The Graph - Final Report

Supervisor: Oliver Baudon
Bordeaux University

May 6, 2019

Member:
Vo Hung Son
Nguyen Ngoc Nhu Y
Tran Quang Nhat

Summary

The first that we want to say thank you to Professor Oliver Baudon and Professor Fabien that helping us during the project. This is the first report that we only introduce about the bibliograhy that we use to do this project and we will existing analysis about the library that Professor Oliver Baudon gave us. After that is requirements analysis and objective, architecture and tests.
The main purpose of this project is improve the library of the Graph, writing interfaces of the graph.
We have four members in our group for Project Graph.
Vo Hung Son - vohungson@gmail.com
Tran Quang Nhat - tranquangnhat2211@gmail.com
Nguyen Hoang Do - hoangdo86@gmail.com
Nguyen Ngoc Nhu Y - nguyenngnhuy@yahoo.com

The goal of this project:

1. To improve an existing library in Java devoted to manipulate graphs.
    - Make the complete review of the library. In particular you may propose modifications linked with version 8 of Java [**cite-key5**].
    - Add javadoc comments [**cite-key6**] in order to allow the distribution of the library with LGPL license [**cite-key7**].
2. Write a simple graphical interface with JavaFx to create graphs.
3. Propose an exportation of the graph drawings with TikZ (a language which allows to add graphics in Latex files).

Contents

Summary **i**

1 Introduction **1**

2 Existing analysis of The graph **1**
2.1 The main source code . . . 1
2.2 Package Collection . . . 1
2.3 Package Graph . . . 1
2.4 Package Util . . . 2

3 Proposed system **2**
3.1 Overview . . . 2
3.2 Functional requirements . . . 2
3.2.1 Graph interfaces . . . 2
3.2.2 Improve and Optimize the performence of algorithms in library . . . 2
3.3 Nonfunctional requirements . . . 3
3.3.1 Usability . . . 3
3.3.2 Reliability . . . 3
3.3.3 Performance . . . 3
3.3.4 Supportability . . . 3
3.3.5 Implementation . . . 3
3.3.6 Interface . . . 3
3.4 System models . . . 3
3.4.1 Use case model . . . 3
3.4.2 Object model . . . 5
3.4.3 User Interface . . . 7

4 Architecture **7**
4.1 Introduction . . . 7
4.1.1 Purpose . . . 7
4.1.2 Scope . . . 7
4.2 Architectural Representation . . . 7
4.2.1 Use Case view . . . 7
4.2.2 Logical view . . . 8
4.3 Architectural Goals and Constraints . . . 8
4.4 Use-Case Realizations . . . 8
4.4.1 Find the shortest path of graph . . . 8
4.4.2 Draw Vertices . . . 9
4.4.3 Draw Edges . . . 10
4.4.4 Delete Edges . . . 11
4.4.5 Delete Vertices . . . 12
4.4.6 Move Vertices . . . 13

5 Application **14**
5.1 Git Hub . . . 14
5.1.1 Repository . . . 14
5.2 Requirement . . . 14
5.3 Architecture . . . 15
5.4 Framework . . . 16
5.5 UML . . . 16
5.6 IDE . . . 17
5.7 Unit Test . . . 17
5.8 Functions . . . 17
5.8.1 Draw vertex . . . 17
5.8.2 Draw vertex custom text . . . 18

5.8.3 Move vertex . . . 20
5.8.4 Draw edge . . . 20
5.8.5 Intermediate Point . . . 22
5.8.6 Remove Object . . . 22
5.9 TikZ package . . . 22
5.9.1 Introduction . . . 23
5.9.2 Basic geometric shapes: Circles, ellipses and polygons . . . 23
5.9.3 Diagrams . . . 23

1 Introduction

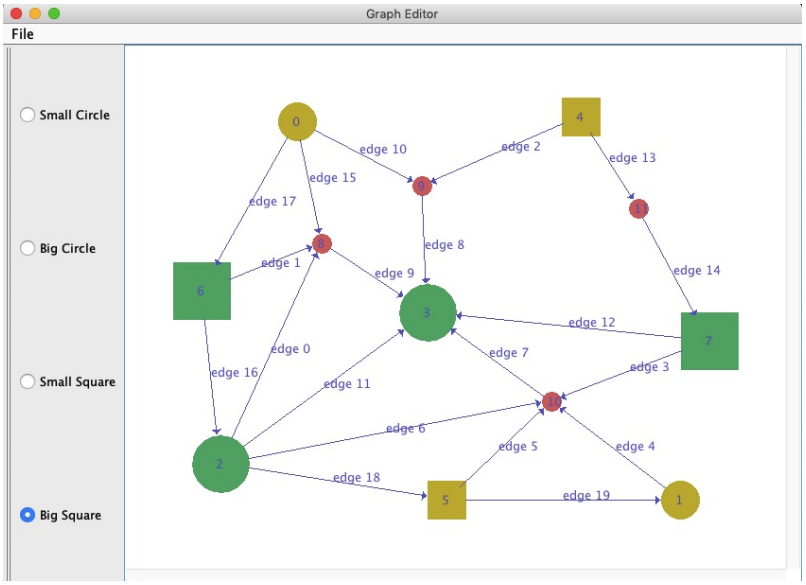

Figure 1: The Graph

Graph Theory is a powerful tool for studying complex combinatorial structures [**cite-key1**]. But our project only focus to make the interfaces of the graph and use the interfaces that we improved in the existing library and apply them in the JavaFx [**cite-key4**] graphic program. We also have to use TikZ language to draw graphic elements in Latex [**cite-key2**] for the report. Tikz is probably the most complex and powerful tool to create graphic elements in LATEX. In this article some of the basics will be explained: lines, dots, curves, circles, rectangles, etc by means of simple examples. Almost the knowledge that we use in this project is from the book DaMN [**cite-key3**].

2 Existing analysis of The graph

The source codes that we have 3 packages and main source code.

2.1 The main source code

In the main source code, we have these classes: Multigraph, GraphWithEmbedding, DirectedEdge, VertexOrEdgeIterator, VertexOrEdgeIterable... In the MultiGraph, we have the SubMultiGraph, Edges, Vertices, Graph, SubGraph, ParticialMultiGraph, InducedSubMultiGraph. These classed that we use to make and manage the set of graphs. The classes VertexOrEdgeIterator, VertexOrEdgeIterable are used for loop vertices and edges.

2.2 Package Collection

The package Collection is an Iterator Design pattern that we can loop, append and print...

2.3 Package Graph

In this package that we have so many classes. The class Edges that we can add, remove, clear and iterator. The class Vertices that we also can add, remove, clear and iterator. The class Graph that we can add Vertex, remove Vertex, add Edge, remove Edge, check the Vertices are neighbour or not, check the Graph contains Vertex V, check the graph contains Edge E, count the degree of the graph and we also can remove all vertices and all edges. The class MultiGraph and SubMultiGraph have the same functions with the class Graph but they have more sets of the graph. There are three interfaces here: PartialGraph,

InducedSubGraph, SubGraph. There are these classes VertexOrEdgeIterator, VertexOrEdgeIterable. This is an Iterator design pattern that use for Vertex and Edge.

2.4 Package Util

This package have many classes like: Graphs, Vizing, Conjecture, ShortestPathsMatrices, NegativeEdgeException, Lamda, Position, DepthFirstSearchDataImpl, RandomGraphVietnam, RandomGraph, FlowResults, NegativeCircuitException, K Coloring. Here applies some algorithms like Coloring, Shortest Paths, Depth First Search, Flow Network... In the flow network that we show the Maximum Flow and minimum cut of the network. Especially in the class Graphs have so many algorithm functions inside like: breathFirstSearch, dijkstra algorithm, bellmanFord algorithm, floydWarshall, kruskal, prim, fordFulkerson...

3 Proposed system

3.1 Overview

Implement a simple application which allow creating the graphs. Implement graphic interfaces to creating the graphs. Improve the code of the existed library. Propose a exportation of the graph drawings with TikZ (a language which allow to add graphics in Latex files).

3.2 Functional requirements

3.2.1 Graph interfaces

- Design user interfaces which allow users can create the graphs that user want.

example:

- User can create a graph by clicking left mouse to create the vertices and after that drag the left mouse from any a vertice to the other vertice to create a edge after drop the left mouse.

- User can input the number of vertices and the edges and after that user click button create graph, the application will create a graph which user inputed.

This two basic options for user interface creating the graphs.

- Design a user interface for solve the shortest path,flow problem.

example:

- User input the start vertice and the end vertice and after that clicking shortest path button that the application will calculate and print the shortest path in graph interface by highlight the edges with red color.

3.2.2 Improve and Optimize the performence of algorithms in library

1. What things which need to Improve and Optimize.
    - Review and remove the duplicate code.
    - Add comment javadoc for the functions.
    - Add more the unit test for detail test functions which added and improved.

3.3 Nonfunctional requirements

3.3.1 Usability

1. This is a desktop application.
    - Runnable on the difference Operating System: Windows,linux or any machine have installed Java virtual machine.
    - Optimize the space memory of the application.
    - The user interface should be simple as posible as and friendly with the user.
    - User can choose the algorithms which user want the application do example: Dijktra or Bellman-Ford for the shortest path problem.
    - Improve and optimize the performence of application.

3.3.2 Reliability

1. Making sure of Reliability
    - Components of the project code will be tested alongside the implementation phase to ensure that they are functional.
    - Final, integrated project Code will be tested with JUnit to ensure that greater than or equal to 80% of the integrated code is covered at run-time, and is functioning properly. The remaining 20% will be inspected through manual testing to ensure the highest chance of being quality code.

3.3.3 Performance

1. Making sure of Performance
    - Drag and drop of the tiles must be smooth without graphical lagging.
    - The time of finding the shortest path must be fast (under 5 seconds).

3.3.4 Supportability

1. Making sure of Supportability
    - The application must not be platform dependent, i.e., it should be able to run on any platform supporting JAVA.

3.3.5 Implementation

1. Making sure of Implementation
    - Project will be implemented in JAVA 8.
    - All project graphical user interfaces will be created using JavaFX.

3.3.6 Interface

3.4 System models

3.4.1 Use case model

Name:	Find The Shortest Path Of Graph
Actor	User
Entry Conditions:	Application is running.
Flow of Events:	1. User choose the algorithm which want to apply. 2. System remember the algorithm which user chose. 3. User click find shortest path button 4. System calculate and give the result. System will highlight with red color for the edges in the shortest path.
Exit Conditions:	The application is now in a new state.

Name:	Draw Vertices
Actor	User
Entry Conditions:	Application is running.
Flow of Events:	1. User choose type of vertice which want to draw. 2. System remember the type which user chose. 3. User click mouse left button on panel. 4. System will draw a vertice on the panel.
Exit Conditions:	The application is now in a new state.

Name:	Draw Edges
Actor	User
Entry Conditions:	Application is running.
Flow of Events:	1. User press left button mouse and alt key on one the start vertices 2. User drag mouse to the end vertice to create a edge by release left button 3. System will create a edge with start and end vertices
Exit Conditions:	The application is now in a new state.

Name:	Delete Edges
Actor	User
Entry Conditions:	Application is running.
Flow of Events:	1. User click right button mouse on edge which user want to delete. 2. System delete edge user clicked right mouse button
Exit Conditions:	The application is now in a new state.

Name:	Delete Vertices
Actor	User
Entry Conditions:	Application is running.
Flow of Events:	1. User click right button mouse on vertices which user want to delete. 2. System delete vertices which user clicked right mouse button
Exit Conditions:	The application is now in a new state.

Name:	Move Vertices
Actor	User
Entry Conditions:	Application is running.
Flow of Events:	1. User press left button mouse on vertices which user want to move. 2. user drag mouse to move the vertices to position where user want 3. System move the vertices
Exit Conditions:	The application is now in a new state.

3.4.2 Object model

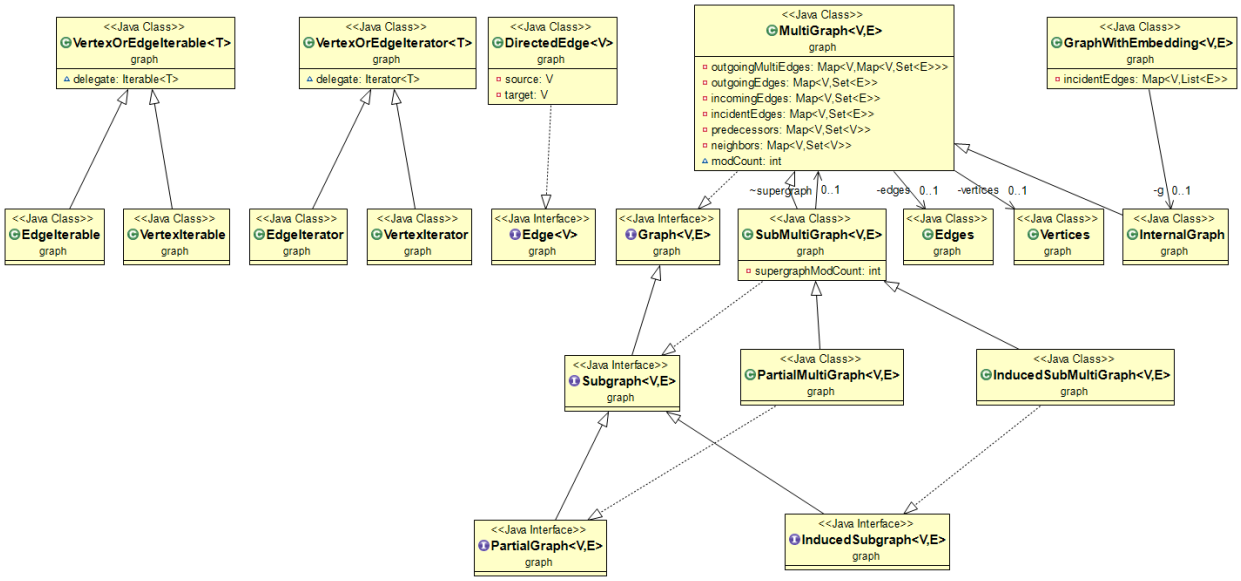

Figure 2: Main

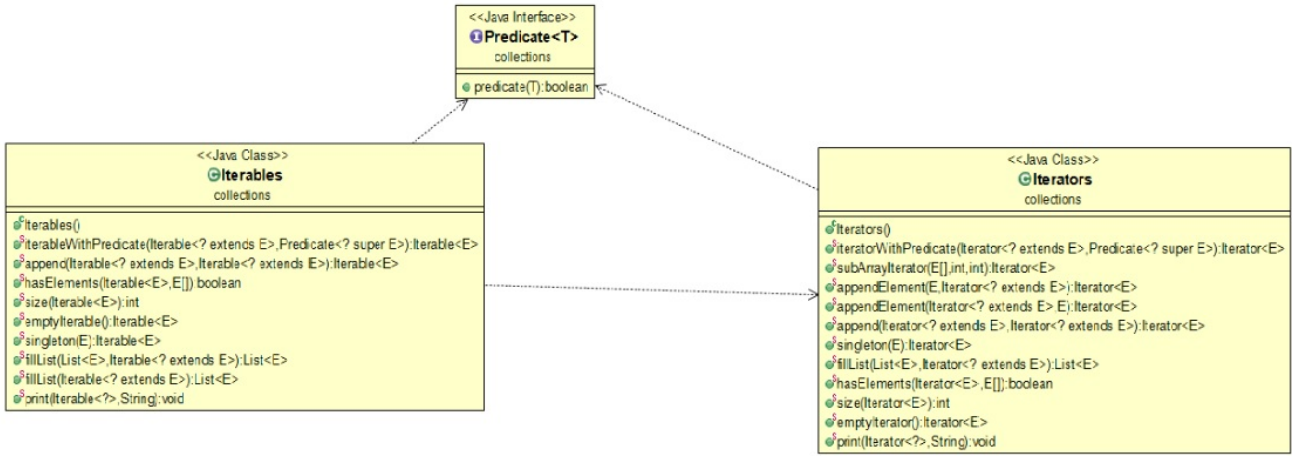

Figure 3: Package Collection

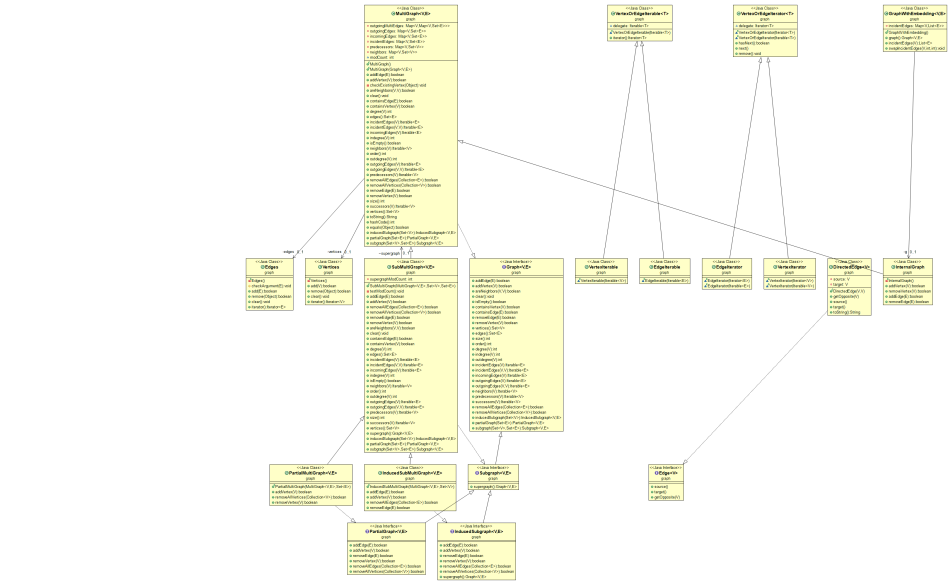

Figure 4: Package Graph

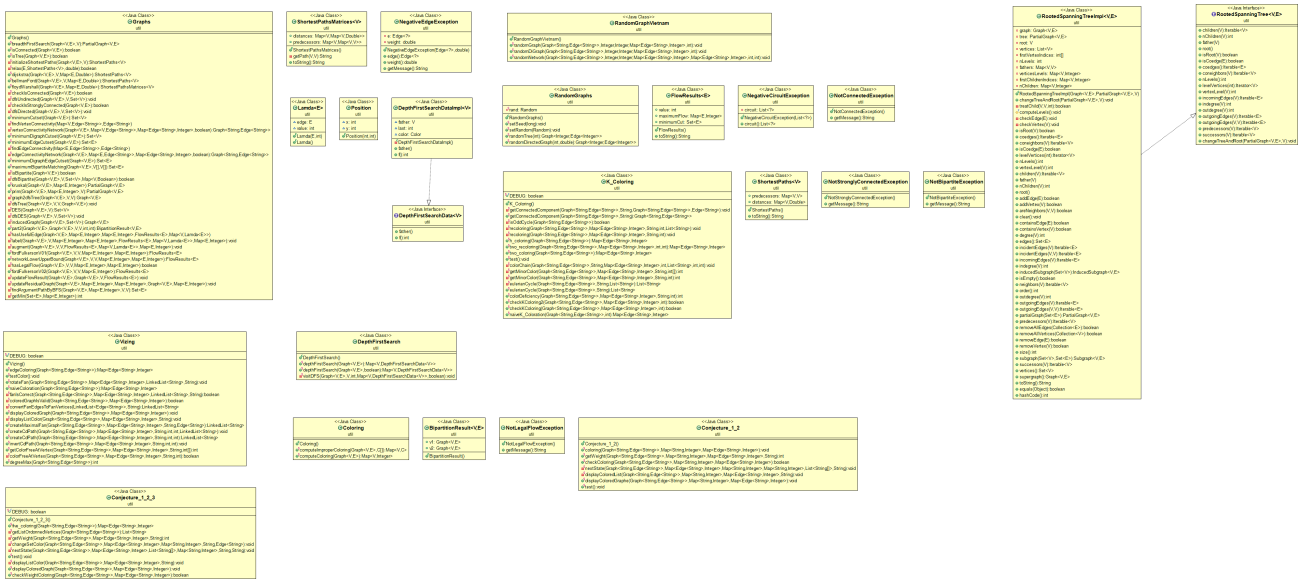

Figure 5: Package Util

3.4.3 User Interface

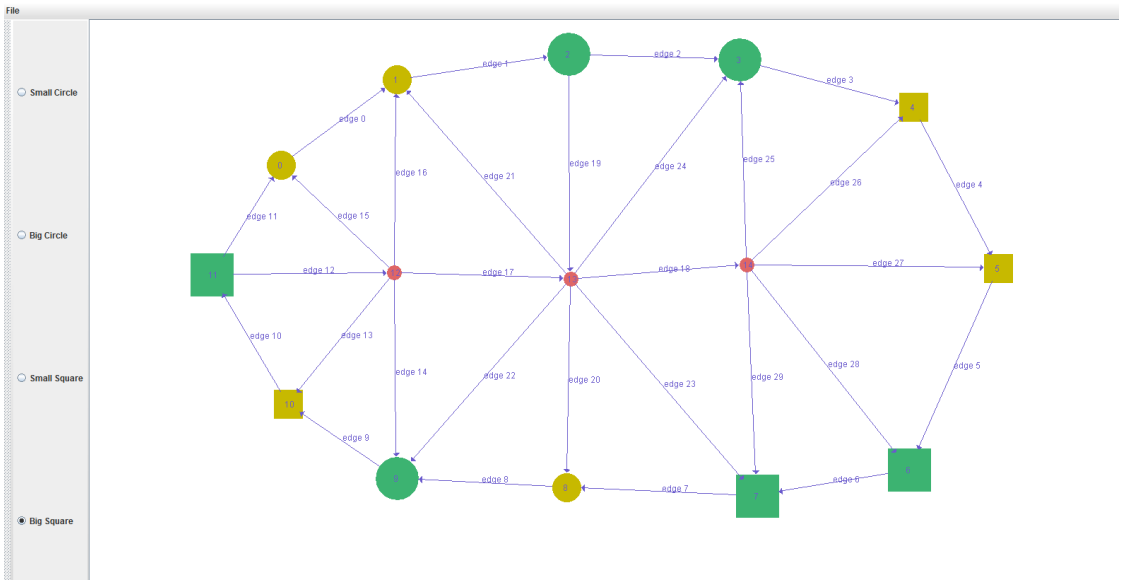

Figure 6: User Interface

4 Architecture

4.1 Introduction

This document provides a high level overview and explains the architecture of Graph Application. The document defines goals of the architecture, the use cases supported by the system, architectural styles and components that have been selected. The document provides a rationale for the architecture and design decisions made from the conceptual idea to its implementation.

4.1.1 Purpose

The Software Architecture Document (SAD) provides a comprehensive architectural overview of the Graph Application It presents a number of different architectural views to depict the different aspects of the system.

4.1.2 Scope

The scope of this SAD is to explain the architecture of the Graph Application. This document describes the various aspects of the Graph application design that are considered to be architecturally significant. These elements and behaviors are fundamental for guiding the construction of the Graph application and for understanding this project as a whole.

4.2 Architectural Representation

4.2.1 Use Case view

Audience: all the stakeholders of the system, including the end-users.

Area: describes the set of scenarios and/or use cases that represent some significant, central functionality of the system. Describes the actors and use cases for the system, this view presents the needs of the user and is elaborated further at the design level to describe discrete flows and constraints in more detail. This domain vocabulary is independent of any processing model or representational syntax (i.e. XML).

4.2.2 Logical view

Audience: Designers.

Area: Functional Requirements: describes the design's object model. Also describes the most important use-case realizations and business requirements of the system.

Related Artifacts: Design model

4.3 Architectural Goals and Constraints

There are some key requirements and system constraints that have a significant bearing on the architecture. They are:

1. The system is meant as a proof of concept for a more complete project prediction system to be built in the future. Therefore one of the primary stakeholders in this document and the system as a whole are future architects and designers, not necessarily users as is normally the case. As a result, one goal of this document is to be useful to future architects and designers.

2. The system will be written using JAVA 8 technologies but will use an open source JavaFX for Design Graphic user interface and will be runable to operating system with Java Virtual Machine be installed.

3. The Application may be reference to the third-party for graph such as JGraphT.

4.4 Use-Case Realizations

4.4.1 Find the shortest path of graph

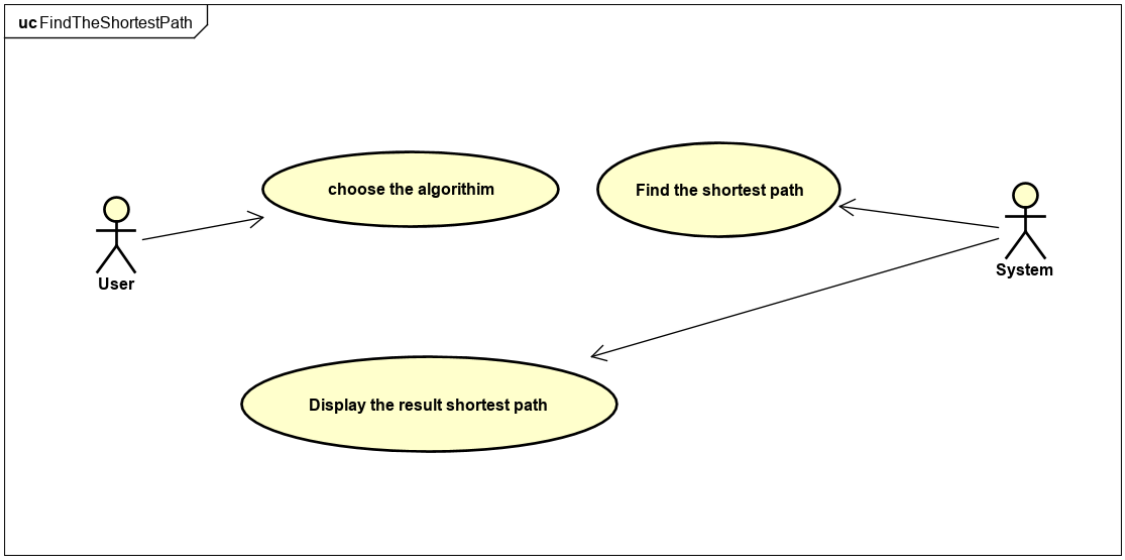

Figure 7: Find the shortest path of graph

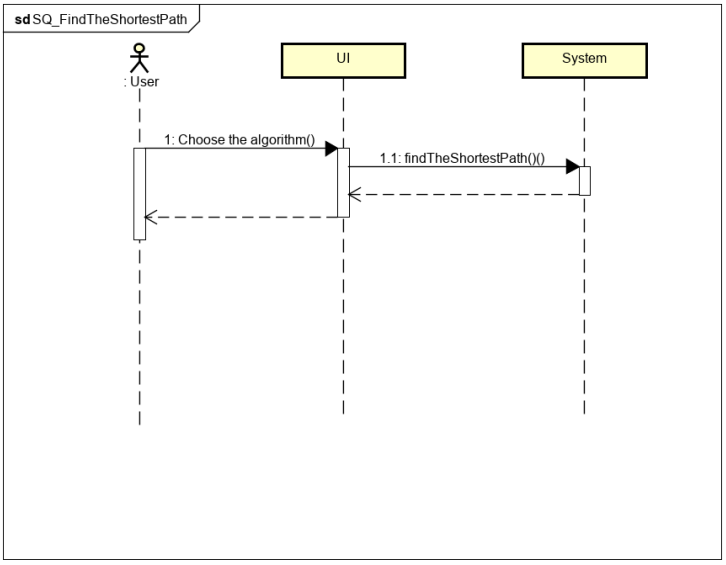

Figure 8: Sequence FindTheShortestPath

4.4.2 Draw Vertices

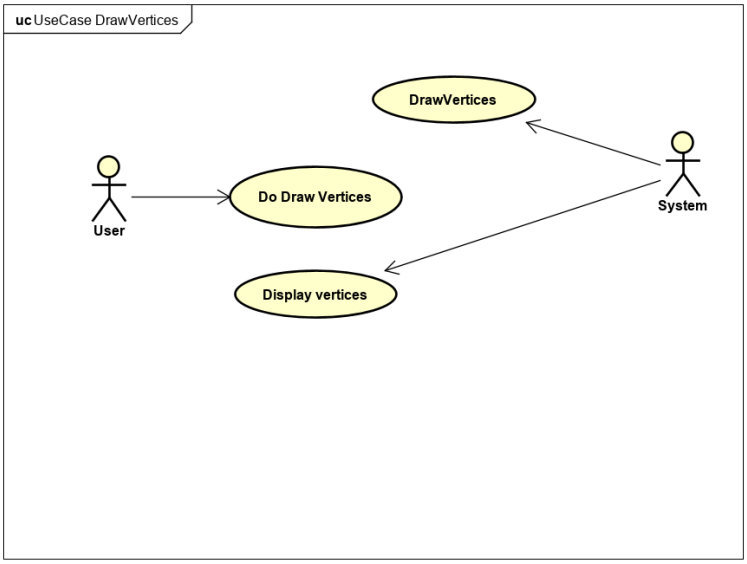

Figure 9: UseCaseDrawVertices

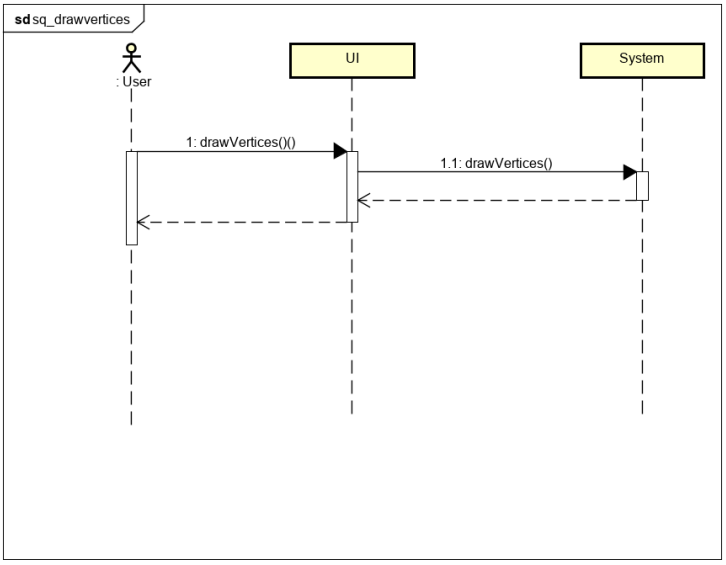

Figure 10: Sequence drawvertices

4.4.3 Draw Edges

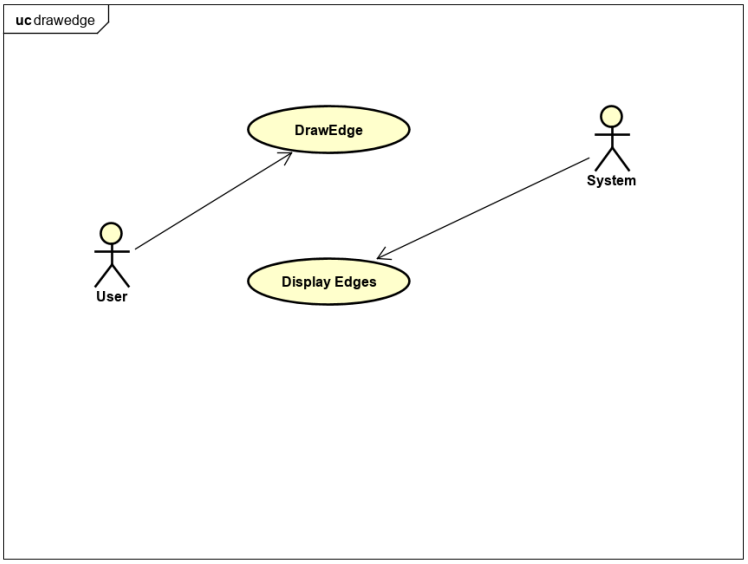

Figure 11: UseCaseDrawEdge

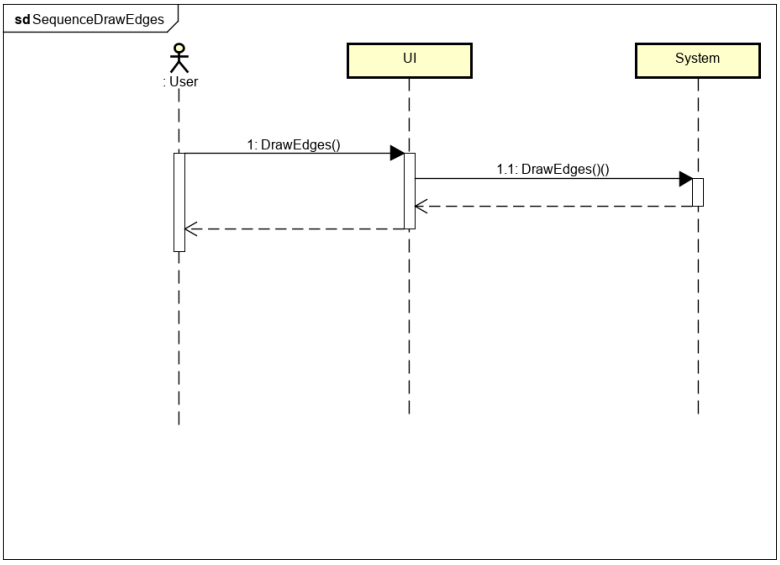

Figure 12: SequenceDrawEdges

4.4.4 Delete Edges

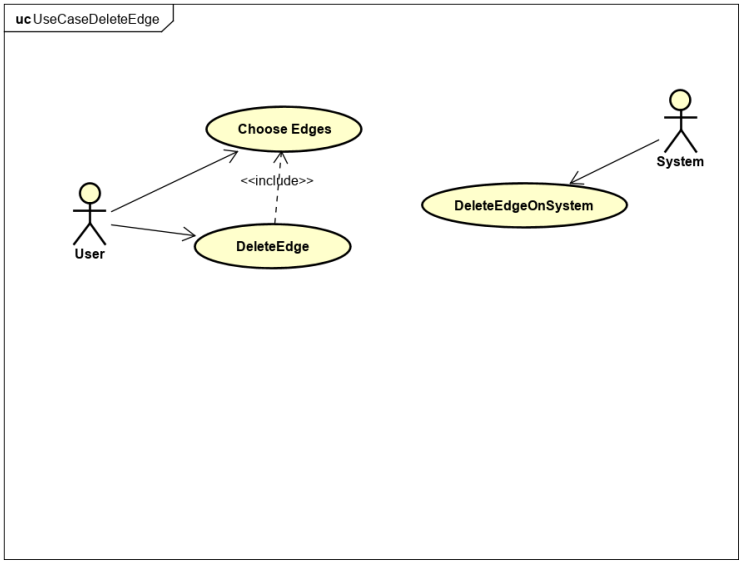

Figure 13: UseCaseDeleteEdge

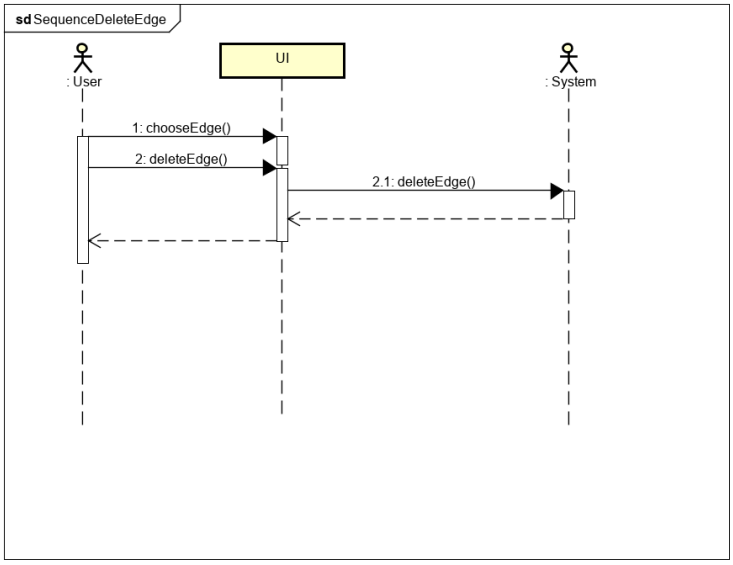

Figure 14: SequenceDeleteEdge

4.4.5 Delete Vertices

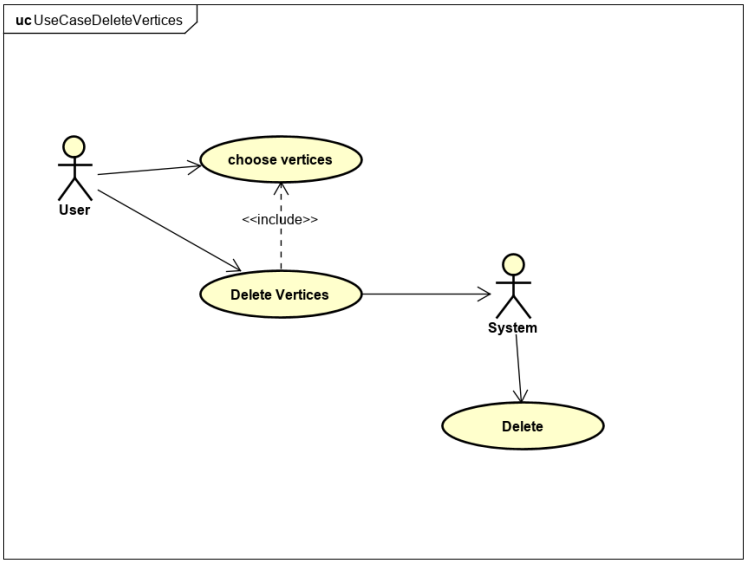

Figure 15: UseCaseDeleteVertices

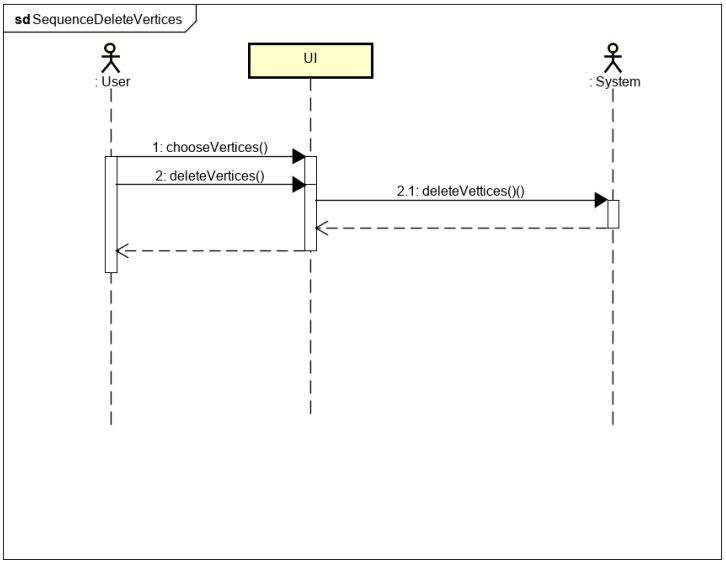

Figure 16: SequenceDeleteVertices

4.4.6 Move Vertices

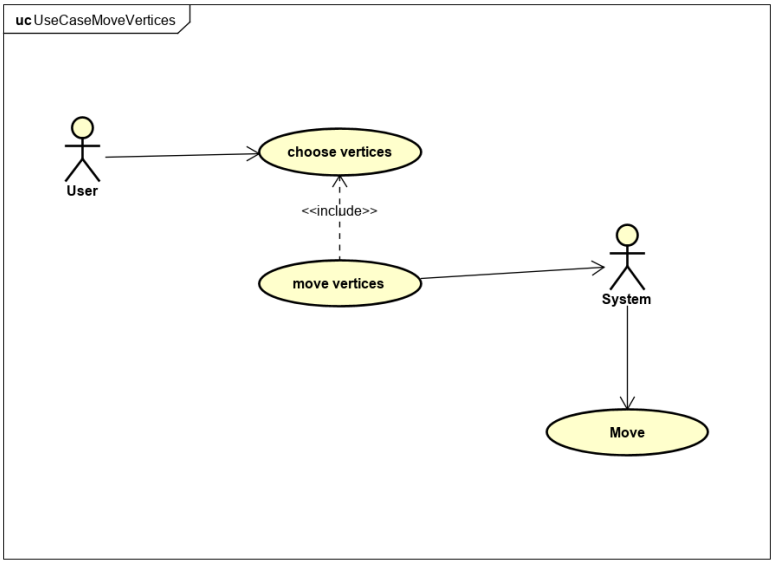

Figure 17: UseCaseMoveVertices

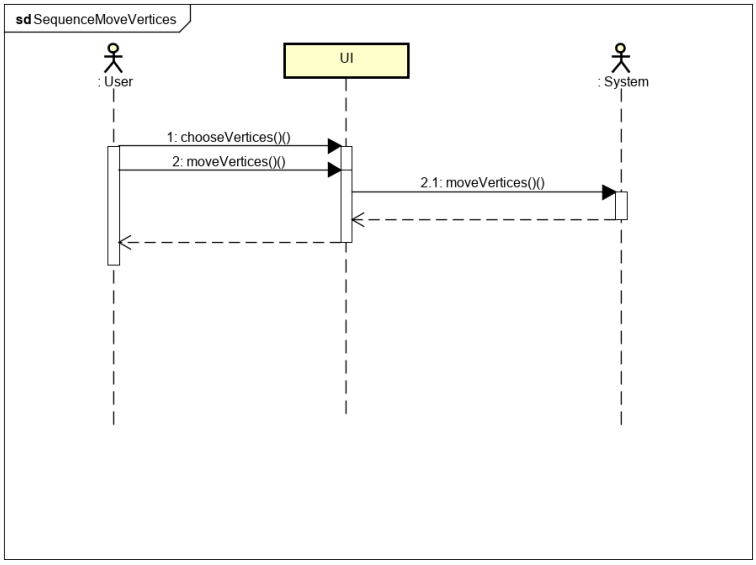

Figure 18: SequenceMoveVertices

5 Application

5.1 Git Hub

5.1.1 Repository

Repository: https://github.com/nhat2211/GraF.git

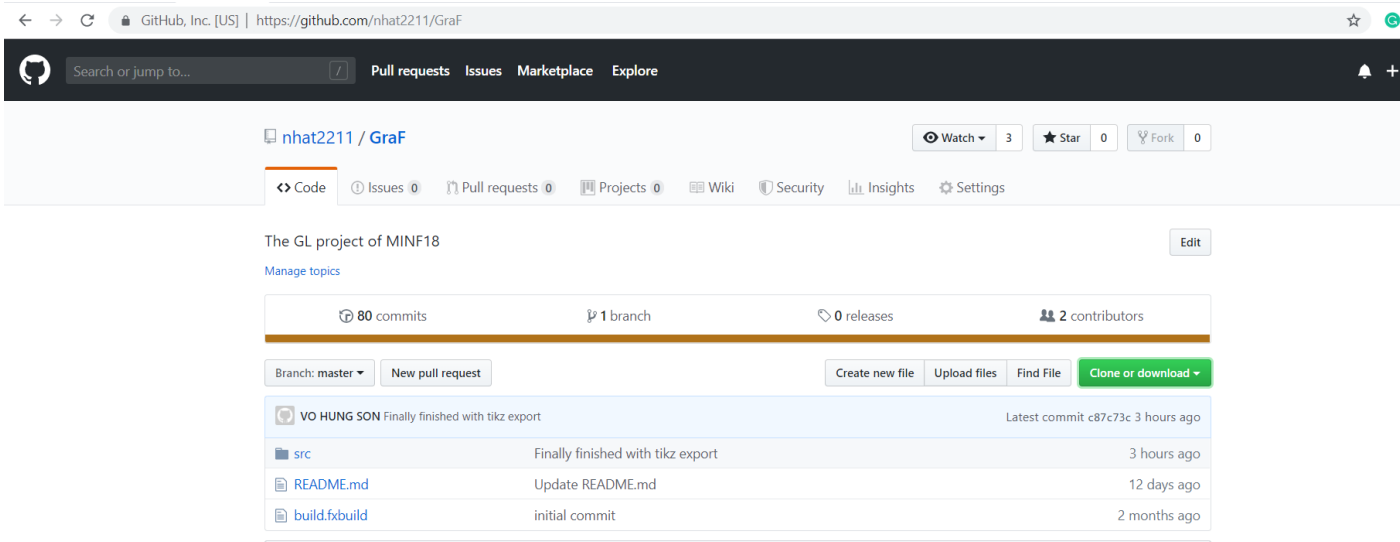

Figure 19: Repository

5.2 Requirement

The GL project of MINF18 here is a set of requests for the project :

1- create a gitHub project with name GraF (for Graph Framework). My login in gitHub is baudon (for the invitation) and the associated adress olivier.baudon@labri.fr Please tell me each time you want that I take a look on your code.

2- Provide an interface which allow to create graphs where a vertex is represented by a circle and an edge by a straight line.

3- Give the possibility to move vertices.

4- Give the possibility to change the form of the vertices and the edges (arc, or segments with intermediate points movable by the user), and to have oriented edges.

5- Give the possibility to suppress a vertex or an edge.

6- Add labels to vertices and edges.

7- Give the possibility to the user to change the position of the labels, and the value of the labels.

Remark : I propose that you provide a toolbar on the left side of the window to propose the different types of vertices and edges and to change the effect of a click (for example, an icon with scissors (or death's head) means supress, an icon with a cross means add an intermediate point to an edge, ...)

For each version, you may also provide the tkz version of the drawing, using a menu for the export.

5.3 Architecture

We use Model-View-Controller in the application development.

1. After that the user see on **VIEW**

2. User use **CONTRONLLER**

3. Apply data (Update,edit,delete,etc..), data on **MODEL** changed

4. Display data of **MODEL** on **VIEW**

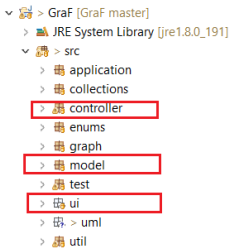

Figure 20: Model-View-Controller Project Structure

5.4 Framework

Why choose JavaFX for Developing Application?:

Our experience proves that implementing an employee desktop front end with native technology is a valid approach and that JavaFX is a good fit.

- JavaFX is available on the leading desktop operating systems (Windows, Linux, and . Mac OS X).
- Although it has gone through some painful changes, its evolution proves its vendor's level of commitment.
- As the successor to Swing, it is being used by an increasing number of Java developers. Regardless of its future, it will benefit from a strong developer community.
- Compared to Swing, it provides a clear and clean architecture and features many enhancements: styling, event management, transitions, scene graph—to name a few.
- It provides the possibility of developing up-to-date user interfaces with animations, multitouch, and the like.
- It is based on a clear and clean language: Java.
- It provides all the professional Java tooling required to debug, analyze, profile, and log a client application.
- It enables a simple app-like installation on the client side, without any prerequisites.
- In short, we believe that JavaFX is the only native UI environment that provides all these features today.

Source: https://www.oracle.com/technetwork/articles/java/casa-1919152.html

5.5 UML

1. Class Diagram:

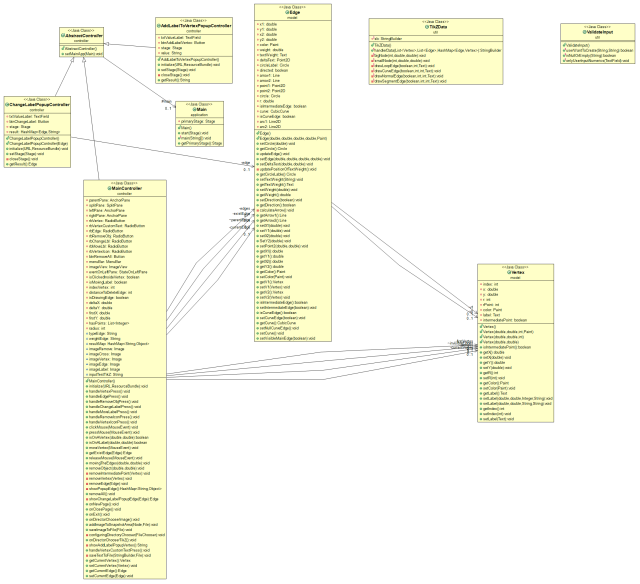

Figure 21: Class Diagram

5.6 IDE

Eclipse

Why choose Eclipse IDE for development:

1. Eclipse is used widely in developer community in the world.
2. Eclipse support many plugins for development.
3. Eclipse is free to download and install.

5.7 Unit Test

Why choose Junit framework for unit test.

1. JUnit is an open source framework, which is used for writing and running tests.
2. Provides annotations to identify test methods.
3. Provides assertions for testing expected results.
4. Provides test runners for running tests.
5. JUnit tests allow you to write codes faster, which increases quality.
6. JUnit is elegantly simple. It is less complex and takes less time.
7. JUnit tests can be run automatically and they check their own results and provide immediate feedback. There's no need to manually comb through a report of test results.
8. JUnit tests can be organized into test suites containing test cases and even other test suites.
9. JUnit shows test progress in a bar that is green if the test is running smoothly, and it turns red when a test fails.

5.8 Functions

5.8.1 Draw vertex

Name:	Draw Vertices
Actor	User
Entry Conditions:	Application is running.
Flow of Events:	1. User choose vertex option on left pane. 2. System remember the type which user chose. 3. User press mouse left button on panel. 4. System will draw a vertice on the panel by a circle.
Exit Conditions:	The application is now in a new state.

Figure 22: Draw vertex

5.8.2 Draw vertex custom text

Name:	Draw vertex custom text
Actor	User
Entry Conditions:	Application is running.
Flow of Events:	1. User choose vertex custom text option on left pane. 2. System remember the type which user chose. 3. User press mouse left button on panel. 4. The popup add vertex be showed 5. User type text in textbox 6. User click add button 7. System draw a vertex with label be text which user input
Exit Conditions:	The application is now in a new state.

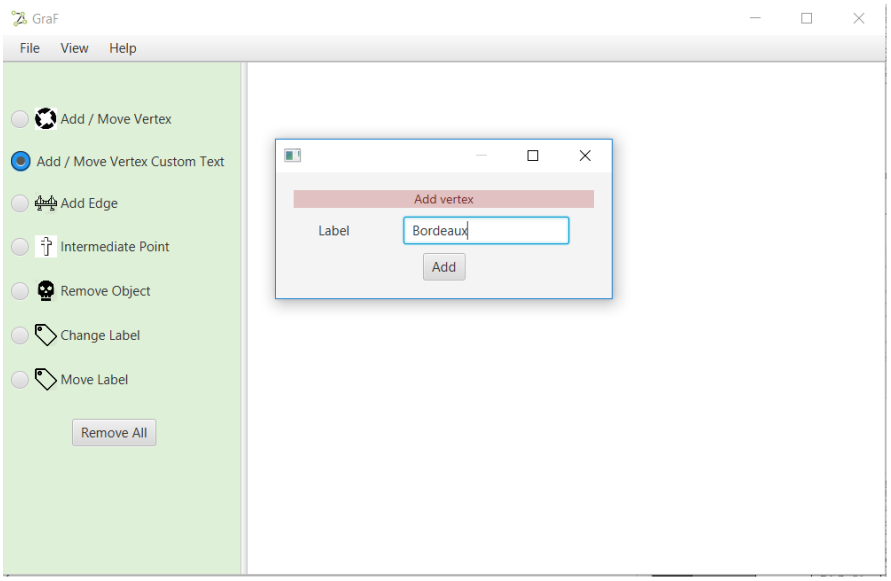

Figure 23: Draw vertex with text

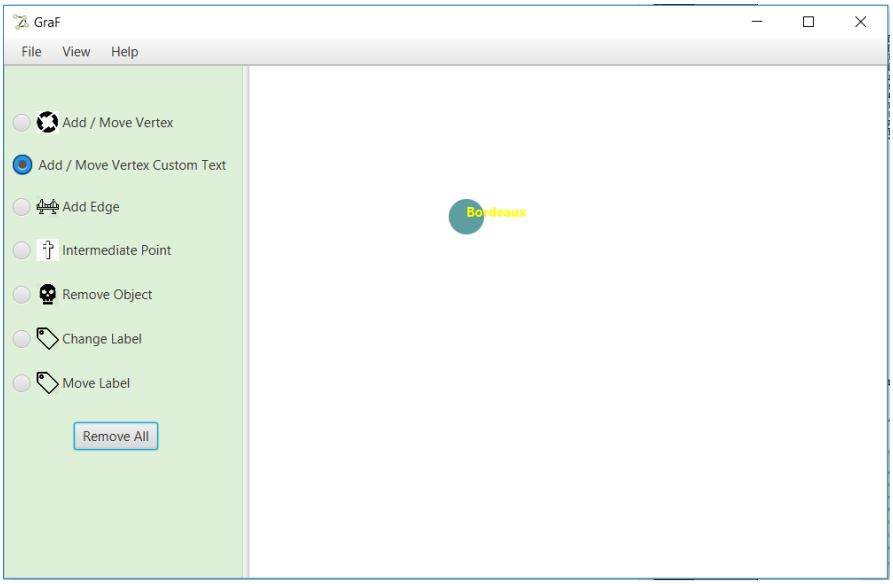

Figure 24: Draw vertex with text

5.8.3 Move vertex

Name:	Move Vertex
Actor	User
Entry Conditions:	Existing at least one vertex for moving.
Flow of Events:	1. User choose move vertex option on left pane. 2. System remember the type which user chose. 3. User press and hold mouse left button on vertex. 4. User hold press the left mouse on the vertex and move the mouse for moving vertex.
Exit Conditions:	The application is now in a new state.

Figure 25: Move vertex

5.8.4 Draw edge

Name:	Draw edge
Actor	User
Entry Conditions:	Existing two at least two vertex on right pane.
Flow of Events:	1. User choose add edge option on left pane. 2. System remember the type which user chose. 3. User press and hold left mouse on the first vertex. 4. User hold press left mouse and move cursor to inside the second vertex. 5. User release mouse. 6. The popup Add edge be showed. 7. User input the weight of edge. 8. User click Directed button or Undirected button for create edge 9. System draw a edge.
Exit Conditions:	The application is now in a new state.

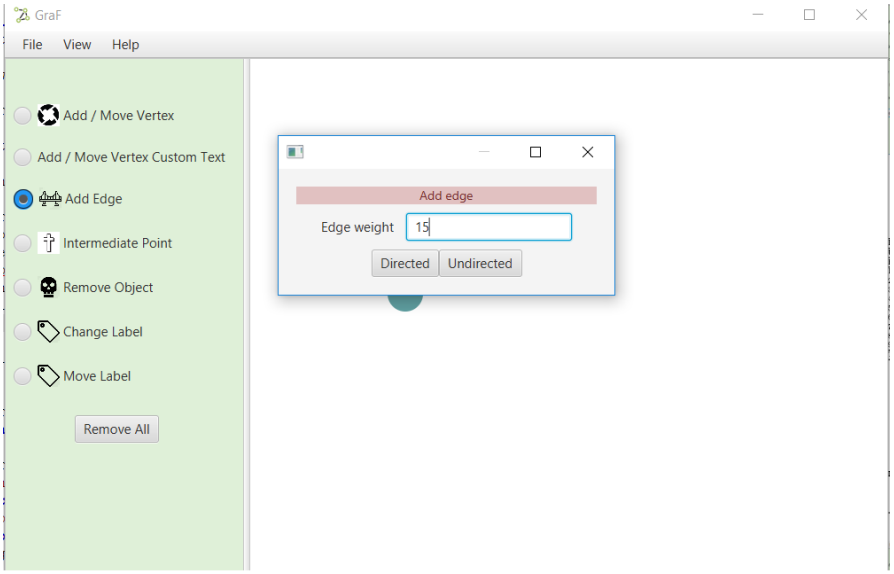

Figure 26: Add edge

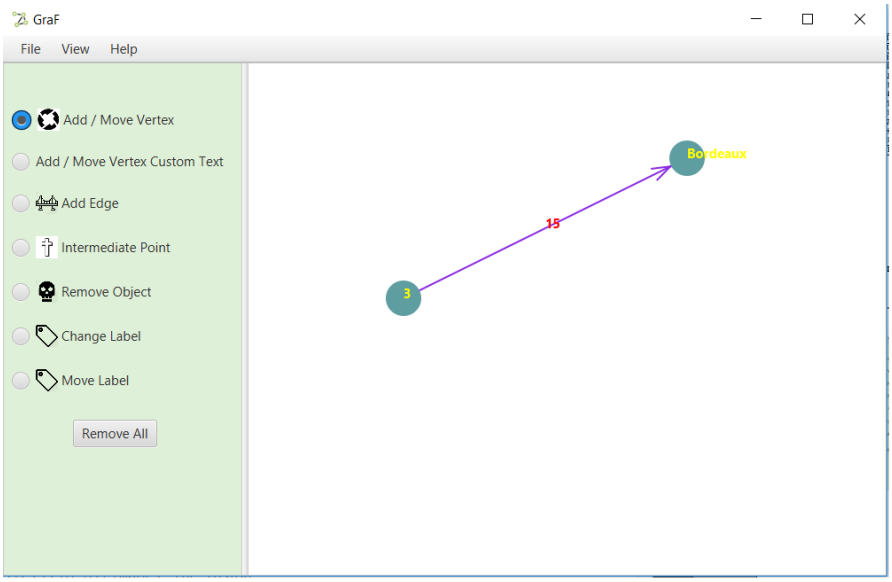

Figure 27: Add edge

5.8.5 Intermediate Point

Name:	Intermediate Point
Actor	User
Entry Conditions:	Existing at least one edge on right pane.
Flow of Events:	1. User choose Intermediate Point option on left pane. 2. System remember the type which user chose. 3. User press left mouse on a edge 4. System draw a Intermediate Point on this edge
Exit Conditions:	The application is now in a new state.

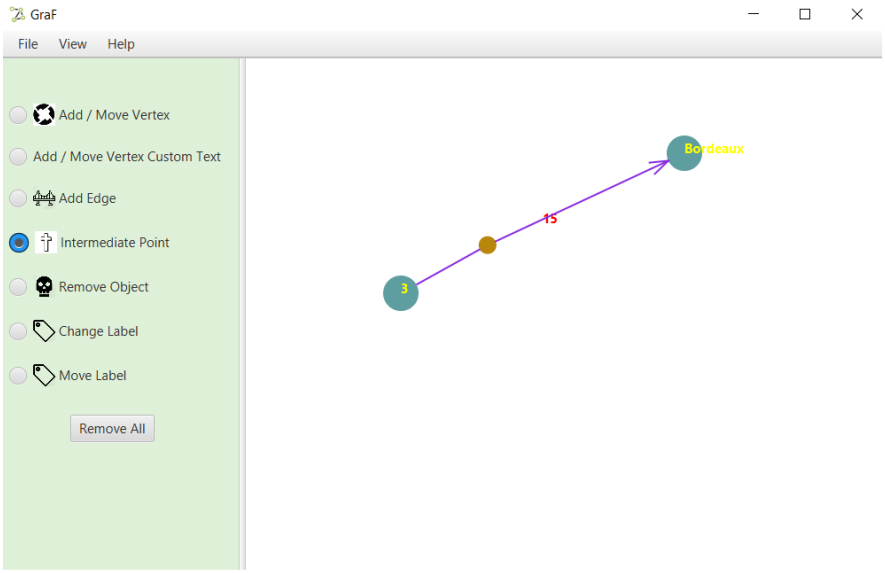

Figure 28: Intermediate Point

5.8.6 Remove Object

Name:	Remove Object
Actor	User
Entry Conditions:	Existing at least one object on right pane.
Flow of Events:	1. User choose Remove Object option on left pane. 2. System remember the type which user chose. 3. User press left mouse on a object 4. System remove object
Exit Conditions:	The application is now in a new state.

5.9 TikZ package

Source Reference: www.overleaf.com/learn/latex/
Tikz is probably the most complex and powerful tool to create graphic elements in LATEX. In this

article some of the basics will be explained: lines, dots, curves, circles, rectangles, etc by means of simple examples.

5.9.1 Introduction

We can create graphic elements easily by defining some of their key properties. First, you declare a tikzpicture environment, before this you must include the line usepackagetikz in the preamble of your document.

5.9.2 Basic geometric shapes: Circles, ellipses and polygons

```
\begin{tikzpicture}

\draw[blue, very thick] (0,0) rectangle (3,2);
\draw[orange, ultra thick] (4,0) -- (6,0) -- (5.7,2) -- cycle;

\end{tikzpicture}
```

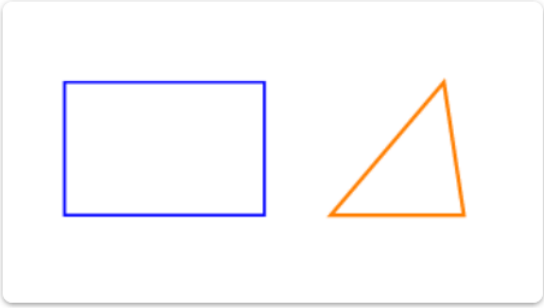

Figure 29:

5.9.3 Diagrams

The nodes are probably the most versatile elements in Tikz. We've already used one node in the introduction to add some text to the figure. In the next example nodes will be used to create a diagram.

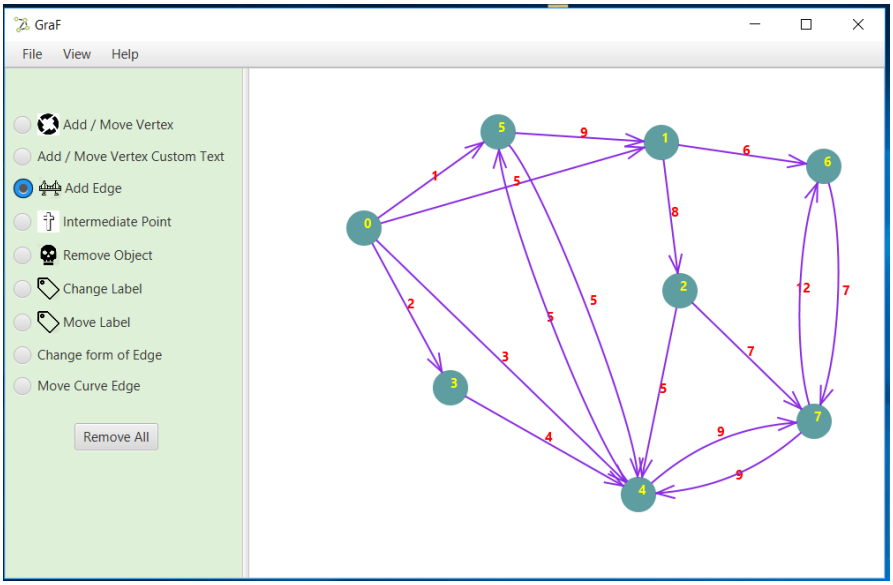

Figure 30: The Graph be drawed by GraF application

```
\documentclass{article}
\usepackage[utf8]{inputenc}
\usepackage{tikz}
\usetikzlibrary{positioning}
\begin{document}
\begin{figure}
\begin{tikzpicture}
[bigNode/.style={circle, draw=green!60, fill=green!5, thick, minimum size=7mm}, smallNode/.style={circle, draw=blue!60, fill=blue!20, thick, minimum size=3mm},]
        \node[bigNode] (0) at ( 1.625, 7.7375) {0};
        \node[bigNode] (1) at ( 5.8375, 8.95) {1};
        \node[bigNode] (2) at ( 6.1, 6.85) {2};
        \node[bigNode] (3) at ( 2.85, 5.475) {3};
        \node[bigNode] (4) at ( 5.5125, 3.9625000000000004) {4};
        \node[bigNode] (5) at ( 3.525, 9.1) {5};
        \node[bigNode] (6) at ( 8.1375, 8.6125) {6};
        \node[bigNode] (7) at ( 8.0, 5.0) {7};
        \draw [->] (0) -- (5) node[midway,fill=red!20] {1};
        \draw [->] (0) -- (3) node[midway,fill=red!20] {2};
        \draw [->] (0) -- (4) node[midway,fill=red!20] {3};
        \draw [->] (3) -- (4) node[midway,fill=red!20] {4};
        \draw [->] (4) to [bend left] node[midway,fill=red!20] {5} (5);
        \draw [->] (0) -- (1) node[midway,fill=red!20] {5};
        \draw [->] (1) -- (6) node[midway,fill=red!20] {6};
        \draw [->] (6) to [bend left] node[midway,fill=red!20] {7} (7);
        \draw [->] (2) -- (7) node[midway,fill=red!20] {7};
        \draw [->] (1) -- (2) node[midway,fill=red!20] {8};
        \draw [->] (5) -- (1) node[midway,fill=red!20] {9};
        \draw [->] (7) to [bend left] node[midway,fill=red!20] {12} (6);
        \draw [->] (5) to [bend left] node[midway,fill=red!20] {5} (4);
        \draw [->] (2) -- (4) node[midway,fill=red!20] {5};
        \draw [->] (4) to [bend left] node[midway,fill=red!20] {9} (7);
        \draw [->] (7) to [bend left] node[midway,fill=red!20] {9} (4);
\end{tikzpicture}
\end{figure}
\end{document}
```

Figure 31: TikZ Generate export code in latex

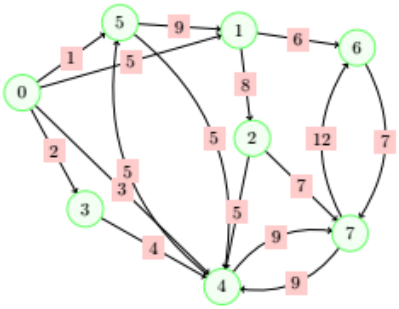

Figure 32: Generate The Graph By TikZ