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

ОТЧЕТ

по учебной практике

Тема: Топологическая сортировка

Студент гр. 0303 Торопыгин А.С.

Студент гр. 0304 Шквиря Е.В.

Руководитель Фирсов М.А.

Санкт-Петербург
2022

ЗАДАНИЕ

НА УЧЕБНУЮ ПРАКТИКУ

Студент Торопыгин А.С. группы 0303

Студент Шквиря Е.В. группы 0304

Тема практики: Топологическая сортировка

Задание на практику:

Командная итеративная разработка визуализатора алгоритма на Kotlin с графическим интерфейсом.

Алгоритм: Топологическая сортировка.

Сроки прохождения практики: 29.06.2020 – 12.07.2020

Дата сдачи отчета: 12.07.2020

Дата защиты отчета: 12.07.2020

Студент		Торопыгин А.С.
Студент		Шквиря Е.В.
Руководитель		Фирсов М.А.

АННОТАЦИЯ

Требуется разработать программу, визуализирующую выполнение алгоритма топологической сортировки на произвольном графе. В качестве дополнительного функционала выступает пошаговое выполнение и считывание и сохранение данных в файл. Алгоритм топологической сортировки используется для упорядочивания вершин в графе.

Целью работы является формирование навыков командной работы, изучение новых языков программирования и работы с фреймворками.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью практической работы является разработка программы, визуализирующей работу алгоритма топологической сортировки на графе.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Исходные Требования к программе

1.1.1. Требования к визуализации

Приложение должно иметь понятный графический интерфейс. Он должен полностью описывать функционал программы. Должна быть возможность при помощи мыши выбирать место создания и удаления элементов для редактирования графа. Должно быть пространство с выводом текстовой информации о выполнении алгоритма (этапы алгоритма, актуальное состояние, промежуточные шаги).

1.1.2. Требования к вводу исходных данных

Исходные данные поступают в приложение посредством считывания с файла или созданием графа вручную пользователем (при помощи графического интерфейса).

1.1.3. Требования к структуре программы

Явное разделение программы на несколько слоёв: слой данных, слой бизнес-логики, слой отображения.

1.1.4. Требования к языку

Написание программы на языке программирования Kotlin с использованием Kotlin Multiplatform и инструментов Jetpack Compose.

1.1.5. Требования к тестированию

Тестирование программы разделено на ручное и автоматическое. Вручную будут тестироваться элементы графического интерфейса (создание графа пользователем, передвижение по алгоритму), автоматически будет тестироваться часть, скрытая от пользователя (считывание из файла и сохранение в файл данных, создание объектов классов, алгоритм). Автоматическое тестирование будет реализовано при помощи UNIT-тестов.

1.2. Уточнение требований

1.2.1. Требования к визуализации

Для прототипа первой версии программы был составлен эскиз интерфейса (см. рис. 1).

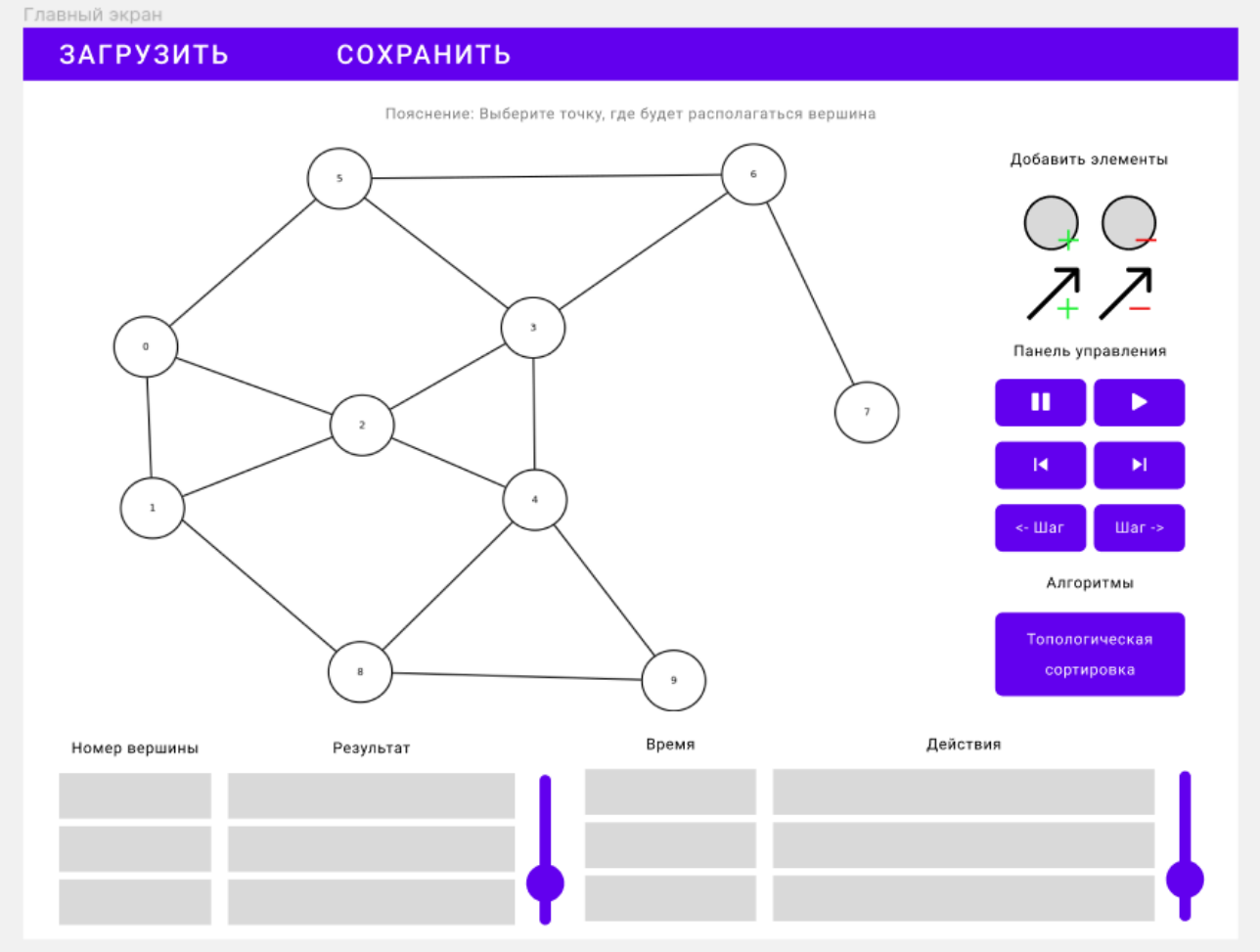

Рис. 1 - Эскиз прототипа программы

На верхней панели инструментов расположены две кнопки - загрузить и сохранить. Первая позволяет открывать для пользователя окно выбора файла, и в случае, если формат файла поддерживается, начинает процедуру считывания информации. В качестве допустимого формата представления данных был выбран JSON. Вторая сохраняет информацию о графе в файл. Способ представления будет описан ниже.

В поле “Пояснение” будет выводиться информация о необходимом действии, которое ожидается от пользователя.

Главная часть экрана приложения будет содержаться холст для работы с графом. На нём будет графическое представление данных в формате отображения вершин и рёбер. Справа от холста располагается панель инструментов для работы с графом - добавление/удаление вершины, добавление/удаление ориентированного ребра. При добавлении вершины на холст сканируется пространство для определения, не закроет ли эта вершина другую вершину, и поместится ли она полностью на холст. После этого откроется окно для ввода названия вершины. При добавлении ребра происходит выбор двух вершин, после чего происходит соединение. Первая при выборе окрашивается в жёлтый цвет. После добавления ребра цвет обоих вершин чёрный. При удалении вершины все связанные с ней рёбра тоже удаляются. При удалении ребра происходит разрыв связи между двумя вершинами. Также при удалении ребра первая вершина окрасится в красный. При удалении ребра его ориентация значения не имеет. После удаления обе вершины также будут иметь чёрный цвет.

Также на панели инструментов находится кнопка для старта работы алгоритма топологической сортировки. Кнопка “Топологическая сортировка” запускает алгоритм топологической сортировки, который работает в автоматическом режиме с некоторой задержкой между шагами алгоритма (по умолчанию, 0.5 секунд между двумя действиями). При нажатии кнопки “Шаг вперёд” или “Шаг назад” управление программой переходит пользователю и он может перемещаться между состояниями алгоритма самостоятельно. При нажатии кнопки “Пауза” программа останавливается на текущем шаге алгоритма и ждёт дальнейших указаний к работе. При нажатии кнопки “Продолжить” алгоритм продолжает свою работу в автоматическом режиме. Также есть две кнопки: “к началу” и “к концу” алгоритма. Первая осуществляет переход к началу работы алгоритма, вторая - к концу. Если в данный момент выполнялась автоматическая визуализация алгоритма - перемещение в начало прерывает её.

Также при нажатии на элемент добавления/удаления вершины/ребра кнопки для запуска алгоритмов и работы с ними блокируются, чтобы не нарушать их работу. Аналогично и с кнопками алгоритма, при старте алгоритма редактирование графа блокируется.

Ниже холста располагаются таблицы с текстовой информацией о работе алгоритма. Левая таблица отвечает за результат работы, правая - за промежуточные шаги. В первой таблице: в левой части располагается номер вершины, в правой части - результат работы алгоритма для этой вершины. Во второй таблице: в левой части располагается время, в которое произошло действие. В правой части - описание действия.

1.2.2. Требования к вводу входных данных.

Входные данные хранятся в файле JSON, в котором есть поля:

- name - имя вершины (строка);
- id - номер вершины (десятичное число);
- point - координата центра вершины (два вещественных числа);
- order - номер порядка (десятичное число);
- edges - список названий вершин, в которые идут рёбра из текущей вершины (список строк).

Было принято решение хранить имя и номер вершины. Это даёт несколько преимуществ:

- Это позволит программе работать с графом, в котором некоторые вершины могут иметь одинаковое название;
- В ходе разработки удобнее будет работать с номером вершины, а не с её именем.

1.2.3. Требования к структуре программы

В программе будет присутствовать разбиение архитектуры на несколько слоёв: слой данных, слой бизнес-логики и слой отображения. Явное разбиение

будет выглядеть таким образом: Data, Models и UI. Взаимодействие между слоями приведено на рисунке 2.

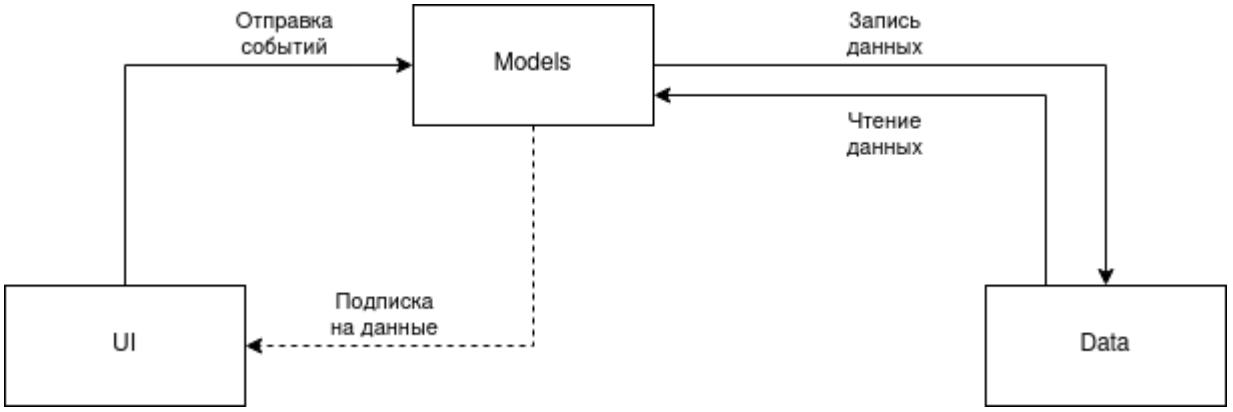

Рис. 2 - Слои архитектуры программы

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

Таблица 1. Поэтапный план разработки приложения

Дата	Задача	Статус выполнения
04.07.2022	Согласование спецификации	Выполнено
05.07.2022	Добавление классов для хранения данных. Загрузка информации о графе из файла. Функции для расположения на холсте.	Выполнено
06.07.2022	Добавление функционала для UI-элементов (добавление элементов графа). Сдача прототипа	Выполнено
07.07.2022	Добавление функционала для UI-элементов (удаление элементов графа). Реализация алгоритма топологической сортировки.	Выполнено
08.07.2022	Сохранение информации о графе. Добавление проверки валидности графа. Сдача 1-й версии программы	Выполнено
09.07.2022	Реализация разбиения работы топологической сортировки на состояния. Передача информации о шагах в таблицу логирования.	Выполнено
10.07.2022	Передача информации о шагах в UI. Добавление функционала шагов вперёд/назад. Проверка валидности данных при считывании из файла. Сдача 2-й версии программы	Выполнено
11.07.2022	Подготовка релизной версии проекта. Внесение корректировок.	Выполнено

	Тестирование.	
12.07.2022	Сдача финальной версии программы	Выполнено

2.2. Распределение ролей в бригаде

Шквиря Е.В.:

- Создание структур данных (ViewObject)
- Проектирование и реализация графического интерфейса.
- Составление архитектуры приложения, разделение приложения на логические слои.
- Реализация графической части работы алгоритма топологической сортировки, в том числе касающейся отрисовки и получения информации об этапах работы алгоритма.
- Тестирование своей части работы.

Торопыгин А.С.:

- Создание структур данных (Data)
- Реализация алгоритма топологической сортировки.
- Разбиение алгоритма на логические части для возможности пошагового графического отображения работы алгоритма, в том числе логирование этапов работы алгоритма.
- Реализация сохранения и загрузки данных.
- Тестирование своей части работы.

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Структуры данных

3.1.1. Граф

Для хранения данных были разработаны структуры данных *Vertex* (класс вершины) и *Graph* (класс графа). В классе вершины есть все необходимые поля для хранения вершины: имя, идентификатор, порядок, координаты центра и список исходящих рёбер. Класс графа хранит список вершин.

3.1.2. Отображение графа

Для явного разделения структур данных для хранения в памяти и для отрисовки был создан класс *VertexVO* наподобие *Vertex*. Данный класс хранит в себе поля: идентификатор вершины, название вершины, координаты центра вершины в формате точки и стандартный цвет для отрисовки. Также в этом классе есть статичные константные поля - радиус вершины для отрисовки и максимально допустимое количество символов. При передаче данных из слоя данных в слой отрисовки данные *Vertex* конвертируются в данные *VertexVO* через специально написанный маппер *Vertex.toVertexVO()*.

3.1.3. Сохранение и загрузка графа

Для сохранения и загрузки графа был написан класс *Parser*.

3.1.4. Алгоритм и его логирование

Для работы с алгоритмом был создан объект *GraphAlgorithm*. Чтобы выводить информацию о работе алгоритма на экран, был создан класс *State*, который позволяет отслеживать текущие состояния алгоритма и передавать их на слой визуализации.

3.1.5. Интерфейс

Для обработки состояний нажатий на кнопки панели редактирования графа и управления визуализаций алгоритмов был написан *enum* класс *GraphToolsState*, в котором собраны все состояния изменения отображаемого на холсте. Также от них зависит отображение данных в таблице действий алгоритма.

3.2. Основные методы

3.2.1. Граф

Для классов *Graph* и *Vertex* были написаны различные геттеры для получения информации о том или ином поле объекта (например, получить имя или идентификатор вершины или получить список вершин из графа.)

3.2.2. Отображение графа

Для отрисовки графа был использован элемент UI - Canvas. Данный виджет позволяет наносить рисунки на свой холст - круг, линию, текст и т.п. Для удобной отрисовки была написана библиотека *CanvasDrawLib*, в которой содержатся публичные методы *drawVertex* и *drawEdge*, получающие на вход холст Canvas и данные для отрисовки вершины и ребра соответственно. Внутри себя они также используют приватные методы, например, для нанесения имени вершины или вычисления координат начала и конца ребра. Для сохранения разделения программы на несколько слоёв был написан класс *GraphCanvasViewModel*, который содержит в себе логику обработки запросов пользователя, а также обновлением данных для их визуализации на холсте. Для поддержания актуального состояния данный класс также содержит подписку на хранилище данных о графе, и в случае его изменения (например,

при загрузке нового графа из файла или добавления необходимости менять цвет вершин), производит отрисовку нового графа.

3.2.3. Сохранение и загрузка графа

Методы класса *Parser* способны считывать и сохранять информацию о графе из файлов типа json при помощи сторонней библиотеки GSON. Для хранения данных графа используется реализованый *singleton*-объект *DataGraphLocator*. В нём есть методы *readGraphData* и *saveGraphData*, для запуска методов *Parser* и обновления состояния графа. *ViewModel`*и элементов UI имеют подписку на данные о графе в этом объекте, поэтому в них всегда загружается актуальное состояние графа.

3.2.4. Алгоритм и его логирование

Для работы алгоритма в объекте *GraphAlgorithm* был описан публичный метод *TopSort.* Для его корректной работы были написаны приватные методы *TopSortUtil, checkGraphForCycle* и т.д. Каждый приватный метод необходим для корректной работы алгоритма.

3.2.5. Интерфейс

Как было описано выше, интерфейс регистрирует запросы на изменение графа или запуск алгоритма через состояния. Для отслеживания состояний использовались *MutableFlow,* доступные из библиотеки *Coroutines*. При создании *ViewModel`*ей в них происходит регистрация на различные состояния, которые необходимо отслеживать для корректной работы элемента UI. Также отдельно стоит отметить, что в приложении используются всплывающие окна, например, для отображения имени вершины или ввода имени файла для загрузки/сохранения графа

4. ТЕСТИРОВАНИЕ

4.1. Тестирование графического интерфейса

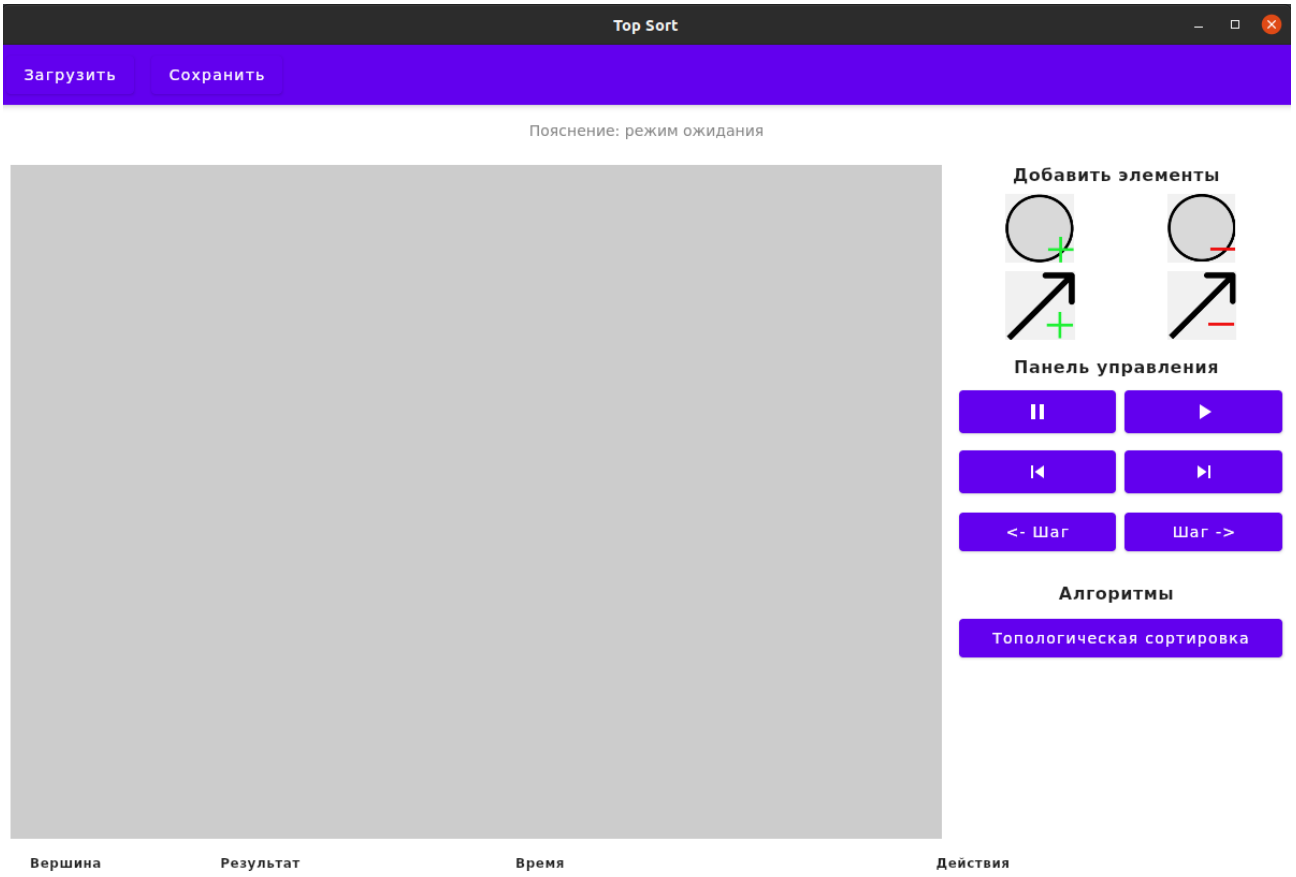

Рис. 3 - Запуск приложения + малый размер окна

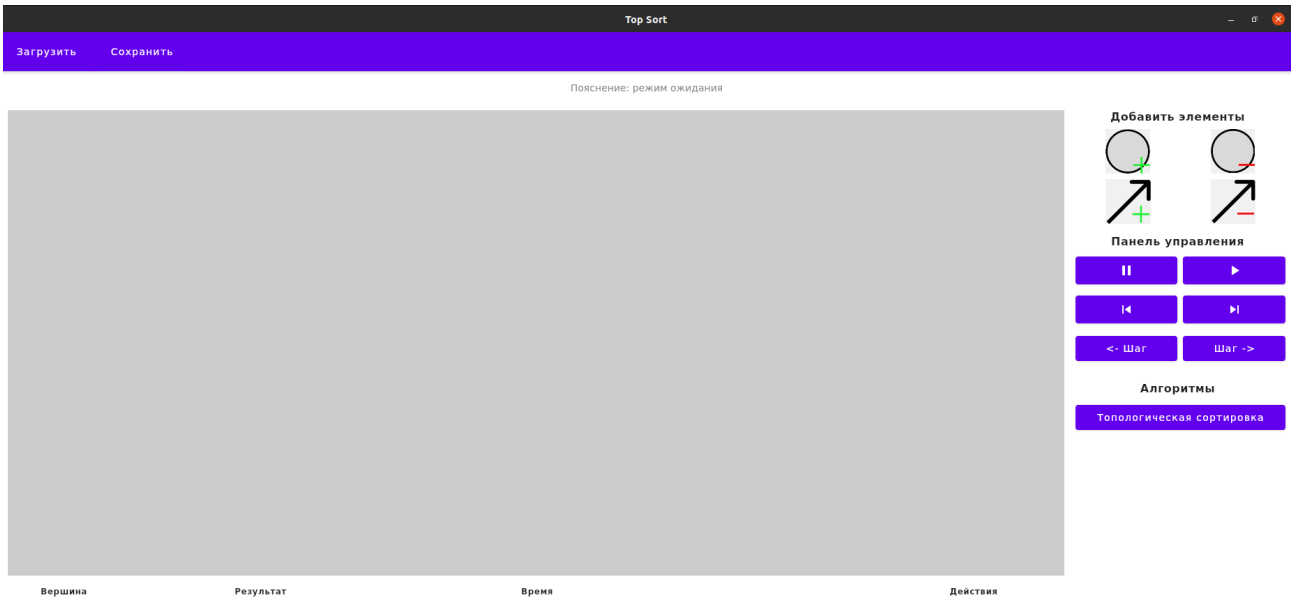

Рис. 4 - Полный размер окна

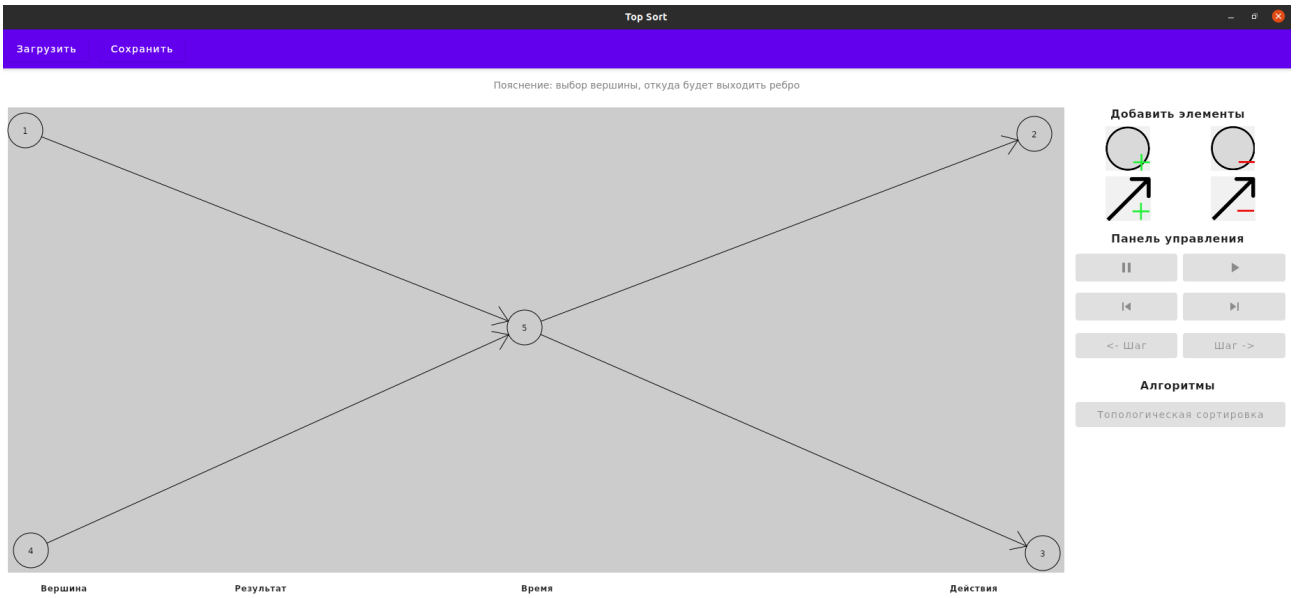

Рис. 5 - Размещение вершин и рёбер в различных местах

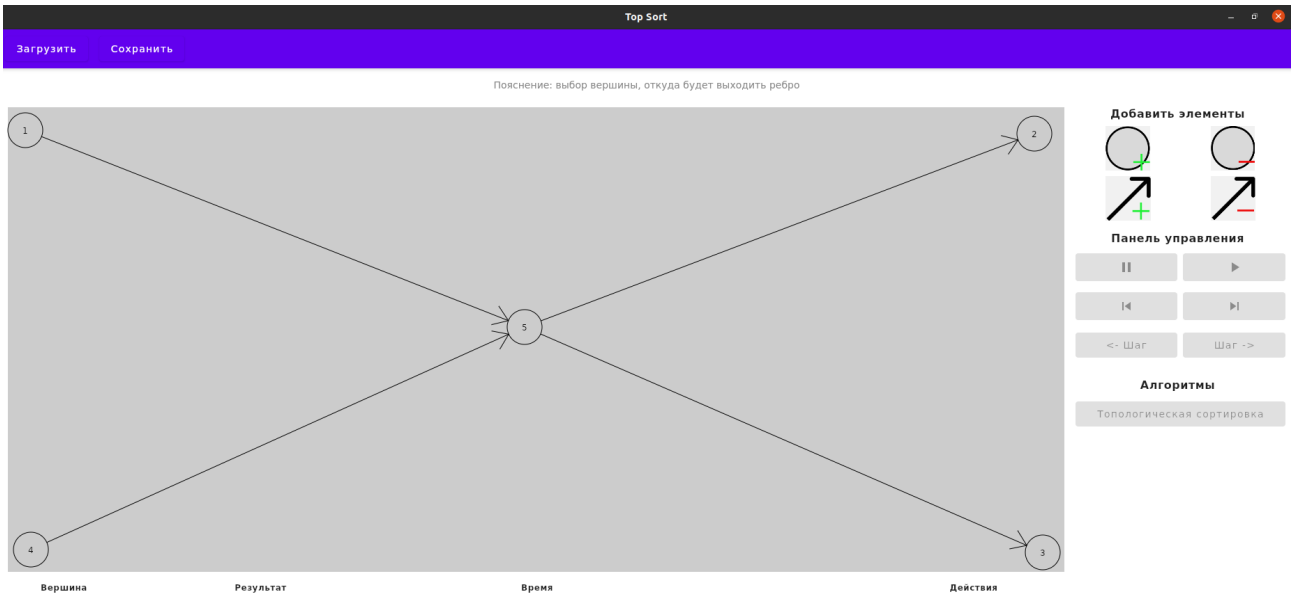

Рис. 6 - Блокировка кнопок алгоритма при редактировании графа

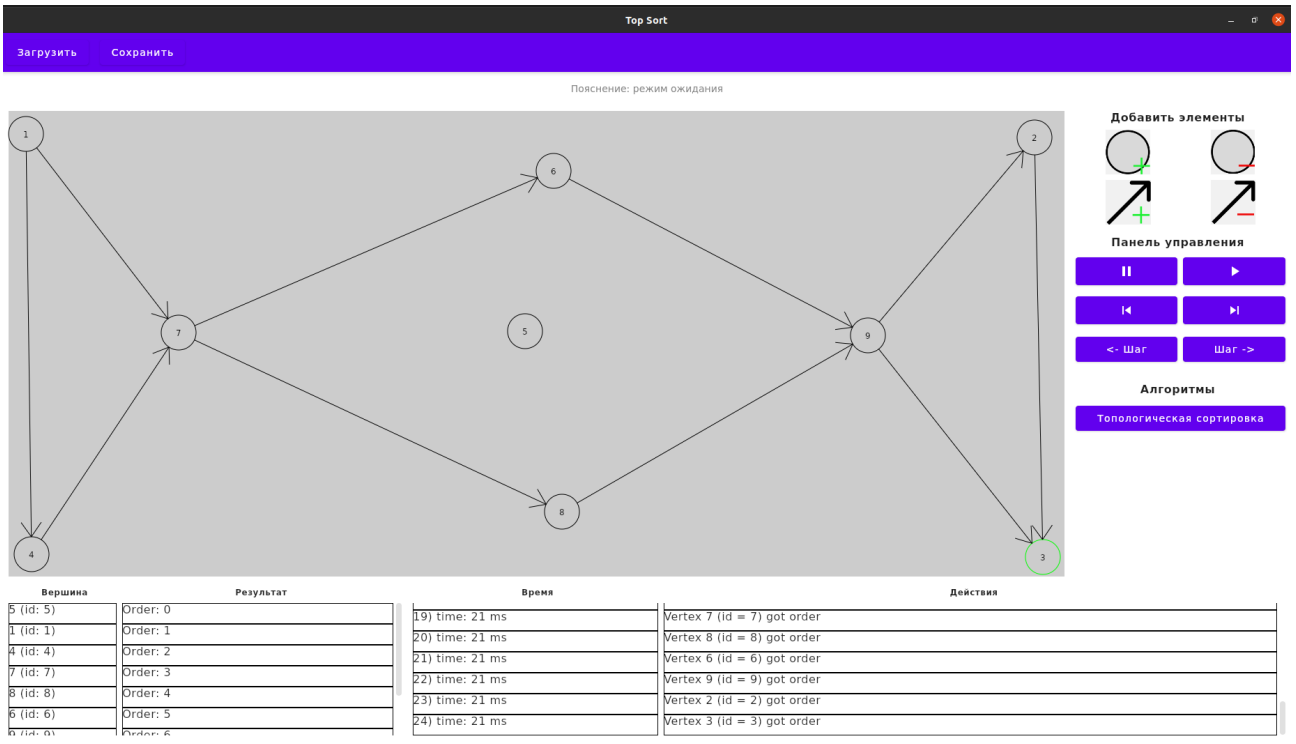

Рис. 7 - Запуск алгоритма

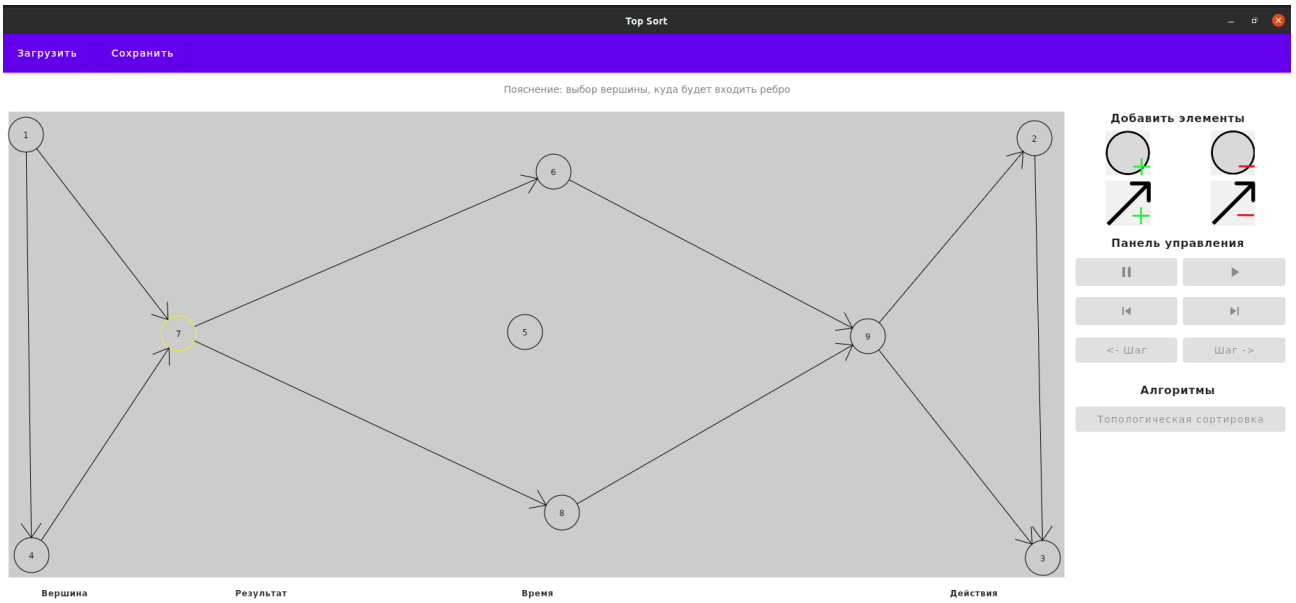

Рис. 8 - Добавление ребра

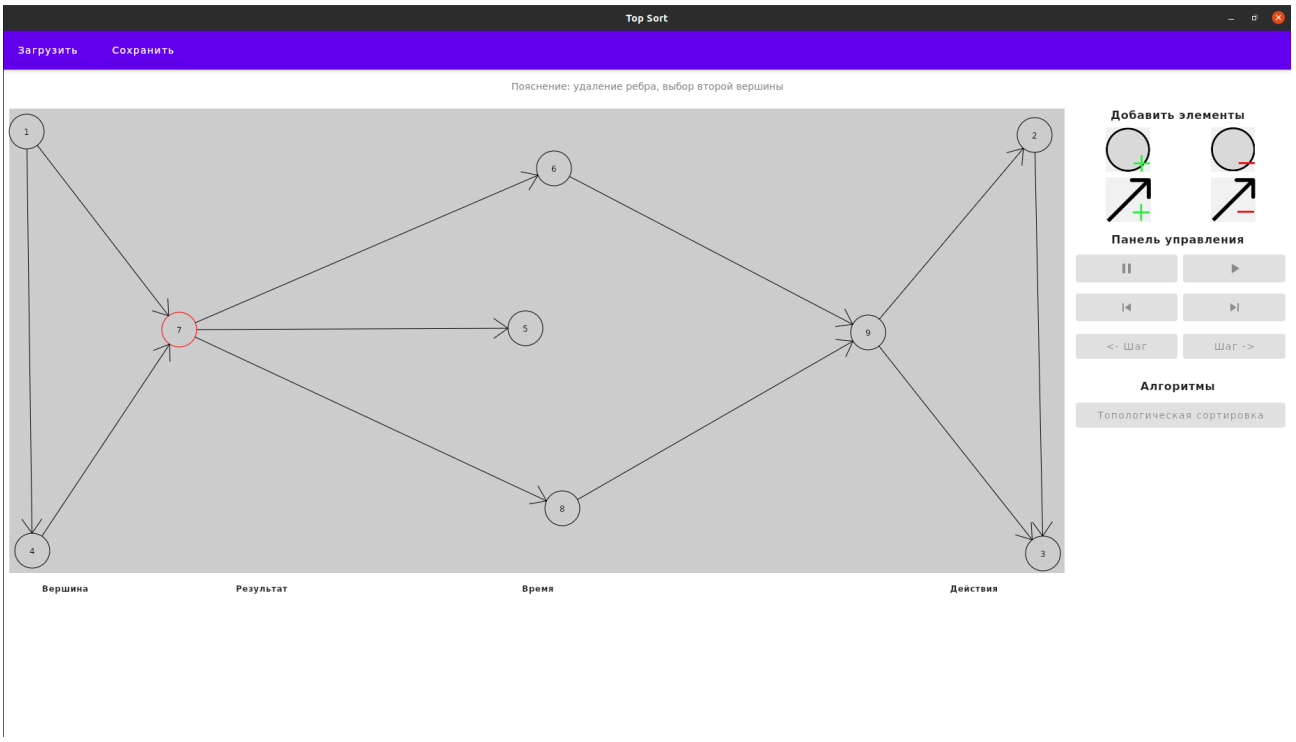

Рис. 9 - Удаление ребра

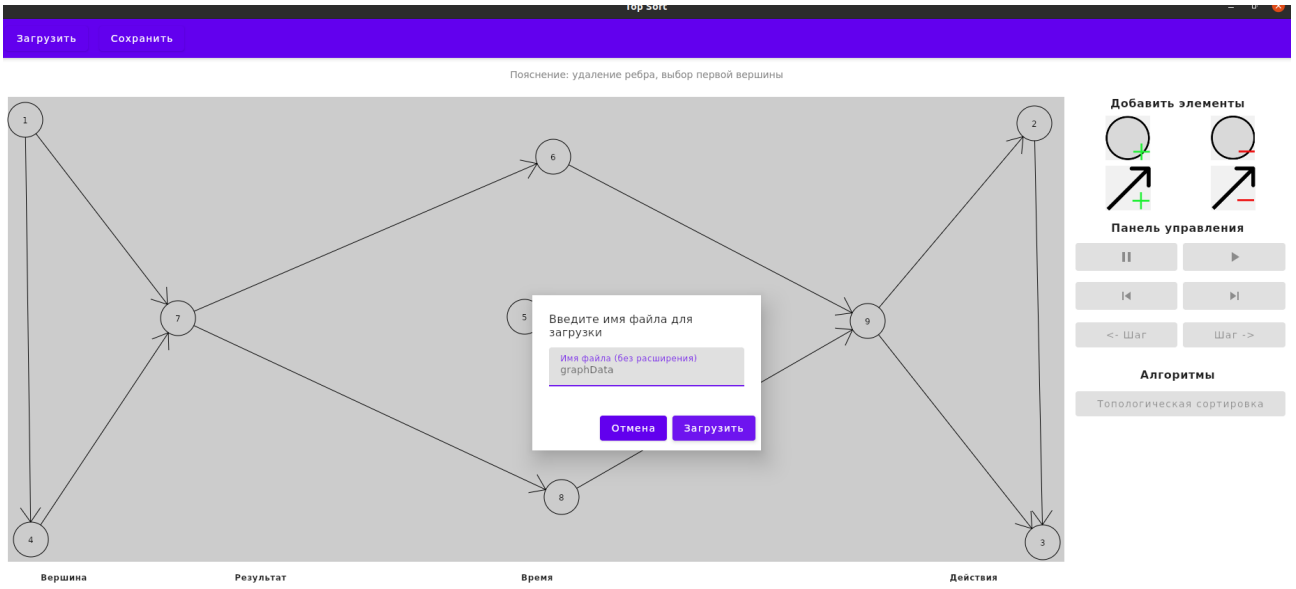

Рис. 10 - Загрузка графа из файла

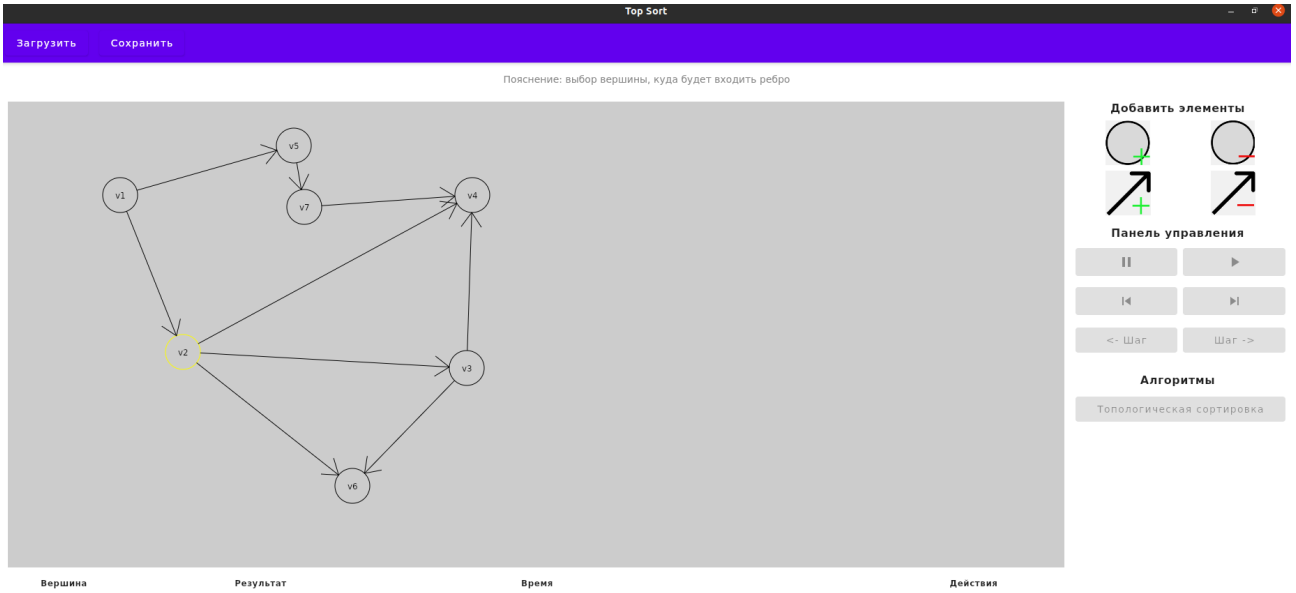

Рис. 11 - Отображение загруженного графа из файла + взаимодействие с ним (добавление ребра от v2)

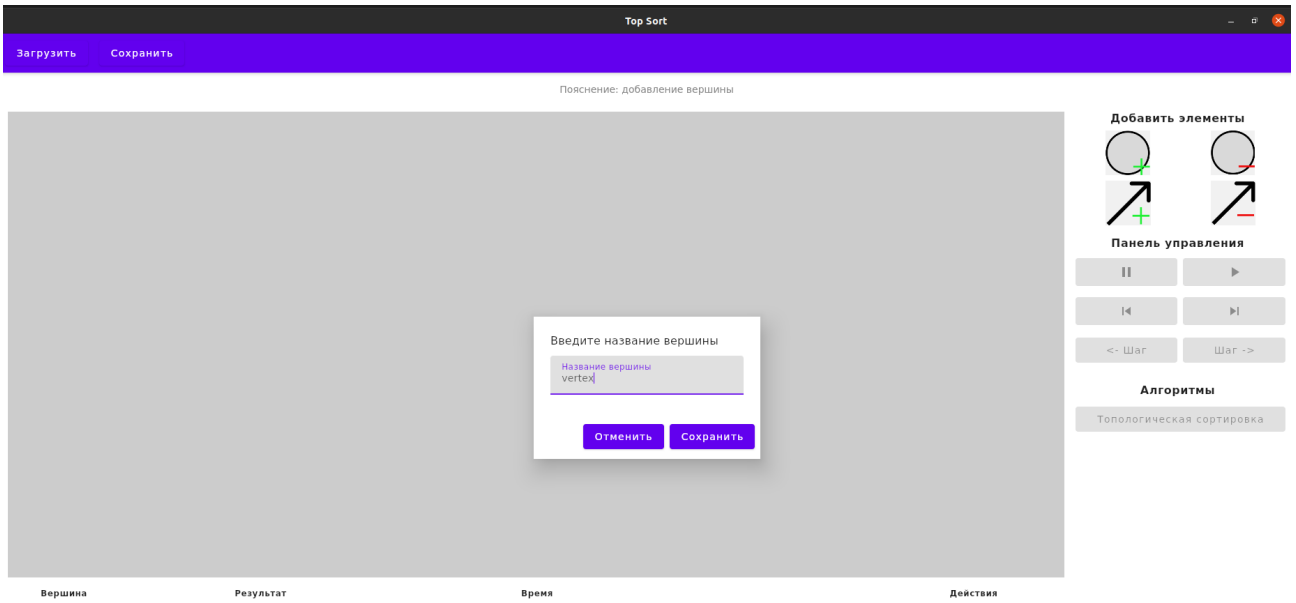

Рис. 12 - Добавление вершины + ввод её имени

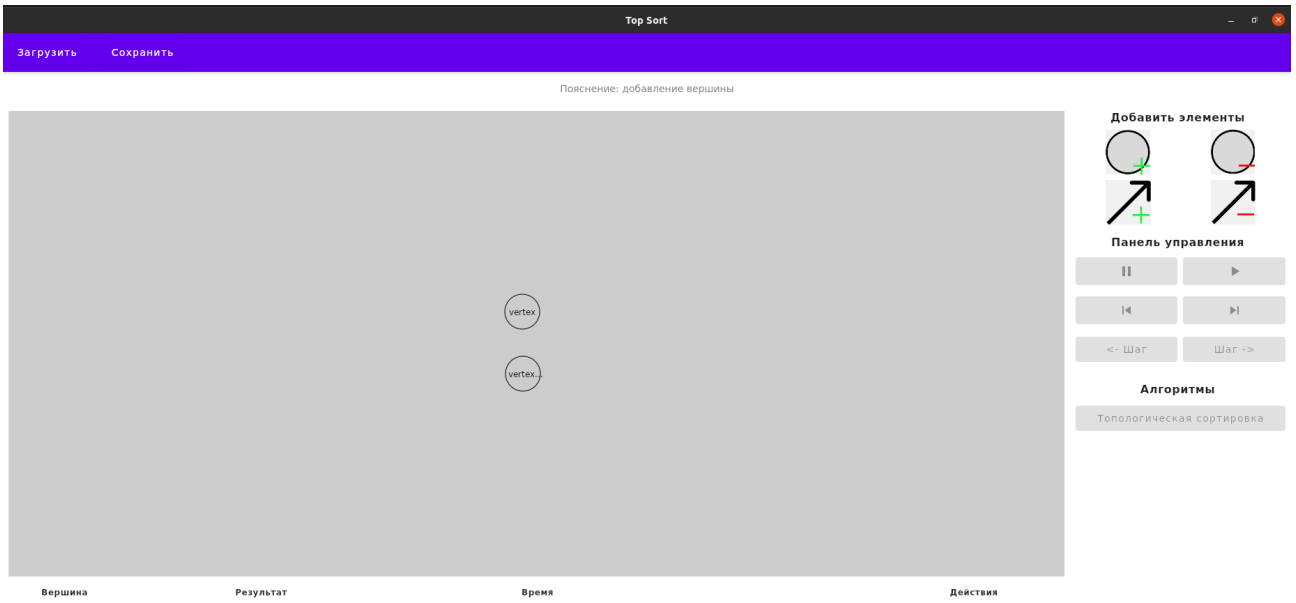

Рис. 13 - Отображение вершины с длинным именем

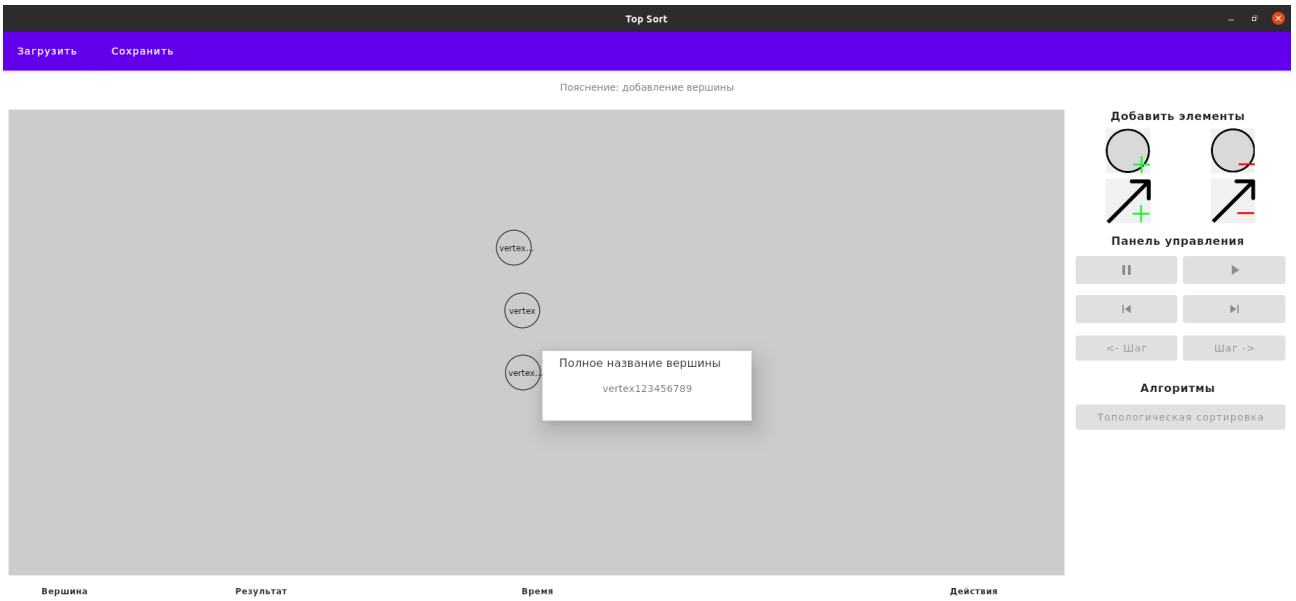

Рис. 14 - Отображение полного имени вершины

4.2. Тестирование кода алгоритма

Для тестирования основных функций алгоритма было реализовано UNIT-тестирование при помощи сторонней библиотеки JUnit5.

Для алгоритма топологической сортировки были написаны тесты, покрывающие следующие случаи: передача в алгоритм пустого графа; передача в алгоритм графа с циклом; обычные случаи, в которых алгоритм должен работать корректно (с разным количеством компонент связности).

Для алгоритма поиска цикла в графе были написаны тесты, покрывающие следующие случаи: передача в алгоритм графа с циклом, передача в алгоритм графа без цикла (с разным количеством компонент связности).

Результаты тестирования данных функций представлены на рисунке ниже.

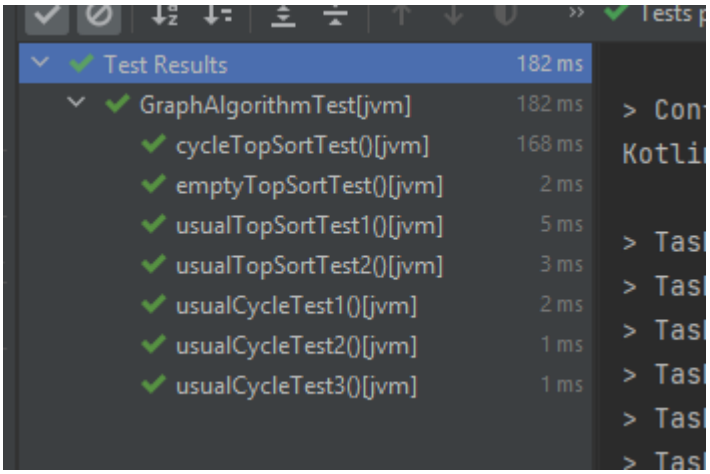

Рис. 15 - результаты тестирования основных методов алгоритма

4.3. Тестирование сохранения и загрузки данных

Для тестирования функций сохранения и загрузки данных была использована сторонняя библиотека JUnit.

Для функции загрузки данных было реализовано тестирование, покрывающее следующие случаи: считывание с пустого файла, считывание с файла с ошибкой в названии полей; считывание с файла с ошибкой в значении полей; считывание с файла без ошибок.

Для функции сохранения данных было реализовано тестирование, покрывающее следующие случаи: загрузка пустого графа; загрузка нормального графа.

Для функции, проверяющей входные данные на валидность, было реализовано тестирование, покрывающее следующие случаи: граф валиден; в графе есть вершины с одинаковым идентификатором; в графе есть вершины, рёбра из которых ведут в несуществующие вершины; в графе есть вершины с отрицательными координатами; в графе есть вершины, визуально перекрывающие друг друга.

Результаты тестирования данных функций представлены на рисунке ниже.

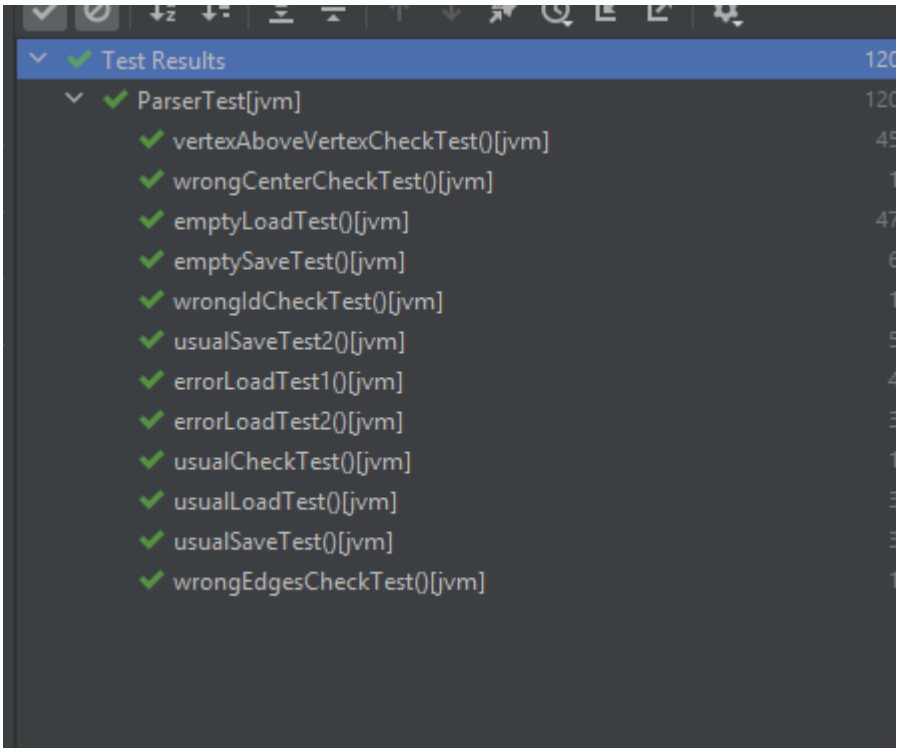

Рис. 16 - Тестирование считывания и загрузки данных

ЗАКЛЮЧЕНИЕ

При помощи языка программирования Kotlin была написана программа, визуализирующая работу алгоритма топологической сортировки на произвольном графе.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Руководство по языку Kotlin // KotlinLang. URL: https://kotlinlang.ru/ (дата обращения: 28.06.2022).
2. Примеры использования Jetpack Compose для Kotlin Multiplatform // Github. URL: https://github.com/jetbrains/compose-jb (дата обращения: 02.07.2022)
3. Уроки по работе с Jetpack Compose // Jetpack Compose Tutorial. URL: https://www.jetpackcompose.net/jetpack-compose-introduction (дата обращения: 04.07.2022).
4. Руководство по UI-элементам и каталог иконок // Material Design. URL: https://material.io/components (дата обращения: 04.07.2022).
5. Руководство по использованию Coroutines в Kotlin // KotlinLang. URL: https://kotlinlang.org/docs/coroutines-basics.html (дата обращения: 09.07.2022).

ПРИЛОЖЕНИЕ А

СХЕМА АРХИТЕКТУРЫ ПРИЛОЖЕНИЯ

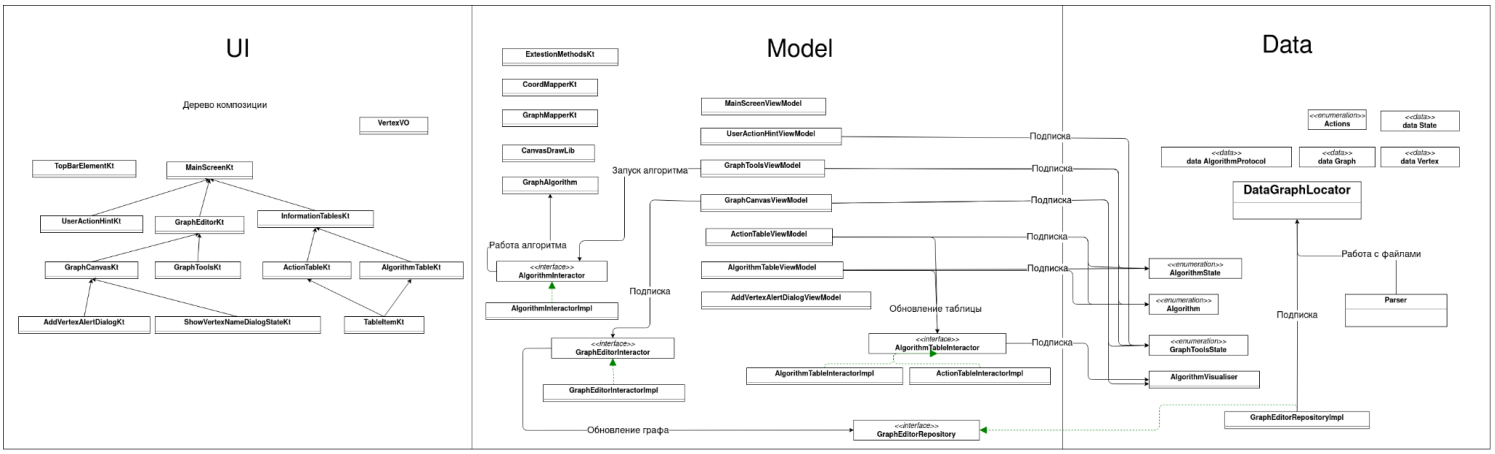

Рис. 17 - Схема архитектуры приложения

ПРИЛОЖЕНИЕ Б

TOP SORT

DataGraphLocator.kt

```
package data.graphData

import data.`object`.Graph
import kotlinx.coroutines.flow.MutableStateFlow
import utils.parsing.Parser

object DataGraphLocator {
   val graphFlow = MutableStateFlow(Graph(arrayListOf()))
   fun readGraphData(path: String): Boolean {
       lateinit var tempGraph: Graph
       try {
           tempGraph = Parser.readDataJSON(path)
       } catch (_: java.lang.Exception) {
           return false
       }
       graphFlow.value = tempGraph
       return true
   }

   fun saveGraphData(path: String) {
       Parser.writeDataJSON(path, graphFlow.value)
   }

}
```

Graph.kt

```
package data.`object`

data class Graph(private var vertexes: ArrayList<Vertex>){

   fun addVertex(vertex: Vertex) { vertexes.add(vertex) }

    fun getVertex(name: String): Vertex? = vertexes.find { it.getName()
== name }

    fun getVertex(id: Long): Vertex? = vertexes.find { it.getId() == id
}

   fun getVertex(index: Int): Vertex? {
       if (index in vertexes.indices) return vertexes[index]
       return null
   }

   fun getVertexes() = vertexes

   fun linkVertexes(vertexId1: Long, vertexId2: Long) {
       getVertex(vertexId1)?.addEdge(vertexId2)
   }

   fun removeLink(vertexId1: Long, vertexId2: Long) {
       if (getVertex(vertexId1)?.removeEdge(vertexId2) != true)
```

```
            getVertex(vertexId2)?.removeEdge(vertexId1)
    }

    fun removeVertex(vertex: Vertex?): Boolean = vertexes.remove(vertex)

                fun     removeVertexById(id:     Long):     Boolean     =
removeVertex(getVertex(id))

    fun setOrderToVertex(vertexId: Long, order: Int) {
        val vertex = getVertex(vertexId)
        if (vertex != null)
            vertex.order = order
    }

    fun clearGraph() { vertexes.clear() }

}
```

Vertex.kt

```
package data.`object`

import org.jetbrains.skia.Point

data class Vertex(
    private var id: Long,
    private var name: String,
    private val center: Point,
    var order: Int = -1,
    private var edges: ArrayList<Long> = ArrayList()
){
    fun getName() = name

    fun getId() = id

    fun getCenter(): Point = center

    fun addEdge(edge: Long) { edges.add(edge) }

    fun removeEdge(edge: Long): Boolean = edges.remove(edge)


    fun getEdges(): ArrayList<Long> = edges
}
```

GraphEditorRepositoryImpl.kt

```
package data.repositoryImpl

import data.`object`.Graph
import data.`object`.Vertex
import kotlinx.coroutines.flow.MutableStateFlow
import models.repository.GraphEditorRepository
import org.jetbrains.skia.Point
```

```
class GraphEditorRepositoryImpl(private val graphFlow:
MutableStateFlow<Graph>): GraphEditorRepository {
                            private fun getFreeId() =
(graphFlow.value.getVertexes().maxOfOrNull { it.getId() } ?: 0) + 1
    override fun getGraph(): MutableStateFlow<Graph> = graphFlow

    override fun addVertex(vertexName: String, center: Point): Vertex {
        val vertex = Vertex(
            id = getFreeId(),
            name = vertexName,
            order = -1,
            center = center,
            edges = arrayListOf()
        )
        graphFlow.value.addVertex(vertex)
        return vertex
    }

    override fun removeVertex(vertexId: Long) {
        graphFlow.value.removeVertexById(vertexId)
    }

    override fun linkVertexes(vertexId1: Long, vertexId2: Long) {
        graphFlow.value.linkVertexes(vertexId1, vertexId2)
    }

    override fun removeLink(vertexId1: Long, vertexId2: Long) {
        graphFlow.value.removeLink(vertexId1, vertexId2)
    }

    override fun setOrder(vertexId: Long, order: Int) {
        graphFlow.value.setOrderToVertex(vertexId, order)
    }
}
```

ActionTableInteractorImpl.kt

```
package models.interactor

import androidx.compose.runtime.MutableState
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import utils.algorithm.AlgorithmVisualiser

class ActionTableInteractorImpl: AlgorithmTableInteractor {
    override fun updateGraphTopSortInfo(table: MutableState<Map<String,
String>>) {
        CoroutineScope(Dispatchers.Main).launch {
            AlgorithmVisualiser.actionsTableData.collect { data ->
                table.value = data.associate { (time, action) ->
                    Pair(
                        time,
                        action
                    )
                }
            }
        }
```

```
    }
}
```

AlgorithmInteractor.kt

```
package models.interactor

interface AlgorithmInteractor {
   fun doTopSortAlgorithm()
}
```

AlgorithmInteractorImpl.kt

```
package models.interactor

import data.`object`.Graph
import kotlinx.coroutines.flow.MutableStateFlow
import utils.algorithm.AlgorithmVisualiser
import utils.algorithm.GraphAlgorithm

class       AlgorithmInteractorImpl(private       val       graphFlow:
MutableStateFlow<Graph>):
   AlgorithmInteractor {
   override fun doTopSortAlgorithm() {
                              val    (topSortResult,    protocol)    =
GraphAlgorithm.TopSortActions(graphFlow.value)
       for ((vertex, order) in topSortResult) {
           vertex.order = order
       }
       if(protocol.isNotEmpty()) {
           AlgorithmVisualiser.loadResult(
               topSortResult = topSortResult,
               protocol = protocol
           )
       }
   }
}
```

AlgorithmTableInteractor.kt

```
package models.interactor

import androidx.compose.runtime.MutableState

interface AlgorithmTableInteractor {
   fun updateGraphTopSortInfo(table: MutableState<Map<String, String>>)
}
```

AlgorithmTableInteractorImpl.kt

```
package models.interactor

import androidx.compose.runtime.MutableState
import kotlinx.coroutines.CoroutineScope
```

```
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import utils.algorithm.AlgorithmVisualiser

class AlgorithmTableInteractorImpl: AlgorithmTableInteractor {
    override fun updateGraphTopSortInfo(table: MutableState<Map<String,
String>>) {
        CoroutineScope(Dispatchers.Main).launch {
            AlgorithmVisualiser.resultTableData.collect { data ->
                table.value = data.associate { (vertex, order) ->
                    Pair(
                        "${vertex.getName()} (id: ${vertex.getId()})",
                        "Order: $order"
                    )
                }
            }
        }
    }
}
```

GraphEditorInteractor.kt

```
package models.interactor

import data.`object`.Graph
import data.`object`.Vertex
import kotlinx.coroutines.flow.MutableStateFlow
import org.jetbrains.skia.Point

interface GraphEditorInteractor {
    fun getGraph(): MutableStateFlow<Graph>
    fun addVertex(vertexName: String, center: Point): Vertex
    fun removeVertex(vertexId: Long)
    fun linkVertexes(vertexId1: Long, vertexId2: Long)
    fun removeLink(vertexId1: Long, vertexId2: Long)
    fun setOrder(vertexId: Long, order: Int)
}
```

GraphEditorInteractorImpl.kt

```
package models.interactor

import data.`object`.Graph
import kotlinx.coroutines.flow.MutableStateFlow
import models.repository.GraphEditorRepository
import org.jetbrains.skia.Point

class GraphEditorInteractorImpl(private val graphEditorRepository:
GraphEditorRepository):
    GraphEditorInteractor {
            override fun getGraph(): MutableStateFlow<Graph> =
graphEditorRepository.getGraph()

        override fun addVertex(vertexName: String, center: Point) =
graphEditorRepository.addVertex(vertexName, center)
```

```
            override fun removeVertex(vertexId: Long) =
graphEditorRepository.removeVertex(vertexId)

     override fun linkVertexes(vertexId1: Long, vertexId2: Long) =
graphEditorRepository.linkVertexes(vertexId1, vertexId2)

      override fun removeLink(vertexId1: Long, vertexId2: Long) =
graphEditorRepository.removeLink(vertexId1, vertexId2)

        override fun setOrder(vertexId: Long, order: Int) =
graphEditorRepository.setOrder(vertexId, order)
}
```

CoordMapper.kt

```
package models.mapper

import androidx.compose.ui.geometry.Offset
import org.jetbrains.skia.Point

fun Offset.toPoint() = Point(this.x, this.y)
fun Point.toOffset() = Offset(this.x, this.y)
```

GraphMapper.kt

```
package models.mapper

import androidx.compose.ui.graphics.Color
import data.`object`.Vertex
import ui.GraphEditor.GraphCanvas.VertexVO

fun Vertex.toVertexVO() = VertexVO(
   id = this.getId(),
   name = this.getName(),
   center = this.getCenter(),
   color = Color.Black
)
```

GraphEditorRepository.kt

```
package models.repository

import data.`object`.Graph
import data.`object`.Vertex
import kotlinx.coroutines.flow.MutableStateFlow
import org.jetbrains.skia.Point

interface GraphEditorRepository {
   fun getGraph(): MutableStateFlow<Graph>
   fun addVertex(vertexName: String, center: Point): Vertex
   fun removeVertex(vertexId: Long)
   fun linkVertexes(vertexId1: Long, vertexId2: Long)
   fun removeLink(vertexId1: Long, vertexId2: Long)
   fun setOrder(vertexId: Long, order: Int)
}
```

AddVertexAlertDialog.kt

```
package ui.GraphEditor.AddVertexAlertDialog

import androidx.compose.foundation.layout.sizeIn
import androidx.compose.material.AlertDialog
import androidx.compose.material.ExperimentalMaterialApi
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.MutableState
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import kotlinx.coroutines.flow.MutableStateFlow
import
ui.GraphEditor.AddVertexAlertDialog.AddVertexAlertDialogViewModel.Confi
rmButton
import
ui.GraphEditor.AddVertexAlertDialog.AddVertexAlertDialogViewModel.Dismi
ssButton
import
ui.GraphEditor.AddVertexAlertDialog.AddVertexAlertDialogViewModel.dismi
ssRequest
import
ui.GraphEditor.AddVertexAlertDialog.AddVertexAlertDialogViewModel.TextF
ieldForName

@OptIn(ExperimentalMaterialApi::class)
@Composable
fun AddVertexAlertDialog(
   vertexName: MutableStateFlow<String>
): MutableState<Boolean> {
   val openDialog = remember { mutableStateOf(false) }
   if (openDialog.value) {
       val name = remember { mutableStateOf("") }
       AlertDialog(
           modifier = Modifier
               .sizeIn(maxWidth = 325.dp, maxHeight = 250.dp),
           onDismissRequest = {
               dismissRequest(openDialog, vertexName)
           },
           confirmButton = {
               ConfirmButton(openDialog, vertexName, name)
           },
           dismissButton = {
               DismissButton(openDialog, vertexName)
           },
           title = {
               Text("Введите название вершины")
           },
           text = {
               TextFieldForName(name)
           }
       )
   }
```

```
    return openDialog
}
```

AddVertexAlertDialogViewModel.kt

```
package ui.GraphEditor.AddVertexAlertDialog

import androidx.compose.material.Button
import androidx.compose.material.Text
import androidx.compose.material.TextField
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.MutableState
import androidx.compose.runtime.remember
import androidx.compose.ui.Modifier
import androidx.compose.ui.focus.FocusRequester
import androidx.compose.ui.focus.focusRequester
import kotlinx.coroutines.flow.MutableStateFlow

object AddVertexAlertDialogViewModel {

    @Composable
    fun TextFieldForName(name: MutableState<String>) {
        val focusRequester = remember { FocusRequester() }
        TextField(
            modifier = Modifier
                .focusRequester(focusRequester),
            value = name.value,
            onValueChange = {
                if (it.length <= 25)
                    name.value = it
            },
            singleLine = true,
            label = { Text("Название вершины") }
        )
        LaunchedEffect(Unit) {
            focusRequester.requestFocus()
        }
    }
    fun dismissRequest(
        openDialog: MutableState<Boolean>,
        vertexName: MutableStateFlow<String>
    ) {
        vertexName.value = ""
        openDialog.value = false
    }


    @Composable
    fun DismissButton(
        openDialog: MutableState<Boolean>,
        vertexName: MutableStateFlow<String>
    ) {
        Button(
            onClick = {
                dismissRequest(openDialog, vertexName)
            }
```

```
        ) {
            Text("Отменить")
        }
    }

    @Composable
    fun ConfirmButton(
        openDialog: MutableState<Boolean>,
        vertexName: MutableStateFlow<String>,
        name: MutableState<String>
    ) {
        Button(
            onClick = {
                openDialog.value = false
                vertexName.value = name.value
            }
        ) {
            Text("Сохранить")
        }
    }
}
```

GraphCanvas.kt

```
package ui.GraphEditor.GraphCanvas

import androidx.compose.desktop.ui.tooling.preview.Preview
import androidx.compose.foundation.Canvas
import androidx.compose.foundation.gestures.detectTapGestures
import androidx.compose.runtime.*
import androidx.compose.ui.Modifier
import androidx.compose.ui.input.pointer.pointerInput
import data.graphData.DataGraphLocator
import data.repositoryImpl.GraphEditorRepositoryImpl
import kotlinx.coroutines.flow.MutableStateFlow
import models.interactor.GraphEditorInteractorImpl
import models.mapper.toPoint
import ui.GraphEditor.AddVertexAlertDialog.AddVertexAlertDialog
import ui.GraphEditor.ShowVertexNameDialog.ShowVertexNameDialogState
import utils.GraphToolsState


@Composable
@Preview
fun GraphCanvas(
    modifier: Modifier = Modifier,
    graphToolsStateFlow: MutableStateFlow<GraphToolsState>
) {
    val graph = remember { mutableStateListOf<VertexVO>() }
    val graphCanvasViewModel = remember {
        GraphCanvasViewModel(
            graphToolsStateFlow,

GraphEditorInteractorImpl(GraphEditorRepositoryImpl(DataGraphLocator.gr
aphFlow)),
            graph
        )
```

```
    }
    val vertexNameForAlertDialog = remember { mutableStateOf("") }
                            val         addVertexAlertDialogState         =
AddVertexAlertDialog(graphCanvasViewModel.vertexNameFlow)
                            val         showVertexNameDialogState         =
ShowVertexNameDialogState(vertexNameForAlertDialog)
    Canvas(
        modifier = modifier
            .pointerInput(Unit) {
                this.detectTapGestures(
                    onTap = { offset ->
                        graphCanvasViewModel.selectPoint(
                            addVertexAlertDialogState,
                            offset.toPoint(),
                            this.size.height,
                            this.size.width
                        )
                    },
                    onLongPress = { offset ->
                        graphCanvasViewModel.showVertexName(
                            offset.toPoint(),
                            vertexNameForAlertDialog,
                            showVertexNameDialogState
                        )
                    }
                )
            }
    ) {
                  graphCanvasViewModel.drawGraph(this.drawContext.canvas,
graph.toList())
    }

}
```

GraphCanvasViewModel.kt

```
package ui.GraphEditor.GraphCanvas

import androidx.compose.runtime.MutableState
import androidx.compose.runtime.snapshots.SnapshotStateList
import androidx.compose.ui.graphics.Canvas
import androidx.compose.ui.graphics.Color
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.collectLatest
import kotlinx.coroutines.launch
import models.interactor.GraphEditorInteractor
import models.mapper.toVertexVO
import org.jetbrains.skia.Font
import org.jetbrains.skia.Point
import utils.CanvasDrawLib
import utils.GraphToolsState
import utils.algorithm.AlgorithmVisualiser
import utils.getDistTo
import kotlin.properties.Delegates
```

```
class GraphCanvasViewModel(
   private val graphToolsStateFlow: MutableStateFlow<GraphToolsState>,
   private val graphEditorInteractor: GraphEditorInteractor,
   private val graphVertex: SnapshotStateList<VertexVO>
) {
   private val font = Font().apply {
       size = 12f
   }
   // изменение вариации вершины перед нанесением на холст
       private  var  postProcessingVertexBeforeDraw:  MutableMap<Long,
(VertexVO) -> VertexVO > = mutableMapOf()

   private var firstVertexForEdge: VertexVO? = null
   val vertexNameFlow = MutableStateFlow("")
   private lateinit var lastPoint: Point
   private var height by Delegates.notNull<Int>()
   private var width by Delegates.notNull<Int>()

   init {
       graphListener()
       editorStateListener()
   }

   private fun graphListener() {
       CoroutineScope(Dispatchers.Main).launch {
           graphEditorInteractor.getGraph().collect { graph ->
               graphVertex.clear()
                           graphVertex.addAll(graph.getVertexes().map  {
it.toVertexVO() })
               postProcessingVertexBeforeDraw.clear()
               firstVertexForEdge = null
           }
       }
       CoroutineScope(Dispatchers.Main).launch {
           AlgorithmVisualiser.graphCanvasData.collect {
               postProcessingVertexBeforeDraw = it.toMutableMap()
           }
       }
   }

   private fun editorStateListener() {
       CoroutineScope(Dispatchers.Main).launch {
           graphToolsStateFlow.collectLatest { state ->
               if ("_SECOND" !in state.name) {
                   postProcessingVertexBeforeDraw.clear()
                   firstVertexForEdge = null
               }
           }
       }
   }

   fun drawGraph(canvas: Canvas, value: List<VertexVO>) {
       val mapVertex: MutableMap<Long, Point> = mutableMapOf()
       for (vertex in value) {
                                           val    vertexForDraw    =
postProcessingVertexBeforeDraw[vertex.id]?.invoke(vertex) ?: vertex
```

```
            CanvasDrawLib.drawVertex(canvas, vertexForDraw, font)
            mapVertex[vertex.id] = vertex.center
        }
        for (vertex in value) {
                                                    val    listOfEdges    =
graphEditorInteractor.getGraph().value.getVertex(vertex.id)?.getEdges()
            for (otherVertexId in listOfEdges ?: ArrayList()) {
                    CanvasDrawLib.drawEdge(canvas, mapVertex[vertex.id],
mapVertex[otherVertexId])
            }
        }
    }

    fun selectPoint(
        addVertexAlertDialogState: MutableState<Boolean>,
        point: Point,
        height: Int,
        width: Int
    ) {
        rememberPoint(point)
        rememberWindowSize(height, width)
        when (graphToolsStateFlow.value) {
            GraphToolsState.SET_VERTEX -> {
                postProcessingVertexBeforeDraw.clear()
                firstVertexForEdge = null
                addVertexAlertDialogState.value = true
                addVertexToCanvas()
            }
            GraphToolsState.REMOVE_VERTEX -> {
                postProcessingVertexBeforeDraw.clear()
                firstVertexForEdge = null
                removeVertexFromCanvas(lastPoint)
            }
            GraphToolsState.SET_EDGE_FIRST -> {
                setFirstEdgePointAdd(lastPoint)
            }
            GraphToolsState.SET_EDGE_SECOND -> {
                setSecondEdgePointForAdd(lastPoint)
            }
            GraphToolsState.REMOVE_EDGE_FIRST -> {
                setFirstEdgePointForRemove(lastPoint)
            }
            GraphToolsState.REMOVE_EDGE_SECOND -> {
                setSecondEdgePointForRemove(lastPoint)
            }
            else -> {}
        }

    }

    private fun rememberWindowSize(height: Int, width: Int) {
        this.height = height
        this.width = width

    }

    private fun setFirstEdgePointAdd(point: Point) {
```

```
        firstVertexForEdge =
                        graphVertex.find  {  it.center.getDistTo(point)  <=
VertexVO.radius }
        if (firstVertexForEdge != null) {
//            postProcessingVertexBeforeDraw[firstVertexForEdge!!.id] =
firstVertexForEdge!!.copy(color = Color.Yellow)
             postProcessingVertexBeforeDraw[firstVertexForEdge!!.id] = {
vertex ->
                vertex.copy(color = Color.Yellow)
            }
            graphToolsStateFlow.value = GraphToolsState.SET_EDGE_SECOND
        }
    }

    private fun setSecondEdgePointForAdd(point: Point) {
        val secondVertex =
                        graphVertex.find  {  it.center.getDistTo(point)  <=
VertexVO.radius }
        if (firstVertexForEdge != null && secondVertex != null) {
            graphEditorInteractor.linkVertexes(
                firstVertexForEdge!!.id,
                secondVertex.id
            )
            graphToolsStateFlow.value = GraphToolsState.SET_EDGE_FIRST
            // уведомляем об изменении графа
            graphVertex[graphVertex.lastIndex] = graphVertex.last()
            postProcessingVertexBeforeDraw.clear()
            firstVertexForEdge = null
        }
    }

    private fun setFirstEdgePointForRemove(point: Point) {
        firstVertexForEdge =
                        graphVertex.find  {  it.center.getDistTo(point)  <=
VertexVO.radius }
        if (firstVertexForEdge != null) {
//            postProcessingVertexBeforeDraw[firstVertexForEdge!!.id] =
firstVertexForEdge!!.copy(color = Color.Red)
             postProcessingVertexBeforeDraw[firstVertexForEdge!!.id] = {
vertex ->
                vertex.copy(color = Color.Red)
            }
//            firstVertexForEdge?.color = Color.Red
                                        graphToolsStateFlow.value    =
GraphToolsState.REMOVE_EDGE_SECOND
        }
    }

    private fun setSecondEdgePointForRemove(point: Point) {
        val secondVertex =
                        graphVertex.find  {  it.center.getDistTo(point)  <=
VertexVO.radius }
        if (firstVertexForEdge != null && secondVertex != null) {
            graphEditorInteractor.removeLink(
                firstVertexForEdge!!.id,
                secondVertex.id
            )
```

```
                                                graphToolsStateFlow.value    =
GraphToolsState.REMOVE_EDGE_FIRST
            // уведомляем об изменении графа
            graphVertex[graphVertex.lastIndex] = graphVertex.last()
//              firstVertexForEdge?.color = Color.Black
            postProcessingVertexBeforeDraw.clear()
            firstVertexForEdge = null
        }
    }

    private fun addVertexToCanvas() {
        CoroutineScope(Dispatchers.Main).launch {
            vertexNameFlow.collect {
                if (it != "") {
                    if (checkVertexPosition(lastPoint, height, width)) {
                        val name = it
                        val vertex = graphEditorInteractor.addVertex(
                            name,
                            lastPoint
                        ).toVertexVO()
                        graphVertex.add(vertex)
                    }
                    vertexNameFlow.value = ""
                }
            }
        }
    }

    private fun removeVertexFromCanvas(point: Point) {
        var removedVertex: VertexVO? = null
        var indexOfRemoveVertex = 0
        for ((index, vertex) in graphVertex.withIndex()) {
            if (vertex.center.getDistTo(point) <= VertexVO.radius) {
                removedVertex = vertex
                indexOfRemoveVertex = index
                break
            }
        }
        if (removedVertex == null)
            return
                                                for       (vertex       in
graphEditorInteractor.getGraph().value.getVertexes()) {
            vertex.getEdges().removeAll { it == removedVertex.id }
        }
        graphEditorInteractor.removeVertex(removedVertex.id)
        graphVertex.removeAt(indexOfRemoveVertex)
    }

         fun   checkVertexPosition(point:   Point?,   canvasHeight:   Int,
canvasWidth: Int) =
              point   !=   null   &&   checkCanvasRange(point,   canvasWidth,
canvasHeight) &&
               checkOtherVertexPosition(point)


    private fun checkCanvasRange(
        point: Point,
```

```
        canvasWidth: Int,
        canvasHeight: Int
    ) =
            point.x in ((0f + VertexVO.radius)..(canvasWidth.toFloat() -
VertexVO.radius)) &&
                                                    point.y   in   ((0f   +
VertexVO.radius)..(canvasHeight.toFloat() - VertexVO.radius))

    private fun checkOtherVertexPosition(point: Point): Boolean {
        for (vertex in graphVertex) {
            if (point.getDistTo(vertex.center) < 2 * VertexVO.radius)
                return false
        }
        return true

    }

    private fun rememberPoint(point: Point) {
        lastPoint = point
    }

    fun showVertexName(
        point: Point,
        vertexNameForAlertDialog: MutableState<String>,
        showVertexNameDialogState: MutableState<Boolean>
    ) {
          val vertex = graphVertex.find { it.center.getDistTo(point) <=
VertexVO.radius }
        if (vertex != null ){
            vertexNameForAlertDialog.value = vertex.name
            showVertexNameDialogState.value = true
        }
    }
}
```

VertexVO.kt

```
package ui.GraphEditor.GraphCanvas

import androidx.compose.ui.graphics.Color
import org.jetbrains.skia.Point

data class VertexVO(
    val id: Long,
    val name: String,
    val center: Point,
    var color: Color = Color.Black
) {

    companion object {
        const val radius = 25f
        const val cntLetters = 6
    }
}
```

GraphTools.kt

```
package ui.GraphEditor.GraphTools

import androidx.compose.desktop.ui.tooling.preview.Preview
import androidx.compose.foundation.Image
import androidx.compose.foundation.layout.*
import androidx.compose.material.Button
import androidx.compose.material.Icon
import androidx.compose.material.IconButton
import androidx.compose.material.Text
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Pause
import androidx.compose.material.icons.filled.PlayArrow
import androidx.compose.material.icons.filled.SkipNext
import androidx.compose.material.icons.filled.SkipPrevious
import androidx.compose.runtime.Composable
import androidx.compose.runtime.MutableState
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import data.graphData.DataGraphLocator
import utils.GraphToolsState
import kotlinx.coroutines.flow.MutableStateFlow
import models.interactor.AlgorithmInteractorImpl
import utils.algorithm.Algorithm
import utils.algorithm.AlgorithmState

@Composable
@Preview
fun GraphTools(
    graphToolsStateFlow: MutableStateFlow<GraphToolsState>,
    currentAlgorithm: MutableStateFlow<Pair<Algorithm, AlgorithmState>>,
    modifier: Modifier = Modifier
) {
    val enableGraphButtons = remember { mutableStateOf(true) }
    val enableAlgorithmButtons = remember { mutableStateOf(true) }
                val     graphToolsViewModel     =     remember     {
mutableStateOf(GraphToolsViewModel(
        enableGraphButtons,
        enableAlgorithmButtons,
        AlgorithmInteractorImpl(DataGraphLocator.graphFlow),
        graphToolsStateFlow,
        currentAlgorithm
    )) }
    Column(
        modifier = modifier
    ) {
        val buttonModifier = Modifier
            .weight(0.5f)
            .align(Alignment.CenterHorizontally)
        val rowButtonsModifier = Modifier
            .fillMaxWidth()
```

```
            .padding(start = 16.dp, end = 8.dp)
                  EditGraphElements(rowButtonsModifier,  buttonModifier,
graphToolsViewModel.value, enableGraphButtons)
                    AlgorithmPanel(rowButtonsModifier,  buttonModifier,
graphToolsViewModel.value, enableAlgorithmButtons)
         AlgorithmButtons(rowButtonsModifier, graphToolsViewModel.value,
enableAlgorithmButtons)
    }
}

@Composable
fun EditGraphElements(
    rowButtonsModifier: Modifier,
    buttonModifier: Modifier,
    graphToolsViewModel: GraphToolsViewModel,
    enableGraphButtons: MutableState<Boolean>
) {
    Text(
        modifier = Modifier
            .fillMaxWidth()
            .padding(bottom = 8.dp),
        text = "Добавить элементы",
        textAlign = TextAlign.Center,
        fontWeight = FontWeight.Bold
    )
    Row(
        modifier = rowButtonsModifier
    ) {
        IconButton(
            modifier = buttonModifier
                .size(64.dp),
            enabled = enableGraphButtons.value,
            onClick = {
                graphToolsViewModel.addVertexTap()
            }
        ) {
            Image(
                painter = painterResource("drawable/add_vertex.ico"),
                contentDescription = null
            )
        }
        IconButton(
            modifier = buttonModifier
                .size(64.dp),
            enabled = enableGraphButtons.value,
            onClick = {
                graphToolsViewModel.removeVertexTap()
            }
        ) {
            Image(
                painter = painterResource("drawable/remove_vertex.ico"),
                contentDescription = null
            )
        }
    }
    Spacer(Modifier.height(8.dp))
    Row(
```

```
        modifier = rowButtonsModifier
    ) {
        IconButton(
            modifier = buttonModifier
                .size(64.dp),
            enabled = enableGraphButtons.value,
            onClick = {
                graphToolsViewModel.addEdgeTap()
            }
        ) {
            Image(
                painter = painterResource("drawable/add_edge.ico"),
                contentDescription = null
            )
        }
        IconButton(
            modifier = buttonModifier
                .size(64.dp),
            enabled = enableGraphButtons.value,
            onClick = {
                graphToolsViewModel.removeEdgeTap()
            }
        ) {
            Image(
                painter = painterResource("drawable/remove_edge.ico"),
                contentDescription = null
            )
        }
    }
}
@Composable
fun AlgorithmPanel(
    rowButtonsModifier: Modifier,
    buttonModifier: Modifier,
    graphToolsViewModel: GraphToolsViewModel,
    enableAlgorithmButtons: MutableState<Boolean>
) {
    Text(
        modifier = Modifier
            .fillMaxWidth()
            .padding(top = 16.dp, bottom = 8.dp),
        text = "Панель управления",
        textAlign = TextAlign.Center,
        fontWeight = FontWeight.Bold
    )
    Row(
        modifier = rowButtonsModifier
            .padding(bottom = 8.dp),
        horizontalArrangement = Arrangement.spacedBy(8.dp)
    ) {
        Button(
            modifier = buttonModifier,
            enabled = enableAlgorithmButtons.value,
            onClick = {
                graphToolsViewModel.pauseTap()
            }
        ) {
```

```
            Icon(
                imageVector = Icons.Default.Pause,
                contentDescription = null
            )
        }
        Button(
            modifier = buttonModifier,
            enabled = enableAlgorithmButtons.value,
            onClick = {
                graphToolsViewModel.continueTap()
            }
        ) {
            Icon(
                imageVector = Icons.Default.PlayArrow,
                contentDescription = null
            )
        }
    }
    Row(
        modifier = rowButtonsModifier
            .padding(bottom = 8.dp),
        horizontalArrangement = Arrangement.spacedBy(8.dp)
    ) {
        Button(
            modifier = buttonModifier,
            enabled = enableAlgorithmButtons.value,
            onClick = {
                graphToolsViewModel.beginOfVisualisation()
            }
        ) {
            Icon(
                imageVector = Icons.Default.SkipPrevious,
                contentDescription = null
            )
        }
        Button(
            modifier = buttonModifier,
            enabled = enableAlgorithmButtons.value,
            onClick = {
                graphToolsViewModel.finishOfVisualisation()
            }
        ) {
            Icon(
                imageVector = Icons.Default.SkipNext,
                contentDescription = null
            )
        }
    }
    Row(
        modifier = rowButtonsModifier
            .padding(bottom = 8.dp),
        horizontalArrangement = Arrangement.spacedBy(8.dp)
    ) {
        Button(
            modifier = buttonModifier,
            enabled = enableAlgorithmButtons.value,
            onClick = {
```

```
                graphToolsViewModel.stepBackTap()
            }
        ) {
            Text("<- Шаг")
        }
        Button(
            modifier = buttonModifier,
            enabled = enableAlgorithmButtons.value,
            onClick = {
                graphToolsViewModel.stepNextTap()
            }
        ) {
            Text("Шаг ->")
        }
    }
}

@Composable
fun AlgorithmButtons(
    rowButtonsModifier: Modifier,
    graphToolsViewModel: GraphToolsViewModel,
    enableAlgorithmButtons: MutableState<Boolean>
) {
    Text(
        modifier = Modifier
            .fillMaxWidth()
            .padding(top = 16.dp, bottom = 8.dp),
        text = "Алгоритмы",
        textAlign = TextAlign.Center,
        fontWeight = FontWeight.Bold
    )
    Row(
        modifier = rowButtonsModifier
    ) {
        Button(
            modifier = Modifier
                .fillMaxWidth(),
            enabled = enableAlgorithmButtons.value,
            onClick = {
                graphToolsViewModel.algTopSortTap()
            }
        ) {
            Text(
                text = "Топологическая сортировка",
                textAlign = TextAlign.Center
            )
        }
    }
}
```

GraphToolsViewModel.kt

```
package ui.GraphEditor.GraphTools

import androidx.compose.runtime.MutableState
import utils.GraphToolsState
import kotlinx.coroutines.flow.MutableStateFlow
```

```
import models.interactor.AlgorithmInteractorImpl
import utils.algorithm.Algorithm
import utils.algorithm.AlgorithmState
import utils.algorithm.AlgorithmVisualiser

class GraphToolsViewModel(
   private val enableGraphButtons: MutableState<Boolean>,
   private val enableAlgorithmButtons: MutableState<Boolean>,
   private val algorithmInteractorImpl: AlgorithmInteractorImpl,
   private val graphToolsStateFlow: MutableStateFlow<GraphToolsState>,
      private val currentAlgorithm: MutableStateFlow<Pair<Algorithm,
AlgorithmState>>
) {
   fun addVertexTap() {
           graphToolsStateFlow.value = if(graphToolsStateFlow.value ==
GraphToolsState.SET_VERTEX) {
           enableAlgorithmButtons.value = true
           GraphToolsState.WAITING
       }
       else {
           enableAlgorithmButtons.value = false
           GraphToolsState.SET_VERTEX
       }
   }

   fun removeVertexTap() {
           graphToolsStateFlow.value = if(graphToolsStateFlow.value ==
GraphToolsState.REMOVE_VERTEX) {
           enableAlgorithmButtons.value = true
           GraphToolsState.WAITING
       }
       else {
           enableAlgorithmButtons.value = false
           GraphToolsState.REMOVE_VERTEX
       }
   }

   fun addEdgeTap() {
           graphToolsStateFlow.value = if(graphToolsStateFlow.value ==
GraphToolsState.SET_EDGE_FIRST) {
           enableAlgorithmButtons.value = true
           GraphToolsState.WAITING
       }
       else {
           enableAlgorithmButtons.value = false
           GraphToolsState.SET_EDGE_FIRST
       }
   }

   fun removeEdgeTap() {
          graphToolsStateFlow.value = if (graphToolsStateFlow.value ==
GraphToolsState.REMOVE_EDGE_FIRST) {
           enableAlgorithmButtons.value = true
           GraphToolsState.WAITING
       }
       else {
           enableAlgorithmButtons.value = false
```

```
            GraphToolsState.REMOVE_EDGE_FIRST
        }
    }

    fun pauseTap() {
        graphToolsStateFlow.value = GraphToolsState.PAUSE
            currentAlgorithm.value  =  Pair(currentAlgorithm.value.first,
AlgorithmState.PAUSE)
        AlgorithmVisualiser.pauseVisualise()
    }

    fun continueTap() {
        graphToolsStateFlow.value = GraphToolsState.CONTINUE
            currentAlgorithm.value  =  Pair(currentAlgorithm.value.first,
AlgorithmState.IN_PROGRESS_AUTO)
        AlgorithmVisualiser.pauseVisualise()

AlgorithmVisualiser.startVisualiseTopSortAlgorithm(AlgorithmVisualiser.
defaultPeriod)
    }

    fun finishOfVisualisation() {
        graphToolsStateFlow.value = GraphToolsState.TO_FINISH
        AlgorithmVisualiser.toFinish()
    }

    fun beginOfVisualisation() {
        graphToolsStateFlow.value = GraphToolsState.TO_BEGIN
        AlgorithmVisualiser.toBegin()
    }

    fun stepBackTap() {
        graphToolsStateFlow.value = GraphToolsState.STEP_BACK
            currentAlgorithm.value  =  Pair(currentAlgorithm.value.first,
AlgorithmState.IN_PROGRESS_USER)
        AlgorithmVisualiser.pauseVisualise()
        AlgorithmVisualiser.stepBack()
    }

    fun stepNextTap() {
        graphToolsStateFlow.value = GraphToolsState.STEP_NEXT
            currentAlgorithm.value  =  Pair(currentAlgorithm.value.first,
AlgorithmState.IN_PROGRESS_USER)
        AlgorithmVisualiser.pauseVisualise()
                                AlgorithmVisualiser.algorithmState    =
AlgorithmState.IN_PROGRESS_USER
        AlgorithmVisualiser.stepNext()
    }

    fun algTopSortTap() {
        if (!enableGraphButtons.value) {
            enableGraphButtons.value = true
            AlgorithmVisualiser.stopVisualise()
        } else {
            enableGraphButtons.value = false
            AlgorithmVisualiser.pauseVisualise()
```

```
                currentAlgorithm.value = Pair(Algorithm.ALG_TOP_SORT,
AlgorithmState.START)
            algorithmInteractorImpl.doTopSortAlgorithm()
                currentAlgorithm.value = Pair(Algorithm.ALG_TOP_SORT,
AlgorithmState.FINISH)

AlgorithmVisualiser.startVisualiseTopSortAlgorithm(AlgorithmVisualiser.
defaultPeriod)
            notifyAutoProcessingAlgorithm()
        }
    }

    private fun notifyAutoProcessingAlgorithm() {
        currentAlgorithm.value =
                                        Pair(Algorithm.ALG_TOP_SORT,
AlgorithmState.IN_PROGRESS_AUTO)
    }



}
```

ShowVertexNameDialogState.kt

```
package ui.GraphEditor.ShowVertexNameDialog

import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.size
import androidx.compose.material.AlertDialog
import androidx.compose.material.ExperimentalMaterialApi
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.MutableState
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp

@OptIn(ExperimentalMaterialApi::class)
@Composable
fun ShowVertexNameDialogState(
    vertexNameForAlertDialog: MutableState<String>
): MutableState<Boolean> {
    val alertDialogState = remember { mutableStateOf(false) }
     if (alertDialogState.value && vertexNameForAlertDialog.value != "")
{
        AlertDialog(
            modifier = Modifier
                .size(width = 300.dp, height = 100.dp),
            onDismissRequest = {
                alertDialogState.value = false
            },
            title = {
                Text("Полное название вершины")
            },
```

```
            text = {
                Box(
                    modifier = Modifier
                        .fillMaxSize()
                ) {
                    Text(
                        modifier = Modifier
                            .align(Alignment.Center),
                        text = vertexNameForAlertDialog.value
                    )
                }
            },
            buttons = {

            }
        )
    }
    return alertDialogState
}
```

GraphEditor.kt

```
package ui.GraphEditor

import androidx.compose.desktop.ui.tooling.preview.Preview
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.BoxWithConstraints
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.layout.width
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import kotlinx.coroutines.flow.MutableStateFlow
import ui.GraphEditor.GraphCanvas.GraphCanvas
import ui.GraphEditor.GraphTools.GraphTools
import utils.algorithm.Algorithm
import utils.GraphToolsState
import utils.algorithm.AlgorithmState

@Composable
@Preview
fun GraphEditor(
    graphToolsStateFlow: MutableStateFlow<GraphToolsState>,
    currentAlgorithm: MutableStateFlow<Pair<Algorithm, AlgorithmState>>,
    modifier: Modifier = Modifier
) {
    BoxWithConstraints(
        modifier = modifier
    ) {
        val boxWithConstraintsScope = this
        val graphEditorWidth = 325.dp
        GraphCanvas(
            graphToolsStateFlow = graphToolsStateFlow,
            modifier = Modifier
```

```
                .align(Alignment.TopStart)
                .size(
                            width = boxWithConstraintsScope.maxWidth -
graphEditorWidth,
                    height = boxWithConstraintsScope.minHeight
                )
                .background(
                    color = Color.LightGray
                )
        )
        GraphTools(
            graphToolsStateFlow = graphToolsStateFlow,
            currentAlgorithm = currentAlgorithm,
            modifier = Modifier
                .align(Alignment.TopEnd)
                .width(graphEditorWidth)
                .height(boxWithConstraintsScope.minHeight)
                .background(Color.White)

        )
    }

}
```

ActionTable.kt

```
package ui.InformationTables.ActionTable

import androidx.compose.foundation.VerticalScrollbar
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.rememberLazyListState
import androidx.compose.foundation.rememberScrollbarAdapter
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import kotlinx.coroutines.flow.MutableStateFlow
import models.interactor.ActionTableInteractorImpl
import ui.InformationTables.TableItem
import utils.algorithm.Algorithm
import utils.algorithm.AlgorithmState


@Composable
fun ActionTable(
    currentAlgorithm: MutableStateFlow<Pair<Algorithm, AlgorithmState>>,
    modifier: Modifier = Modifier
) {
    val lazyColumnState = rememberLazyListState(
        initialFirstVisibleItemIndex = 0
```

```
    )
        val mapOfActions = remember { mutableStateOf(mapOf<String,
String>()) }
    val actionTableViewModel = remember {
        mutableStateOf(
            ActionTableViewModel(
                ActionTableInteractorImpl(),
                currentAlgorithm,
                mapOfActions
            )
        )
    }
    val timeCardWeight = 40f
    val actionCardWeight = 100f

    Column(
        modifier = modifier
    ) {
        Row(Modifier
            .fillMaxWidth()
            .padding(bottom = 8.dp)
        ) {
            Text(
                modifier = Modifier
                    .weight(timeCardWeight),
                text = "Время",
                textAlign = TextAlign.Center,
                fontWeight = FontWeight.Bold,
                fontSize = 12.sp
            )
            Text(
                modifier = Modifier
                    .weight(actionCardWeight),
                text = "Действия",
                textAlign = TextAlign.Center,
                fontWeight = FontWeight.Bold,
                fontSize = 12.sp
            )

        }
        Box {
            LazyColumn(
                state = lazyColumnState,
                modifier = Modifier
                    .padding(end = 12.dp)
            ) {
                actionTableViewModel.value.fillTable()
                for ((time, action) in mapOfActions.value) {
                    item {
                        TableItem(
                            time,
                            action,
                            timeCardWeight,
                            actionCardWeight
                        )
                    }
                }
```

```
            }
            VerticalScrollbar(
                modifier = Modifier
                    .align(Alignment.TopEnd)
                    .fillMaxHeight(),
                adapter = rememberScrollbarAdapter(
                    scrollState = lazyColumnState
                )

            )
        }

    }
}
```

ActionTableViewModel.kt

```
package ui.InformationTables.ActionTable

import androidx.compose.runtime.MutableState
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.launch
import models.interactor.AlgorithmTableInteractor
import utils.algorithm.Algorithm
import utils.algorithm.AlgorithmState

class ActionTableViewModel(
    private val actionTableInteractor: AlgorithmTableInteractor,
        private  val  currentAlgorithm:  MutableStateFlow<Pair<Algorithm,
AlgorithmState>>,
    private val actionTableState: MutableState<Map<String, String>>
) {
    fun fillTable() {
        CoroutineScope(Dispatchers.Main).launch {
            currentAlgorithm.collect { data ->
                when(data.first) {
                    Algorithm.ALG_TOP_SORT -> {
                                                    if  (data.second  ==
AlgorithmState.IN_PROGRESS_AUTO) {

actionTableInteractor.updateGraphTopSortInfo(actionTableState)
                        }
                    }
                    else -> {}
                }
            }
        }
    }
}
```

AlgorithmTable.kt

```
package ui.InformationTables.AlgorithmTable
```

```
import androidx.compose.desktop.ui.tooling.preview.Preview
import androidx.compose.foundation.VerticalScrollbar
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.rememberLazyListState
import androidx.compose.foundation.rememberScrollbarAdapter
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import kotlinx.coroutines.flow.MutableStateFlow
import models.interactor.AlgorithmTableInteractorImpl
import ui.InformationTables.TableItem
import utils.algorithm.Algorithm
import utils.algorithm.AlgorithmState

@Composable
@Preview
fun AlgorithmTable(
   currentAlgorithm: MutableStateFlow<Pair<Algorithm, AlgorithmState>>,
   modifier: Modifier = Modifier
) {
   val lazyColumnState = rememberLazyListState(
       initialFirstVisibleItemIndex = 0
   )
    val mapOfResult = remember { mutableStateOf(mapOf<String, String>())
}
   val algorithmTableViewModel = remember {
       mutableStateOf(
           AlgorithmTableViewModel(
               AlgorithmTableInteractorImpl(),
               currentAlgorithm,
               mapOfResult
           )
       )
   }
   val numberCardWeight = 40f
   val resultCardWeight = 100f

   Column(
       modifier = modifier
   ) {
       Row(Modifier
           .fillMaxWidth()
           .padding(bottom = 8.dp)
       ) {
           Text(
               modifier = Modifier
                   .weight(numberCardWeight),
               text = "Вершина",
               textAlign = TextAlign.Center,
```

```
                fontWeight = FontWeight.Bold,
                fontSize = 12.sp
            )
            Text(
                modifier = Modifier
                    .weight(resultCardWeight),
                text = "Результат",
                textAlign = TextAlign.Center,
                fontWeight = FontWeight.Bold,
                fontSize = 12.sp
            )

        }

        Box {
            LazyColumn(
                state = lazyColumnState,
                modifier = Modifier
                    .padding(end = 12.dp)
            ) {
                algorithmTableViewModel.value.fillTable()
                for ((vertex, result) in mapOfResult.value) {
                item {
                    TableItem(
                        vertex,
                        result,
                        numberCardWeight,
                        resultCardWeight
                    )
                }
                }
            }
            VerticalScrollbar(
                modifier = Modifier
                    .align(Alignment.TopEnd)
                    .fillMaxHeight(),
                adapter = rememberScrollbarAdapter(
                    scrollState = lazyColumnState
                )

            )
        }

    }

}
```

AlgorithmTableViewModel.kt

```
package ui.InformationTables.AlgorithmTable

import androidx.compose.runtime.MutableState
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.launch
import models.interactor.AlgorithmTableInteractor
```

```
import utils.algorithm.Algorithm
import utils.algorithm.AlgorithmState

class AlgorithmTableViewModel(
   private val algorithmTableInteractor: AlgorithmTableInteractor,
      private val currentAlgorithm: MutableStateFlow<Pair<Algorithm,
AlgorithmState>>,
   private val algorithmTableState: MutableState<Map<String, String>>
) {
   fun fillTable() {
       CoroutineScope(Dispatchers.Main).launch {
           currentAlgorithm.collect { data ->
               when(data.first) {
                   Algorithm.ALG_TOP_SORT -> {
                                                    if (data.second ==
AlgorithmState.IN_PROGRESS_AUTO) { // Или лучше AlgorithmState.FINISH?

algorithmTableInteractor.updateGraphTopSortInfo(algorithmTableState)
//                                                  currentAlgorithm.value =
Pair(Algorithm.NONE, AlgorithmState.NONE)
                       }
                   }
                   else -> {}
               }
           }
       }
   }
}
```

InformationTables.kt

```
package ui.InformationTables

import androidx.compose.desktop.ui.tooling.preview.Preview
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.width
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import kotlinx.coroutines.flow.MutableStateFlow
import ui.InformationTables.ActionTable.ActionTable
import ui.InformationTables.AlgorithmTable.AlgorithmTable
import utils.algorithm.Algorithm
import utils.GraphToolsState
import utils.algorithm.AlgorithmState

@Composable
@Preview
fun InformationTables(
   graphToolsStateFlow: MutableStateFlow<GraphToolsState>,
   currentAlgorithm: MutableStateFlow<Pair<Algorithm, AlgorithmState>>,
   modifier: Modifier = Modifier
) {

   Row(
       modifier = modifier
```

```
    ) {
        AlgorithmTable(
            currentAlgorithm = currentAlgorithm,
            modifier = Modifier
                .weight(45f)
        )
        Spacer(Modifier
            .width(16.dp)
        )
        ActionTable(
            currentAlgorithm = currentAlgorithm,
            modifier = Modifier
                .weight(100f)
        )
    }
}
```

TableItem.kt

```
package ui.InformationTables

import androidx.compose.foundation.border
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.height
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp


@Composable
fun TableItem(
    firstElem: String,
    secondElem: String,
    firstCardWeight: Float,
    secondCardWeight: Float
) {
    Row(
        horizontalArrangement = Arrangement.spacedBy(8.dp)
    ) {
        Text(
            modifier = Modifier
                .weight(firstCardWeight)
                .height(30.dp)
                .border(
                    width = 1.dp,
                    color = Color.Black
                ),
            text = firstElem
        )
        Text(
            modifier = Modifier
                .weight(secondCardWeight)
                .height(30.dp)
                .border(
```

```
                    width = 1.dp,
                    color = Color.Black
                ),
            text = secondElem
        )
    }
}
```

MainScreen.kt

```
// Copyright 2000-2021 JetBrains s.r.o. and contributors. Use of this
source code is governed by the Apache 2.0 license that can be found in
the LICENSE file.
import androidx.compose.desktop.ui.tooling.preview.Preview
import androidx.compose.foundation.layout.*
import androidx.compose.material.MaterialTheme
import androidx.compose.material.Scaffold
import androidx.compose.runtime.Composable
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.DpSize
import androidx.compose.ui.unit.dp
import androidx.compose.ui.window.WindowState
import androidx.compose.ui.window.singleWindowApplication
import ui.GraphEditor.GraphEditor
import ui.InformationTables.InformationTables
import ui.MainScreen.MainScreenViewModel
import ui.TopBarElement.MainScreenTopBar
import ui.UserActionHint.UserActionHint

@Composable
@Preview
fun MainScreen() {
   MaterialTheme{
                        val    mainScreenViewModel    =    remember    {
mutableStateOf(MainScreenViewModel()) }
       val enterFileNameState = remember { mutableStateOf(false) }
       val errorFileDialog = remember { mutableStateOf(false) }
       Scaffold(
           topBar = {
               MainScreenTopBar(
                   onClickLoadGraphButton = {

mainScreenViewModel.value.prepareAlertDialogForLoadData(enterFileNameSt
ate, errorFileDialog)

mainScreenViewModel.value.showEnterFileDialog(enterFileNameState)
                   },
                   onClickSaveGraphButton = {

mainScreenViewModel.value.prepareAlertDialogForSaveData(enterFileNameSt
ate)

mainScreenViewModel.value.showEnterFileDialog(enterFileNameState)
                   }
               )
```

```
            }
        ) {
            if (enterFileNameState.value) {

mainScreenViewModel.value.EnterFileNameAlertDialogFactory()
            }
            if (errorFileDialog.value) {
                                mainScreenViewModel.value.ErrorFileDialog  {
errorFileDialog.value = false }
            }
            MainContent(mainScreenViewModel.value)
        }
    }
}
@Composable
fun MainContent(mainScreenViewModel: MainScreenViewModel) {
    Column {
        UserActionHint(
            graphToolsStateFlow = mainScreenViewModel.graphToolsState,
            modifier = Modifier
                .fillMaxWidth()
                .padding(16.dp)
        )
        GraphEditor(
            graphToolsStateFlow = mainScreenViewModel.graphToolsState,
            currentAlgorithm = mainScreenViewModel.currentAlgorithm,
            modifier = Modifier
                .fillMaxWidth()
                .fillMaxHeight(0.75f)
                .padding(vertical = 8.dp)
                .padding(start = 8.dp)
        )
        InformationTables(
            graphToolsStateFlow = mainScreenViewModel.graphToolsState,
            currentAlgorithm = mainScreenViewModel.currentAlgorithm,
            modifier = Modifier
                .fillMaxWidth()
                .defaultMinSize(minHeight = 50.dp, minWidth = 150.dp)
                .padding(8.dp)
        )
    }
}


fun main() = singleWindowApplication(
    title = "Top Sort",
    state = WindowState(
        size = DpSize(1200.dp, 1000.dp),
        isMinimized = true
    )
) {
    MainScreen()
}
```

MainScreenViewModel.kt

```
package ui.MainScreen
```

```
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.size
import androidx.compose.material.*
import androidx.compose.runtime.*
import androidx.compose.ui.Modifier
import androidx.compose.ui.focus.FocusRequester
import androidx.compose.ui.focus.focusRequester
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import data.graphData.DataGraphLocator
import kotlinx.coroutines.flow.MutableStateFlow
import utils.GraphToolsState
import utils.algorithm.Algorithm
import utils.algorithm.AlgorithmState

class MainScreenViewModel {
         val   graphToolsState:   MutableStateFlow<GraphToolsState>   =
MutableStateFlow(GraphToolsState.WAITING)
            val    currentAlgorithm:    MutableStateFlow<Pair<Algorithm,
AlgorithmState>> = MutableStateFlow(
       Pair(Algorithm.NONE, AlgorithmState.NONE))

   // EnterFileNameAlertDialog
   private var alertDialogTitle: String = ""
   private var alertDialogLabel: String = ""
   private var confirmButtonAction: @Composable (String) -> Unit = {}
   private var dismissButtonAction: @Composable () -> Unit = {}
   private var onDismissAction: () -> Unit = {}

   @OptIn(ExperimentalMaterialApi::class)
   @Composable
   fun ErrorFileDialog(
       onDismissAction: () -> Unit
   ) {
       AlertDialog(
           modifier = Modifier
               .size(width = 350.dp, height = 50.dp),
           onDismissRequest = onDismissAction,
           text = {
               Text(
                   modifier = Modifier
                       .fillMaxSize(),
                   text = "Недопустимый формат данных",
                   fontWeight = FontWeight.Bold,
                   textAlign = TextAlign.Center
               )
           },
           buttons = {}
       )
   }


   @OptIn(ExperimentalMaterialApi::class)
   @Composable
   fun EnterFileNameAlertDialogFactory(
```

```
        alertDialogTitle: String = this.alertDialogTitle,
        alertDialogLabel: String = this.alertDialogLabel,
                confirmButtonAction: @Composable (String) -> Unit =
this.confirmButtonAction,
                    dismissButtonAction: @Composable () -> Unit =
this.dismissButtonAction,
        onDismissAction: () -> Unit = this.onDismissAction,
    ) {
        val nameFile = remember { mutableStateOf("") }
        AlertDialog(
            title = { Text(alertDialogTitle) },
            onDismissRequest = {
                onDismissAction()
            },
            confirmButton = {
                confirmButtonAction(nameFile.value)
            },
            dismissButton = {
                dismissButtonAction()
            },
            text = {
                val focusRequester = remember { FocusRequester() }
                TextField(
                    modifier = Modifier.focusRequester(focusRequester),
                    value = nameFile.value,
                    label = { Text(alertDialogLabel) },
                    onValueChange = { name ->
                        nameFile.value = name
                    }
                )
                LaunchedEffect(Unit) {
                    focusRequester.requestFocus()
                }
            }
        )
    }

    fun prepareAlertDialogForLoadData(
        enterFileNameState: MutableState<Boolean>,
        errorFileDialog: MutableState<Boolean>
    ) {
        alertDialogTitle = "Введите имя файла для загрузки"
        alertDialogLabel = "Имя файла (без расширения)"
        confirmButtonAction = { nameFile ->
            Button(
                onClick = {

if(!DataGraphLocator.readGraphData("$nameFile.json"))
                        errorFileDialog.value = true
                    enterFileNameState.value = false
                }
            ) {
                Text("Загрузить")
            }
        }
        dismissButtonAction = {
            Button(
```

```
                onClick = {
                    enterFileNameState.value = false
                }
            ) {
                Text("Отмена")
            }
        }
        onDismissAction = {
            enterFileNameState.value = false
        }
    }

                fun     prepareAlertDialogForSaveData(enterFileNameState:
MutableState<Boolean>) {
        alertDialogTitle = "Введите имя файла для сохранения"
        alertDialogLabel = "Имя файла (без расширения)"
        confirmButtonAction = { nameFile ->
            Button(
                onClick = {
                    DataGraphLocator.saveGraphData("$nameFile.json")
                    enterFileNameState.value = false
                }
            ) {
                Text("Сохранить")
            }
        }
        dismissButtonAction = {
            Button(
                onClick = {
                    enterFileNameState.value = false
                }
            ) {
                Text("Отмена")
            }
        }
        onDismissAction = {
            enterFileNameState.value = false
        }
    }

    fun showEnterFileDialog(enterFileNameState: MutableState<Boolean>) {
        enterFileNameState.value = true
    }

}
```

TopBarElement.kt

```
package ui.TopBarElement

import androidx.compose.desktop.ui.tooling.preview.Preview
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Row
import androidx.compose.material.Button
import androidx.compose.material.Text
import androidx.compose.material.TopAppBar
import androidx.compose.runtime.Composable
```

```
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp

@Composable
@Preview
fun MainScreenTopBar(
   onClickLoadGraphButton: () -> Unit,
   onClickSaveGraphButton: () -> Unit,
   modifier: Modifier = Modifier
) {
   TopAppBar(
       modifier = modifier
   ) {
       Row(
           horizontalArrangement = Arrangement.spacedBy(16.dp)
       ) {
           Button(onClick = {
               onClickLoadGraphButton()
           }) {
               Text("Загрузить")
           }
           Button(onClick = {
               onClickSaveGraphButton()
           }) {
               Text("Сохранить")
           }
       }
   }
}
```

UserActionHint.kt

```
package ui.UserActionHint

import androidx.compose.desktop.ui.tooling.preview.Preview
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.style.TextAlign
import utils.GraphToolsState
import kotlinx.coroutines.flow.MutableStateFlow

@Composable
@Preview
fun UserActionHint(
   graphToolsStateFlow: MutableStateFlow<GraphToolsState>,
   modifier: Modifier = Modifier
) {
   val title = remember { mutableStateOf("") }
               val     userActionHintViewModel     =     remember     {
mutableStateOf(UserActionHintViewModel()) }
       userActionHintViewModel.value.subscribeTitleToEditorState(title,
graphToolsStateFlow)
```

```
    Text(
        text = title.value,
        modifier = modifier,
        style = TextStyle(
            color = Color.Gray
        ),
        textAlign = TextAlign.Center

    )

}
```

UserActionHintViewModel.kt

```
package ui.UserActionHint

import androidx.compose.runtime.MutableState
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.launch
import utils.GraphToolsState

class UserActionHintViewModel {
    private fun getTitle(graphToolsState: GraphToolsState): String =
        when (graphToolsState) {
            GraphToolsState.WAITING -> "Пояснение: режим ожидания"
                   GraphToolsState.SET_VERTEX  -> "Пояснение: добавление
вершины"
                  GraphToolsState.REMOVE_VERTEX  -> "Пояснение: удаление
вершины"
             GraphToolsState.SET_EDGE_FIRST -> "Пояснение: выбор вершины,
откуда будет выходить ребро"
                   GraphToolsState.SET_EDGE_SECOND  -> "Пояснение: выбор
вершины, куда будет входить ребро"
                GraphToolsState.REMOVE_EDGE_FIRST -> "Пояснение: удаление
ребра, выбор первой вершины"
                GraphToolsState.REMOVE_EDGE_SECOND -> "Пояснение: удаление
ребра, выбор второй вершины"
              GraphToolsState.TO_BEGIN -> "Пояснение: перемотка в начало
алгоритма"
               GraphToolsState.TO_FINISH -> "Пояснение: перемотка в конец
алгоритма"
                   GraphToolsState.CONTINUE  -> "Пояснение: визуализация
продолжается"
                     GraphToolsState.PAUSE  -> "Пояснение:  визуализация
приостановлена"
            else -> ""
        }
         fun   subscribeTitleToEditorState(title:   MutableState<String>,
graphToolsStateFlow: MutableStateFlow<GraphToolsState>) {
        CoroutineScope(Dispatchers.Main).launch {
            graphToolsStateFlow.collect { state ->
                title.value = getTitle(state)
            }
        }
```

```
    }
}
```

State.kt

```
package utils.actions.State

data class State(
    val time: String,
    val action: String,
    val srcVertex: Long,
    val dstVertex: Long? = -1
)
```

Actions.kt

```
package utils.actions

enum class Actions {
    DOWN_TO_EDGE, // спуск по ребру
    ADDED_TO_STACK, // вершина добавлена в стэк
    GET_ORDER // вершина получила порядок

}
```

Algorithm.kt

```
package utils.algorithm

enum class Algorithm {
    NONE,
    ALG_TOP_SORT
}
```

AlgorithmProtocol.kt

```
package utils.algorithm

data class AlgorithmProtocol(
    private val listOfActions: List<Pair<String, String>>
) {
    fun getCountActions() = listOfActions.size

        fun  getListActions(range:  IntRange  =  listOfActions.indices):
List<Pair<String, String>> {
        val listActions = mutableListOf<Pair<String, String>>()
        for (index in range) {
            listActions.add(listOfActions[index])
        }
        return listActions
    }
}
```

AlgorithmState.kt

```
package utils.algorithm

enum class AlgorithmState {
   NONE,
   START,
   IN_PROGRESS_USER,
   IN_PROGRESS_AUTO,
   PAUSE,
   FINISH
}
```

AlgorithmVisualiser.kt

```
package utils.algorithm

import utils.actions.State.State
import androidx.compose.ui.graphics.Color
import data.`object`.Vertex
import kotlinx.coroutines.*
import kotlinx.coroutines.flow.MutableStateFlow
import org.jetbrains.skiko.currentNanoTime
import ui.GraphEditor.GraphCanvas.VertexVO
import java.util.ArrayList

object AlgorithmVisualiser {
   private var algorithmProtocolPosition: Int = -1
   var algorithmProtocol: AlgorithmProtocol? = null
   set(value) {
       field = value
       algorithmProtocolPosition = 0
   }
      val graphCanvasData: MutableStateFlow<Map<Long, (VertexVO) ->
VertexVO>> = MutableStateFlow(mapOf())
    private var listOfCanvasVisualise: Map<Int, Map<Long, (VertexVO) ->
VertexVO>> = mapOf()
     val resultTableData: MutableStateFlow<List<Pair<Vertex, Int>>> =
MutableStateFlow(listOf())
    val actionsTableData: MutableStateFlow<List<Pair<String, String>>> =
MutableStateFlow(
       listOf()
   )
   var algorithmState: AlgorithmState = AlgorithmState.NONE
   var visualiseJob: Job? = null
   const val defaultPeriod = 500L

   fun startVisualiseTopSortAlgorithm(periodMs: Long) {
       algorithmState = AlgorithmState.IN_PROGRESS_AUTO
       visualiseJob = autoVisualisationJob(periodMs)
   }

   private fun autoVisualisationJob(periodMs: Long) =
       CoroutineScope(Dispatchers.Default).launch {
           var currentTimeMs = currentNanoTime() / 1000000
                   while (isActive && algorithmProtocolPosition + 1 <
(algorithmProtocol?.getCountActions() ?: 0)) {
```

```
                    while (isActive && currentNanoTime() / 1000000 -
currentTimeMs < periodMs) { }
                stepNext()
                currentTimeMs = currentNanoTime() / 1000000
            }
        }

    fun pauseVisualise() {
        visualiseJob?.cancel()
    }

    fun stopVisualise() {
        pauseVisualise()
        algorithmProtocol = null
        algorithmProtocolPosition = -1
        graphCanvasData.value = mapOf()
        listOfCanvasVisualise = mapOf()
        resultTableData.value = listOf()
        actionsTableData.value = listOf()
        algorithmState = AlgorithmState.NONE
    }

    fun stepNext() {
        if (checkProtocolSize()) return
                        if    (algorithmProtocolPosition    +    1    <
algorithmProtocol!!.getCountActions())
            algorithmProtocolPosition++
        updateAlgorithmVisualiserState()
    }

    fun stepBack() {
        if(checkProtocolSize()) return
        if (algorithmProtocolPosition - 1 >= 0)
            algorithmProtocolPosition--
        updateAlgorithmVisualiserState()
    }

    fun toFinish() {
        if (checkProtocolSize()) return
        pauseVisualise()
                                        algorithmProtocolPosition      =
(algorithmProtocol?.getCountActions() ?: 0) - 1
        updateAlgorithmVisualiserState()
    }
    fun toBegin() {
        if (checkProtocolSize()) return
        algorithmProtocolPosition = 0
        updateAlgorithmVisualiserState()
    }

    private fun checkProtocolSize(): Boolean {
        if ((algorithmProtocol?.getCountActions() ?: -1) <= 0)
            return true
        return false
    }

    fun loadResult(
```

```
        topSortResult: Map<Vertex, Int>,
        protocol: ArrayList<State>
    ) {
        algorithmProtocol = AlgorithmProtocol(
                 listOfActions = protocol.mapIndexed { index, state ->
Pair("${index+1}) time: ${state.time} ms", state.action) }
        )
               resultTableData.value = topSortResult.map { result ->
Pair(result.key, result.value) }
        var index = 0
        listOfCanvasVisualise = protocol.associate { state ->
            index++ to mapOf(
                     state.srcVertex to { vertex -> vertex.copy(color =
if((state.dstVertex ?: -1) == -1L) Color.Green else Color.Yellow) },
                (state.dstVertex ?: -1) to { vertex -> vertex.copy(color
= Color.Red) }
            )
        }
        algorithmState = AlgorithmState.IN_PROGRESS_AUTO
        updateAlgorithmVisualiserState()
    }

    private fun updateAlgorithmVisualiserState() {
              if ((algorithmProtocol?.getListActions()?.size ?: 0) <=
algorithmProtocolPosition)
            return
                                              actionsTableData.value      =
algorithmProtocol?.getListActions(0..algorithmProtocolPosition)      ?:
listOf()
                                               graphCanvasData.value      =
listOfCanvasVisualise[algorithmProtocolPosition] ?: mapOf()
    }
}
```

GraphAlgorithm.kt

```
package utils.algorithm

import data.`object`.Graph
import data.`object`.Vertex
import java.util.*
import kotlin.collections.ArrayList
import utils.actions.Actions
import utils.actions.State.State
import org.jetbrains.skiko.currentNanoTime
import org.jetbrains.skia.Point

object GraphAlgorithm {
    private var startTime = 0L
    private fun addState(
        srcVertex: Vertex, dstVertex: Vertex? = null, action: Actions,
        protocol: ArrayList<State>
    ) {
        when (action) {
            Actions.DOWN_TO_EDGE -> protocol.add(
                State(
                    getCurrentTime(),
```

```
                        "Down from vertex ${srcVertex.getName()} (id =
${srcVertex.getId()})     to     ${dstVertex?.getName()}     (id     =
${dstVertex?.getId()})",
                    srcVertex.getId(),
                    dstVertex?.getId()
                )
            )

            Actions.GET_ORDER -> protocol.add(
                State(
                    getCurrentTime(),
                                "Vertex  ${srcVertex.getName()}  (id  =
${srcVertex.getId()}) got order",
                    srcVertex.getId()
                )
            )

            Actions.ADDED_TO_STACK -> protocol.add(
                State(
                    getCurrentTime(),
                                "Vertex  ${srcVertex.getName()}  (id  =
${srcVertex.getId()}) added to stack",
                    srcVertex.getId()
                )
            )
        }
    }

    private fun getCurrentTime(): String {
        return (currentNanoTime() / 1000000 - startTime).toString()
    }

    fun checkGraphForCycle(graph: Graph) : Boolean {
        val vertexes = graph.getVertexes()
        val visited: ArrayList<Vertex> = arrayListOf()
        val checkList: ArrayList<Boolean> = arrayListOf()
        for (vertex in vertexes) {
            if (vertex !in visited) {
                        checkGraphForCycleUtil(graph,  vertex,  visited,
checkList)
            }
        }
        return (false in checkList)
    }

    private fun checkGraphForCycleUtil(
        graph: Graph,
        current: Vertex,
        visited: ArrayList<Vertex>,
        checkList: ArrayList<Boolean>
    ) {
        visited.add(current)
        val edges = initEdges(graph, current)
        for (edge in edges.values) {
            if (edge !in visited) {
                checkGraphForCycleUtil(graph, edge, visited, checkList)
            } else {
```

```
                checkList.add(false)
            }
        }
        visited.remove(current)
        checkList.add(true)
    }

      private fun initEdges(graph: Graph, current: Vertex):
MutableMap<Long, Vertex> {
        val vertexes = graph.getVertexes()
        val edges: MutableMap<Long, Vertex> = mutableMapOf()
        for (vertex in vertexes) {
            if (vertex.getId() in current.getEdges())
                edges[vertex.getId()] = vertex
        }
        return edges
    }

   private fun TopSortUtil(graph: Graph, current: Vertex, visited:
ArrayList<Vertex>, stackOfVertexes: Stack<Vertex>) {
        visited.add(current)
        val edges: MutableMap<Long, Vertex> = initEdges(graph, current)
        for (edge in edges.values) {
            if (edge !in visited) {
                this.TopSortUtil(graph, edge, visited, stackOfVertexes)
            }
        }
        stackOfVertexes.add(current)
    }

    fun TopSort(graph: Graph): Map<Vertex, Int> {
        val stackOfVertexes: Stack<Vertex> = Stack()
        val vertexes = graph.getVertexes()
        val visited: ArrayList<Vertex> = arrayListOf()
        if (checkGraphForCycle(graph)) return mapOf()
        for (vertex in vertexes) {
            if (vertex !in visited) {
                TopSortUtil(graph, vertex, visited, stackOfVertexes)
            }
        }
        var order = 0
        val result: MutableMap<Vertex, Int> = mutableMapOf()
        while (stackOfVertexes.isNotEmpty()) {
            result[stackOfVertexes.pop()] = order++
        }
        return result
    }


    private fun TopSortUtilActions(
        graph: Graph, current: Vertex, visited: ArrayList<Vertex>,
        stackOfVertexes: Stack<Vertex>, protocol: ArrayList<State>
    ) {
        visited.add(current)
        val edges: MutableMap<Long, Vertex> = initEdges(graph, current)
        for (edge in edges.values) {
            if (edge !in visited) {
```

```
                addState(current, edge, Actions.DOWN_TO_EDGE, protocol)
                        this.TopSortUtilActions(graph, edge, visited,
stackOfVertexes, protocol)
            }
        }
        stackOfVertexes.add(current)
          addState(current, action = Actions.ADDED_TO_STACK, protocol =
protocol)
    }

         fun TopSortActions(graph: Graph): Pair<Map<Vertex, Int>,
ArrayList<State>> {
        val stackOfVertexes: Stack<Vertex> = Stack()
        val protocol: ArrayList<State> = arrayListOf()
        val vertexes = graph.getVertexes()
        startTime = setStartAlgoTime()
        val visited: ArrayList<Vertex> = arrayListOf()
        if (checkGraphForCycle(graph)) return Pair(
            mapOf(),
             arrayListOf(State(getCurrentTime(), "Cycle was found, create
another graph", 0))
        )
        for (vertex in vertexes) {
            if (vertex !in visited) {
                            TopSortUtilActions(graph, vertex, visited,
stackOfVertexes, protocol)
            }
        }
        var order = 0
        val result: MutableMap<Vertex, Int> = mutableMapOf()
        while (stackOfVertexes.isNotEmpty()) {
            val current = stackOfVertexes.pop()
            result[current] = order++
               addState(current, action = Actions.GET_ORDER, protocol =
protocol)
        }
        return Pair(result, protocol)
    }

    private fun setStartAlgoTime() = currentNanoTime() / 1000000
}
```

Parser.kt

```
package utils.parsing

import com.google.gson.Gson
import com.google.gson.GsonBuilder
import data.`object`.Graph
import data.`object`.Vertex
import org.jetbrains.skia.Point
import ui.GraphEditor.GraphCanvas.VertexVO
import utils.getDistTo
import java.io.File

object Parser {
    fun readDataJSON(filePath: String): Graph {
```

```
        val builder = GsonBuilder()
        val gson = builder.create()
        var graph = Graph(arrayListOf())
        val text = File(filePath).readText()
        graph = gson.fromJson(text, graph.javaClass)
        if (!checkValidVertexes(graph)) throw Exception()
        if (!checkGraphForValid(graph)) throw Exception()
        return graph
    }

    private fun checkValidVertexes(graph: Graph?): Boolean {
        if (graph?.getVertexes() == null) {
            return false
        }
        for (vertex in graph.getVertexes()) {
            if (vertex.getId() == null ||
                    vertex.getName() == null ||
                    vertex.getCenter() == null ||
                    vertex.getEdges() == null) {
                return false
            }
        }
        return true
    }

    fun writeDataJSON(filePath: String, graph: Graph){
        val gson = Gson()
        val text = gson.toJson(graph)
        val file = File(filePath)
        file.writeText(text)
    }

    fun checkGraphForValid(graph: Graph) : Boolean {
        // проверка на одинаковые id
        if (!checkDifferentIds(graph.getVertexes())) return false

        // проверка на несуществующие рёбра
        if (!checkAvailableVertexes(graph.getVertexes())) return false

        // проверка на неотрицательность координат
            if(!checkAllCoordsGreaterZeroAndRadius(graph.getVertexes()))
return false

        // проверка на то, что вершины не перекрывают друг друга
        if (!checkNonIntersectionOfVertexes(graph.getVertexes())) return
false

        return true
    }

    private fun checkDifferentIds(vertexes: ArrayList<Vertex>): Boolean
{
        val ids: ArrayList<Long> = getIds(vertexes)
        if (ids.toSet().size != ids.size) return false
        return true
    }
```

```
     private fun checkAvailableVertexes(vertexes: ArrayList<Vertex>):
Boolean {
       val ids: ArrayList<Long> = getIds(vertexes)
       for (vertex in vertexes) {
           val edges = vertex.getEdges()
           for (edge in edges) {
               if (edge !in ids) return false
           }
       }
       return true
   }

          private fun checkAllCoordsGreaterZeroAndRadius(vertexes:
ArrayList<Vertex>): Boolean {
       val centers: ArrayList<Point> = getCenters(vertexes)
       for (center in centers) {
                   if (center.x <= VertexVO.radius || center.y <=
VertexVO.radius) return false
       }
       return true
   }

             private fun checkNonIntersectionOfVertexes(vertexes:
ArrayList<Vertex>): Boolean {
       val centers: ArrayList<Point> = getCenters(vertexes)
       for ((i, centerSrc) in centers.withIndex()) {
           for (j in i + 1 until centers.size) {
               if (i != j) {
                                   if (centerSrc.getDistTo(centers[j]) <
VertexVO.radius) return false
               }
           }
       }
       return true
   }

   private fun getIds(vertexes: ArrayList<Vertex>): ArrayList<Long> {
       val ids: ArrayList<Long> = arrayListOf()
       for (i in 0 until vertexes.size) {
           ids.add(vertexes[i].getId())
       }
       return ids
   }

           private fun getCenters(vertexes: ArrayList<Vertex>):
ArrayList<Point> {
       val centers: ArrayList<Point> = arrayListOf()
       for (i in 0 until vertexes.size) {
           centers.add(vertexes[i].getCenter())
       }
       return centers
   }
}
```

CanvasDrawLib.kt

```
package utils
```

```
import androidx.compose.ui.geometry.Offset
import androidx.compose.ui.graphics.*
import models.mapper.toOffset
import org.jetbrains.skia.Font
import org.jetbrains.skia.Point
import ui.GraphEditor.GraphCanvas.VertexVO
import kotlin.math.abs
import kotlin.math.cos
import kotlin.math.sin

object CanvasDrawLib {

   fun drawVertex(canvas: Canvas, vertex: VertexVO, font: Font) {
       canvas.drawCircle(
           radius = VertexVO.radius,
           center = vertex.center.toOffset(),
           paint = Paint().apply {
               this.color = vertex.color
               this.style = PaintingStyle.Stroke
           }
       )
            val title = if(vertex.name.length > VertexVO.cntLetters)
vertex.name.substring(0, VertexVO.cntLetters) + "…" else vertex.name
       val offset = getTextOffset(font, title)
       drawVertexTitle(canvas, title, vertex, offset, font)
   }

   fun drawEdge(canvas: Canvas, point1: Point?, point2: Point?) {
       if (point1 == null || point2 == null || point1.getDistTo(point2)
< 2 * VertexVO.radius)
           return
       val (newPoint1, newPoint2) = recalcPointsEdge(point1, point2)
       // основная линия
       canvas.drawLine(
           Offset(newPoint1.x, newPoint1.y),
           Offset(newPoint2.x, newPoint2.y),
           Paint().apply {
               color = Color.Black
           }
       )
       // добавление стрелок
       drawArrowForEdge(newPoint1, newPoint2, canvas)
   }

   private fun drawArrowForEdge(
       newPoint1: Point,
       newPoint2: Point,
       canvas: Canvas,
       angle: Float = 35f
   ) {
       val r = VertexVO.radius
       val dist = newPoint1.getDistTo(newPoint2)
        val (sign1X, sign1Y, sign2X, sign2Y) = getPointSign(newPoint1,
newPoint2)
       val newPoint1ForArrow = Point(
```

```
                newPoint2.x + sign2X * (r / dist * abs(newPoint1.x -
newPoint2.x)).toFloat(),
                newPoint2.y + sign2Y * (r / dist * abs(newPoint1.y -
newPoint2.y)).toFloat(),
        )
        // первая часть стрелки
        var newPointForArrow =
            getArrowPosition(newPoint2, newPoint1ForArrow, angle)
        canvas.drawLine(
            newPoint2.toOffset(),
            newPointForArrow,
            Paint().apply {
                color = Color.Black
            }
        )
        // вторая часть стрелки
                    newPointForArrow   =   getArrowPosition(newPoint2,
newPoint1ForArrow, -angle)
        canvas.drawLine(
            newPoint2.toOffset(),
            newPointForArrow,
            Paint().apply {
                color = Color.Black
            }
        )
    }

    private fun getPointSign(
        newPoint1: Point,
        newPoint2: Point
    ): List<Int> {
        val sign1X = if (newPoint1.x < newPoint2.x) 1 else -1
        val sign2X = sign1X * -1
        val sign1Y = if (newPoint1.y < newPoint2.y) 1 else -1
        val sign2Y = sign1Y * -1
        return listOf(sign1X, sign1Y, sign2X, sign2Y)
    }

    private fun getArrowPosition(
        newPoint2: Point,
        newPoint1ForArrow: Point,
        angle: Float
    ): Offset {
        // вектор на прямой
        val vector = Point(
            newPoint2.x - newPoint1ForArrow.x,
            newPoint2.y - newPoint1ForArrow.y
        )
        // повёрнутый вектор
        val rotatedVector = Point(
             vector.x * cos(Math.PI / 180 * angle).toFloat() + vector.y *
sin(Math.PI / 180 * angle).toFloat(),
             vector.y * cos(Math.PI / 180 * angle).toFloat() - vector.x *
sin(Math.PI / 180 * angle).toFloat()
        )
        return Offset(
            newPoint2.x - rotatedVector.x,
```

```
            newPoint2.y - rotatedVector.y,
        )
    }

        private fun recalcPointsEdge(point1: Point, point2: Point):
Pair<Point, Point> {
        val R = point1.getDistTo(point2)
        val r = VertexVO.radius
        val DeltaX = abs(point1.x - point2.x)
        val DeltaY = abs(point1.y - point2.y)
        val deltaX = r / R * DeltaX
        val deltaY = r / R * DeltaY
           val (sign1X, sign1Y, sign2X, sign2Y) = getPointSign(point1,
point2)
        return Pair(
                Point(point1.x + (deltaX * sign1X).toFloat(), point1.y +
(deltaY * sign1Y).toFloat()),
                Point(point2.x + (deltaX * sign2X).toFloat(), point2.y +
(deltaY * sign2Y).toFloat())
        )
    }

    private fun drawVertexTitle(
        canvas: Canvas,
        title: String,
        vertex: VertexVO,
        offset: Int,
        font: Font
    ) {
        canvas.nativeCanvas.drawString(
            s = title,
            x = vertex.center.x - offset,
            y = vertex.center.y + 5f,
            paint = org.jetbrains.skia.Paint().apply {
                color = Color.Black.toArgb()
            },
            font = font
        )
    }


    private fun getTextOffset(
        font: Font,
        textLabel: String
    ): Int {
        val textWidth = font.getWidths(font.getStringGlyphs(textLabel))
        var textOffset = 0
        for (i in 0 until textWidth.size / 2) {
            textOffset += textWidth[i].toInt()
        }
        if (textWidth.size % 2 == 1) {
            textOffset += textWidth[textWidth.size / 2].toInt() / 2
        }
        return textOffset
    }
}
```

ExtestionMethods.kt

```
package utils

import org.jetbrains.skia.Point
import kotlin.math.pow
import kotlin.math.sqrt

fun Point.getDistTo(other: Point): Double = sqrt(
    (this.x - other.x).toDouble().pow(2) +
            (this.y - other.y).toDouble().pow(2)
)
```

GraphToolsState.kt

```
package utils

enum class GraphToolsState {
    WAITING,
    SET_VERTEX,
    REMOVE_VERTEX,
    SET_EDGE_FIRST,
    SET_EDGE_SECOND,
    REMOVE_EDGE_FIRST,
    REMOVE_EDGE_SECOND,
    PAUSE,
    CONTINUE,
    TO_BEGIN,
    TO_FINISH,
    STEP_BACK,
    STEP_NEXT
}
```