**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по учебной практике**

Тема: **«Визуализация алгоритмов на графах на Java»**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 5381 |  | Швецова Н.М. |
| Студентка гр. 5381 |  | Орлова В.В. |
| Студент гр. 5381 |  | Терещенко К.А. |
| Руководитель |  | Чайка К.В. |

Санкт-Петербург

2017

**ЗАДАНИЕ**

**на учебную практику**

|  |  |  |
| --- | --- | --- |
| Студентка Швецова Н.М. группы 5381 | | |
| Студентка Орлова В.В. группы 5381 | | |
| Студент Терещенко К.А. группы 5381  Тема практики: «Визуализация алгоритмов на графах на Java» | | |
| Задание на практику:  Командная итеративная разработка визуализатора алгоритма(ов) на Java с графическим интерфейсом.  Алгоритм: Алгоритм Беллмана-Форда | | |
| Сроки прохождения практики: 21.06.2017 – 30.06.2017 | | |
| Дата сдачи отчета: 30.06.2017 | | |
| Дата защиты отчета: 30.06.2017 | | |
|  | | |
| Студентка |  | Швецова Н.М. |
| Студентка |  | Орлова В.В. |
| Студент |  | Терещенко К.А. |
| Руководитель |  | Чайка К.В. |

**Аннотация**

По зданию требуется разработать программу на языке Java, визуализирующую алгоритм работы Форда-Беллмана для ориентированных графов. При этом должен присутствовать графический интерфейс. Данный алгоритм производит поиск кратчайшего пути во взвешенном графе и демонстрирует подробную работу алгоритма.

**Summary**

The building needs to develop a Java program that visualizes the Ford-Bellman algorithm for oriented graphs. There must be a graphical interface. This algorithm searches for the shortest path in a weighted graph and demonstrates the detailed operation of the algorithm.

**содержание**

Оглавление

[введение 5](#_Toc486585714)

[1. Спецификация программы 6](#_Toc486585715)

[1.1 - Ввод\вывод: 6](#_Toc486585716)

[1.2 Визуализация-Интерфейс: 7](#_Toc486585717)

[2. План разработки и распределение ролей в бригаде 7](#_Toc486585718)

[2.1. План разработки 7](#_Toc486585719)

[2.2. Распределение ролей в бригаде 8](#_Toc486585720)

[3. Особенности реализации 9](#_Toc486585721)

[3.1. Описание программного интерфейса 9](#_Toc486585722)

[4.Тестирование 12](#_Toc486585723)

[заключение 18](#_Toc486585724)

[Приложение А 19](#_Toc486585725)

# введение

*Формулировка задания:*

Требуется разработать программу, визуализирующую алгоритм работы Форда-Беллмана для ориентированных неотрицательных графов. При этом должен присутствовать графический интерфейс.

*Описание алгоритма*

Алгоритм Беллмана-Форда – алгоритм поиска кратчайшего пути во взвешенном графе. За время O(|V| × |E|) алгоритм находит кратчайшие пути от одной вершины графа до всех остальных.

# 1. Спецификация программы

# 1.1 - Ввод\вывод:

В данном проекте работа ведется с графами. Собственно, для этого требуется иметь граф. Конструирование графов будет представлено 3-мя способами:

1. Случайная генерация ориентированного графа с заданием необходимых параметров. При нажатии на определенную кнопку будет появляться окно, где потребуется ввести число вершин для будущего графа.
2. Создание графа с возможностью задавать фиксированный вес ребра. При нажатии на определенную кнопку будет доступна область активного поля, в котором можно будет создавать вершины по двойному нажатию ЛКМ. Если пользователь захочет создать ребро, ему понадобиться выделить последовательно две вершины (порядок имеет значение, ребро будет направлено из 1-й вершины во 2-ю), после чего откроется окно с просьбой задать вес создаваемого ребра, куда нужно будет ввести нужное значение (вес ребра).
3. Создание графа вручную пользователем. Данная реализация будет представлена следующим образом: пользователь может кликать в некоторую область активного поля, там будет появляться вершина со своим номером. При нажатии на одну вершину левой кнопкой мыши и затем на другую, будет воссоздано ребро из 1-й вершины во 2-ю. Вес ребер геометрически зависим.

Фактически выходные данные отсутствуют. Собирается граф, который визуально отображается на экране, где пошагово мы можем просмотреть работу алгоритма Форда-Беллмана. После чего работа завершается.

# 1.2 Визуализация-Интерфейс:

При запуске проекта будет создаваться отдельное окно. Данное окно будет иметь 2 поля:

1. Поле графического представления графа.
2. Поле интерфейса создания графа:
3. Фиксированные ребра. Более подробное описание в разделе ввода данных.
4. Случайная генерация (открывается окошко для ввода числа ребер и числа вершин).
5. Добавление вершин и ребер вручную. Более подробное описание в разделе ввода данных.
6. Готово.
7. Удалить.
8. Очистить.
9. Сл.шаг.
10. Стартовая вершина.
11. Показать результат.

# 2. План разработки и распределение ролей в бригаде

# 2.1. План разработки

Понедельник (26.06.2016) Псевдокод алгоритма Форда-Беллмана.

Среда (28.06.2016) Бета-версия. Предоставление частичной функциональности работы проекта (реализация не всех способов ввода графа).

Пятница (30.06.2016) Конечный проект. Демоверсия. Предоставление проекта с полной функциональностью, с реализованными всеми способами ввода графа. Возможно с некоторыми недочетами.

Понедельник(02.07.2016) Конечный проект. Готовая для сдачи версия.

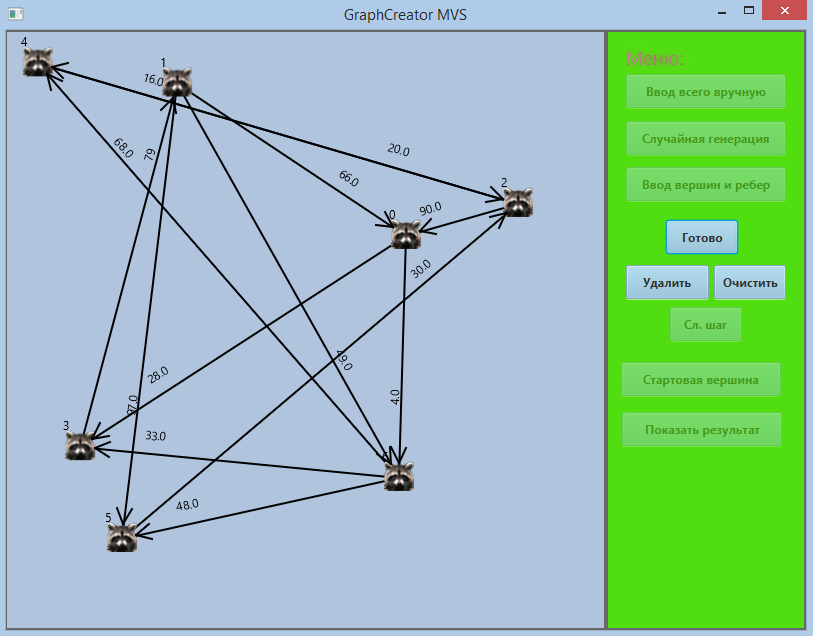
Текущий план представляет собой крайние сроки(deadline) для сдачи определенной функциональности проекта. Возможно данные сроки будут корректироваться и проект будет написан раньше срока, так как реализация определенной функциональности будет проходить параллельно.

# 2.2. Распределение ролей в бригаде

**Швецова**: работа с алгоритмом Форда-Беллмана (описание, реализация). Реализация случайной генерация графа. Реализация отчета.  
**Орлова**: разработка графического интерфейса, создание прототипа, добавление функциональности к элементам графики, пошаговый алгоритм Форда-Беллмана. Реализация ввода графа вручную.   
**Терещенко**: разработка структуры данных ребра (Edge) и вершины (Vertex). Добавление функциональности к элементам графики.

# 3. Особенности реализации

# 3.1. Описание программного интерфейса

*Программный интерфейс. Рис. 1.*

Программа реализована на языке Java с использованием Фреймворка JavaFX. С помощью пользовательского интерфейса можно выполнять следующие действия:

Выбор метода ввода графа при нажатии на соответствующую кнопку.

* + Случайная генерация ориентированного графа с заданием необходимых параметров. При нажатии на определенную кнопку будет появляться окно, где потребуется ввести число вершин для будущего графа.
    - Удалить ребро (выделить ребро, нажать на соответствующую кнопку “Удалить”)
    - Удалить вершину (выделить вершину и нажать на соответствующую кнопку “Удалить”)
    - Очистить рабочую область (нажать на кнопку “Очистить”)
    - Запустить алгоритм Беллмана-Форда для созданного графа (для начала выбрать стартовую вершину («Стартовая вершина»), затем либо последовательно нажимать «Сл.шаг», либо «Показать результат»)
    - Перейти к следующему шагу работы алгоритма (нажать на кнопку “Сл.шаг”)
    - Показать конечный результат без последовательного прохода графа (нажать на кнопку “Показать результат”)
  + Создание графа с возможностью задавать вес ребра.

При нажатии на определенную кнопку будет доступна область активного поля, в котором можно будет создавать вершины по двойному нажатию ЛКМ. Если пользователь захочет создать ребра ему понадобиться выделить два ребра после чего откроется окно с просьбой задать вес создаваемого ребра, куда нужно будет ввести нужное значение (вес ребра).

* + - * Поставить вершину графа (двойной клик ЛКМ на пустом месте рабочей области).
      * Выделить вершину (нажать ЛКМ по вершине)
      * Соединить вершины ребрами (выделить две вершины которые хотим соединить)
      * Удалить ребро (выделить ребро, нажать на соответствующую кнопку “Удалить”)
      * Удалить вершину (выделить вершину и нажать на соответствующую кнопку “Удалить”)
      * Очистить рабочую область (нажать на кнопку “Очистить”)
      * Запустить алгоритм Беллмана-Форда для созданного графа (нажать на кнопку “Начать”)
      * Перейти к следующему шагу работы алгоритма (нажать на кнопку “Сл.шаг”)
      * Показать конечный результат без последовательного прохода графа (нажать на кнопку “Показать результат”)
  + Создание графа вручную пользователем(вес рёбер задаётся автоматически)

Данная реализация будет представлена следующим образом: пользователь может кликать в некоторую область активного поля, там будет появляться вершина со своим номером. При нажатии на одну вершину левой кнопкой мыши и затем на другую, будет воссоздано ребро из 1-й вершины во 2-ю. Вес ребер геометрически зависим.

* + - * Действия над вершинами и рёбрами в рабочей области аналогичны, как с в создании графа с возможностью задания веса ребра (поставить вершину графа; выделить вершину; cоединить вершины ребрами; удалить ребро; удалить вершину; очистить рабочую область; запустить алгоритм Беллмана-Форда для созданного графа; перейти к следующему шагу работы алгоритма; показать конечный результат без последовательного прохода графа)
  + По мере работы алгоритма ребра могут окрашиваться в 4 цвета:
* Красный
* Синий
* Оранжевый
* Зеленый

Красный может обозначать 2 вещи: 1) Это ребро выделено 2) Это ребро является частью кратчайшего Пути.

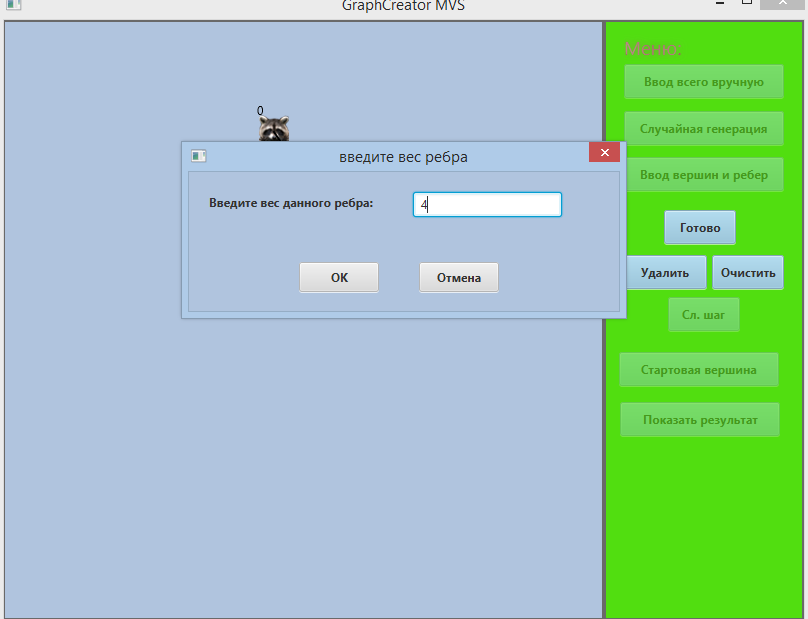
Синий значит, что при работе алгоритма с данным ребром ослабление пути не произошло.

Оранжевый значит, что не существует пути от стартовой вершины к данному ребру.

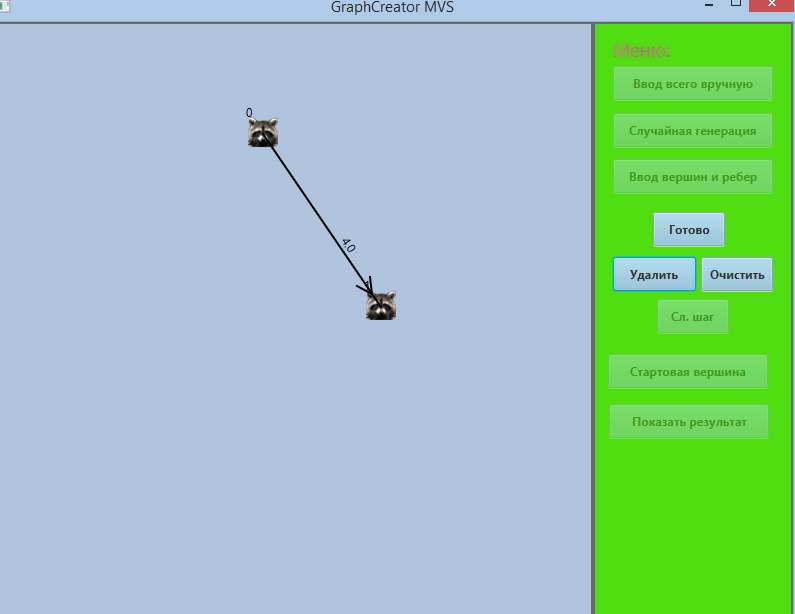
Зеленый - произошло ослабление.

# 4.Тестирование

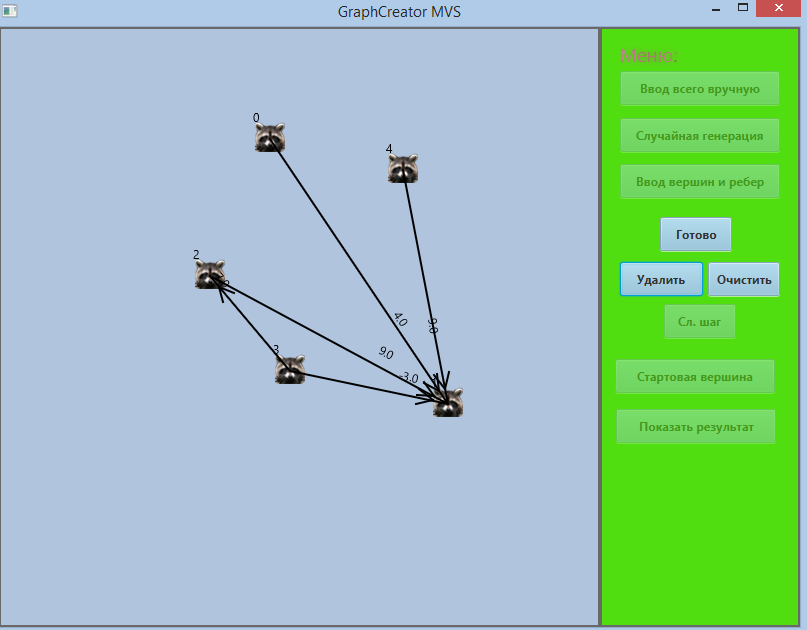
· Создаем ребро в методе ввода всего графа в ручную

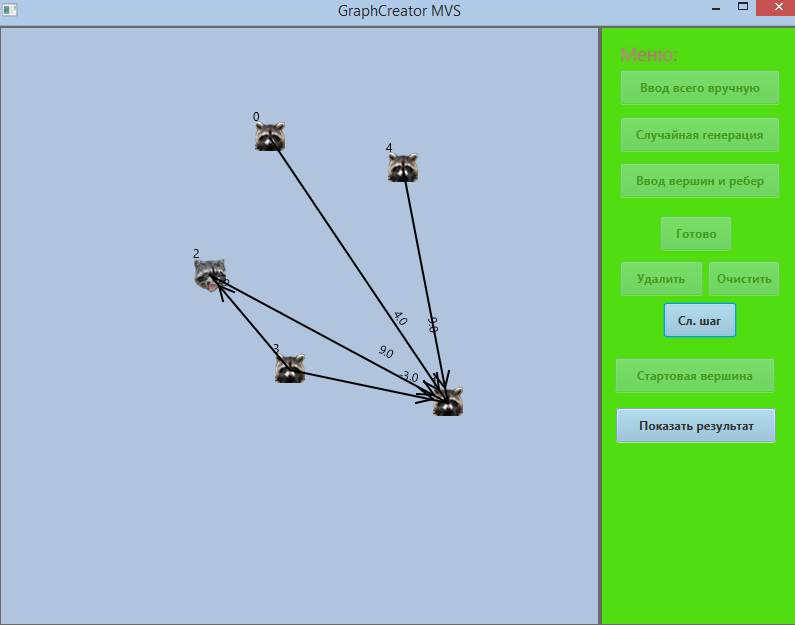


· Ребро создалось.

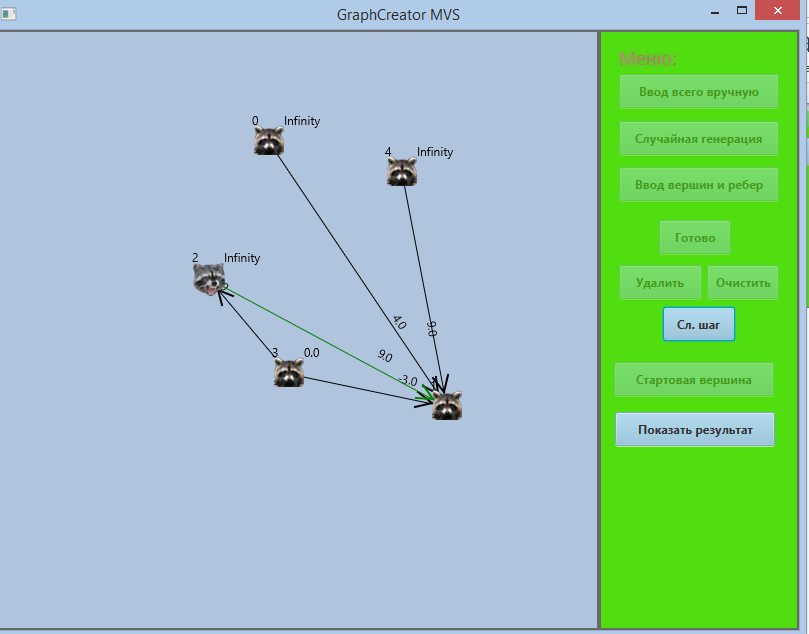


· Аналогичным путем создали еще несколько ребер.

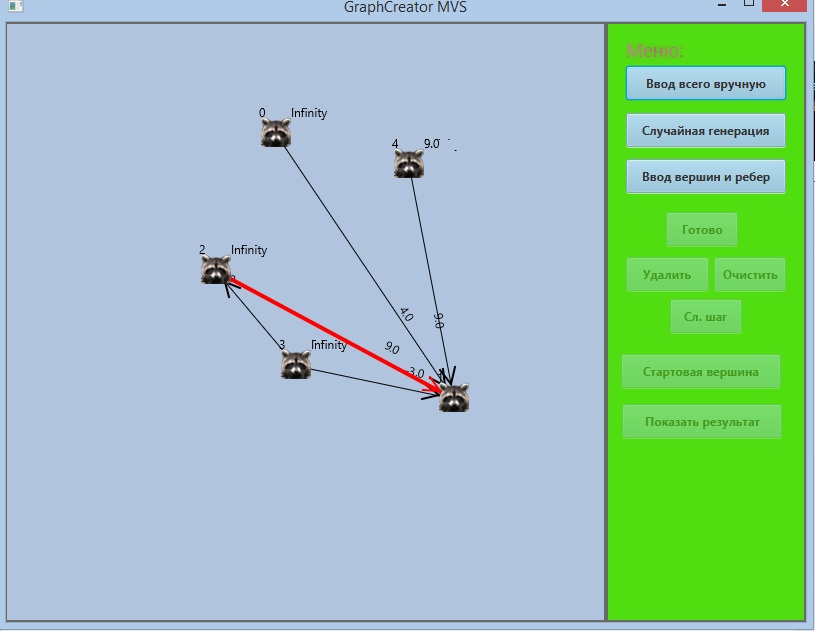


· Выбрали стартовую вершину #2

· Произошло ослабление пути.

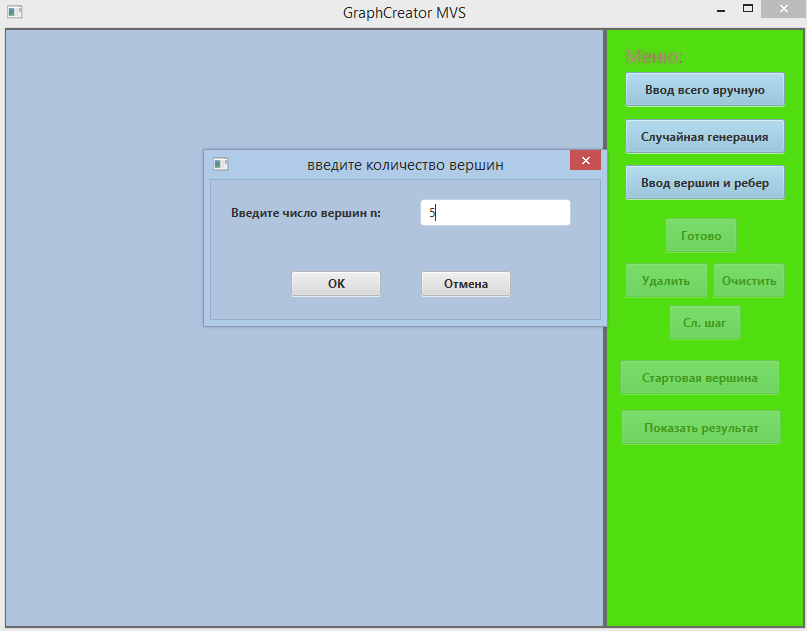
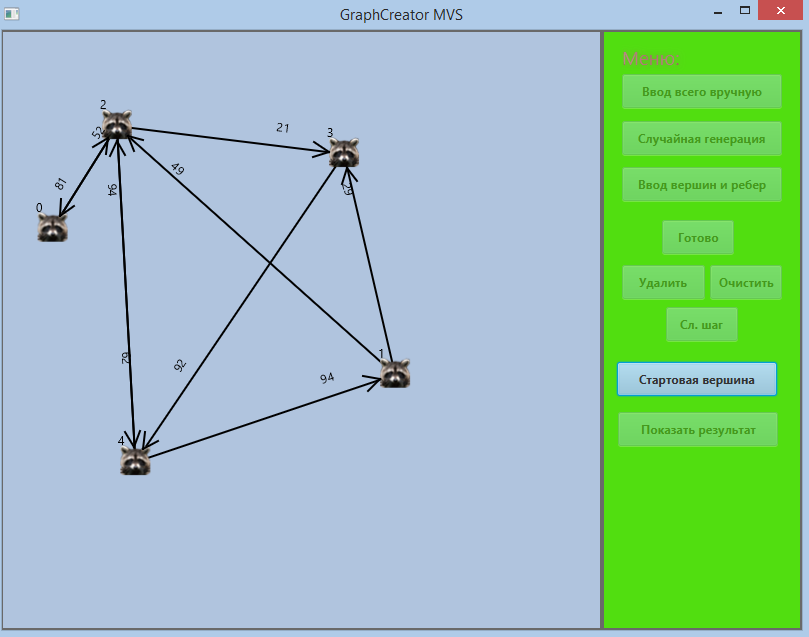


· Кратчайшие пути от вершины №2 ко всем остальным (выделены красным).

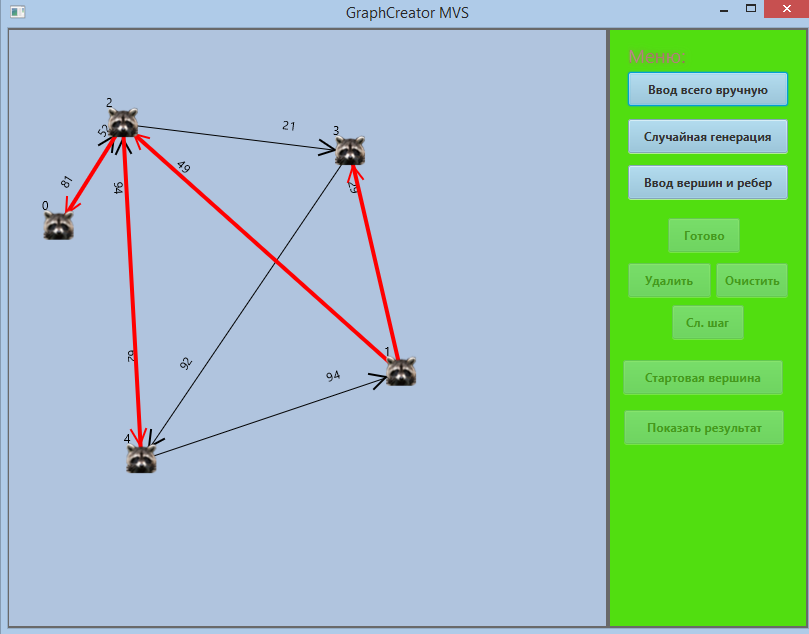


**· СЛУЧАЙНАЯ ГЕНЕРАЦИЯ**

· Случайно сгенерированный граф.

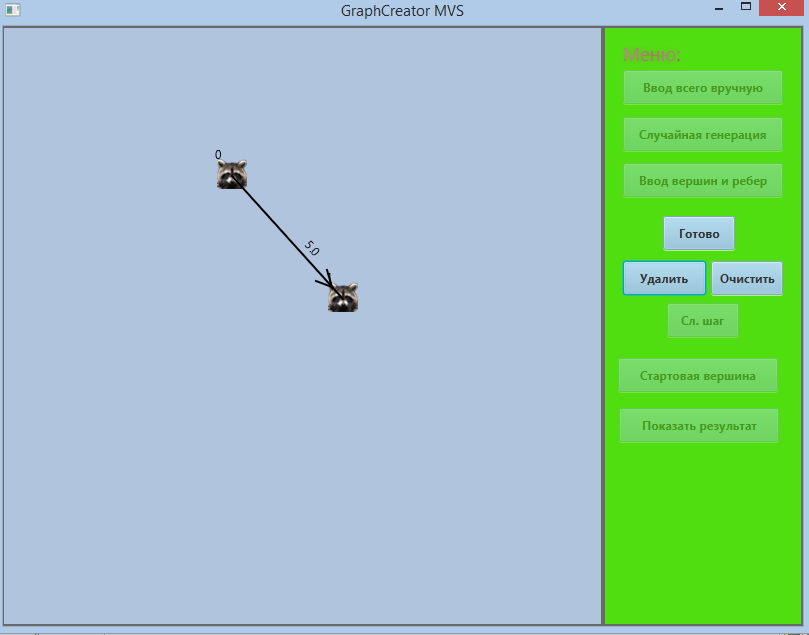
 

· Результат работы алгоритма после нажатия кнопки “Показать результат”.

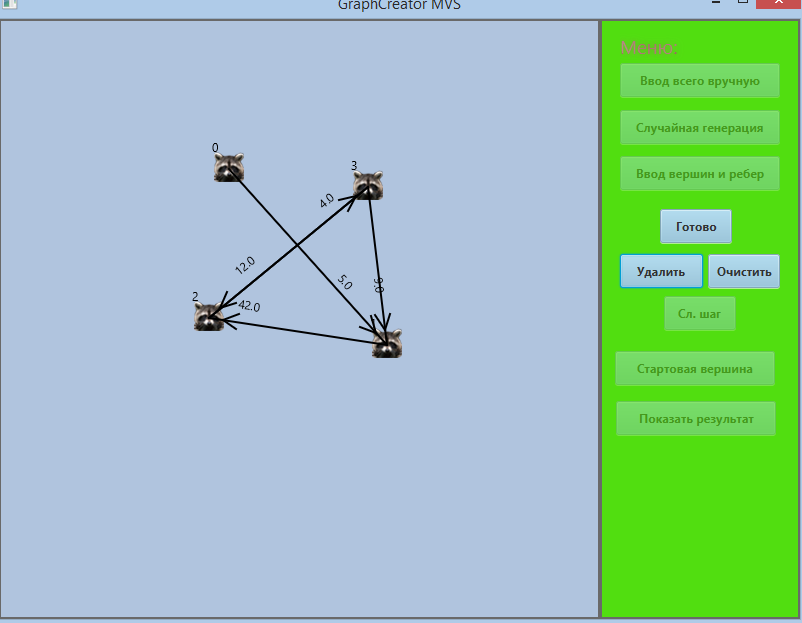


· РЕЖИМ ДОБАВЛЕНИЯ ВРУЧНУЮ

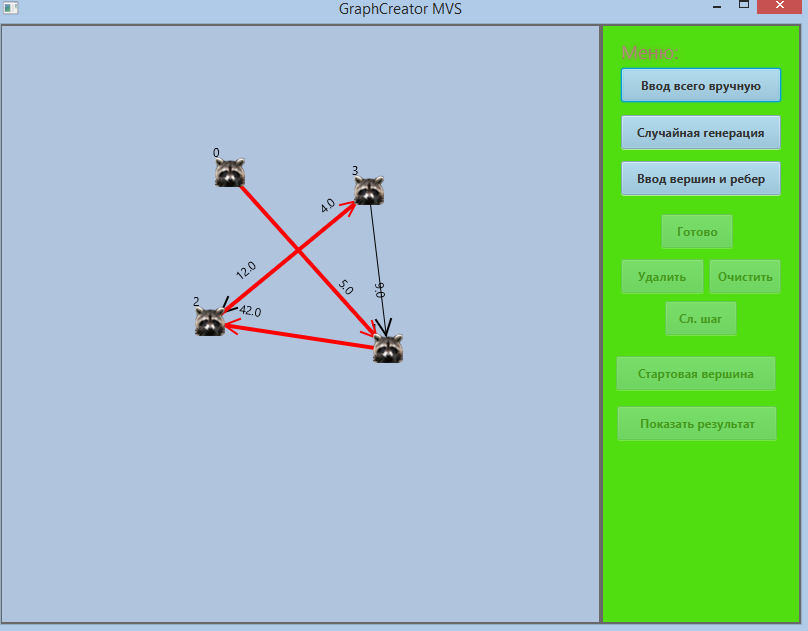
· Создали ребро.



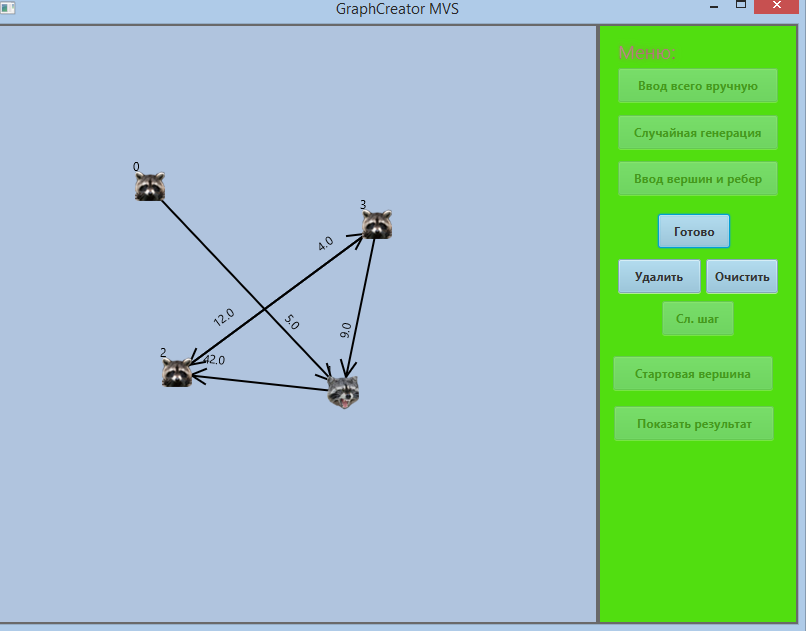
· Создали еще несколько вершин и провели ребра между ними.



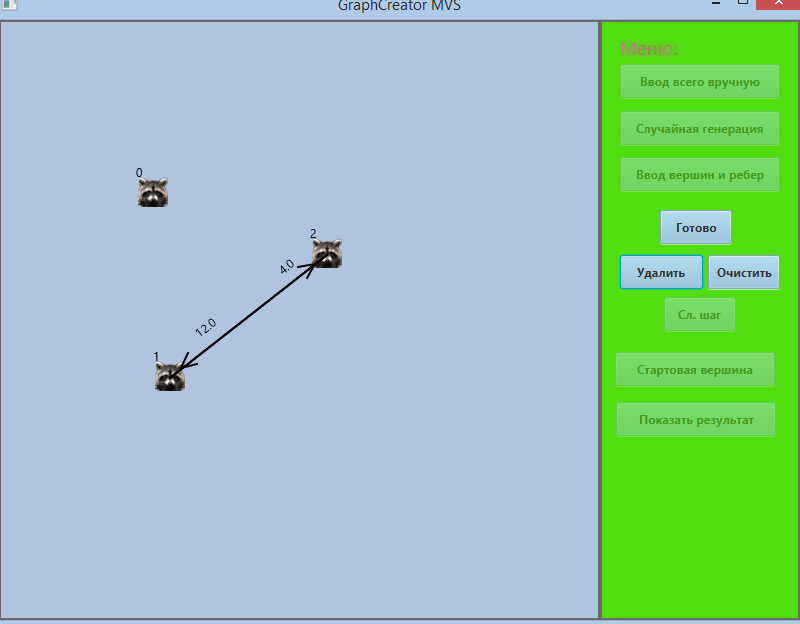
· Результат работы для данного графа.



· Выбрали вершину.



· Удалили ее.



· Нажмем очистить. Панель очистилась.



# заключение

Поставленные задачи успешно выполнены. Проект написан и работает корректно. По мере разработки мы столкнулись с рядом проблем:

* + Проблемы работы с репозиторием
  + Столкнулись с недостатком опыта в написании программ с графическим интерфейсом
  + Как и в каждом проекте столкнулись с множеством багов, которые были успешно устранены

Данный проект визуально демонстрирует работу алгоритма Беллмана-Форда (алгоритм нахождения кратчайших путей в ориентированном графе).

# Приложение А

*Исходный код программы:*

* *edgeSizeDialogController*

**package** controllers;  
  
**import** javafx.event.ActionEvent;  
**import** javafx.fxml.FXML;  
**import** javafx.scene.Node;  
**import** javafx.scene.control.TextField;

**import** javafx.stage.Stage;

**import** javafx.event.EventHandler;

**import** javafx.scene.input.KeyCode;  
**import** javafx.scene.input.KeyEvent;

*/\*\*  
 \* Created by User on 26.06.2017.  
 \*/***public class** edgeSizeDialogController {  
  
 @FXML  
 TextField **edgeSize**;  
 **public static double** *edgeWeight* = -1;  
  
  
 **private boolean** isDigit(String text)  
 {  
 **try**{  
 **int** number = Integer.*parseInt*(text);  
 **return true**;  
 }  
 **catch** (NumberFormatException e) {  
 **return false**;  
 }  
 }  
  
 @FXML  
 **final** EventHandler<KeyEvent> **actionEvent** =  
 **new** EventHandler<KeyEvent>() {  
 **public void** handle(**final** KeyEvent keyEvent) {  
 **if** (keyEvent.getCode() == KeyCode.***ENTER***) {  
 System.***out***.println(**"Ok was pressed"**);  
 System.***out***.println(**"Text "** + **edgeSize**.getText() + **" was entered."**);  
 String text = **edgeSize**.getText();  
 **int** weight;  
 **if** (isDigit(text)) {  
 weight = Integer.*parseInt*(text);  
 **if** (weight !=0) {  
 *edgeWeight* = (**int**) weight;  
 System.***out***.println(**"Ok was pressed"**);  
 createEdgeFixedSize();  
 System.***out***.println(**"Ok was pressed"**);  
 mainController.*smth*.*shouldGenerate* = **false**;  
 mainController.*smth*.*generationType* = **true**;  
 **if** (mainController.*smth*.**DelBut**.isDisable())  
 mainController.*smth*.**DelBut**.setDisable(**false**);  
 **if** (mainController.*smth*.**ClearBut**.isDisable())  
 mainController.*smth*.**ClearBut**.setDisable(**false**);  
 **if** (!mainController.*smth*.**AddArmBut**.isDisable())  
 mainController.*smth*.**AddArmBut**.setDisable(**true**);  
 **if** (!mainController.*smth*.**RandGenBut**.isDisable())  
 mainController.*smth*.**RandGenBut**.setDisable(**true**);  
 **if** (mainController.*smth*.**GotovoBut**.isDisable())  
 mainController.*smth*.**GotovoBut**.setDisable(**false**);  
 **if** (!mainController.*smth*.**FatherBut**.isDisable())  
 mainController.*smth*.**FatherBut**.setDisable(**true**);  
 **if** (!mainController.*smth*.**StepBut**.isDisable())  
 mainController.*smth*.**StepBut**.setDisable(**true**);  
 mainController.*smth*.**activePane**.getChildren().remove(mainController.*smth*.*edges*.get(mainController.*smth*.*edges*.size() - 1).**txtWeight**);  
 mainController.*smth*.*shouldBuild* = **true**;  
 mainController.*smth*.**activePane**.getChildren().add(mainController.*smth*.*edges*.get(mainController.*smth*.*edges*.size() - 1).**txtWeight**);  
 ((Stage) ((Node) keyEvent.getSource()).getScene().getWindow()).close();  
 *//return n;* }  
 **else** {  
 **edgeSize**.clear();  
 **edgeSize**.setPromptText(**" incorrect value "**);  
 }  
 }  
 **else** {  
  
 **edgeSize**.clear();  
 **edgeSize**.setPromptText(**" incorrect value "**);  
 }  
 }  
  
 keyEvent.consume();  
 }  
 };  
 **public void** edgeSizeOK(ActionEvent actionEvent) {  
 System.***out***.println(**"Ok was pressed"**);  
 System.***out***.println(**"Text "** + **edgeSize**.getText() + **" was entered."**);  
 String text = **edgeSize**.getText();  
 **int** weight;  
 **if** (isDigit(text)) {  
 weight = Integer.*parseInt*(text);  
 **if** (weight !=0) {  
 *edgeWeight* = (**int**) weight;  
 System.***out***.println(**"Ok was pressed"**);  
 createEdgeFixedSize();  
 System.***out***.println(**"Ok was pressed"**);  
 mainController.*smth*.*shouldGenerate* = **false**;  
 mainController.*smth*.*generationType* = **true**;  
 **if** (mainController.*smth*.**DelBut**.isDisable())  
 mainController.*smth*.**DelBut**.setDisable(**false**);  
 **if** (mainController.*smth*.**ClearBut**.isDisable())  
 mainController.*smth*.**ClearBut**.setDisable(**false**);  
 **if** (!mainController.*smth*.**AddArmBut**.isDisable())  
 mainController.*smth*.**AddArmBut**.setDisable(**true**);  
 **if** (!mainController.*smth*.**RandGenBut**.isDisable())  
 mainController.*smth*.**RandGenBut**.setDisable(**true**);  
 **if** (mainController.*smth*.**GotovoBut**.isDisable())  
 mainController.*smth*.**GotovoBut**.setDisable(**false**);  
 **if** (!mainController.*smth*.**FatherBut**.isDisable())  
 mainController.*smth*.**FatherBut**.setDisable(**true**);  
 **if** (!mainController.*smth*.**StepBut**.isDisable())  
 mainController.*smth*.**StepBut**.setDisable(**true**);  
 mainController.*smth*.**activePane**.getChildren().remove(mainController.*smth*.*edges*.get(mainController.*smth*.*edges*.size() - 1).**txtWeight**);  
 mainController.*smth*.*shouldBuild* = **true**;  
 mainController.*smth*.**activePane**.getChildren().add(mainController.*smth*.*edges*.get(mainController.*smth*.*edges*.size() - 1).**txtWeight**);  
 ((Stage) ((Node) actionEvent.getSource()).getScene().getWindow()).close();  
 *//return n;* }  
 **else** {  
 **edgeSize**.clear();  
 **edgeSize**.setPromptText(**" incorrect value "**);  
 }  
 }  
 **else** {  
  
 **edgeSize**.clear();  
 **edgeSize**.setPromptText(**" incorrect value "**);  
 }  
 }  
  
 @FXML  
  
  
  
  
 **public void** edgeSizeCancel(ActionEvent actionEvent) {  
 System.***out***.println(**"Cancle was pressed"**);  
 mainController.*smth*.*edges*.get(mainController.*smth*.*edges*.size() - 1).**isChecked** = **true**;  
 mainController.*smth*.*deleteEdge*();  
 *//mainController.deleteEdge();* ((Stage) ((Node) actionEvent.getSource()).getScene().getWindow()).close();  
 }  
  
 **private void** createEdgeFixedSize() {  
 System.***out***.println(**"Create edge"**);  
 **double** weight = *edgeWeight*;  
 mainController.*smth*.**activePane**.getChildren().remove(mainController.*smth*.*edges*.get(mainController.*smth*.*edges*.size() - 1).**txtWeight**);  
 *//mainController.activePane.getChildren().remove(mainController.edges.get(mainController.edges.size() - 1).txtWeight);* mainController.*smth*.*edges*.get(mainController.*smth*.*edges*.size() - 1).setWeight(weight);  
 mainController.*smth*.**activePane**.getChildren().add(mainController.*smth*.*edges*.get(mainController.*smth*.*edges*.size() - 1).**txtWeight**);  
 *// Media sound = new Media(new File("src/content/Edge.wav").toURI().toString());  
 //MediaPlayer mediaPlayer = new MediaPlayer(sound);  
 // mediaPlayer.play();* }  
  
}

* *generationDialogController*

**package** controllers;  
  
**import** javafx.event.ActionEvent;  
**import** javafx.fxml.FXML;  
**import** javafx.scene.Node;  
**import** javafx.scene.control.TextField;  
**import** javafx.scene.image.ImageView;  
**import** javafx.stage.Stage;  
  
**import** objects.Edge;  
**import** objects.Vertex;  
  
*/\*\*  
 \* Created by User on 29.06.2017.  
 \*/***public class** generationDialogController {  
  
 **public static** generationDialogController *smth*;  
 **public** generationDialogController () {  
  
 }  
  
 **public static int** *vertexNumber* = 0;  
  
 @FXML  
 TextField **generationText**;  
  
 **private boolean** isDigit(String text)  
 {  
 **try**{  
 **int** number = Integer.*parseInt*(text);  
 **return true**;  
 }  
 **catch** (NumberFormatException e) {  
 **return false**;  
 }  
 }  
  
 @FXML  
 **public void** gdOK(ActionEvent actionEvent) {  
 System.***out***.println(**"Ok was pressed"**);  
 System.***out***.println(**"Text "** + **generationText**.getText() + **" was entered."**);  
 String text = **generationText**.getText();  
 **int** n;  
 **if** (isDigit(text)) {  
 n = Integer.*parseInt*(text);  
 **if** (n > 0 && n < 700) {  
 *vertexNumber* = n;  
 mainController.*smth*.*shouldGenerate* = **false**;  
 mainController.*smth*.*graphGenerationRandom*();  
 mainController.*smth*.*generationType* = **true**;  
 **if** (mainController.*smth*.**DelBut**.isDisable())  
 mainController.*smth*.**DelBut**.setDisable(**false**);  
 **if** (mainController.*smth*.**ClearBut**.isDisable())  
 mainController.*smth*.**ClearBut**.setDisable(**false**);  
 **if** (!mainController.*smth*.**AddArmBut**.isDisable())  
 mainController.*smth*.**AddArmBut**.setDisable(**true**);  
 **if** (!mainController.*smth*.**RandGenBut**.isDisable())  
 mainController.*smth*.**RandGenBut**.setDisable(**true**);  
 **if** (!mainController.*smth*.**FixedEdgesBut**.isDisable())  
 mainController.*smth*.**FixedEdgesBut**.setDisable(**true**);  
 **if** (mainController.*smth*.**GotovoBut**.isDisable())  
 mainController.*smth*.**GotovoBut**.setDisable(**false**);  
 **if** (!mainController.*smth*.**FatherBut**.isDisable())  
 mainController.*smth*.**FatherBut**.setDisable(**true**);  
 **if** (!mainController.*smth*.**StepBut**.isDisable())  
 mainController.*smth*.**StepBut**.setDisable(**true**);  
 mainController.*smth*.*shouldBuild* = **true**;  
 ((Stage) ((Node) actionEvent.getSource()).getScene().getWindow()).close();  
 *//return n;* }  
 **else** {  
  
 generationDialogController.*smth*.**generationText**.clear();  
 generationDialogController.*smth*.**generationText**.setPromptText(**" 1 < n < 8 "**);  
 *//return 0;* }  
 }  
 **else** {  
 generationDialogController.*smth*.**generationText**.clear();  
 generationDialogController.*smth*.**generationText**.setPromptText(**" 1 < n < 8 "**);  
 *//return 0;* }  
 }  
  
 @FXML  
 **public void** gdCancel(ActionEvent actionEvent) {  
 System.***out***.println(**"Cancle was pressed"**);  
 mainController.*smth*.*shouldGenerate* = **false**;  
 mainController.*smth*.*generationType* = **false**;  
 ((Stage) ((Node) actionEvent.getSource()).getScene().getWindow()).close();  
 }  
  
 **public static** Vertex vertGeneration () {  
 ImageView image = **new** ImageView(**"content/drt1.png"**);  
 Vertex vertex = **new** Vertex(image);  
 **double** x;  
 **double** y;  
 **boolean** OK;  
 **do** {  
 OK = **true**;  
 x = Math.*random*() \* 450;  
 y = Math.*random*() \* 450;  
 **for** (Vertex item : mainController.*smth*.*buttons*) {  
 **double** path = Math.*sqrt*((item.**circle**.getX() - x) \* (item.**circle**.getX() - x)  
 + (item.**circle**.getY() - y) \* (item.**circle**.getY() - y));  
 **if** (path < 65)  
 OK = **false**;  
 }  
 }  
 **while** (OK == **false**);  
 vertex.**circle**.setX(x);  
 vertex.**circle**.setY(y);  
 vertex.setNumber();  
 *//mainController.buttons.add(vertex);* **return** vertex;  
 }  
  
}

* *mainController*

**package** controllers;  
**import** javafx.scene.text.Text;  
**import** javafx.event.Event;  
**import** javafx.event.EventHandler;  
**import** javafx.fxml.FXML;  
**import** javafx.fxml.FXMLLoader;  
**import** javafx.scene.Node;  
**import** javafx.scene.Parent;  
**import** javafx.scene.Scene;  
**import** javafx.scene.control.Button;  
**import** javafx.scene.image.ImageView;  
**import** javafx.scene.input.MouseEvent;  
**import** javafx.scene.layout.\*;  
**import** javafx.scene.paint.Color;  
**import** javafx.scene.shape.Rectangle;  
  
**import static** java.lang.Math.*pow*;  
**import static** java.lang.Math.*sqrt*;  
  
**import** java.io.IOException;  
**import** java.util.ArrayList;  
  
**import** javafx.event.ActionEvent;  
  
**import** javafx.stage.Modality;  
**import** javafx.stage.Stage;  
**import** javafx.stage.WindowEvent;  
**import** objects.Vertex;  
**import** objects.Edge;  
**import** start.Main;  
  
*//import fxml.\*;***public class** mainController {  
  
 **public** mainController () {  
  
}  
 **public final int sizeof2**=25;  
 **public final int MYCONST1** = 1000;  
 **public final int MYCONST2** = 1000;  
 **public final double INFINITE** = (*pow*(**MYCONST1**,**MYCONST2**));  
 **public static** mainController *smth*;  
  
  
  
 **public static** ArrayList<Vertex> *buttons* = **new** ArrayList<Vertex>();  
  
 **public static** ArrayList<Edge> *edges* = **new** ArrayList<Edge>();  
  
 **public static** ArrayList<Double> *vec* = **new** ArrayList<Double>(Vertex.*i*);  
  
 **public static** ArrayList<Integer> *p* = **new** ArrayList<Integer>();  
  
 **public static boolean** *shouldBuild* = **false**;  
 **public static boolean** *shouldGenerate* = **false**;  
 **public static boolean** *fordInWork* = **false**;  
 **public static boolean** *nextStep* = **false**;  
  
 **public static int** *shpEdgesCounter* = 0;  
 **public static int** *relaxCounter* = 0;  
  
 **public static boolean** *last* = **false**;  
 **public static int** *counter* = 0;  
 **public static int** *cycleCounter* = 0;  
 **public static** Vertex *startVert*;  
 **public static boolean** *cycle* = **false**;  
  
 **public static boolean** *fixedSizeType* = **false**;  
  
 **public static boolean** *pressed*;  
  
 **public static boolean** *generationType* = **false**;  
  
  
 @FXML  
 **public void** initialize () {  
 *smth* = **this**;  
 }  
  
 @FXML  
 Pane **activePane**;  
  
 @FXML  
 Button **StepBut**;  
  
 @FXML  
 Button **DelBut**;  
  
 @FXML  
 Button **ClearBut**;  
  
 @FXML  
 Button **RandGenBut**;  
  
 @FXML  
 Button **AddArmBut**;  
  
 @FXML  
 Button **GotovoBut**;  
  
 @FXML  
 Button **FatherBut**;  
  
 @FXML  
 Button **ResultBut**;  
  
 @FXML  
 Button **FixedEdgesBut**;  
  
 **public void** createButton (MouseEvent event) **throws** IOException {  
 **if** (!*pressed*) {  
 **if** (event.getClickCount() == 2)  
 twoMouseClick(event);  
 **else** oneMouseClick(event);  
 }  
 **else** *pressed* = **false**;  
 }  
  
 **private void** twoMouseClick(MouseEvent event)  
 {  
 **if** (createOnCorrectPlace(event)) {  
 **if** (Vertex.*i* == 0)  
 mainController.*smth*.**GotovoBut**.setDisable(**false**);  
 ImageView image = **new** ImageView(**"content/drt1.png"**);  
 Vertex vertex = **new** Vertex(image);  
 vertex.**circle**.setX(event.getX() - **sizeof2**);  
 vertex.**circle**.setY(event.getY() - **sizeof2**);  
 vertex.setNumber();  
 *buttons*.add(vertex);  
 mainController.*smth*.**activePane**.getChildren().addAll(vertex.**circle**, vertex.**txt**);  
 vertex.**circle**.toFront();  
 System.***out***.println(**"created vertex"**);  
 }  
  
 }  
  
 **private boolean** createOnCorrectPlace(MouseEvent event)  
 {  
 **for** (Vertex vertex: *buttons*) {  
 **double** fromCenterToEvent = *sqrt*((event.getX() - vertex.**circle**.getX()-**sizeof2**) \* (event.getX() - vertex.**circle**.getX()-**sizeof2**)  
 + (event.getY() - vertex.**circle**.getY()-**sizeof2**) \* (event.getY() - vertex.**circle**.getY()-**sizeof2**));  
 **if** (fromCenterToEvent <= 40) {  
 **return false**;  
 }  
 }  
 **return true**;  
 }  
 *////////////////////////////////////////////////* **public static** Vertex[] *vertexes* = **new** Vertex[2];  
 **public static int** *countOfCheckedVertex* = 0;  
*///////////////////////////////////////////////////* **public void** oneMouseClick(MouseEvent event) **throws** IOException {  
 **boolean** shouldContinue = **true**;  
 checkButton(event);  
 **for** (Vertex vertex : *buttons*)  
 {  
 **if** (vertex.**isChecked**)  
 {  
 **if** (*countOfCheckedVertex* != 0){  
 **if**(vertex.getNumber() != *vertexes*[0].getNumber()) {  
 *vertexes*[*countOfCheckedVertex*] = vertex;  
 *countOfCheckedVertex*++;  
 }  
 }  
 **else**{  
 *vertexes*[*countOfCheckedVertex*] = vertex;  
 *countOfCheckedVertex*++;  
 }  
 }  
 **if** (*countOfCheckedVertex* == 2)  
 {  
 shouldContinue = checkEdge(event, *vertexes*);  
 *countOfCheckedVertex* = *setNormalView*();  
 **if**(shouldContinue) {  
 createEdge(*vertexes*, event);  
 *vertexes*[0]=**null**;  
 *vertexes*[1]=**null**;  
 *vertexes* = **new** Vertex[2];  
 **break**;  
 }  
 }  
 }  
 }  
  
 **private void** createEdge(Vertex[] vertexes, MouseEvent event) **throws** IOException {  
 System.***out***.println(**"Create edge"**);  
 **double** weight = 0;  
 **if** (*fixedSizeType*)  
 callEdgeSizeWindow(event);  
 weight = *sqrt*(((vertexes[0].**circle**.getX() - vertexes[1].**circle**.getX()) \* (vertexes[0].**circle**.getX() - vertexes[1].**circle**.getX()))  
 + ((vertexes[0].**circle**.getY() - vertexes[1].**circle**.getY()) \* (vertexes[0].**circle**.getY() - vertexes[1].**circle**.getY())));  
 Edge edge = **new** Edge(vertexes[0], vertexes[1], weight);  
 *edges*.add(edge);  
 *// Media sound = new Media(new File("src/content/Edge.wav").toURI().toString());  
 //MediaPlayer mediaPlayer = new MediaPlayer(sound);  
 // mediaPlayer.play();* **for** (**int** i = 0; i < 4; i++)  
 mainController.*smth*.**activePane**.getChildren().addAll(edge.**allLine**[i]);  
 mainController.*smth*.**activePane**.getChildren().add(*edges*.get(*edges*.size() - 1).**txtWeight**);  
 **if** (*fixedSizeType*)  
 mainController.*smth*.**activePane**.getChildren().remove(*edges*.get(*edges*.size() - 1).**txtWeight**);  
 edge.**line**.toBack();  
 System.***out***.println(edge.getWeight());  
  
 }  
  
 **private static int** setNormalView()  
 {  
 **for** (Vertex vertex : *buttons*)  
 {  
 vertex.**circle**.setStyle(**"-fx-image:url('/content/drt1.png');"**);  
 vertex.**isChecked** = **false**;  
 *countOfCheckedVertex* = 0;  
 }  
 **return** 0;  
 }  
  
 **private void** checkButton (MouseEvent event) {  
 **for** (Vertex vertex : *buttons*)  
 {  
 **if** (onCorrectPlace(event,vertex))  
 {  
 **if** (vertex.**isChecked**)  
 {  
 vertex.**circle**.setStyle(**"-fx-image:url('/content/drt1.png');"**);  
 vertex.**isChecked** = **false**;  
 }  
 **else** {  
 vertex.**circle**.setStyle(**"-fx-image:url('/content/drt2.png');"**);  
 vertex.**isChecked** = **true**;  
 }  
 }  
  
 }  
 }  
  
 **private boolean** onCorrectPlace(MouseEvent event,Vertex vertex)  
 {  
 **boolean** inX = event.getX() < vertex.**circle**.getX() + 40 && event.getX() > vertex.**circle**.getX() + 8;  
 **boolean** inY = event.getY() < vertex.**circle**.getY() + 40 && event.getY() > vertex.**circle**.getY() + 8;  
 **return** inX&&inY;  
 }  
  
 **private boolean** checkEdge(MouseEvent event, Vertex[] vertexes) {  
 **boolean** shouldContinue = **true**;  
 **for** (**int** i = 0; i < *edges*.size(); i++)  
 {  
 **if** (twoVertexChecked(i,vertexes))  
 {  
 **if** ((!*edges*.get(i).**isChecked**) && *shpEdgesCounter* == 0)  
 {  
 mainController.*edges*.get(i).**v1**.showPath(*vec*.get(i));  
 mainController.*smth*.**activePane**.getChildren().addAll(mainController.*edges*.get(i).**v1**.**path**);  
 *edges*.get(i).**line**.setStyle(**"-fx-stroke-width:4;-fx-stroke:red"**);  
 *edges*.get(i).drawAllLines(**"red"**,3);  
 vertexesToFront();  
 *//edges.get(i).line.setStyle("-fx-stroke:red");* System.***out***.println(**"Weight of this edge = "** + *edges*.get(i).getWeight());  
 *shpEdgesCounter*++;  
 System.***out***.println(*shpEdgesCounter*);  
 *edges*.get(i).**isChecked** = **true**;  
 }  
 **else** {  
 *edges*.get(i).**line**.setStyle(**"-fx-stroke-width:2;-fx-stroke:black"**);  
 *edges*.get(i).drawAllLines(**"black"**, 2);  
 *shpEdgesCounter*--;  
 vertexesToFront();  
 *edges*.get(i).**isChecked** = **false**;  
 }  
 shouldContinue = **false**;  
 }  
 }  
 **return** shouldContinue;  
 }  
  
 **private boolean** twoVertexChecked(**int** i,Vertex[] vertexes)  
 {  
 **return** (*edges*.get(i).getV1().equals(vertexes[0])&&*edges*.get(i).getV2().equals(vertexes[1]));  
 }  
  
 **public void** buildGraph(ActionEvent actionEvent) {  
 *fixedSizeType* = **false**;  
 *generationType* = **false**;  
 **if** (!mainController.*smth*.*shouldGenerate*)  
 mainController.*smth*.*shouldBuild* = **true**;  
 **if** (mainController.*smth*.**DelBut**.isDisable())  
 mainController.*smth*.**DelBut**.setDisable(**false**);  
 **if** (mainController.*smth*.**ClearBut**.isDisable())  
 mainController.*smth*.**ClearBut**.setDisable(**false**);  
 **if** (!mainController.*smth*.**AddArmBut**.isDisable())  
 mainController.*smth*.**AddArmBut**.setDisable(**true**);  
 **if** (!mainController.*smth*.**RandGenBut**.isDisable())  
 mainController.*smth*.**RandGenBut**.setDisable(**true**);  
 **if** (!mainController.*smth*.**FixedEdgesBut**.isDisable())  
 mainController.*smth*.**FixedEdgesBut**.setDisable(**true**);  
 **if** (Vertex.*i* > 0) {  
 **if** (mainController.*smth*.**GotovoBut**.isDisable())  
 mainController.*smth*.**GotovoBut**.setDisable(**false**);  
 }  
 **if** (!mainController.*smth*.**ResultBut**.isDisable())  
 mainController.*smth*.**ResultBut**.setDisable(**true**);  
 **if** (!mainController.*smth*.**FatherBut**.isDisable())  
 mainController.*smth*.**FatherBut**.setDisable(**true**);  
 **if** (!mainController.*smth*.**StepBut**.isDisable())  
 mainController.*smth*.**StepBut**.setDisable(**true**);  
 **for** (**int** i = 0; i < *edges*.size(); i++) {  
 *edges*.get(i).**line**.setStyle(**"-fx-stroke:black;-fx-stroke-width:2"**);  
 *edges*.get(i).drawAllLines(**"black"**, 2);  
 }  
 **if** (*generationType*) {  
 refreshBut(actionEvent);  
 *generationType* = **false**;  
 *shouldBuild* = **true**;  
 **if** (!mainController.*smth*.**AddArmBut**.isDisable())  
 mainController.*smth*.**AddArmBut**.setDisable(**true**);  
 **if** (!mainController.*smth*.**RandGenBut**.isDisable())  
 mainController.*smth*.**RandGenBut**.setDisable(**true**);  
 **if** (!mainController.*smth*.**FixedEdgesBut**.isDisable())  
 mainController.*smth*.**FixedEdgesBut**.setDisable(**true**);  
 }  
 }  
  
 **public void** tryToBuild(MouseEvent event) **throws** IOException {  
 **if** (*shouldBuild* && !*nextStep*) {  
 **if** (Vertex.*i* > 0) {  
 **if** (mainController.*smth*.**GotovoBut**.isDisable())  
 mainController.*smth*.**GotovoBut**.setDisable(**false**);  
 }  
 **else** mainController.*smth*.**GotovoBut**.setDisable(**true**);  
 createButton(event);  
 System.***out***.println(**"shouldBuild = true!"**);  
 }  
 **else  
 if** (*fordInWork*)  
 {  
 **if** (!mainController.*smth*.**DelBut**.isDisable())  
 mainController.*smth*.**DelBut**.setDisable(**true**);  
 **if** (!mainController.*smth*.**ClearBut**.isDisable())  
 mainController.*smth*.**ClearBut**.setDisable(**true**);  
 **if** (!mainController.*smth*.**AddArmBut**.isDisable())  
 mainController.*smth*.**AddArmBut**.setDisable(**true**);  
 **if** (!mainController.*smth*.**RandGenBut**.isDisable())  
 mainController.*smth*.**RandGenBut**.setDisable(**true**);  
 **if** (!mainController.*smth*.**FixedEdgesBut**.isDisable())  
 mainController.*smth*.**FixedEdgesBut**.setDisable(**true**);  
 **if** (Vertex.*i* > 0) {  
 **if** (!mainController.*smth*.**GotovoBut**.isDisable())  
 mainController.*smth*.**GotovoBut**.setDisable(**true**);  
 }  
 **if** (!mainController.*smth*.**ResultBut**.isDisable())  
 mainController.*smth*.**ResultBut**.setDisable(**true**);  
 **if** (!mainController.*smth*.**FatherBut**.isDisable())  
 mainController.*smth*.**FatherBut**.setDisable(**true**);  
 **if** (mainController.*smth*.**StepBut**.isDisable())  
 mainController.*smth*.**StepBut**.setDisable(**false**);  
 **if** (mainController.*smth*.**ResultBut**.isDisable())  
 mainController.*smth*.**ResultBut**.setDisable(**false**);  
 checkStartVert(event);  
  
 }  
 **else** System.***out***.println(**"shouldBuild = false!"**);  
 }  
  
 **public void** endBuilding(ActionEvent actionEvent) {  
 *shouldBuild* = **false**;  
 *fixedSizeType* = **false**;  
 **if** (!mainController.*smth*.**DelBut**.isDisable())  
 mainController.*smth*.**DelBut**.setDisable(**true**);  
 **if** (!mainController.*smth*.**ClearBut**.isDisable())  
 mainController.*smth*.**ClearBut**.setDisable(**true**);  
 **if** (!mainController.*smth*.**AddArmBut**.isDisable())  
 mainController.*smth*.**AddArmBut**.setDisable(**true**);  
 **if** (!mainController.*smth*.**RandGenBut**.isDisable())  
 mainController.*smth*.**RandGenBut**.setDisable(**true**);  
 **if** (!mainController.*smth*.**FixedEdgesBut**.isDisable())  
 mainController.*smth*.**FixedEdgesBut**.setDisable(**true**);  
 **if** (!mainController.*smth*.**GotovoBut**.isDisable())  
 mainController.*smth*.**GotovoBut**.setDisable(**true**);  
 **if** (mainController.*smth*.**FatherBut**.isDisable())  
 mainController.*smth*.**FatherBut**.setDisable(**false**);  
 **if** (!mainController.*smth*.**StepBut**.isDisable())  
 mainController.*smth*.**StepBut**.setDisable(**true**);  
 **if** (!mainController.*smth*.**ResultBut**.isDisable())  
 mainController.*smth*.**ResultBut**.setDisable(**true**);  
 }  
  
  
 **public void** generateGraph(ActionEvent actionEvent) **throws** Exception {  
  
 **if** (!(*shouldBuild* || *fordInWork*))  
 *shouldGenerate* = **true**;  
 **if** (*shouldGenerate*) {  
 generationDialogController.*vertexNumber* = 0;  
 refreshBut(actionEvent);  
 generation(actionEvent);  
 *shouldGenerate* = **false**;  
 **if** (!**DelBut**.isDisable())  
 **DelBut**.setDisable(**true**);  
 **if** (!**ClearBut**.isDisable())  
 **ClearBut**.setDisable(**true**);  
 **if** (**AddArmBut**.isDisable())  
 **AddArmBut**.setDisable(**true**);  
 **if** (**RandGenBut**.isDisable())  
 **RandGenBut**.setDisable(**true**);  
 **if** (**FixedEdgesBut**.isDisable())  
 **FixedEdgesBut**.setDisable(**true**);  
 **if** (!**GotovoBut**.isDisable())  
 **GotovoBut**.setDisable(**true**);  
 **if** (!**FatherBut**.isDisable())  
 **FatherBut**.setDisable(**true**);  
 **if** (!**StepBut**.isDisable())  
 **StepBut**.setDisable(**true**);  
 **if** (!**ResultBut**.isDisable())  
 **ResultBut**.setDisable(**true**);  
 System.***out***.println(**"shouldGenerate = true!"**);  
 }  
 **else** System.***out***.println(**"shouldGenerate = false!"**);  
 }  
 **public void** generation(ActionEvent actionEvent) **throws** Exception{  
 Stage stage = **new** Stage();  
 Parent panel = FXMLLoader.*load*(getClass().getClassLoader().getResource(**"fxml/generationDialog.fxml"**));  
 stage.setTitle(**"введите количество вершин"**);  
 stage.setMinWidth(200);  
 stage.setMinHeight(150);  
 stage.setResizable(**false**);  
 stage.setScene(**new** Scene(panel));  
 stage.initModality(Modality.***WINDOW\_MODAL***);  
 stage.initOwner(((Node) actionEvent.getSource()).getScene().getWindow());  
 stage.show();  
 *//graphGenerationRandom();* }  
  
  
 **public void** fordInWork(ActionEvent actionEvent) {  
 **if** (!*shouldGenerate* && !*shouldBuild* && !*nextStep*) {  
 *setNormalView*();  
 *fordInWork* = **true**;  
 *//generationType = false;* **if** (!mainController.*smth*.**ResultBut**.isDisable())  
 mainController.*smth*.**ResultBut**.setDisable(**true**);  
 mainController.*smth*.**FatherBut**.setDisable(**true**);  
 }  
 }  
 **public void** checkStartVert(MouseEvent event) {  
 **if** (*fordInWork* && !*nextStep* && *buttons*.size() > 0) {  
  
 mainController.*smth*.**StepBut**.setDisable(**true**);  
 mainController.*smth*.**ResultBut**.setDisable(**true**);  
 checkButton(event);  
 **for** (Vertex vertex : *buttons*) {  
 **if** (vertex.**isChecked**) {  
 mainController.*smth*.**StepBut**.setDisable(**false**);  
 mainController.*smth*.**ResultBut**.setDisable(**false**);  
 *//double INFINITE = 10000;  
 startVert* = vertex;  
  
 **for** (**int** i = 0; i < Vertex.*i*; i++) {  
 *vec*.add(i, **INFINITE**);  
 *p*.add(i,-1);  
 }  
 *vec*.set(vertex.getNumber(), 0.0);  
 *//cycleFord(vertex.getNumber(), Vertex.i, vec);  
 //System.out.println("Ford for " + vertex.getNumber() + " was done!");* **break**;  
 }  
 }  
 mainController.*smth*.**DelBut**.setDisable(**true**);  
 mainController.*smth*.**ClearBut**.setDisable(**true**);  
 mainController.*smth*.**AddArmBut**.setDisable(**true**);  
 mainController.*smth*.**RandGenBut**.setDisable(**true**);  
 mainController.*smth*.**FixedEdgesBut**.setDisable(**true**);  
 mainController.*smth*.**GotovoBut**.setDisable(**true**);  
 mainController.*smth*.**FatherBut**.setDisable(**true**);  
 *//ResultBut.setDisable(false);  
 // StepBut.setDisable(true);* }  
 *//fordInWork=false;* }  
  
  
 **public void** nextStep(ActionEvent actionEvent) **throws** IOException {  
  
 **if** (*fordInWork*) {  
 *nextStep* = **true**;  
 *// StepBut.setDisable(true);* System.***out***.println(**"calling cycleFord"**);  
 **for** (**int** i = 0; i < *edges*.size(); i++) {  
 *edges*.get(i).**line**.setStyle(**"-fx-stroke:black"**);  
 *edges*.get(i).drawAllLines(**"black"**,2);  
 }  
 cycleFord(*startVert*.getNumber(), Vertex.*i*, *vec*);  
  
 **if** (*last*) {  
 **for** (**int** i = 0; i < *edges*.size(); i++) {  
 *edges*.get(i).**line**.setStyle(**"-fx-stroke:black"**);  
 *edges*.get(i).drawAllLines(**"black"**,2);  
 }  
 ArrayList<Integer> path = **new** ArrayList<Integer>();  
 **for** (**int** i = 0; i < Vertex.*i*; i++) {  
 **if** (*startVert*.getNumber() != i) {  
 **for** (**int** cur = i; cur != -1; cur = *p*.get(cur)) {  
 path.add(cur);  
 }  
 reverse(path);  
 **for** (**int** j = 0; j < path.size() - 1; j++){  
 *edges*.get(findEdge(path.get(j),path.get(j+1))).**line**.setStyle(**"-fx-stroke:red;-fx-stroke-width:4"**);  
 *edges*.get(findEdge(path.get(j),path.get(j+1))).**line**.toFront();  
 vertexesToFront();  
 *edges*.get(findEdge(path.get(j), path.get(j+1))).drawAllLines(**"red"**,2);  
 }  
 path.clear();  
 }  
 }  
 path.clear();  
 *p*.clear();  
 *nextStep* = **false**;  
 *fordInWork* = **false**;  
 *last* = **false**;  
 *counter* = 0;  
 mainController.*smth*.**AddArmBut**.setDisable(**false**);  
 mainController.*smth*.**RandGenBut**.setDisable(**false**);  
 mainController.*smth*.**FixedEdgesBut**.setDisable(**false**);  
 mainController.*smth*. **ResultBut**.setDisable(**true**);  
 mainController.*smth*.**StepBut**.setDisable(**true**);  
 *startVert*.**circle**.setStyle(**"-fx-image:url('/content/drt1.png');"**);  
 *startVert*.**isChecked** = **false**;  
 *cycleCounter* = 0;  
 *vec*.clear();  
 }  
 }  
 }  
  
 **public void** vertexesToFront () {  
 **for** (Vertex vertex: *buttons*) {  
 vertex.**circle**.toFront();  
 }  
 }  
  
 **public int** findEdge (**int** v1, **int** v2) {  
 **for** (**int** i = 0; i < *edges*.size(); i++) {  
 **if** ((*edges*.get(i).**v1**.getNumber() == v1) && (*edges*.get(i).**v2**.getNumber() == v2)) {  
 **return** i;  
 }  
  
 }  
 **return** 0;  
 }  
 **public void** cycleFord (**int** s, **int** n, ArrayList<Double> vec) **throws** IOException  
 {  
 **if**(*fordInWork* && !*cycle*)  
 {  
 **if** (!*last*) {  
 **if** (*counter* < *edges*.size())  
 algFord(s, n, *counter*++, vec);  
 **else** {  
 *relaxCounter* = 0;  
 *counter* = 0;  
 *cycleCounter*++;  
 System.***out***.println(*cycleCounter* + **" - cycleCounter"**);  
 *cycle* = **true**;  
 }  
 }  
 }  
 **if** (*fordInWork* && *cycle*) {  
 *last* = **true**;  
 **if** (*counter* < *edges*.size()) {  
 algFord(s, n, *counter*++, vec);  
 *cycle* = **true**;  
 *last* = **false**;  
 }  
 **else** {  
 **if** (*relaxCounter* != 0)  
 *last* = **false**;  
 **else** *last* = **true**;  
 *relaxCounter* = 0;  
 *counter* = 0;  
 *cycleCounter*++;  
 *cycle* = **false**;  
 }  
 }  
 }  
  
 **public void** reverse(ArrayList<Integer> list) {  
 **for** (**int** i = 0; i < list.size() / 2; i++) {  
 **int** tmp = list.get(i);  
 list.set(i,list.get(list.size() - i - 1));  
 list.set(list.size() - i - 1, tmp);  
 }  
 }  
  
 **public void** algFord(**int** s,**int** n, **int** i, ArrayList<Double> vec) **throws** IOException {  
 *// double INFINITE = 10000;* **if** (vec.get(mainController.*edges*.get(i).**v1**.getNumber()) < **INFINITE**) {  
  
 **if** (vec.get(mainController.*edges*.get(i).**v1**.getNumber()) + mainController.*edges*.get(i).getWeight() < vec.get(mainController.*edges*.get(i).**v2**.getNumber())) {  
 Rectangle r = **new** Rectangle();  
 r.setFill(Color.***LIGHTSTEELBLUE***);  
 r.setX((mainController.*edges*.get(i).**v1**.**circle**.getX())+40);  
 r.setY((mainController.*edges*.get(i).**v1**.**circle**.getY()));  
 r.setWidth(42);  
 r.setHeight(10);  
 mainController.*smth*.**activePane**.getChildren().add(r);  
 mainController.*edges*.get(i).**v1**.showPath(vec.get(i));  
 mainController.*smth*.**activePane**.getChildren().addAll(mainController.*edges*.get(i).**v1**.**path**);  
 *edges*.get(i).**line**.setStyle(**"-fx-stroke:green"**);  
 *edges*.get(i).drawAllLines(**"green"**, 2);  
 *edges*.get(i).**line**.toFront();  
 vertexesToFront();  
 vec.set((mainController.*edges*.get(i).**v2**.getNumber()), vec.get(mainController.*edges*.get(i).**v1**.getNumber()) + mainController.*edges*.get(i).getWeight());  
 *p*.set(*edges*.get(i).**v2**.getNumber(), *edges*.get(i).**v1**.getNumber());  
 *last* = **false**;  
 *relaxCounter*++;  
 *cycle* = **false**;  
 }  
 **else** {  
 Rectangle r = **new** Rectangle();  
 r.setFill(Color.***LIGHTSTEELBLUE***);  
 r.setX((mainController.*edges*.get(i).**v1**.**circle**.getX())+40);  
 r.setY((mainController.*edges*.get(i).**v1**.**circle**.getY()));  
 r.setWidth(42);  
 r.setHeight(10);  
 mainController.*smth*.**activePane**.getChildren().add(r);  
 mainController.*edges*.get(i).**v1**.showPath(vec.get(i));  
 mainController.*smth*.**activePane**.getChildren().addAll(mainController.*edges*.get(i).**v1**.**path**);  
 *edges*.get(i).**line**.setStyle(**"-fx-stroke:#2831ff"**);  
 *edges*.get(i).**line**.toFront();  
 vertexesToFront();  
 *edges*.get(i).drawAllLines(**"blue"**, 2);  
 *cycle* = **false**;  
 }  
  
 **for** (**int** y = 0; y < n; y++)  
 {  
  
 System.***out***.println(s + **"->"** + y + **":"** + (vec.get(y)!= **INFINITE** ?(vec.get(y)): **" none"**));  
 }  
 }  
 **else** {  
 Rectangle r = **new** Rectangle();  
 r.setFill(Color.***LIGHTSTEELBLUE***);  
 r.setX((mainController.*edges*.get(i).**v1**.**circle**.getX())+40);  
 r.setY((mainController.*edges*.get(i).**v1**.**circle**.getY()));  
 r.setWidth(42);  
 r.setHeight(10);  
 mainController.*smth*.**activePane**.getChildren().add(r);  
 mainController.*edges*.get(i).**v1**.showPath(vec.get(i));  
 mainController.*smth*.**activePane**.getChildren().addAll(mainController.*edges*.get(i).**v1**.**path**);  
 *edges*.get(i).**line**.setStyle(**"-fx-stroke:orange"**);  
 *edges*.get(i).**line**.toFront();  
 vertexesToFront();  
 *edges*.get(i).drawAllLines(**"orange"**, 2);  
 *//last = false;  
 cycle* = **false**;  
 }  
  
 }  
  
 **public void** deleteButton(ActionEvent actionEvent) {  
 **if**(*buttons*.size()==0){  
 System.***out***.println(**"No Vertexes!"**);  
 **return**;  
 }  
 deleteVertex();  
 *deleteEdge*();  
 }  
  
 **private void** deleteVertex()  
 {  
 **int** deletedNumb = 0;  
 **for**(**int** i=0; i<*buttons*.size();i++)  
 {  
 deletedNumb = 0;  
 **if**(*buttons*.get(i).**isChecked**)  
 {  
 *setNormalView*();  
 deletedNumb = i;  
 mainController.*smth*.**activePane**.getChildren().remove(*buttons*.get(i).**circle**);  
 *buttons*.get(i).**isChecked** = **false**;  
 **for** (**int** j = *edges*.size()-1; j >= 0; j--)  
 {  
 **if** (*edges*.get(j).getV1().equals(*buttons*.get(i)) || *edges*.get(j).getV2().equals(*buttons*.get(i)))  
 {  
 mainController.*smth*.**activePane**.getChildren().removeAll(*edges*.get(j).**allLine**);  
 mainController.*smth*.**activePane**.getChildren().remove(*buttons*.get(i).**txt**);  
 mainController.*smth*.**activePane**.getChildren().removeAll(*edges*.get(j).**txtWeight**);  
 *edges*.remove(*edges*.get(j));  
  
 }  
 }  
 mainController.*smth*.**activePane**.getChildren().remove(*buttons*.get(i).**txt**);  
 *setNormalView*();  
 *buttons*.remove(*buttons*.get(i));  
 }  
 }  
 Vertex.*i*--;  
 **if** (Vertex.*i* > 0) {  
 **for** (**int** i = *buttons*.size(); i > deletedNumb; i--) {  
 mainController.*smth*.**activePane**.getChildren().remove(*buttons*.get(i-1).**txt**);  
 *buttons*.get(i-1).setNumberAfterDelete(i-1);  
 *buttons*.get(i-1).setNumber();  
 mainController.*smth*.**activePane**.getChildren().remove(*buttons*.get(i-1).**txt**);  
 mainController.*smth*.**activePane**.getChildren().add(*buttons*.get(i-1).**txt**);  
 }  
 }  
 }  
  
 **public static void** deleteEdge() {  
 **for**(**int** i=*edges*.size()-1; i>=0;i--)  
 {  
 **if**(*edges*.get(i).**isChecked**)  
 {  
 mainController.*smth*.**activePane**.getChildren().removeAll(*edges*.get(i).**allLine**);  
 mainController.*smth*.**activePane**.getChildren().removeAll(*edges*.get(i).**txtWeight**);  
 *edges*.remove(*edges*.get(i));  
 *setNormalView*();  
 }  
 }  
 *shpEdgesCounter*=0;  
 }  
  
 **public void** refreshBut(ActionEvent actionEvent) {  
 **activePane**.getChildren().clear();  
 **if** (!**DelBut**.isDisable())  
 **DelBut**.setDisable(**true**);  
 **if** (!**ClearBut**.isDisable())  
 **ClearBut**.setDisable(**true**);  
 **if** (**AddArmBut**.isDisable())  
 **AddArmBut**.setDisable(**false**);  
 **if** (**RandGenBut**.isDisable())  
 **RandGenBut**.setDisable(**false**);  
 **if** (**FixedEdgesBut**.isDisable())  
 **FixedEdgesBut**.setDisable(**false**);  
 **if** (!**GotovoBut**.isDisable())  
 **GotovoBut**.setDisable(**true**);  
 **if** (!**FatherBut**.isDisable())  
 **FatherBut**.setDisable(**true**);  
 **if** (!**StepBut**.isDisable())  
 **StepBut**.setDisable(**true**);  
 **if** (!**ResultBut**.isDisable())  
 **ResultBut**.setDisable(**true**);  
 *shouldBuild* = **false**;  
 *shouldGenerate* = **false**;  
 *nextStep* = **false**;  
 *//myGridPane.getChildren().remove(imageSadCat);  
 edges*.clear();  
 *buttons*.clear();  
 *shpEdgesCounter*=0;  
 *countOfCheckedVertex* = 0;  
 *generationType* = **false**;  
 *fixedSizeType* = **false**;  
 Vertex.*i* = 0;  
 }  
  
  
 **private** Vertex checkButtonToReplace (MouseEvent event) {  
 **for** (Vertex vertex : *buttons*)  
 {  
 **if** (onCorrectPlace(event,vertex)) {  
  
 *pressed* = **true**;  
 **return** vertex;  
 }  
  
 }  
 **return null**;  
 }  
  
  
 **public void** changeVertPlace(Event event) **throws** IOException {  
 **if** (*shouldBuild*) {  
 MouseEvent mouseEvent = (MouseEvent) event;  
 Vertex currentVertex = checkButtonToReplace(mouseEvent);  
 **if** (currentVertex != **null**) {  
 currentVertex.**circle**.setX(mouseEvent.getX() - **sizeof2**);  
 currentVertex.**circle**.setY(mouseEvent.getY() - **sizeof2**);  
 currentVertex.syncronize();  
  
 **for** (**int** i = 0; i < *edges*.size(); i++) {  
 **if** (*edges*.get(i).getV1().**circle**.getX() == currentVertex.**circle**.getX() && *edges*.get(i).getV1().**circle**.getY() == currentVertex.**circle**.getY()) {  
 *edges*.get(i).syncronize(currentVertex, *edges*.get(i).getV2(),*generationType*);  
 System.***out***.println(**"Edge"** + i + **"was syncronized"**);  
 System.***out***.println(*edges*.get(i).getV1().**circle**.getX() + **" = x1, "** + *edges*.get(i).getV2().**circle**.getX() + **" = x2"** + **" "**+ *edges*.get(i).getWeight());  
 }  
 **else** {  
 **if** (*edges*.get(i).getV2().**circle**.getX() == currentVertex.**circle**.getX() && *edges*.get(i).getV2().**circle**.getY() == currentVertex.**circle**.getY()) {  
 *edges*.get(i).syncronize(*edges*.get(i).getV1(), currentVertex, *generationType*);  
 System.***out***.println(**"Edge"** + i + **"was syncronized"**);  
 System.***out***.println(*edges*.get(i).getV1().**circle**.getX() + **" = x1, "** + *edges*.get(i).getV2().**circle**.getX() + **" = x2"** + **" "** + *edges*.get(i).getWeight());  
 }  
 }  
 System.***out***.println(**"numb of edges = "** + *edges*.size());  
 }  
 }  
 }  
 }  
  
 **public static void** graphGenerationRandom () {  
 **int** vertexNumber = generationDialogController.*vertexNumber*;  
 **double**[][] genMatrix = **new double**[vertexNumber][vertexNumber];  
 **for** (**int** i = 0; i < vertexNumber; i++) {  
 **for** (**int** j = 0; j < vertexNumber; j++) {  
 **int** resolution = (**int**) (Math.*random*() \* 8);  
 **if** (resolution > 5) {  
 genMatrix[i][j] = Math.*random*()\*99 + 1;  
 }  
 **else** genMatrix[i][j] = 0;  
 }  
 Vertex vertex = generationDialogController.*smth*.*vertGeneration*();  
 mainController.*smth*.**activePane**.getChildren().addAll(vertex.**circle**,vertex.**txt**);  
 vertex.**circle**.toFront();  
 *buttons*.add(vertex);  
 }  
 **int** currentEdge = 0;  
 **for** (**int** i = 0; i < vertexNumber; i++) {  
 **for** (**int** j = 0; j < vertexNumber; j++) {  
 **if** (genMatrix[i][j] != 0 && i != j) {  
 Edge edge = **new** Edge(*buttons*.get(i), *buttons*.get(j),genMatrix[i][j]);  
 *//Edge edge = generationDialogController.edgeGeneration(buttons.get(i), buttons.get(j),genMatrix[i][j]);  
 edges*.add(edge);  
 currentEdge++;  
 *//activePane.getChildren().add(edge.line);* **for** (**int** k = 0; k < 3; k++)  
 mainController.*smth*.**activePane**.getChildren().add(edge.**allLine**[k]);  
 mainController.*smth*.**activePane**.getChildren().add(*edges*.get(currentEdge - 1).**txtWeight**);  
 edge.**line**.toBack();  
 }  
 }  
 }  
  
 }  
  
 **public void** fastResult(ActionEvent actionEvent) {  
 algorithFastResult(*startVert*.getNumber(), Vertex.*i*);  
  
 }  
  
 **public void** algorithFastResult (**int** s, **int** n) {  
 *// double INFINITE = 10000;* **int** o=0;  
 **while** (**true**) {  
 o++;  
 **boolean** last = **true**;  
 **for** (**int** i = 0; i < *edges*.size(); i++) {  
 **if** (*vec*.get(mainController.*edges*.get(i).**v1**.getNumber()) < **INFINITE**) {  
  
 **if** (*vec*.get(mainController.*edges*.get(i).**v1**.getNumber()) + mainController.*edges*.get(i).getWeight() < *vec*.get(mainController.*edges*.get(i).**v2**.getNumber())) {  
 *vec*.set((mainController.*edges*.get(i).**v2**.getNumber()), *vec*.get(mainController.*edges*.get(i).**v1**.getNumber()) + mainController.*edges*.get(i).getWeight());  
 *p*.set(*edges*.get(i).**v2**.getNumber(), *edges*.get(i).**v1**.getNumber());  
 */\* Rectangle r = new Rectangle();  
 r.setFill(Color.BLUE);  
 r.setX((mainController.edges.get(i).v1.circle.getX()));  
 r.setY((mainController.edges.get(i).v1.circle.getX()));  
 r.setWidth(20);  
 r.setHeight(10);  
 mainController.smth.activePane.getChildren().add(r);  
 mainController.edges.get(i).v1.showPath(vec.get(i));  
 mainController.smth.activePane.getChildren().addAll(mainController.edges.get(i).v1.path);\*/* last = **false**;  
 }  
 }  
 }  
 **if**(o > n) {  
 Rectangle r = **new** Rectangle();  
 r.setX(50);  
 r.setY(50);  
 r.setWidth(200);  
 r.setHeight(100);  
 Text txte = **new** Text(100, 100,**"Цикл!!!!1"**);  
 txte.setFill(Color.***RED***);  
 mainController.*smth*.**activePane**.getChildren().addAll(r, txte);  
 **break**;  
 }  
 **if** (last)  
 **break**;  
 }  
  
 **for** (**int** y = 0; y < n; y++)  
 {  
 System.***out***.println(s + **"->"** + y + **":"** + (*vec*.get(y)!= **INFINITE** ?(*vec*.get(y)): **" none"**));  
 }  
  
 **for** (**int** i = 0; i < *edges*.size(); i++) {  
 *edges*.get(i).**line**.setStyle(**"-fx-stroke:black"**);  
 *edges*.get(i).drawAllLines(**"black"**,2);  
 }  
 ArrayList<Integer> path = **new** ArrayList<Integer>();  
 **for** (**int** i = 0; i < Vertex.*i*; i++) {  
 **if** (*startVert*.getNumber() != i) {  
 **for** (**int** cur = i; cur != -1; cur = *p*.get(cur)) {  
 path.add(cur);  
 }  
 reverse(path);  
 **for** (**int** j = 0; j < path.size() - 1; j++){  
 */\*(Rectangle r = new Rectangle();  
 r.setFill(Color.LIGHTSTEELBLUE);  
 r.setX((mainController.edges.get(i).v1.circle.getX())+40);  
 r.setY((mainController.edges.get(i).v1.circle.getY()));  
 r.setWidth(38);  
 r.setHeight(13);  
 mainController.smth.activePane.getChildren().add(r);\*/  
 edges*.get(findEdge(path.get(j),path.get(j+1))).**line**.setStyle(**"-fx-stroke:red;-fx-stroke-width:4"**);  
 *edges*.get(findEdge(path.get(j),path.get(j+1))).**line**.toFront();  
 vertexesToFront();  
 *edges*.get(findEdge(path.get(j), path.get(j+1))).drawAllLines(**"red"**,2);  
 }  
 path.clear();  
 }  
 }  
 path.clear();  
 *p*.clear();  
 *nextStep* = **false**;  
 *fordInWork* = **false**;  
 *last* = **false**;  
 *counter* = 0;  
 mainController.*smth*.**AddArmBut**.setDisable(**false**);  
 mainController.*smth*.**RandGenBut**.setDisable(**false**);  
 mainController.*smth*.**FixedEdgesBut**.setDisable(**false**);  
 mainController.*smth*.**ResultBut**.setDisable(**true**);  
 mainController.*smth*.**StepBut**.setDisable(**true**);  
 *startVert*.**circle**.setStyle(**"-fx-image:url('/content/drt1.png');"**);  
 *startVert*.**isChecked** = **false**;  
 *cycleCounter* = 0;  
 *vec*.clear();  
 }  
  
 **public void** fixedEdges(ActionEvent actionEvent) **throws** IOException{  
 *// refreshBut(actionEvent);  
 shouldBuild* = **true**;  
 **if** (mainController.*smth*.**DelBut**.isDisable())  
 mainController.*smth*.**DelBut**.setDisable(**false**);  
 **if** (mainController.*smth*.**ClearBut**.isDisable())  
 mainController.*smth*.**ClearBut**.setDisable(**false**);  
 **if** (!mainController.*smth*.**AddArmBut**.isDisable())  
 mainController.*smth*.**AddArmBut**.setDisable(**true**);  
 **if** (!mainController.*smth*.**RandGenBut**.isDisable())  
 mainController.*smth*.**RandGenBut**.setDisable(**true**);  
 **if** (!mainController.*smth*.**FixedEdgesBut**.isDisable())  
 mainController.*smth*.**FixedEdgesBut**.setDisable(**true**);  
 **if** (Vertex.*i* > 0) {  
 **if** (mainController.*smth*.**GotovoBut**.isDisable())  
 mainController.*smth*.**GotovoBut**.setDisable(**false**);  
 }  
 **if** (!mainController.*smth*.**ResultBut**.isDisable())  
 mainController.*smth*.**ResultBut**.setDisable(**true**);  
 **if** (!mainController.*smth*.**FatherBut**.isDisable())  
 mainController.*smth*.**FatherBut**.setDisable(**true**);  
 **if** (!mainController.*smth*.**StepBut**.isDisable())  
 mainController.*smth*.**StepBut**.setDisable(**true**);  
 **for** (**int** i = 0; i < *edges*.size(); i++) {  
 *edges*.get(i).**line**.setStyle(**"-fx-stroke:black;-fx-stroke-width:2"**);  
 *edges*.get(i).drawAllLines(**"black"**, 2);  
 }  
 **if** (*generationType*) {  
 *//refreshBut(actionEvent);  
 generationType* = **false**;  
 *shouldBuild* = **true**;  
 **if** (!mainController.*smth*.**AddArmBut**.isDisable())  
 mainController.*smth*.**AddArmBut**.setDisable(**true**);  
 **if** (!mainController.*smth*.**RandGenBut**.isDisable())  
 mainController.*smth*.**RandGenBut**.setDisable(**true**);  
 **if** (!mainController.*smth*.**FixedEdgesBut**.isDisable())  
 mainController.*smth*.**FixedEdgesBut**.setDisable(**true**);  
 }  
 *fixedSizeType* = **true**;  
 *generationType* = **true**;  
 }  
  
 **public void** callEdgeSizeWindow (Event actionEvent)**throws** IOException{  
 Stage stage = **new** Stage();  
 Parent panel = FXMLLoader.*load*(getClass().getClassLoader().getResource(**"fxml/edgeSizeDialog.fxml"**));  
 stage.setTitle(**"введите вес ребра"**);  
 stage.setMinWidth(200);  
 stage.setMinHeight(150);  
 stage.setResizable(**false**);  
 stage.setScene(**new** Scene(panel));  
 stage.initModality(Modality.***WINDOW\_MODAL***);  
 stage.initOwner(((Node) actionEvent.getSource()).getScene().getWindow());  
 stage.show();  
 stage.setOnCloseRequest(**new** EventHandler<WindowEvent>() {  
  
 @Override  
 **public void** handle(WindowEvent event) {  
 event.consume();  
  
 }  
  
 });  
 }  
}

* *Edge*

**package** objects;  
  
  
**import** javafx.scene.media.Media;  
**import** javafx.scene.media.MediaPlayer;  
**import** javafx.scene.shape.Line;  
**import** javafx.scene.text.Text;  
  
**import** java.awt.\*;  
  
**import static** java.lang.Math.*sqrt*;  
  
*/\*\*  
 \* Created by User on 29.06.2017.  
 \*/***public class** Edge {  
 **public final int sizeof1**=15;  
 **public final int sizeof2**=25;  
 **public** Vertex **v1**;  
 **public** Vertex **v2**;  
 **private double weight**;  
 **public** Line **line**= **new** Line();  
 **public** Line **allLine**[];  
 **public boolean isChecked**;  
 **public** Text **txt**;  
 **public** Text **txtWeight**;  
  
 **public** Edge(Vertex v1, Vertex v2, **double** weight) {  
 **this**.**v1** = v1;  
 **this**.**v2** = v2;  
 **this**.**weight** = weight;  
 **this**.**isChecked** = **false**;  
 **this**.**allLine** = createLine (v1,v2);  
 **this**.**line** = **allLine**[0];  
 }  
 **public void** drawAllLines (String color, **int** width) {  
 **allLine**[1].setStyle(**"-fx-stroke:"** + color +**";-fx-stroke-width:"** + width);  
 **allLine**[2].setStyle(**"-fx-stroke:"** + color + **";-fx-stroke-width:"** + width);  
 *//allLine[3].setStyle("-fx-stroke:" + color + ";-fx-stroke-width:" + width);* }  
  
 **public** Line[] createLine(Vertex v1, Vertex v2) {  
 *//Line line = new Line();  
 //Polygon.* **this**.**line**.setStartX(v1.**circle**.getX() + **sizeof2**);  
 **this**.**line**.setStartY(v1.**circle**.getY() + **sizeof2**);  
 **this**.**line**.setEndX(v2.**circle**.getX() + **sizeof2**);  
 **this**.**line**.setEndY(v2.**circle**.getY() + **sizeof2**);  
 **this**.**line**.setStyle(**"-fx-stroke-width:2"**);  
  
 *//Media sound = new Media(new File("").toURI().toString());  
 //MediaPlayer mediaPlayer = new MediaPlayer(sound);  
 //mediaPlayer.play();* **double** arrowHeight=**sizeof1**, arrowWidth=**sizeof1**;  
 **double** dx, dy,dx1,dy1;  
 **int** m;  
 **if** (v2.**circle**.getX() > v1.**circle**.getX()) {  
 dx = v2.**circle**.getX() - v1.**circle**.getX();  
 dy = v2.**circle**.getY() - v1.**circle**.getY();  
 dx1=v1.**circle**.getX()+dx/2+**sizeof1**+dx/4;  
 m = 1;  
 } **else** {  
 dx = -v2.**circle**.getX() + v1.**circle**.getX();  
 dy = -v2.**circle**.getY() + v1.**circle**.getY();  
 dx1=v2.**circle**.getX()+dx/2+10-dx/4;  
 m = -1;  
 }  
  
 **if**(v2.**circle**.getY() > v1.**circle**.getY()){  
 dy1= v1.**circle**.getY()+(v2.**circle**.getY() - v1.**circle**.getY())/2+10+(v2.**circle**.getY() - v1.**circle**.getY())/4;  
 }  
 **else** {  
 dy1 = v2.**circle**.getY()+(-v2.**circle**.getY() + v1.**circle**.getY())/2+10+(v2.**circle**.getY() - v1.**circle**.getY())/4;  
 }  
  
 **double** angle = dy / dx;  
 angle = Math.*atan*(angle);  
  
 **txtWeight** = **new** Text((dx1),(dy1),**""**+(**int**)getWeight());  
 **txtWeight**.setRotate(Math.*toDegrees*(angle));  
  
 Point p5=**new** Point((**int**)(v2.**circle**.getX()-m\*(**sizeof1**)\*Math.*cos*(angle))+**sizeof2**,(**int**)(v2.**circle**.getY()-m\*(**sizeof1**)\*Math.*sin*(angle))+**sizeof2**);  
 Point p2=**new** Point((**int**)(v2.**circle**.getX()-m\*(arrowHeight+**sizeof1**)\*Math.*cos*(angle)),(**int**)(v2.**circle**.getY()-m\*(arrowHeight+**sizeof1**)\*Math.*sin*(angle)));  
 angle = Math.*atan*(((-1) \* dx) / dy);  
 Point p3=**new** Point((**int**)(p2.getX()-(arrowWidth / 2) \* Math.*cos*(angle))+**sizeof2**, (**int**)(p2.getY()- (arrowWidth / 2) \* Math.*sin*(angle))+**sizeof2**);  
 Point p4=**new** Point((**int**)(p2.getX()+(arrowWidth / 2) \* Math.*cos*(angle))+**sizeof2**, (**int**)(p2.getY()+ (arrowWidth / 2) \* Math.*sin*(angle))+**sizeof2**);  
  
 **txt** = **new** Text(v1.**circle**.getX(),v1.**circle**.getY(),**""**+**weight**);  
  
 Line line1 = **new** Line();  
 line1.setStartX(p5.getX());  
 line1.setStartY(p5.getY());  
 line1.setEndX(p3.getX());  
 line1.setEndY(p3.getY());  
 line1.setStyle(**"-fx-stroke-width:2"**);  
 */\*Line line2 = new Line();  
 line2.setStartX(p3.getX());  
 line2.setStartY(p3.getY());  
 line2.setEndX(p4.getX());  
 line2.setEndY(p4.getY());  
 line2.setStyle("-fx-stroke-width:2");\*/* Line line2 = **new** Line();  
 line2.setStartX(p4.getX());  
 line2.setStartY(p4.getY());  
 line2.setEndX(p5.getX());  
 line2.setEndY(p5.getY());  
 line2.setStyle(**"-fx-stroke-width:2"**);  
 Line[] lines=**new** Line[3];  
 lines[0]=**line**;  
 lines[1]=line1;  
 lines[2]=line2;  
 *//lines[3]=line3;* **return** lines;  
  
 }  
  
 **public void** syncronize (Vertex v1,Vertex v2, **boolean** weightChange) {  
 **this**.**v1** = v1;  
 **this**.**v2** = v2;  
 changeLine(v1, v2, weightChange);  
 }  
  
 **public void** changeLine (Vertex v1, Vertex v2, **boolean** weightChange) {  
 **this**.**line**.setStartX(v1.**circle**.getX() + **sizeof2**);  
 **this**.**line**.setStartY(v1.**circle**.getY() + **sizeof2**);  
 **this**.**line**.setEndX(v2.**circle**.getX() + **sizeof2**);  
 **this**.**line**.setEndY(v2.**circle**.getY() + **sizeof2**);  
 **if** (!weightChange) {  
 **this**.**weight** = (**int**)*sqrt*(((v1.**circle**.getX() - v2.**circle**.getX()) \* (v1.**circle**.getX() - v2.**circle**.getX()))  
 + ((v1.**circle**.getY() - v2.**circle**.getY()) \* (v1.**circle**.getY() - v2.**circle**.getY())));  
 }  
 **else** {  
 **this**.**weight** = (**int**)**this**.**weight**;  
 }  
 **double** arrowHeight=**sizeof1**, arrowWidth=**sizeof1**;  
 **double** dx, dy,dx1,dy1;  
 **int** m;  
 **if** (v2.**circle**.getX() > v1.**circle**.getX()) {  
 dx = v2.**circle**.getX() - v1.**circle**.getX();  
 dy = v2.**circle**.getY() - v1.**circle**.getY();  
 dx1=v1.**circle**.getX()+dx/2+**sizeof1**+dx/4;  
 m = 1;  
 } **else** {  
 dx = -v2.**circle**.getX() + v1.**circle**.getX();  
 dy = -v2.**circle**.getY() + v1.**circle**.getY();  
 dx1=v2.**circle**.getX()+dx/2+10-dx/4;  
 m = -1;  
 }  
  
 **if**(v2.**circle**.getY() > v1.**circle**.getY()){  
 dy1= v1.**circle**.getY()+(v2.**circle**.getY() - v1.**circle**.getY())/2+10+(v2.**circle**.getY() - v1.**circle**.getY())/4;  
 }  
 **else** {  
 dy1 = v2.**circle**.getY()+(-v2.**circle**.getY() + v1.**circle**.getY())/2+10+(v2.**circle**.getY() - v1.**circle**.getY())/4;  
 }  
  
 **double** angle = dy / dx;  
 angle = Math.*atan*(angle);  
  
 **txtWeight**.setX(dx1);  
 **txtWeight**.setY(dy1);  
 **txtWeight**.setRotate(Math.*toDegrees*(angle));  
 **txtWeight**.setText(**""** + **weight**);  
  
 Point p5=**new** Point((**int**)(v2.**circle**.getX()-m\*(**sizeof1**)\*Math.*cos*(angle))+**sizeof2**,(**int**)(v2.**circle**.getY()-m\*(**sizeof1**)\*Math.*sin*(angle))+**sizeof2**);  
 Point p2=**new** Point((**int**)(v2.**circle**.getX()-m\*(arrowHeight+**sizeof1**)\*Math.*cos*(angle)),(**int**)(v2.**circle**.getY()-m\*(arrowHeight+**sizeof1**)\*Math.*sin*(angle)));  
 angle = Math.*atan*(((-1) \* dx) / dy);  
 Point p3=**new** Point((**int**)(p2.getX()-(arrowWidth / 2) \* Math.*cos*(angle))+**sizeof2**, (**int**)(p2.getY()- (arrowWidth / 2) \* Math.*sin*(angle))+**sizeof2**);  
 Point p4=**new** Point((**int**)(p2.getX()+(arrowWidth / 2) \* Math.*cos*(angle))+**sizeof2**, (**int**)(p2.getY()+ (arrowWidth / 2) \* Math.*sin*(angle))+**sizeof2**);  
  
 **txt**.setX(p3.getX());  
 **txt**.setY(p3.getY());  
  
 *//Line line1 = new Line();* **allLine**[1].setStartX(p5.getX());  
 **allLine**[1].setStartY(p5.getY());  
 **allLine**[1].setEndX(p3.getX());  
 **allLine**[1].setEndY(p3.getY());  
 *//Line line2 = new Line();  
 /\*allLine[2].setStartX(p3.getX());  
 allLine[2].setStartY(p3.getY());  
 allLine[2].setEndX(p4.getX());  
 allLine[2].setEndY(p4.getY());\*/  
 //Line line3 = new Line();* **allLine**[2].setStartX(p4.getX());  
 **allLine**[2].setStartY(p4.getY());  
 **allLine**[2].setEndX(p5.getX());  
 **allLine**[2].setEndY(p5.getY());  
 *// Line[] lines=new Line[4];  
  
 //allLine[0]=line;  
 //lines[1]=line1;  
 //lines[2]=line2;  
 //lines[3]=line3;  
 //return lines;* }  
  
 **public** Vertex getV1() {  
 **return this**.**v1**;  
 }  
  
 **public** Vertex getV2() {  
 **return this**.**v2**;  
 }  
  
 **public double** getWeight() {  
 **return this**.**weight**;  
 }  
  
 **public void** setWeight (**double** weight) {  
 **this**.**weight** = weight;  
 **this**.**txtWeight**.setText(**""**+weight);  
 }  
}

.

* *Vertex*

**package** objects;  
  
**import** javafx.scene.image.ImageView;  
**import** javafx.scene.text.Text;  
  
**import** java.util.ArrayList;  
*/\*\*  
 \* Created by User on 26.06.2017.  
 \*/***public class** Vertex {  
 **public static int** *i* = 0;  
 **public static int** *j* = 0;  
 **public** ImageView **circle**;  
 **public boolean isChecked**;  
 **private int number**;  
 **public** Text **txt**;  
 **private int nonenumber**;  
 **public** Text **path**;  
 **public** Vertex(ImageView circle)  
 {  
 **this**.**nonenumber**=*j*;  
 **this**.**number** = *i*;  
 **this**.**circle**=circle;  
 **isChecked**=**false**;  
 *i*++;  
 *//txt = new Text(this.circle.getX() - 20,this.circle.getY() - 20,""+getNumber());* }  
 **public int** getPath () {  
 **return this**.**nonenumber**;  
 }  
 **public void** showPath (**double** u){  
 **this**.**path** = **new** Text(**this**.**circle**.getX() + 40, **this**.**circle**.getY() + 8,**""** + u);  
}  
  
 **public void** setNumber (){  
 **this**.**txt** = **new** Text(**this**.**circle**.getX() + 8, **this**.**circle**.getY() + 8,**""** + getNumber());  
 }  
  
 **public void** setNumberAfterDelete (**int** number) {  
 **this**.**number** = number;  
 }  
  
 **public void** syncronize () {  
 **this**.**txt**.setX(**this**.**circle**.getX() + 8);  
 **this**.**txt**.setY(**this**.**circle**.getY() + 8);  
 }  
  
 **public int** getNumber () {  
 **return this**.**number**;  
 }  
  
}

* *Main*

**package** start;  
  
**import** javafx.application.Application;  
**import** javafx.fxml.FXMLLoader;  
**import** javafx.scene.Parent;  
**import** javafx.scene.Scene;  
**import** javafx.stage.Stage;  
  
**public class** Main **extends** Application {  
  
 @Override  
 **public void** start(Stage primaryStage) **throws** Exception {  
 **try** {  
 Parent root = FXMLLoader.*load*(getClass().getClassLoader().getResource(**"fxml/mainInterface.fxml"**));  
 primaryStage.setTitle(**"GraphCreator MVS"**);  
 primaryStage.setScene(**new** Scene(root, 800, 600));  
 primaryStage.setMinHeight(600);  
 primaryStage.setMinWidth(800);  
 primaryStage.show();  
 }  
 **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 *// Pane pane = new Pane();* }  
 **public static void** main(String[] args) {  
 *launch*(args);  
 }  
}