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

ОТЧЕТ

по учебной практике

Тема: Визуализация алгоритмов на графах

Студентка гр. 8303 _____ Самойлова А.С.

Студент гр. 8381 _____ Сахаров М.С.

Студент гр. 8381 _____ Гоголев Е.Е.

Руководитель _____ Фирсов М.А.

Санкт-Петербург
2020

ЗАДАНИЕ

НА УЧЕБНУЮ ПРАКТИКУ

Студентка Самойлова А.С. группы 8303
Студент Сахаров М.С. группы 8381
Студент Гоголев Е.Е. группы 8381

Тема практики: визуализация алгоритмов на графах

Задание на практику:

Командная итеративная разработка визуализатора алгоритма(ов) на Java с графическим интерфейсом.

Алгоритм: алгоритм Косарайю.

Сроки прохождения практики: 29.06.2020 – 12.07.2020

Дата сдачи отчёта: 12.07.2020

Дата защиты отчёта: 12.07.2020

Студентка	_____	Самойлова А.С.
Студент	_____	Сахаров М.С.
Студент	_____	Гоголев Е.Е.
Руководитель	_____	Фирсов М.А.

АННОТАЦИЯ

В отчёте о практике собрана теоретическая и практическая информация о разработке графического приложения на языке Java. Данное приложение реализует построение графа с его визуальным представлением. Главная задача работы — реализовать визуализацию алгоритма Косарайю поиска компонент сильной связности графа. Отчёт состоит из: введения, четырёх глав, поделённых на параграфы, заключения, списка использованных источников и приложения с исходным кодом программы.

Во введении поставлена цель и задачи работы. В первой главе описаны требования к программе. Вторая глава содержит план разработки. В третьей главе описаны особенности реализации проекта. В четвертой главе представлены результаты тестирования программы.

В заключении приведены выводы, полученные в ходе работы.

SUMMARY

The practice report contains theoretical and practical information on the development of a graphical application in the Java language. This application implements the construction of a graph with its visual representation. The main task of the work is to implement the visualization of the Kosarayu algorithm for searching for components of a strongly connected graph. The introduction sets the goal and objectives of the work. The first chapter describes the program requirements. The second chapter contains a development plan. The third chapter describes the features of the project. The fourth chapter presents the results of testing the program.

In conclusion, the conclusions obtained in the course of the work are given.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель практики — построение готового приложения, реализующего работу с графическим представлением графа. Данное приложение должно визуализировать ход выполнения алгоритма Косарайю поиска компонент сильной связности. Компонентой сильной связности (strongly connected component) называется такое (максимальное по включению) подмножество вершин графа, что любые две вершины этого подмножества достижимы друг из друга. Важность данного алгоритма обусловлена широкой применимостью его результатов. Например, сильно связанные компоненты могут быть использованы для решения 2-выполнимости задач. Для выполнения заданной цели необходимо выполнить следующие задачи:

- определить архитектуру проекта
- реализовать представление графа, алгоритм
- создать графический интерфейс
- прописать логику визуализации пошагового выполнения алгоритма

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1 ТЕХНИЧЕСКОЕ ЗАДАНИЕ

1.1.1 Требования к вводу-выводу исходных данных

Пользователь должен иметь возможность:

- Сохранить граф в файл .graph
- Загрузить граф из файла .graph

Хранение графа выполняется в текстовом файле в кодировке UTF8.

Формат хранения графа .graph (построчно):

- Число (N), означающее количество узлов графа.
- N строк вида $"{имя_узла} {позиция_x} {позиция_y}".
- Число (M), означающее количество дуг графа.
- M строк вида $"{имя_исходящего_узла} {имя_входящего_узла}".

1.1.2 Требования редактору графов

На рис. 1 представлен эскиз будущего интерфейса программы.

Пользователь должен иметь возможность:

- Очистить граф (удалить все узлы и рёбра): при выборе в меню «Edit» пункта «Clear» будет вызван метод, который будет итеративно перебирать все существующие узлы и удалять все исходящие из него дуги, а затем и его самого.
- Создать узел на холсте двойным щелчком мыши: в функционале приложения будет предусмотрена возможность добавлять вершину к уже существующему и загруженному в приложение графу. Новая вершина будет определяться «именем» и координатами, определяющимися по щелчку мыши на холсте.
- Создать дугу между двумя узлами: в функционале приложения будет предусмотрена возможность добавлять дугу к уже существующему и загруженному в приложение графу. Новая дуга будет определяться вершиной-источником и вершиной-стоком выбранными пользователем.

- Удалить выделенный узел: после выделения узла и нажатия клавиши Delete будет вызван метод, удаляющий все связанные с этой вершиной дуги, а затем и саму вершину
- Удалить выделенную дугу: после выделения дуги и нажатия клавиши Delete будет вызван метод, удаляющий данную дугу из всех контейнеров, хранящих дуги в графе.
- Переименовать узел: после выделения узла и нажатия клавиши “R” будет выведено окно для ввода нового «имени» узла графа и вызвана функция изменяющее существующие название на введённое пользователем

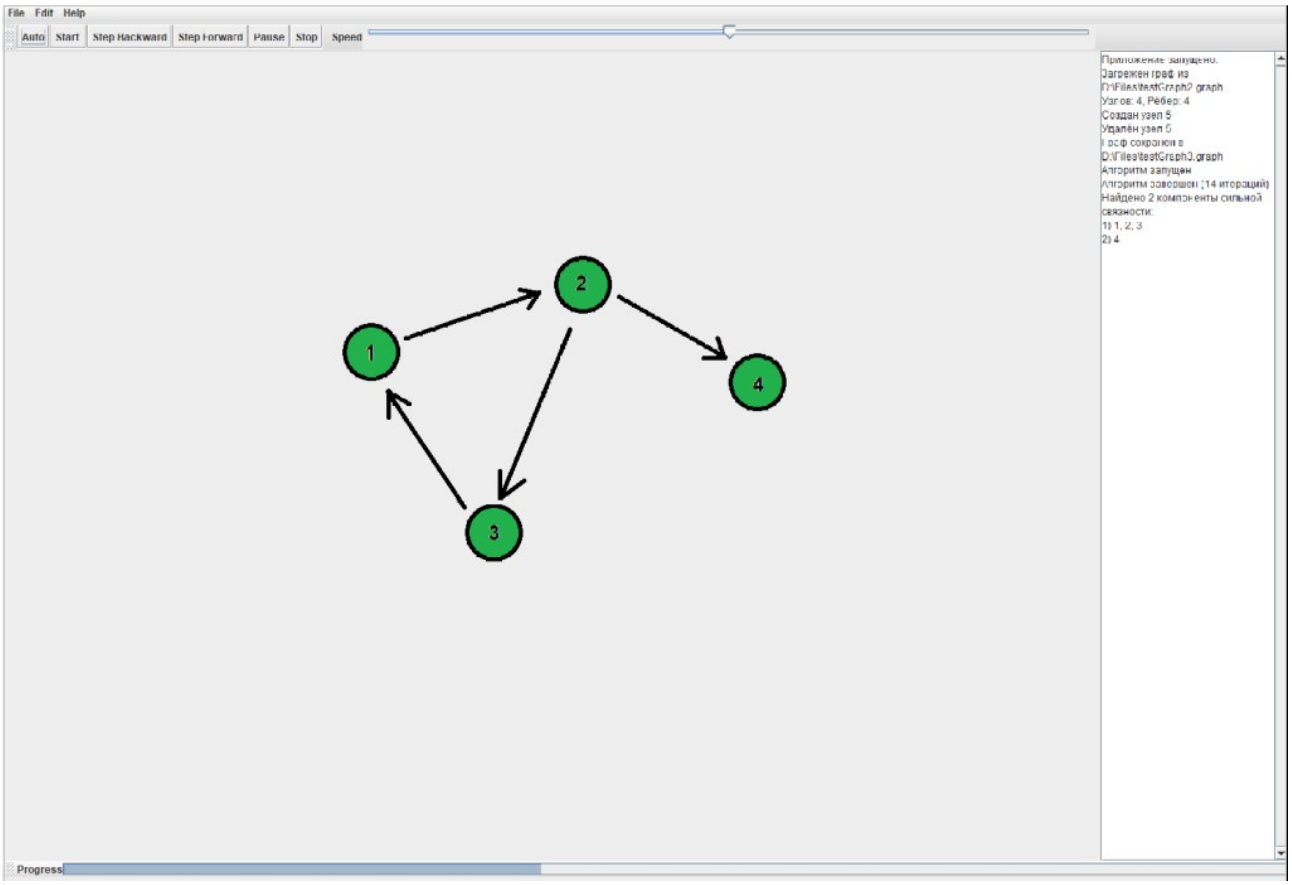

Рисунок 1 — Эскиз интерфейса

1.1.3 Требования к визуализации

На центральной панели программы будут изображены узлы графа в виде окружностей с «именами» внутри согласно координатам, заданным

пользователям, и соединённые стрелочками, которые указывают направления рёбер.

Необходимо реализовать следующие функции:

- Запуск алгоритма
- Запуск алгоритма пошагово
- Выполнить один шаг вперёд
- Выполнить один шаг назад
- Пауза выполнения алгоритма
- Остановка и сброс выполнения алгоритма
- Настройка скорости визуализации

Визуализация алгоритма будет заключаться в выводе шагов обхода графа (путём окрашивания вершин в разные цвета) и изменения направления рёбер графа. Найденные компоненты сильной связности будут окрашиваться в различные цвета. Пользователь сможет останавливать визуализацию алгоритма, вызывать и просматривать предыдущий и следующий шаги алгоритма, отслеживать изменения и пояснения к ним в текстовой форме, выводимые в правой части окна приложения (сообщения о загрузке и сохранении графа, количество вершин и ребер в нем, текущее состояние алгоритма (запущен, остановлен, на паузе, а также текущее количество пройденных, найденных, добавленных в список вершин, количество итераций и сообщение об инвертировании дуг при переходе между этапами алгоритма). Будет указан текущий этап алгоритма, так как алгоритм состоит из двух отличных этапов), какие компоненты были найдены и какие вершины в них входят). Также будет реализована возможность перемещать вершины зажатой левой кнопкой мыши, что позволит пользователю располагать вершины в удобном порядке.

1.2 УТОЧНЕНИЕ ТРЕБОВАНИЙ К ПРОГРАММЕ

1.2.1 Уточнение после сдачи версии 0.9

- Программа не должна создавать какие-либо файлы до того, как получит явное разрешение на это от пользователя
- Должна быть реализована кнопка, которая может и поставить на паузу работу алгоритма, и возобновить её
- Должен быть реализован чекбокс «Производить инвертирование графа мгновенно»
- Вершины должны содержать порядковые метки с учётом которых проводится второй обход графа
- У панели для вывода сообщений должна быть прокрутка
- Повторное нажатие на «стоп» должно очищать лог сообщений

1.2.2 Уточнения после сдачи версии 1.5

- У холста для вывода графа должна быть прокрутка
- Цвет ребра после второго обхода графа должен меняться

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

1. К 6 июля — прототип (реализован графический интерфейс, классы, представляющие граф, базовая отрисовка элементов (отображается сам граф)). Написан совместно составленный план тестирования готового проекта.
2. К 8 июля — первая версия проекта. Реализован алгоритм, частичная реализация отображения выполнения алгоритма (отображение шага выполнения вперед/назад, запуск/пауза). Улучшение интерфейса (добавление горячих клавиш). Проведено частичное тестирование программы, исправлены замечания и баги.
3. К 10 июля — итоговая версия проекта. Полная реализация логики визуализации (выбор между пошаговым и автоматическим ходом алгоритма, настройка скорости анимации), рефакторинг исходного кода, проведено тестирование.

2.2. Распределение ролей в бригаде

- Самойлова Анна (8303) — представление графа, контекстные меню графа
- Гоголев Евгений (8381) — класс алгоритма, логика визуализации (класс множества кадров анимации)
- Сахаров Виктор (8381) — графический интерфейс

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Архитектура проекта

UML-диаграмма проекта представлена ниже на рис. 2

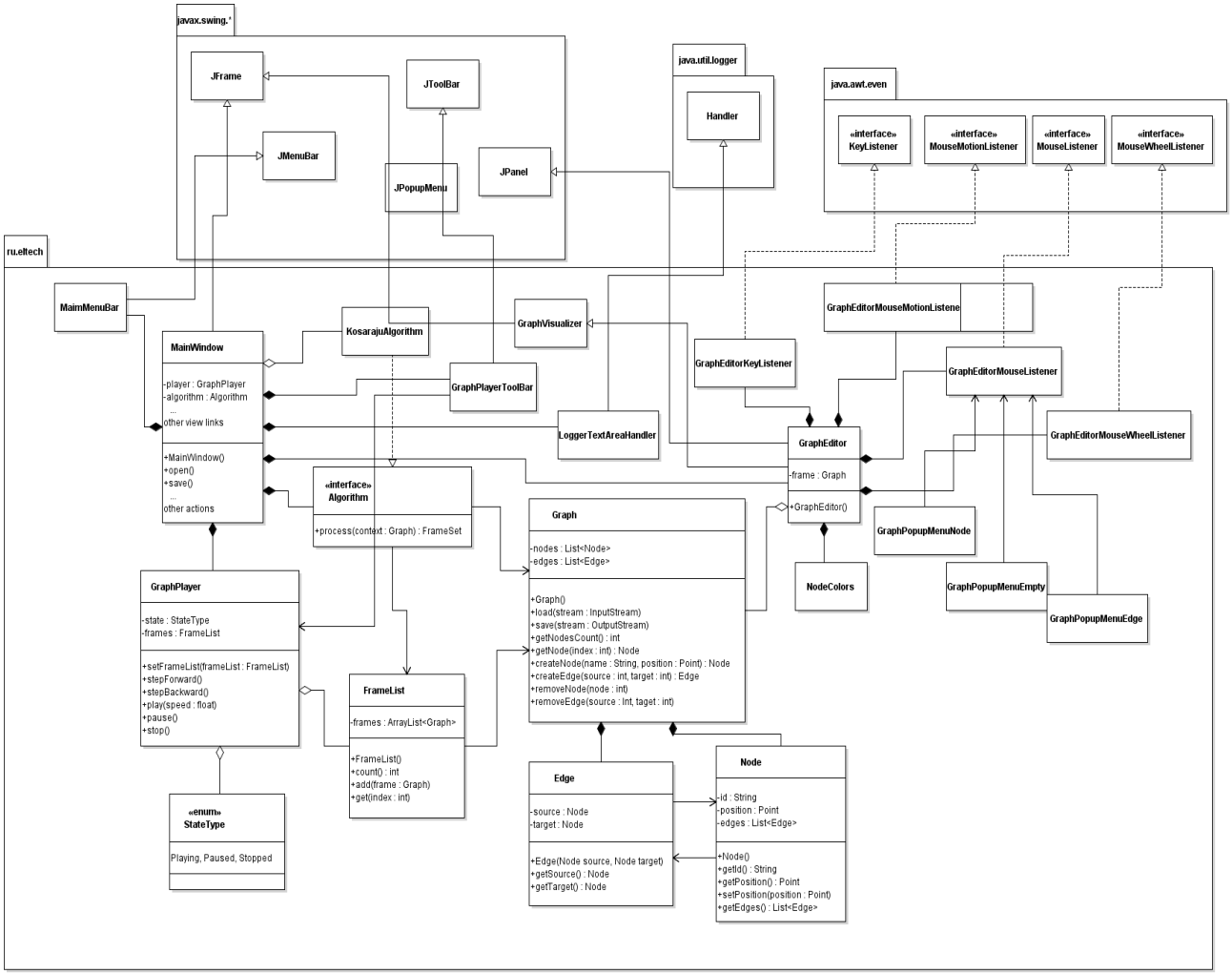

Рисунок 2 — UML-диаграмма проекта

3.2. Особенности хранения графа

Стандартная сериализация java не используется в пользу читабельности файла без использования программы, а также переносимости данных в программы, написанные на других языках программирования.

При загрузке (Graph.load) считывается весь файл целиком и создаётся объект класса Graph, при сохранении (Graph.save) объект класса Graph преобразуется в текстовое представление и записывается в файл с расширением

.graph. Если файл уже существует, то происходит перезапись с удалением старых данных.

В памяти приложения граф (экземпляр класса Graph) представляется в виде списка вершин (экземпляров класса Node) и списка рёбер (экземпляров класса Edge).

3.3 Отображение графа

Отображение графа в окне приложения будет осуществляться с использованием стандартных средств библиотеки swing для рисования.

3.4 Структуры данных и основные методы

Класс Node реализует вершины графа и хранит в себе:

• id — поле типа Integer, хранящее индивидуальный код каждой вершины

• name — поле типа String, хранящее имя вершины

• radius — поле типа int, хранящее информацию о том с каким радиусом рисовать шестиугольник, обозначающий вершину, на карте

• position — поле типа Point, хранящее информацию о положении вершины на карте в виде координат по осям Ox и Oy

• color — поле типа Color, хранящее цвет вершины, заданный пользователем

• visited — поле типа boolen, показывающее посещалась ли вершина во время текущего обхода графа

• highlighted — поле типа boolen, показывающее подсвечена ли вершина в данный момент

• strongComponentld — поле типа int, хранящее индивидуальный номер компоненты сильной связанности графа к которой относится вершина или -1 в случае если компоненты сильной связанности графа ещё не были найдены

• timeOut — поле типа int, хранящее время выхода из вершины при первом обходе графа во время поиска компонент сильной связанности.

Основные методы класса Node:

- getPosition() - возвращает позицию узла на экране

- setPosition() - устанавливает координаты узла
- getName() - возвращает имя узла
- setName() - задает узлу имя

Класс Edge реализует рёбра графа и хранит в себе:

• id — поле типа Integer, хранящее индивидуальный код каждого ребра

• source — поле типа Integer, хранящее индивидуальный код вершины, из которой исходит ребро

• target — поле типа Integer, хранящее индивидуальный код вершины, в которую входит ребро

• color — поле типа Color, хранящее цвет ребра, заданный пользователем

• stroke — поле типа int, хранящее толщину ребра, заданную пользователем

• highlighted — поле типа boolen, показывающее подсвечено ли ребро в данный момент

• connectsStrongComponents — поле типа boolean, показывающее соединяет ли ребро вершины, входящие в одну компоненту сильной связанности.

Основные методы класса Edge:

- getSourse() - возвращает id узла-источника
- getTarget() - возвращает id узла-цели
- invert() - разворачивает ребро

Класс Graph реализует граф и хранит в себе:

• nextNodeId — поле типа int, хранящее индивидуальный код вершины и используемое при переборе всех вершин в графе

• nextEdgeId — поле типа int, хранящее индивидуальный код ребра и используемое при переборе всех ребер в графе

• nodeMap — поле типа HashMap<Integer, Node>, хранящее ассоциативный массив ключами в котором являются индивидуальные номера вершин, а значением экземпляр класса Node

• edgeMap — поле типа HashMap<Integer, Edge>, хранящее ассоциативный массив ключами в котором являются индивидуальные номера рёбер, а значением экземпляр класса Edge

• state — поле типа String, хранящее уточняющее сообщение, которое будет выведено в log

Основные методы класса Graph:

- createNode() - создает узел
- createEdge() - создает ребро
- load() - загружает граф из файла
- save() - сохраняет граф в файл
- getEdgesFromNode() - возвращает список ребер, исходящих из заданного узла

Класс FrameList представляет собой список кадров анимации и является оберткой над списком графов. Его основные методы:

- add() - добавляет кадр в список
- get() - возвращает кадр с заданным индексом
- count() - возвращает число кадров

4. ТЕСТИРОВАНИЕ

4.1. Тестирование графического интерфейса

Примеры использования графического интерфейса представлены на рисунках ниже.

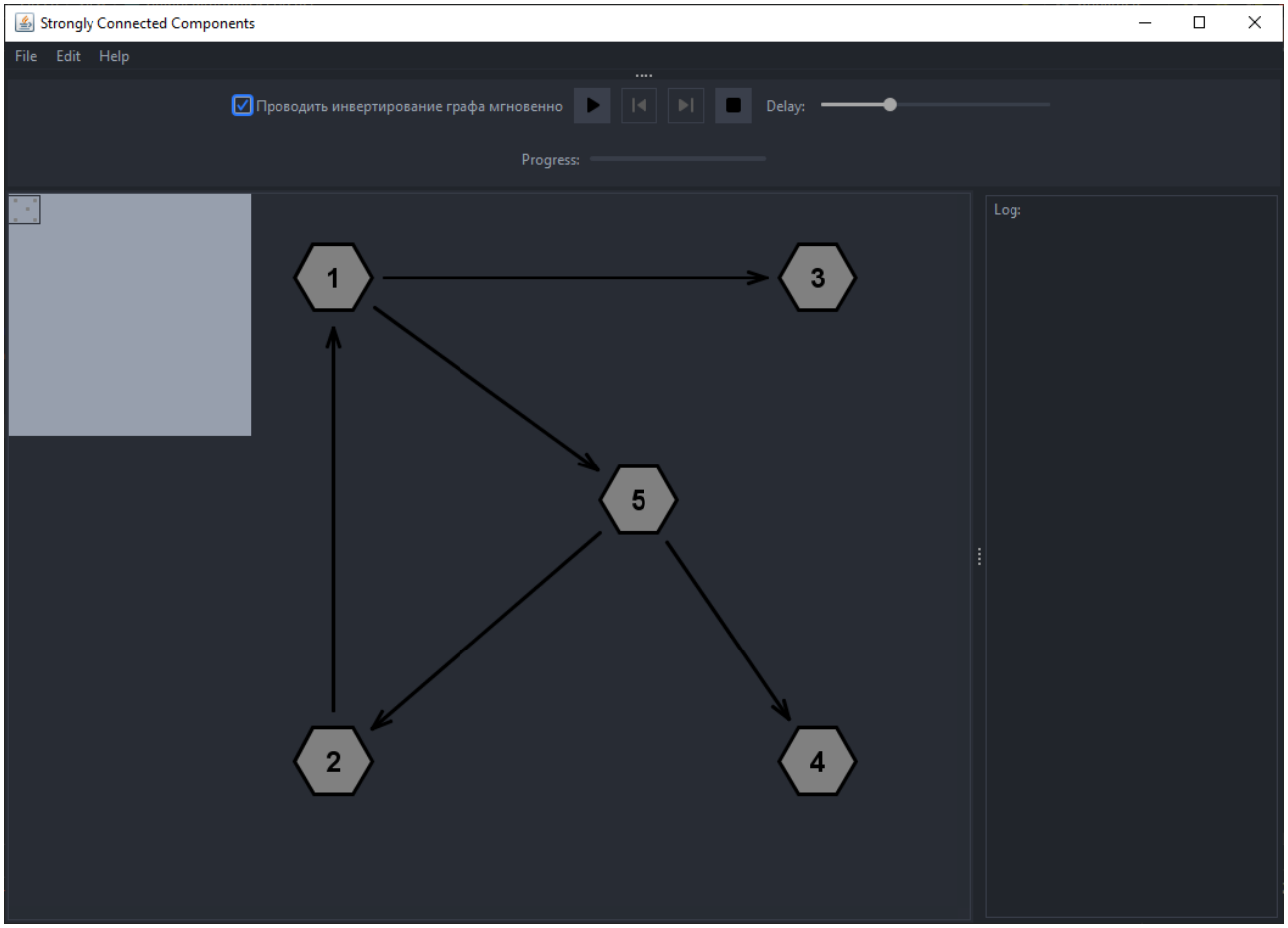

Рисунок 3 - Главное окно программы

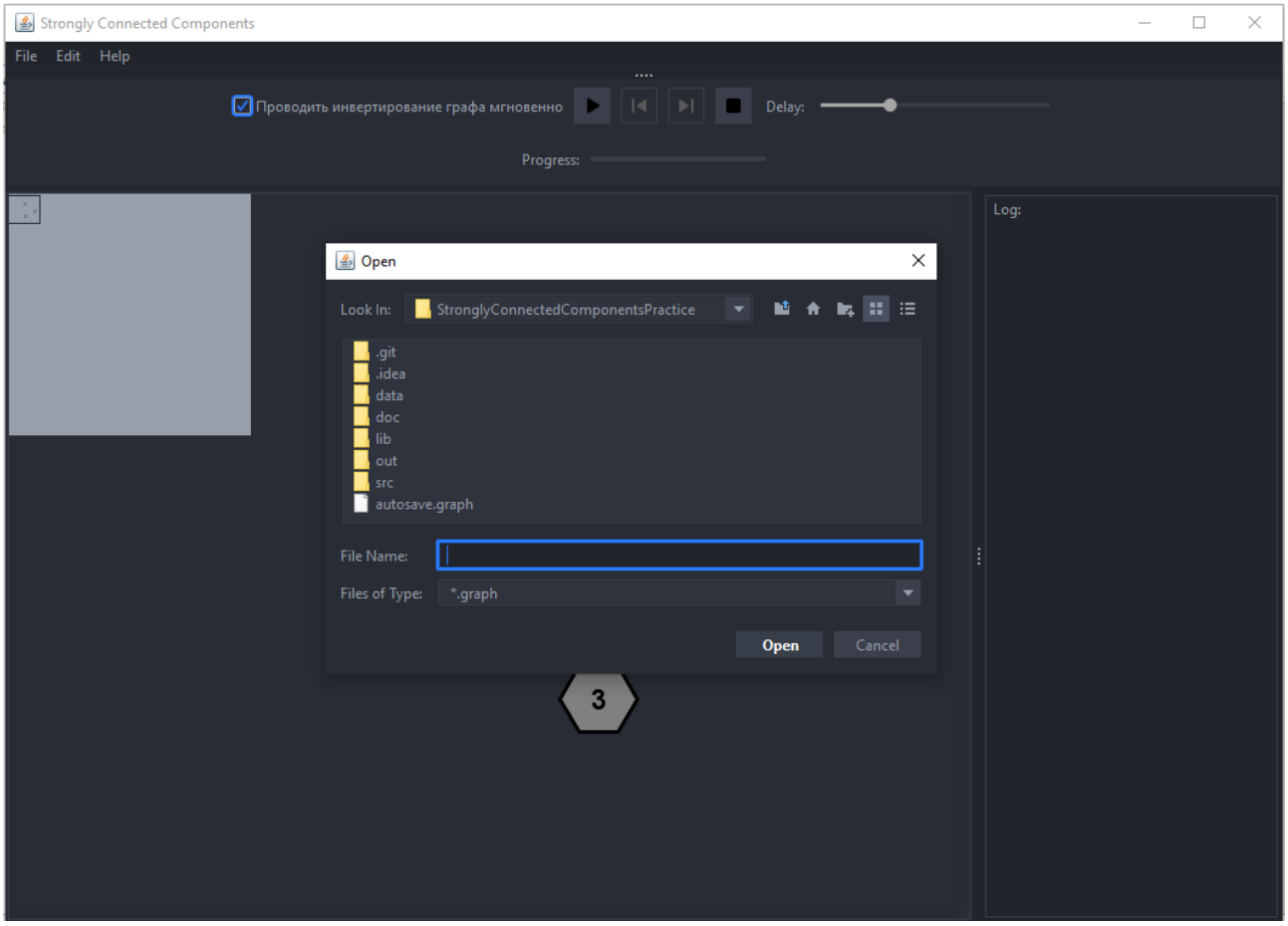

Рисунок 4 — Окно загрузки графа из файла

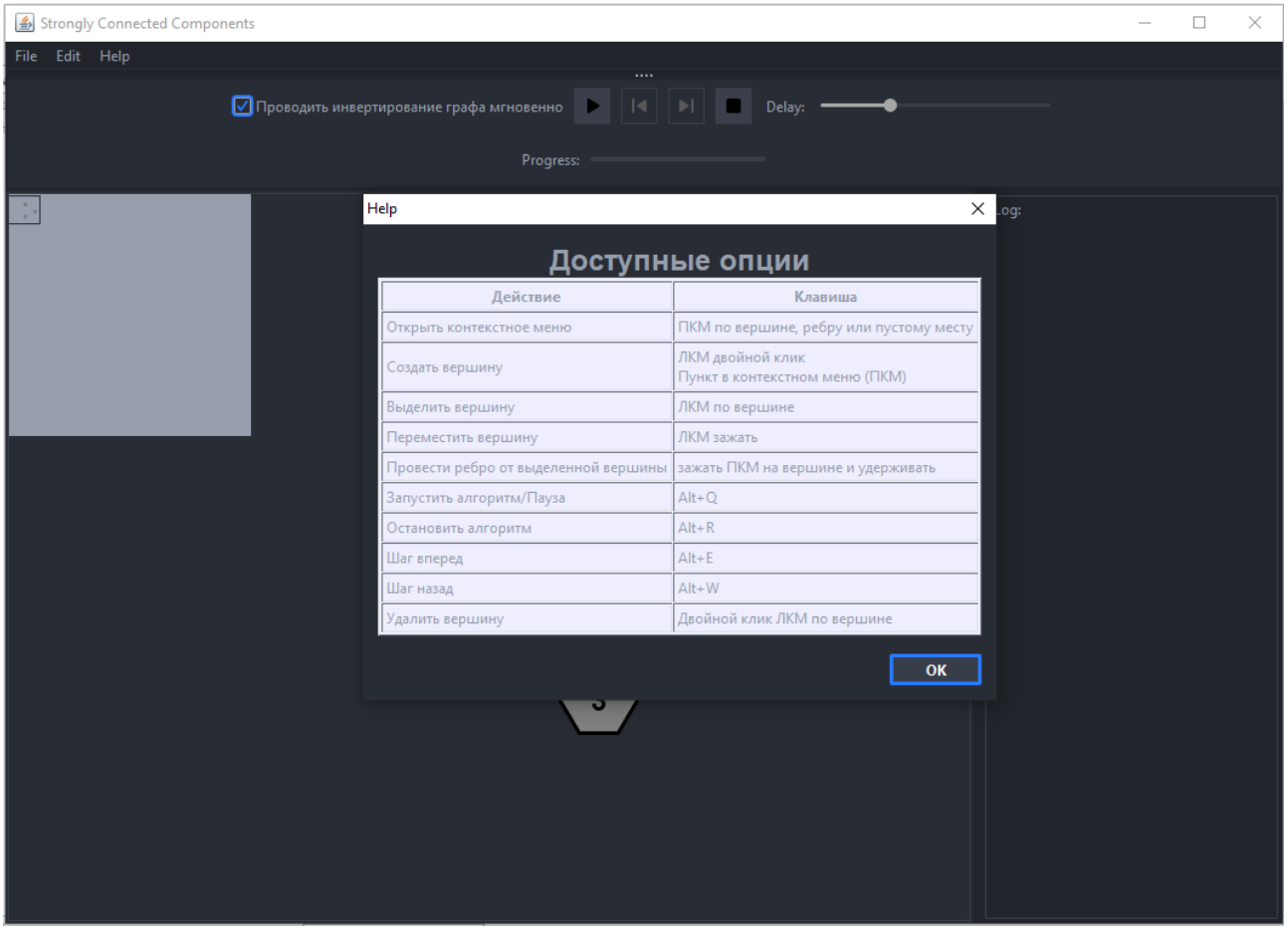

Рисунок 5 — Окно справки о программе

4.2. Тестирование работы алгоритма

Для проверки корректности работы алгоритма был создан граф, имеющий такие особенности как: узлы без ребер, узлы с двойными связями, компоненты сильной связности, соединенные ребром, отдельно стоящие компоненты сильной связности, цепи из нескольких узлов, циклы. На рис. 6 показано, что программа верно определила компоненты сильной связности в каждом из этих случаев.

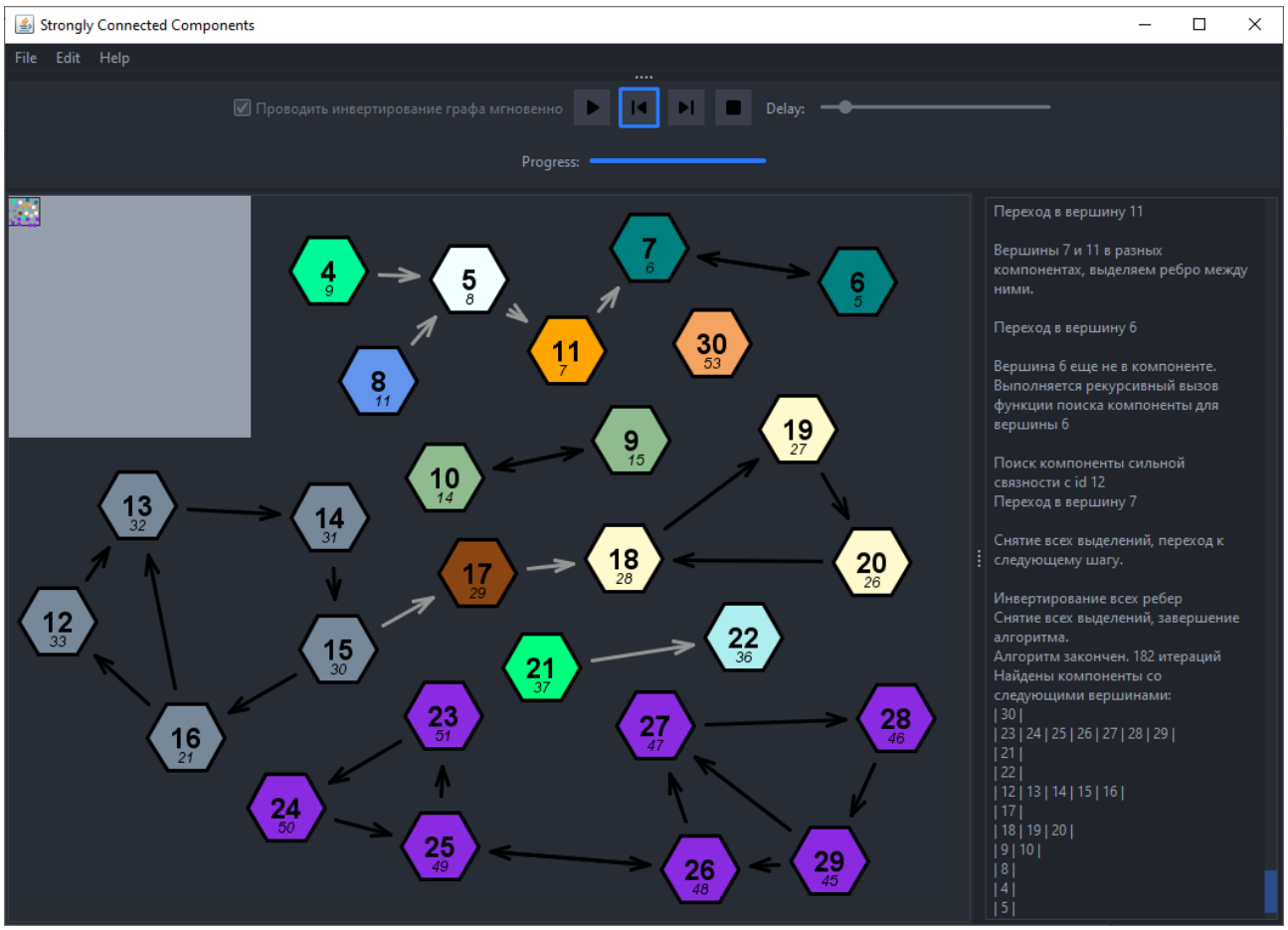

Рисунок 6 — Тестирование работы алгоритма

ЗАКЛЮЧЕНИЕ

В рамках учебной практики было построено приложение с графическим интерфейсом на языке Java, реализующее элементарную работу с графом и визуализирующее ход выполнения алгоритма Косарайю. Приложение также показывает подробную информацию о каждом шаге выводится в отдельное текстовое поле, что упрощает понимание работы алгоритма. Все требования к приложению соблюдены. Цель работы можно считать достигнутой.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Брюс Эккель. Философия JAVA. М.: Питер, 2019. 1168 с
2. Eric Freeman, Elisabeth Robson, Bert Bates, Kathy Sierra. Head First Design Patterns: A Brain-Friendly Guide, 2004 647 с
3. Роберт Лафоре. Структуры данных и алгоритмы java, 2013, 702 с
4. Официальная документация пакета javax.swing. URL: https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html (дата обращения 29.06.2020)
5. What is Thread-Safety and How to Achieve it? URL: https://www.baeldung.com/java-thread-safety (дата обращения 29.06.2020)
6. Поиск компонент сильной связности: алгоритм Косарайю. URL: https://habr.com/ru/post/331904/ (дата обращения 26.06.2020)
7. UML Class Diagrams Reference. URL: https://www.uml-diagrams.org/class-reference.html (дата обращения 27.06.2020)
8. GUI Designer Basics. URL: https://www.jetbrains.com/help/idea/gui-designer-basics.html (дата обращения 30.06.2020)
9. Listeners Supported by Swing Components. URL: https://docs.oracle.com/javase/tutorial/uiswing/events/eventsandcomponents.html (дата обращения 30.06.2020)
10. Gitflow Workflow. URL: https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow (дата обращения 26.06.2020)

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл Main.java

```
package ru.eltech;
import
com.formdev.flatlaf.intellijthemes.materialthemeuilite.FlatAtomOneDarkContrastIJ
Theme;
import ru.eltech.view.MainWindow;
public final class Main {
    public static void main(String[] args) {
        FlatAtomOneDarkContrastIJTheme.install();
        MainWindow window = new MainWindow();
        window.setVisible(true);
    }
}
```

Файл MainWindow.java

```
package ru.eltech.view;

import com.formdev.flatlaf.intellijthemes.materialthemeuilite.FlatGitHubIJTheme;
import ru.eltech.logic.*;

import javax.swing.*;
import javax.swing.filechooser.FileNameExtensionFilter;
import java.awt.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.*;
import java.util.logging.Logger;
import java.util.stream.Collectors;


public final class MainWindow extends JFrame {
    public static Logger log = Logger.getLogger(MainWindow.class.getName());

    private enum Themes {
        DARK,
        BRIGHT
    }

    private Themes currentTheme = Themes.DARK;
    private JPanel content;
    private JTextPane loggerPane;
    private GraphEditor graphEditor;
    private GraphPlayerToolBar graphPlayerToolBar;
    private JScrollPane scrollPane;
    JScrollPane scrolling;

    private Graph graphOrigin;
    private GraphPlayer graphPlayer;
    private LoggerTextAreaHandler loggerTextAreaHandler;

    private final KosarajuAlgorithm algorithm = new KosarajuAlgorithm();

    private final String AUTOSAVE_FILE = "autosave.graph";

    public MainWindow() {
        super("Strongly Connected Components");
        $$$setupUI$$$();
        initialize();
        pack();
```

```
    }

    private void initialize() {
        setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);
        setContentPane(content);
        setMinimumSize(new Dimension(1024, 768));
        setResizable(true);
        setLocationRelativeTo(null);
        setJMenuBar(new MainMenuBar(this));
        //setExtendedState(JFrame.MAXIMIZED_BOTH);
        addWindowListener(new WindowAdapter() {
            @Override
            public void windowClosed(WindowEvent event) {
                System.exit(0);
            }

            @Override
            public void windowClosing(WindowEvent event) {
                int reply = JOptionPane.showConfirmDialog(null, "Сохранить
текущий граф в файл " + AUTOSAVE_FILE + "?", "Выход",
JOptionPane.YES_NO_CANCEL_OPTION);
                if (reply == JOptionPane.YES_OPTION) {
                    serializeGraph(new File(AUTOSAVE_FILE), true);
                } else if (reply == JOptionPane.CANCEL_OPTION) {
                    return;
                } else if (reply == JOptionPane.CLOSED_OPTION) {
                    return;
                }
                windowClosed(event);
            }
        });
        graphOrigin = new Graph();
        graphPlayer = new GraphPlayer(graphPlayerToolBar);
        graphPlayerToolBar.setParent(this);
        loggerTextAreaHandler = new LoggerTextAreaHandler(scrollPane,
loggerPane);
        log.addHandler(loggerTextAreaHandler);
        deserializeGraph(new File(AUTOSAVE_FILE), true);
        scrolling = new JScrollPane(graphEditor,
                JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
                JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
        scrolling.setPreferredSize(new Dimension(800, 600));
        scrolling.setAutoscrolls(true);
        content.add(scrolling);
    }

    public GraphEditor getGraphEditor() {
        return graphEditor;
    }

    public GraphPlayer getGraphPlayer() {
        return graphPlayer;
    }

    /**
     * @return актуальная версия графа
     */
    public Graph getGraphOrigin() {
        if (!graphEditor.isReadOnly && graphEditor.isModified) {
            graphOrigin = graphEditor.getGraphCopy();
        }
        return graphOrigin;
    }
```

```
    /**
     * Принудительно заменяет текущий граф и его визуализацию
     */
    public void setGraphOrigin(Graph graph) {
        graphPlayer.setState(GraphPlayer.State.Stop);
        graphOrigin = graph;
        graphEditor.setGraphCopy(graph);
        graphEditor.isReadOnly = false;
    }

    // region LIFECYCLE

    public void playerPlaying() {
        //log.info("playerPlaying");
        if (!graphEditor.isReadOnly) {
            graphEditor.isReadOnly = true;
            graphOrigin = graphEditor.getGraphCopy();
            graphPlayer.setFrameList(algorithm.process(new
Graph(getGraphOrigin())));
        }
    }

    public void playerPausing() {
        //log.info("Анимация поставлена на паузу");
    }

    public void playerStopping() {
        //log.info("playerStopping");
        if (graphEditor.isReadOnly) {
            graphEditor.isReadOnly = false;
            graphEditor.setGraphCopy(graphOrigin);
        }
    }

    public void playerVisualizing(Graph graph) {
        graphEditor.setGraphCopy(graph);
        if (graph.state != null)
            log.info(graph.state);
    }

    public void setImmediateReverse(boolean immediateReverse) {
        algorithm.setImmediateReverse(immediateReverse);
        switch (graphPlayer.getState()) {
            case Play:
                log.warning("Попытка изменения анимации во время проигрывания");
                break;
            case Pause:
                log.warning("Изменение анимации на паузе не предусмотрено");
            case Stop:
                graphPlayer.setFrameList(algorithm.process(new
Graph(getGraphOrigin())));
                break;
        }
    }

    // endregion

    //region ACTIONS

    public void createNewGraph() {
        setGraphOrigin(new Graph());
    }
```

```
    public void loadExampleGraph() {
        Graph graph = new Graph();
        Node n1 = graph.createNode(100, 100);
        Node n2 = graph.createNode(100, 500);
        Node n3 = graph.createNode(500, 100);
        Node n4 = graph.createNode(500, 500);
        Node n5 = graph.createNode(300, 300);
        graph.createEdge(n2, n1);
        graph.createEdge(n1, n3);
        graph.createEdge(n1, n5);
        graph.createEdge(n5, n4);
        graph.createEdge(n5, n2);
        setGraphOrigin(graph);
    }

    public void serializeGraph() {
        JFileChooser fc = new JFileChooser(".");
        fc.setMultiSelectionEnabled(false);
        FileNameExtensionFilter filter = new FileNameExtensionFilter(" *.graph",
"graph");
        fc.addChoosableFileFilter(filter);
        fc.setFileFilter(filter);

        int chosenOption = fc.showSaveDialog(this);
        if (chosenOption == JFileChooser.APPROVE_OPTION) {
            serializeGraph(fc.getSelectedFile(), false);
        }
    }

    public void serializeGraph(File file, boolean silently) {
        String fileName = file.getAbsolutePath();
        if (!fileName.endsWith(".graph")) file = new File(fileName + ".graph");
        try (FileOutputStream fos = new FileOutputStream(file)) {
            getGraphOrigin().save(fos);
            if (!silently) JOptionPane.showMessageDialog(null, "Граф сохранён
успешно " + file.getAbsolutePath());
        } catch (IOException e) {
            if (!silently)
                JOptionPane.showMessageDialog(this, e.getMessage(), "Файл не
найден!", JOptionPane.ERROR_MESSAGE);
        }
    }

    public void deserializeGraph() {
        JFileChooser fc = new JFileChooser(".");
        fc.setMultiSelectionEnabled(false);
        FileNameExtensionFilter filter = new FileNameExtensionFilter(" *.graph",
"graph");
        fc.addChoosableFileFilter(filter);
        fc.setFileFilter(filter);

        int chosenOption = fc.showOpenDialog(this);
        if (chosenOption == JFileChooser.APPROVE_OPTION) {
            deserializeGraph(fc.getSelectedFile(), false);
        }
    }

    public void deserializeGraph(File file, boolean silently) {
        try (FileInputStream fis = new FileInputStream(file)) {
            setGraphOrigin(new Graph().load(fis));
            if (!silently) JOptionPane.showMessageDialog(null, "Граф загружен
успешно " + file.getAbsolutePath());
```

```
        } catch (IOException e) {
            if (!silently) e.printStackTrace();
            if (!silently)
                JOptionPane.showMessageDialog(this, e.getMessage(), "Файл не
найден!", JOptionPane.ERROR_MESSAGE);
        }
    }

    public void showNodesList() {
        Graph curGraph = graphPlayer.getFrameList() != null ?
graphPlayer.getFrameList().get(graphPlayer.getCurrentFrame()) :
getGraphOrigin();
        StringBuilder msg = new StringBuilder();
        for (Node node : curGraph.getNodes()) {
            msg.append(node.getDescription()).append("\n");
        }
        JOptionPane.showMessageDialog(this, msg.toString(), "Список вершин",
JOptionPane.INFORMATION_MESSAGE);
    }

    public void showEdgesList() {
        Graph curGraph = graphPlayer.getFrameList() != null ?
graphPlayer.getFrameList().get(graphPlayer.getCurrentFrame()) :
getGraphOrigin();
        StringBuilder msg = new StringBuilder();
        for (Edge edge : curGraph.getEdges()) {
            msg.append(edge.getDescription()).append("\n");
        }
        JOptionPane.showMessageDialog(this, msg.toString(), "Список ребер",
JOptionPane.INFORMATION_MESSAGE);
    }

    public void showInstruction() {
        showHtmlFormattedMessageDialog("/resources/docs/helpMessage.html",
"Help", JOptionPane.PLAIN_MESSAGE);
    }

    public void showAuthorsInfo() {
        showHtmlFormattedMessageDialog("/resources/docs/authors.html",
"Authors", JOptionPane.PLAIN_MESSAGE);
    }

    public void clearLog() {
        loggerTextAreaHandler.clear();
    }

    public void changeTheme() {
//        FlatGitHubIJTheme.install();
//        SwingUtilities.updateComponentTreeUI(this);
//        pack();
        switch (currentTheme) {
            case DARK:
                try {

UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
                } catch (ClassNotFoundException |
UnsupportedLookAndFeelException | IllegalAccessException |
InstantiationException e) {
                    try {

UIManager.setLookAndFeel("com.jtattoo.plaf.graphite.GraphiteLookAndFeel");
                    } catch (ClassNotFoundException classNotFoundException) {
                        classNotFoundException.printStackTrace();
```

```
                    } catch (InstantiationException instantiationException) {
                        instantiationException.printStackTrace();
                    } catch (IllegalAccessException illegalAccessException) {
                        illegalAccessException.printStackTrace();
                    } catch (UnsupportedLookAndFeelException
unsupportedLookAndFeelException) {
                        unsupportedLookAndFeelException.printStackTrace();
                    }
                }
                SwingUtilities.updateComponentTreeUI(this);
                pack();
                currentTheme = Themes.DARK;
                break;
            case BRIGHT:
                try {

UIManager.setLookAndFeel("com.jtattoo.plaf.graphite.GraphiteLookAndFeel");
                } catch (ClassNotFoundException |
UnsupportedLookAndFeelException | IllegalAccessException |
InstantiationException e) {
                    try {

UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
                    } catch (ClassNotFoundException classNotFoundException) {
                        classNotFoundException.printStackTrace();
                    } catch (InstantiationException instantiationException) {
                        instantiationException.printStackTrace();
                    } catch (IllegalAccessException illegalAccessException) {
                        illegalAccessException.printStackTrace();
                    } catch (UnsupportedLookAndFeelException
unsupportedLookAndFeelException) {
                        unsupportedLookAndFeelException.printStackTrace();
                    }
                }
                SwingUtilities.updateComponentTreeUI(this);
                pack();
                currentTheme = Themes.BRIGHT;
                break;
        }
    }

    //endregion

    /**
     * @param filename
     * @param title
     * @param messageType
     * @apiNote Shows message dialog, takes message text from a html-encoded
file in resources folder
     */
    private void showHtmlFormattedMessageDialog(String filename, String title,
int messageType) {
        String formattedText = readResourceFileAsString(filename, "UTF-8");
        JLabel message = new JLabel(formattedText);
        JOptionPane.showMessageDialog(this, message, title, messageType);
    }

    /**
     * @return file as String
     * @apiNote Helper function to read html resources
     */
    private String readResourceFileAsString(String filename, String charsetName)
{
```

```
        try (InputStreamReader is = new
InputStreamReader(getClass().getResourceAsStream(filename), charsetName)) {
            return new BufferedReader(is).lines().collect(Collectors.joining("\
n"));
        } catch (UnsupportedEncodingException e) {
            JOptionPane.showMessageDialog(this, "Incorrect encoding", "Error",
JOptionPane.ERROR_MESSAGE);
        } catch (IOException e) {
            e.printStackTrace();
            JOptionPane.showMessageDialog(this, "File not found", "Error",
JOptionPane.ERROR_MESSAGE);
        }
        return null;
    }

    /**
     * Method generated by IntelliJ IDEA GUI Designer
     * >>> IMPORTANT!! <<<
     * DO NOT edit this method OR call it in your code!
     *
     * @noinspection ALL
     */
    private void $$$setupUI$$$() {
        content = new JPanel();
        content.setLayout(new BorderLayout(0, 0));
        content.setMinimumSize(new Dimension(622, 400));
        graphEditor = new GraphEditor();
        graphEditor.setMinimumSize(new Dimension(100, 10));
        graphEditor.setPreferredSize(new Dimension(512, 10));
        content.add(graphEditor, BorderLayout.CENTER);
        final JToolBar toolBar1 = new JToolBar();
        toolBar1.setMinimumSize(new Dimension(64, 22));
        toolBar1.setPreferredSize(new Dimension(256, 22));
        content.add(toolBar1, BorderLayout.EAST);
        scrollPane = new JScrollPane();
        scrollPane.setHorizontalScrollBarPolicy(31);
        scrollPane.setPreferredSize(new Dimension(64, 16));
        toolBar1.add(scrollPane);
        loggerPane = new JTextPane();
        loggerPane.setEditable(false);
        scrollPane.setViewportView(loggerPane);
        graphPlayerToolBar = new GraphPlayerToolBar();
        graphPlayerToolBar.setMaximumSize(new Dimension(10000, 10000));
        graphPlayerToolBar.setMinimumSize(new Dimension(10, 10));
        graphPlayerToolBar.setPreferredSize(new Dimension(1000, 100));
        content.add(graphPlayerToolBar, BorderLayout.NORTH);
    }
    /**
     * @noinspection ALL
     */
    public JComponent $$$getRootComponent$$$() {
        return content;
    }
}
```

Файл NodeColors.java

```
package ru.eltech.view;

import java.awt.*;

public final class NodeColors {
    public static final Color[] colors  = {
            new Color(244,164,96), /*Sandy brown*/
            new Color(138,43,226), /*Blue violet*/
```

```
new Color(0,255,127), /*Spring green*/
new Color(175,238,238), /*Pale turquoise*/
new Color(119,136,153), /*Light slate gray*/
new Color(139,69,19), /*Saddle brown*/
new Color(255,250,205), /*Lemon chiffon*/
new Color(143,188,143), /*Dark sea green*/
new Color(100,149,237), /*Corn flower blue*/
new Color(0,250,154), /*Medium spring green*/
new Color(240,255,255), /*Azure*/
new Color(255,165,0), /*Orange*/
new Color(0,128,128), /*Teal*/
new Color(192,192,192), /*Silver*/
new Color(218,112,214), /*Orchid*/
new Color(255,105,180), /*Hot pink*/
new Color(205,92,92), /*Indian red*/
new Color(0,0,255), /*Blue*/
new Color(75,0,130), /*Indigo*/
new Color(34,139,34), /*Forest green*/
new Color(250,128,114), /*Salmon*/
new Color(255,218,185), /*Peach puff*/
new Color(186,85,211), /*Medium orchid*/
new Color(210,180,140), /*Tan*/
new Color(189,183,107), /*Dark khaki*/
new Color(0,0,205), /*Medium blue*/
new Color(255,0,255), /*Magenta / fuchsia*/
new Color(219,112,147), /*Pale violet red*/
new Color(238,232,170), /*Pale golden rod*/
new Color(205,133,63), /*Peru*/
new Color(165,42,42), /*Brown*/
new Color(255,0,0), /*Red*/
new Color(0,0,128), /*Navy*/
new Color(221,160,221), /*Plum*/
new Color(65,105,225), /*Royal blue*/
new Color(95,158,160), /*Cadet blue*/
new Color(50,205,50), /*Lime green*/
new Color(238,130,238), /*Violet*/
new Color(107,142,35), /*Olive drab*/
new Color(173,255,47), /*Green yellow*/
new Color(102,205,170), /*Medium aqua marine*/
new Color(255,250,240), /*Floral white*/
new Color(0,255,255), /*Cyan*/
new Color(70,130,180), /*Steel blue*/
new Color(169,169,169), /*Dark gray / dark grey*/
new Color(135,206,250), /*Light sky blue*/
new Color(255,255,0), /*Yellow*/
new Color(230,230,250), /*Lavender*/
new Color(222,184,135), /*Burly wood*/
new Color(154,205,50), /*Yellow green*/
new Color(255,239,213), /*Papaya whip*/
new Color(160,82,45), /*Sienna*/
new Color(144,238,144), /*Light green*/
new Color(25,25,112), /*Midnight blue*/
new Color(139,0,139), /*Dark magenta*/
new Color(245,245,220), /*Beige*/
new Color(147,112,219), /*Medium purple*/
new Color(255,228,181), /*Moccasin*/
new Color(72,209,204), /*Medium turquoise*/
new Color(255,140,0), /*Dark orange*/
new Color(0,100,0), /*Dark green*/
new Color(127,255,0), /*Chart reuse*/
new Color(220,20,60), /*Crimson*/
new Color(245,245,245), /*White smoke*/
new Color(0,0,139), /*Dark blue*/
```

```
            new Color(255,215,0), /*Gold*/
            new Color(255,160,122), /*Light salmon*/
            new Color(240,128,128), /*Light coral*/
            new Color(112,128,144), /*Slate gray*/
            new Color(153,50,204), /*Dark orchid*/
            new Color(0,128,0), /*Green*/
            new Color(106,90,205), /*Slate blue*/
            new Color(124,252,0), /*Lawn green*/
            new Color(64,224,208), /*Turquoise*/
            new Color(85,107,47), /*Dark olive green*/
            new Color(0,206,209), /*Dark turquoise*/
            new Color(188,143,143), /*Rosy brown*/
            new Color(255,99,71), /*Tomato*/
            new Color(220,220,220), /*Gainsboro*/
            new Color(173,216,230), /*Light blue*/
            new Color(148,0,211), /*Dark violet*/
            new Color(152,251,152), /*Pale green*/
            new Color(128,128,128), /*Gray / grey*/
            new Color(255,235,205), /*Blanched almond*/
            new Color(255,240,245), /*Lavender blush*/
            new Color(60,179,113), /*Medium sea green*/
            new Color(216,191,216), /*Thistle*/
            new Color(128,0,0), /*Maroon*/
            new Color(245,222,179), /*Wheat*/
            new Color(47,79,79), /*Dark slate gray*/
            new Color(0,255,255), /*Aqua*/
            new Color(255,69,0), /*Orange red*/
            new Color(72,61,139), /*Dark slate blue*/
            new Color(30,144,255), /*Dodger blue*/
            new Color(255,127,80), /*Coral*/
            new Color(127,255,212), /*Aqua marine*/
            new Color(255,182,193), /*Light pink*/
            new Color(123,104,238), /*Medium slate blue*/
            new Color(128,128,0), /*Olive*/
            new Color(139,0,0), /*Dark red*/
            new Color(255,192,203), /*Pink*/
            new Color(32,178,170), /*Light sea green*/
            new Color(255,255,255), /*White*/
            new Color(255,222,173), /*Navajo white*/
            new Color(240,230,140), /*Khaki*/
            new Color(211,211,211), /*Light gray / light grey*/
            new Color(233,150,122), /*Dark salmon*/
            new Color(0,255,0), /*Lime*/
            new Color(210,105,30), /*Chocolate*/
            new Color(46,139,87), /*Sea green*/
            new Color(135,206,235), /*Sky blue*/
            new Color(255,20,147), /*Deep pink*/
            new Color(218,165,32), /*Golden rod*/
            new Color(250,240,230), /*Linen*/
            new Color(0,139,139), /*Dark cyan*/
            new Color(0,191,255), /*Deep sky blue*/
            new Color(184,134,11), /*Dark golden rod*/
            new Color(128,0,128), /*Purple*/
            new Color(255,228,196), /*Bisque*/
            new Color(255,248,220), /*Corn silk*/
            new Color(199,21,133), /*Medium violet red*/
            new Color(178,34,34) /*Firebrick*/
    };
}
```

Файл MainMenuBar.java

```
package ru.eltech.view;

import javax.swing.*;
```

```
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;

/**
 * Обёртка над панелью меню в {@link MainWindow}
 */
@SuppressWarnings("FieldCanBeLocal")
public final class MainMenuBar extends JMenuBar implements ActionListener {
    private final MainWindow parent;

    private final JMenu fileMenu = new JMenu("File");
    private final JMenuItem newGraphMenuItem = new JMenuItem("Новый граф");
    private final JMenuItem showExampleMenuItem = new JMenuItem("Загрузить
пример графа");
    private final JMenuItem saveMenuItem = new JMenuItem("Сохранить...");
    private final JMenuItem loadMenuItem = new JMenuItem("Загрузить...");
    // TODO

    private final JMenu editMenu = new JMenu("Edit");
    private final JMenuItem nodesMenuItem = new JMenuItem("Показать список
нод...");
    private final JMenuItem edgesMenuItem = new JMenuItem("Показать список
дуг...");
    // TODO

    private final JMenu viewMenu = new JMenu("View");
    private final JMenuItem startMenuItem = new JMenuItem("Запустить алгоритм");
    //private final JMenuItem changeThemeMenuItem = new JMenuItem("Сменить
тему");
    // TODO

    private final JMenu helpMenu = new JMenu("Help");
    private final JMenuItem appMenuItem = new JMenuItem("О программе...");
    private final JMenuItem authorMenuItem = new JMenuItem("Об авторах...");

    public MainMenuBar(MainWindow parent) {
        this.parent = parent;
        newGraphMenuItem.addActionListener(this);
        newGraphMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_N,
Toolkit.getDefaultToolkit().getMenuShortcutKeyMask()));
        showExampleMenuItem.addActionListener(this);
        showExampleMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_P,
Toolkit.getDefaultToolkit().getMenuShortcutKeyMask()));
        saveMenuItem.addActionListener(this);
        saveMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_S,
Toolkit.getDefaultToolkit().getMenuShortcutKeyMask()));
        loadMenuItem.addActionListener(this);
        loadMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_L,
Toolkit.getDefaultToolkit().getMenuShortcutKeyMask()));
        nodesMenuItem.addActionListener(this);
        edgesMenuItem.addActionListener(this);
        startMenuItem.addActionListener(this);
        //changeThemeMenuItem.addActionListener(this);
        appMenuItem.addActionListener(this);
        appMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_PERIOD,
Toolkit.getDefaultToolkit().getMenuShortcutKeyMask()));
        authorMenuItem.addActionListener(this);

        fileMenu.add(newGraphMenuItem);
        fileMenu.addSeparator();
        fileMenu.add(showExampleMenuItem);
```

```
        fileMenu.addSeparator();
        fileMenu.add(saveMenuItem);
        fileMenu.add(loadMenuItem);
        add(fileMenu);

        editMenu.add(nodesMenuItem);
        editMenu.add(edgesMenuItem);
        add(editMenu);

        //viewMenu.add(startMenuItem);
        //viewMenu.add(changeThemeMenuItem);
        //add(viewMenu);

        helpMenu.add(appMenuItem);
        helpMenu.add(authorMenuItem);
        add(helpMenu);
    }

    /**
     * Проброс событий меню в {@link MainWindow}
     */
    @Override
    public void actionPerformed(ActionEvent e) {
        Object eSource = e.getSource();
        if (eSource == newGraphMenuItem) {
            parent.createNewGraph();
        } else if (eSource == showExampleMenuItem) {
            parent.loadExampleGraph();
        } else if (eSource == saveMenuItem) {
            parent.serializeGraph();
        } else if (eSource == loadMenuItem) {
            parent.deserializeGraph();
        } else if (eSource == nodesMenuItem) {
            parent.showNodesList();
        } else if (eSource == edgesMenuItem) {
            parent.showEdgesList();
        } else if (eSource == startMenuItem) {
            //parent.startVisualizing();
        //} else if (eSource == changeThemeMenuItem) {
            //parent.changeTheme();
        } else if (eSource == appMenuItem) {
            parent.showInstruction();
        } else if (eSource == authorMenuItem) {
            parent.showAuthorsInfo();
        }
    }
}
```

Файл LoggerTextAreaHandler.java

```
package ru.eltech.view;

import javax.swing.*;
import java.util.ArrayList;
import java.util.logging.Handler;
import java.util.logging.Level;
import java.util.logging.LogRecord;

public final class LoggerTextAreaHandler extends Handler {
    private final JScrollPane scrollPane;
    private final JTextPane textPane;
    private final ArrayList<String> messages = new ArrayList<>();

    public LoggerTextAreaHandler(JScrollPane scrollPane, JTextPane textPane) {
        this.scrollPane = scrollPane;
```

```
        this.textPane = textPane;
        this.textPane.setContentType("text/html");
        render();
    }

    @Override
    public void publish(LogRecord record) {
        String levelLabel = record.getLevel() == Level.INFO ? "" :
record.getLevel().getLocalizedName();
        messages.add(String.format("%s %s", levelLabel, record.getMessage()));
        render();
    }

    private void render() {
        StringBuilder content = new StringBuilder();
        //content.append("<html><body style=\"text-align: justify; text-justify:
inter-word;\">Log:<br>");
        content.append("<html><body>Log:<br>");
        for (String message : messages) {
            content.append(message);
            if (!message.trim().isEmpty()) content.append("<br>");
        }
        content.append("</body></html>");
        textPane.setText(content.toString());
        // Хак для того, чтобы скроллбар оставался строго внизу
        if (!scrollPane.getVerticalScrollBar().getValueIsAdjusting())
SwingUtilities.invokeLater(() -> {
            try {

scrollPane.getVerticalScrollBar().setValue(scrollPane.getVerticalScrollBar().get
Maximum());
            } catch (Exception e) {
                e.printStackTrace();
            }
        });
    }

    @Override
    public void flush() {
        // Nothing
    }
    @Override
    public void close() throws SecurityException {
        clear();
    }
    public void clear() {
        messages.clear();
        render();
    }
}
```

Файл GraphVisualizer.java

```
package ru.eltech.view;

import ru.eltech.logic.Edge;
import ru.eltech.logic.Graph;
import ru.eltech.logic.Node;

import javax.swing.*;
import java.awt.*;

/**
 * Класс, отвечающий за логику отображения графа
 */
```

```
public class GraphVisualizer extends JPanel {
    /**
     * Стандартный стиль линии
     */
    protected static final BasicStroke DEFAULT_STROKE = new BasicStroke(3,
BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND);
    /**
     * Отступ между дугой и нодой
     */
    protected static final double ARROW_OFFSET = 10f;
    /**
     * Ширина крыльев дуги
     */
    protected static final double ARROW_WIDTH = 5f;
    /**
     * Длина тела дуги
     */
    protected static final double ARROW_LENGTH = 15f;
    /**
     * Расстояние, после которого у дуги перестаёт отображаться тело
     */
    protected static final double EPSILON = 0.0000000001d;
    /**
     * Шрифт для отображения имён нод
     */
    protected static final Font font = new Font(Font.DIALOG, Font.BOLD, 24);
    /**
     * Шрифт для отображения времени выхода из ноды
     */
    protected static final Font fontAdditional = new Font(Font.DIALOG,
Font.ITALIC, 12);

    protected Point offset = new Point();
    protected Point scale = new Point(1, 1);
    protected Graph graph = new Graph();

    /**
     * Заменяет текущий граф на копию переданного графа
     *
     * @param graph граф, который требуется визуализировать
     */
    public void setGraphCopy(Graph graph) {
        this.graph = new Graph(graph);
        repaint();
    }

    /**
     * @return копия текущего графа
     */
    public Graph getGraphCopy() {
        return new Graph(graph);
    }

    public Point canvasToGraphSpace(int x, int y) {
        //return new Point((int) (x - offset.x * Math.pow(2, scale.x)), (int) (y
- offset.y * Math.pow(2, scale.y)));
        return new Point(x - offset.x, y - offset.y);
    }

    public Point graphToCanvasSpace(int x, int y) {
        //return new Point((int) (x + offset.x * Math.pow(2, scale.x)), (int) (y
+ offset.y * Math.pow(2, scale.y)));
        return new Point(x + offset.x, y + offset.y);
```

```
    }

    /**
     * Отрисовка графа через {@link Graphics2D}
     */
    @Override
    public void paint(Graphics g) {
        Graphics2D g2d = (Graphics2D) g;
        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);
        g2d.setRenderingHint(RenderingHints.KEY_RENDERING,
RenderingHints.VALUE_RENDER_QUALITY);
        g2d.setRenderingHint(RenderingHints.KEY_TEXT_ANTIALIASING,
RenderingHints.VALUE_TEXT_ANTIALIAS_ON);
        g2d.setColor(UIManager.getColor("Panel.background"));
        //g2d.setColor(new Color(245, 245, 245));
        g2d.fillRect(0, 0, getWidth(), getHeight());
        displayMap(g2d);
        for (Edge edge : graph.getEdges()) {
            displayEdge(g2d, edge);
        }
        for (Node node : graph.getNodes()) {
            displayNode(g2d, node);
        }
    }

    /**
     * TODO отрисовка миникарты
     */
    protected void displayMap(Graphics2D g) {
        g.setColor(UIManager.getColor("Panel.foreground"));
        g.fillRect(0, 0, 200, 200);
        g.setColor(UIManager.getColor("Panel.background"));
        int viewX = -offset.x / 25;
        int viewY = -offset.y / 25;
        g.drawRect(viewX, viewY, getWidth() / 25, getHeight() / 25);

    }

    /**
     * Позволяет настроить стиль тела дуги в {@link GraphEditor}
     */
    protected void decorateEdgeBody(Graphics2D g, Edge edge) {
        if(edge.getColor() != null){
            g.setColor(edge.getColor());
        } else {
            g.setColor(Color.BLACK);
        }
        if(edge.getStroke() != 0){
            g.setStroke(new BasicStroke(DEFAULT_STROKE.getLineWidth()+
(edge.getStroke()-5)/2));
        }else {
            g.setStroke(DEFAULT_STROKE);
        }
    }

    /**
     * Позволяет настроить стиль крыльев дуги в {@link GraphEditor}
     */
    protected void decorateEdgeArrow(Graphics2D g, Edge edge) {
        if(edge.getColor() != null){
            g.setColor(edge.getColor());
        } else {
```

```
            g.setColor(Color.BLACK);
        }
        if(edge.getStroke() != 0){
            g.setStroke(new BasicStroke(DEFAULT_STROKE.getLineWidth()+
(edge.getStroke()-5)/2));
        }else {
            g.setStroke(DEFAULT_STROKE);
        }
    }

    private void displayEdge(Graphics2D g, Edge edge) {
        Node source = graph.getNode(edge.getSource());
        Node target = graph.getNode(edge.getTarget());
        int dx = target.getPosition().x - source.getPosition().x;
        int dy = target.getPosition().y - source.getPosition().y;
        double distance = Math.sqrt(dx * dx + dy * dy);
        double vx = dx / distance;
        double vy = dy / distance;
        if (distance > EPSILON) {
            double startOffset = ARROW_OFFSET + source.getRadius();
            double endOffset = ARROW_OFFSET + target.getRadius();
            if (distance > startOffset + endOffset) {
                int startX = (int) Math.round(source.getPosition().x + vx *
startOffset);
                int startY = (int) Math.round(source.getPosition().y + vy *
startOffset);
                int endX = (int) Math.round(target.getPosition().x - vx *
endOffset);
                int endY = (int) Math.round(target.getPosition().y - vy *
endOffset);
                int leftX = (int) Math.round(endX - vy * ARROW_WIDTH - vx *
ARROW_LENGTH);
                int leftY = (int) Math.round(endY + vx * ARROW_WIDTH - vy *
ARROW_LENGTH);
                int rightX = (int) Math.round(endX + vy * ARROW_WIDTH - vx *
ARROW_LENGTH);
                int rightY = (int) Math.round(endY - vx * ARROW_WIDTH - vy *
ARROW_LENGTH);
                Point start = graphToCanvasSpace(startX, startY);
                Point end = graphToCanvasSpace(endX, endY);
                Point left = graphToCanvasSpace(leftX, leftY);
                Point right = graphToCanvasSpace(rightX, rightY);

                decorateEdgeBody(g, edge);
                g.drawLine(start.x, start.y, end.x, end.y); // Тело дуги
                decorateEdgeArrow(g, edge);
                g.drawLine(left.x, left.y, end.x, end.y); // Левое крыло стрелки
                g.drawLine(end.x, end.y, right.x, right.y); // Правое крыло
стрелки
            } else {
                int endX = (int) Math.round((target.getPosition().x +
source.getPosition().x) * 0.5d);
                int endY = (int) Math.round((target.getPosition().y +
source.getPosition().y) * 0.5d);
                int leftX = (int) Math.round(endX - vy * ARROW_WIDTH - vx *
ARROW_LENGTH);
                int leftY = (int) Math.round(endY + vx * ARROW_WIDTH - vy *
ARROW_LENGTH);
                int rightX = (int) Math.round(endX + vy * ARROW_WIDTH - vx *
ARROW_LENGTH);
                int rightY = (int) Math.round(endY - vx * ARROW_WIDTH - vy *
ARROW_LENGTH);
                Point end = graphToCanvasSpace(endX, endY);
```

```
                Point left = graphToCanvasSpace(leftX, leftY);
                Point right = graphToCanvasSpace(rightX, rightY);

                decorateEdgeArrow(g, edge);
                g.drawLine(left.x, left.y, end.x, end.y); // Левое крыло стрелки
                g.drawLine(end.x, end.y, right.x, right.y); // Правое крыло
стрелки
            }
        }
    }


    /**
     * Позволяет настроить стиль заливки ноды в {@link GraphEditor}
     */
    protected void decorateNodeInner(Graphics2D g, Node node) {
        if(node.getColor()!=null){
            g.setColor(node.getColor());
        }else {
            g.setColor(Color.GRAY);
        }
        g.setStroke(DEFAULT_STROKE);
    }

    /**
     * Позволяет настроить стиль обводки ноды в {@link GraphEditor}
     */
    protected void decorateNodeOuter(Graphics2D g, Node node) {
        g.setColor(Color.BLACK);
        g.setStroke(DEFAULT_STROKE);
    }

    /**
     * Позволяет настроить стиль текста внутри ноды в {@link GraphEditor}
     */
    protected void decorateNodeText(Graphics2D g, Node node) {
        g.setFont(font);
        g.setColor(Color.BLACK);
        g.setStroke(DEFAULT_STROKE);
    }

    protected void displayNode(Graphics2D g, Node node) {
        int radius = node.getRadius();
        int diameter = radius * 2;
        Polygon hex = createHexagon(node.getPosition(), radius);

        decorateNodeInner(g, node);
        //g.fillOval(node.getPosition().x - radius, node.getPosition().y -
radius, diameter, diameter);
        //TODO
        g.fillOval(node.getPosition().x / 25, node.getPosition().y / 25, 3, 3);
        g.fillPolygon(hex);
        decorateNodeOuter(g, node);
        //g.drawOval(node.getPosition().x - radius, node.getPosition().y -
radius, diameter, diameter);
        g.drawPolygon(hex);
        decorateNodeText(g, node);
        FontMetrics fm = g.getFontMetrics();
        int tx = node.getPosition().x - fm.stringWidth(node.getName()) / 2;
        int ty = node.getPosition().y - fm.getHeight() / 2 + fm.getAscent();
        Point tPosition = graphToCanvasSpace(tx, ty);
        g.drawString(node.getName(), tPosition.x, tPosition.y);
    }
```

```
    private Polygon createHexagon(Point position, int radius) {
        Polygon polygon = new Polygon();
        for (int i = 0; i < 6; i++) {
            int x = (int) (position.x + radius * Math.cos(i * 2 * Math.PI /
6D));
            int y = (int) (position.y + radius * Math.sin(i * 2 * Math.PI /
6D));
            Point pos = graphToCanvasSpace(x, y);
            polygon.addPoint(pos.x, pos.y);
        }
        return polygon;
    }
}
```

Файл GraphPopupMenuNode.java

```
package ru.eltech.view;

import javax.swing.*;

/**
 * Контекстное меню при клике ПКМ*2 на {@link ru.eltech.logic.Node}
 */
public final class GraphPopupMenuNode extends JPopupMenu {
    public GraphPopupMenuNode(GraphEditor graphEditor, Integer id) {
        JMenuItem removeNodeMenuItem = new JMenuItem("Удалить узел");
        removeNodeMenuItem.addActionListener((action) ->
graphEditor.removeNode(id));
        add(removeNodeMenuItem);
        addSeparator();
        JMenuItem addEdgeMenuItem = new JMenuItem("Создать дугу");
        addEdgeMenuItem.addActionListener((action) ->
graphEditor.initializeAddEdge(id));
        //add(addEdgeMenuItem); TODO
        addSeparator();
        JMenuItem changeNodeRadiusMenuItem = new JMenuItem("Изменить радиус
узла");
        changeNodeRadiusMenuItem.addActionListener((action) ->
graphEditor.changeNodeRadius(id));
        add(changeNodeRadiusMenuItem);
        JMenuItem changeNodeColorMenuItem = new JMenuItem("Изменить цвет узла");
        changeNodeColorMenuItem.addActionListener((action) ->
graphEditor.changeNodeColor(id));
        add(changeNodeColorMenuItem);
        JMenuItem changeTextMenuItem = new JMenuItem("Изменить имя узла");
        changeTextMenuItem.addActionListener((action) ->
graphEditor.changeNodeText(id));
        add(changeTextMenuItem);
    }
}
```

Файл GraphPopupMenuEmpty.java

```
package ru.eltech.view;

import javax.swing.*;

/**
 * Контекстное меню при клике ПКМ*2 на пустую область внутри {@link GraphEditor}
 */
public final class GraphPopupMenuEmpty extends JPopupMenu {
    private int x;
    private int y;

    public GraphPopupMenuEmpty(GraphEditor graphEditor, int x, int y) {
```

```
        this.x = x;
        this.y = y;
        JMenuItem newNodeMenuItem = new JMenuItem("Создать узел");
        newNodeMenuItem.addActionListener((action) ->
graphEditor.createNewNode(x, y));
        add(newNodeMenuItem);
        addSeparator();
        JMenuItem clearMenuItem = new JMenuItem("Очистить граф");
        clearMenuItem.addActionListener((action) -> graphEditor.clearGraph());
        add(clearMenuItem);
        JMenuItem scrollDownMenuItem = new JMenuItem("Расширить холст вниз");
        //scrollDownMenuItem.addActionListener((action) ->
graphEditor.scrollDown(100)); TODO remove
        add(scrollDownMenuItem);
        JMenuItem scrollRightMenuItem = new JMenuItem("Расширить холст вправо");
        //scrollRightMenuItem.addActionListener((action) ->
graphEditor.scrollRight(100)); TODO remove
        add(scrollRightMenuItem);
    }
}
```

Файл GraphPopupMenuEdge.java

```
package ru.eltech.view;

import javax.swing.*;

/**
 * Контекстное меню при клике ПКМ*2 на пустую область внутри {@link GraphEditor}
 */
public final class GraphPopupMenuEmpty extends JPopupMenu {
    private int x;
    private int y;

    public GraphPopupMenuEmpty(GraphEditor graphEditor, int x, int y) {
        this.x = x;
        this.y = y;
        JMenuItem newNodeMenuItem = new JMenuItem("Создать узел");
        newNodeMenuItem.addActionListener((action) ->
graphEditor.createNewNode(x, y));
        add(newNodeMenuItem);
        addSeparator();
        JMenuItem clearMenuItem = new JMenuItem("Очистить граф");
        clearMenuItem.addActionListener((action) -> graphEditor.clearGraph());
        add(clearMenuItem);
        JMenuItem scrollDownMenuItem = new JMenuItem("Расширить холст вниз");
        //scrollDownMenuItem.addActionListener((action) ->
graphEditor.scrollDown(100)); TODO remove
        add(scrollDownMenuItem);
        JMenuItem scrollRightMenuItem = new JMenuItem("Расширить холст вправо");
        //scrollRightMenuItem.addActionListener((action) ->
graphEditor.scrollRight(100)); TODO remove
        add(scrollRightMenuItem);
    }
}
```

Файл GraphPlayerToolBar.java

```
package ru.eltech.view;

import ru.eltech.logic.Algorithm;
import ru.eltech.logic.Graph;
import ru.eltech.logic.GraphPlayer;

import javax.swing.*;
import javax.swing.event.ChangeEvent;
```

```
import javax.swing.event.ChangeListener;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import static ru.eltech.logic.GraphPlayer.State.*;

/**
 * Панель управления анимацией, в отдельном классе, так как IntelliJ не умеет
нормально работать с нажатиями на кнопки
 */
public final class GraphPlayerToolBar extends JToolBar implements
ActionListener, ChangeListener {
    private MainWindow parent;

    private final JPanel upperPanel = new JPanel();
    private final JCheckBox toolBarSetReverseCheckBox = new JCheckBox("Проводить
инвертирование графа мгновенно", true);
    private final JButton toolBarAutoButton = new JButton(new
ImageIcon(getClass().getResource("/resources/icons/repeat-24px.png")));
    private final JButton toolBarPlayButton = new JButton(new
ImageIcon(getClass().getResource("/resources/icons/play_arrow-24px.png")));
    private final JButton toolBarPauseButton = new JButton(new
ImageIcon(getClass().getResource("/resources/icons/pause-24px.png")));
    private final JButton toolBarStepBackwardButton = new JButton(new
ImageIcon(getClass().getResource("/resources/icons/skip_previous-24px.png")));
    private final JButton toolBarStepForwardButton = new JButton(new
ImageIcon(getClass().getResource("/resources/icons/skip_next-24px.png")));
    private final JButton toolBarStopButton = new JButton(new
ImageIcon(getClass().getResource("/resources/icons/stop-24px.png")));
    private final Separator separator = new Separator();
    private final JLabel speedLabel = new JLabel("Delay: ");
    private final JSlider toolBarSpeedSlider = new JSlider(1, 1000, 500);
    private final JPanel bottomPanel = new JPanel();
    private final JLabel progressLabel = new JLabel("Progress: ");
    private final JProgressBar toolBarProgressBar = new JProgressBar(1, 100);

    public GraphPlayerToolBar() {
        toolBarSetReverseCheckBox.addActionListener(this);
        toolBarAutoButton.addActionListener(this);
        toolBarPlayButton.addActionListener(this);
        toolBarPlayButton.setMnemonic('Q');
        toolBarPlayButton.setToolTipText("Auto (Alt+Q)");
        toolBarPauseButton.addActionListener(this);
        toolBarPauseButton.setMnemonic('Q');
        toolBarPauseButton.setToolTipText("Pause (Alt+Q)");
        toolBarStepBackwardButton.addActionListener(this);
        toolBarStepBackwardButton.setMnemonic('W');
        toolBarStepBackwardButton.setToolTipText("Step Backward (Alt+W)");
        toolBarStepForwardButton.addActionListener(this);
        toolBarStepForwardButton.setMnemonic('E');
        toolBarStepForwardButton.setToolTipText("Step Forward (Alt+E)");
        toolBarStopButton.addActionListener(this);
        toolBarStopButton.setMnemonic('R');
        toolBarStopButton.setToolTipText("Stop (Alt+R)");

        setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
        setOrientation(SwingConstants.VERTICAL);
        add(upperPanel);
        //upperPanel.add(toolBarAutoButton);
        upperPanel.add(toolBarSetReverseCheckBox);
        upperPanel.add(toolBarPlayButton);
        upperPanel.add(toolBarPauseButton);
        upperPanel.add(toolBarStepBackwardButton);
```

```
        upperPanel.add(toolBarStepForwardButton);
        upperPanel.add(toolBarStopButton);
        upperPanel.add(separator);
        upperPanel.add(speedLabel);
        toolBarSpeedSlider.setValue(500);
        upperPanel.add(toolBarSpeedSlider);
        toolBarSpeedSlider.addChangeListener(this);

        add(bottomPanel);
        bottomPanel.add(progressLabel);
        bottomPanel.add(toolBarProgressBar);

        //toolBar1.setMinimumSize(new Dimension(613, 64));

        //toolBar1.setPreferredSize(new Dimension(777, 100));
        //panel1.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));
        //panel1.setPreferredSize(new Dimension(10, 50));
        //toolBarSpeedSlider.setMaximumSize(new Dimension(300, 31));

        //panel2.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));
        //panel2.setMinimumSize(new Dimension(70, 10));
        //panel2.setPreferredSize(new Dimension(206, 10));
    }

    public void setParent(MainWindow parent) {
        this.parent = parent;
        playerChanged(parent.getGraphPlayer());
    }

    // region FROM TOOLBAR TO PLAYER

    @Override
    public void actionPerformed(ActionEvent e) {
        GraphPlayer graphPlayer = parent.getGraphPlayer();
        Object eSource = e.getSource();
        if (eSource == toolBarSetReverseCheckBox) {
            JCheckBox currentCheck = (JCheckBox) eSource;
            parent.setImmediateReverse(currentCheck.isSelected());
        } else if (eSource == toolBarAutoButton) {
            //startVisualizing();
        } else if (eSource == toolBarPlayButton) {
            //if (graphPlayer.getState() == GraphPlayer.State.Empty)
startVisualizing();
            toolBarSetReverseCheckBox.setEnabled(false);
            toolBarSpeedSlider.setEnabled(false);
            graphPlayer.setState(Play);
        } else if (eSource == toolBarPauseButton) {
            graphPlayer.setState(Pause);
            toolBarSpeedSlider.setEnabled(true);
        } else if (eSource == toolBarStepBackwardButton) {
            graphPlayer.stepBackward();
        } else if (eSource == toolBarStepForwardButton) {
            graphPlayer.stepForward();
        } else if (eSource == toolBarStopButton) {
            if (graphPlayer.getState() == Stop) parent.clearLog();
            else {
                toolBarSetReverseCheckBox.setEnabled(true);
                toolBarSpeedSlider.setEnabled(true);
                graphPlayer.setState(Stop);
            }
        }
    }
```

```
    @Override
    public void stateChanged(ChangeEvent e) {
        GraphPlayer graphPlayer = parent.getGraphPlayer();
        JSlider slider = (JSlider) e.getSource();
        if (slider == toolBarSpeedSlider) {
            graphPlayer.setDelay(slider.getValue());
        }
    }

    // endregion

    // region FROM PLAYER TO TOOLBAR

    public void playerChanged(GraphPlayer graphPlayer) {
        if (graphPlayer.sliderChanged) {
            graphPlayer.sliderChanged = false;
            toolBarSpeedSlider.setValue(graphPlayer.getDelay());
            return;
        }
        toolBarSetReverseCheckBox.setVisible(true);
        toolBarPlayButton.setVisible(graphPlayer.getState() != Play);
        toolBarPauseButton.setVisible(graphPlayer.getState() == Play);
        toolBarStepBackwardButton.setEnabled(graphPlayer.getState() != Stop);
        toolBarStepForwardButton.setEnabled(graphPlayer.getState() != Stop);
        toolBarSpeedSlider.setValue(graphPlayer.getDelay());
        toolBarProgressBar.setMaximum(graphPlayer.getFrameList() != null ?
graphPlayer.getFrameList().count() - 1 : 0);
        toolBarProgressBar.setValue(graphPlayer.getCurrentFrame());

        switch (graphPlayer.getState()) {
            case Play:
                parent.playerPlaying();
                if (graphPlayer.getFrameList() != null) {

parent.playerVisualizing(graphPlayer.getFrameList().get(graphPlayer.getCurrentFr
ame()));
                }
                break;
            case Pause:
                parent.playerPausing();
                if (graphPlayer.getFrameList() != null) {
                    //костыль чтобы на паузе не дублировался лог
                    Graph graph =
graphPlayer.getFrameList().get(graphPlayer.getCurrentFrame());
                    if (graphPlayer.stepBackwardInPause ||
graphPlayer.stepForwardInPause) {
                        graphPlayer.stepForwardInPause = false;
                        graphPlayer.stepBackwardInPause = false;
                    } else {
                        graph.state = "";
                    }
                    parent.playerVisualizing(graph);
                }
                break;
            case Stop:
                parent.playerStopping();
                break;
        }
    }
    // endregion
}
```

Файл GraphEditorMouseWheelListener.java

```
package ru.eltech.view;
```

```
import java.awt.*;
import java.awt.event.MouseWheelEvent;
import java.awt.event.MouseWheelListener;

public class GraphEditorMouseWheelListener implements MouseWheelListener {

    @Override
    public void mouseWheelMoved(MouseWheelEvent e) {
        GraphEditor graphEditor = (GraphEditor) e.getSource();
        Point pos = graphEditor.canvasToGraphSpace(e.getX(), e.getY());
        graphEditor.zoom(pos.x, pos.y, -e.getWheelRotation());
    }
}
```

Файл GraphEditorMouseMotionListener.java

```
package ru.eltech.view;

import java.awt.*;
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionListener;

/**
 * Логика движения мыши в {@link GraphEditor}
 */
public final class GraphEditorMouseMotionListener implements MouseMotionListener
{
    @Override
    public void mouseDragged(MouseEvent e) {
        GraphEditor graphEditor = (GraphEditor) e.getSource();
        Point pos = graphEditor.canvasToGraphSpace(e.getX(), e.getY());
        graphEditor.drag(e.getX(), e.getY(), pos.x, pos.y);
        graphEditor.connecting(e.getX(), e.getY());
        setMouseCursor(graphEditor, e);
    }

    @Override
    public void mouseMoved(MouseEvent e) {
        GraphEditor graphEditor = (GraphEditor) e.getSource();
        setMouseCursor(graphEditor, e);
    }

    private void setMouseCursor(GraphEditor graph, MouseEvent e) {
        Point pos = graph.canvasToGraphSpace(e.getX(), e.getY());

        if (graph.isReadOnly)
graph.setCursor(Cursor.getPredefinedCursor(Cursor.CROSSHAIR_CURSOR));
        else if (graph.findNode(pos.x, pos.y) != null)
graph.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
        else if (graph.findEdge(pos.x, pos.y) != null)
graph.setCursor(Cursor.getPredefinedCursor(Cursor.CROSSHAIR_CURSOR));
        else if (graph.isMoving())
graph.setCursor(Cursor.getPredefinedCursor(Cursor.CROSSHAIR_CURSOR));
        else if (graph.isConnecting())
graph.setCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));
        else if (e.getButton() == MouseEvent.BUTTON1)
graph.setCursor(Cursor.getPredefinedCursor(Cursor.MOVE_CURSOR));
        else graph.setCursor(Cursor.getPredefinedCursor(Cursor.DEFAULT_CURSOR));

    }
}
```

Файл GraphEditorMouseListener.java

```
package ru.eltech.view;
```

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;

/**
 * Логика нажатия клавиш мыши в {@link GraphEditor}
 */
public final class GraphEditorMouseListener implements MouseListener {
    @Override
    public void mouseClicked(MouseEvent e) {
        GraphEditor graphEditor = (GraphEditor) e.getSource();
        Point pos = graphEditor.canvasToGraphSpace(e.getX(), e.getY());
        if (e.getClickCount() == 2) {
            if (e.getButton() == MouseEvent.BUTTON1) {
                if (graphEditor.hasSelected()) graphEditor.destroySelected();
                else graphEditor.createNode(pos.x, pos.y);
            }
        }
        if (e.getButton() == MouseEvent.BUTTON3 && !graphEditor.isReadOnly) {
            JPopupMenu menu = createPopupMenu(graphEditor, pos.x, pos.y);
            menu.show(graphEditor, e.getX(), e.getY());
        }
        e.consume();
    }

    private JPopupMenu createPopupMenu(GraphEditor graph, int x, int y) {
        Integer foundNode = graph.findNode(x, y);
        if (foundNode != null) {
            return new GraphPopupMenuNode(graph, foundNode);
        }
        Integer foundEdge = graph.findEdge(x, y);
        if (foundEdge != null) {
            return new GraphPopupMenuEdge(graph, foundEdge);
        }
        return new GraphPopupMenuEmpty(graph, x, y);
    }

    @Override
    public void mouseEntered(MouseEvent e) {
    }

    @Override
    public void mouseExited(MouseEvent e) {
        GraphEditor graphEditor = (GraphEditor) e.getSource();
        Point pos = graphEditor.canvasToGraphSpace(e.getX(), e.getY());
        graphEditor.endDrag(e.getX(), e.getY(), pos.x, pos.y);
        if (e.getClickCount() == 2) {
            //graphEditor.scrollAreaAroundPoint(pos.x, pos.y, 100);
        }
    }

    @Override
    public void mousePressed(MouseEvent e) {
        GraphEditor graphEditor = (GraphEditor) e.getSource();
        Point pos = graphEditor.canvasToGraphSpace(e.getX(), e.getY());
        if (e.getButton() == MouseEvent.BUTTON1) {
            graphEditor.select(pos.x, pos.y);
            graphEditor.startDrag(e.getX(), e.getY(), pos.x, pos.y);
        } else if (e.getButton() == MouseEvent.BUTTON3) {
            graphEditor.startConnecting(pos.x, pos.y);
        }
```

```
    }

    @Override
    public void mouseReleased(MouseEvent e) {
        GraphEditor graphEditor = (GraphEditor) e.getSource();
        Point pos = graphEditor.canvasToGraphSpace(e.getX(), e.getY());
        if (e.getButton() == MouseEvent.BUTTON1) {
            graphEditor.endDrag(e.getX(), e.getY(), pos.x, pos.y);
            //если создали новую ноду
            if (e.getClickCount() == 2) {
                //graphEditor.scrollAreaAroundPoint(e.getX(), e.getY(), 100);
            }
        } else if (e.getButton() == MouseEvent.BUTTON3) {
            graphEditor.endConnecting(pos.x, pos.y, true, true);
        }
    }
}
```

Файл GraphEditorKeyListener.java

```
package ru.eltech.view;

import java.awt.*;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;

/**
 * Логика нажатия клавиш клавиатуры в {@link GraphEditor}
 */
public final class GraphEditorKeyListener implements KeyListener {
    @Override
    public void keyTyped(KeyEvent e) {
        //
    }

    @Override
    public void keyPressed(KeyEvent e) {
        GraphEditor graphEditor = (GraphEditor) e.getSource();
        switch (e.getKeyCode()) {
            case KeyEvent.VK_LEFT:
                graphEditor.moveGraphStep(e.isShiftDown() ? -10 : -1, 0);
                break;
            case KeyEvent.VK_RIGHT:
                graphEditor.moveGraphStep(e.isShiftDown() ? 10 : 1, 0);
                break;
            case KeyEvent.VK_UP:
                graphEditor.moveGraphStep(0, e.isShiftDown() ? -10 : -1);
                break;
            case KeyEvent.VK_DOWN:
                graphEditor.moveGraphStep(0, e.isShiftDown() ? 10 : 1);
                break;
            case KeyEvent.VK_DELETE:
                graphEditor.destroySelected();
                break;
            case KeyEvent.VK_R:
                graphEditor.setColor(Color.RED);
                break;
            case KeyEvent.VK_G:
                graphEditor.setColor(Color.GREEN);
                break;
            case KeyEvent.VK_B:
                graphEditor.setColor(Color.BLUE);
                break;
            case KeyEvent.VK_EQUALS:
                graphEditor.changeSize(1);
```

```
                break;
            case KeyEvent.VK_MINUS:
                graphEditor.changeSize(-1);
                break;
        }
    }

    @Override
    public void keyReleased(KeyEvent e) {
        //
    }
}
```

Файл GraphEditor.java

```
package ru.eltech.view;

import ru.eltech.logic.Edge;
import ru.eltech.logic.Graph;
import ru.eltech.logic.Node;

import javax.swing.*;
import java.awt.*;

/**
 * Класс, отвечающий за логику редактирования графа
 *
 * @author сахар
 */
public final class GraphEditor extends GraphVisualizer {
    /**
     * Ширина коллайдера дуги
     */
    private static final int EDGE_DRAG_DISTANCE = 20;
    /**
     * Стиль выделенной дуги
     */
    private static final BasicStroke SELECTED_STROKE = new BasicStroke(6,
BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND);
    /**
     * Стиль создаваемой дуги
     */
    private static final BasicStroke CONNECTING_STROKE = new BasicStroke(6,
BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND, 10, new float[]{5, 10}, 0);

    public int lastCreatedNodeRadius = 0;
    public boolean isReadOnly = false;
    public boolean isModified = false;
    private final Point draggingLast = new Point();
    private final Point draggingLastCanvas = new Point();
    private final Point connectingLast = new Point();
    private Node draggingNode;
    private Node selectedNode;
    private Edge draggingEdge;
    private Edge selectedEdge;
    private Node connectingSourceNode;
    private boolean draggingGraph = false;

    public GraphEditor() {
        addMouseListener(new GraphEditorMouseListener());
        addMouseMotionListener(new GraphEditorMouseMotionListener());
        addMouseWheelListener(new GraphEditorMouseWheelListener());
        addKeyListener(new GraphEditorKeyListener());
    }
```

```
@Override
public void setGraphCopy(Graph renderGraph) {
    draggingNode = null;
    selectedNode = null;
    draggingEdge = null;
    selectedEdge = null;
    connectingSourceNode = null;
    draggingGraph = false;
    super.setGraphCopy(renderGraph);
    isModified = false;
}

/**
 * @return Имеет ли граф выделенные ноды/дуги
 */
public boolean hasSelected() {
    return selectedNode != null || selectedEdge != null;
}

/**
 * @return Имеет ли граф создаваемую дугу
 */
public boolean isConnecting() {
    return connectingSourceNode != null;
}

public boolean isMoving() {
    return draggingGraph;
}

/**
 * Выделяет ноду или дугу, нода в приоритете
 *
 * @return выделилось ли что-либо
 */
@SuppressWarnings("UnusedReturnValue")
public boolean select(int x, int y) {
    if (isReadOnly) return false;
    if (innerSelectNode(x, y)) {
        selectedEdge = null;
        repaint();
        return true;
    } else if (selectedNode == null && innerSelectEdge(x, y)) {
        repaint();
        return true;
    }
    return false;
}

private boolean innerSelectNode(int x, int y) {
    Node current = selectedNode;
    selectedNode = innerFindNode(x, y);
    return current != selectedNode;
}

private boolean innerSelectEdge(int x, int y) {
    Edge current = selectedEdge;
    selectedEdge = innerFindEdge(x, y);
    return current != selectedEdge;
}

/**
 * Масштабирует визуализацию графа
```

```
     */
    public void zoom(int x, int y, int change) {
        scale.translate(change, change);
        repaint();
        // TODO relative scale
    }

    /**
     * Инициирует перемещение выделенного объекта
     */
    public void startDrag(int canvasX, int canvasY, int x, int y) {
        draggingLast.move(x, y);
        draggingLastCanvas.move(canvasX, canvasY);
        draggingNode = selectedNode;
        draggingEdge = selectedEdge;
        if (isReadOnly || draggingNode == null && draggingEdge == null)
draggingGraph = true;
    }

    /**
     * Перемещает выделенный объект, нода в приоритете
     */
    public void drag(int canvasX, int canvasY, int x, int y) {
        int mouseDX = x - draggingLast.x;
        int mouseDY = y - draggingLast.y;

        if (draggingGraph || isReadOnly) {
            offset.translate(canvasX - draggingLastCanvas.x, canvasY -
draggingLastCanvas.y);
            repaint();
        } else if (draggingNode != null) {
            draggingNode.getPosition().translate(mouseDX, mouseDY);
            repaint();
        } else if (draggingEdge != null) {
            Node fromNode = graph.getNode(draggingEdge.getSource());
            Node toNode = graph.getNode(draggingEdge.getTarget());
            fromNode.getPosition().translate(mouseDX, mouseDY);
            toNode.getPosition().translate(mouseDX, mouseDY);
            repaint();
        }

        draggingLastCanvas.move(canvasX, canvasY);
        draggingLast.move(x, y);
    }

    /**
     * Завершает процесс перемещения объекта
     */
    public void endDrag(int canvasX, int canvasY, int x, int y) {
        draggingNode = null;
        draggingEdge = null;
        draggingGraph = false;
    }

    /**
     * Начало создания дуги. От ноды под координаитой x,y до координат мыши
     */
    public void startConnecting(int x, int y) {
        if (isReadOnly) return;
        connectingSourceNode = innerFindNode(x, y);
        connectingLast.move(x, y);
    }
```

```
    /**
     * Обновляет позицию создаваемой дуги, если она существует
     */
    public void connecting(int x, int y) {
        if (connectingSourceNode == null || isReadOnly) return;
        connectingLast.move(x, y);
        repaint();
    }

    /**
     * Завершает создание дуги в точке x,y
     *
     * @param x                x
     * @param y                y
     * @param willCreateNode   будет ли создаваться новая нода, если на
координатах x,y нет ноды
     * @param willDestroyClone будет ли удаляться дуга, если происходит попытка
создать совпадающую с ней дугу
     */
    public void endConnecting(int x, int y, boolean willCreateNode, boolean
willDestroyClone) {
        if (connectingSourceNode == null || isReadOnly) return;
        Node node = innerFindNode(x, y);
        if (node != connectingSourceNode) {
            if (node != null) {
                Edge currentEdge = graph.getEdge(connectingSourceNode, node,
true);
                if (currentEdge != null) {
                    if (willDestroyClone) graph.destroyEdge(currentEdge);
                } else
                    graph.createEdge(connectingSourceNode, node);
            } else if (willCreateNode) {
                Node newNode = graph.createNode(x, y);
                graph.createEdge(connectingSourceNode, newNode);
            }
        }
        connectingSourceNode = null;
        repaint();
    }

    /**
     * Создаёт ноду на координатах x,y
     */
    public void createNode(int x, int y) {
        if (isReadOnly) return;
        graph.createNode(x, y);
        repaint();
    }

    /**
     * Удаляет выделенный объект
     */
    public void destroySelected() {
        if (isReadOnly) return;
        if (innerDestroySelectedNode() | innerDestroySelectedEdge()) repaint();
    }

    private boolean innerDestroySelectedNode() {
        if (selectedNode == null) return false;
        graph.destroyNode(selectedNode);
        if (draggingNode == selectedNode) draggingNode = null;
        if (connectingSourceNode == selectedNode) connectingSourceNode = null;
```

```
        if (selectedEdge != null && !graph.containsEdge(selectedEdge))
selectedEdge = null;
        selectedNode = null;
        return true;
    }

    private boolean innerDestroySelectedEdge() {
        if (selectedEdge == null) return false;
        graph.destroyEdge(selectedEdge);
        if (draggingEdge == selectedEdge) draggingEdge = null;
        selectedEdge = null;
        return true;
    }

    /**
     * Поиск ноды по координатам x,y
     *
     * @return id найденной ноды, иначе null
     */
    public Integer findNode(int x, int y) {
        Node node = innerFindNode(x, y);
        return node != null ? node.id : null;
    }

    private Node innerFindNode(int x, int y) {
        Node foundNode = null;
        int foundSqrDist = 0;
        for (Node node : graph.getNodes()) {
            int sqrDist = (int) Point.distanceSq(node.getPosition().x,
node.getPosition().y, x, y);
            if (sqrDist <= node.getRadius() * node.getRadius() && (foundNode ==
null || foundSqrDist > sqrDist)) {
                foundNode = node;
                foundSqrDist = sqrDist;
            }
        }
        return foundNode;
    }

    /**
     * Поиск дуги по координатам x,y
     *
     * @return id найденной дуги, иначе null
     */
    public Integer findEdge(int x, int y) {
        Edge edge = innerFindEdge(x, y);
        return edge != null ? edge.id : null;
    }

    private Edge innerFindEdge(int x, int y) {
        Edge foundEdge = null;
        double foundDist = 0;

        for (Edge edge : graph.getEdges()) {
            Node source = graph.getNode(edge.getSource());
            Node target = graph.getNode(edge.getTarget());
            int dx = target.getX() - source.getX();
            int dy = target.getY() - source.getY();
            double dist = Math.sqrt(dx * dx + dy * dy);
            double vx = dx / dist;
            double vy = dy / dist;
            double lx = vx * (x - source.getX()) + vy * (y - source.getY());
            if (lx < 0 || lx > dist) continue;
```

```
            double ly = vy * (x - source.getX()) - vx * (y - source.getY());
            double distToLine = Math.abs(ly);
            if (distToLine > EDGE_DRAG_DISTANCE) continue;
            if (foundEdge == null || distToLine < foundDist) {
                foundEdge = edge;
                foundDist = distToLine;
            }
        }
        return foundEdge;
    }

    @Override
    public void repaint() {
        isModified = true;
        super.repaint();
    }

    @Override
    public void paint(Graphics g) {
        super.paint(g);
        Graphics2D g2d = (Graphics2D) g;
        if (connectingSourceNode != null) {
            g2d.setColor(Color.GREEN);
            g2d.setStroke(CONNECTING_STROKE);
            Point position =
graphToCanvasSpace(connectingSourceNode.getPosition().x,
connectingSourceNode.getPosition().y);
            g2d.drawLine(position.x, position.y, connectingLast.x,
connectingLast.y);
        }
        if (isModified) {
            //g2d.drawString("*", 20, 20);
        }
    }

    @Override
    protected void decorateEdgeBody(Graphics2D g, Edge edge) {
        super.decorateEdgeBody(g, edge);
        if (draggingEdge == edge) {
            g.setStroke(SELECTED_STROKE);
        }
        if (selectedEdge == edge) {
            g.setColor(Color.GREEN);
        }
        if (isReadOnly) {
            if (edge.highlighted) {
                g.setColor(Color.ORANGE);
            } else if (edge.connectsStrongComponents) {
                g.setColor(new Color(150, 150, 150));
            }
        }
    }

    @Override
    protected void decorateEdgeArrow(Graphics2D g, Edge edge) {
        super.decorateEdgeArrow(g, edge);
        if (isReadOnly) {
            if (edge.highlighted) {
                g.setColor(Color.ORANGE);
            } else if (edge.connectsStrongComponents) {
                g.setColor(new Color(150, 150, 150));
            }
        }
```

```
    }

    @Override
    protected void decorateNodeInner(Graphics2D g, Node node) {
        super.decorateNodeInner(g, node);
        if (isReadOnly) {
            if (node.visited) {
                g.setColor(Color.LIGHT_GRAY);
            }
            if (node.strongComponentId != -1) {
                g.setColor(NodeColors.colors[node.strongComponentId %
NodeColors.colors.length]);
            }
        }
    }

    @Override
    protected void decorateNodeOuter(Graphics2D g, Node node) {
        super.decorateNodeOuter(g, node);
        if (draggingNode == node) {
            g.setStroke(SELECTED_STROKE);
        }
        if (selectedNode == node) {
            g.setColor(Color.GREEN);
        }
        if (isReadOnly) {
            if (node.highlighted) {
                g.setColor(Color.ORANGE);
            }
        }
    }

    @Override
    protected void decorateNodeText(Graphics2D g, Node node) {
        super.decorateNodeText(g, node);
    }

    @Override
    protected void displayNode(Graphics2D g, Node node) {
        super.displayNode(g, node);
        if (isReadOnly) {
            g.setFont(fontAdditional);
            g.setColor(Color.BLACK);
            g.setStroke(DEFAULT_STROKE);
            FontMetrics fm = g.getFontMetrics();
            int tx = node.getPosition().x - fm.stringWidth(node.getName()) / 2;
            int ty = node.getPosition().y + fm.getHeight() / 2 + fm.getAscent();
            Point position = graphToCanvasSpace(tx, ty);
            g.drawString(Integer.toString(node.timeOut), position.x,
position.y);
        }
    }

    //region ACTIONS POPUP

    public void destroyEdge(Integer id) {
        if (isReadOnly) return;
        Edge edge = graph.getEdge(id);
        if (edge != null) graph.destroyEdge(edge);
        repaint();
    }

    public void changeEdgeStroke(Integer id) {
```

```
        if (isReadOnly) return;
        Object[] radiusValues = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
        int radius = (Integer) JOptionPane.showInputDialog(this, "Введите
толщину", "Модификация", JOptionPane.PLAIN_MESSAGE, null, radiusValues,
radiusValues[radiusValues.length / 2]);
        graph.getEdge(id).setStroke(radius);
        repaint();
    }

    public void changeEdgeColor(Integer id) {
        if (isReadOnly) return;
        Color color = JColorChooser.showDialog(this, "Модификация",
Color.BLACK);
        graph.getEdge(id).setColor(color);
        repaint();
    }

    public void createNewNode(int x, int y) {
        if (isReadOnly) return;
        graph.createNode(x, y);
        repaint();
    }

    public void clearGraph() {
        if (isReadOnly) return;
        graph.clear();
        repaint();
    }

    public void removeNode(Integer id) {
        if (isReadOnly) return;
        Node node = graph.getNode(id);
        if (node != null) graph.destroyNode(node);
        repaint();
    }

    public void initializeAddEdge(Integer id) {
        if (isReadOnly) return;
        // TODO
    }

    public void changeNodeRadius(Integer id) {
        if (isReadOnly) return;
        Object[] radiusValues = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
        int radius = (Integer) JOptionPane.showInputDialog(this, "Введите
радиус", "Модификация", JOptionPane.PLAIN_MESSAGE, null, radiusValues,
radiusValues[radiusValues.length / 2]);
        graph.getNode(id).setRadius(32+3*(radius-5));
        repaint();
    }

    public void changeNodeColor(Integer id) {
        if (isReadOnly) return;
        Color color = JColorChooser.showDialog(this, "Модификация",
Color.BLACK);
        graph.getNode(id).setColor(color);
        repaint();
    }

    public void changeNodeText(Integer id) {
        if (isReadOnly) return;
        String name = JOptionPane.showInputDialog(this, "Введите новое имя",
"Модификация", JOptionPane.QUESTION_MESSAGE);
```

```
        graph.getNode(id).setName(name);
        repaint();
    }

    //endregion

    //region ACTIONS KEYSTROKE

    public void moveGraphStep(int x, int y) {
        offset.translate(x, y);
        repaint();
    }

    public void setColor(Color color) {
        if (isReadOnly) return;
        // TODO
    }

    public void changeSize(int delta) {
        if (isReadOnly) return;
        // TODO?
    }

    //endregion
}
```

Файл Node.java

```
package ru.eltech.logic;

import java.awt.*;

/**
 * @author Samoilova Anna
 */
public final class Node {
    public final Integer id;
    private String name;
    private int radius;
    private Point position;
    private Color color = null;

    public boolean visited = false;
    public boolean highlighted = false;
    public int strongComponentId = -1;
    public int timeOut = 0;

    public Node(Integer id, int x, int y) {
        this.id = id;
        this.setName(id.toString());
        this.setRadius(32);
        this.setPosition(new Point(x, y));
    }

    private Node(Node other) {
        this.id = other.id;
        this.setName(other.getName());
        this.setRadius(other.getRadius());
        this.setPosition((Point) other.position.clone());
        this.visited = other.visited;
        this.highlighted = other.highlighted;
        this.strongComponentId = other.strongComponentId;
        this.timeOut = other.timeOut;
        this.color = other.color;
    }
```

```
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getRadius() {
        return radius;
    }

    public void setRadius(int radius) {
        this.radius = radius;
    }

    public Point getPosition() {
        return position;
    }

    public int getX() {
        return position.x;
    }

    public int getY() {
        return position.y;
    }

    public void setPosition(Point position) {
        this.position = position;
    }

    public void setX(int x) {
        this.position.x = x;
    }

    public void setY(int y) {
        this.position.y = y;
    }

    public Color getColor(){
        return color;
    }

    public void setColor(Color color){
        this.color = color;
    }

    @SuppressWarnings("MethodDoesntCallSuperMethod")
    @Override
    public Node clone() {
        return new Node(this);
    }

    public String getDescription() {
        String componentIdString = (strongComponentId != -1) ?
Integer.toString(strongComponentId) : "еще на найден";
        return "Вершина " + id + " Имя: " + name +
                " ID компоненты: " + componentIdString +
                " Позиция на экране: x = " + getX() + " | y = " + getY();
    }
}
```

Файл KosarajuAlgorithm.java

```
package ru.eltech.logic;

import ru.eltech.view.MainWindow;

import java.util.ArrayList;
import java.util.Collection;
import java.util.Collections;

public final class KosarajuAlgorithm implements Algorithm {

    private ArrayList<Node> timeOutList;
    private FrameList frames;
    private int timer = 0;
    private boolean immediateReverse = true;
    private String currentStep = "";

    public boolean isImmediateReverse() {
        return immediateReverse;
    }

    public void setImmediateReverse(boolean immediateReverse) {
        this.immediateReverse = immediateReverse;
    }

    /**
     * @param context Граф, над которым требуется выполнить алгоритм
     * @return
     * @implNote sort nodes by timeout
     * result goes to global "timeOutList"
     * frames are created by timeOut() and written down to global "frames
     */
    @Override
    public FrameList process(Graph context) {
        //вершины по времени выхода в порядке возрастания
        timeOutList = new ArrayList<>();
        //возвращаемые фреймы
        frames = new FrameList();
        frames.add(context, "");

        timer = 0;
        Collection<Node> nodes = context.getNodes();
        for (Node node : nodes) {
            if (!node.visited) {
                timeOut(node, context);
            }
        }
        Collections.reverse(timeOutList);

        //just in case
        if (timeOutList.size() != context.getNodesCount()) {
            MainWindow.log.severe("timeoutsize " + timeOutList.size() + " nodes
count " + context.getNodesCount());
        }

        clearAll(context);
        frames.add(context, "Снятие всех выделений, переход к следующему
шагу.<br>");

        reverseGraph(context, immediateReverse);

        clearHighlightedEdges(context);
```

```
        frames.add(context, "Снятие всех выделений, переход к следующему
шагу.<br>");

        int currentComponentId = 0;
        for (Node currentNode : timeOutList) {
            if (currentNode.strongComponentId == -1) {
                findComponent(currentComponentId, currentNode, context);
                ++currentComponentId;
            }
        }

        clearAll(context);
        frames.add(context, "Снятие всех выделений, переход к следующему
шагу.<br>");

        reverseGraph(context, immediateReverse);

        clearHighlightedEdges(context);
        frames.add(context, "Снятие всех выделений, завершение алгоритма.<br>" +
                "Алгоритм закончен. " + frames.count() + " итераций");

        String components = "Найдены компоненты со следующими вершинами:<br>";
        for (int i = 0; context.componentExists(i);i++){
            components += context.getNodesInComponentById(i) + "<br>";
        }
        frames.add(context, components + "<br>");
        //MainWindow.log.info("Просчет алгоритма закончен. " + frames.count() +
" итераций");
        return frames;
    }

    /**
     * @param startNode
     * @param graph
     * @implNote Первый обход dfs
     */
    private void timeOut(Node startNode, Graph graph) {
        startNode.visited = true;
        timer++;

        startNode.highlighted = true;
        currentStep += "<br>Подсчет времени выхода<br>Текущая вершина: " +
startNode.getName() + "<br>";
        frames.add(graph, currentStep);//for animation

        Collection<Edge> edgeList = graph.getEdgesFromNode(startNode);
        Node nextNode = null;
        currentStep = "";
        for (Edge currentEdge : edgeList) {

            currentEdge.highlighted = true;
            nextNode = graph.getNode(currentEdge.getTarget());
            currentStep += "Переход в вершину " + nextNode.getName() + "<br>";
            frames.add(graph, currentStep);
            currentStep = "";

            if (nextNode.visited) {
                frames.add(graph, "Вершина " + nextNode.getName() + " уже
посещена<br>");
            } else {
                currentStep += "Вершина " + nextNode.getName() + " не
посещена,&#10;&#13; рекурсивно обходим ее<br>";
                timeOut(nextNode, graph);
```

```
            }
        }
        startNode.timeOut = timer;
        currentStep = "Выход из вызова функции обхода&#10;&#13; для вершины " +
startNode.getName() + "<br>" +
                "Вершина добавлена в список &#10;&#13;для поиска компонент, ее
время выхода " +
                timer + "<br>";
        frames.add(graph, currentStep);

        timeOutList.add(startNode);
        currentStep = "";
        timer++;
    }

    /**
     * @param graph
     * @implNote Разворачивает ребра графа
     */
    private void reverseGraph(Graph graph, boolean inOneStep) {

        if (inOneStep) {
            highlightAllEdges(graph);
            frames.add(graph, "Инвертирование всех ребер");
            for (Edge current : graph.getEdges()) {
                current.invert();
            }
            frames.add(graph);
            clearHighlightedEdges(graph);
            frames.add(graph);
            return;
        }
        for (Edge current : graph.getEdges()) {
            current.highlighted = true;
            String from = graph.getNode(current.getSource()).getName();
            String to = graph.getNode(current.getTarget()).getName();
            frames.add(graph, "Инвертирование ребра из " + from + " в " + to +
"<br>");//for animation
            current.invert();
            frames.add(graph);
        }
    }

    /**
     * @param componentId
     * @param node
     * @param graph
     * @implNote Второй обход dfs
     */
    private void findComponent(int componentId, Node node, Graph graph) {

        node.strongComponentId = componentId;
        node.visited = true;
        node.highlighted = true;
        frames.add(graph,"Поиск компоненты сильной связности с id " +
componentId);

        Collection<Edge> edgeList = graph.getEdgesFromNode(node);
        for (Edge currentEdge : edgeList) {

            currentEdge.highlighted = true;
            Node nextNode = graph.getNode(currentEdge.getTarget());
            frames.add(graph, "Переход в вершину " +
```

```
                    graph.getNode(currentEdge.getTarget()).getName() + "<br>");

            if (nextNode.strongComponentId == -1) {
                frames.add(graph, "Вершина " + nextNode.getName() + " еще не в
компоненте"+
                        ". Выполняется рекурсивный вызов функции поиска
компоненты для вершины " +
                        nextNode.getName() + "<br>");
                findComponent(componentId, nextNode, graph);
//                frames.add(graph, "Найдена компонента сильной связности с id
"+nextNode.strongComponentId+
//                        "<br>В нее входят вершины " +
graph.getNodesInComponentById(nextNode.strongComponentId) + "<br>");
            } else if (nextNode.strongComponentId != node.strongComponentId) {
                currentEdge.highlighted = false;
                currentEdge.connectsStrongComponents = true;
                frames.add(graph,"Вершины " + node.getName() + " и " +
nextNode.getName() +
                        " в разных компонентах, выделяем ребро между
ними.<br>");
            }
        }
    }

    private void clearVisitedNodes(Graph graph) {
        for (Node node : graph.getNodes()) {
            node.visited = false;
        }
    }

    private void clearHighlightedNodes(Graph graph) {
        for (Node node : graph.getNodes()) {
            node.highlighted = false;
        }
    }

    private void clearHighlightedEdges(Graph graph) {
        for (Edge edge : graph.getEdges()) {
            edge.highlighted = false;
        }
    }

    private void highlightAllEdges(Graph graph) {
        for (Edge edge :
                graph.getEdges()) {
            edge.highlighted = true;
        }
    }

    /**
     * @param graph
     * @implNote Стирает все выделения, отмечает все вершины как непосещенные
     */
    private void clearAll(Graph graph) {
        clearVisitedNodes(graph);
        clearHighlightedNodes(graph);
        clearHighlightedEdges(graph);
    }
}
```

Файл GraphPlayer.java

```
package ru.eltech.logic;

import ru.eltech.view.GraphPlayerToolBar;
```

```
import ru.eltech.view.MainWindow;

import java.util.Timer;
import java.util.TimerTask;

/**
 * Проигрыватель данных из {@link FrameList}
 */
public final class GraphPlayer {
    private Timer timer = new Timer();
    private FrameList frameList;
    private State state = State.Stop;
    private volatile int currentFrame = 0;
    private int delay = 300;
    public boolean sliderChanged = false;
    public boolean stepForwardInPause = false;
    public boolean stepBackwardInPause = false;

    private final GraphPlayerToolBar toolBar;

    public GraphPlayer(GraphPlayerToolBar toolBar) {
        this.toolBar = toolBar;
    }

    public FrameList getFrameList() {
        return frameList;
    }

    public void setFrameList(FrameList frameList) {
        //MainWindow.log.info("Frame list setup");
        this.frameList = frameList;
        toolBar.playerChanged(this);
    }

    public State getState() {
        return state;
    }

    public synchronized void setState(State state) {
        if (this.state == state) return;
        switch (state) {
            case Play:
                MainWindow.log.info("Старт анимации");
                if (frameList != null && currentFrame == frameList.count() - 1)
currentFrame = 0;
                timer.cancel();
                timer = new Timer();
                timer.schedule(new PlayerTask(), delay);
                break;
            case Pause:
                //MainWindow.log.info("Пауза анимации на шаге: <br>");
                timer.cancel();
                break;
            case Stop:
                MainWindow.log.info("Стоп анимации");
                currentFrame = 0;
                timer.cancel();
                break;
        }
        this.state = state;
        toolBar.playerChanged(this);
    }
```

```
    public int getCurrentFrame(){
        return currentFrame;
    }

    public void setCurrentFrame(int currentFrame) {
        this.currentFrame = currentFrame;
        toolBar.playerChanged(this);
    }

    public int getDelay() {
        return delay;
    }

    public void setDelay(int delay) {
        this.delay = delay;
        sliderChanged = true;
        toolBar.playerChanged(this);
    }

    /**
     *
     * @return Переключился ли фрейм
     */
    public synchronized boolean stepForward() {
        if (frameList == null || state == State.Stop) return false;
        if (currentFrame >= frameList.count() - 1) {
            MainWindow.log.info("Конец анимации");
            frameList.get(frameList.count() - 1).state = "";
            currentFrame = frameList.count() - 1;
            setState(State.Pause);
            return false;
        }
        currentFrame++;
        stepForwardInPause = true;
        toolBar.playerChanged(this);

//MainWindow.log.info(String.valueOf(frameList.get(getCurrentFrame()).state));
        return true;
    }

    /**
     *
     * @return Переключился ли фрейм
     */
    public synchronized boolean stepBackward() {
        if (frameList == null || state == State.Stop) return false;
        if (currentFrame <= 0) {
            //MainWindow.log.warning("Нет предыдущего фрейма");
            currentFrame = 0;
            setState(State.Pause);
            return false;
        }
        currentFrame--;
        stepBackwardInPause = true;
        toolBar.playerChanged(this);

//MainWindow.log.info(String.valueOf(frameList.get(getCurrentFrame()).state));
        return true;
    }

    public class PlayerTask extends TimerTask {

        @Override
```

```
        public void run() {
            if (stepForward()) {
                timer.cancel();
                timer = new Timer();
                timer.schedule(new PlayerTask(), delay);
            }
        }
    }
    public enum State {
        Play,
        Pause,
        Stop
    }
}
```

Файл Graph.java

```
package ru.eltech.logic;

import java.io.*;
import java.util.ArrayList;
import java.util.Collection;
import java.util.HashMap;
import java.util.StringTokenizer;

/**
 * @author Samoilova Anna
 */
public final class Graph {
    private int nextNodeId;
    private int nextEdgeId;
    private final HashMap<Integer, Node> nodeMap;
    private final HashMap<Integer, Edge> edgeMap;
    public String state = "";

    public Graph() {
        nextNodeId = 1;
        nextEdgeId = 1;
        nodeMap = new HashMap<>();
        edgeMap = new HashMap<>();
    }

    public Graph(Graph other) {
        this.nextNodeId = other.nextNodeId;
        this.nextEdgeId = other.nextEdgeId;
        this.state = other.state;
        this.nodeMap = new HashMap<>();
        for (Node node : other.getNodes()) nodeMap.put(node.id, node.clone());
        this.edgeMap = new HashMap<>();
        for (Edge edge : other.getEdges()) edgeMap.put(edge.id, edge.clone());
    }


    public Node createNode(int x, int y) {
        while (nodeMap.containsKey(nextNodeId)) nextNodeId++;
        Node node = new Node(nextNodeId++, x, y);
        nodeMap.put(node.id, node);
        return node;
    }

    @SuppressWarnings("UnusedReturnValue")
    public Edge createEdge(Node from, Node to) {
        while (nodeMap.containsKey(nextEdgeId)) nextEdgeId++;
        Edge edge = new Edge(nextEdgeId++, from, to);
        edgeMap.put(edge.id, edge);
```

```
        return edge;
    }

    public Collection<Node> getNodes() {
        return nodeMap.values();
    }

    public Collection<Edge> getEdges() {
        return edgeMap.values();
    }

    public int getNodesCount() {
        return nodeMap.size();
    }

    public int getEdgesCount() {
        return edgeMap.size();
    }

    public int getVisitedNodesCount() {
        int count = 0;
        for(Node node : getNodes()) {
            if(node.visited) ++count;
        }
        return count;
    }

    public Node getNode(Integer nodeId) {
        return nodeMap.get(nodeId);
    }

    public Node getNode(String nodeName) {
        for (Node node : getNodes()) {
            if (node.getName().equals(nodeName)) return node;
        }
        return null;
    }

    public Edge getEdge(Integer edgeId) {
        return edgeMap.get(edgeId);
    }

    public Edge getEdge(Node source, Node target, boolean ignoreDirections) {
        if (!containsNode(source) || !containsNode(target)) return null;
        return getEdge(source.id, target.id, ignoreDirections);
    }

    public Edge getEdge(Integer source, Integer target, boolean
ignoreDirections) {
        for (Edge edge : getEdges()) {
            if ((edge.getSource().equals(source) &&
edge.getTarget().equals(target)) || (!ignoreDirections &&
edge.getSource().equals(target) && edge.getTarget().equals(source))) {
                return edge;
            }
        }
        return null;
    }

    public Collection<Edge> getEdgesFromNode(Node node) {
        return getEdgesFromNodeNoAlloc(node, new ArrayList<>());
    }
```

```
    public Collection<Edge> getEdgesFromNodeNoAlloc(Node node, Collection<Edge>
result) {
        result.clear();
        for (Edge edge : getEdges()) {
            if (edge.getSource().equals(node.id)) result.add(edge);
        }
        return result;
    }

    @SuppressWarnings("BooleanMethodIsAlwaysInverted")
    public boolean containsNode(Node node) {
        return nodeMap.get(node.id) == node;
    }

    public boolean containsEdge(Edge edge) {
        return edgeMap.get(edge.id) == edge;
    }

    public boolean destroyNode(Node node) {
        Node removed = nodeMap.remove(node.id);
        if (removed == node) {
            edgeMap.values().removeIf(e -> e.getSource().equals(node.id) ||
e.getTarget().equals(node.id));
            return true;
        }
        nodeMap.put(removed.id, removed);
        return false;
    }

    public boolean destroyEdge(Edge edge) {
        if (edge == null) return false;
        Edge removed = edgeMap.remove(edge.id);
        if (removed == edge) return true;
        edgeMap.put(removed.id, removed);
        return false;
    }

    public void clear() {
        this.nextNodeId = 1;
        this.nextEdgeId = 1;
        nodeMap.clear();
        edgeMap.clear();
    }

    public Graph load(InputStream stream) throws IOException {
        clear();
        BufferedReader reader = new BufferedReader(new
InputStreamReader(stream));
        try {
            int nodesCount = Integer.parseInt(reader.readLine());
            for (int i = 0; i < nodesCount; i++) {
                StringTokenizer tokenizer = new
StringTokenizer(reader.readLine());
                String name = tokenizer.nextToken();
                int posX = Integer.parseInt(tokenizer.nextToken());
                int posY = Integer.parseInt(tokenizer.nextToken());
                Node node = createNode(posX, posY);
                node.setName(name);
            }
            int edgesCount = Integer.parseInt(reader.readLine());
            for (int i = 0; i < edgesCount; i++) {
                StringTokenizer tokenizer = new
StringTokenizer(reader.readLine());
```

```
                Node source = getNode(tokenizer.nextToken());
                Node target = getNode(tokenizer.nextToken());
                if (source == null || target == null) {
                    System.out.println("Edge with missed nodes " + source + " "
+ target);
                    continue;
                }
                createEdge(source, target);
            }
        } catch (Exception e) {
            throw new IOException(e);
        }
        return this;
    }

    public void save(OutputStream stream) {
        PrintWriter p = new PrintWriter(stream);
        p.println(nodeMap.size());
        for (Node node : nodeMap.values()) {
            p.println(String.format("%s %d %d", node.getName(), node.getX(),
node.getY()));
        }
        p.println(edgeMap.size());
        for (Edge edge : edgeMap.values()) {
            Node source = getNode(edge.getSource());
            Node target = getNode(edge.getTarget());
            p.println(String.format("%s %s", source.getName(),
target.getName()));
        }
        p.flush();
    }

    public String getNodesInComponentById(int strongComponentId) {
        String nodesStr = "| ";
        for (Node node : getNodes()) {
            if (node.strongComponentId == strongComponentId) {
                nodesStr += node.getName() + " | ";
            }
        }
        return nodesStr;
    }

    public boolean componentExists(int id) {
        boolean res = false;
        for (Node node : getNodes()) {
            if (node.strongComponentId == id) {
                res = true;
            }
        }
        return res;
    }
}
```

Файл FrameList.java

```
package ru.eltech.logic;

import java.util.ArrayList;

/**
 * TODO
 */
public final class FrameList {
    private ArrayList<Graph> frames;
```

```
    public FrameList() {
        this.frames = new ArrayList<Graph>();
    }

    public int count() {
        return frames.size();
    }

    public void add(Graph frame) {
        frame.state = "";
        frames.add(new Graph(frame));
    }

    public void add(Graph frame, String state) {
        frame.state = state;
        frames.add(new Graph(frame));
    }

    public Graph get(int index) {
        return frames.get(index);
    }

//    public void generateLastStateInfo(String whatStep, int timeOutListSize,
String edgeInvertion) {
//
//    }
}
```

Файл Edge.java

```
package ru.eltech.logic;

import java.awt.*;

/**
 * @author Samoilova Anna
 */
public final class Edge {
    public final Integer id;
    private Integer source;
    private Integer target;
    private Color color = null;
    private int stroke = 0;

    public boolean highlighted = false;
    public boolean connectsStrongComponents = false;

    public Edge(Integer id, Node source, Node target) {
        this.id = id;
        this.source = source.id;
        this.target = target.id;
    }

    public Edge(Edge other) {
        this.id = other.id;
        this.source = other.source;
        this.target = other.target;
        this.highlighted = other.highlighted;
        this.connectsStrongComponents = other.connectsStrongComponents;
        this.color = other.color;
        this.stroke = other.stroke;
    }

    public Integer getSource() {
        return source;
```

```
    }

    public Integer getTarget() {
        return target;
    }

    public Color getColor(){
        return color;
    }

    public void setColor(Color color){
        this.color = color;
    }

    public int getStroke(){
        return  stroke;
    }

    public void setStroke(int stroke){
        this.stroke = stroke;
    }

    public void invert() {
        Integer temp = source;
        source = target;
        target = temp;
    }

    public String getDescription() {
        String highlight = highlighted ? "подсвечено" : "не подсвечено";
        return "Ребро с id " + id + ". Из вершины " + source +
                " в вершину " + target + ". Сейчас " + highlight;
    }

    @SuppressWarnings("MethodDoesntCallSuperMethod")
    @Override
    public Edge clone() {
        return new Edge(this);
    }
}
```

Файл Algorithm.java

```
package ru.eltech.logic;

public interface Algorithm {
    /**
     * @param context Граф, над которым требуется выполнить алгоритм
     * @return Набор кадров, показывающий состояние графа в последовательные
моменты времени исполнения алгоритма
     */
    FrameList process(Graph context);
}
```