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

ОТЧЕТ

по учебной практике

Тема: Кратчайшие пути в графе. Алгоритм Дейкстры

Студент гр. 0304 _____ Аристархов И.Е.

Студентка гр. 0303 _____ Костебелова Е.К.

Студент гр. 0303 _____ Рагрин Д.Р.

Руководитель _____ Тиняков С.А.

Санкт-Петербург
2022

ЗАДАНИЕ

НА УЧЕБНУЮ ПРАКТИКУ

Студент Аристархов И.Е. группы 0304
Студентка Костебелова Е.К. группы 0303
Студент Рагрин Д.Р. группы 0303

Тема практики: Кратчайшие пути в графе. Алгоритм Дейкстры

Задание на практику:

Командная итеративная разработка визуализатора алгоритма на Java с графическим интерфейсом.

Алгоритм: Дейкстра.

Сроки прохождения практики: 29.06.2020 – 12.07.2020

Дата сдачи отчета: 10.07.2020

Дата защиты отчета: 10.07.2020

Студент	Аристархов И.Е.
Студентка	Костебелова Е.К.
Студент	Рагрин Д.Р.
Руководитель	Тиняков С.А.

АННОТАЦИЯ

Задача практики – реализовать графическое приложение, отображающее последовательную работу алгоритма Дейкстры. Перед выполнением были поставлены точные цели и сроки их реализации. После чего выполнялась работа по установленным срокам.

SUMMARY

The task of practice is to implement a graphical application that displays the sequential operation of Dijkstra's algorithm. Before implementation, precise goals and deadlines were set. After that, the work was carried out according to the established deadlines.

СОДЕРЖАНИЕ

	Введение	6
1.	Требования к программе	7
1.1.	Исходные требования к программе	7
1.1.1	Требования к функциональности	7
1.1.2	Требования к интерфейсу	10
1.1.3	Требования к визуализации алгоритма	16
1.1.4	Требования к архитектуре приложения	18
1.1.5	Требования к тестированию	22
1.1.6	Заключение	22
2.	План разработки и распределение ролей в бригаде	24
2.1.	План разработки	24
2.2.	Распределение ролей в бригаде	24
3.	Особенности реализации	26
3.1.	Структуры данных	26
3.1.1	Граф	26
3.1.2	Отображение графа	26
3.1.3	Сохранение графа	27
3.1.4	Алгоритм и разбиение на шаги	27
3.1.5	Интерпретатор шагов	27
3.1.6	Интерфейс	28
3.2.	Основные методы	28
3.2.1	Граф	28
3.2.2	Отображение графа	28
3.2.3	Сохранение графа	29
3.2.4	Алгоритм и разбиение на шаги	29
3.2.5	Интерпретатор шагов	29
3.2.6	Интерфейс	29
4.	Тестирование	31
4.1	Тестирование графического интерфейса	
4.2	Тестирование кода графа	
4.3	Тестирование кода алгоритма	
4.4	Тестирование кода сохранения графа	
	Заключение	
	Список использованных источников	
	Приложение А. Исходный код	

ВВЕДЕНИЕ

Данная практическая работа состоит в реализации графического отображения работы алгоритма Дейкстры. Результат работы – готовое приложение с графическим интерфейсом, позволяющее пользователю удобно настраивать параметры работы алгоритма и наблюдать за каждым этапом его работы.

Алгоритм Дейкстры – это алгоритм поиска кратчайшего пути между двумя вершинами на произвольном графе. Алгоритм проходит по всем ребрам графа, выбирая наиболее выгодный путь.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Исходные Требования к программе

1.1.1. Требования к функциональности

Программа должна выполнять ряд задач.

В первую очередь, программа должна уметь работать с графами любого типа: ориентированными и неориентированными. Будет лишь установлено ограничение в максимальном количестве ребер между любыми двумя вершинами.

В обоих видах графа будут запрещены «петли» - ребра, начало и конец которых совпадают, т.к. их наличие в графе не повлияет на результат работы алгоритма.

Также в ориентированных графах будет разрешено лишь одно ребро между двумя вершинами. Это также не повлияет на результат алгоритма, т.к., при наличии кратных ребер, алгоритм бы выбирал ребро с минимальной стоимостью, что может сделать пользователь вручную в процессе создания графа.

В неориентированных графах можно будет иметь максимум два ребра между любыми двумя вершинами, при условии, что эти ребра будут направлены в противоположные стороны. Опять же, такое сокращение не повлияет на результат алгоритма по тем же причинам, что и в ориентированных графах.

Во-вторых, она должна уметь реализовывать алгоритм Дейкстры. Алгоритм Дейкстры — это алгоритм поиска кратчайшего пути до всех вершин графа от некой фиксированной вершины S. В самом начале алгоритма считается, что расстояние до всех вершин графа бесконечно большое. Выбирается вершина с кратчайшим путём, которая помечается как проверенная с неким расстоянием. После чего, выбирается непроверенная вершина, смежная с любой из проверенных, которая имеет наименьший вес пути от вершины S. Вес считается по формуле $V_{рассматриваемая}=V_{ребра\_между\_вершинами}+V_{текущая}$. Если в процессе поиска обнаруживается более короткий путь, чем ранее найденный, то информация о найденном пути обновляется.

Приложение позволит наглядно продемонстрировать работу алгоритма Дейкстры. Для этого в процессе работы, алгоритм будет разделен на ряд шагов,

которые будут обозначать какие-либо действия в алгоритме, поддающиеся несложной анимации. Переключаться между шагами можно будет с помощью специальных кнопок управления на интерфейсе пользователя. Или же можно будет включить автоматическое воспроизведение шагов, когда шаги будут показываться друг за другом с определенной пользователем в настройках паузой. Также можно приостановить алгоритм с помощью кнопки паузы или отключить показ совсем на кнопку стоп. Пошаговое отображение позволит пользователю остановится в любой момент или же рассматривать алгоритм в обратном порядке.

В-третьих, приложение должно иметь интуитивно понятный интерфейс, позволяющий пользователю удобно пользоваться всеми возможность программы, позволяющий графически отображать граф и все его изменения, позволяющий реализовать некоторое количество инструментов по работе с графом, а именно: кнопки по постройке графа (режим перетаскивания вершин, режим добавления вершины, режим добавления ребра, режим очистки графа), кнопка настройки начальной и конечной точки, для работы алгоритма Дейкстры, кнопки запуска и остановки вычислений алгоритма, кнопки пошагового вывода следующего и предыдущего шага алгоритма Дейкстры, а также позволяющий наглядно видеть все шаги программы в соответствующей текстовой области. В отдельной области экрана должна выводится информация по происходящему на экране. Если пользователь взял инструмент, туда выведется информация о том, для чего предназначен этот инструмент. Во время работы алгоритма, каждый шаг будет в текстовом виде выведен в эту область.

Важным моментом является возможность сохранять граф. Для этого в отдельном месте интерфейса будут располагаться кнопки для обычного сохранения и так называемого «сохранения как», позволяющего настроить путь и название сохраняемого файла. При первом сохранении обе кнопки будут работать, как «сохранить как». Для сохранения графа будет использоваться технология JSON.

Также граф можно будет загружать из файла. Для этого также будет создана кнопка, при нажатии на которую, откроется интерфейс, сходный с проводником, для выбора загружаемого файла.

Помимо этого, должна быть возможность выводить граф на экран в виде ряда вершин (кругов), соединенных ребрами (отрезками). Нужно уметь

выводить как ориентированный граф, так и неориентированный с соответствующими изменениями в виде ребер (отрезки со стрелкой или без). У каждого элемента графа будет своя приписка, обозначающая какую-либо информацию об этом элементе. В случае вершины – это название вершины, в случае ребра – его вес. Кроме того, нужно иметь холст, на котором и будет рисоваться граф. При этом мы должны уметь увеличивать или уменьшать масштаб отображения холста.

В случае загрузки из файла, граф должен добавится на экран с использованием какого-либо правила расстановки вершин.

Также пользователь должен иметь возможность взаимодействовать с графом. Это включает в себя перемещение вершин графа. Нажимая и удерживая ЛКМ на вершине, можно будет переместить мышь, и вершина будет перемещаться с ней; создание ребер и вершин; стирание всего графа; выделение вершин для поиска кратчайшего пути.

Создание вершин подразумевает под собой нажатие по свободному участку экрана, где создастся вершина, под которой можно будет ввести ее название. Название не будет ограничено по размеру, но отображаться будут максимум лишь 4 символа с многоточием после них, если название не поместилось. Наведя на вершину, можно будет посмотреть ее полное название. Перемещение вершин в этом режиме будет разрешено.

Создание ребер будет выполнятся, как выбор пары вершин: начала и конца (в случае неориентированного графа это будет задавать направление). Выбор каждой вершины отображается в виде окантовки круга в определенный цвет. После чего будет нарисовано ребро, посередине которого нужно будет ввести его вес. Вес может быть только целочисленной переменной больше 0 и меньше 2*10^9.

Стирание графа – удаление из экрана всех построенных элементов графа.

Под выделением вершин для алгоритма поиска имеется в виду выбор двух вершин (с графическим подтверждением их выбора, как это было при создании ребра, но уже с другим цветом). Можно будет выбрать максимум 2 вершины. Нажатие на уже выбранную вершину будет снимать выбор.

Все элементы графа должны поддаваться стилистике вне классов самих элементов. Все стили должны указываться в специальном css файле.

1.1.2. Требования к интерфейсу

Интерфейс должен быть интуитивно понятен пользователю, т.е. назначение элементов интерфейса должно быть понятно без дополнительных надписей или с малым их количеством, и удобен в использовании.

Он будет состоять из двух режимов: режима настройки и режима воспроизведения. Режим настройки должен включать в себя набор инструментов, позволяющий настраивать граф перед запуском алгоритма (это включает в себя инструменты создания, перемещения и очистки графа, а также инструмент выбора начальной и конечной вершины последующего поиска кратчайшего пути). В то же время, режим воспроизведения будет содержать следующие инструменты: переход на шаг вперед/назад, запуск автоматического проигрывания шагов, пауза и стоп.

В самом верху программы будет меню с различными вкладками: File, Edit и Help. Вкладка File содержит в себе инструменты для создания нового графа с нуля, для загрузки графа из файла и для сохранения графа в файл, а также кнопку выхода из программы. Вкладка Edit состоит из продублированных инструментов из обоих режимов работы, из функции переключения этих самым режимов и из еще двух кнопок, одна из которых будет открывать окно с логами (куда будет выводится информация о любом взятом пользователем инструменте и информация по каждому проигранному шагу алгоритма), а другая – настройки, в которых можно настроить некоторые параметры отображения графа и стандартный путь сохранения всех графов (он будет автоматически открываться при первом сохранении). Остается вкладка Help, в которой будет кнопка, открывающая репозиторий с исходным кодом, и кнопка, открывающая окно с информацией о самом приложении.

Посмотрим на сам интерфейс:

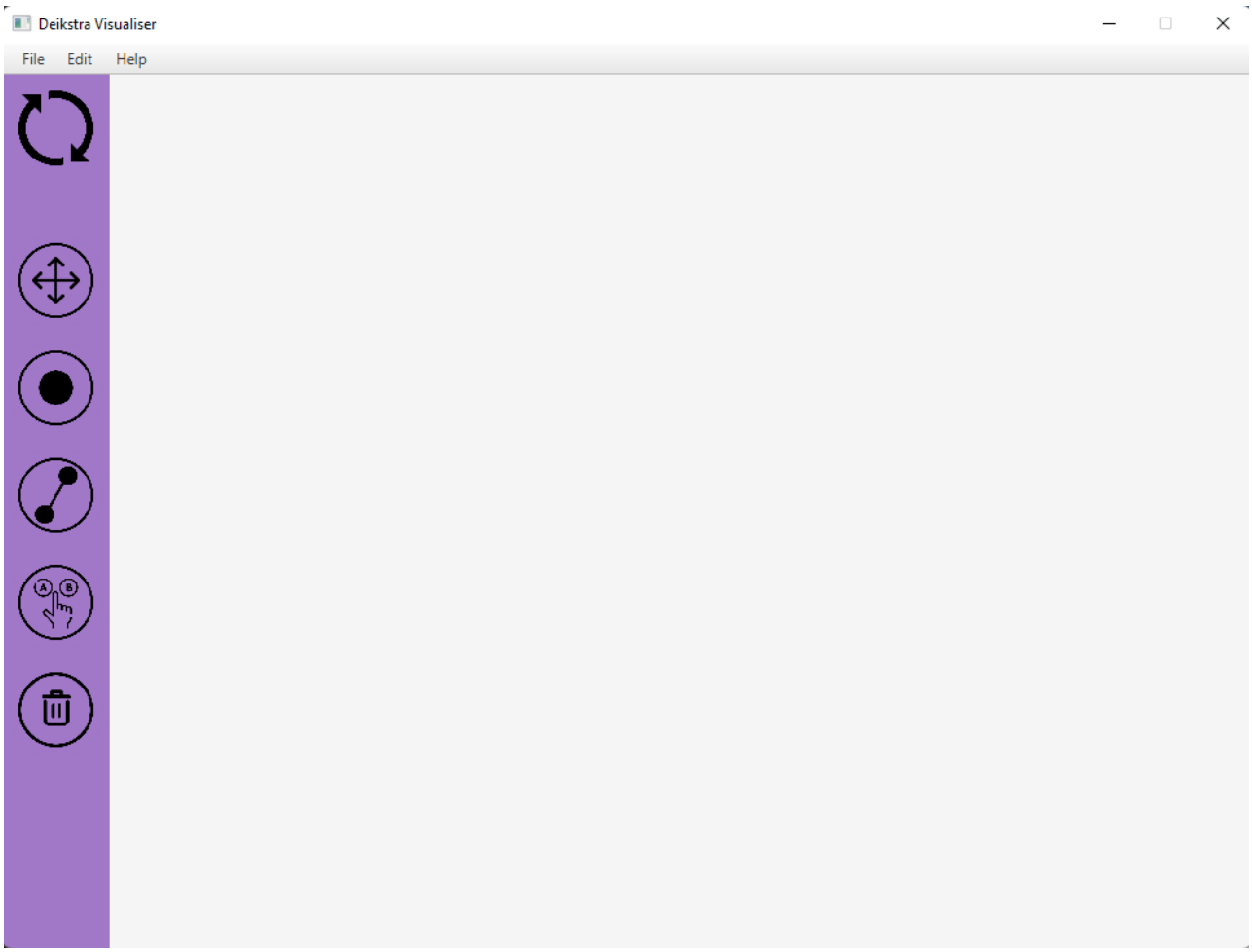


Рисунок 1 – Начальный экран программы.

Запустившись, приложение выводит то, что показано на рис. 1. Это окно можно менять в размерах. Пока что, ни один из инструментов не доступен для пользователя, т.к. для начала ему нужен граф. Он должен перейти во вкладку File в верхнем меню и выбрать подходящий для него метод создания или загрузки графа (см. рис. 2).

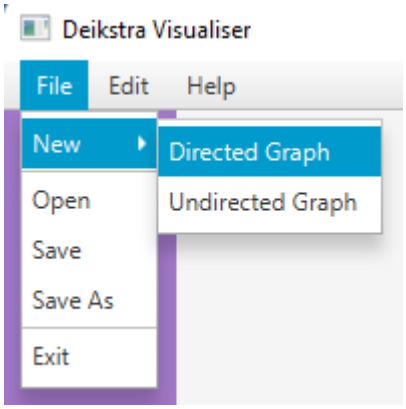


Рисунок 2 – Вкладка File из меню и ее содержимое.

Как видно из рисунка, существует два способа создать граф (создать ориентированный или неориентированный граф) и два способа его сохранить. При первом сохранении кнопка «Save» будет вести себя, как «Save As», т.е. будет открывать новое окно с проводником, спрашивая пользователя, куда он хочет сохранить свой граф. В той же вкладке меня можно выйти из приложения (стандартный способ для всех приложений также работает).

Создав или загрузив граф, пользователю открываются для использования инструменты его настройки. Рассмотрим каждый в отдельности:

Кнопка на рис. 3 отвечает за перемещение вершин графа. Пока она активна каждую вершину можно перетащить вышкой по холсту.

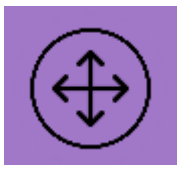


Рисунок 3 – Кнопка перемещения вершин графа

Следующая кнопка на рис. 4 отвечает за создание новых вершин. Активировав ее, можно будет создавать вершины, просто ткнув в любое свободное от элементов графа место на холсте.

Рисунок 4 – Кнопка создания новых вершин

Далее идет кнопка создания новых ребер. Она расположена на рис. 5. С ее помощью, можно будет выделить 2 вершины, между которыми образуется новое ребро.

Рисунок 5 – Кнопка создания новых ребер

Важной кнопкой является кнопка выбора двух вершин для последующей работы алгоритма. Это кнопка на рис. 6.

Рисунок 6 – Кнопка выбора вершин для алгоритма

И остается кнопка, которая полностью стирает весь граф с экрана. Она на рис. 7.

Рисунок 7 – Кнопка стирания всего графа

На самом деле осталась еще одна кнопка в самом верхнем левом углу (см. рис. 8). Но она занимается не настройкой графа, а переключением режимов. Такая кнопка есть в обоих режимах.

Рисунок 8 – Кнопка переключения режимов

Нажмем на нее и посмотрим, что находится в другом режиме (см. рис. 9). Как видно из рисунка, панель слева убралась и появилась панель сверху. Эта панель уже имеет другой окрас и другой набор инструментов. Это мы переключились в режим воспроизведения.

В этом режиме нам доступны инструменты управления воспроизведением, такие как: переход на шаг назад или вперед, старт воспроизведения, пауза и стоп.

Если в предыдущем режиме не были выбраны вершины для алгоритма, то все инструменты этого режима будут заблокированы. В ином же случае, после переключения будет доступна для нажатия кнопка запуска, как бы показывая, что можно начать показ шагов. При нажатии на эту кнопку, запустится автоматическое воспроизведение шагов с небольшой, но достаточной для принятия решения приостановки показа, паузой и разблокируются все остальные кнопки. Приостановку показа можно сделать, нажав на кнопку паузы. Если автоматическое воспроизведение приостановлено, то шагами можно управлять кнопками налево и направо, перемещаясь по одному шагу назад и вперед соответственно. Или же можно нажать на эти кнопки

самостоятельно, и тогда программа сама отключит автоматический показ шагов. Если дальнейший просмотр алгоритма не нужен или хочется быстро вернутся в его начало, то можно полностью остановить показ на кнопку стоп.

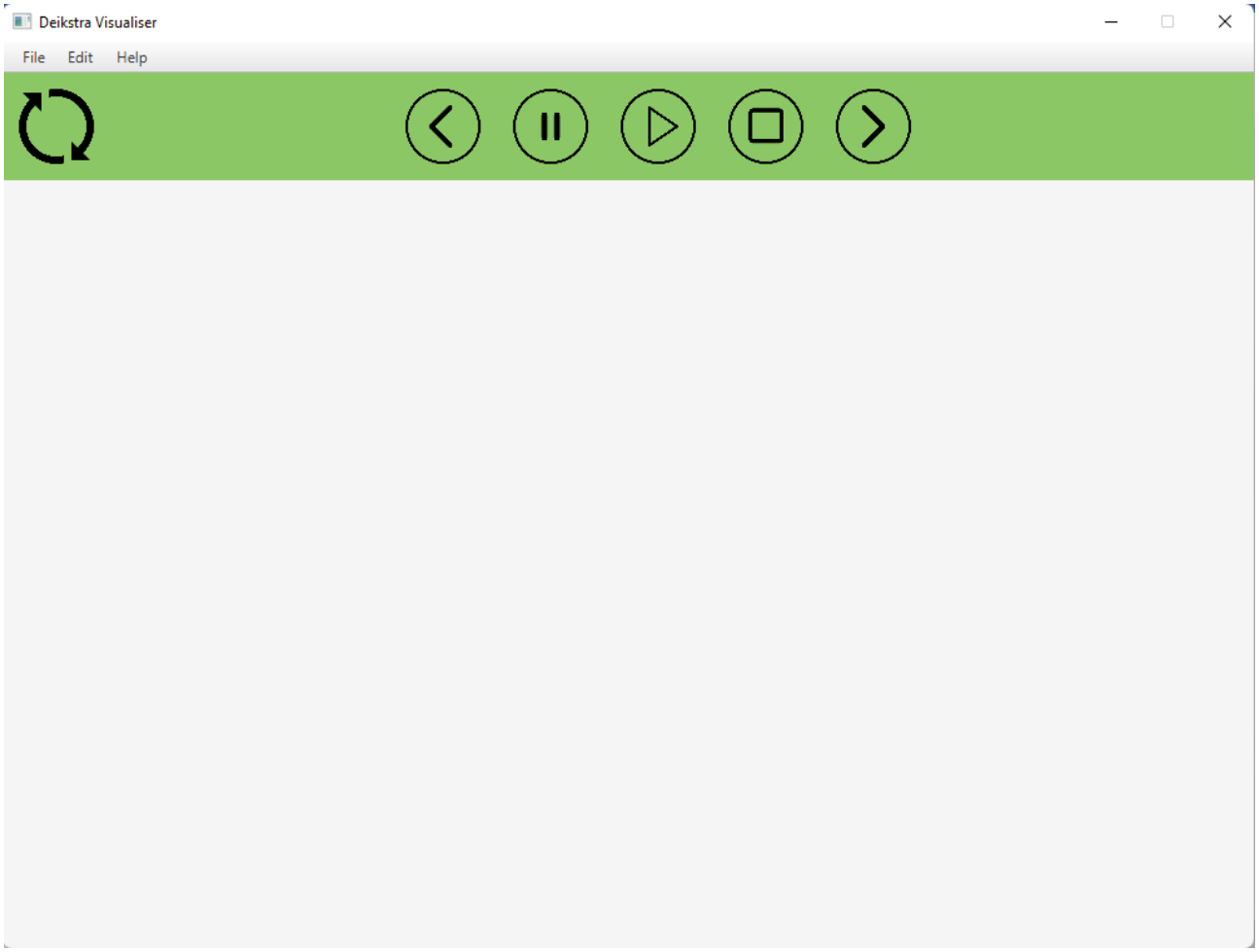


Рисунок 9 – Режим воспроизведения

Теперь вернемся ко вкладкам панели меню. Во вкладке Edit (см. рис. 10) располагаются кнопка переключения режимов, два списка с инструментами режимов (доступность этих инструментов соответствует описанному выше), кнопка, открывающая логи (см. рис. 11), и кнопка, открывающая настройки приложения (см. рис. 12), в которых можно изменить некоторые параметры отображения графа и стандартный путь сохранения.

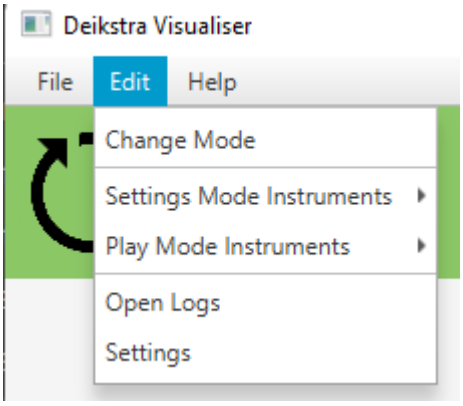


Рисунок 10 – Вкладка Edit

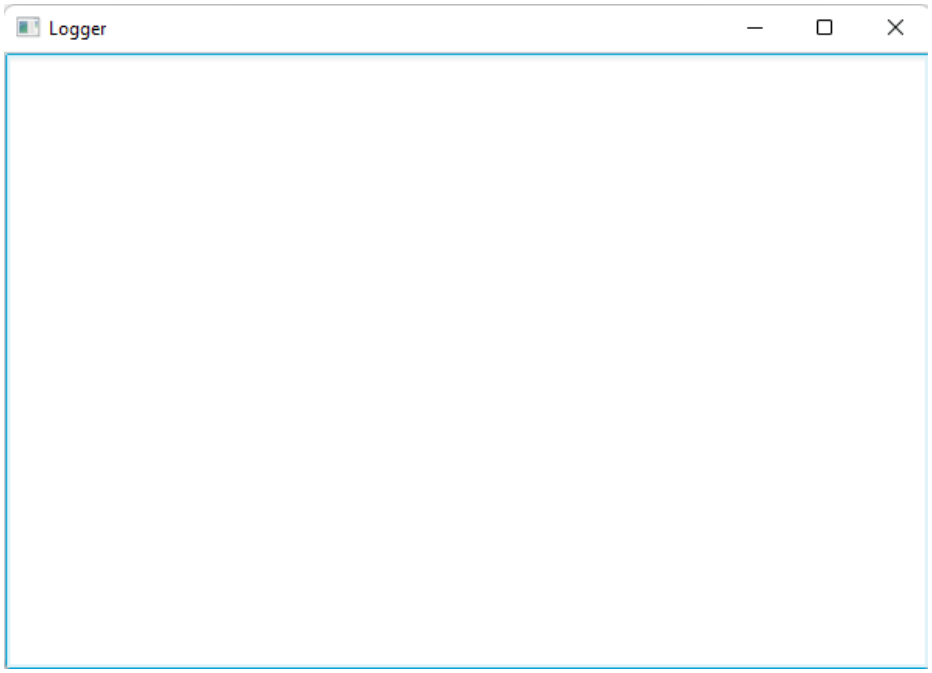


Рисунок 11 – Окно логгера

Рисунок 12 – Окно настроек

И остается последняя вкладка Help (см. рис. 13), в которой традиционно располагается кнопка, открывающая окно с информацией о программе (см. рис. 14), и кнопка, отсылающая на репозиторий проекта.

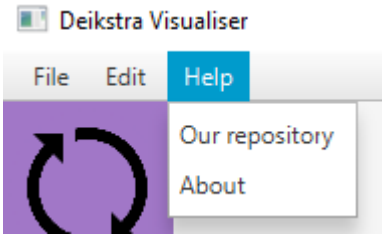


Рисунок 13 – Вкладка Help

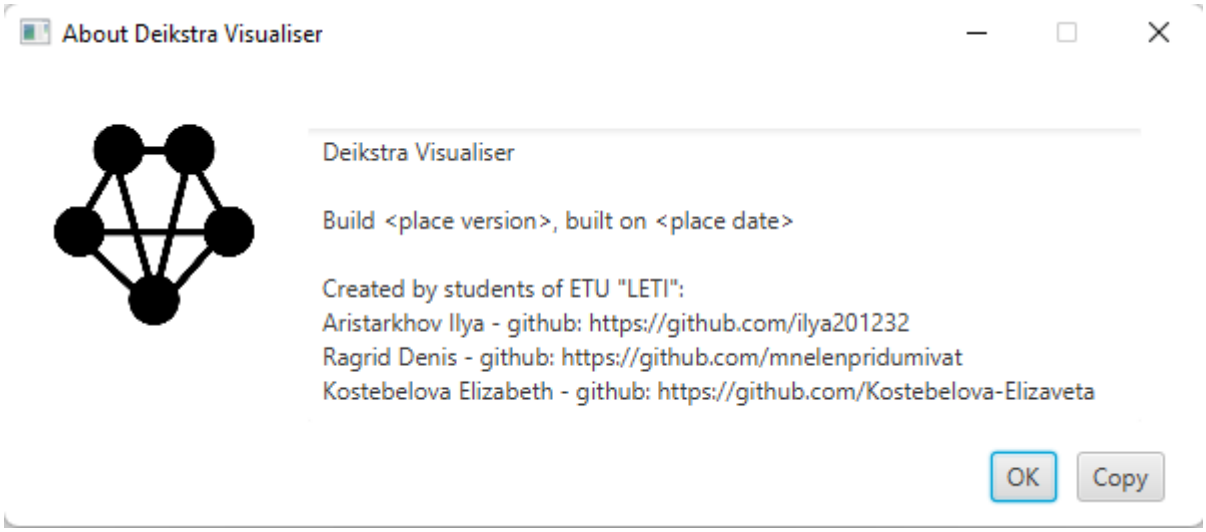


Рисунок 14 – Окно с информацией о программе

1.1.3. Требования к визуализации алгоритма

Работа алгоритма должна разделятся на логические части, в каждой из которых будет сохранятся состояние алгоритма в отдельный список. Далее, используя этот список, визуализатор будет пошагово (управление переключением шагов остается за пользователем) выводить работу алгоритма.

Будут выделены следующие выводимые этапы алгоритма:

- Проверка ребёнка у узла:
  - Для каждого узла, исходящего из текущего, проверяется нахождение нового более дешёвого пути.
  - Визуализация: цветом выделяется окантовка проверяемой вершины и ребро до него.
- Обновление информации у ребёнка:
  - В случае нахождения более дешёвого пути до ребра в ней обновляется информация, такая как наименьший вес пути до неё и узел, из которого можно до неё добраться с наименьшим весом.
  - Визуализация: полная перекраска вершины определённым цветом.
- Переход к узлу:

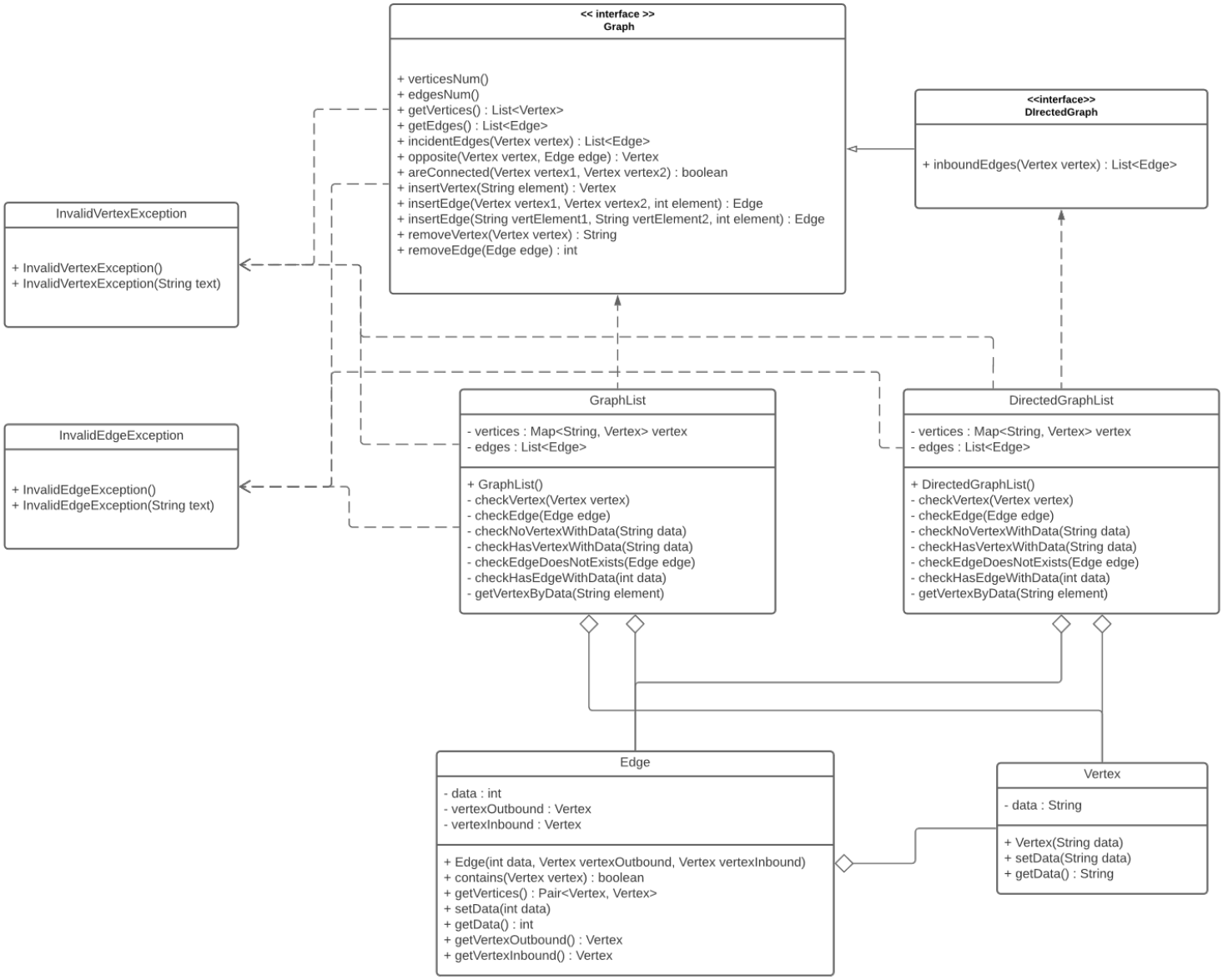
- Из вершин из очереди на обработку, выбирается новая текущая вершина с наименьшим весом
- Визуализация: полная отрисовка пути с нуля.

Шагом в работе алгоритма будет считаться выполнение любого из перечисленного этапа.

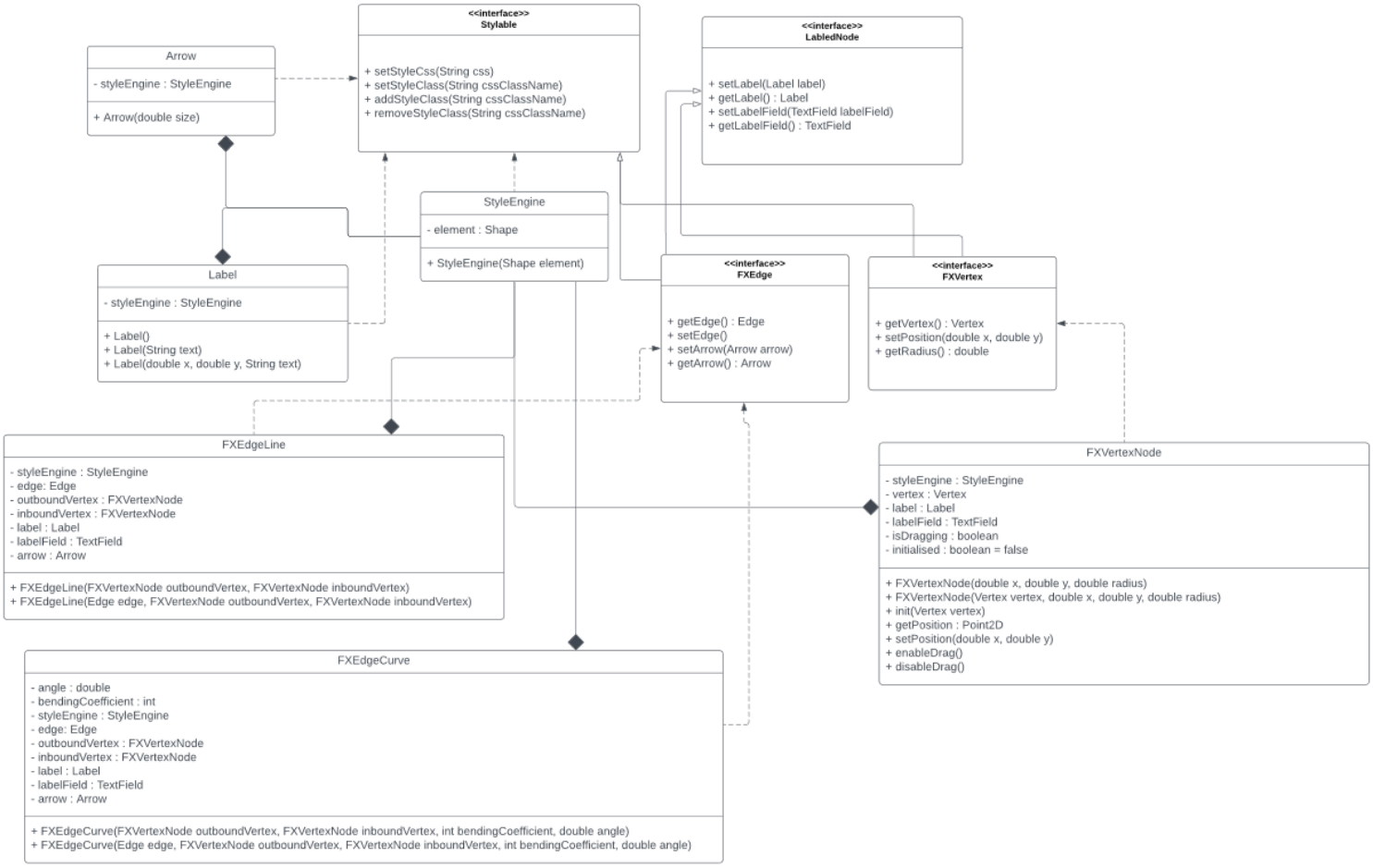
При запуске алгоритма на каждом шаге будет сохраняться информация о текущем состоянии графа и очереди вершин. После завершения работы алгоритма класс пошагового отображения сможет получить информацию о пройденных шагах, чтобы отобразить их в графическом интерфейсе с возможностью переключения между шагами, причём как вперёд, так и назад. Все пути будут анимированы (ребра будут перекрашиваться постепенно друг за другом с определенной задержкой).

1.1.4. Требования к архитектуре приложения

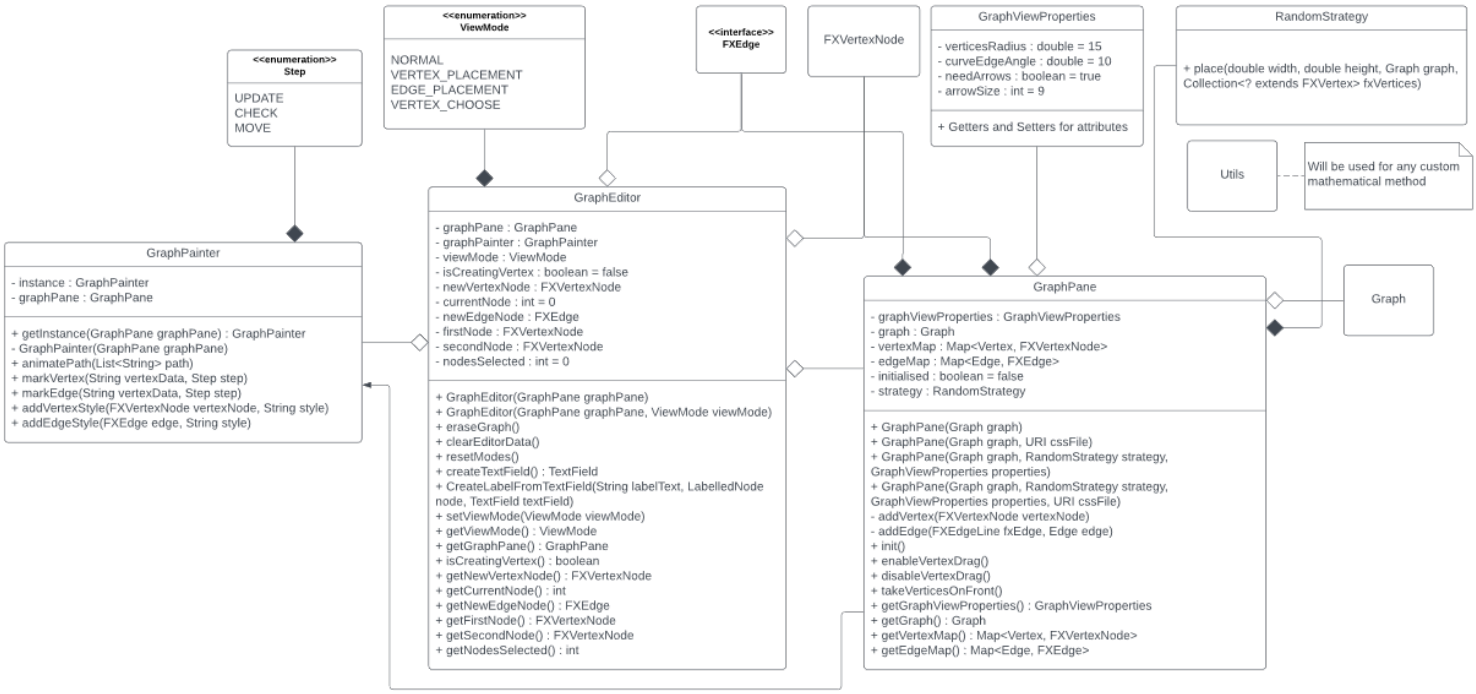
UML-схема базовых классов для хранения и работы с графами различного типа:



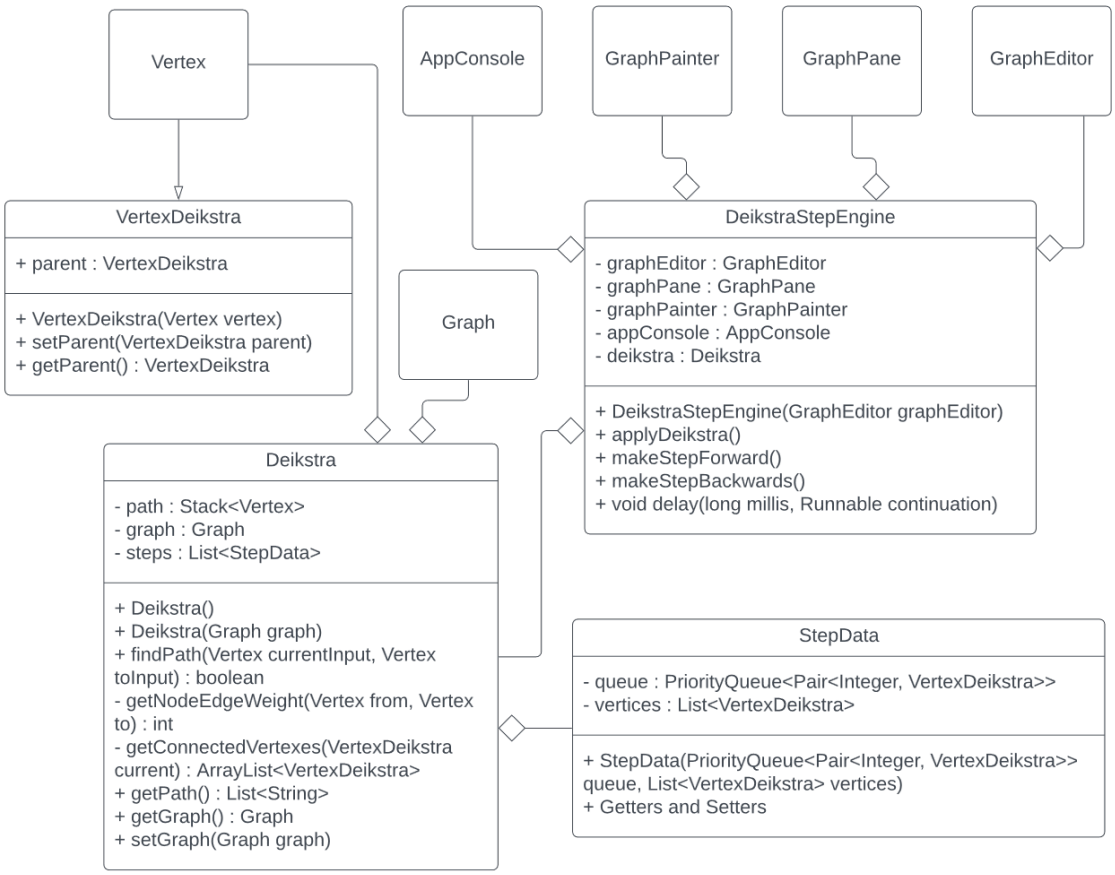
Базовые классы для работы с графической составляющей создания графа:



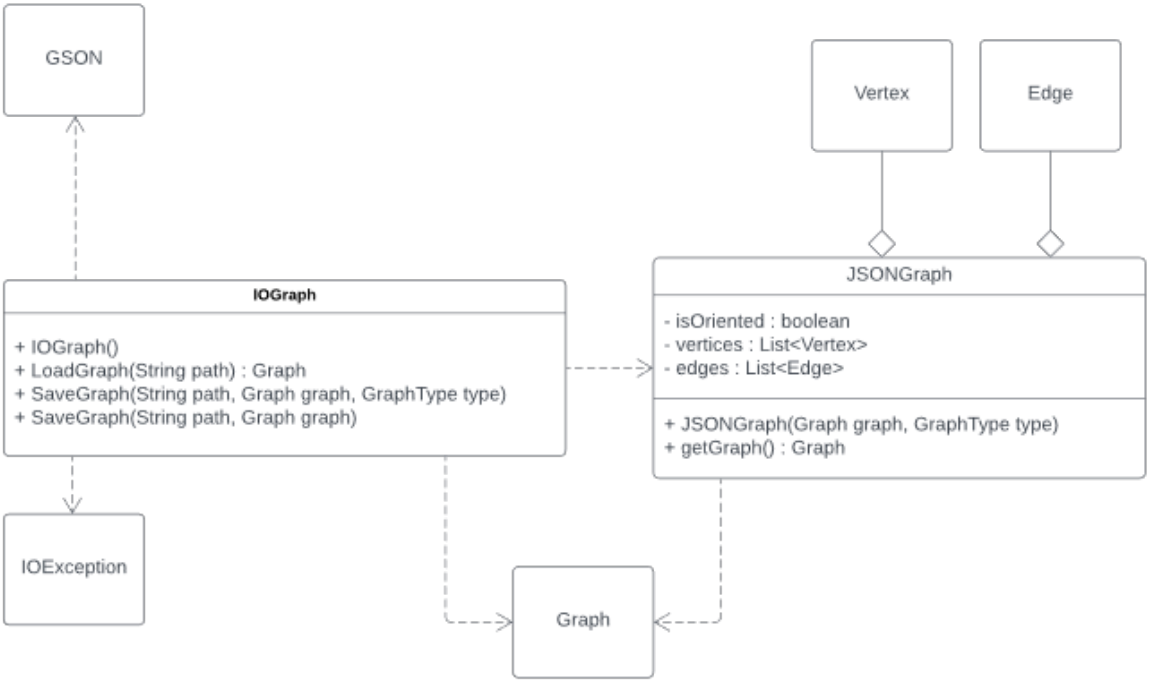
Классы отрисовки:



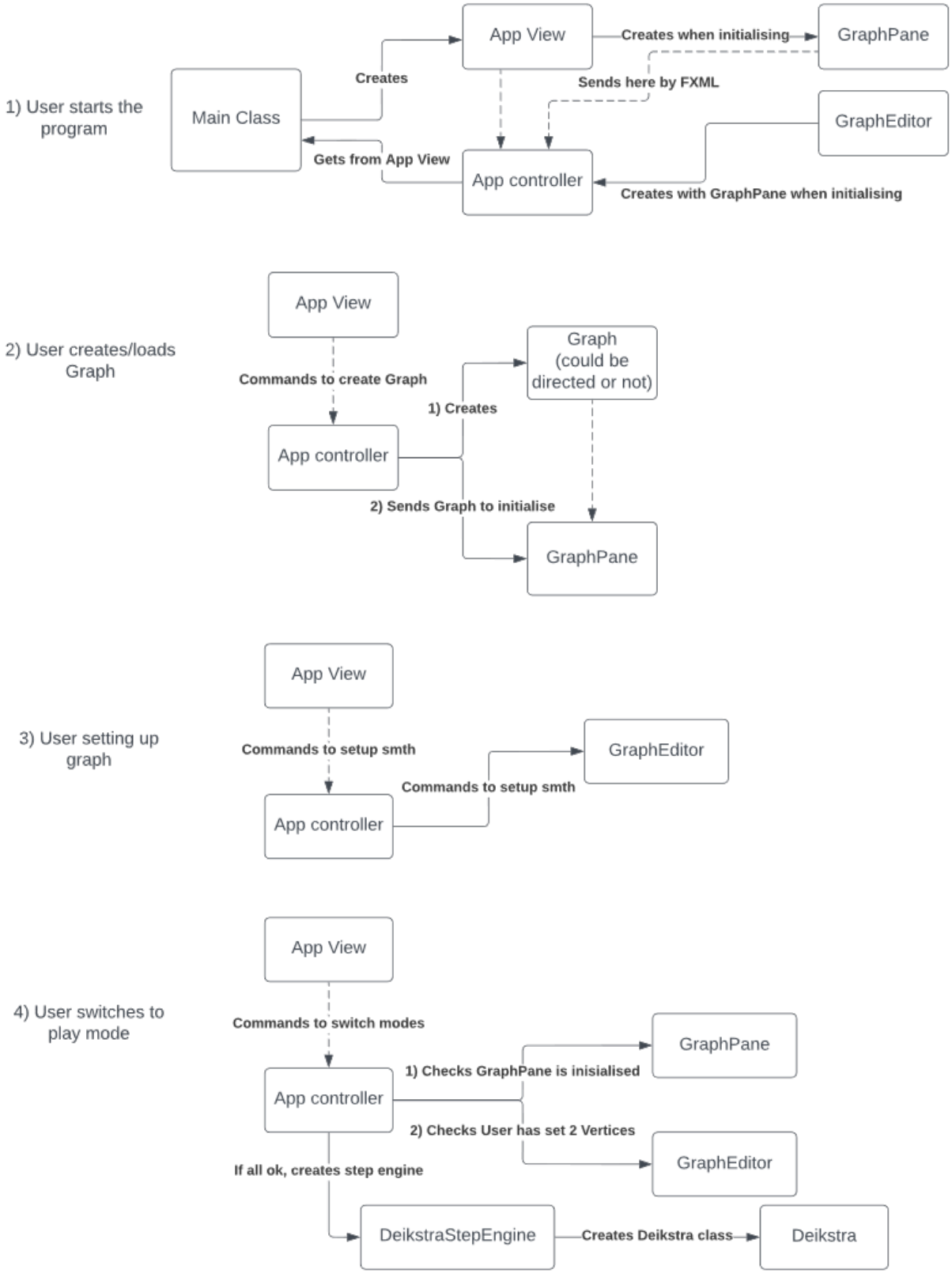
Классы управления алгоритмом Дейкстры:

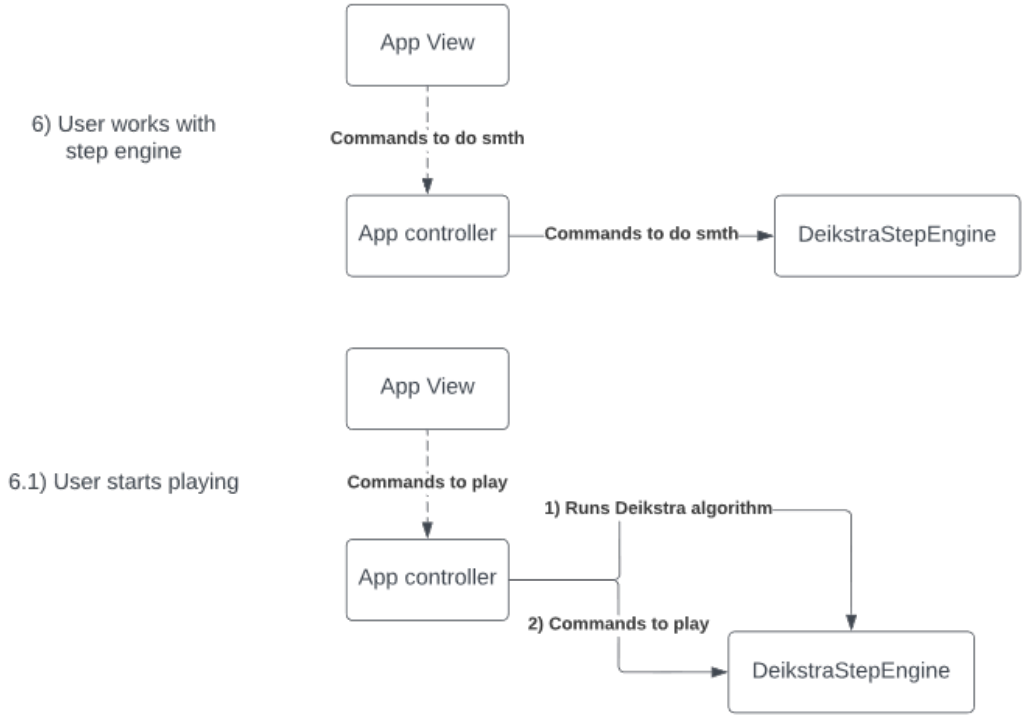


Классы для сохранения и загрузки графа:



Архитектура GUI:





1.1.5. Требования к тестированию

Тестирование приложения будет разбито на 2 вида: автоматическое тестирование и ручное. Автоматическое тестирование (далее Unit-тесты) будет реализовано при помощи библиотеки JUnit и будет применятся к методам и классам, в которых подобное тестирование будет возможно. Это все классы, не связанные с реализацией графического интерфейса. В остальных случаях будет применено ручное тестирование.

Unit-тесты планируется написать до непосредственной реализации методов для последующего ускорения разработки. Ручное тестирование будет производится после создания элементов интерфейса.

1.1.6. Заключение

Подводя итоги вышенаписанного, приходим к тому, что программа должна уметь выполнять алгоритм Дейкстры, должна уметь выводить его пошагово на экран с определенными анимациями шагов на построенном ранее графе. Шагами можно управлять кнопками переключения шагов или кнопкой автоматического воспроизведения, паузы и полной остановки. Остановить показ можно кнопкой стоп.

Граф можно построить вручную с помощью инструментов создания вершин и ребер, очистки графа (предварительно создав новый граф) или загрузить ранее сохраненный вариант. Граф можно будет увеличивать или

уменьшать в размере. На построенном графе можно будет выбрать начальную и конечную точки алгоритма.

Пользователю будет доступно окно с различными настройками программы, окно с выводом текстовой информации о происходящем на экране (логгер) и окно с информацией о программе.

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

Предполагается следующий план разработки:

- Реализация структуры графа: до 2.07.2022
- Реализация алгоритма, основываясь на реализованной структуре графа: до 3.07.2022
- Реализация разделения алгоритма на составные шаги: до 3.07.2022
- Реализация вывода графа на экран: до 4.07.2022
- Реализация системы вывода сообщений: до 6.07.2022
- Реализация вывода шагов алгоритма на экран: до 6.07.2022
- Реализация первой версии интерфейса: до 6.07.2022
- Реализация сохранения и загрузки графа: до 6.07.2022
- Реализация логики работы окна настроек: до 8.07.2022
- Исправление возникших проблем: до 9.07.2022

2.2. Распределение ролей в бригаде

Костебелова Е.К.:

- Проектирование и реализация графического интерфейса пользователя (за исключением вывода графа на экран).
- Создание системы вывода сообщений из любого класса в специально отведенное место в интерфейсе.
- Реализация работы окна настроек.
- Ручное тестирование реализованных частей интерфейса (включая графическое отображение графа).

Рагрин Д.Р.

- Реализация алгоритма Дейкстры.
- Разбиение реализованного алгоритма Дейкстры на логические части для возможности его последующего пошагового отображения в графическом интерфейсе.
- Реализация возможности сохранения и загрузки созданного графа с использованием технологии JSON.

- Написание тестов к своей части работы

Аристархов И.Е.:

- Создание базовых классов для хранения ориентированного и неориентированного типов графа.
- Создание классов для графического отображения графа
- Реализация графической части отображения алгоритма Дейкстры (включает в себя анимации построения пути и реализацию методов пошагового отображения из данных разбиения работы алгоритма)
- Написание тестов к своей части работы

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Структуры данных

3.1.1. Граф

Структура графа не представляет из себя ничего сложного. Это объект, который хранит в себе в отдельных списках вершины и ребра. Вершиной будет считаться объект, хранящий в себе только свои данные, а ребром – объект, хранящий данные и начальную и конечную вершины (в случае неориентированного графа эти вершины равнозначны). Граф предоставляет ряд методов по управлению им (см. п. 3.2.1).

3.1.2. Отображение графа

Для отображения графа требуются:

- Вершины. Их отображение реализовано через класс, который наследуется от класса круга. Дополнительно наш класс может хранить в себе ссылки на отображения своих данных и на текстовое поле ввода, которое используется при создании вершины пользователем.
- Ребра. Они бывают двух видов: прямые и кривые. Оба вида реализуются тем же способом, что и вершины, только в случае прямых мы наследуемся от прямой, а в случае кривой – от параметрической кривой с одной контрольной точкой. Также, если требуются стрелки, то они будут хранится здесь.
- Стрелки. Этот объект создан, как две наклонные линии по 45 градусов, соединенные таким образом, чтобы они образовывали стрелку, смотрящую направо. В дальнейшем с помощью функции atan2 ребра разворачивают стрелку так, как им надо.
- Подписи. Это просто класс, наследующийся от класса текста. Если в нем создать слишком большую надпись, то он создаст для себя ToolTip, который будет показываться при наведении.

Кроме того, все вышеописанные классы реализуют интерфейс, позволяющий более просто настраивать стили. Для всех классов реализация этого интерфейса одинаковая, т.ч. был создан класс StyleEngine, который и реализует нужные методы.

Для отображения всех вышеописанных элементов существует класс GraphPane, который наследуется от Pane и имплементирует интерфейс для стилей (StyleEngine в нем также хранится). После создания, элементы графа добавляются на GraphPane, как его дети. Также GraphPane хранит в себе сам граф и хранит все установленные на него вершины и ребра, соотнося их с реальными элементами графа.

Для редактирования графа существует класс GraphEditor, который работает с GraphPane и может манипулировать объектами в нем.

3.1.3. Сохранение графа

Для сохранения графа использовалась сторонняя библиотека GSON, т.ч. для сохранения требовалось только создать промежуточный класс для сохранения/загрузки графа и класс, который будет работать с этим промежуточным форматом графа и библиотекой.

3.1.4. Алгоритм и разбиение на шаги

Для работы с алгоритмом был создан класс Dijkstra, который хранит очередь с приоритетом для самого алгоритма, найденный путь и лист для хранения данных выделенных шагов в алгоритме.

Для алгоритма пришлось переопределить понятие вершины, чтобы она сохраняла своего предка. Также она сохраняет объект оригинальной вершины, из которой была образована.

Для хранения данных шагов также был создан класс. Он хранит тип шага, вершину из которой шаг начинался, вершину с которой шаг в итоге работал и вес пути.

3.1.5. Интерпретатор шагов

Интерпретатор шагов представляет из себя класс, который хранит в себе объект класса Dijkstra, может его вызвать и сформировать список действий по отображению созданных алгоритмом шагов. Эти действия интерпретатор хранит в специальном объекте, который предназначен для работы с двумя потоками. Также интерпретатор хранит множество списков с данными, которые использовали действия всех показанных шагов. Эти списки нужны для переключения шагов назад.

Исключительно для интерпретатора также был создан класс GraphPainter для графических взаимодействий с уже построенным графом.

3.1.6. Интерфейс

Интерфейс в программе нарисован с помощью программы Scene Builder и хранится в виде fxml файлов. Были нарисованы интерфейсы для окон настроек, логгера, окна about и основного окна приложения. Каждый интерфейс имеет свой контроллер, который хранит разные данные, зависящие от самого интерфейса. Но каждое из них хранит в себе некоторые объекты, расположенные на самом интерфейсе, чтобы в дальнейшем работать с ними.

Отдельного внимания требует логгер, т.к. ему помимо контроллера понадобился класс, реализующий singleton паттерн. Он хранит в себе объект контроллера окна логгера и позволяет любому классу получить себя для вывода сообщений в логгер.

3.2. Основные методы

3.2.1. Граф

В структуре графа существует множество методов, задача которых позволить работать с самим графом и его элементами. Так, в нем есть методы, позволяющие:

- Получить все вершины
- Получить все ребра
- Получить все ребра, смежные с определенной вершиной
- Получить ребро, соединяющее одну вершину с другой
- Добавить вершину
- Добавить ребро

И так далее…

3.2.2. Отображение графа

Основные элементы отображения графа содержат методы по настройке их вида и методы получения и установки для тех или иных параметров, которые можно хранить в этих объектах. Так, вершине можно указать или получить ее радиус и объект вершины из графа. Ребру можно установить или получить стрелку и сам объект ребра графа. На оба этих объекта можно добавить тестовые подписи через соответствующие методы.

GraphPane содержит методы по загрузке готового графа, по очистке всего поля, по добавлению и удалению графических объектов и еще несколько вспомогательных методов.

У GraphEditor важно отметить метод setViewMode, который позволяет задать то, как программа будет вести себя при нажатии левой кнопкой мыши по GraphPane. Также он реализует метод очистки графа, который свои очищает внутренние данные и GraphPane.

3.2.3. Сохранение графа

Промежуточный класс графа имеет функцию получения нормального класса графа из своих данных. Класс, который переводит промежуточный класс в json и обратно, имеет соответствующие методы.

3.2.4. Алгоритм и разбиение на шаги

Класс Dijkstra имеет метод поиска кратчайшего пути, который выдает true или false в зависимости от того, нашел он путь или нет. Найденный путь можно получить отдельным методом. Также можно получить список выделенных алгоритмом шагов.

3.2.5. Интерпретатор шагов

Среди методов интерпретатора важно выделить метод applyDijkstra(), который выполняет поиск пути, забирает полученные шаги, а затем формирует список действий, требующихся для графического отображения каждого шага.

Также важны методы:

- Авто воспроизведения, когда 2 потока по очереди с определенной паузой выполняют шаги.
- Переход на шаг вперед, когда очередной шаг вручную выполняется.
- Переход назад, когда восстанавливается предыдущий шаг.

У graphPainter важен метод последовательной отрисовки пути до вершины, метод окрашивания вершины и метод окрашивания ребра (оба имеют разные параметры окраски).

3.2.6. Интерфейс

Интерфейс программы, как было сказано ранее, собран из 4 различных окон, каждое из которых имеет свой контроллер со своими методами.

Так, контроллер настроек имеет методы по добавлению в него данных и по отправке их в контроллер главного экрана. Контроллер логгера имеет метод, записывающий определенную информацию в логгер. Контроллер окна about имеет метод, позволяющий копировать текст в буфер обмена.

Самый важный контролер – это контроллер главного экрана. В нем расположены все управляющие методы всех кнопок интерфейса. В нем же

реализуется эффект увеличения и уменьшения. Через него работает окно настроек (метод применения настроек).

4. ТЕСТИРОВАНИЕ

4.1. Тестирование графического интерфейса

Было произведено ручное тестирование на двух операционных системах (Linux и Windows) всех элементов интерфейса, а именно:

- Режима изменения графа
- Режима воспроизведения алгоритма Дейкстры
- Панели инструментов (меню File, Edit, Help)
- Инструментов сохранения и загрузки
- Окна настроек
- Окна логирования

Во время тестирования **режима изменения графа** была проверена корректная работоспособность каждого инструмента: перетаскивание вершин графа, создание вершины, создание ребра, выбор начальной и конечной вершины для Алгоритма Дейкстры, очистка графа. Инструменты работают исправно, конфликта не возникает. При выборе инструмента, который отвечает редактирование графа, графически подсвечивается какая кнопка была выбрана пользователем, при этом невозможно выбрать одновременно два режима, только один.

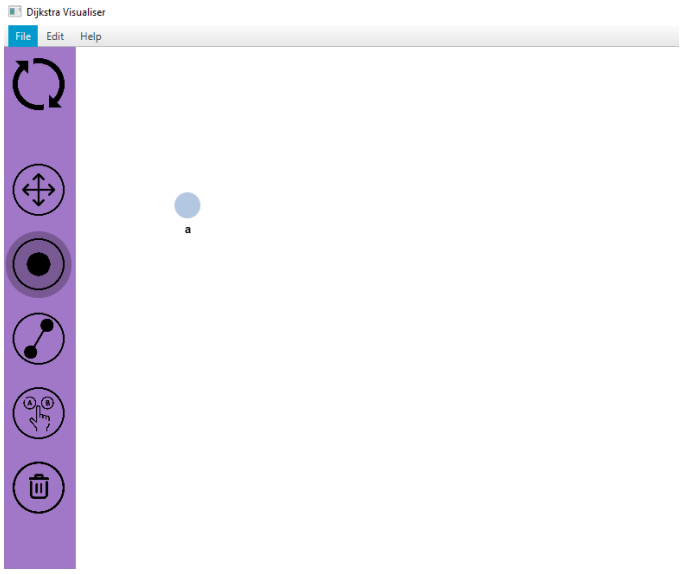


Рисунок 15 — Создание вершины

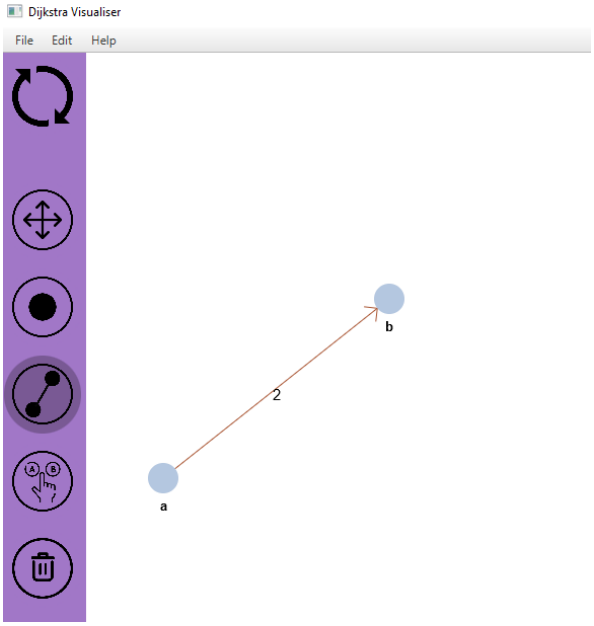


Рисунок 16 — Изменение позиции вершины и создание ребра между вершинами

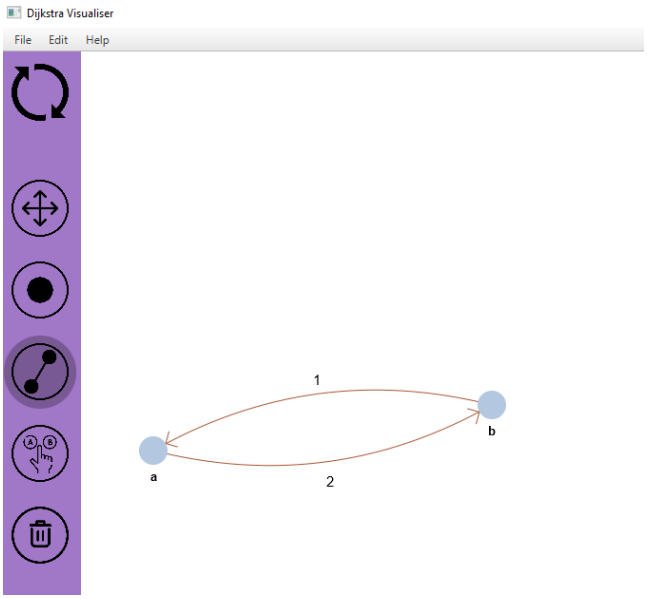


Рисунок 17 — Построение двух рёбер между вершинами в ориентированном графе

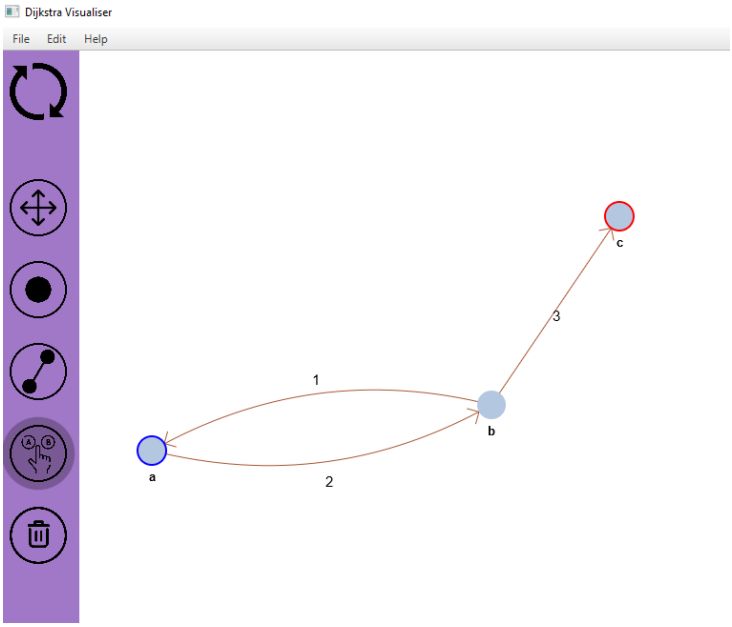


Рисунок 18 — Выделение начальной и конечной вершины

Во время тестирования **режима воспроизведения** также была проверена работоспособность каждого инструмента: следующий/предыдущий шаг, пауза, старт, стоп. Если пользователь не выбрал начальную и/или конченые вершину – инструменты по воспроизведению остаются недоступными для пользователя, аналогичный запрет распространяется на меню Edit в верхней панели инструментов. Были протестированы ситуации, когда пользователь, во время проигрывания алгоритма Дейкстры, начинает менять настройки или как-либо взаимодействовать с инструментами, которые ему доступны, никаких конфликтов не возникает, в случае переключения на режим редактирования графа или изменения каких-либо настроек – воспроизведение останавливается, пользователю необходимо заново его начать, так как он изменил какие-либо настройки. В случае, когда настройки не были изменены (окно настроек было открыто, однако изменений не последовало) – алгоритм продолжает воспроизводиться. Открытие окна логирования никак не влияет на проигрывание, в нём наоборот отражается каждый шаг и что на нём произошло.

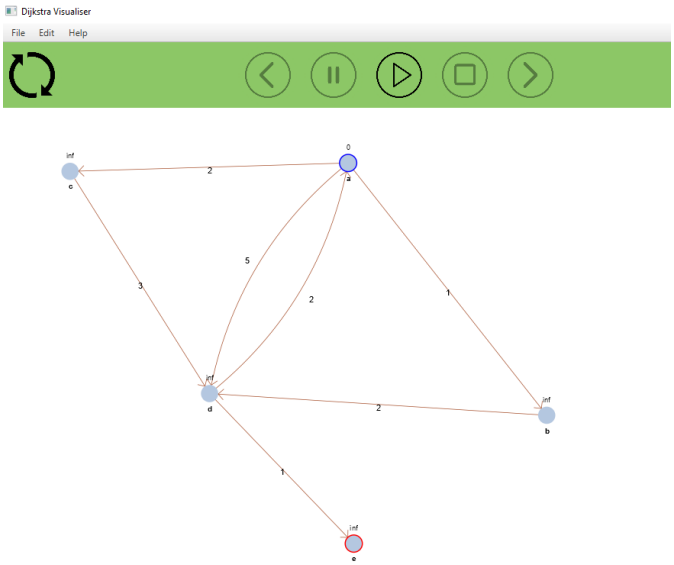


Рисунок 19 — Начало воспроизведения алгоритма

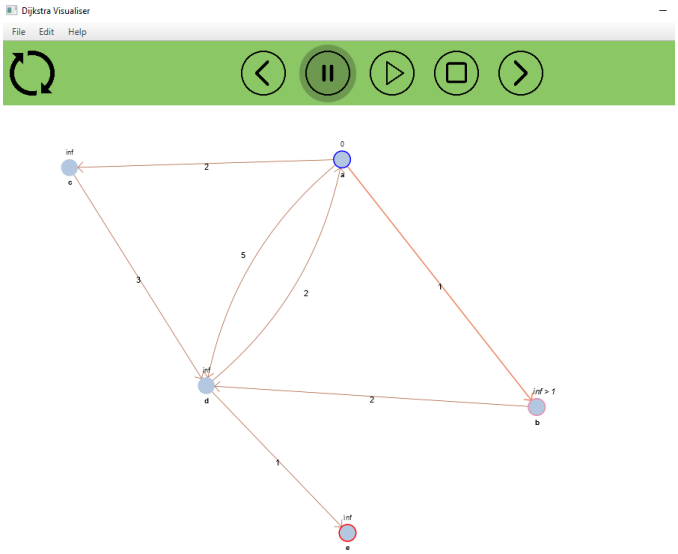


Рисунок 20 — Проверка вершины при воспроизведении

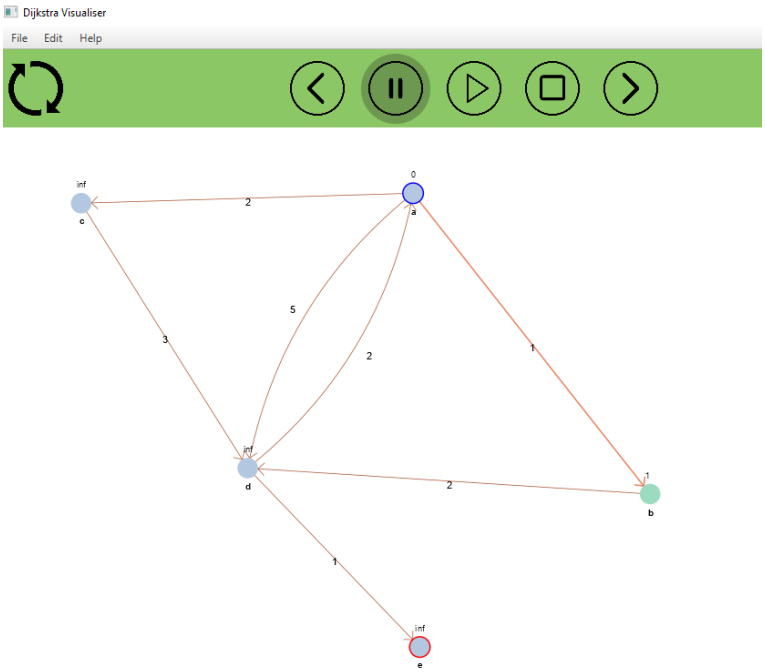


Рисунок 21 — Обновление информации о вершине при воспроизведении

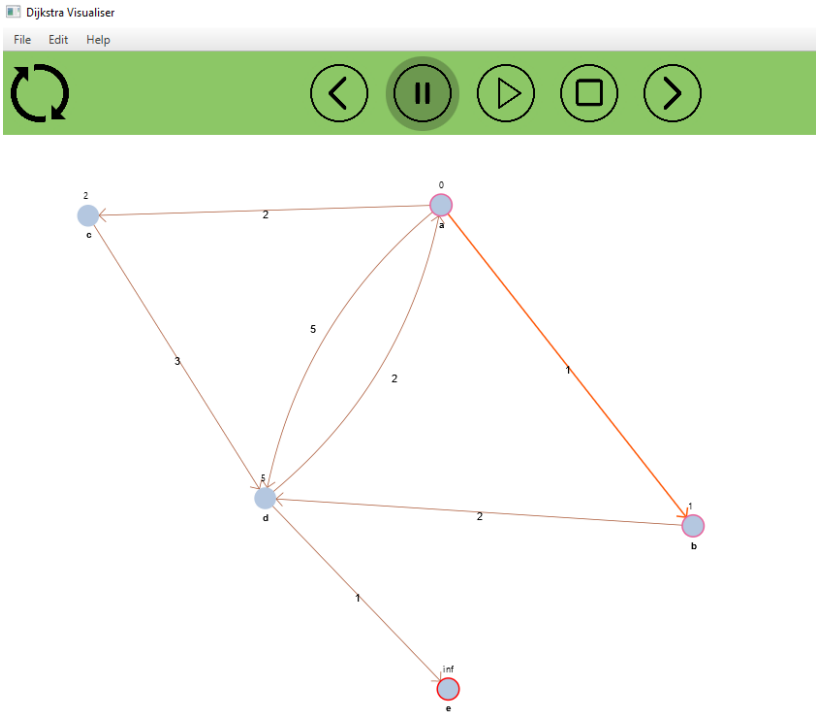


Рисунок 22 — Переход к следующей вершине

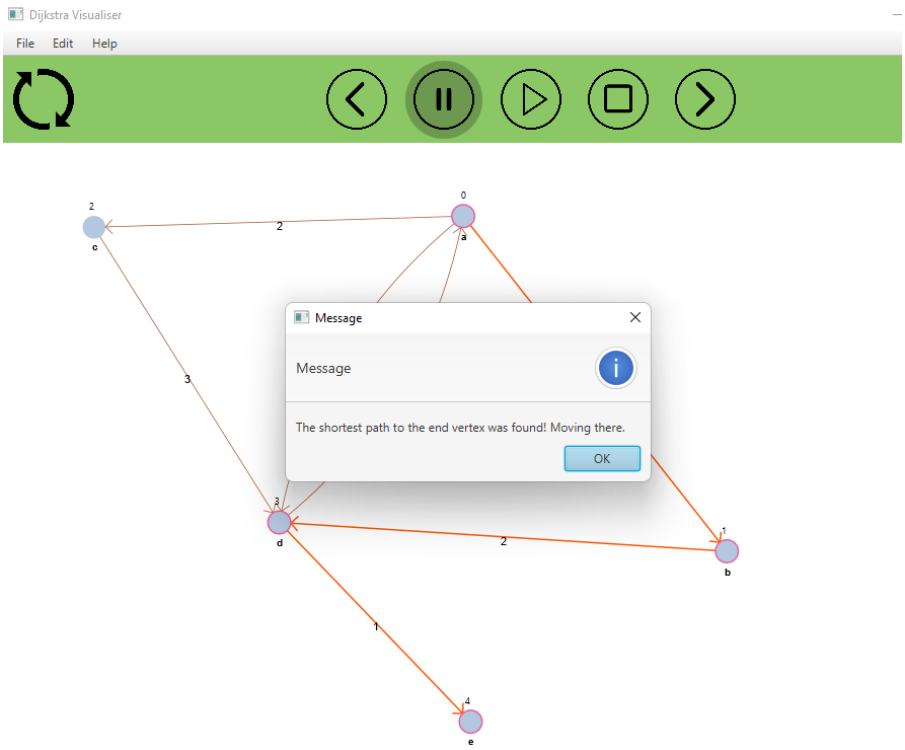


Рисунок 23 — Вывод сообщения о нахождении пути

Каждое меню на **панели инструментов** вверху программы было протестировано на работоспособность. В любой момент времени пользователю доступны только необходимые инструменты. Например, если пользователь ещё не выбрал тип графа, с которым он хочет работать, ему не доступны никакие инструменты редактирования или воспроизведения. Окно настроек и окно логирования может быть открыто, даже если тип графа ещё не был выбран, однако никаких конфликтных ситуаций не будет, изменение любых параметров будет учитываться в созданном графе.

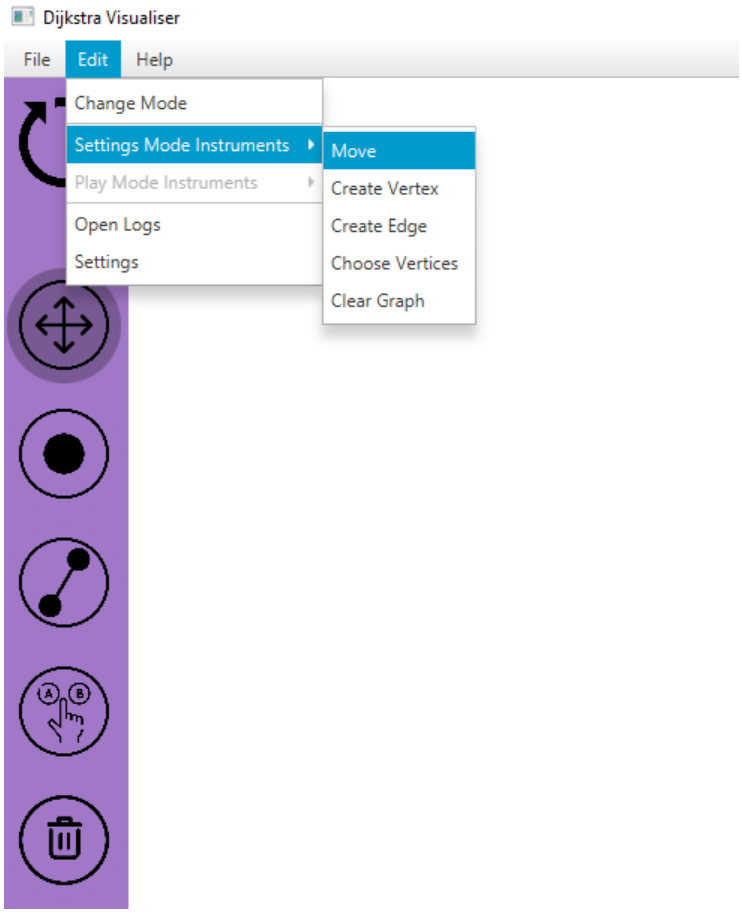


Рисунок 24 — панель инструментов в режиме редактирования

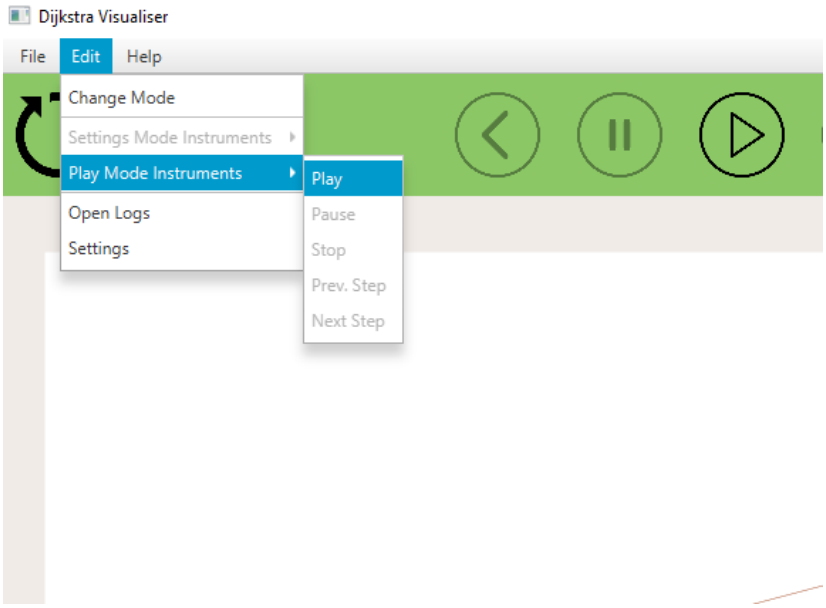


Рисунок 25 — Панель инструментов в режиме воспроизведения

Инструменты сохранения и загрузки работают корректно. При первом сохранении графа (по сути, безымянного) с помощью инструмента “Save” – будет открыт проводник, такой же, как при нажатии на “Save as”. Последующее использование инструмента “Save” будет производить сохранение без вызова

проводника. Расширение у сохраняемых файлов будет .json. Инструмент загрузки файлов будет загружать только целые файлы формата .json, то есть, если файл был повреждён каким-либо образом – приложение не сможет его загрузить, конфликта не возникнет.

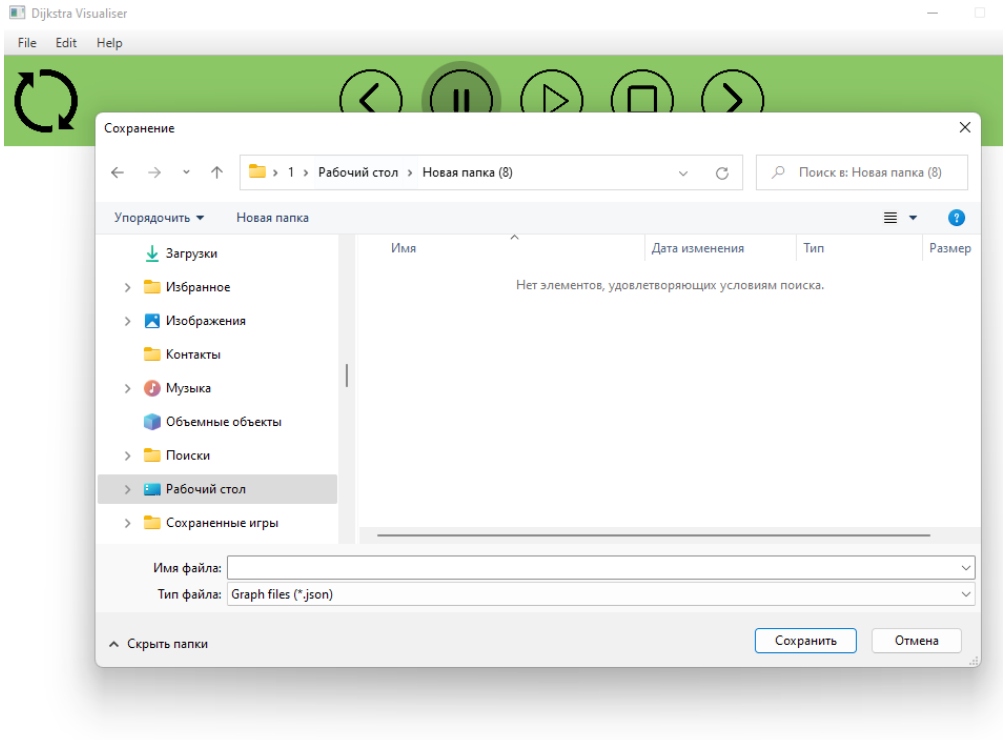


Рисунок 24 — Окно сохранения

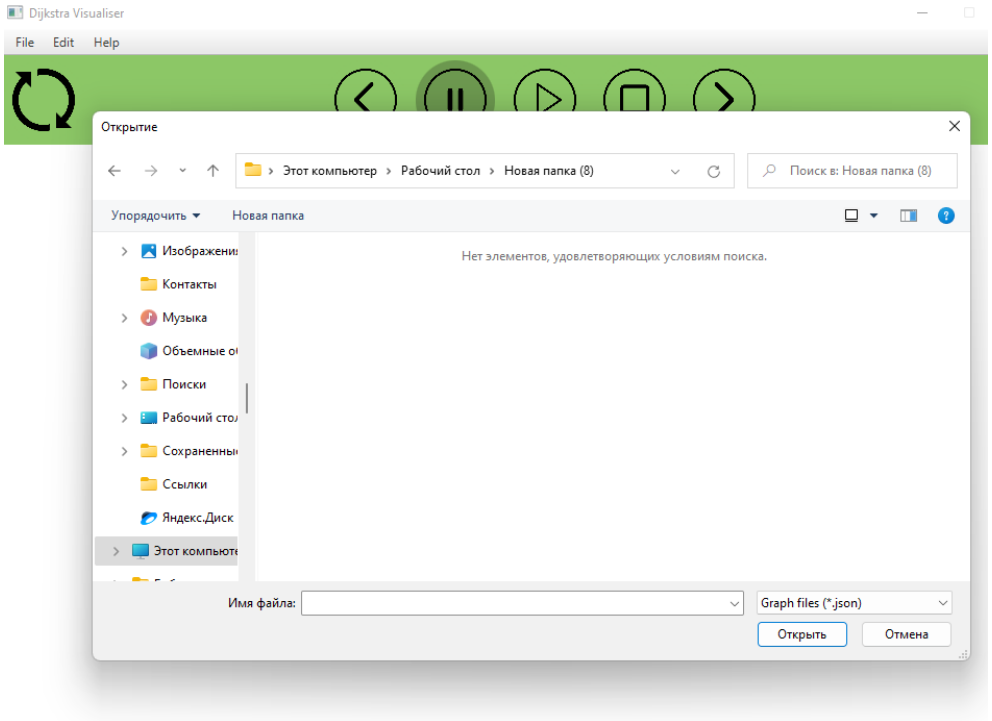


Рисунок 25 — Окно загрузки

Окно настроек тестировалось многократно, таким образом, в любой момент времени пользователь может безопасно изменить необходимый ему параметр, что не вызовет никаких конфликтов. Окно настроек не может быть одновременно открыто дважды, если пользователь решил выбрать инструмент открытия окна настроек при уже запущенном окне – фокус просто перейдёт на уже открытое окно.

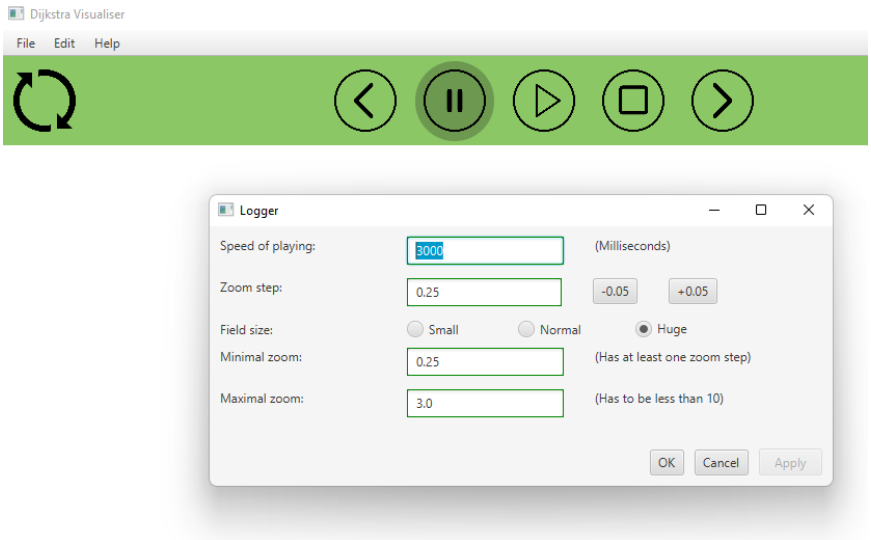


Рисунок 26 — Окно настроек

Окно логирования не доступно для редактирования пользователем, что было протестировано. Любое действие пользователя записывается в окно логирования. Аналогично окну настроек – пользователь не сможет открыть несколько окон логирования, при возникновении такой ситуации фокус просто перейдёт на уже открытое окно логирования.

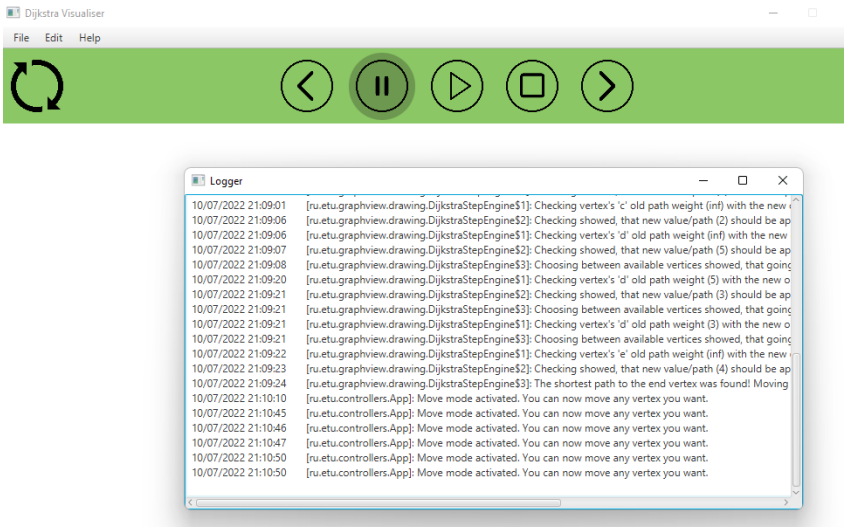


Рисунок 27 — Окно логера

4.2. Тестирования кода графа

Тестирования реализации графа подразумевает под собой создание двух групп автоматического тестирования для каждого вида графа, т.к. реализация некоторых функций у разных типов разная.

Для тестирования использовалась библиотека JUnit, которая сильно упростила этот процесс. Удобной возможностью оказалась функция, которая вызывается перед каждым тестом, в которой производился сброс графа.

Каждая функция в обоих видах тестировалась минимум тремя тестами: тестом на нормальную работу, на получение ошибки при передаче null значения или на получения ошибки при других обстоятельствах.

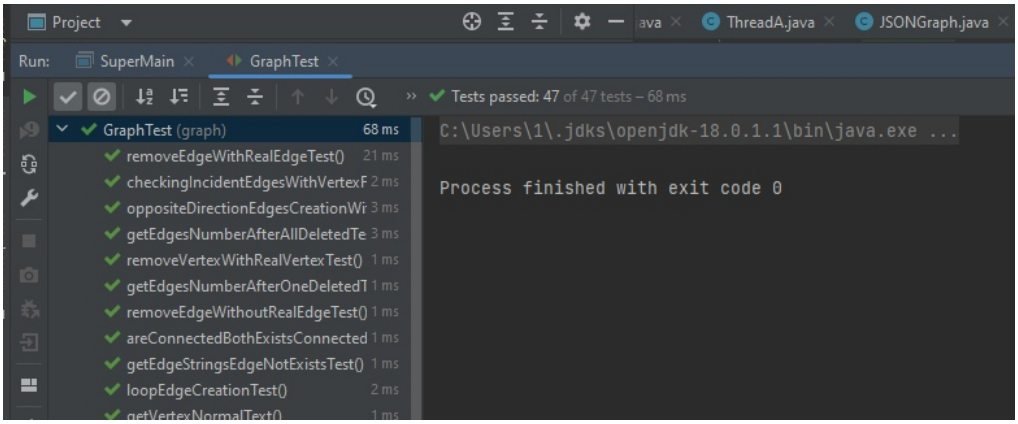


Рисунок 28 – Тесты для неориентированного графа

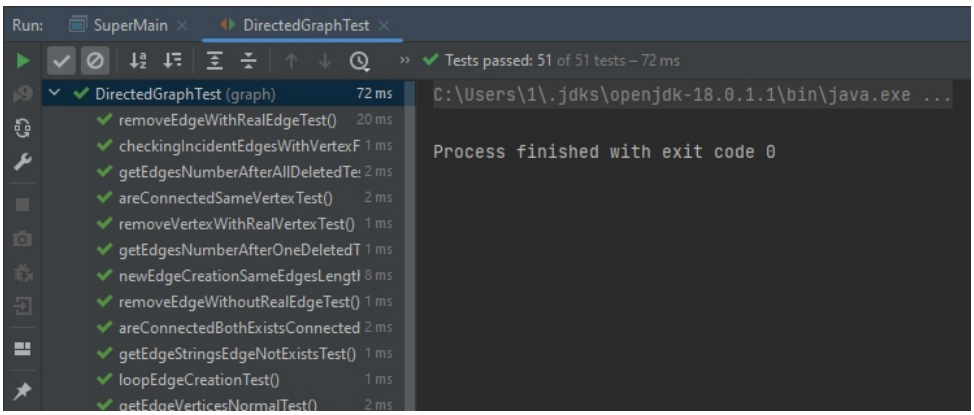


Рисунок 29 – Тесты для ориентированного графа

4.3. Тестирование кода алгоритма

Для тестирования кода алгоритма были созданы несколько автоматических тестов с помощью библиотеки JUnit. С её помощью тестировался сам алгоритм на ориентированном и неориентированном графе, а также ситуации, когда путь не существует, либо же когда существуют несколько кратчайших путей. Сами тесты подбирались, чтобы охватывать все описанные случаи.

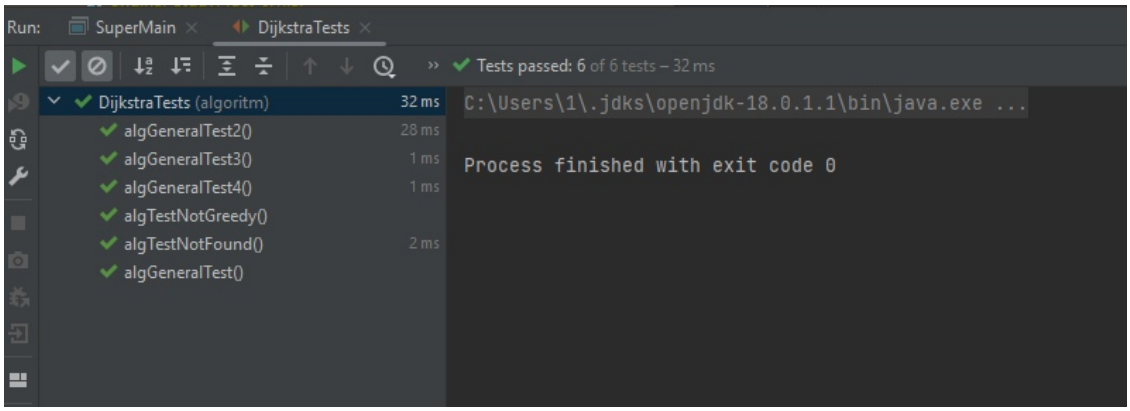


Рисунок 30 – Тестирование алгоритма на неориентированном графе

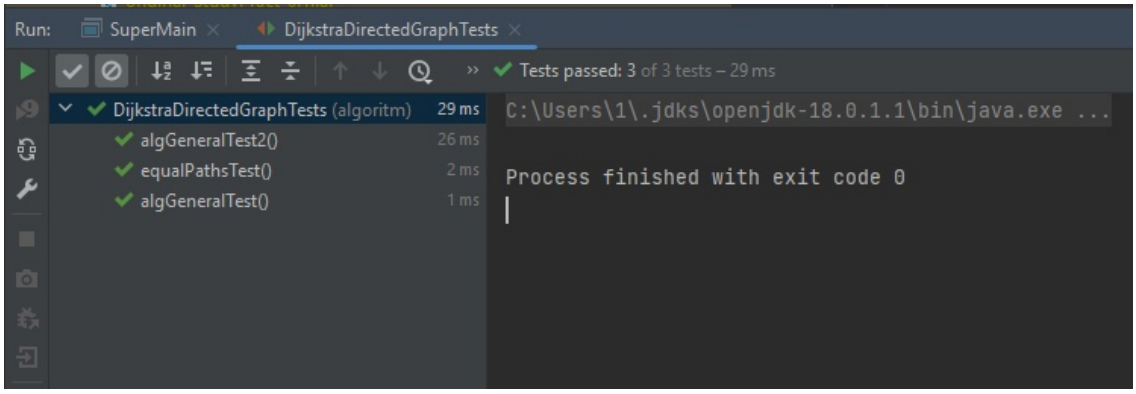


Рисунок 31 – Тестирование алгоритма на ориентированном графе

4.4. Тестирование кода сохранения графа

Для тестирования кода сохранения графа были созданы несколько автоматических тестов с помощью библиотеки JUnit. С её помощью тестировалась как загрузка, так и записи графа в текстовый файл формата JSON, причём сохранение тестировалось в разных директориях, как относительно запущенной программы, так и абсолютно начиная с дисков, а также ситуации, если невозможно открыть файл. Сами тесты были подобраны таким образом, чтобы охватить все описанные случаи, причём над ними была проведена дополнительная настройка, чтобы порядок выполнения тестов соответствовал их порядку в коде, так как некоторые из них взаимосвязаны. Например, сначала идут тесты сохранения графов в файлы в разных директориях, после чего идут тесты на загрузку графов из сохранённых ранее директорий. К тому же, после выполнения тестов все созданные в тестах файлы с графами автоматически удаляются к компьютера.

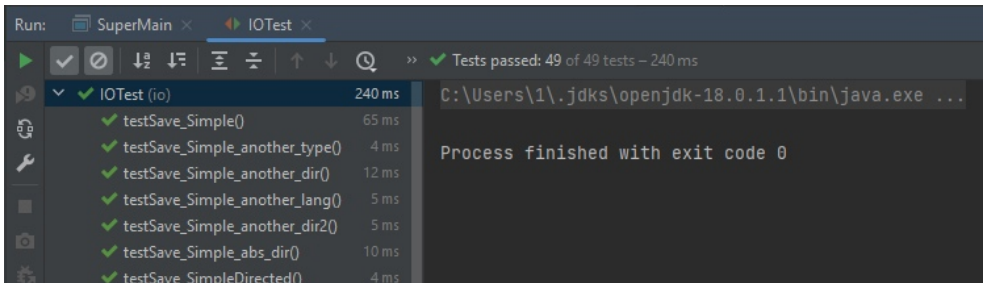


Рисунок 32 – Тесты сохранения графов разлиных типов

ЗАКЛЮЧЕНИЕ

Подводя итоги проделанной работы, выполнение всех поставленных задач было выполнено в короткие сроки с минимальным количеством проблем. Некоторые концепции и способы реализации были подкорректированы в процессе разработки, опираясь на реальность их реализации за поставленное время. Тем не менее, в конечном итоге получился качественный продукт.

Конечно, в программе могло остаться некоторое количество багов, которые не были обнаружены при тестировании. На их поиск и исправление требуется дополнительное время, которое нельзя уместить в заложенные сроки практики.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Форум программистов, где можно получить помощь по любому интересующему вопросу // Stack overflow. URL: https://stackoverflow.com
2. Документация по JavaFX // Oracle: https://docs.oracle.com/javase/8/javase-clienttechnologies.htm
3. Курс обучения Java // Stepik: https://stepik.org/course/187/promo
4. Сайт с бесплатными иконками для приложений // Flaticon: https://www.flaticon.com
5. Статья по работе с JavaFX // Habr: https://habr.com/ru/post/474292/

ПРИЛОЖЕНИЕ А

НАЗВАНИЕ ПРИЛОЖЕНИЯ

Dijkstra.java

```
package ru.etu.algoritm;

import javafx.util.Pair;
import ru.etu.graph.Edge;
import ru.etu.graph.Graph;
import ru.etu.graph.Vertex;

import java.util.*;

public class Dijkstra {

    /**
     * stack with shortest path from beginning to end
     */
    private Stack<Vertex> path;
    /**
     * Graph to use for pathfinding
     */
    private Graph graph;
    private ArrayList<StepDeltaData> steps;

    public Dijkstra() {
    }

    /**
     * Constructor
     *
     * @param graph graph to use for pathfinding
     */
    public Dijkstra(Graph graph) {
        this.graph = graph;
        path = new Stack<>();
    }

    /**
     * Method to launch pathfinding. Founded path can be accessed via
getPath() method.
     *
     * @param currentInput pathfinding beginning vertex
     * @param toInput      pathfinding end vertex
     * @return true on success (path found), false on failure (path not
found)
     */
    public boolean findPath(Vertex currentInput, Vertex toInput) {
        path.removeAllElements();
```

```
        steps = new ArrayList<>();
        VertexDijkstra current = new VertexDijkstra(currentInput);
        VertexDijkstra to = new VertexDijkstra(toInput);

        PriorityQueue<Pair<Integer, VertexDijkstra>> queue = new
PriorityQueue<>(Comparator.comparingInt(Pair::getKey));
        HashMap<String, Pair<Integer, VertexDijkstra>> nodesToProcess =
new HashMap<>();
        ArrayList<String> checkedVertexes = new ArrayList<>();

        int addPoints = 0;

        while (!current.equals(to)) {
            ArrayList<VertexDijkstra> vertexes =
getConnectedVertexes(current);
            for (Vertex elem : vertexes) {
                if (checkedVertexes.contains(elem.getData())) {
                    continue;
                }
                VertexDijkstra output = new VertexDijkstra(elem);
                int vertexCost = addPoints + getNodeEdgeWeight(current,
output);
                createStep(StepType.CHECK, current, output, vertexCost);
                if (nodesToProcess.containsKey(output.getData())) {
                    if (nodesToProcess.get(output.getData()).getKey() >
vertexCost) {
                        createStep(StepType.UPDATE, current, output,
vertexCost);
                        queue.add(new Pair<>(vertexCost, output));
                        nodesToProcess.put(output.getData(), new
Pair<>(vertexCost, output));
                        output.setParent(current);
                    }
                } else {
                    createStep(StepType.UPDATE, current, output,
vertexCost);
                    nodesToProcess.put(output.getData(), new
Pair<>(vertexCost, output));
                    queue.add(new Pair<>(vertexCost, output));
                    output.setParent(current);
                }
            }

            if (queue.isEmpty()) {
                return false;
            }
            checkedVertexes.add(current.getData());
            current = queue.remove().getValue();
            addPoints = nodesToProcess.get(current.getData()).getKey();
```

```
            createStep(StepType.MOVE, current.getParent(), current,
addPoints);
        }
        Stack<Vertex> reverse = new Stack<>();
        while (current.getParent() != null) {
            reverse.add(current);
            current = current.getParent();
        }
        reverse.add(current);
        while (!reverse.isEmpty()) {
            path.add(reverse.pop());
        }
        return true;
    }

    /**
     * Method of finding a distance betveen 2 Vertexes
     *
     * @param from parent vertex
     * @param to   current vertex
     * @return distance between vertexes
     */
    private int getNodeEdgeWeight(Vertex from, Vertex to) {
        var edge = graph.getEdge(from, to);

        if (edge != null) {
            return edge.getData();
        }

        return Integer.MAX_VALUE;
    }

    private ArrayList<VertexDijkstra> getConnectedVertexes(VertexDijkstra
current) {
        ArrayList<VertexDijkstra> vertexes = new ArrayList<>();
        List<Edge> edges = graph.incidentEdges(current);
        for (Edge elem : edges) {
            vertexes.add(new VertexDijkstra(graph.opposite(current,
elem)));
        }
        return vertexes;
    }

    /**
     * Get a path from last find iteration
     *
     * @return founded path
     */
    public List<String> getPath() {
        return path.stream().map(Vertex::getData).toList();
```

```
    }

    public Graph getGraph() {
        return graph;
    }

    public void setGraph(Graph graph) {
        this.graph = graph;
    }

    private void createStep(StepType type, VertexDijkstra current,
VertexDijkstra child, int weight) {
        steps.add(new StepDeltaData(type, current, child, weight));
    }

    public ArrayList<StepDeltaData> getSteps() {
        return steps;
    }
}
```

StepDeltaData.java

```
package ru.etu.algoritm;

public class StepDeltaData {
    private StepType type;
    private VertexDijkstra current;
    private VertexDijkstra child;
    private int weight;

    public StepDeltaData(StepType type, VertexDijkstra current,
VertexDijkstra child, int weight) {
        this.type = type;
        this.current = current;
        this.child = child;
        this.weight = weight;
    }

    public StepType getType() {
        return type;
    }

    public VertexDijkstra getCurrent() {
        return current;
    }

    public VertexDijkstra getChild() {
        return child;
    }

    public int getWeight() {
```

```
        return weight;
    }
}
```

StepType.java

```
package ru.etu.algoritm;

public enum StepType {
    CHECK,
    MOVE,
    UPDATE
}
```

VertexDijkstra.java

```
package ru.etu.algoritm;

import ru.etu.graph.Vertex;

import java.util.Objects;

/**
 * Extended version of Vertex for Dijkstra algorithm with "parent"
reference
 */
public class VertexDijkstra extends Vertex {

    private VertexDijkstra parent;

    private final Vertex original;

    /**
     * Constructor
     *
     * @param vertex original vertex
     */
    public VertexDijkstra(Vertex vertex) {
        super(vertex.getData());

        original = vertex;
    }

    /**
     * Add parent vertex (vertex with shortest way from beginning)
     *
     * @param parent parent vertex
     */
    public void setParent(VertexDijkstra parent) {
        this.parent = parent;
    }
```

```
    /**
     * Get parent vertex (vertex with shortest way from beginning)
     *
     * @return parent vertex
     */
    public VertexDijkstra getParent() {
        return parent;
    }

    /**
     * Returns original Vertex from which this one was created
     *
     * @return original Vertex from which this one was created
     */
    public Vertex getOriginal() {
        return original;
    }

    @Override
    public boolean equals(Object o) {
        if (o.getClass() == Vertex.class) return o.equals(this);
        if (!(o instanceof VertexDijkstra)) return false;
        //System.out.println(((VertexDijkstra) o).getElement()+" and
"+this.getElement()+" = "+Objects.equals(((VertexDijkstra)
o).getElement(), this.getElement()));
        return Objects.equals(((VertexDijkstra) o).getData(),
this.getData());
    }
}
```

About.java

```
package ru.etu.controllers;

import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.Button;
import javafx.scene.control.TextArea;
import javafx.scene.control.Tooltip;
import javafx.stage.Stage;

import java.awt.*;
import java.awt.datatransfer.Clipboard;
import java.awt.datatransfer.StringSelection;

import java.net.URI;
import java.net.URL;
import java.util.ResourceBundle;

public class About implements Initializable {
```

```
    public Button okBtn;
    public Button copyBtn;
    public TextArea textArea;

    private App appController;

    @Override
    public void initialize(URL url, ResourceBundle resourceBundle) {
        okBtn.setOnAction(event -> {
            Stage stage = (Stage) okBtn.getScene().getWindow();
            stage.close();
            appController.setAboutOpen(false);
        });

        copyBtn.setOnAction(event -> {
            StringSelection stringSelection = new
StringSelection(textArea.getText());
            Clipboard clipboard =
Toolkit.getDefaultToolkit().getSystemClipboard();
            clipboard.setContents(stringSelection, null);

            Stage stage = (Stage) okBtn.getScene().getWindow();
            stage.close();
            appController.setAboutOpen(false);
        });
    }

    public void setAppController(App appController) {
        this.appController = appController;
    }

    @FXML
    private void arisGitHubLink() {
        try{
            Desktop.getDesktop().browse(new
URI("https://github.com/ilya201232"));
        } catch (Exception ignored) {}
    }

    @FXML
    private void ragrGitHubLink() {
        try{
            Desktop.getDesktop().browse(new
URI("https://github.com/mnelenpridumivat"));
        } catch (Exception ignored) {}
    }

    @FXML
    private void kostGitHubLink() {
        try{
```

```
            Desktop.getDesktop().browse(new
URI("https://github.com/Kostebelova-Elizaveta"));
        } catch (Exception ignored) {}
    }
}
```

App.java

```
package ru.etu.controllers;

import javafx.application.Platform;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.ReadOnlyDoubleWrapper;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.geometry.Bounds;
import javafx.scene.Cursor;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.control.Button;
import javafx.scene.control.Menu;
import javafx.scene.control.MenuBar;
import javafx.scene.control.MenuItem;
import javafx.scene.layout.AnchorPane;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.Pane;
import javafx.stage.FileChooser;
import javafx.stage.Stage;
import ru.etu.graph.DirectedGraphList;
import ru.etu.graph.Graph;
import ru.etu.graph.GraphList;
import ru.etu.graphview.drawing.DijkstraStepEngine;
import ru.etu.graphview.GraphPane;
import ru.etu.graphview.GraphViewProperties;
import ru.etu.graphview.drawing.GraphEditor;
import ru.etu.graphview.drawing.ViewMode;
import ru.etu.io.IOGraph;
import ru.etu.logger.Logger;

import java.awt.*;
import java.io.File;
import java.io.IOException;
import java.net.URI;
import java.net.URL;
import java.util.ResourceBundle;

public class App implements Initializable {
```

```
/*
INTERFACE OBJECTS
 */

// Buttons
public Button prevBtn;
public Button nextBtn;
public ToggleButton pauseBtn;
public ToggleButton playBtn;
public Button stopBtn;
public ToggleButton moveBtn;
public ToggleButton vertexBtn;
public ToggleButton edgeBtn;
public ToggleButton chooseBtn;
public Button clearBtn;

// Menus and menu items
public MenuItem saveMeMenuItem;
public MenuItem saveAsMenuItem;
public MenuItem changeMenuItem;
public Menu playMenu;
public Menu settingsMenu;
public MenuItem exitMenuItem;
public MenuItem prevMenuItem;
public MenuItem nextMenuItem;
public MenuItem pauseMenuItem;
public MenuItem playMenuItem;
public MenuItem stopMenuItem;
public MenuItem moveMenuItem;
public MenuItem vertexMenuItem;
public MenuItem edgeMenuItem;
public MenuItem chooseMenuItem;
public MenuItem clearMenuItem;

// Other objects
public BorderPane contentBorderPane;
public AnchorPane mainPane;
public MenuBar menuBar;
public Pane graphPanePane;

/*
SCALING VARIABLES
 */
/**
 * Minimum scale factor. Must be a multiple of scaleStep
 */
private double minScale = 0.25;

/**
 * Maximum scale factor. Must be a multiple of scaleStep
```

```
     */
    private double maxScale = 3;

    /**
     * Step of scaling
     */
    private double scaleStep = 0.25;

    private int standardWidth = 1000;
    private int standardHeight = 700;
    private int paneSizeFactor = 3;
    private boolean isJunk = true;

    private final DoubleProperty scaleFactor = new
ReadOnlyDoubleWrapper(1);

    /*
    SAVE/LOAD VARIABLES
     */
    //TODO: change type to .graph
    private final String fileType = ".json";
    //private String filePath = "default"+fileType;
    private String filePath = null;

    /*
    STATE VARIABLES AND CONNECTED OBJECT TO THIS STATES
     */
    private boolean isPlaying = false;
    private DijkstraStepEngine stepEngine;
    private int playSpeed = 3000;

    private boolean isGraphCreated = false;
    public GraphPane graphPane;
    private GraphEditor graphEditor;

    private boolean isLeft = true;
    public Node top;
    public Node left;
    private ViewMode lastInstrumentType;

    private boolean isLoggerOpen = false;
    private Stage loggerStage;
    private Logger loggerInstance;

    private boolean isSettingsOpen = false;
    private Stage settingsStage;

    private boolean isAboutOpen = false;
    private Stage aboutStage;
```

```
    Settings settingsController = null;

    @Override
    public void initialize(URL url, ResourceBundle resources) {
        /*
        GRAPH PANE INITIALISATION
         */

        graphPane = new GraphPane();
        graphPane.setPrefHeight(standardHeight * paneSizeFactor);
        graphPane.setPrefWidth(standardWidth * paneSizeFactor);

        graphPanePane.getChildren().add(graphPane);
        graphPane.toFront();

        menuBar.setViewOrder(-500);
        graphPanePane.setViewOrder(500);

        // Graph editor init
        graphEditor = new GraphEditor(graphPane);

        // Scaling init
        enablePanAndZoom();

        /*
        PANELS INITIALIZATION
         */

        top = contentBorderPane.getTop();
        left = contentBorderPane.getLeft();

        contentBorderPane.setTop(null);

        /*
        SETTINGS INITIALIZATION
         */
        try {
            FXMLLoader settingsFxmlLoader = new
FXMLLoader(getClass().getResource("/ru/etu/studypract/settings.fxml"));

            Scene appScene = new Scene(settingsFxmlLoader.load());
            settingsStage = new Stage();
            settingsStage.setTitle("Logger");
            settingsStage.setScene(appScene);

            settingsStage.setOnCloseRequest((event) -> {
                settingsStage.hide();
                isSettingsOpen = false;
            });
```

```
            settingsController = settingsFxmlLoader.getController();
            settingsController.setApp(this);
            settingsController.setData(playSpeed, maxScale, minScale,
scaleStep, paneSizeFactor);
        } catch (IOException ioException) {
            System.err.println("Failed to load settings fxml!");
        } catch (Exception exception) {
            exception.printStackTrace();
        }


        /*
        LOGGER INITIALIZATION
         */

        LoggerView loggerViewController = null;
        try {
            FXMLLoader loggerFxmlLoader = new
FXMLLoader(getClass().getResource("/ru/etu/studypract/loggerView.fxml"));

            Scene appScene = new Scene(loggerFxmlLoader.load());
            loggerStage = new Stage();
            loggerStage.setTitle("Logger");
            loggerStage.setScene(appScene);
            loggerStage.setMinWidth(600);
            loggerStage.setMinHeight(400);

            loggerStage.setOnCloseRequest((event) -> {
                loggerStage.hide();
                isLoggerOpen = false;
            });

            loggerViewController = loggerFxmlLoader.getController();
        } catch (IOException ioException) {
            System.err.println("Failed to load logger fxml!");
        } catch (Exception exception) {
            exception.printStackTrace();
        }

        Logger.initialiseInstance(loggerViewController);
        loggerInstance = Logger.getInstance();

        /*
        BUTTONS AND MENU INITIALIZATION
         */

        setTopButtonsDisable(true);

        settingsMenu.setDisable(true);
        saveMeMenuItem.setDisable(true);
```

```
        saveAsMenuItem.setDisable(true);

        setLeftButtonsDisable(true);

        playMenu.setDisable(true);

        setUpBtnsActions();
    }

    private void setUpBtnsActions() {
        //buttons/menu items
        exitMenuItem.setOnAction(event -> eventCloseWindow());

        changeMenuItem.setOnAction(event -> {
            changeMode();
        });

        EventHandler<ActionEvent> vertexBtnSelectHandler = event -> {
            setVertexCreationMode();
            if (!vertexBtn.isSelected()) {
                vertexBtn.setSelected(true);
            }
        };
        EventHandler<ActionEvent> edgeBtnSelectHandler = event -> {
            setEdgeCreationMode();
            if (!edgeBtn.isSelected()) {
                edgeBtn.setSelected(true);
            }
        };
        EventHandler<ActionEvent> moveBtnSelectHandler = event -> {
            setMoveMode();
            if (!moveBtn.isSelected()) {
                moveBtn.setSelected(true);
            }
        };
        EventHandler<ActionEvent> chooseBtnSelectHandler = event -> {
            setChooseMode();
            if (!chooseBtn.isSelected()) {
                chooseBtn.setSelected(true);
            }
        };
        EventHandler<ActionEvent> clearBtnSelectHandler = event -> {
            clearGraph();
        };
        EventHandler<ActionEvent> playBtnSelectHandler = event -> {
            play();
            if (!playBtn.isSelected()) {
                playBtn.setSelected(true);
            }
        };
```

```
        EventHandler<ActionEvent> stopBtnSelectHandler = event -> {
            stop();
            if (playBtn.isSelected()) {
                playBtn.setSelected(false);
            }
            if (pauseBtn.isSelected()) {
                pauseBtn.setSelected(false);
            }
        };
        EventHandler<ActionEvent> nextBtnSelectHandler = event -> {
            stepForward();
            pauseBtn.setSelected(true);
        };
        EventHandler<ActionEvent> prevBtnSelectHandler = event -> {
            stepBack();
            pauseBtn.setSelected(true);
        };
        EventHandler<ActionEvent> pauseBtnSelectHandler = event -> {
            pause();
            if (!pauseBtn.isSelected()) {
                pauseBtn.setSelected(true);
            }
        };

        vertexBtn.setOnAction(vertexBtnSelectHandler);
        edgeBtn.setOnAction(edgeBtnSelectHandler);
        moveBtn.setOnAction(moveBtnSelectHandler);
        chooseBtn.setOnAction(chooseBtnSelectHandler);
        clearBtn.setOnAction(clearBtnSelectHandler);
        playBtn.setOnAction(playBtnSelectHandler);
        stopBtn.setOnAction(stopBtnSelectHandler);
        nextBtn.setOnAction(nextBtnSelectHandler);
        prevBtn.setOnAction(prevBtnSelectHandler);
        pauseBtn.setOnAction(pauseBtnSelectHandler);

        vertexMenuItem.setOnAction(vertexBtnSelectHandler);
        edgeMenuItem.setOnAction(edgeBtnSelectHandler);
        moveMenuItem.setOnAction(moveBtnSelectHandler);
        chooseMenuItem.setOnAction(chooseBtnSelectHandler);
        clearMenuItem.setOnAction(clearBtnSelectHandler);
        playMenuItem.setOnAction(playBtnSelectHandler);
        stopMenuItem.setOnAction(stopBtnSelectHandler);
        nextMenuItem.setOnAction(nextBtnSelectHandler);
        prevMenuItem.setOnAction(prevBtnSelectHandler);
        pauseMenuItem.setOnAction(pauseBtnSelectHandler);
    }

    private void enablePanAndZoom() {

        graphPanePane.setOnScroll(event -> {
```

```
            String OS = System.getProperty("os.name",
"generic").toLowerCase();
            //System.out.println(OS);
            if(!OS.contains("win")){
                if (isJunk) {
                    isJunk = false;
                    return;
                } else {
                    isJunk = true;
                }
            }

            var direction = event.getDeltaY() >= 0 ? 1 : -1;

            var curScale = scaleFactor.getValue();
            var computedScale = curScale + direction * scaleStep;

            if (Double.compare(computedScale, minScale) < 0) {
                computedScale = minScale;
            } else if (Double.compare(computedScale, maxScale) > 0) {
                computedScale = maxScale;
            }

            if (curScale != computedScale) {

                graphPane.setScaleX(computedScale);
                graphPane.setScaleY(computedScale);

                if (computedScale == minScale) {
                    if (isLeft) {
                        graphPane.setTranslateX(-
graphPanePane.getTranslateX() - 43 - standardWidth / 2f *
(Math.max((paneSizeFactor - 1), 0)));
                        graphPane.setTranslateY(-
graphPanePane.getTranslateY() - standardHeight / 2f *
(Math.max((paneSizeFactor - 1), 0)));
                    } else {
                        graphPane.setTranslateX(-
graphPanePane.getTranslateX() - standardWidth / 2f *
(Math.max((paneSizeFactor - 1), 0)));
                        graphPane.setTranslateY(-
graphPanePane.getTranslateY() - 43 - standardHeight / 2f *
(Math.max((paneSizeFactor - 1), 0)));
                    }
                } else {
                    scaleFactor.setValue(computedScale);

                    // Положение в сцене
```

```
                    Bounds bounds =
graphPane.localToScene(graphPane.getBoundsInLocal());
                    double f = (computedScale / curScale) - 1;
                    double dx = (event.getX() - (bounds.getWidth() / 2 +
bounds.getMinX()));
                    double dy = (event.getY() - (bounds.getHeight() / 2 +
bounds.getMinY()));

                    graphPane.setTranslateX(graphPane.getTranslateX() - f
* dx);
                    graphPane.setTranslateY(graphPane.getTranslateY() - f
* dy);
                }
            }

            event.consume();

        });

        final DragData dragData = new DragData();

        graphPanePane.setOnMousePressed(event -> {
            if (event.isSecondaryButtonDown()) {
                dragData.cursorType =
graphPanePane.getScene().getCursor();
                graphPanePane.getScene().setCursor(Cursor.MOVE);

                dragData.mouseAnchorX = event.getX();
                dragData.mouseAnchorY = event.getY();

                dragData.translateAnchorX = graphPane.getTranslateX();
                dragData.translateAnchorY = graphPane.getTranslateY();
            }
        });

        graphPanePane.setOnMouseReleased(event -> {
            graphPanePane.getScene().setCursor(dragData.cursorType);
        });

        graphPanePane.setOnMouseDragged(event -> {
            if (event.isSecondaryButtonDown()) {
                graphPane.setTranslateX(dragData.translateAnchorX +
event.getX() - dragData.mouseAnchorX);
                graphPane.setTranslateY(dragData.translateAnchorY +
event.getY() - dragData.mouseAnchorY);
            }
        });
    }

    public void setTopButtonsDisable(boolean a) {
```

```
    prevBtn.setDisable(a);
    nextBtn.setDisable(a);
    pauseBtn.setDisable(a);
    playBtn.setDisable(a);
    stopBtn.setDisable(a);
}

public void setTopMenuItemsDisable(boolean a) {
    prevMenuItem.setDisable(a);
    nextMenuItem.setDisable(a);
    pauseMenuItem.setDisable(a);
    playMenuItem.setDisable(a);
    stopMenuItem.setDisable(a);
}

public void setLeftButtonsDisable(boolean a) {
    moveBtn.setDisable(a);
    vertexBtn.setDisable(a);
    edgeBtn.setDisable(a);
    chooseBtn.setDisable(a);
    clearBtn.setDisable(a);
}

public void setLeftMenuItemsDisable(boolean a) {
    moveMenuItem.setDisable(a);
    vertexMenuItem.setDisable(a);
    edgeMenuItem.setDisable(a);
    chooseMenuItem.setDisable(a);
    clearMenuItem.setDisable(a);
}

@FXML
private void changeMode() {
    if (!isLeft) {
        // Setting mode

        if (lastInstrumentType != null) {
            switch (lastInstrumentType) {
                case NORMAL -> {
                    setMoveMode();
                }
                case VERTEX_PLACEMENT -> {
                    setVertexCreationMode();
                }
                case VERTEX_CHOOSE -> {
                    setChooseMode();
                }
                case EDGE_PLACEMENT -> {
                    setEdgeCreationMode();
                }
```

```
                }
            }

            // Stopping threads in stepEngine if necessary
            if (isPlaying) {
                stop();
            }

            if (graphEditor.getNodesSelected1() == 2) {
                graphEditor.removeAlgLabels();
            }

            // Setting buttons accessibility
            playMenu.setDisable(true);

            if (isGraphCreated) {
                setLeftButtonsDisable(false);
                setLeftMenuItemsDisable(false);
                settingsMenu.setDisable(false);
            } else {
                setLeftButtonsDisable(true);
                settingsMenu.setDisable(true);
            }

            // Translating coordinate if scale is right
            if (scaleFactor.get() == minScale + scaleStep) {
                graphPane.setTranslateX(-graphPanePane.getTranslateX() -
43 - standardWidth / 2f * (Math.max((paneSizeFactor - 1), 0)));
                graphPane.setTranslateY(-graphPanePane.getTranslateY() -
standardHeight / 2f * (Math.max((paneSizeFactor - 1), 0)));
            }

            // Push log message
            loggerInstance.printMessage(getClass().getName(), "Switching
to Setting mode.");

            // Switching panels
            contentBorderPane.setTop(null);
            contentBorderPane.setLeft(left);
            isLeft = true;

        } else {
            // Play mode

            // Setting buttons accessibility
            settingsMenu.setDisable(true);

            if (isGraphCreated) {
                setMoveModeWithoutSaving();
            }
```

```
            if (graphPane.isGraphSet() && graphEditor.getNodesSelected1()
== 2) {
                // Initialising stepEngine and preparing vertices for
work

graphEditor.addAlgLabels(graphEditor.getFirstNodeSelect().getVertex());
                stepEngine = new DijkstraStepEngine(graphEditor,
playSpeed);

                playBtn.setDisable(false);

                playMenu.setDisable(false);
                setTopMenuItemsDisable(true);
                playMenuItem.setDisable(false);
            } else {
                setTopButtonsDisable(true);
                setTopMenuItemsDisable(true);
            }

            // Translating coordinate if scale is right
            if (scaleFactor.get() == minScale + scaleStep) {
                graphPane.setTranslateX(-graphPanePane.getTranslateX() -
standardWidth / 2f * (Math.max((paneSizeFactor - 1), 0)));
                graphPane.setTranslateY(-graphPanePane.getTranslateY() -
43 - standardHeight / 2f * (Math.max((paneSizeFactor - 1), 0)));
            }

            // Push log message
            loggerInstance.printMessage(getClass().getName(), "Switching
to play mode.");

            // Switching panels
            contentBorderPane.setTop(top);
            contentBorderPane.setLeft(null);
            isLeft = false;
        }
    }

    public void eventCloseWindow() {
        if (isGraphCreated && isPlaying) {
            stepEngine.stop();
        }

        Platform.exit();
    }

    public void closeWindow(Stage stage) {
        stage.setOnCloseRequest(event -> eventCloseWindow());
    }
```

```
    /*
    MENU BAR'S METHODS
     */
    @FXML
    private void saveGraph() {
        if(filePath==null){
            saveGraphAs();
        } else {
            try {
                IOGraph saver = new IOGraph();
                if (isGraphCreated) {
                    saver.saveGraph(filePath, graphPane.getGraph());
                } else {
                    loggerInstance.printMessage(getClass().toString(),
"Cannot save uncreated graph!", true);
                }
            } catch (IOException ex) {
                loggerInstance.printMessage(getClass().toString(),
"Cannot save graph to "+filePath, true);
            }
        }
    }

    @FXML
    private void saveGraphAs() {
        FileChooser fileChooser = new FileChooser();
        FileChooser.ExtensionFilter extensionFilter = new
FileChooser.ExtensionFilter("Graph files (*"+fileType+")", "*"+fileType);
        fileChooser.getExtensionFilters().add(extensionFilter);
        Stage stage = new Stage();
        stage.setTitle("Save graph");
        File path = fileChooser.showSaveDialog(stage);
        if (path != null) {
            try {
                IOGraph saver = new IOGraph();
                if (isGraphCreated) {
                    saver.saveGraph(fixFileName(path),
graphPane.getGraph());
                } else {
                    loggerInstance.printMessage(getClass().toString(),
"Cannot save uncreated graph!", true);
                }
                filePath = fixFileName(path);
            } catch (IOException ex) {
                loggerInstance.printMessage(getClass().toString(),
"Cannot save graph to "+fixFileName(path), true);
            }
        }
    }
```

```
    @FXML
    private void loadGraph() {
        FileChooser fileChooser = new FileChooser();
        FileChooser.ExtensionFilter extensionFilter = new
FileChooser.ExtensionFilter("Graph files (*"+fileType+")", "*"+fileType);
        fileChooser.getExtensionFilters().add(extensionFilter);
        Stage stage = new Stage();
        stage.setTitle("Load graph");
        File path = fileChooser.showOpenDialog(stage);
        if (path != null) {
            if (isPlaying) {
                changeMode();
            }
            try {
                IOGraph saver = new IOGraph();
                Graph newGraph = saver.loadGraph(path.toString());
                var properties = new GraphViewProperties();
                properties.setNeedArrows(newGraph instanceof
DirectedGraphList);
                properties.setNeedDoubleEdges(newGraph instanceof
DirectedGraphList);
                graphEditor.eraseGraph();
                graphPane.loadGraph(newGraph, properties);

                setLeftButtonsDisable(false);
                settingsMenu.setDisable(false);
                isGraphCreated = true;
                filePath = path.toString();
            } catch (IOException ex) {
                loggerInstance.printMessage(getClass().toString(),
"Cannot open graph "+path.toString(), true);
            }
            if (isGraphCreated) {
                saveMeMenuItem.setDisable(false);
                saveAsMenuItem.setDisable(false);
                setMoveMode();
                moveBtn.setSelected(true);
            }
        }
    }

    public void setAboutOpen(boolean aboutOpen) {
        isAboutOpen = aboutOpen;
    }

    @FXML
    private void openRepo() {
        try {
```

```
            Desktop.getDesktop().browse(new
URI("https://github.com/ilya201232/StudyPract"));
        } catch (Exception ignored) {
        }

    }

    @FXML
    private void openLogger() {
        if (!isLoggerOpen) {
            isLoggerOpen = true;
            loggerStage.show();
        } else {
            loggerStage.requestFocus();
        }
    }

    @FXML
    private void openSettings() throws IOException {
        if (!isSettingsOpen || settingsController.getClosed()) {
            isSettingsOpen = true;
            settingsController.setClosed(false);
            settingsStage.show();
        } else {
            settingsStage.requestFocus();
        }
    }

    public void setSettings(int playSpeed, double zoomStep, double
zoomMax, double zoomMin, int typeOfSize) {

        this.playSpeed = playSpeed;
        scaleStep = zoomStep;
        maxScale = zoomMax;
        minScale = zoomMin;
        paneSizeFactor = typeOfSize;

        if (!isLeft) {
            changeMode();
        }

        graphPane.setPrefHeight(standardHeight * paneSizeFactor);
        graphPane.setPrefWidth(standardWidth * paneSizeFactor);

        scaleFactor.setValue(minScale);

        graphPane.setScaleX(minScale);
        graphPane.setScaleY(minScale);
```

```
        graphPane.setTranslateX(-graphPanePane.getTranslateX() - 43 -
standardWidth / 2f * (Math.max((paneSizeFactor - 1), 0)));
        graphPane.setTranslateY(-graphPanePane.getTranslateY() -
standardHeight / 2f * (Math.max((paneSizeFactor - 1), 0)));


        enablePanAndZoom();

        // TODO:: реализовать тут изменение параметров (вызвать
соответствующие методы для изменения)
    }

    @FXML
    private void openAbout() throws IOException {

        if (!isAboutOpen) {
            isAboutOpen = true;

            FXMLLoader aboutFxmlLoader = new
FXMLLoader(getClass().getResource("/ru/etu/studypract/about.fxml"));

            Scene appScene = new Scene(aboutFxmlLoader.load());

            About aboutController = aboutFxmlLoader.getController();

            Stage stage = new Stage();
            stage.setTitle("About Dijkstra Visualiser");
            stage.setScene(appScene);
            stage.show();

            aboutController.setAppController(this);

            stage.setResizable(false);

            aboutStage = stage;

            stage.setOnCloseRequest((event) -> {
                isAboutOpen = false;
            });
        } else {
            aboutStage.requestFocus();
        }


    }

    // Graph creation
    @FXML
    private void createNewGraph() {
        Graph graph = new GraphList();
```

```
    var properties = new GraphViewProperties();
    properties.setNeedArrows(false);

    graphEditor.eraseGraph();
    graphPane.loadGraph(graph, properties);

    // Unblock left side tools
    setLeftButtonsDisable(false);
    settingsMenu.setDisable(false);

    // Unblock save buttons
    saveMeMenuItem.setDisable(false);
    saveAsMenuItem.setDisable(false);

    // Set move tool as active
    setMoveMode();
    moveBtn.setSelected(true);

    isGraphCreated = true;
    filePath=null;
}

@FXML
private void createNewDirectedGraph() {
    Graph graph = new DirectedGraphList();
    var properties = new GraphViewProperties();
    properties.setNeedArrows(true);

    graphEditor.eraseGraph();
    graphPane.loadGraph(graph, properties);

    // Unblock left side tools
    setLeftButtonsDisable(false);
    settingsMenu.setDisable(false);

    // Unblock save buttons
    saveMeMenuItem.setDisable(false);
    saveAsMenuItem.setDisable(false);

    // Set move tool as active
    setMoveMode();
    moveBtn.setSelected(true);

    isGraphCreated = true;
    filePath=null;
}

/*
GRAPH EDITING TOOLS' METHODS
 */
```

```
    private void setMoveModeWithoutSaving() {
        graphEditor.setViewMode(ViewMode.NORMAL);
        loggerInstance.printMessage(getClass().getName(), "Move mode
activated. You can now move any vertex you want.");
    }

    @FXML
    private void setMoveMode() {
        lastInstrumentType = ViewMode.NORMAL;
        graphEditor.setViewMode(ViewMode.NORMAL);
        loggerInstance.printMessage(getClass().getName(), "Move mode
activated. You can now move any vertex you want.");
    }

    @FXML
    private void setVertexCreationMode() {
        lastInstrumentType = ViewMode.VERTEX_PLACEMENT;
        graphEditor.setViewMode(ViewMode.VERTEX_PLACEMENT);
        loggerInstance.printMessage(getClass().getName(), "Vertex
creation mode activated. You can now place new vertices on the screen
(Moving vertices is allowed). Just click on an empty space of the
pane.");
    }

    @FXML
    private void setEdgeCreationMode() {
        lastInstrumentType = ViewMode.EDGE_PLACEMENT;
        graphEditor.setViewMode(ViewMode.EDGE_PLACEMENT);
        loggerInstance.printMessage(getClass().getName(), "Edge creation
mode activated. You can now place new edges on the screen (Moving
vertices is forbidden). Choose two vertices to create edge between
them.");
    }

    @FXML
    private void setChooseMode() {
        lastInstrumentType = ViewMode.VERTEX_CHOOSE;
        graphEditor.setViewMode(ViewMode.VERTEX_CHOOSE);
        loggerInstance.printMessage(getClass().getName(), "Vertex choose
mode activated. You can now choose two vertices to make Dijkstra
algorithm find the shortest bath between them.");
    }

    @FXML
    private void clearGraph() {
        graphEditor.eraseGraph();
        loggerInstance.printMessage(getClass().getName(), "You just
cleared the graph. Now you can place new elements.");
    }
```

```
/*
ALGORITHM PLAY TOOLS' METHOD
 */
@FXML
private void stepBack() {
    stepEngine.makeStepBackwards();
}

@FXML
private void stepForward() {
    stepEngine.makeStepForward();
}

@FXML
private void play() {
    setMoveMode();
    if (!stepEngine.isInitialised()) {
        stepEngine.applyDijkstra();
        if (stepEngine.isPathExists()) {
            stepEngine.startAutoPlay();
            isPlaying = true;
        }

    } else if (stepEngine.isPaused()) {
        stepEngine.resumeAutoPlay();
        isPlaying = true;
    } else if (!isPlaying) {
        stepEngine.startAutoPlay();
        isPlaying = true;
    }

    if (stepEngine.isPathExists()) {
        setTopButtonsDisable(false);
        setTopMenuItemsDisable(false);
        playMenu.setDisable(false);
    }
}

@FXML
private void pause() {
    stepEngine.pauseAutoPlay();
}

@FXML
private void stop() {
    stepEngine.stop();

    setTopButtonsDisable(true);
    playBtn.setDisable(false);
```

```
        setTopMenuItemsDisable(true);
        playMenuItem.setDisable(false);

        pauseBtn.setSelected(false);
        playBtn.setSelected(false);

        isPlaying = false;
    }

    private String fixFileName(File path){
        String pathStr = path.toString();
        String OS = System.getProperty("os.name",
"generic").toLowerCase();
        //System.out.println(OS);
        // TODO протестировать на debian
        if(!pathStr.endsWith(fileType)&&!OS.contains("win")){
            //loggerInstance.printMessage(getClass().getName(), "Saved to
directory "+pathStr+fileType);
            return pathStr+fileType;
        }
        //loggerInstance.printMessage(getClass().getName(), "Saved to
directory "+pathStr);
        return pathStr;
    }
}

class DragData {

    Cursor cursorType;

    double mouseAnchorX;
    double mouseAnchorY;

    double translateAnchorX;
    double translateAnchorY;
}
```

LoggerView.java

```
package ru.etu.controllers;

import javafx.beans.value.ObservableValue;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.TextArea;

import java.net.URL;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.ResourceBundle;
```

```
public class LoggerView implements Initializable {

    @FXML
    private TextArea textArea;

    @Override
    public void initialize(URL url, ResourceBundle resourceBundle) {
        textArea.textProperty().addListener((ObservableValue<?>
observable, Object oldValue,
                                             Object newValue) -> {
            textArea.setScrollTop(Double.MAX_VALUE);
        });
    }

    public void printMessage(String className, String message) {
        SimpleDateFormat formatter = new SimpleDateFormat("dd/MM/yyyy
HH:mm:ss");
        Date date = new Date();
        textArea.appendText(formatter.format(date) + "\t[" + className +
"]: " + message + "\n");
    }
}
```

Settings.java

```
package ru.etu.controllers;

import javafx.fxml.FXML;
import javafx.scene.Node;
import javafx.scene.control.Button;
import javafx.scene.control.RadioButton;
import javafx.scene.control.TextField;
import javafx.scene.control.ToggleGroup;

import java.util.ArrayList;

public class Settings {

    public ToggleGroup fieldSizeToggleGroup;

    private int playSpeed = 500;
    private double zoomStep = 0.25;
    private double zoomMax = 10;
    private double zoomMin = 0.5;
    private int typeOfSize = 2;

    private int playSpeedTmp;
    private double zoomStepTmp;
    private double zoomMaxTmp;
    private double zoomMinTmp;
```

```
private int typeOfSizeTmp;

@FXML
private Button cancelBtn;
@FXML
private Button applyBtn;

@FXML
private TextField maxZoomTextField;

@FXML
private TextField minZoomTextField;

@FXML
private Button okBtn;
@FXML
private Button increaseBtn;
@FXML
private Button decreaseBtn;

@FXML
private RadioButton smallBtn;
@FXML
private RadioButton normalBtn;
@FXML
private RadioButton hugeBtn;

@FXML
private TextField playSpeedTextField;

@FXML
private TextField zoomStepTextField;

private boolean isClosed = false;

private App myApp;

private boolean isPlaySpeedOK = true;
private boolean isMaxZoomOK = true;
private boolean isMinZoomOK = true;

@FXML
void initialize() {
    smallBtn.setUserData(1);
    normalBtn.setUserData(2);
    hugeBtn.setUserData(3);

    applyBtn.setOnAction(event -> {

        playSpeed = playSpeedTmp;
```

```
            zoomStep = zoomStepTmp;
            zoomMax = zoomMaxTmp;
            zoomMin = zoomMinTmp;
            typeOfSize = typeOfSizeTmp;

            myApp.setSettings(playSpeed, zoomStep, zoomMax, zoomMin,
typeOfSize);
//            setAllTexts();
            applyBtn.setDisable(true);
        });

        okBtn.setOnAction(event -> {
            ((Node) (event.getSource())).getScene().getWindow().hide();
            isClosed = true;

            if (!(playSpeed == playSpeedTmp && zoomMax == zoomMaxTmp &&
zoomMin == zoomMinTmp && zoomStep == zoomStepTmp && typeOfSize ==
typeOfSizeTmp)) {
                playSpeed = playSpeedTmp;
                zoomStep = zoomStepTmp;
                zoomMax = zoomMaxTmp;
                zoomMin = zoomMinTmp;
                typeOfSize = typeOfSizeTmp;

                myApp.setSettings(playSpeed, zoomStep, zoomMax, zoomMin,
typeOfSize);
                applyBtn.setDisable(true);
            }


        });

        cancelBtn.setOnAction(event -> {
            ((Node) (event.getSource())).getScene().getWindow().hide();
            isClosed = true;

            playSpeedTmp = playSpeed;
            zoomStepTmp = zoomStep;
            zoomMaxTmp = zoomMax;
            zoomMinTmp = zoomMin;
            typeOfSizeTmp = typeOfSize;

            setAllTexts();

            playSpeedTextField.setStyle("-fx-border-color: green;");
            zoomStepTextField.setStyle("-fx-border-color: green;");
            minZoomTextField.setStyle("-fx-border-color: green;");
            maxZoomTextField.setStyle("-fx-border-color: green;");
        });
```

```
        increaseBtn.setOnAction(event -> {
            if (zoomStepTmp < 0.5) {
                zoomStepTmp = (zoomStepTmp * 100 + 5) / 100;
            }
            zoomStepTextField.setText(String.valueOf(zoomStepTmp));
            validateMinFieldData();
            validateMaxFieldData();

            checkSettings();
        });

        decreaseBtn.setOnAction(event -> {
            if (zoomStepTmp > 0.05) {
                zoomStepTmp = (zoomStepTmp * 100 - 5) / 100;
            }
            zoomStepTextField.setText(String.valueOf(zoomStepTmp));
            validateMinFieldData();
            validateMaxFieldData();

            checkSettings();
        });


fieldSizeToggleGroup.selectedToggleProperty().addListener(((observable,
oldValue, newValue) -> {
            typeOfSizeTmp = (int) newValue.getUserData();

            checkSettings();
        }));

        playSpeedTextField.textProperty().addListener(
                (observable, oldValue, newValue) -> {

                    if (!newValue.matches("\\d*")) {

playSpeedTextField.setText(newValue.replaceAll("\\D", ""));
                    }

                    validatePlaySpeed();
                    if (isPlaySpeedOK) {
                        playSpeedTmp =
Integer.parseInt(playSpeedTextField.getText());
                    }
                });

        maxZoomTextField.textProperty().addListener(
                (observable, oldValue, newValue) -> {

                    if (!newValue.matches("\\d*\\.\\d*")) {
```

```
maxZoomTextField.setText(newValue.replaceAll("[^\\d.]", ""));
                    }

                    validateMaxFieldData();
                    if (isMaxZoomOK) {
                        zoomMaxTmp =
Double.parseDouble(maxZoomTextField.getText());
                    }
                    checkSettings();
                });

        minZoomTextField.textProperty().addListener(
                (observable, oldValue, newValue) -> {

                    if (!newValue.matches("\\d*\\.\\d*")) {

minZoomTextField.setText(newValue.replaceAll("[^\\d.]", ""));
                    }

                    validateMinFieldData();
                    if (isMinZoomOK) {
                        zoomMinTmp =
Double.parseDouble(minZoomTextField.getText());
                    }
                    checkSettings();
                });


    }

    private void validatePlaySpeed() {
        if (playSpeedTextField.getText().length() == 0) {
            playSpeedTextField.setStyle("-fx-border-color: red;");
            isPlaySpeedOK = false;
            checkSettings();
        } else {
            try {
                int input =
Integer.parseInt(playSpeedTextField.getText());
                if (input <= 0) {
                    playSpeedTextField.setStyle("-fx-border-color:
red;");
                    isPlaySpeedOK = false;
                    checkSettings();
                } else {
                    playSpeedTextField.setStyle("-fx-border-color:
green;");
                    isPlaySpeedOK = true;
                    checkSettings();
```

```
                }

            } catch (Exception e) {
                playSpeedTextField.setStyle("-fx-border-color: red;");
                isPlaySpeedOK = false;
                checkSettings();
            }
        }

    }

    private void validateMaxFieldData() {
        if (maxZoomTextField.getText().length() == 0) {
            maxZoomTextField.setStyle("-fx-border-color: red;");
            isMaxZoomOK = false;
            checkSettings();
        } else {
            try {
                double input =
Double.parseDouble(maxZoomTextField.getText());
                if (input > 10 || input < zoomStepTmp || ((int) (input *
100) % (int) (zoomStepTmp * 100)) != 0) {
                    maxZoomTextField.setStyle("-fx-border-color: red;");
                    isMaxZoomOK = false;
                    checkSettings();
                } else {
                    maxZoomTextField.setStyle("-fx-border-color:
green;");
                    isMaxZoomOK = true;
                    checkSettings();
                }
            } catch (NumberFormatException e) {
                maxZoomTextField.setStyle("-fx-border-color: red;");
                isMaxZoomOK = false;
                checkSettings();
            }
        }
    }

    private void validateMinFieldData() {
        if (minZoomTextField.getText().length() == 0) {
            minZoomTextField.setStyle("-fx-border-color: red;");
            isMinZoomOK = false;
            checkSettings();
        } else {
            try {
                double input =
Double.parseDouble(minZoomTextField.getText());
                if (input > 10 || input < zoomStepTmp || ((int) (input *
100) % (int) (zoomStepTmp * 100)) != 0) {
```

```
                    minZoomTextField.setStyle("-fx-border-color: red;");
                    isMinZoomOK = false;
                    checkSettings();
                } else {
                    minZoomTextField.setStyle("-fx-border-color:
green;");
                    isMinZoomOK = true;
                    checkSettings();
                }
            } catch (NumberFormatException e) {
                minZoomTextField.setStyle("-fx-border-color: red;");
                isMinZoomOK = false;
                checkSettings();
            }
        }
    }

    private void setAllTexts() {
        playSpeedTextField.setText(String.valueOf(playSpeed));
        zoomStepTextField.setText(String.valueOf(zoomStep));
        minZoomTextField.setText(String.valueOf(zoomMin));
        maxZoomTextField.setText(String.valueOf(zoomMax));
    }

    private void checkSettings() {
        if (isPlaySpeedOK && isMaxZoomOK && isMinZoomOK) {
            if (playSpeed == playSpeedTmp && zoomMax == zoomMaxTmp &&
zoomMin == zoomMinTmp && zoomStep == zoomStepTmp && typeOfSize ==
typeOfSizeTmp) {
                okBtn.setDisable(false);
                applyBtn.setDisable(true);
            } else {
                okBtn.setDisable(false);
                applyBtn.setDisable(false);
            }
        } else {
            okBtn.setDisable(true);
            applyBtn.setDisable(true);
        }
    }

    public void setApp(App app) {
        myApp = app;
    }

    public void setData(int playSpeed, double zoomMax, double zoomMin,
double zoomStep, int typeOfSize) {
        this.playSpeed = playSpeed;
        this.zoomMax = zoomMax;
        this.zoomMin = zoomMin;
```

```
        this.zoomStep = zoomStep;
        this.typeOfSize = typeOfSize;

        playSpeedTmp = this.playSpeed;
        zoomStepTmp = this.zoomStep;
        zoomMaxTmp = this.zoomMax;
        zoomMinTmp = this.zoomMin;
        typeOfSizeTmp = this.typeOfSize;

        isClosed = true;
        setAllTexts();
        applyBtn.setDisable(true);

        playSpeedTextField.setStyle("-fx-border-color: green;");
        zoomStepTextField.setStyle("-fx-border-color: green;");
        minZoomTextField.setStyle("-fx-border-color: green;");
        maxZoomTextField.setStyle("-fx-border-color: green;");

        switch (this.typeOfSize) {
            case 1 -> fieldSizeToggleGroup.selectToggle(smallBtn);
            case 2 -> fieldSizeToggleGroup.selectToggle(normalBtn);
            case 3 -> fieldSizeToggleGroup.selectToggle(hugeBtn);
        }
    }

    public boolean getClosed() {
        return isClosed;
    }

    public void setClosed(boolean b) {
        isClosed = b;
    }
}
```

DirectedGraph.java

```
package ru.etu.graph;

import java.util.List;

/**
 * Directed Graph is a set of vertices connected by edges. Edges have
direction.
 *
 * @see Graph
 * @see Edge
 * @see Vertex
 */
public interface DirectedGraph extends Graph {

    /**
```

```
     * Finds and returns list of incident edges that start from
<code>vertexFrom</code>
     *
     * @param vertexFrom vertex from which to find incident edges that
directed from it.
     * @return list of incident edges that directed from
<code>vertexFrom</code>
     * @throws InvalidVertexException if vertex is invalid or does not
belong to the graph
     */
    @Override
    List<Edge> incidentEdges(Vertex vertexFrom) throws
InvalidVertexException;

    /**
     * Finds and returns list of incident edges that finish in
<code>vertexTo</code>
     *
     * @param vertexTo vertex from which to find incident edges that
finish in it
     * @return list of incident edges that finish in
<code>vertexTo</code>
     * @throws InvalidVertexException if vertex is invalid or does not
belong to the graph
     */
    List<Edge> inboundEdges(Vertex vertexTo) throws
InvalidVertexException;

    /**
     * Determines if two vertices are connected with any edge
     *
     * @param vertexOut start vertex
     * @param vertexIn  finish vertex
     * @return true if connected, false - otherwise
     * @throws InvalidVertexException if any of the vertices is invalid
or does not belong to the graph
     */
    @Override
    boolean areConnected(Vertex vertexOut, Vertex vertexIn) throws
InvalidEdgeException;

    /**
     * Inserts new edge to the graph
     *
     * @param vertexOut   start vertex link
     * @param vertexIn    finish vertex link
     * @param edgeElement data to be stored in the edge
     * @return link to newly created edge
     * @throws InvalidVertexException if any of the vertices is invalid
or does not belong to the graph
```

```
     * @throws InvalidEdgeException   if there is already an edge with
the same data and vertices
     */
    @Override
    Edge insertEdge(Vertex vertexOut, Vertex vertexIn, int edgeElement)
throws InvalidVertexException, InvalidEdgeException;

    /**
     * Inserts new edge to the graph
     *
     * @param vertElement1 start vertex data
     * @param vertElement2 finish vertex data
     * @param edgeElement  data to be stored in the edge
     * @return link to newly created edge
     * @throws InvalidVertexException if any of the vertices is invalid
or does not belong to the graph
     * @throws InvalidEdgeException   if there is already an edge with
the same data and vertices
     */
    @Override
    Edge insertEdge(String vertElement1, String vertElement2, int
edgeElement) throws InvalidVertexException, InvalidEdgeException;
}
```

DirectedGraphList.java

```
package ru.etu.graph;

import java.util.*;


public class DirectedGraphList implements DirectedGraph {

    // This implementation does not allow duplicates of vertices
    // Allows max two edge between any 2 vertices. But they should be
multi-directional.
    private final Map<String, Vertex> vertices;
    private final List<Edge> edges;

    public DirectedGraphList() {
        this.vertices = new HashMap<>();
        this.edges = new ArrayList<>();
    }

    @Override
    public int verticesNum() {
        return vertices.size();
    }

    @Override
    public int edgesNum() {
```

```
        return edges.size();
    }

    @Override
    public List<Vertex> getVertices() {
        return new ArrayList<>(vertices.values());
    }

    @Override
    public List<Edge> getEdges() {
        return new ArrayList<>(edges);
    }

    @Override
    public List<Edge> incidentEdges(Vertex vertexFrom) throws
InvalidVertexException {
        checkVertex(vertexFrom);

        List<Edge> incidentEdges = new ArrayList<>();

        for (var edge : edges) {
            if (edge.getVertexOutbound().equals(vertexFrom)) {
                incidentEdges.add(edge);
            }
        }

        return incidentEdges;
    }

    @Override
    public List<Edge> inboundEdges(Vertex vertexTo) throws
InvalidVertexException {
        checkVertex(vertexTo);

        List<Edge> incidentEdges = new ArrayList<>();
        for (var edge : edges) {

            if (edge.getVertexInbound().equals(vertexTo)) {
                incidentEdges.add(edge);
            }
        }
        return incidentEdges;
    }

    @Override
    public Vertex opposite(Vertex vertex, Edge edge) throws
InvalidVertexException, InvalidEdgeException {
        checkVertex(vertex);
        checkEdge(edge);
```

```
        if (!edge.contains(vertex)) {
            return null;
        }

        if (edge.getVertices().getKey().equals(vertex)) {
            return edge.getVertices().getValue();
        } else {
            return edge.getVertices().getKey();
        }
    }

    @Override
    public boolean areConnected(Vertex vertexOut, Vertex vertexIn) throws
InvalidEdgeException {
        checkVertex(vertexIn);
        checkVertex(vertexOut);

        if (vertexOut.equals(vertexIn)) {
            return false;
        }

        for (var edge : edges) {
            if (edge.getVertexOutbound().equals(vertexOut) &&
edge.getVertexInbound().equals(vertexIn)) {
                return true;
            }
        }

        return false;
    }

    @Override
    public Vertex insertVertex(String element) throws
InvalidVertexException {
        if (element == null) throw new IllegalArgumentException("Cannot
create vertex with 'null' as name.");

        checkNoVertexWithData(element);

        Vertex newVertex = new Vertex(element);

        vertices.put(element, newVertex);

        return newVertex;
    }

    @Override
    public Edge insertEdge(Vertex vertexOut, Vertex vertexIn, int
edgeElement) throws InvalidVertexException, InvalidEdgeException {
```

```
        checkVertex(vertexOut);
        checkVertex(vertexIn);

        if (vertexOut.equals(vertexIn)) {
            throw new InvalidVertexException("Cannot make loop edge.");
        }

        Edge newEdge = new Edge(edgeElement, vertexOut, vertexIn);

        checkEdgeDoesNotExists(newEdge);

        edges.add(newEdge);

        return newEdge;
    }

    @Override
    public Edge insertEdge(String vertElement1, String vertElement2, int
edgeElement) throws InvalidVertexException, InvalidEdgeException {

        if (vertElement1 == null) throw new
IllegalArgumentException("Cannot get vertex with 'null' as name.");
        if (vertElement2 == null) throw new
IllegalArgumentException("Cannot get vertex with 'null' as name.");

        checkHasVertexWithData(vertElement1);
        checkHasVertexWithData(vertElement2);

        Vertex outVertex = getVertex(vertElement1);
        Vertex inVertex = getVertex(vertElement2);

        if (vertElement1.equals(vertElement2)) {
            throw new InvalidVertexException("Cannot make loop edge.");
        }

        Edge newEdge = new Edge(edgeElement, outVertex, inVertex);

        checkEdgeDoesNotExists(newEdge);

        edges.add(newEdge);

        return newEdge;
    }

    @Override
    public String removeVertex(Vertex vertex) throws
InvalidVertexException {
        checkVertex(vertex);

        String element = vertex.getData();
```

```
        //remove incident edges (inbound and outbound)
        Collection<Edge> inOutEdges = incidentEdges(vertex);
        inOutEdges.addAll(inboundEdges(vertex));

        for (var edge : inOutEdges) {
            edges.remove(edge);
        }

        vertices.remove(vertex.getData());

        return element;
    }

    @Override
    public int removeEdge(Edge edge) throws InvalidEdgeException {
        checkEdge(edge);

        int element = edge.getData();
        edges.remove(edge);

        return element;
    }

    @Override
    public Edge getEdge(Vertex vertexFrom, Vertex vertexTo) {
        checkVertex(vertexFrom);
        checkVertex(vertexTo);

        for (var edge : edges) {
            if (edge.getVertexOutbound().equals(vertexFrom) &&
edge.getVertexInbound().equals(vertexTo)) {
                return edge;
            }
        }
        return null;
    }

    @Override
    public Edge getEdge(String vertexElFrom, String vertexElTo) {
        if (vertexElFrom == null || vertexElTo == null)
            throw new IllegalArgumentException("Vertex name cannot be
null.");

        checkHasVertexWithData(vertexElFrom);
        checkHasVertexWithData(vertexElTo);

        var vertexFrom = getVertex(vertexElFrom);
        var vertexTo = getVertex(vertexElTo);
```

```
        for (var edge : edges) {
            if (edge.getVertexOutbound().equals(vertexFrom) &&
edge.getVertexInbound().equals(vertexTo)) {
                return edge;
            }
        }
        return null;
    }

    @Override
    public Vertex getVertex(String vertexEl) {
        if (vertexEl == null)
            throw new IllegalArgumentException("Vertex name cannot be
null.");

        checkHasVertexWithData(vertexEl);

        for (Vertex vertex : vertices.values()) {
            if (vertex.getData().equals(vertexEl)) {
                return vertex;
            }
        }
        return null;
    }

    @Override
    public String toString() {
        var builder = new StringBuilder();

        builder.append("Graph (oriented) with ").append(verticesNum())
                .append(" vertices and ").append(edgesNum()).append("
edges:\n");

        builder.append("Vertices: \n");
        for (var vertex : vertices.values()) {
            builder.append("\t").append(vertex.toString()).append("\n");
        }

        builder.append("Edges: \n");
        for (var edge : edges) {
            builder.append("\t").append(edge.toString()).append("\n");
        }


        return builder.toString();
    }


    private void checkVertex(Vertex vertex) throws InvalidVertexException
{
```

```
        if (vertex == null) throw new InvalidVertexException("Vertex is
null.");

        if (!vertices.containsKey(vertex.getData())) {
            throw new InvalidVertexException("Vertex does not belong to
this graph.");
        }
    }

    private void checkEdge(Edge edge) throws InvalidEdgeException {
        if (edge == null) throw new InvalidEdgeException("Edge is
null.");

        for (var graphEdge : edges) {
            var edgeVal = graphEdge.getData();

            if (edgeVal == edge.getData() &&

(graphEdge.getVertexInbound().equals(edge.getVertexInbound()) &&

graphEdge.getVertexOutbound().equals(edge.getVertexOutbound()))) {
                return;
            }
        }

        throw new InvalidEdgeException("Edge does not belong to this
graph.");
    }

    private void checkNoVertexWithData(String data) throws
InvalidVertexException {
        if (vertices.containsKey(data)) {
            throw new InvalidVertexException("There's already a vertex
with this element.");
        }
    }

    private void checkHasVertexWithData(String data) throws
InvalidVertexException {
        if (!vertices.containsKey(data)) {
            throw new InvalidVertexException("No vertex contains " +
data.toString());
        }
    }

    private void checkEdgeDoesNotExists(Edge edge) throws
InvalidEdgeException {

        for (var edgeObj : edges) {
```

```
            if
((edgeObj.getVertexInbound().equals(edge.getVertexInbound()) &&
edgeObj.getVertexOutbound().equals(edge.getVertexOutbound()))) {
                throw new InvalidEdgeException("There's already a edge
between these vertices (in direction '" +
                        edge.getVertexOutbound() + "' -> '" +
edge.getVertexInbound() + "').");
            }
        }
    }

    private void checkHasEdgeWithData(int data) throws
InvalidEdgeException {
        boolean contains = false;
        for (var graphEdge : edges) {
            if (graphEdge.getData() == data) {
                contains = true;
                break;
            }
        }
        if (!contains) {
            throw new InvalidEdgeException("No edge contains " + data);
        }
    }
}
```

Edge.java

```
package ru.etu.graph;

import javafx.util.Pair;

import java.util.Objects;

/**
 * Class of Edge
 *
 * @see Vertex
 */
public class Edge {

    private int data;
    private final Vertex vertexOutbound;
    private final Vertex vertexInbound;

    public Edge(int data, Vertex vertexOutbound, Vertex vertexInbound) {
        this.data = data;
        this.vertexOutbound = vertexOutbound;
        this.vertexInbound = vertexInbound;
    }
```

```
    public boolean contains(Vertex vertex) {
        return Objects.equals(vertexOutbound.getData(), vertex.getData())
|| Objects.equals(vertexInbound.getData(), vertex.getData());
    }

    /**
     * Returns the pair of vertices (from and to)
     *
     * @return pair of vertices
     */
    public Pair<Vertex, Vertex> getVertices() {
        return new Pair<>(vertexOutbound, vertexInbound);
    }

    public void setData(int data) {
        this.data = data;
    }

    public int getData() {
        return data;
    }

    public Vertex getVertexOutbound() {
        return vertexOutbound;
    }

    public Vertex getVertexInbound() {
        return vertexInbound;
    }

    @Override
    public String toString() {
        return "Edge{" +
                "element=" + data +
                ", vertexOutbound=" + vertexOutbound +
                ", vertexInbound=" + vertexInbound +
                '}';
    }
}
```

Graph.java

```
package ru.etu.graph;

import java.util.List;

/**
 * Graph is a set of vertices connected by edges. Edges have no
direction.
 *
 * @see Edge
```

```
 * @see Vertex
 */
public interface Graph {

    /**
     * Method that returns number of vertices in the graph
     *
     * @return number of vertices in the graph
     */
    int verticesNum();

    /**
     * Method that returns number of edges in the graph
     *
     * @return number of edges in the graph
     */
    int edgesNum();

    /**
     * Method that returns list of vertices
     *
     * @return list of vertices
     */
    List<Vertex> getVertices();

    /**
     * Method that returns list of edges
     *
     * @return list of edges
     */
    List<Edge> getEdges();

    /**
     * Finds and returns list of incident edges for <code>vertex</code>
     * Incident - originates from here (In oriented graph edge originates
from both ends.
     * So it may be both in outbound or in inbound property of the edge)
     *
     * @param vertex vertex from which to find incident edges
     * @return list of incident edges
     * @throws InvalidVertexException if vertex is invalid or does not
belong to the graph
     */
    List<Edge> incidentEdges(Vertex vertex) throws
InvalidVertexException;

    /**
     * Returns opposite vertex from edge.
     *
     * @param vertex the first vertex on the edge
```

```
     * @param edge   the edge itself
     * @return opposite vertex for current edge
     * @throws InvalidVertexException if vertex is invalid or does not
belong to the graph
     * @throws InvalidEdgeException   if edge is invalid or does not
belong to the graph
     */
    Vertex opposite(Vertex vertex, Edge edge) throws
InvalidVertexException, InvalidEdgeException;

    /**
     * Determines if two vertices are connected with any edge
     *
     * @param vertex1 first vertex
     * @param vertex2 second vertex
     * @return true if connected, false - otherwise
     * @throws InvalidVertexException if any of the vertices is invalid
or does not belong to the graph
     */
    boolean areConnected(Vertex vertex1, Vertex vertex2) throws
InvalidVertexException;

    /**
     * Inserts new vertex to the graph
     *
     * @param element data for new vertex
     * @return link to newly created vertex
     * @throws InvalidVertexException if there is already a vertex with
the same data
     */
    Vertex insertVertex(String element) throws InvalidVertexException;

    /**
     * Inserts new edge to the graph
     *
     * @param vertex1     first vertex link
     * @param vertex2     second vertex link
     * @param edgeElement data to be stored in the edge
     * @return link to newly created edge
     * @throws InvalidVertexException if any of the vertices is invalid
or does not belong to the graph
     * @throws InvalidEdgeException   if there is already an edge with
the same data and vertices
     */
    Edge insertEdge(Vertex vertex1, Vertex vertex2, int edgeElement)
throws InvalidVertexException, InvalidEdgeException;

    /**
     * Inserts new edge to the graph
     *
```

```
     * @param vertElement1 first vertex data
     * @param vertElement2 second vertex data
     * @param edgeElement  data to be stored in the edge
     * @return link to newly created edge
     * @throws InvalidVertexException if any of the vertices is invalid
or does not belong to the graph
     * @throws InvalidEdgeException   if there is already an edge with
the same data and vertices
     */
    Edge insertEdge(String vertElement1, String vertElement2, int
edgeElement) throws InvalidVertexException, InvalidEdgeException;

    /**
     * Removes vertex from graph
     *
     * @param vertex link to the vertex to delete
     * @return data from removed vertex
     * @throws InvalidVertexException if vertex is invalid or does not
belong to the graph
     */
    String removeVertex(Vertex vertex) throws InvalidVertexException;

    /**
     * Removes edge from graph
     *
     * @param edge link to the edge to delete
     * @return data from removed edge
     * @throws InvalidEdgeException if edge is invalid or does not belong
to the graph
     */
    int removeEdge(Edge edge) throws InvalidEdgeException;

    Edge getEdge(Vertex vertexFrom, Vertex vertexTo);

    Edge getEdge(String vertexElFrom, String vertexElTo);

    Vertex getVertex(String vertexEl);
}
```

GraphList.java

```
package ru.etu.graph;

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

public class GraphList implements Graph {

    // This implementation does not allow duplicates of vertices
```

```
    // Allows only one edge between any 2 vertices.
    private final Map<String, Vertex> vertices;
    private final List<Edge> edges;

    public GraphList() {
        vertices = new HashMap<>();
        edges = new ArrayList<>();
    }

    @Override
    public int verticesNum() {
        return vertices.size();
    }

    @Override
    public int edgesNum() {
        return edges.size();
    }

    @Override
    public List<Vertex> getVertices() {
        return new ArrayList<>(vertices.values());
    }

    @Override
    public List<Edge> getEdges() {
        return new ArrayList<>(edges);
    }

    @Override
    public List<Edge> incidentEdges(Vertex vertex) throws
InvalidVertexException {

        checkVertex(vertex);

        List<Edge> incidentEdges = new ArrayList<>();

        for (Edge edge : edges) {
            if (edge.contains(vertex)) {
                incidentEdges.add(edge);
            }
        }

        return incidentEdges;
    }

    @Override
    public Vertex opposite(Vertex vertex, Edge edge) throws
InvalidVertexException, InvalidEdgeException {
```

```
        checkVertex(vertex);
        checkEdge(edge);

        if (!edge.contains(vertex)) {
            return null;
        }

        if (edge.getVertices().getKey().equals(vertex)) {
            return edge.getVertices().getValue();
        } else {
            return edge.getVertices().getKey();
        }
    }

    @Override
    public boolean areConnected(Vertex vertex1, Vertex vertex2) throws
InvalidVertexException {
        checkVertex(vertex1);
        checkVertex(vertex2);

        if (vertex1.equals(vertex2)) {
            return false;
        }

        for (Edge edge : edges) {
            if (edge.contains(vertex2) && edge.contains(vertex1)) {
                return true;
            }
        }

        return false;
    }

    @Override
    public Vertex insertVertex(String element) throws
InvalidVertexException {

        if (element == null) throw new IllegalArgumentException("Cannot
create vertex with 'null' as name.");

        checkNoVertexWithData(element);

        Vertex newVertex = new Vertex(element);

        vertices.put(element, newVertex);

        return newVertex;
    }

    @Override
```

```
    public Edge insertEdge(Vertex vertex1, Vertex vertex2, int
edgeElement) throws InvalidVertexException, InvalidEdgeException {

        checkVertex(vertex1);
        checkVertex(vertex2);

        if (vertex1.equals(vertex2)) {
            throw new InvalidVertexException("Cannot make loop edge.");
        }

        Edge newEdge = new Edge(edgeElement, vertex1, vertex2);

        checkEdgeDoesNotExists(newEdge);

        edges.add(newEdge);

        return newEdge;
    }

    @Override
    public Edge insertEdge(String vertElement1, String vertElement2, int
edgeElement) throws InvalidVertexException, InvalidEdgeException {

        if (vertElement1 == null) throw new
IllegalArgumentException("Cannot get vertex with 'null' as name.");
        if (vertElement2 == null) throw new
IllegalArgumentException("Cannot get vertex with 'null' as name.");

        checkHasVertexWithData(vertElement1);
        checkHasVertexWithData(vertElement2);

        if (vertElement1.equals(vertElement2)) {
            throw new InvalidVertexException("Cannot make loop edge.");
        }

        Vertex outVertex = getVertex(vertElement1);
        Vertex inVertex = getVertex(vertElement2);

        Edge newEdge = new Edge(edgeElement, outVertex, inVertex);

        checkEdgeDoesNotExists(newEdge);

        edges.add(newEdge);

        return newEdge;
    }


    @Override
```

```
    public String removeVertex(Vertex vertex) throws
InvalidVertexException {
        checkVertex(vertex);

        String element = vertex.getData();

        for (var edge : incidentEdges(vertex)) {
            edges.remove(edge);
        }

        vertices.remove(vertex.getData());

        return element;
    }

    @Override
    public int removeEdge(Edge edge) throws InvalidEdgeException {
        checkEdge(edge);

        int element = edge.getData();
        edges.remove(edge);

        return element;
    }

    @Override
    public Edge getEdge(Vertex vertexFrom, Vertex vertexTo) {
        checkVertex(vertexFrom);
        checkVertex(vertexTo);

        for (var edge : edges) {
            if (edge.contains(vertexFrom) && edge.contains(vertexTo)) {
                return edge;
            }
        }
        return null;
    }

    @Override
    public Edge getEdge(String vertexElFrom, String vertexElTo) {
        if (vertexElFrom == null || vertexElTo == null)
            throw new IllegalArgumentException("Vertex name cannot be
null.");

        checkHasVertexWithData(vertexElFrom);
        checkHasVertexWithData(vertexElTo);

        var vertexFrom = getVertex(vertexElFrom);
        var vertexTo = getVertex(vertexElTo);
```

```
        for (var edge : edges) {
            if (edge.contains(vertexFrom) && edge.contains(vertexTo)) {
                return edge;
            }
        }
        return null;
    }

    @Override
    public Vertex getVertex(String vertexEl) {
        if (vertexEl == null)
            throw new IllegalArgumentException("Vertex name cannot be
null.");

        checkHasVertexWithData(vertexEl);

        for (Vertex vertex : vertices.values()) {
            if (vertex.getData().equals(vertexEl)) {
                return vertex;
            }
        }
        return null;
    }

    @Override
    public String toString() {

        var builder = new StringBuilder();

        builder.append("Graph (not oriented) with
").append(verticesNum())
                .append(" vertices and ").append(edgesNum()).append("
edges:\n");

        builder.append("Vertices: \n");
        for (var vertex : vertices.values()) {
            builder.append("\t").append(vertex.toString()).append("\n");
        }

        builder.append("Edges: \n");
        for (var edge : edges) {
            builder.append("\t").append(edge.toString()).append("\n");
        }


        return builder.toString();
    }
```

```
    private void checkVertex(Vertex vertex) throws InvalidVertexException
{
        if (vertex == null) throw new InvalidVertexException("Vertex is
null.");

        if (!vertices.containsKey(vertex.getData())) {
            throw new InvalidVertexException("Vertex does not belong to
this graph.");
        }
    }

    private void checkEdge(Edge edge) throws InvalidEdgeException {
        if (edge == null) throw new InvalidEdgeException("Edge is
null.");

        for (var edgeObj : edges) {
            var edgeVal = edgeObj.getData();

            if (edgeVal == edge.getData() &&

((edgeObj.getVertexInbound().getData().equals(edge.getVertexInbound().get
Data()) &&

edgeObj.getVertexOutbound().getData().equals(edge.getVertexOutbound().get
Data())) ||

(edgeObj.getVertexInbound().getData().equals(edge.getVertexOutbound().get
Data()) &&

edgeObj.getVertexOutbound().getData().equals(edge.getVertexInbound().getD
ata()))))) {
                return;
            }
        }

        throw new InvalidEdgeException("Edge does not belong to this
graph.");
    }

    private void checkNoVertexWithData(String data) throws
InvalidVertexException {
        if (vertices.containsKey(data)) {
            throw new InvalidVertexException("There's already a vertex
with this element.");
        }
    }

    private void checkHasVertexWithData(String data) throws
InvalidVertexException {
        if (!vertices.containsKey(data)) {
```

```
            throw new InvalidVertexException("No vertex contains " +
data);
        }
    }

    private void checkEdgeDoesNotExists(Edge edge) throws
InvalidEdgeException {

        for (var edgeObj : edges) {
            if
((edgeObj.getVertexInbound().equals(edge.getVertexInbound()) &&
edgeObj.getVertexOutbound().equals(edge.getVertexOutbound())) ||

(edgeObj.getVertexInbound().equals(edge.getVertexOutbound()) &&
edgeObj.getVertexOutbound().equals(edge.getVertexInbound()))) {
                throw new InvalidEdgeException("There's already a edge
between these vertices.");
            }
        }
    }

    private void checkHasEdgeWithData(int data) throws
InvalidEdgeException {

        for (var edgeObj : edges) {
            var edgeVal = edgeObj.getData();

            if (edgeVal == data) {
                return;
            }
        }

        throw new InvalidEdgeException("No edge contains " + data);

    }
}
```

InvalidEdgeException.java

```
package ru.etu.graph;

/**
 * Exception for invalid edge. "Invalid" means that it does not belong to
needed graph or any other problem.
 */
public class InvalidEdgeException extends RuntimeException {

    public InvalidEdgeException() {
        super("The edge is invalid or does not belong to the graph.");
    }
```

```
    public InvalidEdgeException(String message) {
        super(message);
    }
}
```

InvalidVertexException.java

```
package ru.etu.graph;

/**
 * Exception for invalid vertex. "Invalid" means that it does not belong
to needed graph or any other problem.
 */
public class InvalidVertexException extends RuntimeException {

    public InvalidVertexException() {
        super("The vertex is invalid or does not belong to the graph.");
    }

    public InvalidVertexException(String message) {
        super(message);
    }

}
```

Vertex.java

```
package ru.etu.graph;

import java.util.Objects;

/**
 * Vertex class
 */
public class Vertex {

    private String data;

    public Vertex(String data) {
        this.data = data;
    }

    public void setData(String data) {
        this.data = data;
    }

    public String getData() {
        return data;
    }

    @Override
```

```
    public String toString() {
        return "Vertex{" +
                "element='" + data + '\'' +
                '}';
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof Vertex)) return false;
        Vertex vertex = (Vertex) o;
        return getData().equals(vertex.getData());
    }

    @Override
    public int hashCode() {
        return Objects.hash(getData());
    }
}
```

Arrow.java

```
package ru.etu.graphview.base;

import javafx.scene.shape.LineTo;
import javafx.scene.shape.MoveTo;
import javafx.scene.shape.Path;
import ru.etu.graphview.styling.Stylable;
import ru.etu.graphview.styling.StyleEngine;

/**
 * Class for Arrow View. Basically it is two lines that was drawn
manually.
 * This class implements Stylable interface. Therefore, its style can be
changed conveniently by methods from this interface.
 *
 * @see Stylable
 */
public class Arrow extends Path implements Stylable {

    /**
     * Style engine for simpler implementation of Stylable interface
     */
    private final StyleEngine styleEngine;

    public Arrow(double size) {
        getElements().add(new MoveTo(0, 0));
        getElements().add(new LineTo(-size, size));
        getElements().add(new MoveTo(0, 0));
        getElements().add(new LineTo(-size, -size));
```

```
        styleEngine = new StyleEngine(this);
        setStyleClass("arrow");
    }

    @Override
    public void setStyleCss(String css) {
        styleEngine.setStyleCss(css);
    }

    @Override
    public void setStyleClass(String cssClassName) {
        styleEngine.setStyleClass(cssClassName);
    }

    @Override
    public void addStyleClass(String cssClassName) {
        styleEngine.addStyleClass(cssClassName);
    }

    @Override
    public boolean removeStyleClass(String cssClassName) {
        return styleEngine.removeStyleClass(cssClassName);
    }
}
```

FXEdge.java

```
package ru.etu.graphview.base;

import ru.etu.graph.Edge;
import ru.etu.graphview.styling.Stylable;

/**
 * Base interface for Edge View implementation.
 * Can be labeled and conveniently styled.
 *
 * @see LabelledNode
 * @see Stylable
 */
public interface FXEdge extends LabelledNode, Stylable {

    /**
     * Method, that gets stored edge, represented by this view
     *
     * @return edge link, stored inside
     */
    Edge getEdge();

    /**
     * Sets edge to be represented by this view
     *
```

```
     * @param edge edge link
     */
    void setEdge(Edge edge);

    /**
     * Saves arrow and setups its coordinates and rotation for proper
view.
     *
     * @param arrow arrow link
     */
    void setArrow(Arrow arrow);

    /**
     * Gets arrow link, stored inside
     *
     * @return arrow link, stored inside
     */
    Arrow getArrow();


}
```

FXEdgeCurve.java

```
package ru.etu.graphview.base;

import javafx.geometry.Point2D;
import javafx.scene.control.TextField;
import javafx.scene.shape.QuadCurve;
import javafx.scene.transform.Rotate;
import javafx.scene.transform.Translate;
import ru.etu.graph.Edge;
import ru.etu.graphview.styling.StyleEngine;

/**
 * Implementation of FXEdge interface with curve line.
 *
 * @see FXEdge
 */
public class FXEdgeCurve extends QuadCurve implements FXEdge {

    private double angle;
    private final int bendingCoefficient; // Can be either -1 or 1. Sets
side of bending

    private final StyleEngine styleEngine;

    private Edge edge;

    private final FXVertexNode inboundVertex;
    private final FXVertexNode outboundVertex;
```

```
    private Label label = null;
    private TextField labelField = null;
    private Arrow arrow = null;


    // Used, when edge creation will be done after curve creation.
    public FXEdgeCurve(FXVertexNode outboundVertex, FXVertexNode
inboundVertex, int bendingCoefficient, double angle) {
        if (inboundVertex == null || outboundVertex == null) {
            throw new IllegalArgumentException("Edge should have both
inbound and outbound vertex.");
        }

        this.angle = angle;

        this.bendingCoefficient = bendingCoefficient;

        this.inboundVertex = inboundVertex;
        this.outboundVertex = outboundVertex;

        this.startXProperty().bind(outboundVertex.centerXProperty());
        this.startYProperty().bind(outboundVertex.centerYProperty());

        this.endXProperty().bind(inboundVertex.centerXProperty());
        this.endYProperty().bind(inboundVertex.centerYProperty());

        update();
        enableListeners();

        this.styleEngine = new StyleEngine(this);
        styleEngine.setStyleClass("curved-edge");

    }

    public FXEdgeCurve(Edge edge, FXVertexNode outboundVertex,
FXVertexNode inboundVertex, int bendingCoefficient, double angle) {
        if (inboundVertex == null || outboundVertex == null) {
            throw new IllegalArgumentException("Edge should have both
inbound and outbound vertex.");
        }

        this.angle = angle;
        this.bendingCoefficient = bendingCoefficient;

        if (bendingCoefficient != 1 && bendingCoefficient != -1) {
            throw new IllegalArgumentException("bendingCoefficient should
be either 1 or -1.");
        }
```

```
        this.edge = edge;

        this.inboundVertex = inboundVertex;
        this.outboundVertex = outboundVertex;

        this.startXProperty().bind(outboundVertex.centerXProperty());
        this.startYProperty().bind(outboundVertex.centerYProperty());

        this.endXProperty().bind(inboundVertex.centerXProperty());
        this.endYProperty().bind(inboundVertex.centerYProperty());

        update();
        enableListeners();

        this.styleEngine = new StyleEngine(this);
        styleEngine.setStyleClass("curved-edge");

    }

    /**
     * This method updates control point with new start and finish
points.
     */
    private void update() {
        Point2D midPoint = new Point2D((outboundVertex.getCenterX() +
inboundVertex.getCenterX()) / 2,
                (outboundVertex.getCenterY() +
inboundVertex.getCenterY()) / 2);

        Point2D startPoint = new Point2D(outboundVertex.getCenterX(),
outboundVertex.getCenterY());
        Point2D endPoint = new Point2D(inboundVertex.getCenterX(),
inboundVertex.getCenterY());


        double distance = startPoint.distance(endPoint);

        var angle1 = angle - (distance * angle / 1700);

        Point2D midPoint1 = Utils.rotate(midPoint, (bendingCoefficient ==
1) ? startPoint : endPoint, angle1 * bendingCoefficient);
        Point2D midPoint2 = Utils.rotate(midPoint, (bendingCoefficient ==
1) ? endPoint : startPoint, -angle1 * bendingCoefficient);


        setControlX((midPoint1.getX() + midPoint2.getX()) / 2);
        setControlY((midPoint1.getY() + midPoint2.getY()) / 2);
    }

    /**
```

```
     * Sets calling update() method, when any property was changed.
     */
    private void enableListeners() {
        this.startXProperty().addListener((ov, t, t1) -> update());
        this.startYProperty().addListener((ov, t, t1) -> update());
        this.endXProperty().addListener((ov, t, t1) -> update());
        this.endYProperty().addListener((ov, t, t1) -> update());
    }

    @Override
    public Edge getEdge() {
        return edge;
    }

    @Override
    public void setEdge(Edge edge) {
        this.edge = edge;
    }

    @Override
    public void setArrow(Arrow arrow) {
        this.arrow = arrow;

        arrow.translateXProperty().bind(endXProperty());
        arrow.translateYProperty().bind(endYProperty());

        Rotate rotation = new Rotate();
        rotation.pivotXProperty().bind(translateXProperty());
        rotation.pivotYProperty().bind(translateYProperty());
        rotation.angleProperty().bind(Utils.toDegrees(Utils.atan2(
                endYProperty().subtract(controlYProperty()),
                endXProperty().subtract(controlXProperty())
        )));

        arrow.getTransforms().add(rotation);

        arrow.getTransforms().add(new Translate(-
inboundVertex.getRadius(), 0));
    }

    @Override
    public Arrow getArrow() {
        return arrow;
    }

    @Override
    public void setLabel(Label label) {
        this.label = label;

        label.setStyleClass("edge-label");
```

```
label.xProperty().bind(controlXProperty().add(controlXProperty().add(star
tXProperty().add(endXProperty()).divide(2)).divide(2)).divide(2).subtract
(label.getLayoutBounds().getWidth() / 2));

label.yProperty().bind(controlYProperty().add(controlYProperty().add(star
tYProperty().add(endYProperty()).divide(2)).divide(2)).divide(2).add(labe
l.getLayoutBounds().getHeight() / 2));
    }

    @Override
    public Label getLabel() {
        return label;
    }

    @Override
    public void setLabelField(TextField labelField) {
        this.labelField = labelField;


labelField.translateXProperty().bind(controlXProperty().add(controlXPrope
rty().add(startXProperty().add(endXProperty()).divide(2)).divide(2)).divi
de(2).subtract(70 / 2));

labelField.translateYProperty().bind(controlYProperty().add(controlYPrope
rty().add(startYProperty().add(endYProperty()).divide(2)).divide(2)).divi
de(2).add(15 / 2));
    }

    @Override
    public TextField getLabelField() {
        return labelField;
    }

    @Override
    public void setStyleCss(String css) {
        styleEngine.setStyleCss(css);
    }

    @Override
    public void setStyleClass(String cssClassName) {
        styleEngine.setStyleClass(cssClassName);
    }

    @Override
    public void addStyleClass(String cssClassName) {
        styleEngine.addStyleClass(cssClassName);
    }
```

```
    @Override
    public boolean removeStyleClass(String cssClassName) {
        return styleEngine.removeStyleClass(cssClassName);
    }
}
```

FXEdgeLine.java

```
package ru.etu.graphview.base;

import javafx.scene.control.TextField;
import javafx.scene.shape.Line;
import javafx.scene.transform.Rotate;
import javafx.scene.transform.Translate;
import ru.etu.graph.Edge;
import ru.etu.graphview.styling.StyleEngine;

/**
 * Implementation of FXEdge interface with strait line.
 *
 * @see FXEdge
 */
public class FXEdgeLine extends Line implements FXEdge {

    private final StyleEngine styleEngine;

    private Edge edge;

    private final FXVertexNode inboundVertex;
    private final FXVertexNode outboundVertex;

    private Label label = null;
    private TextField labelField = null;
    private Arrow arrow = null;

    // Used, when edge creation will be done after line creation.
    public FXEdgeLine(FXVertexNode outboundVertex, FXVertexNode
inboundVertex) {
        if (inboundVertex == null || outboundVertex == null) {
            throw new IllegalArgumentException("Edge should have both
inbound and outbound vertex.");
        }

        this.inboundVertex = inboundVertex;
        this.outboundVertex = outboundVertex;

        this.startXProperty().bind(outboundVertex.centerXProperty());
        this.startYProperty().bind(outboundVertex.centerYProperty());

        this.endXProperty().bind(inboundVertex.centerXProperty());
        this.endYProperty().bind(inboundVertex.centerYProperty());
```

```
        styleEngine = new StyleEngine(this);
        styleEngine.setStyleClass("edge");
    }

    public FXEdgeLine(Edge edge, FXVertexNode outboundVertex,
FXVertexNode inboundVertex) {
        if (inboundVertex == null || outboundVertex == null) {
            throw new IllegalArgumentException("Edge should have both
inbound and outbound vertex.");
        }

        this.edge = edge;

        this.inboundVertex = inboundVertex;
        this.outboundVertex = outboundVertex;

        this.startXProperty().bind(outboundVertex.centerXProperty());
        this.startYProperty().bind(outboundVertex.centerYProperty());

        this.endXProperty().bind(inboundVertex.centerXProperty());
        this.endYProperty().bind(inboundVertex.centerYProperty());

        styleEngine = new StyleEngine(this);
        styleEngine.setStyleClass("edge");
    }

    @Override
    public Edge getEdge() {
        return edge;
    }

    @Override
    public void setEdge(Edge edge) {
        this.edge = edge;
    }

    @Override
    public void setArrow(Arrow arrow) {
        this.arrow = arrow;

        arrow.translateXProperty().bind(endXProperty());
        arrow.translateYProperty().bind(endYProperty());

        Rotate rotation = new Rotate();
        rotation.pivotXProperty().bind(translateXProperty());
        rotation.pivotYProperty().bind(translateYProperty());
        rotation.angleProperty().bind(Utils.toDegrees(Utils.atan2(
                endYProperty().subtract(startYProperty()),
                endXProperty().subtract(startXProperty())
```

```
        )));

        arrow.getTransforms().add(rotation);

        arrow.getTransforms().add(new Translate(-
outboundVertex.getRadius(), 0));
    }

    @Override
    public Arrow getArrow() {
        return arrow;
    }

    @Override
    public void setLabel(Label label) {
        this.label = label;

        label.setStyleClass("edge-label");


label.xProperty().bind(startXProperty().add(endXProperty()).divide(2).sub
tract(label.getLayoutBounds().getWidth() / 2));

label.yProperty().bind(startYProperty().add(endYProperty()).divide(2).add
(label.getLayoutBounds().getHeight() / 1.5));
    }

    @Override
    public Label getLabel() {
        return label;
    }

    @Override
    public void setLabelField(TextField labelField) {
        this.labelField = labelField;

labelField.translateXProperty().bind(startXProperty().add(endXProperty())
.divide(2).subtract(70 / 2));

labelField.translateYProperty().bind(startYProperty().add(endYProperty())
.divide(2).add(15 / 1.5));
    }

    @Override
    public TextField getLabelField() {
        return labelField;
    }

    @Override
    public void setStyleCss(String css) {
```

```
        styleEngine.setStyleCss(css);
    }

    @Override
    public void setStyleClass(String cssClassName) {
        styleEngine.setStyleClass(cssClassName);
    }

    @Override
    public void addStyleClass(String cssClassName) {
        styleEngine.addStyleClass(cssClassName);
    }

    @Override
    public boolean removeStyleClass(String cssClassName) {
        return styleEngine.removeStyleClass(cssClassName);
    }
}
```

FXVertex.java

```
package ru.etu.graphview.base;

import ru.etu.graph.Vertex;
import ru.etu.graphview.styling.Stylable;

/**
 * Base interface for Vertex View implementation.
 * Can be labeled and conveniently styled.
 *
 * @see Stylable
 * @see LabelledNode
 */
public interface FXVertex extends Stylable, LabelledNode {

    /**
     * Sets vertex to its view
     *
     * @param vertex vertex link
     */
    void setVertex(Vertex vertex);

    /**
     * Method, that gets stored vertex, represented by this view
     *
     * @return vertex link, stored inside
     */
    Vertex getVertex();

    /**
     * Sets position for the vertex
```

```
     *
     * @param x x coordinate
     * @param y y coordinate
     */
    void setPosition(double x, double y);

    /**
     * Returns radius of vertex
     *
     * @return radius
     */
    double getRadius();

    // These two methods are not in the LabelledNode interface, because
they are used only in FXVertex for algorithm data display

    /**
     * Saves label for algorithm and setups its coordinates for proper
view.
     *
     * @param algLabel label for algorithm link
     */
    void setAlgLabel(Label algLabel);

    /**
     * Gets label for algorithm link, stored inside
     *
     * @return label for algorithm link, stored inside
     */
    Label getAlgLabel();
}
```

FXVertexNode.java

```
package ru.etu.graphview.base;

import javafx.animation.Timeline;
import javafx.geometry.Point2D;
import javafx.scene.Cursor;
import javafx.scene.control.TextField;
import javafx.scene.shape.Circle;
import ru.etu.graph.Vertex;
import ru.etu.graphview.styling.StyleEngine;

/**
 * Implementation of FXVertex interface with Circle element.
 *
 * @see FXVertex
 */
public class FXVertexNode extends Circle implements FXVertex {
```

```
    private final StyleEngine styleEngine;

    private Vertex vertex;

    private Label label;
    private Label algLabel;
    private TextField labelField;
    private boolean isDragging;

    // Used, when vertex creation will be done after circle creation.
    public FXVertexNode(double x, double y, double radius) {
        super(x, y, radius);

        enableDrag();

        styleEngine = new StyleEngine(this);
        styleEngine.setStyleClass("vertex");

        this.label = null;
        this.isDragging = false;
    }

    public FXVertexNode(Vertex vertex, double x, double y, double radius)
{
        super(x, y, radius);

        enableDrag();

        styleEngine = new StyleEngine(this);
        styleEngine.setStyleClass("vertex");

        this.vertex = vertex;
        this.label = null;
        this.isDragging = false;
    }

    /**
     * Gets position of the center of this vertex view
     *
     * @return position of the center of this vertex view
     */
    public Point2D getPosition() {
        return new Point2D(getCenterX(), getCenterY());
    }

    @Override
    public void setPosition(double x, double y) {
        if (isDragging) {
            return;
        }
```

```
        setCenterX(x);
        setCenterY(y);
    }

    /**
     * Enables user drag
     */
    public void enableDrag() {
        class Point {
            double x, y;

            public Point(double x, double y) {
                this.x = x;
                this.y = y;
            }
        }

        final Point dragDelta = new Point(0, 0);

        setOnMousePressed(event -> {
            if (event.isPrimaryButtonDown()) {
                dragDelta.x = getCenterX() - event.getX();
                dragDelta.y = getCenterY() - event.getY();

                getScene().setCursor(Cursor.MOVE);
                isDragging = true;

                event.consume();
            }
        });

        setOnMouseReleased(event -> {
            getScene().setCursor(Cursor.HAND);
            isDragging = false;

            event.consume();
        });

        setOnMouseDragged(event -> {
            if (event.isPrimaryButtonDown()) {
                double newX = event.getX() + dragDelta.x;
                double x = boundCenterCoordinate(newX, 0,
getParent().getLayoutBounds().getWidth());
                setCenterX(x);

                double newY = event.getY() + dragDelta.y;
                double y = boundCenterCoordinate(newY, 0,
getParent().getLayoutBounds().getHeight());
                setCenterY(y);
```

```
            }
        });

        setOnMouseEntered(event -> {
            if (!event.isPrimaryButtonDown()) {
                getScene().setCursor(Cursor.HAND);
            }

        });

        setOnMouseExited(event -> {
            if (!event.isPrimaryButtonDown()) {
                getScene().setCursor(Cursor.DEFAULT);
            }
        });
    }

    /**
     * Disables user drag
     */
    public void disableDrag() {
        setOnMousePressed(event -> {
        });

        setOnMouseReleased(event -> {
        });

        setOnMouseDragged(event -> {
        });

        setOnMouseEntered(event -> {
        });

        setOnMouseExited(event -> {
        });
    }

    /**
     * Sets value to fit in boundaries: (min + radius; max - radius)
     *
     * @param value - current value
     * @param min   - min value
     * @param max   - max value
     * @return bound value
     */
    private double boundCenterCoordinate(double value, double min, double
max) {
        double radius = getRadius();

        if (value < min + radius) {
```

```
            return min + radius;
        } else if (value > max - radius) {
            return max - radius;
        } else {
            return value;
        }
    }

    @Override
    public void setVertex(Vertex vertex) {
        this.vertex = vertex;
    }

    @Override
    public Vertex getVertex() {
        return vertex;
    }

    @Override
    public void setLabel(Label label) {
        this.label = label;

        label.setStyleClass("vertex-label");


label.xProperty().bind(centerXProperty().subtract(label.getLayoutBounds()
.getWidth() / 2.0));
        label.yProperty().bind(centerYProperty().add(getRadius() +
label.getLayoutBounds().getHeight()));
    }

    @Override
    public Label getLabel() {
        return label;
    }

    @Override
    public void setAlgLabel(Label algLabel) {
        this.algLabel = algLabel;

        algLabel.setStyleClass("vertex-alg-label");


algLabel.xProperty().bind(centerXProperty().subtract(algLabel.getLayoutBo
unds().getWidth() / 2.0));
        algLabel.yProperty().bind(centerYProperty().subtract(getRadius()
+ algLabel.getLayoutBounds().getHeight() / 2.0));
    }

    @Override
```

```
    public Label getAlgLabel() {
        return algLabel;
    }

    @Override
    public void setLabelField(TextField labelField) {
        this.labelField = labelField;

labelField.translateXProperty().bind(centerXProperty().subtract(70 /
2.0));

labelField.translateYProperty().bind(centerYProperty().add(getRadius() +
8));
    }

    @Override
    public TextField getLabelField() {
        return labelField;
    }

    @Override
    public void setStyleCss(String css) {
        styleEngine.setStyleCss(css);
    }

    @Override
    public void setStyleClass(String cssClassName) {
        styleEngine.setStyleClass(cssClassName);
    }

    @Override
    public void addStyleClass(String cssClassName) {
        styleEngine.addStyleClass(cssClassName);
    }

    @Override
    public boolean removeStyleClass(String cssClassName) {
        return styleEngine.removeStyleClass(cssClassName);
    }
}
```

Label.java

```
package ru.etu.graphview.base;

import javafx.scene.control.Tooltip;
import javafx.scene.text.Text;
import javafx.util.Duration;
import ru.etu.graphview.styling.Stylable;
import ru.etu.graphview.styling.StyleEngine;
```

```
/**
 * This class represent Label to use in vertex and edge views.
 * Implements Stylable interface for easier styling.
 *
 * @see Stylable
 * @see LabelledNode
 */
public class Label extends Text implements Stylable {
    private final StyleEngine styleEngine;

    public Label() {
        this(0, 0, "");
    }

    public Label(String text) {
        this(0, 0, text);
    }

    public Label(double x, double y, String text) {
        super(x, y, "");
        if (text.length() > 4) {
            setText(text.substring(0, 4) + "...");
            Tooltip tooltip = new Tooltip(text);
            tooltip.setShowDelay(Duration.millis(400));
            tooltip.setHideDelay(Duration.millis(200));
            Tooltip.install(this, tooltip);
        } else {
            setText(text);
        }
        styleEngine = new StyleEngine(this);
    }

    @Override
    public void setStyleCss(String css) {
        styleEngine.setStyleCss(css);
    }

    @Override
    public void setStyleClass(String cssClassName) {
        styleEngine.setStyleClass(cssClassName);
    }

    @Override
    public void addStyleClass(String cssClassName) {
        styleEngine.addStyleClass(cssClassName);
    }

    @Override
    public boolean removeStyleClass(String cssClassName) {
        return styleEngine.removeStyleClass(cssClassName);
```

```
    }
}
```

LabelledNode.java

```
package ru.etu.graphview.base;


import javafx.scene.control.TextField;

/**
 * This interface gives some methods to work with labels and textFields.
 *
 * @see Label
 */
public interface LabelledNode {

    /**
     * Saves label and setups its coordinates for proper view.
     *
     * @param label label link
     */
    void setLabel(Label label);

    /**
     * Gets label link, stored inside
     *
     * @return label link, stored inside
     */
    Label getLabel();

    /**
     * Saves text field and setups its coordinates for proper view.
     *
     * @param labelField text field link
     */
    void setLabelField(TextField labelField);

    /**
     * Gets text field link, stored inside
     *
     * @return text field link, stored inside
     */
    TextField getLabelField();
}
```

Utils.java

```
package ru.etu.graphview.base;

import javafx.beans.binding.Bindings;
```

```
import javafx.beans.binding.DoubleBinding;
import javafx.beans.value.ObservableDoubleValue;
import javafx.geometry.Point2D;

/**
 * A bunch of mathematical methods used in base classes
 */
public class Utils {

    /**
     * Transforms radians to the degrees. Change its value alongside with
angleRad.
     *
     * @param angleRad angle in radians
     * @return transformed angle in degrees
     */
    public static DoubleBinding toDegrees(final ObservableDoubleValue
angleRad) {
        return Bindings.createDoubleBinding(() ->
Math.toDegrees(angleRad.get()), angleRad);
    }

    /**
     * Gets angle from x-axis to vector from (0,0) to (x, y) in 2D.
     *
     * @param y y coordinate
     * @param x x coordinate
     * @return angle from x-axis to vector from (0,0) to (x, y) in 2D.
     */
    public static DoubleBinding atan2(final ObservableDoubleValue y,
final ObservableDoubleValue x) {
        return Bindings.createDoubleBinding(() -> Math.atan2(y.get(),
x.get()), y, x);
    }

    /**
     * Rotation of point by some pivot. Based on formula for vector (from
(0,0)) rotation <br>
     * x_2 = cos(angle) * x_1 - sin(angle) * y_1 <br>
     * y_2 = sin(angle) * x_1 + cos(angle) * y_1
     *
     * @param point point which to rotate
     * @param pivot point of rotation
     * @param angle angle of rotation
     * @return coordinates for rotated point
     */
    public static Point2D rotate(final Point2D point, final Point2D
pivot, double angle) {
        var tempAngle = Math.toRadians(angle);
```

```
        double sin = Math.sin(tempAngle);
        double cos = Math.cos(tempAngle);

        // Making vector that starts at (0,0)
        Point2D result = point.subtract(pivot);

        // Rotating it
        Point2D rotatedOrigin = new Point2D(
                result.getX() * cos - result.getY() * sin,
                result.getX() * sin + result.getY() * cos);

        // Making it back
        result = rotatedOrigin.add(pivot);

        return result;
    }
}
```

MyRunnable.java

```
package ru.etu.graphview.drawing.multithreading;

/**
 * A little extension for runnable interface, so it was possible to
interrupt smth
 */
public interface MyRunnable extends Runnable {

    void interrupt();
}
```

ResourceLock.java

```
package ru.etu.graphview.drawing.multithreading;

import java.util.List;
import java.util.concurrent.atomic.AtomicInteger;

/**
 * Some resources for threads
 */
public class ResourceLock {
    /**
     * Determines whose turn now to execute
     */
    public volatile int flag = 0;

    /**
     * Current task number
     */
    public volatile AtomicInteger currentTask = new AtomicInteger(0);
```

```
    /**
     * List of task to run
     */
    public final List<MyRunnable> tasks;

    /**
     * Controls if threads show be working
     */
    public volatile boolean working = true;

    /**
     * Determines wait time BEFORE executing task
     */
    public final int waitTime;

    public ResourceLock(List<MyRunnable> tasks, int waitTime) {
        this.tasks = tasks;
        this.waitTime = waitTime;
    }
}
```

ThreadA.java

```
package ru.etu.graphview.drawing.multithreading;

/**
 * First of two threads which implement step by step executing with
waiting between.
 */
public class ThreadA extends Thread {
    final ResourceLock lock;

    public ThreadA(ResourceLock lock) {
        this.lock = lock;
    }

    @Override
    public void run() {
        try {
            synchronized (lock) {
                while (lock.working) {
                    // Wait for out turn
                    while (lock.flag != 0) {
                        lock.wait();
                    }

                    // Wait some time
                    Thread.sleep(lock.waitTime);
                    if (!lock.working) { // If was ordered to stop while
waiting
```

```
                    break;
                }

                // Run Task
                lock.tasks.get(lock.currentTask.get()).run();

                // Determine to get new task and say next thread to
work or stop
                if (lock.currentTask.get() + 1 < lock.tasks.size()) {
                    lock.currentTask.incrementAndGet();
                    lock.flag = 1;
                    lock.notifyAll();
                } else {
                    lock.currentTask.incrementAndGet();
                    lock.flag = 1;
                    lock.working = false;
                    lock.notifyAll();
                }
            }
        }
    } catch (Exception ignored) {
        if (lock.currentTask.get() < lock.tasks.size())
            lock.tasks.get(lock.currentTask.get()).interrupt();
    }
  }
}
```

ThreadB.java

```
package ru.etu.graphview.drawing.multithreading;

/**
 * Second thread which implements step by step executing with waiting
between.
 */
public class ThreadB extends Thread {
    final ResourceLock lock;

    public ThreadB(ResourceLock lock) {
        this.lock = lock;
    }

    @Override
    public void run() {
        try {
            synchronized (lock) {
                while (lock.working) {
                    // Wait for out turn
                    while (lock.flag != 1) {
                        lock.wait();
                    }
```

```
                    // Wait some time
                    Thread.sleep(lock.waitTime);
                    if (!lock.working) { // If was ordered to stop while
waiting
                        break;
                    }

                    // Run Task
                    lock.tasks.get(lock.currentTask.get()).run();

                    // Determine to get new task and say next thread to
work or stop
                    if (lock.currentTask.get() + 1 < lock.tasks.size()) {
                        lock.currentTask.incrementAndGet();
                        lock.flag = 0;
                        lock.notifyAll();
                    } else {
                        lock.currentTask.incrementAndGet();
                        lock.flag = 0;
                        lock.working = false;
                        lock.notifyAll();
                    }
                }
            }
        } catch (Exception ignored) {
            if (lock.currentTask.get() < lock.tasks.size())
                lock.tasks.get(lock.currentTask.get()).interrupt();
        }
    }
}
```

DijkstraStepEngine.java

```
package ru.etu.controllers;

import javafx.application.Platform;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.ReadOnlyDoubleWrapper;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.geometry.Bounds;
import javafx.scene.Cursor;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.control.Button;
import javafx.scene.control.Menu;
```

```
import javafx.scene.control.MenuBar;
import javafx.scene.control.MenuItem;
import javafx.scene.layout.AnchorPane;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.Pane;
import javafx.stage.FileChooser;
import javafx.stage.Stage;
import ru.etu.graph.DirectedGraphList;
import ru.etu.graph.Graph;
import ru.etu.graph.GraphList;
import ru.etu.graphview.drawing.DijkstraStepEngine;
import ru.etu.graphview.GraphPane;
import ru.etu.graphview.GraphViewProperties;
import ru.etu.graphview.drawing.GraphEditor;
import ru.etu.graphview.drawing.ViewMode;
import ru.etu.io.IOGraph;
import ru.etu.logger.Logger;

import java.awt.*;
import java.io.File;
import java.io.IOException;
import java.net.URI;
import java.net.URL;
import java.util.ResourceBundle;

public class App implements Initializable {
    /*
    INTERFACE OBJECTS
     */

    // Buttons
    public Button prevBtn;
    public Button nextBtn;
    public ToggleButton pauseBtn;
    public ToggleButton playBtn;
    public Button stopBtn;
    public ToggleButton moveBtn;
    public ToggleButton vertexBtn;
    public ToggleButton edgeBtn;
    public ToggleButton chooseBtn;
    public Button clearBtn;

    // Menus and menu items
    public MenuItem saveMeMenuItem;
    public MenuItem saveAsMenuItem;
    public MenuItem changeMenuItem;
    public Menu playMenu;
    public Menu settingsMenu;
    public MenuItem exitMenuItem;
    public MenuItem prevMenuItem;
```

```
    public MenuItem nextMenuItem;
    public MenuItem pauseMenuItem;
    public MenuItem playMenuItem;
    public MenuItem stopMenuItem;
    public MenuItem moveMenuItem;
    public MenuItem vertexMenuItem;
    public MenuItem edgeMenuItem;
    public MenuItem chooseMenuItem;
    public MenuItem clearMenuItem;

    // Other objects
    public BorderPane contentBorderPane;
    public AnchorPane mainPane;
    public MenuBar menuBar;
    public Pane graphPanePane;

    /*
    SCALING VARIABLES
     */
    /**
     * Minimum scale factor. Must be a multiple of scaleStep
     */
    private double minScale = 0.25;

    /**
     * Maximum scale factor. Must be a multiple of scaleStep
     */
    private double maxScale = 3;

    /**
     * Step of scaling
     */
    private double scaleStep = 0.25;

    private int standardWidth = 1000;
    private int standardHeight = 700;
    private int paneSizeFactor = 3;
    private boolean isJunk = true;

    private final DoubleProperty scaleFactor = new
ReadOnlyDoubleWrapper(1);

    /*
    SAVE/LOAD VARIABLES
     */
    //TODO: change type to .graph
    private final String fileType = ".json";
    //private String filePath = "default"+fileType;
    private String filePath = null;
```

```
/*
STATE VARIABLES AND CONNECTED OBJECT TO THIS STATES
 */
private boolean isPlaying = false;
private DijkstraStepEngine stepEngine;
private int playSpeed = 3000;

private boolean isGraphCreated = false;
public GraphPane graphPane;
private GraphEditor graphEditor;

private boolean isLeft = true;
public Node top;
public Node left;
private ViewMode lastInstrumentType;

private boolean isLoggerOpen = false;
private Stage loggerStage;
private Logger loggerInstance;

private boolean isSettingsOpen = false;
private Stage settingsStage;

private boolean isAboutOpen = false;
private Stage aboutStage;

Settings settingsController = null;

@Override
public void initialize(URL url, ResourceBundle resources) {
    /*
    GRAPH PANE INITIALISATION
     */

    graphPane = new GraphPane();
    graphPane.setPrefHeight(standardHeight * paneSizeFactor);
    graphPane.setPrefWidth(standardWidth * paneSizeFactor);

    graphPanePane.getChildren().add(graphPane);
    graphPane.toFront();

    menuBar.setViewOrder(-500);
    graphPanePane.setViewOrder(500);

    // Graph editor init
    graphEditor = new GraphEditor(graphPane);

    // Scaling init
    enablePanAndZoom();
```

```
        /*
        PANELS INITIALIZATION
         */

        top = contentBorderPane.getTop();
        left = contentBorderPane.getLeft();

        contentBorderPane.setTop(null);

        /*
        SETTINGS INITIALIZATION
         */
        try {
            FXMLLoader settingsFxmlLoader = new
FXMLLoader(getClass().getResource("/ru/etu/studypract/settings.fxml"));

            Scene appScene = new Scene(settingsFxmlLoader.load());
            settingsStage = new Stage();
            settingsStage.setTitle("Logger");
            settingsStage.setScene(appScene);

            settingsStage.setOnCloseRequest((event) -> {
                settingsStage.hide();
                isSettingsOpen = false;
            });

            settingsController = settingsFxmlLoader.getController();
            settingsController.setApp(this);
            settingsController.setData(playSpeed, maxScale, minScale,
scaleStep, paneSizeFactor);
        } catch (IOException ioException) {
            System.err.println("Failed to load settings fxml!");
        } catch (Exception exception) {
            exception.printStackTrace();
        }


        /*
        LOGGER INITIALIZATION
         */

        LoggerView loggerViewController = null;
        try {
            FXMLLoader loggerFxmlLoader = new
FXMLLoader(getClass().getResource("/ru/etu/studypract/loggerView.fxml"));

            Scene appScene = new Scene(loggerFxmlLoader.load());
            loggerStage = new Stage();
            loggerStage.setTitle("Logger");
            loggerStage.setScene(appScene);
```

```
            loggerStage.setMinWidth(600);
            loggerStage.setMinHeight(400);

            loggerStage.setOnCloseRequest((event) -> {
                loggerStage.hide();
                isLoggerOpen = false;
            });

            loggerViewController = loggerFxmlLoader.getController();
        } catch (IOException ioException) {
            System.err.println("Failed to load logger fxml!");
        } catch (Exception exception) {
            exception.printStackTrace();
        }

        Logger.initialiseInstance(loggerViewController);
        loggerInstance = Logger.getInstance();

        /*
        BUTTONS AND MENU INITIALIZATION
         */

        setTopButtonsDisable(true);

        settingsMenu.setDisable(true);
        saveMeMenuItem.setDisable(true);
        saveAsMenuItem.setDisable(true);

        setLeftButtonsDisable(true);

        playMenu.setDisable(true);

        setUpBtnsActions();
    }

    private void setUpBtnsActions() {
        //buttons/menu items
        exitMenuItem.setOnAction(event -> eventCloseWindow());

        changeMenuItem.setOnAction(event -> {
            changeMode();
        });

        EventHandler<ActionEvent> vertexBtnSelectHandler = event -> {
            setVertexCreationMode();
            if (!vertexBtn.isSelected()) {
                vertexBtn.setSelected(true);
            }
        };
        EventHandler<ActionEvent> edgeBtnSelectHandler = event -> {
```

```
        setEdgeCreationMode();
        if (!edgeBtn.isSelected()) {
            edgeBtn.setSelected(true);
        }
    };
    EventHandler<ActionEvent> moveBtnSelectHandler = event -> {
        setMoveMode();
        if (!moveBtn.isSelected()) {
            moveBtn.setSelected(true);
        }
    };
    EventHandler<ActionEvent> chooseBtnSelectHandler = event -> {
        setChooseMode();
        if (!chooseBtn.isSelected()) {
            chooseBtn.setSelected(true);
        }
    };
    EventHandler<ActionEvent> clearBtnSelectHandler = event -> {
        clearGraph();
    };
    EventHandler<ActionEvent> playBtnSelectHandler = event -> {
        play();
        if (!playBtn.isSelected()) {
            playBtn.setSelected(true);
        }
    };
    EventHandler<ActionEvent> stopBtnSelectHandler = event -> {
        stop();
        if (playBtn.isSelected()) {
            playBtn.setSelected(false);
        }
        if (pauseBtn.isSelected()) {
            pauseBtn.setSelected(false);
        }
    };
    EventHandler<ActionEvent> nextBtnSelectHandler = event -> {
        stepForward();
        pauseBtn.setSelected(true);
    };
    EventHandler<ActionEvent> prevBtnSelectHandler = event -> {
        stepBack();
        pauseBtn.setSelected(true);
    };
    EventHandler<ActionEvent> pauseBtnSelectHandler = event -> {
        pause();
        if (!pauseBtn.isSelected()) {
            pauseBtn.setSelected(true);
        }
    };
```

```
        vertexBtn.setOnAction(vertexBtnSelectHandler);
        edgeBtn.setOnAction(edgeBtnSelectHandler);
        moveBtn.setOnAction(moveBtnSelectHandler);
        chooseBtn.setOnAction(chooseBtnSelectHandler);
        clearBtn.setOnAction(clearBtnSelectHandler);
        playBtn.setOnAction(playBtnSelectHandler);
        stopBtn.setOnAction(stopBtnSelectHandler);
        nextBtn.setOnAction(nextBtnSelectHandler);
        prevBtn.setOnAction(prevBtnSelectHandler);
        pauseBtn.setOnAction(pauseBtnSelectHandler);

        vertexMenuItem.setOnAction(vertexBtnSelectHandler);
        edgeMenuItem.setOnAction(edgeBtnSelectHandler);
        moveMenuItem.setOnAction(moveBtnSelectHandler);
        chooseMenuItem.setOnAction(chooseBtnSelectHandler);
        clearMenuItem.setOnAction(clearBtnSelectHandler);
        playMenuItem.setOnAction(playBtnSelectHandler);
        stopMenuItem.setOnAction(stopBtnSelectHandler);
        nextMenuItem.setOnAction(nextBtnSelectHandler);
        prevMenuItem.setOnAction(prevBtnSelectHandler);
        pauseMenuItem.setOnAction(pauseBtnSelectHandler);
    }

    private void enablePanAndZoom() {

        graphPanePane.setOnScroll(event -> {

            String OS = System.getProperty("os.name",
"generic").toLowerCase();
            //System.out.println(OS);
            if(!OS.contains("win")){
                if (isJunk) {
                    isJunk = false;
                    return;
                } else {
                    isJunk = true;
                }
            }

            var direction = event.getDeltaY() >= 0 ? 1 : -1;

            var curScale = scaleFactor.getValue();
            var computedScale = curScale + direction * scaleStep;

            if (Double.compare(computedScale, minScale) < 0) {
                computedScale = minScale;
            } else if (Double.compare(computedScale, maxScale) > 0) {
                computedScale = maxScale;
            }
```

```
            if (curScale != computedScale) {

                graphPane.setScaleX(computedScale);
                graphPane.setScaleY(computedScale);

                if (computedScale == minScale) {
                    if (isLeft) {
                        graphPane.setTranslateX(-
graphPanePane.getTranslateX() - 43 - standardWidth / 2f *
(Math.max((paneSizeFactor - 1), 0)));
                        graphPane.setTranslateY(-
graphPanePane.getTranslateY() - standardHeight / 2f *
(Math.max((paneSizeFactor - 1), 0)));
                    } else {
                        graphPane.setTranslateX(-
graphPanePane.getTranslateX() - standardWidth / 2f *
(Math.max((paneSizeFactor - 1), 0)));
                        graphPane.setTranslateY(-
graphPanePane.getTranslateY() - 43 - standardHeight / 2f *
(Math.max((paneSizeFactor - 1), 0)));
                    }
                } else {
                    scaleFactor.setValue(computedScale);

                    // Положение в сцене
                    Bounds bounds =
graphPane.localToScene(graphPane.getBoundsInLocal());
                    double f = (computedScale / curScale) - 1;
                    double dx = (event.getX() - (bounds.getWidth() / 2 +
bounds.getMinX()));
                    double dy = (event.getY() - (bounds.getHeight() / 2 +
bounds.getMinY()));

                    graphPane.setTranslateX(graphPane.getTranslateX() - f
* dx);
                    graphPane.setTranslateY(graphPane.getTranslateY() - f
* dy);
                }
            }

            event.consume();

        });

        final DragData dragData = new DragData();

        graphPanePane.setOnMousePressed(event -> {
            if (event.isSecondaryButtonDown()) {
                dragData.cursorType =
graphPanePane.getScene().getCursor();
```

```
                graphPanePane.getScene().setCursor(Cursor.MOVE);

                dragData.mouseAnchorX = event.getX();
                dragData.mouseAnchorY = event.getY();

                dragData.translateAnchorX = graphPane.getTranslateX();
                dragData.translateAnchorY = graphPane.getTranslateY();
            }
        });

        graphPanePane.setOnMouseReleased(event -> {
            graphPanePane.getScene().setCursor(dragData.cursorType);
        });

        graphPanePane.setOnMouseDragged(event -> {
            if (event.isSecondaryButtonDown()) {
                graphPane.setTranslateX(dragData.translateAnchorX +
event.getX() - dragData.mouseAnchorX);
                graphPane.setTranslateY(dragData.translateAnchorY +
event.getY() - dragData.mouseAnchorY);
            }
        });
    }

    public void setTopButtonsDisable(boolean a) {
        prevBtn.setDisable(a);
        nextBtn.setDisable(a);
        pauseBtn.setDisable(a);
        playBtn.setDisable(a);
        stopBtn.setDisable(a);
    }

    public void setTopMenuItemsDisable(boolean a) {
        prevMenuItem.setDisable(a);
        nextMenuItem.setDisable(a);
        pauseMenuItem.setDisable(a);
        playMenuItem.setDisable(a);
        stopMenuItem.setDisable(a);
    }

    public void setLeftButtonsDisable(boolean a) {
        moveBtn.setDisable(a);
        vertexBtn.setDisable(a);
        edgeBtn.setDisable(a);
        chooseBtn.setDisable(a);
        clearBtn.setDisable(a);
    }

    public void setLeftMenuItemsDisable(boolean a) {
        moveMenuItem.setDisable(a);
```

```
    vertexMenuItem.setDisable(a);
    edgeMenuItem.setDisable(a);
    chooseMenuItem.setDisable(a);
    clearMenuItem.setDisable(a);
}

@FXML
private void changeMode() {
    if (!isLeft) {
        // Setting mode

        if (lastInstrumentType != null) {
            switch (lastInstrumentType) {
                case NORMAL -> {
                    setMoveMode();
                }
                case VERTEX_PLACEMENT -> {
                    setVertexCreationMode();
                }
                case VERTEX_CHOOSE -> {
                    setChooseMode();
                }
                case EDGE_PLACEMENT -> {
                    setEdgeCreationMode();
                }
            }
        }

        // Stopping threads in stepEngine if necessary
        if (isPlaying) {
            stop();
        }

        if (graphEditor.getNodesSelected1() == 2) {
            graphEditor.removeAlgLabels();
        }

        // Setting buttons accessibility
        playMenu.setDisable(true);

        if (isGraphCreated) {
            setLeftButtonsDisable(false);
            setLeftMenuItemsDisable(false);
            settingsMenu.setDisable(false);
        } else {
            setLeftButtonsDisable(true);
            settingsMenu.setDisable(true);
        }

        // Translating coordinate if scale is right
```

```
            if (scaleFactor.get() == minScale + scaleStep) {
                graphPane.setTranslateX(-graphPanePane.getTranslateX() -
43 - standardWidth / 2f * (Math.max((paneSizeFactor - 1), 0)));
                graphPane.setTranslateY(-graphPanePane.getTranslateY() -
standardHeight / 2f * (Math.max((paneSizeFactor - 1), 0)));
            }

            // Push log message
            loggerInstance.printMessage(getClass().getName(), "Switching
to Setting mode.");

            // Switching panels
            contentBorderPane.setTop(null);
            contentBorderPane.setLeft(left);
            isLeft = true;

        } else {
            // Play mode

            // Setting buttons accessibility
            settingsMenu.setDisable(true);

            if (isGraphCreated) {
                setMoveModeWithoutSaving();
            }

            if (graphPane.isGraphSet() && graphEditor.getNodesSelected1()
== 2) {
                // Initialising stepEngine and preparing vertices for
work

graphEditor.addAlgLabels(graphEditor.getFirstNodeSelect().getVertex());
                stepEngine = new DijkstraStepEngine(graphEditor,
playSpeed);

                playBtn.setDisable(false);

                playMenu.setDisable(false);
                setTopMenuItemsDisable(true);
                playMenuItem.setDisable(false);
            } else {
                setTopButtonsDisable(true);
                setTopMenuItemsDisable(true);
            }

            // Translating coordinate if scale is right
            if (scaleFactor.get() == minScale + scaleStep) {
                graphPane.setTranslateX(-graphPanePane.getTranslateX() -
standardWidth / 2f * (Math.max((paneSizeFactor - 1), 0)));
```

```
                graphPane.setTranslateY(-graphPanePane.getTranslateY() -
43 - standardHeight / 2f * (Math.max((paneSizeFactor - 1), 0)));
            }

            // Push log message
            loggerInstance.printMessage(getClass().getName(), "Switching
to play mode.");

            // Switching panels
            contentBorderPane.setTop(top);
            contentBorderPane.setLeft(null);
            isLeft = false;
        }
    }

    public void eventCloseWindow() {
        if (isGraphCreated && isPlaying) {
            stepEngine.stop();
        }

        Platform.exit();
    }

    public void closeWindow(Stage stage) {
        stage.setOnCloseRequest(event -> eventCloseWindow());
    }

    /*
    MENU BAR'S METHODS
     */
    @FXML
    private void saveGraph() {
        if(filePath==null){
            saveGraphAs();
        } else {
            try {
                IOGraph saver = new IOGraph();
                if (isGraphCreated) {
                    saver.saveGraph(filePath, graphPane.getGraph());
                } else {
                    loggerInstance.printMessage(getClass().toString(),
"Cannot save uncreated graph!", true);
                }
            } catch (IOException ex) {
                loggerInstance.printMessage(getClass().toString(),
"Cannot save graph to "+filePath, true);
            }
        }
    }
```

```
    @FXML
    private void saveGraphAs() {
        FileChooser fileChooser = new FileChooser();
        FileChooser.ExtensionFilter extensionFilter = new
FileChooser.ExtensionFilter("Graph files (*"+fileType+")", "*"+fileType);
        fileChooser.getExtensionFilters().add(extensionFilter);
        Stage stage = new Stage();
        stage.setTitle("Save graph");
        File path = fileChooser.showSaveDialog(stage);
        if (path != null) {
            try {
                IOGraph saver = new IOGraph();
                if (isGraphCreated) {
                    saver.saveGraph(fixFileName(path),
graphPane.getGraph());
                } else {
                    loggerInstance.printMessage(getClass().toString(),
"Cannot save uncreated graph!", true);
                }
                filePath = fixFileName(path);
            } catch (IOException ex) {
                loggerInstance.printMessage(getClass().toString(),
"Cannot save graph to "+fixFileName(path), true);
            }
        }
    }

    @FXML
    private void loadGraph() {
        FileChooser fileChooser = new FileChooser();
        FileChooser.ExtensionFilter extensionFilter = new
FileChooser.ExtensionFilter("Graph files (*"+fileType+")", "*"+fileType);
        fileChooser.getExtensionFilters().add(extensionFilter);
        Stage stage = new Stage();
        stage.setTitle("Load graph");
        File path = fileChooser.showOpenDialog(stage);
        if (path != null) {
            if (isPlaying) {
                changeMode();
            }
            try {
                IOGraph saver = new IOGraph();
                Graph newGraph = saver.loadGraph(path.toString());
                var properties = new GraphViewProperties();
                properties.setNeedArrows(newGraph instanceof
DirectedGraphList);
                properties.setNeedDoubleEdges(newGraph instanceof
DirectedGraphList);
                graphEditor.eraseGraph();
                graphPane.loadGraph(newGraph, properties);
```

```
                setLeftButtonsDisable(false);
                settingsMenu.setDisable(false);
                isGraphCreated = true;
                filePath = path.toString();
            } catch (IOException ex) {
                loggerInstance.printMessage(getClass().toString(),
"Cannot open graph "+path.toString(), true);
            }
            if (isGraphCreated) {
                saveMeMenuItem.setDisable(false);
                saveAsMenuItem.setDisable(false);
                setMoveMode();
                moveBtn.setSelected(true);
            }
        }
    }

    public void setAboutOpen(boolean aboutOpen) {
        isAboutOpen = aboutOpen;
    }

    @FXML
    private void openRepo() {
        try {
            Desktop.getDesktop().browse(new
URI("https://github.com/ilya201232/StudyPract"));
        } catch (Exception ignored) {
        }

    }

    @FXML
    private void openLogger() {
        if (!isLoggerOpen) {
            isLoggerOpen = true;
            loggerStage.show();
        } else {
            loggerStage.requestFocus();
        }
    }

    @FXML
    private void openSettings() throws IOException {
        if (!isSettingsOpen || settingsController.getClosed()) {
            isSettingsOpen = true;
            settingsController.setClosed(false);
            settingsStage.show();
        } else {
            settingsStage.requestFocus();
```

```
        }
    }

    public void setSettings(int playSpeed, double zoomStep, double
zoomMax, double zoomMin, int typeOfSize) {

        this.playSpeed = playSpeed;
        scaleStep = zoomStep;
        maxScale = zoomMax;
        minScale = zoomMin;
        paneSizeFactor = typeOfSize;

        if (!isLeft) {
            changeMode();
        }

        graphPane.setPrefHeight(standardHeight * paneSizeFactor);
        graphPane.setPrefWidth(standardWidth * paneSizeFactor);

        scaleFactor.setValue(minScale);

        graphPane.setScaleX(minScale);
        graphPane.setScaleY(minScale);

        graphPane.setTranslateX(-graphPanePane.getTranslateX() - 43 -
standardWidth / 2f * (Math.max((paneSizeFactor - 1), 0)));
        graphPane.setTranslateY(-graphPanePane.getTranslateY() -
standardHeight / 2f * (Math.max((paneSizeFactor - 1), 0)));


        enablePanAndZoom();

        // TODO:: реализовать тут изменение параметров (вызвать
соответствующие методы для изменения)
    }

    @FXML
    private void openAbout() throws IOException {

        if (!isAboutOpen) {
            isAboutOpen = true;

            FXMLLoader aboutFxmlLoader = new
FXMLLoader(getClass().getResource("/ru/etu/studypract/about.fxml"));

            Scene appScene = new Scene(aboutFxmlLoader.load());

            About aboutController = aboutFxmlLoader.getController();

            Stage stage = new Stage();
```

```
            stage.setTitle("About Dijkstra Visualiser");
            stage.setScene(appScene);
            stage.show();

            aboutController.setAppController(this);

            stage.setResizable(false);

            aboutStage = stage;

            stage.setOnCloseRequest((event) -> {
                isAboutOpen = false;
            });
        } else {
            aboutStage.requestFocus();
        }


    }

    // Graph creation
    @FXML
    private void createNewGraph() {
        Graph graph = new GraphList();
        var properties = new GraphViewProperties();
        properties.setNeedArrows(false);

        graphEditor.eraseGraph();
        graphPane.loadGraph(graph, properties);

        // Unblock left side tools
        setLeftButtonsDisable(false);
        settingsMenu.setDisable(false);

        // Unblock save buttons
        saveMeMenuItem.setDisable(false);
        saveAsMenuItem.setDisable(false);

        // Set move tool as active
        setMoveMode();
        moveBtn.setSelected(true);

        isGraphCreated = true;
        filePath=null;
    }

    @FXML
    private void createNewDirectedGraph() {
        Graph graph = new DirectedGraphList();
        var properties = new GraphViewProperties();
```

```
        properties.setNeedArrows(true);

        graphEditor.eraseGraph();
        graphPane.loadGraph(graph, properties);

        // Unblock left side tools
        setLeftButtonsDisable(false);
        settingsMenu.setDisable(false);

        // Unblock save buttons
        saveMeMenuItem.setDisable(false);
        saveAsMenuItem.setDisable(false);

        // Set move tool as active
        setMoveMode();
        moveBtn.setSelected(true);

        isGraphCreated = true;
        filePath=null;
    }

    /*
    GRAPH EDITING TOOLS' METHODS
     */

    private void setMoveModeWithoutSaving() {
        graphEditor.setViewMode(ViewMode.NORMAL);
        loggerInstance.printMessage(getClass().getName(), "Move mode
activated. You can now move any vertex you want.");
    }

    @FXML
    private void setMoveMode() {
        lastInstrumentType = ViewMode.NORMAL;
        graphEditor.setViewMode(ViewMode.NORMAL);
        loggerInstance.printMessage(getClass().getName(), "Move mode
activated. You can now move any vertex you want.");
    }

    @FXML
    private void setVertexCreationMode() {
        lastInstrumentType = ViewMode.VERTEX_PLACEMENT;
        graphEditor.setViewMode(ViewMode.VERTEX_PLACEMENT);
        loggerInstance.printMessage(getClass().getName(), "Vertex
creation mode activated. You can now place new vertices on the screen
(Moving vertices is allowed). Just click on an empty space of the
pane.");
    }

    @FXML
```

```
    private void setEdgeCreationMode() {
        lastInstrumentType = ViewMode.EDGE_PLACEMENT;
        graphEditor.setViewMode(ViewMode.EDGE_PLACEMENT);
        loggerInstance.printMessage(getClass().getName(), "Edge creation
mode activated. You can now place new edges on the screen (Moving
vertices is forbidden). Choose two vertices to create edge between
them.");
    }

    @FXML
    private void setChooseMode() {
        lastInstrumentType = ViewMode.VERTEX_CHOOSE;
        graphEditor.setViewMode(ViewMode.VERTEX_CHOOSE);
        loggerInstance.printMessage(getClass().getName(), "Vertex choose
mode activated. You can now choose two vertices to make Dijkstra
algorithm find the shortest bath between them.");
    }

    @FXML
    private void clearGraph() {
        graphEditor.eraseGraph();
        loggerInstance.printMessage(getClass().getName(), "You just
cleared the graph. Now you can place new elements.");
    }

    /*
    ALGORITHM PLAY TOOLS' METHOD
     */
    @FXML
    private void stepBack() {
        stepEngine.makeStepBackwards();
    }

    @FXML
    private void stepForward() {
        stepEngine.makeStepForward();
    }

    @FXML
    private void play() {
        setMoveMode();
        if (!stepEngine.isInitialised()) {
            stepEngine.applyDijkstra();
            if (stepEngine.isPathExists()) {
                stepEngine.startAutoPlay();
                isPlaying = true;
            }

        } else if (stepEngine.isPaused()) {
            stepEngine.resumeAutoPlay();
```

```
            isPlaying = true;
        } else if (!isPlaying) {
            stepEngine.startAutoPlay();
            isPlaying = true;
        }

        if (stepEngine.isPathExists()) {
            setTopButtonsDisable(false);
            setTopMenuItemsDisable(false);
            playMenu.setDisable(false);
        }
    }

    @FXML
    private void pause() {
        stepEngine.pauseAutoPlay();
    }

    @FXML
    private void stop() {
        stepEngine.stop();

        setTopButtonsDisable(true);
        playBtn.setDisable(false);

        setTopMenuItemsDisable(true);
        playMenuItem.setDisable(false);

        pauseBtn.setSelected(false);
        playBtn.setSelected(false);

        isPlaying = false;
    }

    private String fixFileName(File path){
        String pathStr = path.toString();
        String OS = System.getProperty("os.name",
"generic").toLowerCase();
        //System.out.println(OS);
        // TODO протестировать на debian
        if(!pathStr.endsWith(fileType)&&!OS.contains("win")){
            //loggerInstance.printMessage(getClass().getName(), "Saved to
directory "+pathStr+fileType);
            return pathStr+fileType;
        }
        //loggerInstance.printMessage(getClass().getName(), "Saved to
directory "+pathStr);
        return pathStr;
    }
}
```

```
class DragData {

    Cursor cursorType;

    double mouseAnchorX;
    double mouseAnchorY;

    double translateAnchorX;
    double translateAnchorY;
}
```

DrawingType.java

```
package ru.etu.graphview.drawing;

public enum DrawingType {
    PATH_STROKE,
    CHECK_STROKE,
    UPDATE_FILL
}
```

GraphEditor.java

```
package ru.etu.graphview.drawing;

import javafx.geometry.Pos;
import javafx.scene.Cursor;
import javafx.scene.control.TextField;
import javafx.scene.input.KeyCode;
import ru.etu.controllers.LoggerView;
import ru.etu.graph.DirectedGraphList;
import ru.etu.graph.Edge;
import ru.etu.graph.InvalidVertexException;
import ru.etu.graph.Vertex;
import ru.etu.graphview.GraphPane;
import ru.etu.graphview.base.*;
import ru.etu.logger.Logger;

import java.util.ArrayList;
import java.util.List;

/**
 * This method manage any changes in graph from GUI
 */
public class GraphEditor {

    Logger loggerInstance;

    private GraphPane graphPane; // To have access to pane and graph
itself
```

```
    private ViewMode viewMode; // To manage current tool

    // For Vertex creation
    private boolean isCreatingVertex = false;
    private FXVertexNode newVertexNode;

    // For Edge creation
    private int currentNode = 0;
    private FXEdge newEdgeNode;
    private boolean movedOppositeEdge = false;
    private Edge connectedEdge;
    private String connectedEdgeNodeWeight;

    // For Edge creation
    private FXVertexNode firstNode;
    private FXVertexNode secondNode;
    private int nodesSelected = 0;

    // For Vertex selection
    private FXVertexNode firstNodeSelect;
    private FXVertexNode secondNodeSelect;
    private int nodesSelected1 = 0;

    private final List<Label> algLabels; // Stores algorithm labels for
easy access

    public GraphEditor(GraphPane graphPane) {
        this(graphPane, ViewMode.NORMAL);
    }

    public GraphEditor(GraphPane graphPane, ViewMode viewMode) {
        this.graphPane = graphPane;
        this.viewMode = viewMode;

        algLabels = new ArrayList<>();
        loggerInstance = Logger.getInstance();
    }

    /**
     * Deletes everything from graph (and its view) and resets all
instrument data
     */
    public void eraseGraph() {
        clearEditorData();
        graphPane.clearPane();
    }

    /**
```

```
     * Resets all instrument data, considering that later graph will be
reset as well
     */
    private void clearEditorData() {
        if (currentNode == 2) {
            graphPane.getChildren().remove(newEdgeNode.getLabelField());
            graphPane.getChildren().remove(newEdgeNode.getArrow());
            graphPane.getChildren().remove(newEdgeNode);

            if (graphPane.getGraph().getClass() ==
DirectedGraphList.class && movedOppositeEdge) {

                var fxEdgeCurve =
graphPane.getEdgeMap().get(connectedEdge);


graphPane.getChildren().remove(fxEdgeCurve.getLabelField());
                graphPane.getChildren().remove(fxEdgeCurve.getArrow());
                graphPane.getChildren().remove(fxEdgeCurve);
            }
        }

        if (isCreatingVertex) {

graphPane.getChildren().remove(newVertexNode.getLabelField());
            graphPane.getChildren().remove(newVertexNode);
        }


        isCreatingVertex = false;
        newVertexNode = null;
        newEdgeNode = null;
        firstNode = null;
        secondNode = null;
        currentNode = 0;
        movedOppositeEdge = false;
        nodesSelected = 0;
        nodesSelected1 = 0;
    }

    /**
     * Resets all instrument data
     */
    private void resetModes() {
        // Vertex creation mode
        if (isCreatingVertex) {

graphPane.getChildren().remove(newVertexNode.getLabelField());
            graphPane.getChildren().remove(newVertexNode);
            isCreatingVertex = false;
```

```
        }

        // Edge creation mode
        switch (currentNode) {
            case 1 -> firstNode.removeStyleClass("vertex-selected");
            case 2 -> {
                firstNode.removeStyleClass("vertex-selected");
                secondNode.removeStyleClass("vertex-selected");

graphPane.getChildren().remove(newEdgeNode.getLabelField());
                graphPane.getChildren().remove(newEdgeNode.getArrow());
                graphPane.getChildren().remove(newEdgeNode);

                if (graphPane.getGraph().getClass() ==
DirectedGraphList.class && movedOppositeEdge) {

                    var fxEdgeCurve =
graphPane.getEdgeMap().get(connectedEdge);


graphPane.getChildren().remove(fxEdgeCurve.getLabelField());

graphPane.getChildren().remove(fxEdgeCurve.getArrow());

graphPane.getChildren().remove(fxEdgeCurve.getLabel());
                    graphPane.getChildren().remove(fxEdgeCurve);

                    graphPane.getEdgeMap().remove(connectedEdge);

                    var fxEdgeLine = new FXEdgeLine(connectedEdge,
graphPane.getVertexMap().get(connectedEdge.getVertexOutbound()),
graphPane.getVertexMap().get(connectedEdge.getVertexInbound()));

                    var label = new Label(connectedEdgeNodeWeight);
                    fxEdgeLine.setLabel(label);
                    graphPane.getChildren().add(label);

                    if
(graphPane.getGraphViewProperties().isNeedArrows()) {
                        var arrow = new
Arrow(graphPane.getGraphViewProperties().getArrowSize());
                        fxEdgeLine.setArrow(arrow);
                        graphPane.getChildren().add(arrow);
                    }

                    graphPane.getChildren().add(fxEdgeLine);
                    graphPane.getEdgeMap().put(connectedEdge,
fxEdgeLine);

                    graphPane.takeVerticesOnFront();
```

```
                }
            }
        }

        nodesSelected = 0;

        movedOppositeEdge = false;

        currentNode = 0;

        graphPane.enableVertexDrag();
    }

    /**
     * Creates TextField
     *
     * @return created TextField
     */
    private TextField createTextField() {
        TextField dataInput = new TextField();
        dataInput.setAlignment(Pos.CENTER);
        dataInput.setPrefHeight(10);
        dataInput.setPrefWidth(70);

        return dataInput;
    }

    /**
     * Creates Label view with passed data and removes TextField.
     *
     * @param labelText text to put in label
     * @param node      view object to which to set created label
     * @param textField textField to be deleted
     */
    private void CreateLabelFromTextField(String labelText, LabelledNode
node, TextField textField) {
        Label label = new Label(labelText);

        node.setLabel(label);
        graphPane.getChildren().add(label);
        graphPane.getChildren().remove(textField);
    }

    /**
     * Sets instrument by viewMode
     *
     * @param viewMode instrument type to set up
     */
    public void setViewMode(ViewMode viewMode) {
        this.viewMode = viewMode;
```

```
        switch (viewMode) {
            case NORMAL -> {

                resetModes();

                graphPane.setOnMousePressed((e) -> {
                });
                graphPane.setOnMouseEntered(event -> {
                });
                graphPane.setOnMouseExited(event -> {
                });
                graphPane.setOnKeyPressed((e) -> {
                });

            }
            case VERTEX_PLACEMENT -> {

                resetModes();

                graphPane.setOnMousePressed((e) -> {
                    if (e.isPrimaryButtonDown() && !isCreatingVertex) {
                        isCreatingVertex = true;

                        newVertexNode = new FXVertexNode(
                                e.getX(),
                                e.getY(),

graphPane.getGraphViewProperties().getVerticesRadius());

                        TextField vertexDataInput = createTextField();

                        newVertexNode.setLabelField(vertexDataInput);

                        graphPane.getChildren().add(vertexDataInput);
                        graphPane.getChildren().add(newVertexNode);
                    }
                });

                graphPane.setOnKeyPressed((e) -> {
                    if (e.getCode() == KeyCode.ENTER && isCreatingVertex)
{
                        String data =
newVertexNode.getLabelField().getText();

                        if (data.length() == 0) {

loggerInstance.printMessage(getClass().getName(), "Vertex name can't be
empty.", true);
                        } else if (data.contains(" ")) {
```

```
loggerInstance.printMessage(getClass().getName(), "Vertex name can't have
spaces in its name.", true);
                        } else {
                            Vertex newVertex;
                            try {
                                newVertex =
graphPane.getGraph().insertVertex(data);

                                newVertexNode.setVertex(newVertex);
                                graphPane.getVertexMap().put(newVertex,
newVertexNode);


CreateLabelFromTextField(newVertexNode.getVertex().getData(),
newVertexNode, newVertexNode.getLabelField());

                                isCreatingVertex = false;

                            } catch (InvalidVertexException exception) {

loggerInstance.printMessage(getClass().getName(), "Vertex with the same
name already exists! Try another name.", true);
                            }
                        }
                    } else if (e.getCode() == KeyCode.ESCAPE &&
isCreatingVertex) {

graphPane.getChildren().remove(newVertexNode.getLabelField());
                        graphPane.getChildren().remove(newVertexNode);
                        isCreatingVertex = false;
                    }
                });

                graphPane.setOnMouseEntered(event -> {
                    if (!event.isPrimaryButtonDown()) {
                        graphPane.getScene().setCursor(Cursor.HAND);
                    }
                });

                graphPane.setOnMouseExited(event -> {
                    if (!event.isPrimaryButtonDown()) {
                        graphPane.getScene().setCursor(Cursor.DEFAULT);
                    }
                });

            }
            case EDGE_PLACEMENT -> {
                resetModes();
```

```
                graphPane.disableVertexDrag();

                graphPane.setOnMousePressed((e) -> {

                    if (e.isPrimaryButtonDown() && currentNode == 0) {
                        if
(e.getPickResult().getIntersectedNode().getClass() == FXVertexNode.class)
{
                            firstNode = (FXVertexNode)
e.getPickResult().getIntersectedNode();

                            firstNode.addStyleClass("vertex-selected");

                            currentNode = 1;
                        } else {

loggerInstance.printMessage(getClass().getName(), "Not a vertex! Pointed
object is: " + e.getPickResult().getIntersectedNode().getClass());
                        }
                    } else if (e.isPrimaryButtonDown() && currentNode ==
1) {
                        if
(e.getPickResult().getIntersectedNode().getClass() == FXVertexNode.class)
{
                            secondNode = (FXVertexNode)
e.getPickResult().getIntersectedNode();

                            if (firstNode == secondNode) {

loggerInstance.printMessage(getClass().getName(), "You chose the same
vertices! Try another again.", true);
                            } else if
(graphPane.getGraph().areConnected(firstNode.getVertex(),
secondNode.getVertex())) {

loggerInstance.printMessage(getClass().getName(), "Those Vertices are
already connected! Try another pair.", true);
                            } else {
                                secondNode.addStyleClass("vertex-
selected");

                                if (graphPane.getGraph().getClass() ==
DirectedGraphList.class &&
graphPane.getGraph().areConnected(secondNode.getVertex(),
firstNode.getVertex())) {

                                    movedOppositeEdge = true;
```

```
                                    connectedEdge =
graphPane.getGraph().getEdge(secondNode.getVertex(),
firstNode.getVertex());
                                    connectedEdgeNodeWeight =
graphPane.getEdgeMap().get(connectedEdge).getLabel().getText();


graphPane.removeEdgeNode(graphPane.getEdgeMap().get(connectedEdge));

graphPane.getEdgeMap().remove(connectedEdge);

                                    var fxConnectedEdge = new
FXEdgeCurve(connectedEdge, secondNode, firstNode, -1,
graphPane.getGraphViewProperties().getCurveEdgeAngle());
                                    var fxConnectedEdgeLabel = new
Label(connectedEdgeNodeWeight);

fxConnectedEdge.setLabel(fxConnectedEdgeLabel);


                                    if
(graphPane.getGraphViewProperties().isNeedArrows()) {
                                        var arrow = new
Arrow(graphPane.getGraphViewProperties().getArrowSize());
                                        fxConnectedEdge.setArrow(arrow);
                                    }

                                    newEdgeNode = new
FXEdgeCurve(firstNode, secondNode, 1,
graphPane.getGraphViewProperties().getCurveEdgeAngle());

                                    if
(graphPane.getGraphViewProperties().isNeedArrows()) {
                                        var arrow = new
Arrow(graphPane.getGraphViewProperties().getArrowSize());
                                        newEdgeNode.setArrow(arrow);

graphPane.getChildren().add(arrow);
                                    }

                                    newEdgeNode.addStyleClass("edge-in-
creation");


graphPane.addEdgeNode(fxConnectedEdge, connectedEdge);


graphPane.getChildren().add((FXEdgeCurve) newEdgeNode);
```

```
                                } else {
                                    newEdgeNode = new
FXEdgeLine(firstNode, secondNode);

                                    if
(graphPane.getGraphViewProperties().isNeedArrows()) {
                                        var arrow = new
Arrow(graphPane.getGraphViewProperties().getArrowSize());
                                        newEdgeNode.setArrow(arrow);

graphPane.getChildren().add(arrow);
                                    }

                                    newEdgeNode.addStyleClass("edge-in-
creation");

graphPane.getChildren().add((FXEdgeLine) newEdgeNode);
                                }

                                graphPane.takeVerticesOnFront();

                                TextField edgeDataInput =
createTextField();
                                newEdgeNode.setLabelField(edgeDataInput);

graphPane.getChildren().add(edgeDataInput);

                                currentNode = 2;
                            }

                        } else {

loggerInstance.printMessage(getClass().getName(), "Not a vertex! Pointed
object is: " + e.getPickResult().getIntersectedNode().getClass());
                        }
                    }
                });

                graphPane.setOnKeyPressed((e) -> {
                    if (e.getCode() == KeyCode.ENTER && currentNode == 2)
{

                        int data;
                        Edge newEdge;

                        try {
                            data =
Integer.parseInt(newEdgeNode.getLabelField().getText());

                            if (data <= 0) {
```

```
loggerInstance.printMessage(getClass().getName(), "Entered value is less
then 0 or equal it! Try again.", true);
                            } else {
                                newEdge =
graphPane.getGraph().insertEdge(firstNode.getVertex(),
secondNode.getVertex(), data);

                                newEdgeNode.setEdge(newEdge);
                                graphPane.getEdgeMap().put(newEdge,
newEdgeNode);


CreateLabelFromTextField(Integer.toString(data), newEdgeNode,
newEdgeNode.getLabelField());

                                newEdgeNode.removeStyleClass("edge-in-
creation");
                                firstNode.removeStyleClass("vertex-
selected");
                                secondNode.removeStyleClass("vertex-
selected");

                                currentNode = 0;
                            }
                        } catch (NumberFormatException
numberFormatException) {

loggerInstance.printMessage(getClass().getName(), "Entered value is not a
number! Try again.", true);
                        }

                    } else if (e.getCode() == KeyCode.ESCAPE &&
currentNode == 2) {
                        firstNode.removeStyleClass("vertex-selected");
                        secondNode.removeStyleClass("vertex-selected");

graphPane.getChildren().remove(newEdgeNode.getLabelField());

graphPane.getChildren().remove(newEdgeNode.getArrow());
                        graphPane.getChildren().remove(newEdgeNode);
                        currentNode = 0;

                        if (graphPane.getGraph().getClass() ==
DirectedGraphList.class && movedOppositeEdge) {

                            var fxEdgeCurve =
graphPane.getEdgeMap().get(connectedEdge);
```

```
graphPane.getChildren().remove(fxEdgeCurve.getLabelField());

graphPane.getChildren().remove(fxEdgeCurve.getArrow());

graphPane.getChildren().remove(fxEdgeCurve.getLabel());
                            graphPane.getChildren().remove(fxEdgeCurve);

                            graphPane.getEdgeMap().remove(connectedEdge);

                            var fxEdgeLine = new
FXEdgeLine(connectedEdge,
graphPane.getVertexMap().get(connectedEdge.getVertexOutbound()),
graphPane.getVertexMap().get(connectedEdge.getVertexInbound()));

                            var label = new
Label(connectedEdgeNodeWeight);
                            fxEdgeLine.setLabel(label);
                            graphPane.getChildren().add(label);

                            if
(graphPane.getGraphViewProperties().isNeedArrows()) {
                                var arrow = new
Arrow(graphPane.getGraphViewProperties().getArrowSize());
                                fxEdgeLine.setArrow(arrow);
                                graphPane.getChildren().add(arrow);
                            }

                            graphPane.getChildren().add(fxEdgeLine);
                            graphPane.getEdgeMap().put(connectedEdge,
fxEdgeLine);

                            graphPane.takeVerticesOnFront();
                        }

                        movedOppositeEdge = false;

                    }
                });
            }
            case VERTEX_CHOOSE -> {
                resetModes();

                graphPane.disableVertexDrag();

                graphPane.setOnMousePressed((e) -> {

                    if (e.isPrimaryButtonDown() && nodesSelected1 == 0) {
```

```
                        if
(e.getPickResult().getIntersectedNode().getClass() == FXVertexNode.class)
{
                            firstNodeSelect = (FXVertexNode)
e.getPickResult().getIntersectedNode();

                            firstNodeSelect.addStyleClass("vertex-
selected-for-algorithm-1");

                            nodesSelected1 = 1;
                        } else {

loggerInstance.printMessage(getClass().getName(), "Not a vertex! Pointed
object is: " + e.getPickResult().getIntersectedNode().getClass());
                        }
                    } else if (e.isPrimaryButtonDown() && nodesSelected1
== 1) {
                        if
(e.getPickResult().getIntersectedNode().getClass() == FXVertexNode.class)
{
                            secondNodeSelect = (FXVertexNode)
e.getPickResult().getIntersectedNode();

                            if (firstNodeSelect == secondNodeSelect) {
                                firstNodeSelect.removeStyleClass("vertex-
selected-for-algorithm-1");
                                firstNodeSelect = null;
                                nodesSelected1 = 0;
                            } else {
                                secondNodeSelect.addStyleClass("vertex-
selected-for-algorithm-2");

                                nodesSelected1 = 2;
                            }
                        } else {

loggerInstance.printMessage(getClass().getName(), "Not a vertex! Pointed
object is: " + e.getPickResult().getIntersectedNode().getClass());
                        }
                    } else if (e.isPrimaryButtonDown() && nodesSelected1
== 2) {
                        if
(e.getPickResult().getIntersectedNode().getClass() == FXVertexNode.class)
{
                            var node = (FXVertexNode)
e.getPickResult().getIntersectedNode();

                            if (node == firstNodeSelect) {
                                firstNodeSelect.removeStyleClass("vertex-
selected-for-algorithm-1");
```

```
secondNodeSelect.removeStyleClass("vertex-selected-for-algorithm-2");
                                firstNodeSelect = secondNodeSelect;
                                firstNodeSelect.addStyleClass("vertex-
selected-for-algorithm-1");
                                secondNodeSelect = null;
                                nodesSelected1 = 1;
                            } else if (node == secondNodeSelect) {

secondNodeSelect.removeStyleClass("vertex-selected-for-algorithm-2");
                                secondNodeSelect = null;
                                nodesSelected1 = 1;
                            }
                        } else {

loggerInstance.printMessage(getClass().getName(), "Not a vertex! Pointed
object is: " + e.getPickResult().getIntersectedNode().getClass());
                        }
                    }
                });

            }
        }
    }

    /**
     * Adds label for algorithm data to be stored in.
     *
     * @param startVertex vertex link to which to pin new label
     */
    public void addAlgLabels(Vertex startVertex) {

        for (var fxVertex : graphPane.getVertexMap().values()) {

            Label algLabel;

            if (fxVertex.getVertex().equals(startVertex)) {
                algLabel = new Label("0");
            } else {
                algLabel = new Label("inf");
            }

            fxVertex.setAlgLabel(algLabel);

            graphPane.getChildren().add(algLabel);

            algLabels.add(algLabel);
        }
    }
```

```
    /**
     * Removes all algorithm labels
     */
    public void removeAlgLabels() {

        if (algLabels.size() == 0)
            throw new RuntimeException("You need to create labels before
removing them.");

        for (var label : algLabels) {
            graphPane.getChildren().remove(label);
        }

        algLabels.clear();
    }

    /**
     * Restores initial data in all algorithm labels
     */
    public void clearAlgLabels() {
        for (var fxVertex : graphPane.getVertexMap().values()) {

            Label algLabel = fxVertex.getAlgLabel();

            if (!algLabel.getText().equals("0")) {
                algLabel.setText("inf");
            }
        }
    }


    public ViewMode getViewMode() {
        return viewMode;
    }

    public GraphPane getGraphPane() {
        return graphPane;
    }

    public boolean isCreatingVertex() {
        return isCreatingVertex;
    }

    public FXVertexNode getNewVertexNode() {
        return newVertexNode;
    }

    public int getCurrentNode() {
        return currentNode;
    }
```

```
    public FXEdge getNewEdgeNode() {
        return newEdgeNode;
    }

    public FXVertexNode getFirstNode() {
        return firstNode;
    }

    public FXVertexNode getSecondNode() {
        return secondNode;
    }

    public int getNodesSelected() {
        return nodesSelected;
    }

    public FXVertexNode getFirstNodeSelect() {
        return firstNodeSelect;
    }

    public FXVertexNode getSecondNodeSelect() {
        return secondNodeSelect;
    }

    public int getNodesSelected1() {
        return nodesSelected1;
    }
}
```

GraphPainter.java

```
package ru.etu.graphview.drawing;

import ru.etu.graph.Edge;
import ru.etu.graph.Vertex;
import ru.etu.graphview.GraphPane;
import ru.etu.graphview.base.FXEdge;
import ru.etu.graphview.base.FXVertexNode;
import ru.etu.graphview.base.Label;
import ru.etu.graphview.drawing.multithreading.MyRunnable;
import ru.etu.graphview.drawing.multithreading.ResourceLock;
import ru.etu.graphview.drawing.multithreading.ThreadA;
import ru.etu.graphview.drawing.multithreading.ThreadB;

import java.util.ArrayList;
import java.util.List;

/**
 * This class manages all major changes in GraphPane components.
Implements singleton pattern
```

```
 */
public class GraphPainter {
    private static GraphPainter instance;

    private final int PATH_ANIMATION_DELAY = 200;

    private ResourceLock loopResource; // Resources for threads
    private ThreadA threadA;
    private ThreadB threadB;

    private GraphPane graphPane;

    /**
     * Gets instance of this class (with initialising if necessary )
     *
     * @param graphPane GraphPane link
     * @return GraphPainter instance
     */
    public static GraphPainter getInstance(GraphPane graphPane) {
        if (instance == null) {
            instance = new GraphPainter(graphPane);
        }

        return instance;
    }

    private GraphPainter(GraphPane graphPane) {
        this.graphPane = graphPane;
    }

    /**
     * Animates path by coloring edges one by one with certain delay
     *
     * @param path list of vertex names on the path
     */
    public void animatePath(List<String> path) {

        var eventsList = new ArrayList<MyRunnable>();

        eventsList.add(new MyRunnable() {
            @Override
            public void interrupt() {

            }

            @Override
            public void run() {

addMarkVertex(graphPane.getGraph().getVertex(path.get(0)),
DrawingType.PATH_STROKE);
```

```
            }
        });

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

            int finalI = i;
            eventsList.add(new MyRunnable() {
                @Override
                public void interrupt() {

                }

                @Override
                public void run() {
                    var edge =
graphPane.getGraph().getEdge(path.get(finalI), path.get(finalI + 1));

                    addMarkEdge(edge, true);

addMarkVertex(graphPane.getGraph().getVertex(path.get(finalI + 1)),
DrawingType.PATH_STROKE);
                }
            });

        }


        loopResource = new ResourceLock(eventsList,
PATH_ANIMATION_DELAY);

        threadA = new ThreadA(loopResource);
        threadB = new ThreadB(loopResource);

        threadA.start();
        threadB.start();
    }

    /**
     * Removes marking of the path, created by animatePath(...)
     *
     * @param path list of vertex names on the path
     */
    public void removePathMark(List<String> path) {
        stopPathAnimation();
        for (int i = 0; i < path.size() - 1; i++) {
            var edge = graphPane.getGraph().getEdge(path.get(i),
path.get(i + 1));


            removeMarkEdge(edge, true);
```

```
            removeMarkVertex(graphPane.getGraph().getVertex(path.get(i)),
DrawingType.PATH_STROKE);
        }

removeMarkVertex(graphPane.getGraph().getVertex(path.get(path.size() -
1)), DrawingType.PATH_STROKE);
    }

    /**
     * Stops running animation
     */
    public void stopPathAnimation() {
        threadA.interrupt();
        threadB.interrupt();
    }

    /**
     * Marks vertex according to the drawing type
     *
     * @param vertex      vertex link
     * @param drawingType type of drawing
     */
    public void addMarkVertex(Vertex vertex, DrawingType drawingType) {
        var fxVertex = graphPane.getVertexMap().get(vertex);

        switch (drawingType) {
            case PATH_STROKE -> {
                fxVertex.addStyleClass("vertex-path");
            }
            case CHECK_STROKE -> {
                fxVertex.addStyleClass("vertex-check");
            }
            case UPDATE_FILL -> {
                fxVertex.addStyleClass("vertex-update");
            }
        }

    }

    /**
     * Removes vertex marking according to the drawing type
     *
     * @param vertex      vertex link
     * @param drawingType type of drawing was used for marking
     */
    public void removeMarkVertex(Vertex vertex, DrawingType drawingType)
{
        var fxVertex = graphPane.getVertexMap().get(vertex);

        switch (drawingType) {
```

```
            case PATH_STROKE -> {
                fxVertex.removeStyleClass("vertex-path");
            }
            case CHECK_STROKE -> {
                fxVertex.removeStyleClass("vertex-check");
            }
            case UPDATE_FILL -> {
                fxVertex.removeStyleClass("vertex-update");
            }
        }
    }

    /**
     * Marks edge
     *
     * @param edge   edge link
     * @param isPath is marking for path mark
     */
    public void addMarkEdge(Edge edge, boolean isPath) {
        var fxEdge = graphPane.getEdgeMap().get(edge);

        if (isPath) {
            fxEdge.addStyleClass("edge-path");
            if (fxEdge.getArrow() != null)
                fxEdge.getArrow().addStyleClass("arrow-path");
        } else {
            fxEdge.addStyleClass("edge-check");
            if (fxEdge.getArrow() != null)
                fxEdge.getArrow().addStyleClass("arrow-check");
        }

    }

    /**
     * Removes edge marking
     *
     * @param edge   edge link
     * @param isPath was marking for path mark
     */
    public void removeMarkEdge(Edge edge, boolean isPath) {
        var fxEdge = graphPane.getEdgeMap().get(edge);

        if (isPath) {
            fxEdge.removeStyleClass("edge-path");
            if (fxEdge.getArrow() != null)
                fxEdge.getArrow().removeStyleClass("arrow-path");
        } else {
            fxEdge.removeStyleClass("edge-check");
            if (fxEdge.getArrow() != null)
                fxEdge.getArrow().removeStyleClass("arrow-check");
```

```
        }
    }

    /**
     * Adds label mark
     *
     * @param label      label link
     * @param isChecking is marking for CHECK state
     */
    public void addMarkLabel(Label label, boolean isChecking) {
        if (isChecking) {
            label.addStyleClass("checking-label");
        } else {
            label.addStyleClass("updating-label");
        }
    }

    /**
     * Removes label mark
     *
     * @param label      label link
     * @param isChecking was marking for CHECK state
     */
    public void removeMarkLabel(Label label, boolean isChecking) {
        if (isChecking) {
            label.removeStyleClass("checking-label");
        } else {
            label.removeStyleClass("updating-label");
        }
    }
}
```

ViewMode.java

```
package ru.etu.graphview.drawing;

public enum ViewMode {
    NORMAL, // Only vertex drag
    VERTEX_PLACEMENT, // Vertex drag + vertex creation
    EDGE_PLACEMENT, // Edge creation
    VERTEX_CHOOSE // Choosing 2 vertices for algorithm
}
```

Stylable.java

```
package ru.etu.graphview.styling;

/**
 * Interface that helps to easily apply styles to Shapes
 *
 * @see StyleEngine
```

```
 */
public interface Stylable {

    /**
     * Sets new style. Old style will be deleted.
     *
     * @param css new style in css format
     */
    void setStyleCss(String css);

    /**
     * Sets new style class. Old style classes will be deleted.
     *
     * @param cssClassName new style class names
     */
    void setStyleClass(String cssClassName);

    /**
     * Adds new style class. Old style classes will not be deleted.
     *
     * @param cssClassName new style class names
     */
    void addStyleClass(String cssClassName);

    /**
     * Removes style class
     *
     * @param cssClassName style class name
     */
    boolean removeStyleClass(String cssClassName);


}
```

StyleEngine.java

```
package ru.etu.graphview.styling;

import javafx.scene.shape.Shape;

/**
 * General implementation of Stylable interface for all objects that
needed it.
 *
 * @see Stylable
 */
public class StyleEngine implements Stylable {

    private final Shape element;

    public StyleEngine(Shape element) {
```

```
        this.element = element;
    }

    @Override
    public void setStyleCss(String css) {
        element.setStyle(css);
    }

    @Override
    public void setStyleClass(String cssClassName) {
        element.getStyleClass().clear();
        element.setStyle(null);
        element.getStyleClass().add(cssClassName);
    }

    @Override
    public void addStyleClass(String cssClassName) {
        element.getStyleClass().add(cssClassName);
    }

    @Override
    public boolean removeStyleClass(String cssClassName) {
        return element.getStyleClass().remove(cssClassName);
    }
}
```

GraphPane.java

```
package ru.etu.graphview;

import javafx.beans.property.DoubleProperty;
import javafx.beans.property.ReadOnlyDoubleWrapper;
import javafx.css.StyleClass;
import javafx.scene.Node;
import javafx.scene.layout.Pane;
import ru.etu.graph.Edge;
import ru.etu.graph.Graph;
import ru.etu.graph.Vertex;
import ru.etu.graphview.base.*;
import ru.etu.graphview.styling.Stylable;
import ru.etu.graphview.styling.StyleEngine;

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

/**
 * Main place for all graph view elements to base on.
 * It is responsible for managing storage and view/interaction properties
 */
```

```
public class GraphPane extends Pane {

    // Stored view properties
    private GraphViewProperties graphViewProperties;

    private Graph graph;
    private Map<Vertex, FXVertexNode> vertexMap;
    private Map<Edge, FXEdge> edgeMap;

    private StyleEngine styleEngine;

    // Controls if graph was set
    private boolean graphSet = false;

    public GraphPane() {
        edgeMap = new HashMap<>();
        vertexMap = new HashMap<>();

        getStylesheets().add("graphStyles.css");
        getStyleClass().add("graph-pane");
    }

    /**
     * Inserts graph to GraphPane. Creates all necessary view components.
     *
     * @param graph      graph link
     * @param properties view properties
     */
    public void loadGraph(Graph graph, GraphViewProperties properties) {
        clearPane();

        this.graph = graph;
        graphViewProperties = properties;

        for (var vertex : graph.getVertices()) {
            FXVertexNode fxVertex = new FXVertexNode(vertex, 0, 0,
                    properties.getVerticesRadius());
            vertexMap.put(vertex, fxVertex);
        }

        for (var edge : graph.getEdges()) {
            var fxVertexFrom = vertexMap.get(edge.getVertexOutbound());
            var fxVertexTo = vertexMap.get(edge.getVertexInbound());

            if (properties.isNeedDoubleEdges()) {
                var backwardsEdge = graph.getEdge(fxVertexTo.getVertex(),
fxVertexFrom.getVertex());
                if (backwardsEdge != null) {
                    if (!edgeMap.containsKey(backwardsEdge) && !
edgeMap.containsKey(edge)) {
```

```
                        var fxBackwardsEdge = new
FXEdgeCurve(backwardsEdge, fxVertexTo, fxVertexFrom, -1,
graphViewProperties.getCurveEdgeAngle());
                        var fxEdge = new FXEdgeCurve(edge, fxVertexFrom,
fxVertexTo, 1, graphViewProperties.getCurveEdgeAngle());

                        addEdgeNode(fxBackwardsEdge, backwardsEdge);
                        addEdgeNode(fxEdge, edge);
                    }
                } else {
                    var fxEdge = new FXEdgeLine(edge, fxVertexFrom,
fxVertexTo);

                    addEdgeNode(fxEdge, edge);
                }
            } else {
                var fxEdge = new FXEdgeLine(edge, fxVertexFrom,
fxVertexTo);

                addEdgeNode(fxEdge, edge);
            }
        }

        for (var vertexNode : vertexMap.values()) {
            addVertexNode(vertexNode);
        }

        RandomStrategy.place(
                this.widthProperty().doubleValue(),
                this.heightProperty().doubleValue(),
                vertexMap.values());

        graphSet = true;
    }

    /**
     * Clears GraphPane by deleting all view objects and all vertices
from the graph
     * (edges delete automatically)
     */
    public void clearPane() {
        for (var edgeNode : edgeMap.values()) {
            removeEdgeNode(edgeNode);
        }

        for (var vertexNode : vertexMap.values()) {
            removeVertexNode(vertexNode);
        }

        for (var vertex : vertexMap.keySet()) {
```

```
            graph.removeVertex(vertex);
        }

        edgeMap.clear();
        vertexMap.clear();
    }

    /**
     * Adds vertex view and it's label to the screen
     *
     * @param vertexNode vertex view
     */
    private void addVertexNode(FXVertexNode vertexNode) {
        this.getChildren().add(vertexNode);

        Label label = new Label(vertexNode.getVertex().getData());
        vertexNode.setLabel(label);
        this.getChildren().add(label);
    }

    /**
     * Adds edge view and it's label (and arrow if needed) to the screen
     *
     * @param fxEdge edge view
     * @param edge   edge link
     */
    public void addEdgeNode(FXEdge fxEdge, Edge edge) {
        if (fxEdge.getClass() == FXEdgeLine.class) {
            this.getChildren().add((FXEdgeLine) fxEdge);
        } else if (fxEdge.getClass() == FXEdgeCurve.class) {
            this.getChildren().add((FXEdgeCurve) fxEdge);
        } else {
            throw new RuntimeException("Can't determine edge class: " +
fxEdge.getClass());
        }

        edgeMap.put(edge, fxEdge);

        //var newWeight = fxEdge.getEdge().getData();

        Label label = new
Label(Integer.toString(fxEdge.getEdge().getData()));
        fxEdge.setLabel(label);
        this.getChildren().add(label);

        if (graphViewProperties.isNeedArrows()) {
            Arrow arrow = new Arrow(graphViewProperties.getArrowSize());
            fxEdge.setArrow(arrow);
            this.getChildren().add(arrow);
        }
```

```
    }

    /**
     * Removes vertex view from the screen with all its additional
objects
     *
     * @param vertexNode vertex view
     */
    public void removeVertexNode(FXVertexNode vertexNode) {
        this.getChildren().remove(vertexNode);
        this.getChildren().remove(vertexNode.getLabel());

        if (vertexNode.getAlgLabel() != null) {
            this.getChildren().remove(vertexNode.getAlgLabel());
        }
    }

    /**
     * Removes edge view from the screen with all its additional objects
     *
     * @param fxEdge edge view
     */
    public void removeEdgeNode(FXEdge fxEdge) {
        this.getChildren().remove(fxEdge);
        this.getChildren().remove(fxEdge.getLabel());
        if (fxEdge.getArrow() != null) {
            this.getChildren().remove(fxEdge.getArrow());
        }
    }

    /**
     * Enables dragging mechanism in vertices
     */
    public void enableVertexDrag() {
        for (var fxVertex : vertexMap.values()) {
            fxVertex.enableDrag();
        }
    }

    /**
     * Disables dragging mechanism in vertices
     */
    public void disableVertexDrag() {
        for (var fxVertex : vertexMap.values()) {
            fxVertex.disableDrag();
        }
    }

    /**
```

```
     * Replaces vertices, so they were on top of any other object in
GraphPane
     */
    public void takeVerticesOnFront() {
        // Replacing vertices so that they were on front
        List<Node> vertexNodes = new ArrayList<>();

        for (var node : this.getChildren()) {
            if (node.getClass() == FXVertexNode.class) {
                vertexNodes.add(node);
            }
        }

        for (var vertexNode : vertexNodes) {
            this.getChildren().remove(vertexNode);
            this.getChildren().add(vertexNode);
        }
    }

    public GraphViewProperties getGraphViewProperties() {
        return graphViewProperties;
    }

    public Graph getGraph() {
        return graph;
    }

    public Map<Vertex, FXVertexNode> getVertexMap() {
        return vertexMap;
    }

    public Map<Edge, FXEdge> getEdgeMap() {
        return edgeMap;
    }

    public boolean isGraphSet() {
        return graphSet;
    }
}
```

GraphViewProperties.java

```
package ru.etu.graphview;

/**
 * This class stores some view properties for the graph.
 */
public class GraphViewProperties {

    /**
     * Radius of vertices in pixels
```

```
 */
private double verticesRadius = 15;

/**
 * Max angle of bending for curve lines
 */
private double curveEdgeAngle = 30;

/**
 * Sets if arrows must be placed
 */
private boolean needArrows = true;

/**
 * Sets if double edges can be placed
 */
private boolean needDoubleEdges = true;

/**
 * Arrow size in pixels
 */
private int arrowSize = 9;

/**
 * Minimum scale factor. Must be a multiple of scaleStep
 */
private double minScale = 0.25;

/**
 * Maximum scale factor. Must be a multiple of scaleStep
 */
private double maxScale = 3;

/**
 * Step of scaling
 */
private double scaleStep = 0.25;

public double getVerticesRadius() {
    return verticesRadius;
}

public void setVerticesRadius(double verticesRadius) {
    this.verticesRadius = verticesRadius;
}

public double getCurveEdgeAngle() {
    return curveEdgeAngle;
}
```

```
public void setCurveEdgeAngle(double curveEdgeAngle) {
    this.curveEdgeAngle = curveEdgeAngle;
}

public boolean isNeedArrows() {
    return needArrows;
}

public void setNeedArrows(boolean needArrows) {
    this.needArrows = needArrows;
}

public int getArrowSize() {
    return arrowSize;
}

public void setArrowSize(int arrowSize) {
    this.arrowSize = arrowSize;
}

public boolean isNeedDoubleEdges() {
    return needDoubleEdges;
}

public void setNeedDoubleEdges(boolean needDoubleEdges) {
    this.needDoubleEdges = needDoubleEdges;
}

public double getMinScale() {
    return minScale;
}

public void setMinScale(double minScale) {
    this.minScale = minScale;
}

public double getMaxScale() {
    return maxScale;
}

public void setMaxScale(double maxScale) {
    this.maxScale = maxScale;
}

public double getScaleStep() {
    return scaleStep;
}

public void setScaleStep(double scaleStep) {
    this.scaleStep = scaleStep;
```

```
    }
}
```

RandomStrategy.java

```
package ru.etu.graphview;

import ru.etu.graph.Graph;
import ru.etu.graphview.base.FXVertex;

import java.util.Collection;
import java.util.Random;

/**
 * This class needs to randomly place vertices (their views) on the plot
 */
public class RandomStrategy {

    /**
     * Randomly places vertices (their views) on the plot
     *
     * @param width      - width of the plot
     * @param height     - width of the plot
     * @param fxVertices - list of vertices (must implement FXVertex)
     */
    public static void place(double width, double height, Collection<?
extends FXVertex> fxVertices) {

        Random random = new Random();

        for (var vertex : fxVertices) {
            double x = random.nextDouble() * width;

            if (x < (vertex.getRadius() + 40)) x = (vertex.getRadius() +
40);
            else if (x > width - (vertex.getRadius() + 40)) x = width -
(vertex.getRadius() + 40);

            double y = random.nextDouble() * height;

            if (y < (vertex.getRadius() + 40)) y = (vertex.getRadius() +
40);
            else if (y > height - (vertex.getRadius() + 40)) y = height -
(vertex.getRadius() + 40);

            vertex.setPosition(x, y);
        }
    }
}
```

IOGraph.java

```
package ru.etu.io;

import com.google.gson.stream.JsonReader;
import com.google.gson.stream.JsonWriter;
import javafx.util.Pair;
import ru.etu.graph.DirectedGraph;
import ru.etu.graph.Edge;
import ru.etu.graph.Graph;
import ru.etu.graph.GraphList;
import com.google.gson.Gson;

import java.io.*;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.List;

public class IOGraph {
    public IOGraph() {
    }

    public Graph loadGraph(String name) throws IOException {
        try (JsonReader reader = new JsonReader(new
InputStreamReader(Files.newInputStream(Paths.get(name)),
StandardCharsets.UTF_8))){
            Gson gson = new Gson();
            //JsonReader reader = new JsonReader(new
InputStreamReader(Files.newInputStream(Paths.get(name)),
StandardCharsets.UTF_8));
            JSONGraph savedGraph = gson.fromJson(reader,
JSONGraph.class);
            //reader.close();
            Graph newGraph = savedGraph.getGraph();
            verifyGraph(newGraph);
            return newGraph;
        } catch (Exception ex) {
            throw new IOException(ex.getLocalizedMessage());
        }
    }

    public void saveGraph(String name, Graph graph, boolean isOriented)
throws IOException {
        Gson gson = new Gson();
        JSONGraph graphToSave = new JSONGraph(graph, isOriented);
        Path filePath = Paths.get(name);
        if (!filePath.toFile().exists() && filePath.getParent() != null)
{
            Files.createDirectories(filePath.getParent());
```

```
        }

        JsonWriter writer = new JsonWriter(new
OutputStreamWriter(Files.newOutputStream(filePath),
StandardCharsets.UTF_8));
        gson.toJson(graphToSave, JSONGraph.class, writer);
        writer.close();
    }

    public void saveGraph(String name, Graph graph) throws IOException {
        saveGraph(name, graph, (graph instanceof DirectedGraph));
    }

    private void verifyGraph(Graph graph) throws IOException{
        for (Edge edge : graph.getEdges()){
            if (edge.getData()<=0){
                throw new IOException("Invalid file: found an edge with
non-positive weight!");
            }
        }
    }
}
```

JSONGraph.java

```
package ru.etu.io;

import ru.etu.graph.*;

import java.util.List;

public class JSONGraph {
    private boolean isOriented;
    private List<Vertex> vertices;
    private List<Edge> edges;

    public JSONGraph(Graph graph, boolean isOriented) {
        this.isOriented = isOriented;
        this.vertices = graph.getVertices();
        this.edges = graph.getEdges();
    }

    public Graph getGraph() {
        Graph newGraph = (isOriented ? new DirectedGraphList() : new
GraphList());
        for (Vertex vertex : vertices) {
            newGraph.insertVertex(vertex.getData());
        }
        for (Edge edge : edges) {
            newGraph.insertEdge(edge.getVertexOutbound().getData(),
edge.getVertexInbound().getData(), edge.getData());
```

```
        }
        return newGraph;
    }


}
```

Logger.java

```
package ru.etu.logger;

import javafx.application.Platform;
import javafx.scene.control.ButtonType;
import ru.etu.controllers.LoggerView;
import javafx.scene.control.Alert;

public class Logger {

    private static Logger instance;

    private LoggerView loggerView;

    private Logger() {}
    private Logger(LoggerView view) {
        loggerView = view;
    }

    public static Logger getInstance() {
        if (instance == null)
            instance = new Logger();
        return instance;
    }

    public static void initialiseInstance(LoggerView view) {
        if (view == null)
            throw new IllegalArgumentException("View can't be null.");
        if (instance == null) {
            instance = new Logger(view);
        } else if (instance.loggerView == null) {
            instance.loggerView = view;
        }
    }

    public void printMessage(String className, String message) {
        loggerView.printMessage(className, message);
    }

    public void printMessage(String className, String message, boolean
isError){
        if(isError){
```

```
            Alert err = new Alert(Alert.AlertType.ERROR, message,
ButtonType.OK);
            err.showAndWait();
        }
        printMessage(className, message);
    }

    public void printMessage(String className, String message, boolean
isError, Alert.AlertType type){
        Platform.runLater(() -> {
            if(isError){
                Alert err = new Alert(type, message, ButtonType.OK);
                err.show();
//              err.showAndWait();
            }
            printMessage(className, message);
        });
    }
}
```

MainApplication.java

```
package ru.etu.studypract;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;
import ru.etu.controllers.App;

import java.io.IOException;

public class MainApplication extends Application {
    @Override
    public void start(Stage stage) throws IOException {

        FXMLLoader appFxmlLoader = new
FXMLLoader(getClass().getResource("/ru/etu/studypract/app.fxml"));

        Scene appScene = new Scene(appFxmlLoader.load());
        stage.setTitle("Dijkstra Visualiser");
        stage.setScene(appScene);
        stage.show();


        stage.setResizable(false);
        App appController = appFxmlLoader.getController();
        appController.closeWindow(stage);
    }
```

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

SuperMain.java

```
package ru.etu.studypract;

public class SuperMain {

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

module-info.java

```
module ru.etu.studypract {
    requires javafx.controls;
    requires javafx.fxml;

    requires com.google.gson;
    requires java.datatransfer;
    requires java.desktop;

    opens ru.etu.controllers to javafx.fxml;
    opens ru.etu.io to com.google.gson;
    opens ru.etu.graph to com.google.gson;

    exports ru.etu.algoritm;
    exports ru.etu.controllers;
    exports ru.etu.graph;
    exports ru.etu.io;
    exports ru.etu.graphview;
    exports ru.etu.graphview.base;
    exports ru.etu.graphview.drawing;
    exports ru.etu.graphview.styling;
    exports ru.etu.studypract;
}
```

about.fxml

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

<?import javafx.geometry.Insets?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Hyperlink?>
<?import javafx.scene.control.TextArea?>
<?import javafx.scene.image.Image?>
<?import javafx.scene.image.ImageView?>
<?import javafx.scene.layout.BorderPane?>
```

```
<?import javafx.scene.layout.HBox?>
<?import javafx.scene.layout.Pane?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.text.Font?>
<?import javafx.scene.text.Text?>

<BorderPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-
Infinity" minWidth="-Infinity" prefHeight="230.0" prefWidth="600.0"
style="-fx-background-color: #FFFFFF;"
stylesheets="@../../../styles/about.css" xmlns="http://javafx.com/javafx/
18" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="ru.etu.controllers.About">
    <left>
        <VBox alignment="TOP_CENTER" prefHeight="200.0" prefWidth="150.0"
BorderPane.alignment="CENTER">
            <children>
                <ImageView fitHeight="100.0" fitWidth="100.0"
pickOnBounds="true" preserveRatio="true">
                    <image>
                        <Image url="@../../../Icons/MainIcon.png" />
                    </image>
                </ImageView>
            </children>
            <BorderPane.margin>
                <Insets top="30.0" />
            </BorderPane.margin>
        </VBox>
    </left>
    <bottom>
        <HBox alignment="CENTER_RIGHT" prefHeight="50.0" spacing="10.0"
BorderPane.alignment="CENTER">
            <children>
                <Button fx:id="okBtn" mnemonicParsing="false" text="OK" /
>
                <Button fx:id="copyBtn" mnemonicParsing="false"
text="Copy" />
            </children>
            <BorderPane.margin>
                <Insets right="20.0" />
            </BorderPane.margin>
        </HBox>
    </bottom>
   <right>
        <VBox prefHeight="200.0" prefWidth="200.0"
BorderPane.alignment="CENTER">
            <padding>
                <Insets top="10.0" />
            </padding>
        </VBox>
   </right>
```

```
   <center>
      <Pane prefHeight="200.0" prefWidth="200.0"
BorderPane.alignment="CENTER">
         <children>
                <Text layoutY="37.0" strokeType="OUTSIDE"
strokeWidth="0.0" text="  Deikstra Visualiser"
wrappingWidth="449.970703125">
                    <font>
                        <Font name="Dubai Regular" size="20.0" />
                    </font>
                </Text>
                <TextArea fx:id="textArea" editable="false"
layoutY="46.0" prefHeight="135.0" prefWidth="450.0" styleClass="text-
area" text="Build &lt;place version&gt;, built on &lt;place
date&gt;&#10;&#10;Created by students of ETU
&quot;LETI&quot;:&#10;Aristarkhov Ilya - github: &#10;Ragrid Denis -
github: &#10;Kostebelova Elizabeth - github: ">
                    <padding>
                        <Insets top="15.0" />
                    </padding>
                </TextArea>
            <Hyperlink layoutX="140.0" layoutY="112.0" prefHeight="24.0"
prefWidth="169.0" text="https://github.com/ilya201232"
onAction="#arisGitHubLink"/>
            <Hyperlink layoutX="124.0" layoutY="129.0" prefHeight="25.0"
prefWidth="211.0" text="https://github.com/mnelenpridumivat"
onAction="#ragrGitHubLink"/>
            <Hyperlink layoutX="174.0" layoutY="145.0" prefHeight="27.0"
prefWidth="228.0" text="https://github.com/Kostebelova-Elizaveta"
onAction="#kostGitHubLink"/>
         </children>
      </Pane>
   </center>
</BorderPane>
```

app.fxml

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

<?import javafx.geometry.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.image.*?>
<?import javafx.scene.layout.*?>

<AnchorPane fx:id="mainPane" prefHeight="500.0" prefWidth="700.0"
stylesheets="@../../../styles/app.css"
xmlns="http://javafx.com/javafx/16" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="ru.etu.controllers.App">
    <children>
        <MenuBar fx:id="menuBar" prefHeight="25.0" prefWidth="1000.0">
            <menus>
```

```
                <Menu mnemonicParsing="false" text="File">
                    <items>
                        <Menu mnemonicParsing="false" text="New">
                            <items>
                                <MenuItem mnemonicParsing="false"
onAction="#createNewDirectedGraph" text="Directed Graph" />
                                <MenuItem mnemonicParsing="false"
onAction="#createNewGraph" text="Undirected Graph" />
                            </items>
                        </Menu>
                        <SeparatorMenuItem mnemonicParsing="false" />
                        <MenuItem mnemonicParsing="false"
onAction="#loadGraph" text="Open" />
                        <MenuItem fx:id="saveMeMenuItem"
mnemonicParsing="false" onAction="#saveGraph" text="Save" />
                        <MenuItem fx:id="saveAsMenuItem"
mnemonicParsing="false" onAction="#saveGraphAs" text="Save As" />
                        <SeparatorMenuItem mnemonicParsing="false" />
                        <MenuItem fx:id="exitMenuItem"
mnemonicParsing="false" text="Exit" />
                    </items>
                </Menu>
                <Menu mnemonicParsing="false" text="Edit">
                    <items>
                        <MenuItem fx:id="changeMenuItem"
mnemonicParsing="false" text="Change Mode" />
                        <SeparatorMenuItem mnemonicParsing="false" />
                        <Menu fx:id="settingsMenu"
mnemonicParsing="false" text="Settings Mode Instruments">
                            <items>
                                <MenuItem fx:id="moveMenuItem"
mnemonicParsing="false" text="Move" />
                                <MenuItem fx:id="vertexMenuItem"
mnemonicParsing="false" text="Create Vertex" />
                                <MenuItem fx:id="edgeMenuItem"
mnemonicParsing="false" text="Create Edge" />
                                <MenuItem fx:id="chooseMenuItem"
mnemonicParsing="false" text="Choose Vertices" />
                                <MenuItem fx:id="clearMenuItem"
mnemonicParsing="false" text="Clear Graph" />
                            </items>
                        </Menu>
                        <Menu fx:id="playMenu" mnemonicParsing="false"
text="Play Mode Instruments">
                            <items>
                                <MenuItem fx:id="playMenuItem"
mnemonicParsing="false" text="Play" />
                                <MenuItem fx:id="pauseMenuItem"
mnemonicParsing="false" text="Pause" />
```

```
                                <MenuItem fx:id="stopMenuItem"
mnemonicParsing="false" text="Stop" />
                                <MenuItem fx:id="prevMenuItem"
mnemonicParsing="false" text="Prev. Step" />
                                <MenuItem fx:id="nextMenuItem"
mnemonicParsing="false" text="Next Step" />
                            </items>
                        </Menu>
                        <SeparatorMenuItem mnemonicParsing="false" />
                        <MenuItem mnemonicParsing="false"
onAction="#openLogger" text="Open Logs" />
                        <MenuItem mnemonicParsing="false"
onAction="#openSettings" text="Settings" />
                    </items>
                </Menu>
                <Menu mnemonicParsing="false" text="Help">
                    <items>
                        <MenuItem mnemonicParsing="false"
onAction="#openRepo" text="Our repository" />
                        <MenuItem mnemonicParsing="false"
onAction="#openAbout" text="About" />
                    </items>
                </Menu>
            </menus>
        </MenuBar>
        <BorderPane fx:id="contentBorderPane" layoutY="25.0"
prefHeight="700.0" prefWidth="1000.0">
            <top>
                <HBox maxHeight="86.0" minHeight="86.0" prefHeight="86.0"
prefWidth="1000.0" styleClass="top-panel" BorderPane.alignment="CENTER">
                    <children>
                        <Button mnemonicParsing="false"
onAction="#changeMode" prefHeight="76.0" prefWidth="76.0" style="-fx-
background-radius: 400;" styleClass="top-btn">
                            <graphic>
                                <ImageView fitHeight="60.0"
fitWidth="60.0" pickOnBounds="true" preserveRatio="true">
                                    <image>
                                        <Image
url="@../../../Icons/ChangeMode.png" />
                                    </image>
                                </ImageView>
                            </graphic>
                        </Button>
                        <HBox alignment="CENTER" maxHeight="76.0"
minHeight="76.0" prefHeight="76.0" prefWidth="886.0" spacing="10.0">
                            <children>
                                <Button fx:id="prevBtn" maxHeight="76.0"
maxWidth="76.0" minHeight="76.0" minWidth="76.0" mnemonicParsing="false"
```

```
onAction="#stepBack" prefHeight="76.0" prefWidth="76.0" style="-fx-
background-radius: 400;" styleClass="top-btn">
                                    <graphic>
                                        <ImageView fitHeight="60.0"
fitWidth="60.0" pickOnBounds="true" preserveRatio="true">
                                            <image>
                                                <Image
url="@../../../Icons/left-arrow.png" />
                                            </image>
                                        </ImageView>
                                    </graphic>
                                </Button>
                                <ToggleButton fx:id="pauseBtn"
maxHeight="76.0" maxWidth="76.0" minHeight="76.0" minWidth="76.0"
mnemonicParsing="false" onAction="#pause" prefHeight="76.0"
prefWidth="76.0" style="-fx-background-radius: 400;" styleClass="top-
btn">
                                    <toggleGroup>
                                        <ToggleGroup fx:id="PlayGroup" />
                                    </toggleGroup>
                                    <graphic>
                                        <ImageView fitHeight="60.0"
fitWidth="60.0" pickOnBounds="true" preserveRatio="true">
                                            <image>
                                                <Image
url="@../../../Icons/pause.png" />
                                            </image>
                                        </ImageView>
                                    </graphic>
                                </ToggleButton>
                                <ToggleButton fx:id="playBtn"
maxHeight="76.0" maxWidth="76.0" minHeight="76.0" minWidth="76.0"
mnemonicParsing="false" onAction="#play" prefHeight="76.0"
prefWidth="76.0" style="-fx-background-radius: 400;" styleClass="top-btn"
toggleGroup="$PlayGroup">
                                    <graphic>
                                        <ImageView fitHeight="60.0"
fitWidth="60.0" pickOnBounds="true" preserveRatio="true">
                                            <image>
                                                <Image
url="@../../../Icons/Play.png" />
                                            </image>
                                        </ImageView>
                                    </graphic>
                                </ToggleButton>
                                <Button fx:id="stopBtn" maxHeight="76.0"
maxWidth="76.0" minHeight="76.0" minWidth="76.0" mnemonicParsing="false"
onAction="#stop" prefHeight="76.0" prefWidth="76.0" style="-fx-
background-radius: 400;" styleClass="top-btn">
                                    <graphic>
```

```
                                        <ImageView fitHeight="60.0"
fitWidth="60.0" pickOnBounds="true" preserveRatio="true">
                                            <image>
                                                <Image
url="@../../../Icons/stop-button.png" />
                                            </image>
                                        </ImageView>
                                    </graphic>
                                </Button>
                                <Button fx:id="nextBtn" maxHeight="76.0"
maxWidth="76.0" minHeight="76.0" minWidth="76.0" mnemonicParsing="false"
onAction="#stepForward" prefHeight="76.0" prefWidth="76.0" style="-fx-
background-radius: 400;" styleClass="top-btn">
                                    <graphic>
                                        <ImageView fitHeight="60.0"
fitWidth="60.0" pickOnBounds="true" preserveRatio="true">
                                            <image>
                                                <Image
url="@../../../Icons/right-arrow.png" />
                                            </image>
                                        </ImageView>
                                    </graphic>
                                </Button>
                            </children>
                        </HBox>
                    </children>
                    <BorderPane.margin>
                        <Insets />
                    </BorderPane.margin>
                    <padding>
                        <Insets left="5.0" top="5.0" />
                    </padding>
                </HBox>
            </top>
            <left>
                <VBox maxWidth="86.0" minWidth="86.0" prefHeight="619.0"
prefWidth="86.0" styleClass="left-panel" BorderPane.alignment="CENTER">
                    <children>
                        <Button maxHeight="76.0" maxWidth="76.0"
minHeight="76.0" minWidth="76.0" mnemonicParsing="false"
onAction="#changeMode" prefHeight="76.0" prefWidth="76.0" style="-fx-
background-radius: 400;" styleClass="left-btn">
                            <graphic>
                                <ImageView fitHeight="60.0"
fitWidth="60.0" pickOnBounds="true" preserveRatio="true">
                                    <image>
                                        <Image
url="@../../../Icons/ChangeMode.png" />
                                    </image>
                                </ImageView>
```

```
                            </graphic>
                        </Button>
                        <VBox alignment="CENTER" maxWidth="76.0"
minWidth="76.0" prefHeight="512.0" prefWidth="76.0" spacing="10.0">
                            <children>
                                <ToggleButton fx:id="moveBtn"
mnemonicParsing="false" onAction="#setMoveMode" prefHeight="76.0"
prefWidth="76.0" style="-fx-background-radius: 400;" styleClass="left-
btn">
                                    <graphic>
                                        <ImageView fitHeight="84.0"
fitWidth="60.0" preserveRatio="true">
                                            <image>
                                                <Image
url="@../../../Icons/Move.png" />
                                            </image>
                                        </ImageView>
                                    </graphic>
                                    <toggleGroup>
                                        <ToggleGroup
fx:id="SettingsGroup" />
                                    </toggleGroup>
                                </ToggleButton>
                                <ToggleButton fx:id="vertexBtn"
mnemonicParsing="false" onAction="#setVertexCreationMode"
prefHeight="76.0" prefWidth="76.0" style="-fx-background-radius: 400;"
styleClass="left-btn" toggleGroup="$SettingsGroup">
                                    <graphic>
                                        <ImageView fitHeight="60.0"
fitWidth="86.0" pickOnBounds="true" preserveRatio="true">
                                            <image>
                                                <Image
url="@../../../Icons/VertexCreation.png" />
                                            </image>
                                        </ImageView>
                                    </graphic>
                                </ToggleButton>
                                <ToggleButton fx:id="edgeBtn"
mnemonicParsing="false" onAction="#setEdgeCreationMode" prefHeight="76.0"
prefWidth="76.0" style="-fx-background-radius: 400;" styleClass="left-
btn" toggleGroup="$SettingsGroup">
                                    <graphic>
                                        <ImageView fitHeight="60.0"
fitWidth="60.0" pickOnBounds="true" preserveRatio="true">
                                            <image>
                                                <Image
url="@../../../Icons/EdgeCreation.png" />
                                            </image>
                                        </ImageView>
                                    </graphic>
```

```
                                </ToggleButton>
                                <ToggleButton fx:id="chooseBtn"
mnemonicParsing="false" onAction="#setChooseMode" prefHeight="76.0"
prefWidth="76.0" style="-fx-background-radius: 400;" styleClass="left-
btn" toggleGroup="$SettingsGroup">
                                    <graphic>
                                        <ImageView fitHeight="60.0"
fitWidth="60.0" pickOnBounds="true" preserveRatio="true">
                                            <image>
                                                <Image
url="@../../../Icons/ChooseVertices.png" />
                                            </image>
                                        </ImageView>
                                    </graphic>
                                </ToggleButton>
                                <Button fx:id="clearBtn"
mnemonicParsing="false" onAction="#clearGraph" prefHeight="76.0"
prefWidth="76.0" style="-fx-background-radius: 400;" styleClass="left-
btn">
                                    <graphic>
                                        <ImageView fitHeight="60.0"
fitWidth="60.0" pickOnBounds="true" preserveRatio="true">
                                            <image>
                                                <Image
url="@../../../Icons/deleteGraph.png" />
                                            </image>
                                        </ImageView>
                                    </graphic>
                                </Button>
                            </children>
                        </VBox>
                    </children>
                    <BorderPane.margin>
                        <Insets />
                    </BorderPane.margin>
                    <padding>
                        <Insets left="5.0" right="5.0" top="5.0" />
                    </padding>
                </VBox>
            </left>
            <center>
                <Pane fx:id="graphPanePane"
maxHeight="1.7976931348623157E308" maxWidth="1.7976931348623157E308"
styleClass="graph-pane-pane">

                </Pane>
            </center>
        </BorderPane>
    </children>
</AnchorPane>
```

loggerView.fxml

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

<?import javafx.scene.control.TextArea?>
<?import javafx.scene.layout.BorderPane?>

<BorderPane xmlns="http://javafx.com/javafx/18"
xmlns:fx="http://javafx.com/fxml/1"
fx:controller="ru.etu.controllers.LoggerView">
   <center>
          <TextArea fx:id="textArea" editable="false"
BorderPane.alignment="CENTER" />
   </center>
</BorderPane>
```

settings.fxml

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

<?import javafx.geometry.Insets?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.RadioButton?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.control.ToggleGroup?>
<?import javafx.scene.layout.BorderPane?>
<?import javafx.scene.layout.HBox?>
<?import javafx.scene.layout.VBox?>

<BorderPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-
Infinity" minWidth="-Infinity" prefHeight="250.0" prefWidth="600.0"
xmlns="http://javafx.com/javafx/18" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="ru.etu.controllers.Settings">
    <center>
        <VBox prefHeight="200.0" prefWidth="100.0"
BorderPane.alignment="CENTER">
            <BorderPane.margin>
                <Insets bottom="10.0" left="10.0" right="10.0" top="10.0"
/>
            </BorderPane.margin>
            <children>
                <HBox prefHeight="30.0" prefWidth="200.0" spacing="30.0">
                    <children>
                        <Label prefWidth="150.0" text="Speed of playing:"
/>
                        <TextField fx:id="playSpeedTextField" />
                        <Label text="(Milliseconds)" />
                    </children>
                    <VBox.margin>
```

```
                        <Insets bottom="10.0" />
                    </VBox.margin>
                </HBox>
                <HBox prefHeight="30.0" prefWidth="351.0" spacing="30.0">
                    <children>
                        <Label prefWidth="150.0" text="Zoom step:" />
                        <TextField fx:id="zoomStepTextField"
editable="false" prefHeight="26.0" prefWidth="149.0" />
                        <Button fx:id="decreaseBtn"
mnemonicParsing="false" text="-0.05" />
                        <Button fx:id="increaseBtn"
mnemonicParsing="false" text="+0.05" />
                    </children>
                    <VBox.margin>
                        <Insets bottom="10.0" />
                    </VBox.margin>
                </HBox>
                <HBox prefHeight="27.0" prefWidth="579.0">
                    <children>
                        <Label prefHeight="18.0" prefWidth="180.0"
text="Field size:" />
                        <RadioButton fx:id="smallBtn"
mnemonicParsing="false" prefHeight="18.0" prefWidth="107.0" text="Small">
                            <toggleGroup>
                                <ToggleGroup fx:id="fieldSizeToggleGroup"
/>
                            </toggleGroup>
                        </RadioButton>
                        <RadioButton fx:id="normalBtn"
mnemonicParsing="false" prefHeight="18.0" prefWidth="113.0" text="Normal"
toggleGroup="$fieldSizeToggleGroup" />
                        <RadioButton fx:id="hugeBtn"
mnemonicParsing="false" text="Huge" toggleGroup="$fieldSizeToggleGroup" /
>
                    </children>
                </HBox>
                <HBox prefHeight="30.0" prefWidth="200.0" spacing="30.0">
                    <children>
                        <Label prefWidth="150.0" text="Minimal zoom:" />
                        <TextField fx:id="minZoomTextField" />
                        <Label text="(Has at least one zoom step)" />
                    </children>
                    <VBox.margin>
                        <Insets bottom="10.0" />
                    </VBox.margin>
                </HBox>
                <HBox prefHeight="30.0" prefWidth="200.0" spacing="30.0">
                    <children>
                        <Label prefWidth="150.0" text="Maximal zoom:" />
                        <TextField fx:id="maxZoomTextField" />
```

```
                        <Label text="(Has to be less than 10)" />
                    </children>
                    <VBox.margin>
                        <Insets bottom="10.0" />
                    </VBox.margin>
                </HBox>
            </children>
        </VBox>
    </center>
    <bottom>
        <HBox alignment="BOTTOM_RIGHT" prefHeight="30.0"
prefWidth="200.0" spacing="10.0" BorderPane.alignment="CENTER">
            <children>
                <Button fx:id="okBtn" mnemonicParsing="false" text="OK" /
>
                <Button fx:id="cancelBtn" mnemonicParsing="false"
text="Cancel" />
                <Button fx:id="applyBtn" mnemonicParsing="false"
prefHeight="26.0" prefWidth="61.0" text="Apply" />
            </children>
            <BorderPane.margin>
                <Insets bottom="10.0" right="10.0" />
            </BorderPane.margin>
        </HBox>
    </bottom>
</BorderPane>
```

about.css

```
.text-area {
    -fx-background-color: transparent;
}

.text-area .scroll-pane {
    -fx-background-color: transparent;
}

.text-area .scroll-pane .viewport{
    -fx-background-color: transparent;
}

.text-area .scroll-pane .content{
    -fx-background-color: transparent;
}
```

app.css

```
.top-panel, .top-btn {
    -fx-background-color: #86c460;
```

```
}

.top-btn:hover {
    -fx-background-color: #76ab55;
}

.top-btn:armed {
    -fx-background-color: #669349;
}

.top-btn:selected {
    -fx-background-color: #669349;
}

.left-panel, .left-btn {
    -fx-background-color: #9c71c4;
}

.left-btn:hover {
    -fx-background-color: #8460a6;
}

.left-btn:armed {
    -fx-background-color: #73538f;
}

.left-btn:selected {
    -fx-background-color: #73538f;

}

.graph-pane-pane {
    -fx-background-color: rgba(239, 232, 226, 0.77);
}
```

graphStyles.css

```
.graph-pane {
    -fx-background-color: white;
}

.vertex {
    -fx-fill: #B0C4DE;
}

.vertex-selected {
    -fx-stroke-width: 2;
    -fx-stroke: #7fdc3c;
}
```

```
.vertex-selected-for-algorithm-1 {
    -fx-stroke-width: 2;
    -fx-stroke: #ff0073;
}

.vertex-selected-for-algorithm-2 {
    -fx-stroke-width: 2;
    -fx-stroke: #ad0052;
}

.vertex-check {
    -fx-stroke-width: 2;
    -fx-stroke: #e584ac;
}

.vertex-update {
    -fx-fill: #96dabc;
}

.vertex-path {
    -fx-stroke-width: 2;
    -fx-stroke: #e85b97;
}

.vertex-label {
    -fx-font: bold 10pt Sans-serif;
}


.edge {
    -fx-stroke: #a64f2c;
    -fx-stroke-width: 1;
}

.curved-edge {
    -fx-fill: none;
    -fx-stroke: #a64f2c;
    -fx-stroke-width: 1;
}

.edge-check {
    -fx-stroke-width: 2;
    -fx-stroke: #ea7b57;
}

.edge-path {
    -fx-stroke: #ff5000;
    -fx-stroke-width: 2;
}
```

```
.edge-in-creation {
    -fx-stroke: #c26034;
    -fx-stroke-width: 2;
    -fx-stroke-dash-array: 3;
}

.edge-label {
    -fx-font: normal 11pt Sans-serif;
}

.arrow {
    -fx-stroke: #a64f2c;
    -fx-stroke-width: 1;
}

.arrow-path {
    -fx-stroke: #ff5000;
    -fx-stroke-width: 2;
}

.arrow-check {
    -fx-stroke-width: 2;
    -fx-stroke: #ea7b57;
}


.checking-label {
    -fx-font: italic 11pt Sans-serif;
}

.updating {
    -fx-font-weight: bold;
    -fx-font: 11pt Sans-serif;
}
```

DijkstraDirectedGraphTests.java

```
package algoritm;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import ru.etu.algoritm.Dijkstra;
import ru.etu.graph.DirectedGraphList;
import ru.etu.graph.Graph;
import ru.etu.graph.Vertex;

import java.util.ArrayList;
import java.util.List;

public class DijkstraDirectedGraphTests {
```

```
Graph graph;

@BeforeEach
void before(){
    graph = new DirectedGraphList();
}

@Test
void algGeneralTest() {
    Vertex v1 = graph.insertVertex("A");
    Vertex v2 = graph.insertVertex("B");
    Vertex v3 = graph.insertVertex("C");
    Vertex v4 = graph.insertVertex("D");
    Vertex v5 = graph.insertVertex("E");

    graph.insertEdge("A", "B", 1);
    graph.insertEdge("B", "C", 1);
    graph.insertEdge("C", "D", 1);
    graph.insertEdge("D", "E", 1);

    Dijkstra alg = new Dijkstra(graph);

    Assertions.assertTrue(alg.findPath(v1, v5));

    List<String> expected = new ArrayList<>();
    expected.add("A");
    expected.add("B");
    expected.add("C");
    expected.add("D");
    expected.add("E");

    Assertions.assertEquals(expected, alg.getPath());
}

@Test
void algGeneralTest2() {
    graph.insertVertex("A");
    graph.insertVertex("B");
    graph.insertVertex("C");
    graph.insertVertex("D");
    var finish = graph.insertVertex("E");
    var start = graph.insertVertex("F");

    graph.insertEdge("A", "B", 1);
    graph.insertEdge("A", "C", 2);
    graph.insertEdge("D", "A", 3);
    graph.insertEdge("B", "C", 4);
    graph.insertEdge("D", "C", 4);
    graph.insertEdge("B", "E", 5);
```

```
        graph.insertEdge("F", "D", 6);

        Dijkstra alg = new Dijkstra(graph);

        Assertions.assertTrue(alg.findPath(start, finish));

        List<String> expected = new ArrayList<>();
        expected.add("F");
        expected.add("D");
        expected.add("A");
        expected.add("B");
        expected.add("E");

        Assertions.assertEquals(expected, alg.getPath());
    }

    @Test
    void equalPathsTest() {
        var start = graph.insertVertex("A");
        graph.insertVertex("B");
        graph.insertVertex("C");
        var finish = graph.insertVertex("D");

        graph.insertEdge("A", "B", 4);
        graph.insertEdge("B", "D", 3);
        graph.insertEdge("A", "C", 3);
        graph.insertEdge("C", "D", 4);

        Dijkstra alg = new Dijkstra(graph);

        Assertions.assertTrue(alg.findPath(start, finish));

        List<String> expected = new ArrayList<>();
        expected.add("A");
        expected.add("C");
        expected.add("D");

        Assertions.assertEquals(expected, alg.getPath());
    }
}
```

DijkstraTests.java

```
package algoritm;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import ru.etu.algoritm.Dijkstra;
import ru.etu.graph.Graph;
import ru.etu.graph.GraphList;
```

```
import ru.etu.graph.Vertex;

import java.util.ArrayList;
import java.util.List;


public class DijkstraTests {

    Graph graph;

    @BeforeEach
    void before(){
        graph = new GraphList();
    }

    @Test
    void algGeneralTest() {
        Vertex v1 = graph.insertVertex("A");
        Vertex v2 = graph.insertVertex("B");
        Vertex v3 = graph.insertVertex("C");
        Vertex v4 = graph.insertVertex("D");
        Vertex v5 = graph.insertVertex("E");

        graph.insertEdge("B", "A", 1);
        graph.insertEdge("C", "B", 1);
        graph.insertEdge("D", "C", 1);
        graph.insertEdge("E", "D", 1);

        Dijkstra alg = new Dijkstra(graph);

        Assertions.assertTrue(alg.findPath(v1, v5));

        List<String> expected = new ArrayList<>();
        expected.add("A");
        expected.add("B");
        expected.add("C");
        expected.add("D");
        expected.add("E");

        Assertions.assertEquals(expected, alg.getPath());
    }

    @Test
    void algGeneralTest2() {
        Vertex v1 = graph.insertVertex("A");
        Vertex v21 = graph.insertVertex("B1");
        Vertex v22 = graph.insertVertex("B2");
        Vertex v31 = graph.insertVertex("C1");
        Vertex v32 = graph.insertVertex("C2");
        Vertex v41 = graph.insertVertex("D1");
```

```
    Vertex v42 = graph.insertVertex("D2");
    Vertex v5 = graph.insertVertex("E");

    graph.insertEdge("B1", "A", 1);
    graph.insertEdge("B2", "A", 1);
    graph.insertEdge("C1", "B1", 1);
    graph.insertEdge("C2", "B2", 1);
    graph.insertEdge("D1", "C1", 1);
    graph.insertEdge("D2", "C2", 1);
    graph.insertEdge("E", "D1", 1);
    graph.insertEdge("E", "D2", 1);

    Dijkstra alg = new Dijkstra(graph);

    Assertions.assertTrue(alg.findPath(v1, v5));

    List<String> expected = new ArrayList<>();
    expected.add("A");
    expected.add("B1");
    expected.add("C1");
    expected.add("D1");
    expected.add("E");

    Assertions.assertEquals(expected, alg.getPath());
    //System.out.println(alg.getPath().toString());
}

@Test
void algGeneralTest3() {
    Vertex v1 = graph.insertVertex("A");
    Vertex v21 = graph.insertVertex("B1");
    Vertex v22 = graph.insertVertex("B2");
    Vertex v31 = graph.insertVertex("C1");
    Vertex v32 = graph.insertVertex("C2");
    Vertex v41 = graph.insertVertex("D1");
    Vertex v42 = graph.insertVertex("D2");
    Vertex v5 = graph.insertVertex("E");

    graph.insertEdge("B1", "A", 2);
    graph.insertEdge("B2", "A", 1);
    graph.insertEdge("C1", "B1", 1);
    graph.insertEdge("C2", "B2", 1);
    graph.insertEdge("D1", "C1", 1);
    graph.insertEdge("D2", "C2", 1);
    graph.insertEdge("E", "D1", 1);
    graph.insertEdge("E", "D2", 1);

    Dijkstra alg = new Dijkstra(graph);

    Assertions.assertTrue(alg.findPath(v1, v5));
```

```
    List<String> expected = new ArrayList<>();
    expected.add("A");
    expected.add("B2");
    expected.add("C2");
    expected.add("D2");
    expected.add("E");

    Assertions.assertEquals(expected, alg.getPath());
}

@Test
void algGeneralTest4() {
    Vertex v1 = graph.insertVertex("A");
    Vertex v21 = graph.insertVertex("B1");
    Vertex v22 = graph.insertVertex("B2");
    Vertex v31 = graph.insertVertex("C1");
    Vertex v32 = graph.insertVertex("C2");
    Vertex v41 = graph.insertVertex("D1");
    Vertex v42 = graph.insertVertex("D2");
    Vertex v5 = graph.insertVertex("E");

    graph.insertEdge("B1", "A", 1);
    graph.insertEdge("B2", "A", 1);
    graph.insertEdge("C1", "B1", 1);
    graph.insertEdge("C2", "B2", 1);
    graph.insertEdge("D1", "C1", 1);
    graph.insertEdge("D2", "C2", 1);
    graph.insertEdge("E", "D1", 1);
    graph.insertEdge("E", "D2", 2);

    Dijkstra alg = new Dijkstra(graph);

    Assertions.assertTrue(alg.findPath(v1, v5));

    List<String> expected = new ArrayList<>();
    expected.add("A");
    expected.add("B1");
    expected.add("C1");
    expected.add("D1");
    expected.add("E");

    Assertions.assertEquals(expected, alg.getPath());
}

@Test
void algTestNotGreedy() {
    Vertex v1 = graph.insertVertex("A");
    Vertex v21 = graph.insertVertex("B1");
    Vertex v22 = graph.insertVertex("B2");
```

```
    Vertex v31 = graph.insertVertex("C1");
    Vertex v32 = graph.insertVertex("C2");
    Vertex v41 = graph.insertVertex("D1");
    Vertex v42 = graph.insertVertex("D2");
    Vertex v5 = graph.insertVertex("E");

    graph.insertEdge("B1", "A", 10);
    graph.insertEdge("B2", "A", 1);
    graph.insertEdge("C1", "B1", 10);
    graph.insertEdge("C2", "B2", 1);
    graph.insertEdge("D1", "C1", 10);
    graph.insertEdge("D2", "C2", 1);
    graph.insertEdge("E", "D1", 10);
    graph.insertEdge("E", "D2", 50);

    Dijkstra alg = new Dijkstra(graph);

    Assertions.assertTrue(alg.findPath(v1, v5));

    List<String> expected = new ArrayList<>();
    expected.add("A");
    expected.add("B1");
    expected.add("C1");
    expected.add("D1");
    expected.add("E");

    Assertions.assertEquals(expected, alg.getPath());
}

@Test
void algTestNotFound() {
    Vertex v1 = graph.insertVertex("A");
    Vertex v21 = graph.insertVertex("B1");
    Vertex v22 = graph.insertVertex("B2");
    Vertex v31 = graph.insertVertex("C1");
    Vertex v32 = graph.insertVertex("C2");
    Vertex v41 = graph.insertVertex("D1");
    Vertex v42 = graph.insertVertex("D2");
    Vertex v51 = graph.insertVertex("E1");
    Vertex v52 = graph.insertVertex("E2");

    graph.insertEdge("B1", "A", 10);
    graph.insertEdge("B2", "A", 1);
    graph.insertEdge("C1", "B1", 10);
    graph.insertEdge("C2", "B2", 1);
    graph.insertEdge("D1", "C1", 10);
    graph.insertEdge("D2", "C2", 1);
    graph.insertEdge("E2", "D1", 10);
    graph.insertEdge("E2", "D2", 50);
```

```
        Dijkstra alg = new Dijkstra(graph);

        Assertions.assertFalse(alg.findPath(v1, v51));
    }
}
```

DirectedGraphTest.java

```
package graph;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import ru.etu.graph.*;

import java.util.ArrayList;
import java.util.List;

public class DirectedGraphTest {
    DirectedGraphList graph;

    @BeforeEach
    void before() {
        graph = new DirectedGraphList();
    }


    @Test
    void getVerticesNumberTest() {
        graph.insertVertex("A");
        graph.insertVertex("B");
        graph.insertVertex("C");

        Assertions.assertEquals(3, graph.verticesNum());
    }

    @Test
    void getVerticesNumberAfterOneDeletedTest() {
        graph.insertVertex("A");
        graph.insertVertex("B");
        var vertexC = graph.insertVertex("C");

        graph.removeVertex(vertexC);

        Assertions.assertEquals(2, graph.verticesNum());
    }

    @Test
    void getVerticesNumberAfterAllDeletedTest() {
        var vertexA = graph.insertVertex("A");
        var vertexB = graph.insertVertex("B");
```

```
    var vertexC = graph.insertVertex("C");

    graph.removeVertex(vertexA);
    graph.removeVertex(vertexB);
    graph.removeVertex(vertexC);

    Assertions.assertEquals(0, graph.verticesNum());
}


@Test
void getEdgesNumberTest() {
    var vertexA = graph.insertVertex("A");
    var vertexB = graph.insertVertex("B");
    var vertexC = graph.insertVertex("C");
    var vertexD = graph.insertVertex("D");

    graph.insertEdge(vertexA, vertexB, 5);
    graph.insertEdge(vertexA, vertexC, 4);
    graph.insertEdge(vertexA, vertexD, 5);

    Assertions.assertEquals(3, graph.edgesNum());
}

@Test
void getEdgesNumberAfterOneDeletedTest() {
    var vertexA = graph.insertVertex("A");
    var vertexB = graph.insertVertex("B");
    var vertexC = graph.insertVertex("C");
    var vertexD = graph.insertVertex("D");

    graph.insertEdge(vertexA, vertexB, 5);
    graph.insertEdge(vertexA, vertexC, 4);
    var edge = graph.insertEdge(vertexA, vertexD, 5);

    graph.removeEdge(edge);

    Assertions.assertEquals(2, graph.edgesNum());
}

@Test
void getEdgesNumberAfterAllDeletedTest() {
    var vertexA = graph.insertVertex("A");
    var vertexB = graph.insertVertex("B");
    var vertexC = graph.insertVertex("C");
    var vertexD = graph.insertVertex("D");

    var edge1 = graph.insertEdge(vertexA, vertexB, 5);
    var edge2 = graph.insertEdge(vertexA, vertexC, 4);
    var edge3 = graph.insertEdge(vertexA, vertexD, 5);
```

```
    graph.removeEdge(edge1);
    graph.removeEdge(edge2);
    graph.removeEdge(edge3);

    Assertions.assertEquals(0, graph.edgesNum());
}


@Test
void getVerticesTest() {
    var vertexA = graph.insertVertex("A");
    var vertexB = graph.insertVertex("B");
    var vertexC = graph.insertVertex("C");

    Assertions.assertEquals(3, graph.getVertices().size());

    ArrayList<Vertex> expected = new ArrayList<>() {{
        add(vertexA);
        add(vertexB);
        add(vertexC);
    }};

    List<Vertex> real = graph.getVertices();

    for (int i = 0; i < expected.size(); i++) {
        Assertions.assertEquals(expected.get(i), real.get(i));
    }
}

@Test
void getVerticesOneDeletedTest() {
    var vertexA = graph.insertVertex("A");
    var vertexB = graph.insertVertex("B");
    var vertexC = graph.insertVertex("C");

    graph.removeVertex(vertexC);

    Assertions.assertEquals(2, graph.getVertices().size());

    ArrayList<Vertex> expected = new ArrayList<>() {{
        add(vertexA);
        add(vertexB);
    }};

    List<Vertex> real = graph.getVertices();

    for (int i = 0; i < expected.size(); i++) {
        Assertions.assertEquals(expected.get(i), real.get(i));
    }
```

```
}

@Test
void getVerticesAllDeletedTest() {
    var vertexA = graph.insertVertex("A");
    var vertexB = graph.insertVertex("B");
    var vertexC = graph.insertVertex("C");

    graph.removeVertex(vertexA);
    graph.removeVertex(vertexB);
    graph.removeVertex(vertexC);

    Assertions.assertEquals(0, graph.getVertices().size());
}


@Test
void getEdgesTest() {
    var vertexA = graph.insertVertex("A");
    var vertexB = graph.insertVertex("B");
    var vertexC = graph.insertVertex("C");
    var vertexD = graph.insertVertex("D");

    var edge1 = graph.insertEdge(vertexA, vertexB, 5);
    var edge2 = graph.insertEdge(vertexA, vertexC, 4);
    var edge3 = graph.insertEdge(vertexA, vertexD, 5);

    Assertions.assertEquals(3, graph.getEdges().size());

    ArrayList<Edge> expected = new ArrayList<>() {{
        add(edge1);
        add(edge2);
        add(edge3);
    }};

    List<Edge> real = graph.getEdges();

    for (int i = 0; i < expected.size(); i++) {
        Assertions.assertEquals(expected.get(i), real.get(i));
    }
}

@Test
void getEdgesOneDeletedTest() {
    var vertexA = graph.insertVertex("A");
    var vertexB = graph.insertVertex("B");
    var vertexC = graph.insertVertex("C");
    var vertexD = graph.insertVertex("D");

    var edge1 = graph.insertEdge(vertexA, vertexB, 5);
```

```
    var edge2 = graph.insertEdge(vertexA, vertexC, 4);
    var edge3 = graph.insertEdge(vertexA, vertexD, 5);

    graph.removeEdge(edge3);

    Assertions.assertEquals(2, graph.getEdges().size());

    ArrayList<Edge> expected = new ArrayList<>() {{
        add(edge1);
        add(edge2);
    }};

    List<Edge> real = graph.getEdges();

    for (int i = 0; i < expected.size(); i++) {
        Assertions.assertEquals(expected.get(i), real.get(i));
    }
}

@Test
void getEdgesAllDeletedTest() {
    var vertexA = graph.insertVertex("A");
    var vertexB = graph.insertVertex("B");
    var vertexC = graph.insertVertex("C");
    var vertexD = graph.insertVertex("D");

    var edge1 = graph.insertEdge(vertexA, vertexB, 5);
    var edge2 = graph.insertEdge(vertexA, vertexC, 4);
    var edge3 = graph.insertEdge(vertexA, vertexD, 5);

    graph.removeEdge(edge1);
    graph.removeEdge(edge2);
    graph.removeEdge(edge3);

    Assertions.assertEquals(0, graph.getEdges().size());
}


@Test
void checkingIncidentEdges() {
    var vertexA = graph.insertVertex("A");
    var vertexB = graph.insertVertex("B");
    var vertexC = graph.insertVertex("C");
    var vertexD = graph.insertVertex("D");

    graph.insertEdge(vertexA, vertexB, 2);
    graph.insertEdge(vertexA, vertexC, 2);
    graph.insertEdge(vertexD, vertexA, 2);
    Assertions.assertEquals(2, graph.incidentEdges(vertexA).size());
}
```

```
    @Test
    void checkingIncidentEdgesWithVertexFromGraphTest() {
        Assertions.assertDoesNotThrow(() -> {
            var vertexA = graph.insertVertex("A");
            var vertexB = graph.insertVertex("B");

            graph.insertEdge(vertexA, vertexB, 2);
            graph.incidentEdges(vertexA);
        });
    }

    @Test
    void checkingIncidentEdgesWithVertexNotFromGraphTest() {
        var thrown =
Assertions.assertThrows(InvalidVertexException.class, () -> {
            var vertexA = graph.insertVertex("A");
            var vertexB = new Vertex("B");
            var vertexC = graph.insertVertex("C");

            graph.insertEdge(vertexA, vertexC, 2);
            graph.incidentEdges(vertexB);
        });
    }


    @Test
    void checkingInboundEdgesEdges() {
        var vertexA = graph.insertVertex("A");
        var vertexB = graph.insertVertex("B");
        var vertexD = graph.insertVertex("D");

        graph.insertEdge(vertexA, vertexB, 2);
        graph.insertEdge(vertexA, vertexD, 2);
        graph.insertEdge(vertexD, vertexB, 2);
        Assertions.assertEquals(2, graph.inboundEdges(vertexB).size());
    }

    @Test
    void checkingInboundEdgesWithVertexFromGraphTest() {
        Assertions.assertDoesNotThrow(() -> {
            var vertexA = graph.insertVertex("A");
            var vertexB = graph.insertVertex("B");

            graph.insertEdge(vertexA, vertexB, 2);
            graph.inboundEdges(vertexA);
        });
    }

    @Test
```

```
    void checkingInboundEdgesWithVertexNotFromGraphTest() {
        var thrown =
Assertions.assertThrows(InvalidVertexException.class, () -> {
            var vertexA = graph.insertVertex("A");
            var vertexB = new Vertex("B");
            var vertexC = graph.insertVertex("C");

            graph.insertEdge(vertexA, vertexC, 2);
            graph.inboundEdges(vertexB);
        });
    }


    @Test
    void checkingOppositeVertexWithVertexAndEdgeBothInGraphTest() {
        Assertions.assertDoesNotThrow(() -> {
            var vertexA = graph.insertVertex("A");
            var vertexB = graph.insertVertex("B");

            var edgeAB = graph.insertEdge(vertexA, vertexB, 2);
            var result = graph.opposite(vertexA, edgeAB);
            Assertions.assertEquals(vertexB, result);
            result = graph.opposite(vertexB, edgeAB);
            Assertions.assertEquals(vertexA, result);
        });


    }

    @Test
    void checkingOppositeVertexWithVertexAndEdgeAnyNotInGraphTest() {
        Assertions.assertThrows(InvalidVertexException.class, () -> {
            var vertexA = graph.insertVertex("A");
            var vertexB = new Vertex("B");
            var vertexC = graph.insertVertex("C");

            var edgeAC = graph.insertEdge(vertexA, vertexC, 2);
            graph.opposite(vertexB, edgeAC);
        });

        graph = new DirectedGraphList();

        Assertions.assertThrows(InvalidEdgeException.class, () -> {
            var vertexA = graph.insertVertex("A");
            var vertexB = graph.insertVertex("B");

            var edgeAB = new Edge(2, vertexA, vertexB);
            graph.opposite(vertexB, edgeAB);
        });
```

```
        graph = new DirectedGraphList();

        Assertions.assertThrows(InvalidVertexException.class, () -> {
            var vertexA = graph.insertVertex("A");
            var vertexB = new Vertex("B");

            var edgeAB = new Edge(2, vertexA, vertexB);
            graph.opposite(vertexB, edgeAB);
        });
    }

    @Test
    void checkingOppositeVertexWhileEdgeDoesNotHaveVertexTest() {
        var vertexA = graph.insertVertex("A");
        var vertexB = graph.insertVertex("B");
        var vertexC = graph.insertVertex("C");

        var edgeAC = graph.insertEdge(vertexA, vertexC, 2);
        var result = graph.opposite(vertexB, edgeAC);

        Assertions.assertNull(result);
    }


    @Test
    void areConnectedBothExistsConnectedTest() {
        var vertexA = graph.insertVertex("A");
        var vertexB = graph.insertVertex("B");

        var edgeAB = graph.insertEdge(vertexA, vertexB, 2);

        var result = graph.areConnected(vertexB, vertexA);
        Assertions.assertFalse(result);

        result = graph.areConnected(vertexA, vertexB);
        Assertions.assertTrue(result);
    }

    @Test
    void areConnectedSameVertexTest() {
        var vertexA = graph.insertVertex("A");

        var result = graph.areConnected(vertexA, vertexA);
        Assertions.assertFalse(result);
    }

    @Test
    void areConnectedBothExistsNotConnectedTest() {
        var vertexA = graph.insertVertex("A");
        var vertexB = graph.insertVertex("B");
```

```
        var result = graph.areConnected(vertexB, vertexA);
        Assertions.assertFalse(result);

        result = graph.areConnected(vertexA, vertexB);
        Assertions.assertFalse(result);
    }

    @Test
    void areConnectedAnyDoesNotExistTest() {
        var vertexA = graph.insertVertex("A");
        var vertexB = new Vertex("B");

        var edgeAB = new Edge(2, vertexA, vertexB);

        Assertions.assertThrows(InvalidVertexException.class, () -> {
            graph.areConnected(vertexB, vertexA);
        });

        graph = new DirectedGraphList();

        var vertexA1 = new Vertex("A");
        var vertexB1 = graph.insertVertex("B");

        var edgeAB1 = new Edge(2, vertexA, vertexB);

        Assertions.assertThrows(InvalidVertexException.class, () -> {
            graph.areConnected(vertexB1, vertexA1);
        });
    }


    @Test
    void newVerticesCreationSameVerticesNameTest() {
        var thrown =
Assertions.assertThrows(InvalidVertexException.class, () -> {
            var vertexA1 = graph.insertVertex("A");
            var vertexA2 = graph.insertVertex("A");
        });
    }

    @Test
    void newVertexCreationTest() {
        Assertions.assertDoesNotThrow(() -> {
            var vertexB = graph.insertVertex("B");
        });
    }

    @Test
    void newVertexNullNameCreationTest() {
```

```
        var thrown =
Assertions.assertThrows(IllegalArgumentException.class, () -> {
            var vertexNull = graph.insertVertex(null);
        });
    }


    @Test
    void creatingEdgeWithVertexNotFromGraphTest() {
        var thrown =
Assertions.assertThrows(InvalidVertexException.class, () -> {
            var vertexA = graph.insertVertex("A");
            var vertexB = new Vertex("B");

            graph.insertEdge(vertexA, vertexB, 2);
        });
    }

    @Test
    void loopEdgeCreationTest() {
        Assertions.assertThrows(InvalidVertexException.class, () -> {
            var vertexA = graph.insertVertex("A");

            graph.insertEdge(vertexA, vertexA, 2);
        });
    }

    @Test
    void newEdgeCreationSameEdgesLengthWithDifferentPairsTest() {
        Assertions.assertDoesNotThrow(() -> {
            var vertexA = graph.insertVertex("A");
            var vertexB = graph.insertVertex("B");
            var vertexC = graph.insertVertex("C");

            graph.insertEdge(vertexA, vertexB, 2);
            graph.insertEdge(vertexA, vertexC, 2);
        });
    }

    @Test
    void newEdgeCreationSameEdgesLengthWithSamePairsTest() {
        var thrown = Assertions.assertThrows(InvalidEdgeException.class,
() -> {
            var vertexA = graph.insertVertex("A");
            var vertexB = graph.insertVertex("B");

            graph.insertEdge(vertexA, vertexB, 2);
            graph.insertEdge(vertexA, vertexB, 2);
        });
    }
```

```
    @Test
    void newEdgeCreationSameEdgesLengthWithSamePairsDifferentWeightTest()
{
        var thrown = Assertions.assertThrows(InvalidEdgeException.class,
() -> {
            var vertexA = graph.insertVertex("A");
            var vertexB = graph.insertVertex("B");

            graph.insertEdge(vertexA, vertexB, 2);
            graph.insertEdge(vertexA, vertexB, 4);
        });
    }

    @Test
    void newEdgeCreationWithoutRealVerticesTest() {
        var thrown =
Assertions.assertThrows(InvalidVertexException.class, () -> {
            graph.insertEdge("A", "B", 2);
        });
    }

    @Test
    void oppositeDirectionEdgesWithSameWeightTest() {
        Assertions.assertDoesNotThrow(() -> {
            var vertexA = graph.insertVertex("A");
            var vertexB = graph.insertVertex("B");

            graph.insertEdge(vertexA, vertexB, 2);
            graph.insertEdge(vertexB, vertexA, 2);
        });
    }

    @Test
    void oppositeDirectionEdgesWithDifferentWeightTest() {
        Assertions.assertDoesNotThrow(() -> {
            var vertexA = graph.insertVertex("A");
            var vertexB = graph.insertVertex("B");

            graph.insertEdge(vertexA, vertexB, 2);
            graph.insertEdge(vertexB, vertexA, 12);
        });
    }


    @Test
    void removeVertexWithoutRealVertexTest() {
        Assertions.assertDoesNotThrow(() -> {
            graph.insertVertex("A");
```

```
            graph.removeVertex(new Vertex("A"));
        });
    }

    @Test
    void removeVertexWithRealVertexTest() {
        Assertions.assertDoesNotThrow(() -> {
            var vertex = graph.insertVertex("A");

            graph.removeVertex(vertex);
        });
    }

    @Test
    void removeVertexWithoutAnyVerticesTest() {
        Assertions.assertThrows(InvalidVertexException.class, () -> {
            graph.removeVertex(new Vertex("A"));
        });
    }

    @Test
    void removeEdgeWithoutRealEdgeTest() {
        Assertions.assertDoesNotThrow(() -> {
            var vertexA = graph.insertVertex("A");
            var vertexB = graph.insertVertex("B");

            graph.insertEdge(vertexA, vertexB, 2);

            graph.removeEdge(new Edge(2, new Vertex("A"), vertexB));
        });
    }
    @Test
    void removeEdgeWithRealEdgeTest() {
        Assertions.assertDoesNotThrow(() -> {
            var vertexA = graph.insertVertex("A");
            var vertexB = graph.insertVertex("B");

            var edge = graph.insertEdge(vertexA, vertexB, 2);

            graph.removeEdge(edge);
        });
    }

    @Test
    void removeEdgeWithoutAnyEdgesTest() {
        Assertions.assertThrows(InvalidEdgeException.class, () -> {

            graph.removeEdge(new Edge(2, new Vertex("A"), new
Vertex("B")));
        });
```

```
}


@Test
void getEdgeVerticesNormalTest() {
    var vertexA = graph.insertVertex("A");
    var vertexB = graph.insertVertex("B");
    var vertexC = graph.insertVertex("C");

    var origEdge = graph.insertEdge(vertexA, vertexB, 2);
    graph.insertEdge(vertexA, vertexC, 2);

    Assertions.assertDoesNotThrow(() -> {
        graph.getEdge(vertexA, vertexB);
    });

    var gotEdge = graph.getEdge(vertexA, vertexB);

    Assertions.assertEquals(origEdge, gotEdge);
}

@Test
void getEdgeVerticesVertexNotInGraphTest() {
    var vertexA = graph.insertVertex("A");
    var vertexB = graph.insertVertex("B");
    var vertexC = graph.insertVertex("C");

    graph.insertEdge(vertexA, vertexB, 2);
    graph.insertEdge(vertexA, vertexC, 2);

    Assertions.assertThrows(InvalidVertexException.class, () -> {
        graph.getEdge(vertexA, new Vertex("D"));
    });
}

@Test
void getEdgeVerticesEdgeNotExistsTest() {
    var vertexA = graph.insertVertex("A");
    var vertexB = graph.insertVertex("B");

    var gotEdge = graph.getEdge(vertexA, vertexB);

    Assertions.assertNull(gotEdge);
}


@Test
void getEdgeStringsNormalTest() {
    var vertexA = graph.insertVertex("A");
    var vertexB = graph.insertVertex("B");
```

```
    var vertexC = graph.insertVertex("C");

    var origEdge = graph.insertEdge(vertexA, vertexB, 2);
    graph.insertEdge(vertexA, vertexC, 2);

    Assertions.assertDoesNotThrow(() -> {
        graph.getEdge("A", "B");
    });

    var gotEdge = graph.getEdge("A", "B");

    Assertions.assertEquals(origEdge, gotEdge);
}

@Test
void getEdgeStringsVertexNotInGraphTest() {
    var vertexA = graph.insertVertex("A");
    var vertexB = graph.insertVertex("B");
    var vertexC = graph.insertVertex("C");

    graph.insertEdge(vertexA, vertexB, 2);
    graph.insertEdge(vertexA, vertexC, 2);

    Assertions.assertThrows(InvalidVertexException.class, () -> {
        graph.getEdge("A", "D");
    });
}

@Test
void getEdgeStringsEdgeNotExistsTest() {
    var vertexA = graph.insertVertex("A");
    var vertexB = graph.insertVertex("B");

    var gotEdge = graph.getEdge("A", "B");

    Assertions.assertNull(gotEdge);
}


@Test
void GetVertexNormalText() {
    var vertexA = graph.insertVertex("A");
    var vertexB = graph.insertVertex("B");

    Assertions.assertDoesNotThrow(() -> {
        graph.getVertex("A");
    });
}

@Test
```

```
    void GetVertexNotExistsText() {
        var vertexA = graph.insertVertex("A");
        var vertexB = graph.insertVertex("B");

        Assertions.assertThrows(InvalidVertexException.class, () -> {
            graph.getVertex("C");
        });
    }

    @Test
    void GetVertexNullText() {
        var vertexA = graph.insertVertex("A");
        var vertexB = graph.insertVertex("B");

        Assertions.assertThrows(IllegalArgumentException.class, () -> {
            graph.getVertex(null);
        });
    }


}
```

GraphTest.java

```
package graph;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import ru.etu.graph.*;

import java.util.ArrayList;
import java.util.List;

public class GraphTest {
    Graph graph;

    @BeforeEach
    void before() {
        graph = new GraphList();
    }

    @Test
    void getVerticesNumberTest() {
        graph.insertVertex("A");
        graph.insertVertex("B");
        graph.insertVertex("C");

        Assertions.assertEquals(3, graph.verticesNum());
    }
```

```
@Test
void getVerticesNumberAfterOneDeletedTest() {
    graph.insertVertex("A");
    graph.insertVertex("B");
    var vertexC = graph.insertVertex("C");

    graph.removeVertex(vertexC);

    Assertions.assertEquals(2, graph.verticesNum());
}

@Test
void getVerticesNumberAfterAllDeletedTest() {
    var vertexA = graph.insertVertex("A");
    var vertexB = graph.insertVertex("B");
    var vertexC = graph.insertVertex("C");

    graph.removeVertex(vertexA);
    graph.removeVertex(vertexB);
    graph.removeVertex(vertexC);

    Assertions.assertEquals(0, graph.verticesNum());
}


@Test
void getEdgesNumberTest() {
    var vertexA = graph.insertVertex("A");
    var vertexB = graph.insertVertex("B");
    var vertexC = graph.insertVertex("C");
    var vertexD = graph.insertVertex("D");

    graph.insertEdge(vertexA, vertexB, 5);
    graph.insertEdge(vertexA, vertexC, 4);
    graph.insertEdge(vertexA, vertexD, 5);

    Assertions.assertEquals(3, graph.edgesNum());
}

@Test
void getEdgesNumberAfterOneDeletedTest() {
    var vertexA = graph.insertVertex("A");
    var vertexB = graph.insertVertex("B");
    var vertexC = graph.insertVertex("C");
    var vertexD = graph.insertVertex("D");

    graph.insertEdge(vertexA, vertexB, 5);
    graph.insertEdge(vertexA, vertexC, 4);
    var edge = graph.insertEdge(vertexA, vertexD, 5);
```

```
        graph.removeEdge(edge);

        Assertions.assertEquals(2, graph.edgesNum());
    }

    @Test
    void getEdgesNumberAfterAllDeletedTest() {
        var vertexA = graph.insertVertex("A");
        var vertexB = graph.insertVertex("B");
        var vertexC = graph.insertVertex("C");
        var vertexD = graph.insertVertex("D");

        var edge1 = graph.insertEdge(vertexA, vertexB, 5);
        var edge2 = graph.insertEdge(vertexA, vertexC, 4);
        var edge3 = graph.insertEdge(vertexA, vertexD, 5);

        graph.removeEdge(edge1);
        graph.removeEdge(edge2);
        graph.removeEdge(edge3);

        Assertions.assertEquals(0, graph.edgesNum());
    }


    @Test
    void getVerticesTest() {
        var vertexA = graph.insertVertex("A");
        var vertexB = graph.insertVertex("B");
        var vertexC = graph.insertVertex("C");

        Assertions.assertEquals(3, graph.getVertices().size());

        ArrayList<Vertex> expected = new ArrayList<>() {{
            add(vertexA);
            add(vertexB);
            add(vertexC);
        }};

        List<Vertex> real = graph.getVertices();

        for (int i = 0; i < expected.size(); i++) {
            Assertions.assertEquals(expected.get(i), real.get(i));
        }
    }

    @Test
    void getVerticesOneDeletedTest() {
        var vertexA = graph.insertVertex("A");
        var vertexB = graph.insertVertex("B");
        var vertexC = graph.insertVertex("C");
```

```
    graph.removeVertex(vertexC);

    Assertions.assertEquals(2, graph.getVertices().size());

    ArrayList<Vertex> expected = new ArrayList<>() {{
        add(vertexA);
        add(vertexB);
    }};

    List<Vertex> real = graph.getVertices();

    for (int i = 0; i < expected.size(); i++) {
        Assertions.assertEquals(expected.get(i), real.get(i));
    }
}

@Test
void getVerticesAllDeletedTest() {
    var vertexA = graph.insertVertex("A");
    var vertexB = graph.insertVertex("B");
    var vertexC = graph.insertVertex("C");

    graph.removeVertex(vertexA);
    graph.removeVertex(vertexB);
    graph.removeVertex(vertexC);

    Assertions.assertEquals(0, graph.getVertices().size());
}


@Test
void getEdgesTest() {
    var vertexA = graph.insertVertex("A");
    var vertexB = graph.insertVertex("B");
    var vertexC = graph.insertVertex("C");
    var vertexD = graph.insertVertex("D");

    var edge1 = graph.insertEdge(vertexA, vertexB, 5);
    var edge2 = graph.insertEdge(vertexA, vertexC, 4);
    var edge3 = graph.insertEdge(vertexA, vertexD, 5);

    Assertions.assertEquals(3, graph.getEdges().size());

    ArrayList<Edge> expected = new ArrayList<>() {{
        add(edge1);
        add(edge2);
        add(edge3);
    }};
```

```
        List<Edge> real = graph.getEdges();

        for (int i = 0; i < expected.size(); i++) {
            Assertions.assertEquals(expected.get(i), real.get(i));
        }
    }

    @Test
    void getEdgesOneDeletedTest() {
        var vertexA = graph.insertVertex("A");
        var vertexB = graph.insertVertex("B");
        var vertexC = graph.insertVertex("C");
        var vertexD = graph.insertVertex("D");

        var edge1 = graph.insertEdge(vertexA, vertexB, 5);
        var edge2 = graph.insertEdge(vertexA, vertexC, 4);
        var edge3 = graph.insertEdge(vertexA, vertexD, 5);

        graph.removeEdge(edge3);

        Assertions.assertEquals(2, graph.getEdges().size());

        ArrayList<Edge> expected = new ArrayList<>() {{
            add(edge1);
            add(edge2);
        }};

        List<Edge> real = graph.getEdges();

        for (int i = 0; i < expected.size(); i++) {
            Assertions.assertEquals(expected.get(i), real.get(i));
        }
    }

    @Test
    void getEdgesAllDeletedTest() {
        var vertexA = graph.insertVertex("A");
        var vertexB = graph.insertVertex("B");
        var vertexC = graph.insertVertex("C");
        var vertexD = graph.insertVertex("D");

        var edge1 = graph.insertEdge(vertexA, vertexB, 5);
        var edge2 = graph.insertEdge(vertexA, vertexC, 4);
        var edge3 = graph.insertEdge(vertexA, vertexD, 5);

        graph.removeEdge(edge1);
        graph.removeEdge(edge2);
        graph.removeEdge(edge3);

        Assertions.assertEquals(0, graph.getEdges().size());
```

```
    }


    @Test
    void checkingIncidentEdges() {
        var vertexA = graph.insertVertex("A");
        var vertexB = graph.insertVertex("B");
        var vertexC = graph.insertVertex("C");
        var vertexD = graph.insertVertex("D");

        graph.insertEdge(vertexA, vertexB, 2);
        graph.insertEdge(vertexA, vertexC, 2);
        graph.insertEdge(vertexD, vertexA, 2);
        Assertions.assertEquals(3, graph.incidentEdges(vertexA).size());
    }

    @Test
    void checkingIncidentEdgesWithVertexFromGraphTest() {
        Assertions.assertDoesNotThrow(() -> {
            var vertexA = graph.insertVertex("A");
            var vertexB = graph.insertVertex("B");

            graph.insertEdge(vertexA, vertexB, 2);
            graph.incidentEdges(vertexA);
        });
    }

    @Test
    void checkingIncidentEdgesWithVertexNotFromGraphTest() {
        var thrown =
Assertions.assertThrows(InvalidVertexException.class, () -> {
            var vertexA = graph.insertVertex("A");
            var vertexB = new Vertex("B");
            var vertexC = graph.insertVertex("C");

            graph.insertEdge(vertexA, vertexC, 2);
            graph.incidentEdges(vertexB);
        });
    }


    @Test
    void checkingOppositeVertexWithVertexAndEdgeBothInGraphTest() {
        Assertions.assertDoesNotThrow(() -> {
            var vertexA = graph.insertVertex("A");
            var vertexB = graph.insertVertex("B");

            var edgeAB = graph.insertEdge(vertexA, vertexB, 2);
            var result = graph.opposite(vertexA, edgeAB);
            Assertions.assertEquals(vertexB, result);
```

```
        result = graph.opposite(vertexB, edgeAB);
        Assertions.assertEquals(vertexA, result);
    });


}

@Test
void checkingOppositeVertexWithVertexAndEdgeAnyNotInGraphTest() {
    Assertions.assertThrows(InvalidVertexException.class, () -> {
        var vertexA = graph.insertVertex("A");
        var vertexB = new Vertex("B");
        var vertexC = graph.insertVertex("C");

        var edgeAC = graph.insertEdge(vertexA, vertexC, 2);
        graph.opposite(vertexB, edgeAC);
    });

    graph = new GraphList();

    Assertions.assertThrows(InvalidEdgeException.class, () -> {
        var vertexA = graph.insertVertex("A");
        var vertexB = graph.insertVertex("B");

        var edgeAB = new Edge(2, vertexA, vertexB);
        graph.opposite(vertexB, edgeAB);
    });

    graph = new GraphList();

    Assertions.assertThrows(InvalidVertexException.class, () -> {
        var vertexA = graph.insertVertex("A");
        var vertexB = new Vertex("B");

        var edgeAB = new Edge(2, vertexA, vertexB);
        graph.opposite(vertexB, edgeAB);
    });
}

@Test
void checkingOppositeVertexWhileEdgeDoesNotHaveVertexTest() {
    var vertexA = graph.insertVertex("A");
    var vertexB = graph.insertVertex("B");
    var vertexC = graph.insertVertex("C");

    var edgeAC = graph.insertEdge(vertexA, vertexC, 2);
    var result = graph.opposite(vertexB, edgeAC);

    Assertions.assertNull(result);
}
```

```
@Test
void areConnectedBothExistsConnectedTest() {
    var vertexA = graph.insertVertex("A");
    var vertexB = graph.insertVertex("B");

    var edgeAB = graph.insertEdge(vertexA, vertexB, 2);

    var result = graph.areConnected(vertexB, vertexA);
    Assertions.assertTrue(result);

    result = graph.areConnected(vertexA, vertexB);
    Assertions.assertTrue(result);
}

@Test
void areConnectedBothExistsNotConnectedTest() {
    var vertexA = graph.insertVertex("A");
    var vertexB = graph.insertVertex("B");

    var result = graph.areConnected(vertexB, vertexA);
    Assertions.assertFalse(result);

    result = graph.areConnected(vertexA, vertexB);
    Assertions.assertFalse(result);
}

@Test
void areConnectedAnyDoesNotExistTest() {
    var vertexA = graph.insertVertex("A");
    var vertexB = new Vertex("B");

    var edgeAB = new Edge(2, vertexA, vertexB);

    Assertions.assertThrows(InvalidVertexException.class, () -> {
        graph.areConnected(vertexB, vertexA);
    });

    graph = new GraphList();

    var vertexA1 = new Vertex("A");
    var vertexB1 = graph.insertVertex("B");

    var edgeAB1 = new Edge(2, vertexA, vertexB);

    Assertions.assertThrows(InvalidVertexException.class, () -> {
        graph.areConnected(vertexB1, vertexA1);
    });
}
```

```
    @Test
    void newVerticesCreationSameVerticesNameTest() {
        var thrown =
Assertions.assertThrows(InvalidVertexException.class, () -> {
            var vertexA1 = graph.insertVertex("A");
            var vertexA2 = graph.insertVertex("A");
        });
    }

    @Test
    void newVertexCreationTest() {
        Assertions.assertDoesNotThrow(() -> {
            var vertexB = graph.insertVertex("B");
        });
    }

    @Test
    void newVertexNullNameCreationTest() {
        var thrown =
Assertions.assertThrows(IllegalArgumentException.class, () -> {
            var vertexNull = graph.insertVertex(null);
        });
    }


    @Test
    void loopEdgeCreationTest() {
        Assertions.assertThrows(InvalidVertexException.class, () -> {
            var vertexA = graph.insertVertex("A");

            graph.insertEdge(vertexA, vertexA, 2);
        });
    }

    @Test
    void newEdgeCreationSameEdgesLengthWithDifferentPairsTest() {
        Assertions.assertDoesNotThrow(() -> {
            var vertexA = graph.insertVertex("A");
            var vertexB = graph.insertVertex("B");
            var vertexC = graph.insertVertex("C");

            graph.insertEdge(vertexA, vertexB, 2);
            graph.insertEdge(vertexA, vertexC, 2);
        });
    }

    @Test
    void newEdgeCreationSameEdgesLengthWithSamePairsTest() {
```

```
        var thrown = Assertions.assertThrows(InvalidEdgeException.class,
() -> {
            var vertexA = graph.insertVertex("A");
            var vertexB = graph.insertVertex("B");

            graph.insertEdge(vertexA, vertexB, 2);
            graph.insertEdge(vertexA, vertexB, 2);
        });
    }


    @Test
    void newEdgeCreationWithoutRealVerticesTest() {
        var thrown =
Assertions.assertThrows(InvalidVertexException.class, () -> {
            graph.insertEdge("A", "B", 2);
        });
    }

    @Test
    void newEdgeCreationByVerticesNamesTest() {
        Assertions.assertDoesNotThrow(() -> {
            graph.insertVertex("A");
            graph.insertVertex("B");
            graph.insertEdge("A", "B", 2);
        });
    }

    @Test
    void newEdgeCreationByNullsTest() {
        Assertions.assertThrows(IllegalArgumentException.class, () -> {
            graph.insertEdge((String) null, null, 2);
        });
    }

    @Test
    void oppositeDirectionEdgesCreationWithSameWeightTest() {
        var thrown = Assertions.assertThrows(InvalidEdgeException.class,
() -> {
            var vertexA = graph.insertVertex("A");
            var vertexB = graph.insertVertex("B");

            graph.insertEdge(vertexA, vertexB, 2);
            graph.insertEdge(vertexB, vertexA, 2);
        });
    }

    @Test
    void oppositeDirectionEdgesCreationWithDifferentWeightTest() {
```

```
        var thrown = Assertions.assertThrows(InvalidEdgeException.class,
() -> {
            var vertexA = graph.insertVertex("A");
            var vertexB = graph.insertVertex("B");

            graph.insertEdge(vertexA, vertexB, 2);
            graph.insertEdge(vertexB, vertexA, 12);
        });
    }


    @Test
    void removeVertexWithoutRealVertexTest() {
        Assertions.assertDoesNotThrow(() -> {
            graph.insertVertex("A");

            graph.removeVertex(new Vertex("A"));
        });
    }

    @Test
    void removeVertexWithRealVertexTest() {
        Assertions.assertDoesNotThrow(() -> {
            var vertex = graph.insertVertex("A");

            graph.removeVertex(vertex);
        });
    }

    @Test
    void removeVertexWithoutAnyVerticesTest() {
        Assertions.assertThrows(InvalidVertexException.class, () -> {
            graph.removeVertex(new Vertex("A"));
        });
    }


    @Test
    void removeEdgeWithoutRealEdgeTest() {
        Assertions.assertDoesNotThrow(() -> {
            var vertexA = graph.insertVertex("A");
            var vertexB = graph.insertVertex("B");

            graph.insertEdge(vertexA, vertexB, 2);

            graph.removeEdge(new Edge(2, new Vertex("A"), vertexB));
        });
    }
    @Test
    void removeEdgeWithRealEdgeTest() {
```

```
        Assertions.assertDoesNotThrow(() -> {
            var vertexA = graph.insertVertex("A");
            var vertexB = graph.insertVertex("B");

            var edge = graph.insertEdge(vertexA, vertexB, 2);

            graph.removeEdge(edge);
        });
    }

    @Test
    void removeEdgeWithoutAnyEdgesTest() {
        Assertions.assertThrows(InvalidEdgeException.class, () -> {

            graph.removeEdge(new Edge(2, new Vertex("A"), new
Vertex("B")));
        });
    }


    @Test
    void getEdgeVerticesNormalTest() {
        var vertexA = graph.insertVertex("A");
        var vertexB = graph.insertVertex("B");
        var vertexC = graph.insertVertex("C");

        var origEdge = graph.insertEdge(vertexA, vertexB, 2);
        graph.insertEdge(vertexA, vertexC, 2);

        Assertions.assertDoesNotThrow(() -> {
            graph.getEdge(vertexA, vertexB);
        });

        var gotEdge = graph.getEdge(vertexA, vertexB);

        Assertions.assertEquals(origEdge, gotEdge);
    }

    @Test
    void getEdgeVerticesVertexNotInGraphTest() {
        var vertexA = graph.insertVertex("A");
        var vertexB = graph.insertVertex("B");
        var vertexC = graph.insertVertex("C");

        graph.insertEdge(vertexA, vertexB, 2);
        graph.insertEdge(vertexA, vertexC, 2);

        Assertions.assertThrows(InvalidVertexException.class, () -> {
            graph.getEdge(vertexA, new Vertex("D"));
        });
```

```
}

@Test
void getEdgeVerticesEdgeNotExistsTest() {
    var vertexA = graph.insertVertex("A");
    var vertexB = graph.insertVertex("B");

    var gotEdge = graph.getEdge(vertexA, vertexB);

    Assertions.assertNull(gotEdge);
}


@Test
void getEdgeStringsNormalTest() {
    var vertexA = graph.insertVertex("A");
    var vertexB = graph.insertVertex("B");
    var vertexC = graph.insertVertex("C");

    var origEdge = graph.insertEdge(vertexA, vertexB, 2);
    graph.insertEdge(vertexA, vertexC, 2);

    Assertions.assertDoesNotThrow(() -> {
        graph.getEdge("A", "B");
    });

    var gotEdge = graph.getEdge("A", "B");

    Assertions.assertEquals(origEdge, gotEdge);
}

@Test
void getEdgeStringsVertexNotInGraphTest() {
    var vertexA = graph.insertVertex("A");
    var vertexB = graph.insertVertex("B");
    var vertexC = graph.insertVertex("C");

    graph.insertEdge(vertexA, vertexB, 2);
    graph.insertEdge(vertexA, vertexC, 2);

    Assertions.assertThrows(InvalidVertexException.class, () -> {
        graph.getEdge("A", "D");
    });
}

@Test
void getEdgeStringsEdgeNotExistsTest() {
    var vertexA = graph.insertVertex("A");
    var vertexB = graph.insertVertex("B");
```

```
        var gotEdge = graph.getEdge("A", "B");

        Assertions.assertNull(gotEdge);
    }


    @Test
    void getVertexNormalText() {
        var vertexA = graph.insertVertex("A");
        var vertexB = graph.insertVertex("B");

        Assertions.assertDoesNotThrow(() -> {
            graph.getVertex("A");
        });
    }

    @Test
    void getVertexNotExistsText() {
        var vertexA = graph.insertVertex("A");
        var vertexB = graph.insertVertex("B");

        Assertions.assertThrows(InvalidVertexException.class, () -> {
            graph.getVertex("C");
        });
    }

    @Test
    void getVertexNullText() {
        var vertexA = graph.insertVertex("A");
        var vertexB = graph.insertVertex("B");

        Assertions.assertThrows(IllegalArgumentException.class, () -> {
            graph.getVertex(null);
        });
    }




}
```

IOTest.java

```
package io;

import org.junit.jupiter.api.*;
import ru.etu.graph.DirectedGraphList;
import ru.etu.graph.Graph;
import ru.etu.graph.GraphList;
import ru.etu.io.IOGraph;
```

```
import java.io.File;
import java.io.IOException;

@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
public class IOTest {

    private IOGraph saver;
    private Graph simple;
    private Graph simpleDirected;
    private Graph complex;
    private Graph complexDirected;
    private Graph simpleTest;
    private Graph simpleDirectedTest;
    private Graph complexTest;
    private Graph complexDirectedTest;

    @BeforeEach
    void before(){
        simple = new GraphList();
        simple.insertVertex("A");
        simple.insertVertex("B");
        simple.insertVertex("C");
        simple.insertVertex("D");
        simple.insertVertex("E");
        simple.insertEdge("B", "A", 1);
        simple.insertEdge("C", "B", 1);
        simple.insertEdge("D", "C", 1);
        simple.insertEdge("E", "D", 1);
        saver = new IOGraph();
        /*try {
            saver.saveGraph("test_simple.txt", simple);
            saver.saveGraph("test_simple_another_type.wtf", simple);
            saver.saveGraph("new_dir"+ File.separator
+"test_simple_another_dir.txt", simple);
            saver.saveGraph("test_simple_кириллица.txt", simple);
            saver.saveGraph(".."+ File.separator
+"test_simple_another_dir2.wtf", simple);
            saver.saveGraph("C:"+ File.separator +"delete_me"+
File.separator +"test_simple_abs_dir.txt", simple);
        } catch (IOException ex){
            System.out.println("Cannot create file
"+ex.getLocalizedMessage());
        }*/
        simpleDirected = new DirectedGraphList();
        simpleDirected.insertVertex("A");
        simpleDirected.insertVertex("B");
        simpleDirected.insertVertex("C");
        simpleDirected.insertVertex("D");
        simpleDirected.insertVertex("E");
        simpleDirected.insertEdge("B", "A", 1);
```

```
        simpleDirected.insertEdge("C", "B", 1);
        simpleDirected.insertEdge("D", "C", 1);
        simpleDirected.insertEdge("E", "D", 1);
        /*try {
            saver.saveGraph("test_simpledirected.txt", simpleDirected);
            saver.saveGraph("test_simpledirected_another_type.wtf",
simpleDirected);
            saver.saveGraph("new_dir"+ File.separator
+"test_simpledirected_another_dir.txt", simpleDirected);
            saver.saveGraph("test_simpledirected_кириллица.txt",
simpleDirected);
            saver.saveGraph(".."+ File.separator
+"test_simpledirected_another_dir2.wtf", simpleDirected);
            saver.saveGraph("C:"+ File.separator +"delete_me"+
File.separator +"test_simpledirected_abs_dir.txt", simpleDirected);
        } catch (IOException ex){
            System.out.println("Cannot create file
"+ex.getLocalizedMessage());
        }*/
        complex = new GraphList();
        complex.insertVertex("A");
        complex.insertVertex("B1");
        complex.insertVertex("B2");
        complex.insertVertex("C1");
        complex.insertVertex("C2");
        complex.insertVertex("D1");
        complex.insertVertex("D2");
        complex.insertVertex("E");
        complex.insertEdge("B1", "A", 10);
        complex.insertEdge("B2", "A", 1);
        complex.insertEdge("C1", "B1", 10);
        complex.insertEdge("C2", "B2", 1);
        complex.insertEdge("D1", "C1", 10);
        complex.insertEdge("D2", "C2", 1);
        complex.insertEdge("E", "D1", 10);
        complex.insertEdge("E", "D2", 50);
        /*try {
            saver.saveGraph("test_complex.txt", complex);
            saver.saveGraph("test_complex_another_type.wtf", complex);
            saver.saveGraph("new_dir"+ File.separator
+"test_complex_another_dir.txt", complex);
            saver.saveGraph("test_complex_кириллица.txt", complex);
            saver.saveGraph(".."+ File.separator
+"test_complex_another_dir2.wtf", complex);
            saver.saveGraph("C:"+ File.separator +"delete_me"+
File.separator +"test_complex_abs_dir.txt", complex);
        } catch (IOException ex){
            System.out.println("Cannot create file
"+ex.getLocalizedMessage());
        }*/
```

```
        complexDirected = new DirectedGraphList();
        complexDirected.insertVertex("A");
        complexDirected.insertVertex("B1");
        complexDirected.insertVertex("B2");
        complexDirected.insertVertex("C1");
        complexDirected.insertVertex("C2");
        complexDirected.insertVertex("D1");
        complexDirected.insertVertex("D2");
        complexDirected.insertVertex("E");
        complexDirected.insertEdge("B1", "A", 10);
        complexDirected.insertEdge("B2", "A", 1);
        complexDirected.insertEdge("C1", "B1", 10);
        complexDirected.insertEdge("C2", "B2", 1);
        complexDirected.insertEdge("D1", "C1", 10);
        complexDirected.insertEdge("D2", "C2", 1);
        complexDirected.insertEdge("E", "D1", 10);
        complexDirected.insertEdge("E", "D2", 50);
        /*try {
            saver.saveGraph("test_complexdirected.txt", complexDirected);
            saver.saveGraph("test_complexdirected_another_type.wtf",
complexDirected);
            saver.saveGraph("new_dir"+ File.separator
+"test_complexdirected_another_dir.txt", complexDirected);
            saver.saveGraph("test_complexdirected_кириллица.txt",
complexDirected);
            saver.saveGraph(".."+ File.separator
+"test_complexdirected_another_dir2.wtf", complexDirected);
            saver.saveGraph("C:"+ File.separator +"delete_me"+
File.separator +"test_complexdirected_abs_dir.txt", complexDirected);
        } catch (IOException ex){
            System.out.println("Cannot create file
"+ex.getLocalizedMessage());
        }*/
    }

    @Test
    @Order(1)
    void testSave_Simple(){
        Assertions.assertDoesNotThrow(() ->
{ saver.saveGraph("test_Simple.txt", simple);});
    }

    @Test
    @Order(2)
    void testSave_Simple_another_type(){
        Assertions.assertDoesNotThrow(() ->
{ saver.saveGraph("test_Simple_another_type.wtf", simple);});
    }

    @Test
```

```
    @Order(3)
    void testSave_Simple_another_dir(){
        Assertions.assertDoesNotThrow(() -> { saver.saveGraph("new_dir"+
File.separator +"test_Simple_another_dir.txt", simple);});
    }

    @Test
    @Order(4)
    void testSave_Simple_another_lang(){
        Assertions.assertDoesNotThrow(() ->
{ saver.saveGraph("test_Simple_кириллица.txt", simple);});
    }

    @Test
    @Order(5)
    void testSave_Simple_another_dir2(){
        Assertions.assertDoesNotThrow(() -> { saver.saveGraph(".."+
File.separator +"test_Simple_another_dir2.wtf", simple);});
    }

    @Test
    @Order(6)
    void testSave_Simple_abs_dir(){
        Assertions.assertDoesNotThrow(() -> { saver.saveGraph("C:"+
File.separator +"delete_me"+ File.separator +"test_Simple_abs_dir.txt",
simple);});
    }

    @Test
    @Order(7)
    void testSave_SimpleDirected(){
        Assertions.assertDoesNotThrow(() ->
{ saver.saveGraph("test_SimpleDirected.txt", simpleDirected);});
    }

    @Test
    @Order(8)
    void testSave_SimpleDirected_another_type(){
        Assertions.assertDoesNotThrow(() ->
{ saver.saveGraph("test_SimpleDirected_another_type.wtf",
simpleDirected);});
    }

    @Test
    @Order(9)
    void testSave_SimpleDirected_another_dir(){
        Assertions.assertDoesNotThrow(() -> { saver.saveGraph("new_dir"+
File.separator +"test_SimpleDirected_another_dir.txt",
simpleDirected);});
    }
```

```
    @Test
    @Order(10)
    void testSave_SimpleDirected_another_lang(){
        Assertions.assertDoesNotThrow(() ->
{ saver.saveGraph("test_SimpleDirected_кириллица.txt",
simpleDirected);});
    }

    @Test
    @Order(11)
    void testSave_SimpleDirected_another_dir2(){
        Assertions.assertDoesNotThrow(() -> { saver.saveGraph(".."+
File.separator +"test_Simple_anotherDirected_dir2.wtf",
simpleDirected);});
    }

    @Test
    @Order(12)
    void testSave_SimpleDirected_abs_dir(){
        Assertions.assertDoesNotThrow(() -> { saver.saveGraph("C:"+
File.separator +"delete_me"+ File.separator
+"test_SimpleDirected_abs_dir.txt", simpleDirected);});
    }

    @Test
    @Order(13)
    void testSave_Complex(){
        Assertions.assertDoesNotThrow(() ->
{ saver.saveGraph("test_Complex.txt", complex);});
    }

    @Test
    @Order(14)
    void testSave_Complex_another_type(){
        Assertions.assertDoesNotThrow(() ->
{ saver.saveGraph("test_Complex_another_type.wtf", complex);});
    }

    @Test
    @Order(15)
    void testSave_Complex_another_dir(){
        Assertions.assertDoesNotThrow(() -> { saver.saveGraph("new_dir"+
File.separator +"test_Complex_another_dir.txt", complex);});
    }

    @Test
    @Order(16)
    void testSave_Complex_another_lang(){
```

```
        Assertions.assertDoesNotThrow(() ->
{ saver.saveGraph("test_Complex_кириллица.txt", complex);});
    }

    @Test
    @Order(17)
    void testSave_Complex_another_dir2(){
        Assertions.assertDoesNotThrow(() -> { saver.saveGraph(".."+
File.separator +"test_Complex_another_dir2.wtf", complex);});
    }

    @Test
    @Order(18)
    void testSave_Complex_abs_dir(){
        Assertions.assertDoesNotThrow(() -> { saver.saveGraph("C:"+
File.separator +"delete_me"+ File.separator +"test_Complex_abs_dir.txt",
complex);});
    }

    @Test
    @Order(19)
    void testSave_ComplexDirected(){
        Assertions.assertDoesNotThrow(() ->
{ saver.saveGraph("test_ComplexDirected.txt", complexDirected);});
    }

    @Test
    @Order(20)
    void testSave_ComplexDirected_another_type(){
        Assertions.assertDoesNotThrow(() ->
{ saver.saveGraph("test_ComplexDirected_another_type.wtf",
complexDirected);});
    }

    @Test
    @Order(21)
    void testSave_ComplexDirected_another_dir(){
        Assertions.assertDoesNotThrow(() -> { saver.saveGraph("new_dir"+
File.separator +"test_ComplexDirected_another_dir.txt",
complexDirected);});
    }

    @Test
    @Order(22)
    void testSave_ComplexDirected_another_lang(){
        Assertions.assertDoesNotThrow(() ->
{ saver.saveGraph("test_ComplexDirected_кириллица.txt",
complexDirected);});
    }
```

```
    @Test
    @Order(23)
    void testSave_ComplexDirected_another_dir2(){
        Assertions.assertDoesNotThrow(() -> { saver.saveGraph(".."+
File.separator +"test_ComplexDirected_another_dir2.wtf",
complexDirected);});
    }

    @Test
    @Order(24)
    void testSave_ComplexDirected_abs_dir(){
        Assertions.assertDoesNotThrow(() -> { saver.saveGraph("C:"+
File.separator +"delete_me"+ File.separator
+"test_ComplexDirected_abs_dir.txt", complexDirected);});
    }

    @Test
    @Order(25)
    void testNotFound(){
        Assertions.assertThrows(IOException.class,() ->
{ saver.loadGraph("test_404.txt");});
    }

    @Test
    @Order(26)
    void testOpen_Simple(){
        Assertions.assertDoesNotThrow(() -> { simpleTest =
saver.loadGraph("test_Simple.txt");});
        Assertions.assertEquals(simpleTest.toString(),
simple.toString());
    }

    @Test
    @Order(27)
    void testOpen_Simple_another_type(){
        Assertions.assertDoesNotThrow(() -> { simpleTest =
saver.loadGraph("test_Simple_another_type.wtf");});
        Assertions.assertEquals(simpleTest.toString(),
simple.toString());
    }

    @Test
    @Order(28)
    void testOpen_Simple_another_dir(){
        Assertions.assertDoesNotThrow(() -> { simpleTest =
saver.loadGraph("new_dir"+ File.separator
+"test_Simple_another_dir.txt");});
        Assertions.assertEquals(simpleTest.toString(),
simple.toString());
    }
```

```
    @Test
    @Order(29)
    void testOpen_Simple_another_lang(){
        Assertions.assertDoesNotThrow(() -> { simpleTest =
saver.loadGraph("test_Simple_кириллица.txt");});
        Assertions.assertEquals(simpleTest.toString(),
simple.toString());
    }

    @Test
    @Order(30)
    void testOpen_Simple_another_dir2(){
        Assertions.assertDoesNotThrow(() -> { simpleTest =
saver.loadGraph(".."+ File.separator +"test_Simple_another_dir2.wtf");});
        Assertions.assertEquals(simpleTest.toString(),
simple.toString());
    }

    @Test
    @Order(31)
    void testOpen_Simple_abs_dir(){
        Assertions.assertDoesNotThrow(() -> { simpleTest =
saver.loadGraph("C:"+ File.separator +"delete_me"+ File.separator
+"test_Simple_abs_dir.txt");});
        Assertions.assertEquals(simpleTest.toString(),
simple.toString());
    }

    //

    @Test
    @Order(32)
    void testOpen_Complex(){
        Assertions.assertDoesNotThrow(() -> { complexTest =
saver.loadGraph("test_Complex.txt");});
        Assertions.assertEquals(complexTest.toString(),
complex.toString());
    }

    @Test
    @Order(33)
    void testOpen_Complex_another_type(){
        Assertions.assertDoesNotThrow(() -> { complexTest =
saver.loadGraph("test_Complex_another_type.wtf");});
        Assertions.assertEquals(complexTest.toString(),
complex.toString());
    }

    @Test
```

```
    @Order(34)
    void testOpen_Complex_another_dir(){
        Assertions.assertDoesNotThrow(() -> { complexTest =
saver.loadGraph("new_dir"+ File.separator
+"test_Complex_another_dir.txt");});
        Assertions.assertEquals(complexTest.toString(),
complex.toString());
    }

    @Test
    @Order(35)
    void testOpen_Complex_another_lang(){
        Assertions.assertDoesNotThrow(() -> { complexTest =
saver.loadGraph("test_Complex_кириллица.txt");});
        Assertions.assertEquals(complexTest.toString(),
complex.toString());
    }

    @Test
    @Order(36)
    void testOpen_Complex_another_dir2(){
        Assertions.assertDoesNotThrow(() -> { complexTest =
saver.loadGraph(".."+ File.separator
+"test_Complex_another_dir2.wtf");});
        Assertions.assertEquals(complexTest.toString(),
complex.toString());
    }

    @Test
    @Order(37)
    void testOpen_Complex_abs_dir(){
        Assertions.assertDoesNotThrow(() -> { complexTest =
saver.loadGraph("C:"+ File.separator +"delete_me"+ File.separator
+"test_Complex_abs_dir.txt");});
        Assertions.assertEquals(complexTest.toString(),
complex.toString());
    }

    //

    @Test
    @Order(38)
    void testOpen_SimpleDirected(){
        Assertions.assertDoesNotThrow(() -> { simpleDirectedTest =
saver.loadGraph("test_SimpleDirected.txt");});
        Assertions.assertEquals(simpleDirectedTest.toString(),
simpleDirected.toString());
    }

    @Test
```

```
    @Order(39)
    void testOpen_SimpleDirected_another_type(){
        Assertions.assertDoesNotThrow(() -> { simpleDirectedTest =
saver.loadGraph("test_SimpleDirected_another_type.wtf");});
        Assertions.assertEquals(simpleDirectedTest.toString(),
simpleDirected.toString());
    }

    @Test
    @Order(40)
    void testOpen_SimpleDirected_another_dir(){
        Assertions.assertDoesNotThrow(() -> { simpleDirectedTest =
saver.loadGraph("new_dir"+ File.separator
+"test_SimpleDirected_another_dir.txt");});
        Assertions.assertEquals(simpleDirectedTest.toString(),
simpleDirected.toString());
    }

    @Test
    @Order(41)
    void testOpen_SimpleDirected_another_lang(){
        Assertions.assertDoesNotThrow(() -> { simpleDirectedTest =
saver.loadGraph("test_SimpleDirected_кириллица.txt");});
        Assertions.assertEquals(simpleDirectedTest.toString(),
simpleDirected.toString());
    }

    @Test
    @Order(42)
    void testOpen_SimpleDirected_another_dir2(){
        Assertions.assertDoesNotThrow(() -> { simpleDirectedTest =
saver.loadGraph(".."+ File.separator
+"test_Simple_anotherDirected_dir2.wtf");});
        Assertions.assertEquals(simpleDirectedTest.toString(),
simpleDirected.toString());
    }

    @Test
    @Order(43)
    void testOpen_SimpleDirected_abs_dir(){
        Assertions.assertDoesNotThrow(() -> { simpleDirectedTest =
saver.loadGraph("C:"+ File.separator +"delete_me"+ File.separator
+"test_SimpleDirected_abs_dir.txt");});
        Assertions.assertEquals(simpleDirectedTest.toString(),
simpleDirected.toString());
    }

    //

    @Test
```

```
    @Order(44)
    void testOpen_ComplexDirected(){
        Assertions.assertDoesNotThrow(() -> { complexDirectedTest =
saver.loadGraph("test_ComplexDirected.txt");});
        Assertions.assertEquals(complexDirectedTest.toString(),
complexDirected.toString());
    }

    @Test
    @Order(45)
    void testOpen_ComplexDirected_another_type(){
        Assertions.assertDoesNotThrow(() -> { complexDirectedTest =
saver.loadGraph("test_ComplexDirected_another_type.wtf");});
        Assertions.assertEquals(complexDirectedTest.toString(),
complexDirected.toString());
    }

    @Test
    @Order(46)
    void testOpen_ComplexDirected_another_dir(){
        Assertions.assertDoesNotThrow(() -> { complexDirectedTest =
saver.loadGraph("new_dir"+ File.separator
+"test_ComplexDirected_another_dir.txt");});
        Assertions.assertEquals(complexDirectedTest.toString(),
complexDirected.toString());
    }

    @Test
    @Order(47)
    void testOpen_ComplexDirected_another_lang(){
        Assertions.assertDoesNotThrow(() -> { complexDirectedTest =
saver.loadGraph("test_ComplexDirected_кириллица.txt");});
        Assertions.assertEquals(complexDirectedTest.toString(),
complexDirected.toString());
    }

    @Test
    @Order(48)
    void testOpen_ComplexDirected_another_dir2(){
        Assertions.assertDoesNotThrow(() -> { complexDirectedTest =
saver.loadGraph(".."+ File.separator
+"test_ComplexDirected_another_dir2.wtf");});
        Assertions.assertEquals(complexDirectedTest.toString(),
complexDirected.toString());
    }

    @Test
    @Order(49)
    void testOpen_ComplexDirected_abs_dir(){
```

```
        Assertions.assertDoesNotThrow(() -> { complexDirectedTest =
saver.loadGraph("C:"+ File.separator +"delete_me"+ File.separator
+"test_ComplexDirected_abs_dir.txt");});
        Assertions.assertEquals(complexDirectedTest.toString(),
complexDirected.toString());
    }

    @AfterAll
    static void testClearGarbage(){
        try {
            if(!new File("test_Simple.txt").delete()){
                throw new Exception();
            }
        } catch (Exception ex){
            System.out.println("Cannot delete "+"test_Simple.txt"+"!
Please, delete it manually.");
        }
        try {
            if(!new File("test_Simple_another_type.wtf").delete()){
                throw new Exception();
            }
        } catch (Exception ex){
            System.out.println("Cannot delete
"+"test_Simple_another_type.wtf"+"! Please, delete it manually.");
        }
        try {
            if(!new File("new_dir"+ File.separator
+"test_Simple_another_dir.txt").delete()){
                throw new Exception();
            }
        } catch (Exception ex){
            System.out.println("Cannot delete "+"new_dir"+ File.separator
+"test_Simple_another_dir.txt"+"! Please, delete it manually.");
        }
        try {
            if(!new File("test_Simple_кириллица.txt").delete()){
                throw new Exception();
            }
        } catch (Exception ex){
            System.out.println("Cannot delete
"+"test_Simple_кириллица.txt"+"! Please, delete it manually.");
        }
        try {
            if(!new File(".."+ File.separator
+"test_Simple_another_dir2.wtf").delete()){
                throw new Exception();
            }
        } catch (Exception ex){
            System.out.println("Cannot delete "+".."+ File.separator
+"test_Simple_another_dir2.wtf"+"! Please, delete it manually.");
```

```
        }
        try {
            if(!new File("C:"+ File.separator +"delete_me"+
File.separator +"test_Simple_abs_dir.txt").delete()){
                throw new Exception();
            }
        } catch (Exception ex){
            System.out.println("Cannot delete "+"C:"+ File.separator
+"delete_me"+ File.separator +"test_Simple_abs_dir.txt"+"! Please, delete
it manually.");
        }
        try {
            if(!new File("test_SimpleDirected.txt").delete()){
                throw new Exception();
            }
        } catch (Exception ex){
            System.out.println("Cannot delete
"+"test_SimpleDirected.txt"+"! Please, delete it manually.");
        }
        try {
            if(!new
File("test_SimpleDirected_another_type.wtf").delete()){
                throw new Exception();
            }
        } catch (Exception ex){
            System.out.println("Cannot delete
"+"test_SimpleDirected_another_type.wtf"+"! Please, delete it
manually.");
        }
        try {
            if(!new File("new_dir"+ File.separator
+"test_SimpleDirected_another_dir.txt").delete()){
                throw new Exception();
            }
        } catch (Exception ex){
            System.out.println("Cannot delete "+"new_dir"+ File.separator
+"test_SimpleDirected_another_dir.txt"+"! Please, delete it manually.");
        }
        try {
            if(!new File("test_SimpleDirected_кириллица.txt").delete()){
                throw new Exception();
            }
        } catch (Exception ex){
            System.out.println("Cannot delete
"+"test_SimpleDirected_кириллица.txt"+"! Please, delete it manually.");
        }
        try {
            if(!new File(".."+ File.separator
+"test_Simple_anotherDirected_dir2.wtf").delete()){
                throw new Exception();
```

```
            }
        } catch (Exception ex){
            System.out.println("Cannot delete "+".."+ File.separator
+"test_Simple_anotherDirected_dir2.wtf"+"! Please, delete it manually.");
        }
        try {
            if(!new File("C:"+ File.separator +"delete_me"+
File.separator +"test_SimpleDirected_abs_dir.txt").delete()){
                throw new Exception();
            }
        } catch (Exception ex){
            System.out.println("Cannot delete "+"C:"+ File.separator
+"delete_me"+ File.separator +"test_SimpleDirected_abs_dir.txt"+"!
Please, delete it manually.");
        }
        try {
            if(!new File("test_Complex.txt").delete()){
                throw new Exception();
            }
        } catch (Exception ex){
            System.out.println("Cannot delete "+"test_Complex.txt"+"!
Please, delete it manually.");
        }
        try {
            if(!new File("test_Complex_another_type.wtf").delete()){
                throw new Exception();
            }
        } catch (Exception ex){
            System.out.println("Cannot delete
"+"test_Complex_another_type.wtf"+"! Please, delete it manually.");
        }
        try {
            if(!new File("new_dir"+ File.separator
+"test_Complex_another_dir.txt").delete()){
                throw new Exception();
            }
        } catch (Exception ex){
            System.out.println("Cannot delete "+"new_dir"+ File.separator
+"test_Complex_another_dir.txt"+"! Please, delete it manually.");
        }
        try {
            if(!new File("test_Complex_кириллица.txt").delete()){
                throw new Exception();
            }
        } catch (Exception ex){
            System.out.println("Cannot delete
"+"test_Complex_кириллица.txt"+"! Please, delete it manually.");
        }
        try {
```

```
            if(!new File(".."+ File.separator
+"test_Complex_another_dir2.wtf").delete()){
                throw new Exception();
            }
        } catch (Exception ex){
            System.out.println("Cannot delete "+".."+ File.separator
+"test_Complex_another_dir2.wtf"+"! Please, delete it manually.");
        }
        try {
            if(!new File("C:"+ File.separator +"delete_me"+
File.separator +"test_Complex_abs_dir.txt").delete()){
                throw new Exception();
            }
        } catch (Exception ex){
            System.out.println("Cannot delete "+"C:"+ File.separator
+"delete_me"+ File.separator +"test_Complex_abs_dir.txt"+"! Please,
delete it manually.");
        }
        try {
            if(!new File("test_ComplexDirected.txt").delete()){
                throw new Exception();
            }
        } catch (Exception ex){
            System.out.println("Cannot delete
"+"test_ComplexDirected.txt"+"! Please, delete it manually.");
        }
        try {
            if(!new
File("test_ComplexDirected_another_type.wtf").delete()){
                throw new Exception();
            }
        } catch (Exception ex){
            System.out.println("Cannot delete
"+"test_ComplexDirected_another_type.wtf"+"! Please, delete it
manually.");
        }
        try {
            if(!new File("new_dir"+ File.separator
+"test_ComplexDirected_another_dir.txt").delete()){
                throw new Exception();
            }
        } catch (Exception ex){
            System.out.println("Cannot delete "+"new_dir"+ File.separator
+"test_ComplexDirected_another_dir.txt"+"! Please, delete it manually.");
        }
        try {
            if(!new File("test_ComplexDirected_кириллица.txt").delete()){
                throw new Exception();
            }
        } catch (Exception ex){
```

```
            System.out.println("Cannot delete
"+"test_ComplexDirected_кириллица.txt"+"! Please, delete it manually.");
        }
        try {
            if(!new File(".."+ File.separator
+"test_ComplexDirected_another_dir2.wtf").delete()){
                throw new Exception();
            }
        } catch (Exception ex){
            System.out.println("Cannot delete "+".."+ File.separator
+"test_ComplexDirected_another_dir2.wtf"+"! Please, delete it
manually.");
        }
        try {
            if(!new File("C:"+ File.separator +"delete_me"+
File.separator +"test_ComplexDirected_abs_dir.txt").delete()){
                throw new Exception();
            }
        } catch (Exception ex){
            System.out.println("Cannot delete "+"C:"+ File.separator
+"delete_me"+ File.separator +"test_ComplexDirected_abs_dir.txt"+"!
Please, delete it manually.");
        }
        try{
            if(!new File("new_dir").delete()){
                throw new Exception();
            }
        } catch (Exception ex){
            System.out.println("Cannot delete dir "+"new_dir"+"! Please,
delete it manually.");
        }
        try{
            if(!new File("C:"+ File.separator +"delete_me").delete()){
                throw new Exception();
            }
        } catch (Exception ex){
            System.out.println("Cannot delete dir "+"C:"+ File.separator
+"delete_me"+"! Please, delete it manually.");
        }
    }
}
```

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
```

```
    <groupId>ru.etu</groupId>
    <artifactId>StudyPract</artifactId>
    <version>0.1</version>
    <name>StudyPract</name>

    <properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <junit.version>5.8.2</junit.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.openjfx</groupId>
            <artifactId>javafx-controls</artifactId>
            <version>18.0.1</version>
        </dependency>
        <dependency>
            <groupId>org.openjfx</groupId>
            <artifactId>javafx-fxml</artifactId>
            <version>18.0.1</version>
        </dependency>

        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-api</artifactId>
            <version>${junit.version}</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <version>${junit.version}</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>com.google.code.gson</groupId>
            <artifactId>gson</artifactId>
            <version>2.9.0</version>
        </dependency>
        <!-- media -->
        <dependency>
            <groupId>org.openjfx</groupId>
            <artifactId>javafx-media</artifactId>
            <version>18.0.1</version>
            <classifier>win</classifier>
        </dependency>

        <dependency>
            <groupId>org.openjfx</groupId>
```

```
        <artifactId>javafx-media</artifactId>
        <version>18.0.1</version>
        <classifier>linux</classifier>
    </dependency>

    <dependency>
        <groupId>org.openjfx</groupId>
        <artifactId>javafx-media</artifactId>
        <version>18.0.1</version>
        <classifier>mac</classifier>
    </dependency>


    <!-- graphics -->
    <dependency>
        <groupId>org.openjfx</groupId>
        <artifactId>javafx-graphics</artifactId>
        <version>18.0.1</version>
        <classifier>win</classifier>
    </dependency>

    <dependency>
        <groupId>org.openjfx</groupId>
        <artifactId>javafx-graphics</artifactId>
        <version>18.0.1</version>
        <classifier>linux</classifier>
    </dependency>

    <dependency>
        <groupId>org.openjfx</groupId>
        <artifactId>javafx-graphics</artifactId>
        <version>18.0.1</version>
        <classifier>mac</classifier>
    </dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.10.1</version>
            <configuration>
                <source>17</source>
                <target>17</target>
            </configuration>
        </plugin>
        <plugin>
            <groupId>org.openjfx</groupId>
            <artifactId>javafx-maven-plugin</artifactId>
```

```
                <version>0.0.8</version>
                <executions>
                    <execution>
                        <!-- Default configuration for running with: mvn
clean javafx:run -->
                        <id>default-cli</id>
                        <configuration>

<mainClass>ru.etu.studypract/ru.etu.studypract.MainApplication</mainClass
>
                            <launcher>app</launcher>
                            <jlinkZipName>app</jlinkZipName>
                            <jlinkImageName>app</jlinkImageName>
                            <noManPages>true</noManPages>
                            <stripDebug>true</stripDebug>
                            <noHeaderFiles>true</noHeaderFiles>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
            <plugin>
                <artifactId>maven-shade-plugin</artifactId>
                <version>3.2.1</version>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>shade</goal>
                        </goals>
                        <configuration>
                            <transformers>
                                <transformer
implementation="org.apache.maven.plugins.shade.resource.ManifestResourceT
ransformer">

<mainClass>ru.etu.studypract.SuperMain</mainClass>
                                </transformer>4
                            </transformers>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
</project>
```