5=6, 6=7, 7=8, 8=9]");
        expectedEdgeSteps.add("[1=2, 2=3]");
        expectedEdgeSteps.add("");

        for (AStar graph : resList) {
            String str = graph.getEdgeSteps().toString();
            Assertions.assertEquals(expectedEdgeSteps.get(i++), str,
String.format("Test # %s failed.", i) + " EdgesSteps");
        }
    }

    @Test
    public void testFinalPath() {
        int i = 0;
        ArrayList<String> expectedFinalPath = new ArrayList<>();
        expectedFinalPath.add("[1=3, 3=4]");
        expectedFinalPath.add("[1=5, 5=6, 6=7, 7=8, 8=9]");
        expectedFinalPath.add("");
        expectedFinalPath.add("");

        for (AStar graph : resList) {
            String str = graph.getFinalPath().toString();
            Assertions.assertEquals(expectedFinalPath.get(i++),
str,String.format("Test # %s failed.", i) + " FinalPath");
        }
    }

    @Test
    public void testCountSteps() {
        int i = 0;
        ArrayList<Integer> expectedCountSteps = new ArrayList<>();
        expectedCountSteps.add(2);
        expectedCountSteps.add(7);
        expectedCountSteps.add(2);
        expectedCountSteps.add(0);

        for (AStar graph : resList) {
            Integer n = graph.getCountSteps();
            Assertions.assertEquals(expectedCountSteps.get(i++), n,
String.format("Test # %s failed.", i) + " CountSteps");
        }
    }

    @Test
    public void testPathLen() {
        int i = 0;
        ArrayList<Double> expectedPathLen = new ArrayList<>();
        expectedPathLen.add(8.0);
        expectedPathLen.add(27.0);
        expectedPathLen.add(0.0);
        expectedPathLen.add(0.0);

        for (AStar graph : resList) {
            Double n = graph.getPathLen();
            Assertions.assertEquals(expectedPathLen.get(i++), n,
String.format("Test # %s failed.", i) + " PathLen");
        }
    }

```

```

    }

    @Test
    public void testHeuristics() {
        int i = 0;
        ArrayList<String> expectedHeuristics = new ArrayList<>();
        expectedHeuristics.add("[{2=583.0=580.0, 3=297.0=290.0}, {4=8.0=0.0}]");
        expectedHeuristics.add("[{2=125.0=123.0, 4=273.0=266.0, 5=251.0=245.0}, {3=63.0=59.0}, {}, {6=247.0=236.0}, {7=187.0=173.0}, {8=104.0=82.0}, {9=27.0=0.0}]");
        expectedHeuristics.add("[{2=152.65855435387664=150.65855435387664}, {3=228.69757453074567=224.69757453074567}, {}]");
        expectedHeuristics.add("[{}]");

        for (AStar graph : resList) {
            String str = graph.getHeuristicSteps().toString();
            Assertions.assertEquals(expectedHeuristics.get(i++), str,
                String.format("Test #%%s failed.", i)+" HeuristicSteps");
        }
    }
}

```

Название файла: *GraphForTest.java*

```

package com.example.a_star;

import javafx.util.Pair;

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Scanner;

public class GraphForTest extends Graph {
    private Integer start;
    private Integer end;
    private Choice.HEURISTIC heur;

    public GraphForTest(File file) {
        super();
        try(Scanner sc = new Scanner(file)){
            String data = sc.nextLine();
            int N = Integer.parseInt(data);

            //Pair<Double, Double>[] nodes = new Pair[N];
            //Collection<Pair<Integer, Double>>[] edges = new Collection[N];

            String[] tmp;
            for(int i = 0; i < N; i++){
                tmp = sc.nextLine().split(" ");
                double x = Double.parseDouble(tmp[0]);
                double y = Double.parseDouble(tmp[1]);
                this.addVertex(i+1, new Pair<>(x, y));
            }

            for(int i = 0; i < N; i++){
                tmp = sc.nextLine().split(" ");
                Collection<Pair<Integer, Double>> collection = new
ArrayList<>();
                for(int j = 0; j < N; j++) {
                    double weight = Double.parseDouble(tmp[j]);
                    if (i != j && weight != 0) collection.add(new Pair<>(j + 1,

```

```

weight));
        }
        if(collection.size()>0)
            this.setEdges(i+1, collection);
    }

    //считывание начальной вершины
    data = sc.nextLine();
    start = Integer.parseInt(data);

    //считывание конечной вершины
    data = sc.nextLine();
    end = Integer.parseInt(data);

    //считывание эвристики
    heur = Choice.HEURISTIC.values()[sc.nextInt()];

    } catch (IOException e){
        e.printStackTrace();
    }
}

public Integer getStart() { return start; }
public Integer getEnd() { return end; }
public Choice.HEURISTIC getHeur() { return heur; }
}

```

Название файла: *GraphTest.java*

```

package com.example.a_star;

import junit.framework.TestCase;
import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;

import java.io.File;
import java.util.ArrayList;

public class GraphTest extends TestCase {

    static ArrayList<GraphForTest> graphList = new ArrayList<>();
    static int COUNT_TEST = 4;

    @BeforeAll
    public static void readGraphs() {
        GraphForTest graph;
        for (int i = 1; i <= COUNT_TEST; i++) {
            graph = new GraphForTest(new
File(String.format("./src/test/java/com/example/a_star/graphs/%s.txt", i));
            graphList.add(graph);
        }
    }
    @AfterAll
    public static void clean() {
        graphList = new ArrayList<>();
    }

    @Test
    public void testEdgesInfo() {
        int i = 0;
        ArrayList<String> expectedEdgesInfo = new ArrayList<>();
        expectedEdgesInfo.add("{1=[2=3.0, 3=7.0], 2=[3=5.0], 3=[4=1.0]}");
    }
}

```

```

        expectedEdgesInfo.add("{1=[2=2.0, 4=7.0, 5=6.0], 2=[3=2.0], 4=[9=10.0], 5=[6=5.0], 6=[7=3.0], 7=[8=8.0], 8=[9=5.0]}");
        expectedEdgesInfo.add("{1=[2=2.0], 2=[3=2.0], 3=[1=2.0]}");
        expectedEdgesInfo.add("{2=[1=5.0, 4=8.0], 4=[5=7.0], 5=[2=4.0, 3=10.0]}");

        for (GraphForTest graph : graphList) {
            String actual = graph.getEdgesInfo().toString();
            Assertions.assertEquals(expectedEdgesInfo.get(i++), actual, String.format("Test #%s failed.", i)+" EdgesInfo");
        }
    }

    @Test
    public void testVerticesInfo() {
        int i = 0;
        ArrayList<String> expectedVerticesInfo = new ArrayList<>();
        expectedVerticesInfo.add("{1=479.0=483.0, 2=305.0=307.5, 3=595.0=190.5, 4=885.0=424.5}");
        expectedVerticesInfo.add("{1=425.0=438.0, 2=423.0=372.0, 3=420.0=311.0, 4=317.0=421.0, 5=487.0=430.0, 6=534.0=374.0, 7=526.0=319.0, 8=480.0=274.0, 9=417.0=255.0}");
        expectedVerticesInfo.add("{1=418.0=464.0, 2=383.0=382.0, 3=331.0=443.0, 4=416.0=235.0}");
        expectedVerticesInfo.add("{1=410.0=454.0, 2=350.0=354.0, 3=453.0=354.0, 4=487.0=256.0, 5=311.0=250.0}");

        for (GraphForTest graph : graphList) {
            String actual = graph.getVerticesInfo().toString();
            Assertions.assertEquals(expectedVerticesInfo.get(i++), actual, String.format("Test #%s failed.", i)+" VerticesInfo");
        }
    }
}

```

Название файла: *TestRunner.java*

```

package com.example.a_star;

import org.junit.runner.Result;
import org.junit.runner.JUnit4;
import org.junit.runner.notification.Failure;

public class TestRunner {
    public static void main(String[] args) {
        Result result = JUnit4.runClasses(AStarTest.class, GraphTest.class);
        System.out.println("Total number of tests " + result.getRunCount());
        System.out.println("Total number of tests failed " + result.getFailureCount());

        for (Failure failure : result.getFailures()) {
            System.out.println(failure.toString());
        }
        if (result.wasSuccessful()) System.out.println("All tests passed.");
    }
}

```