

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Projekt 1

Graph Viewer

Projektdokumentation

19.01.2014

Berger Stefan (bergs13)

Menzi Stephan (menzs2)

Gugelmann Christian (gugec1)

Inhalt

[1 Allgemeines 3](#_Toc377938392)

[1.1 Projektbeschreibung 3](#_Toc377938393)

[1.2 Projektorganisation 3](#_Toc377938394)

[2 Anforderungen 4](#_Toc377938395)

[2.1 Erstellen und Bearbeiten von Graphen 4](#_Toc377938396)

[2.2 Import/Export von Graphen 4](#_Toc377938397)

[2.3 Visualisierung von Graphen Algorithmen 4](#_Toc377938398)

[3 Umsetzung 5](#_Toc377938399)

[3.1 Klassendiagramm (Bild liegt separat auch noch bei) 5](#_Toc377938400)

[3.2 GUI 6](#_Toc377938401)

[3.3 Graph-, Vertex-, Edge-Formate 6](#_Toc377938402)

[3.3.1 Attribute Graph (GraphFormat.java) 7](#_Toc377938403)

[3.3.2 Attribute Vertex (VertexFormat.java) 7](#_Toc377938404)

[3.3.3 Attribute Edge (EdgeFormat.java) 7](#_Toc377938405)

[3.4 Graph Editor/Visualisierung Graph 7](#_Toc377938406)

[3.4.1 Aufbau Visualisierung 7](#_Toc377938407)

[3.4.2 Recorder 7](#_Toc377938408)

[3.5 Import/Export 8](#_Toc377938409)

[3.6 Spezielles 8](#_Toc377938410)

[4 Arbeitsaufteilung 9](#_Toc377938411)

[5 Benutzerhandbuch 10](#_Toc377938412)

[5.1 Editieren eines Graphen 10](#_Toc377938413)

[5.2 Neuen Graphen erstellen 12](#_Toc377938414)

[5.3 Graph speichern/laden 12](#_Toc377938415)

[5.4 Visualisierung von Graphen Algorithmen 13](#_Toc377938416)

[5.5 Attribute zur Visualisierung für bestehende Algorithmen/den Custom Algorithm setzen 13](#_Toc377938417)

# Allgemeines

Dieses Dokument beinhaltet die grundlegenden Anforderungen, welche es im Rahmen des Projektes *Projekt 1* umzusetzen gilt, beschreibt die gemachten Ideen und Überlegungen und deren Umsetzung.

## Projektbeschreibung

Projektbeschreibung gemäss Aufgabenstellung:

*Es soll eine Software erstellt werden, welche Graphen darstellen kann, bzw. es erlaubt, Graphen mit einem graphischen Editor zu spezifizieren. Gleichzeitig soll die Software der Visualisierung der Traversierung von Graphen dienen. Ein Algorithmus, wie etwa derjenige von Dijkstra soll mit diesem Werkzeug so auf einfache Weise visualisierbar werden. Das Werkzeug soll sich als didaktisches Hilfsmittel bzw. als Debugging Tool für beliebige Graphen-Algorithmen eignen.*

## Projektorganisation

|  |  |
| --- | --- |
| Projektteam | Stefan Berger, Stephan Menzi, Christian Gugelmann |
| Projektbetreuung | Peter Schwab |

# Anforderungen

Das Graph Viewer Tool bietet dem Anwender eine graphische Oberfläche, mit welcher er Graphen erstellen und bearbeiten kann. Erstellte Graphen können abgespeichert und wieder geladen werden. Zusätzlich bietet das Tool die Visualisierung von vordefinierten oder eigenen Graphen Algorithmen anhand der erstellten Graphen an.

## Erstellen und Bearbeiten von Graphen

Graphen können auf einfach Art & Weise erstellt und bearbeitet werden. Dazu werden folgende Funktionalitäten angeboten:

* Hinzufügen/Entfernen von Eckpunkten
* Hinzufügen/Entfernen von Kanten
* Auswahl gerichtete/ungerichtete Graphen
* Benennung der Eckpunkte
* Gewichtung der Kanten
* Verschiebung der Eckpunkte mittels Drag & Drop
* Darstellung der Eckpunkte, Kanten und Labels mittels gewünschter Farbe

## Import/Export von Graphen

Erstellte Graphen können abgespeichert und für eine spätere Verwendung wieder geladen werden. Das Abspeichern erfolgt als XML-File.

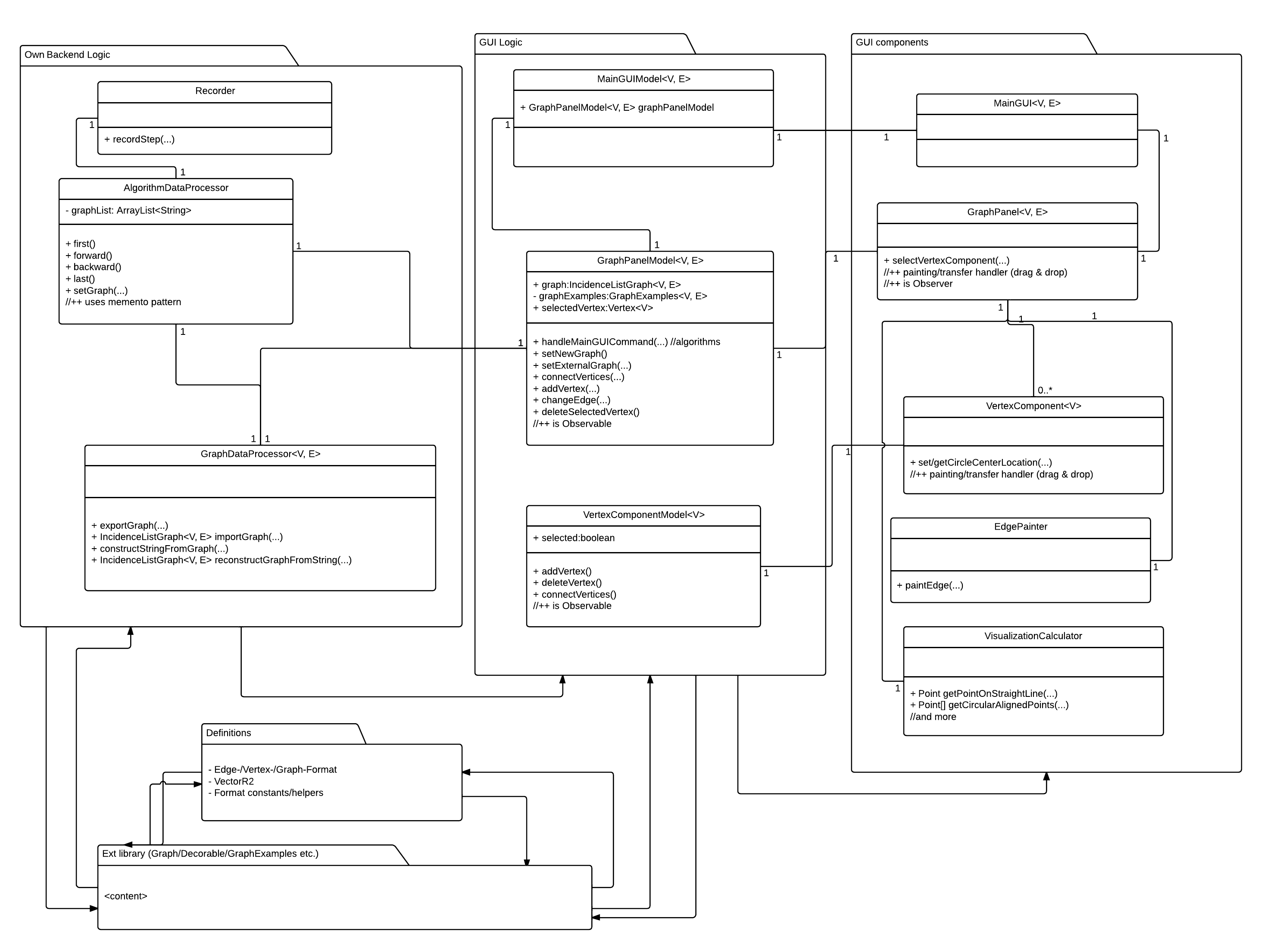
## Visualisierung von Graphen Algorithmen

Vordefinierte (Dijkstra, Kruskal, BFS) oder eigene (Custom Algorithm) Graphen Algorithmen können mittels dem Graph Viewer Tool visualisiert werden. Das Tool bietet die Möglichkeit, die Graphen Algorithmen Step-by-Step zu durchlaufen (vorwärts und rückwärts) und stellt den aktuellen Stand graphisch dar (Highlighting). Dazu gehören:

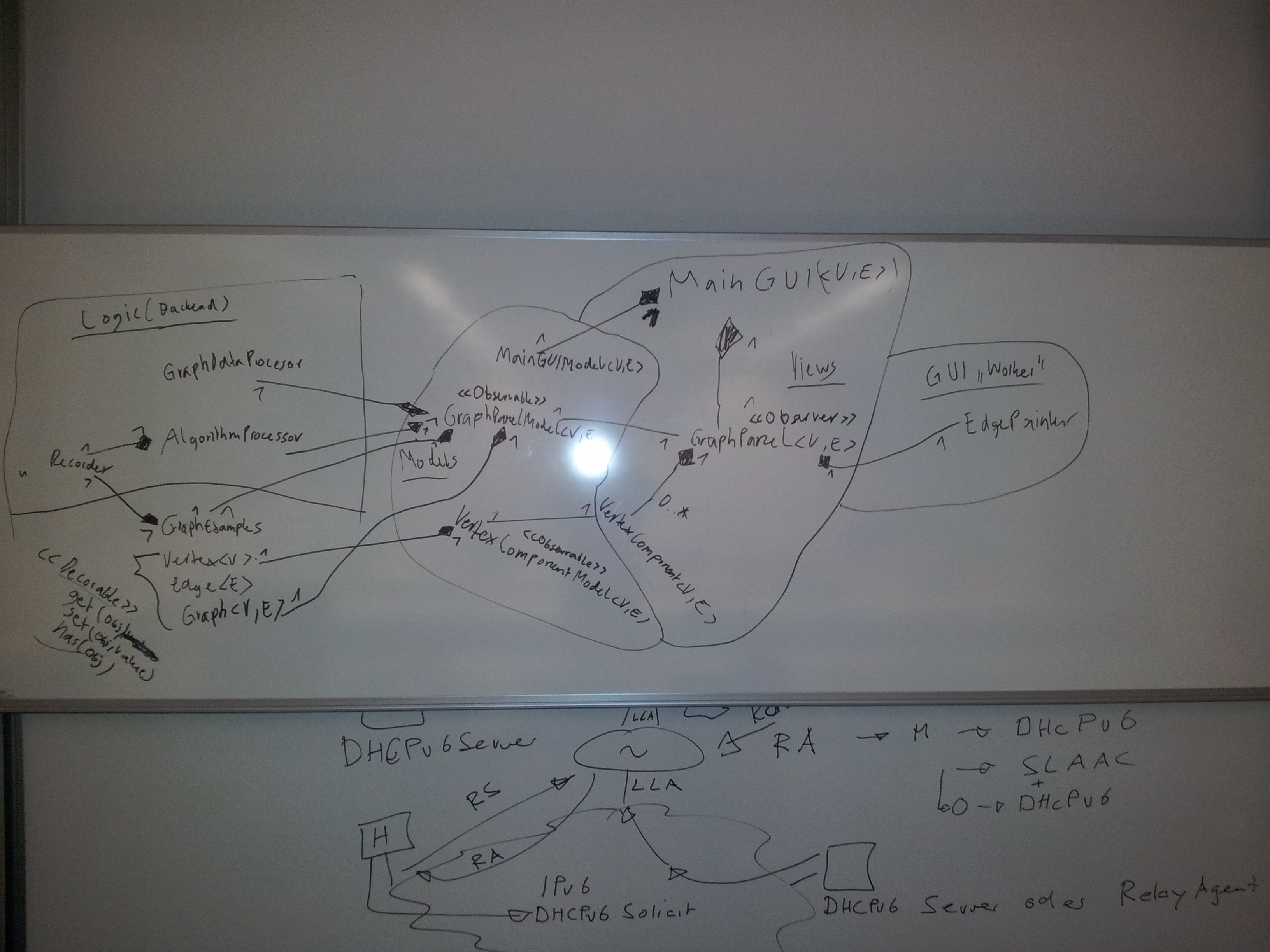
* Aktiver Eckpunkt/aktive Kante
* Besuchter Eckpunkt/besuchte Kante
* Gewichtung der Kanten in Algorithmus anpassbar
* Distanz zu einem Eckpunkt in Algorithmus anpassbar

# Umsetzung

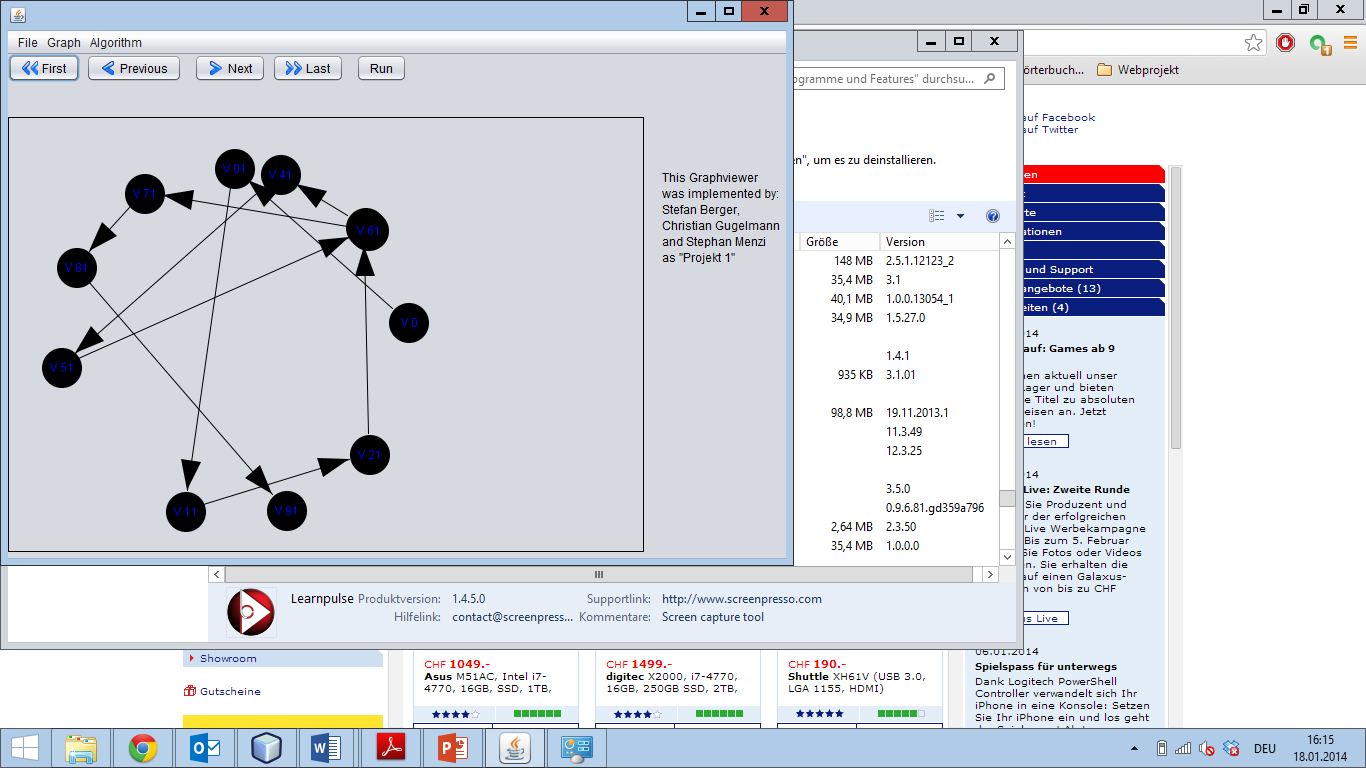
## Klassendiagramm (Bild liegt separat auch noch bei)



Erste Skizze des Diagrams (Ebenfalls separat vorhanden)



## GUI



Das Haupt-GUI, ausser dem zentralen GraphPanel zur Anzeige des Graphen, wurde mittels NetBeans-GUI-Editor erstellt.

## Graph-, Vertex-, Edge-Formate

Graphen, Vertices und Edges besitzen verschiedene Attribute, welche jeweils in einer separaten Format-Klasse definiert werden und der entsprechenden Komponente als Attribut gesetzt werden (Decorable).

### Attribute Graph (GraphFormat.java)

* Labels sichtbar/unsichtbar (isLabelVisible)?
* Gerichteter Graph (isDirected)?
* Gewichteter Graph (isWeighted)?
* Aktive Farbe (activeColor)
* Farbe besuchtes Element (visitedColor)
* Farbe unbesuchtes Element (unvisitedColor)

### Attribute Vertex (VertexFormat.java)

* Position Mitte des Punktes (centerPoint)
* Label (label)
* Aktiv/inaktiv (active)
* Besucht/unbesucht (visited)
* Distanz (distance)

### Attribute Edge (EdgeFormat.java)

* Aktiv/inaktiv (active)
* Besucht/unbesucht (visited)
* Ausgangspunkt (fromPoint)
* Zielpunkt (toPoint)

## Graph Editor/Visualisierung Graph

### Aufbau Visualisierung

Auf dem Haupt-GUI (Erstellt mit Netbeans) befindet sich ein Panel für die Visualisierung der Graphen. Dieses Panel stellt mehrere Vertex-Komponenten dar (JComponent eines decorable Vertex). Die Vertex-Komponenten sind via Drag & Drop verschiebbar (Transferable interace) Der EdgePainter zeichnet die Verbindung der verschiedenen Vertices stets nach (Zeichnet eine Liste von Edges, welche decorable sind). Die Darstellung und Position der verschiedenen Elemente wird aus ihrem Format, welches via decorable interface gesetzt wird, entnommen.

### Recorder

Um den Algorithmus zu visualisieren wird der Graph im jeweils aktuellen Zustand abgespeichert und Schritt für Schritt wieder geladen.

Dazu muss im Algorithmus mit Hilfe einer Instanz der Klasse Recorder ein recordStep() gesetzt werden. In der Klasse GraphExamples wird beim Start des Programms ein solcher Recorder instantziert. Die Verantwortung, dass die richtigen Zustände des Graphen aufgezeichnet werden liegt deshalb beim Ersteller des Algorithmus.

Das verwalten und speichern der Graphen wird in der Klasse AlgorithmDataprocessor gemacht.

Der Graph wird mit Hilfe der unten beschriebenen Klasse GraphDataProcessor in eine String umgewandelt und in einer Arraylist gespeichert. Beim Starten eines Algorithmus wird diese Arraylist geleert. Anschliessend wird der Algorithmus ausgeführt und die Graphen der Reihe nach in der Arraylist gespeichert. Am Schluss wird der erste gespeicherte Graph angezeigt.

Beim vor und zurückspielen des der einzelnen Schritte wird der gespeicherte String mit Hilfe der Klasse GraphDataProcessor wieder geladen und im GraphPanel angezeigt

## Import/Export

Ein Graph wird als XML-File abgespeichert. Die Klasse GraphDataProcessor besitzt dafür die Methode constructStringFromGraph, welcher ein Graph übergeben werden kann und als Ergebnis einen String im XML-Format zurückgibt (mittels eines XML-Parsers (DocumentBuilderFactory) werden die einzelnen Attribute des Graphen als XML-Baum angelegt). Dieser String wird dann mittels FileWriter auf dem Dateisystem am gewünschten Ort abgelegt.

Zum Importieren eines Graphen wird der Methode reconstructGraphFromString ein String im XML-Format übergeben. Als Ergebnis wird ein IncidentListGraph zurückgegeben.

## Spezielles

Es wurden folgende bekannten Patterns angewendet:

* Das Observer Pattern für die Benachrichtigung für den Austausch der Daten zwischen Model und View (GraphPanel überwacht (observes) GraphPanelModel => Daten bereit => Anzeige Daten)
* Das Memento Pattern zum Speichern der Zustandsliste des Graphen (Algorithmus Schritt für Schritt in Graph durchlaufen)
* Das MVC Pattern. Zu den wichtigsten GUI Elementen existiert ein Model, der Controller existiert in Form von Model- und View-Events
* Das Decorable pattern zum setzen und darstellen von Formaten von GUI-Komponenten

# Arbeitsaufteilung

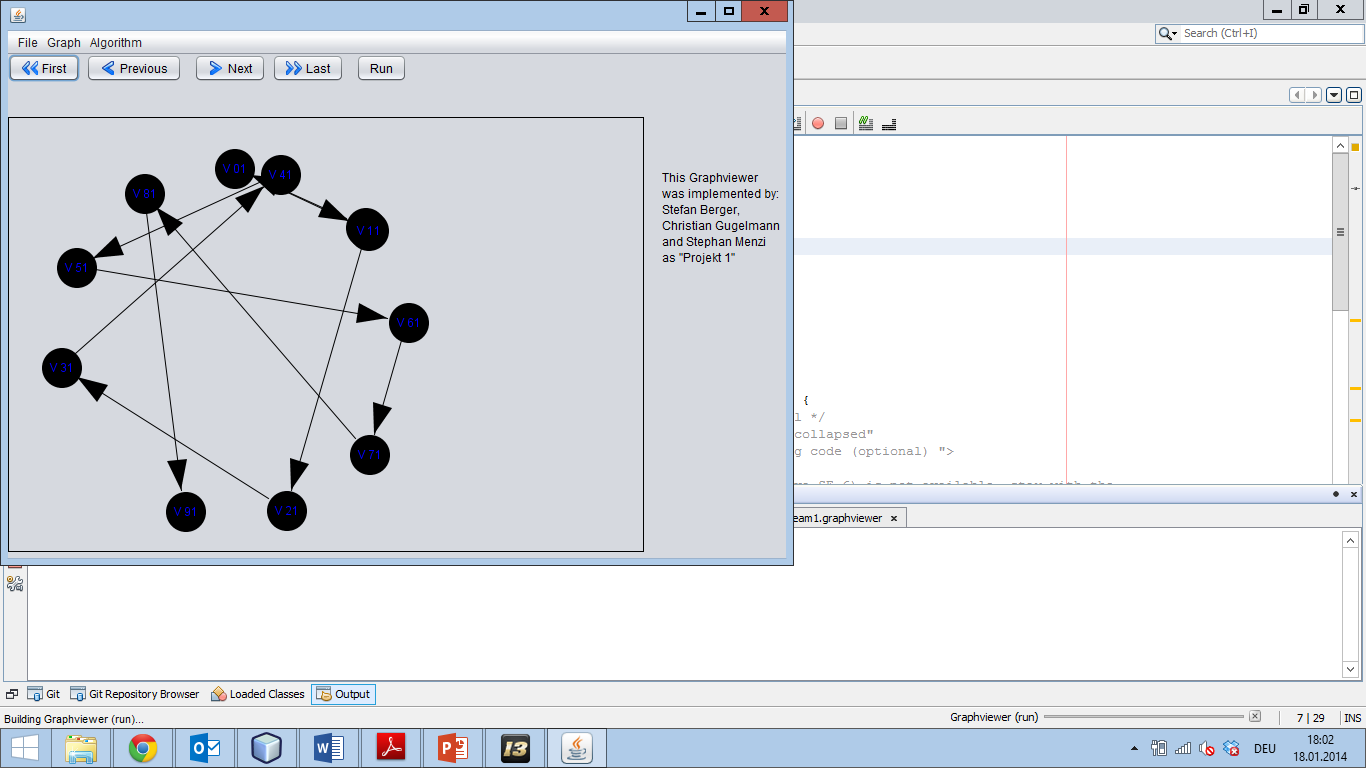
Das Erarbeiten des Grundaufbaus des Tools sowie die Definitionen eines Graphen, Vertex und Edges erfolgten im Team. Danach konnten die Arbeiten in drei Hauptkategorien aufgeteilt werden, welche unter den Teammitgliedern verteilt wurden:

* Stefan Berger: Logik & Darstellung Graph
* Stephan Menzi: Realisierung GUI, Recorder
* Christian Gugelmann: Import/Export Graph

Das Projektteam traf sich wöchentlich zur gemeinsamen Besprechung der getätigten Arbeiten und zum Definieren des weiteren Vorgehens. Da grundsätzlich in der BFH implementiert wurde, konnte man sich beim Implementieren gegenseitig unterstützen und allfällige Probleme zusammen lösen.

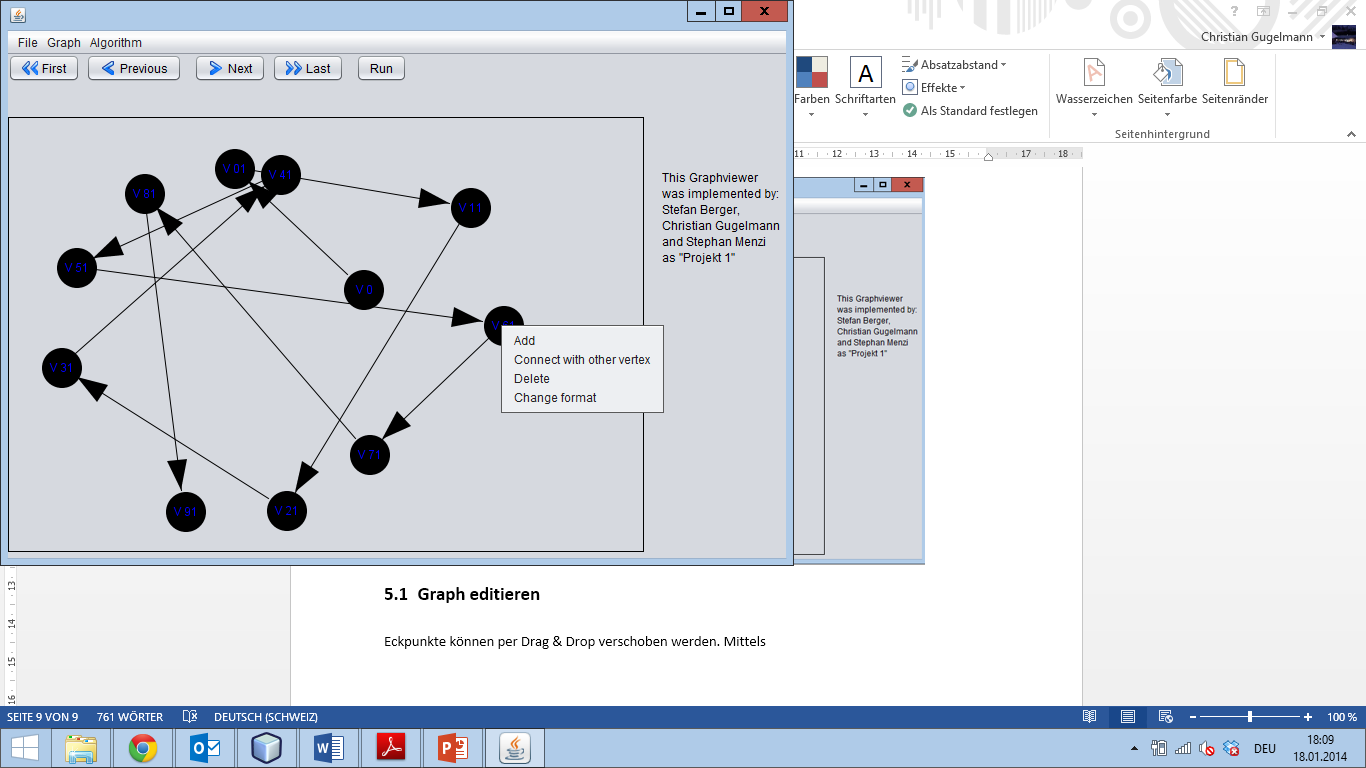
# Benutzerhandbuch

Beim Programmstart wird ein Default-Graph geladen und angezeigt (Dieser kann durch modifizieren des Program.java (Starter-Klasse im default package) angepasst/entfernt werden:



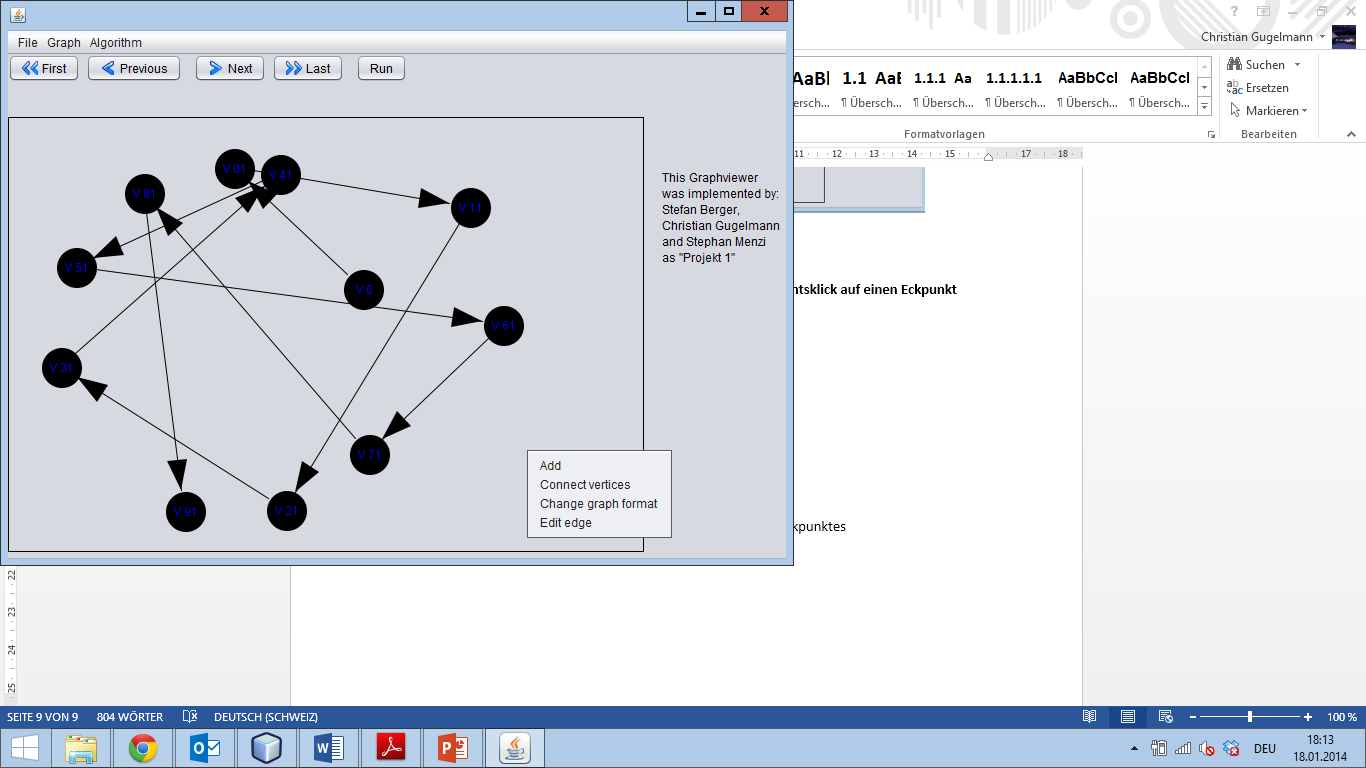
## Editieren eines Graphen

Eckpunkte können per Drag & Drop verschoben werden. Mittels **Rechtsklick auf einen Eckpunkt** erscheint folgender Dialog:

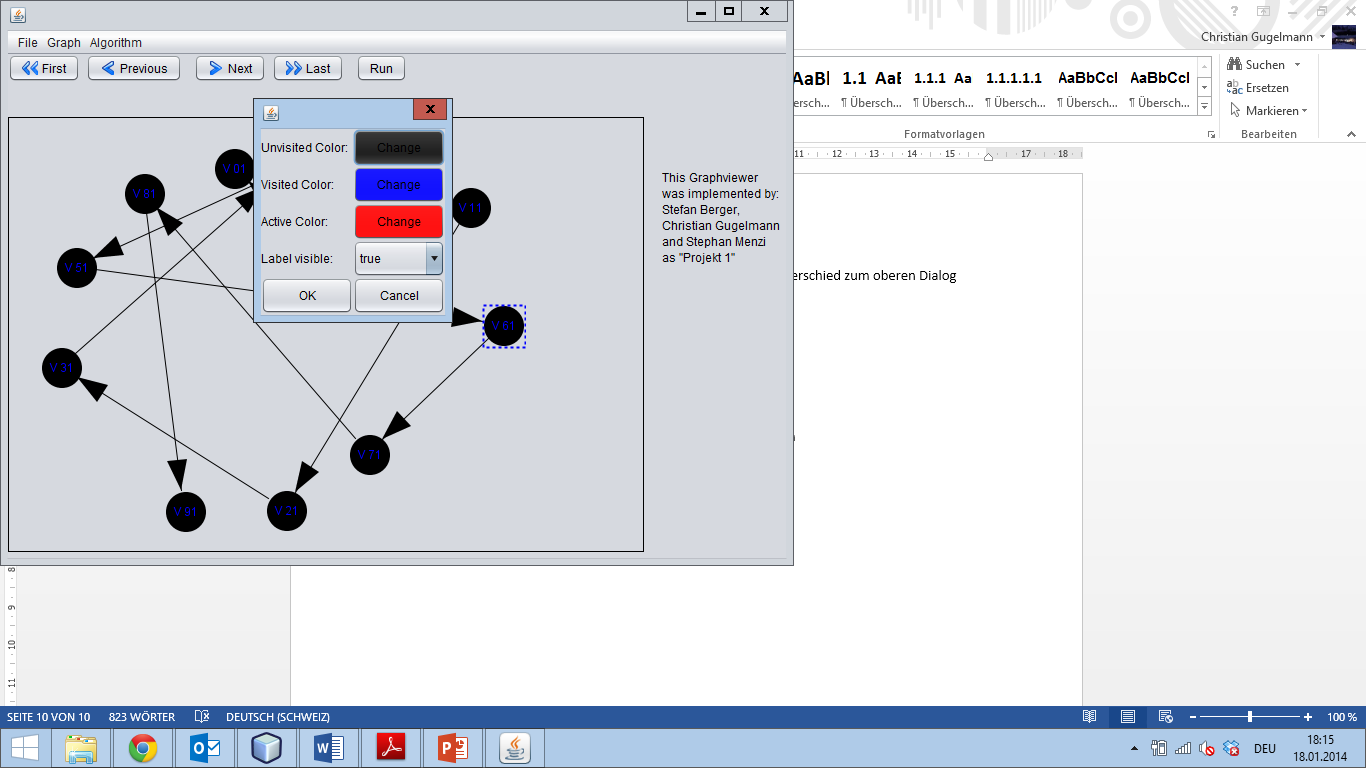


* Add: Neuer Eckpunkt hinzufügen
* Connect with other vertex: Verbinden von Eckpunkten
* Delete: Löschen des gewählten Eckpunktes
* Change format: Anpassen des Formats des gewählten Eckpunktes

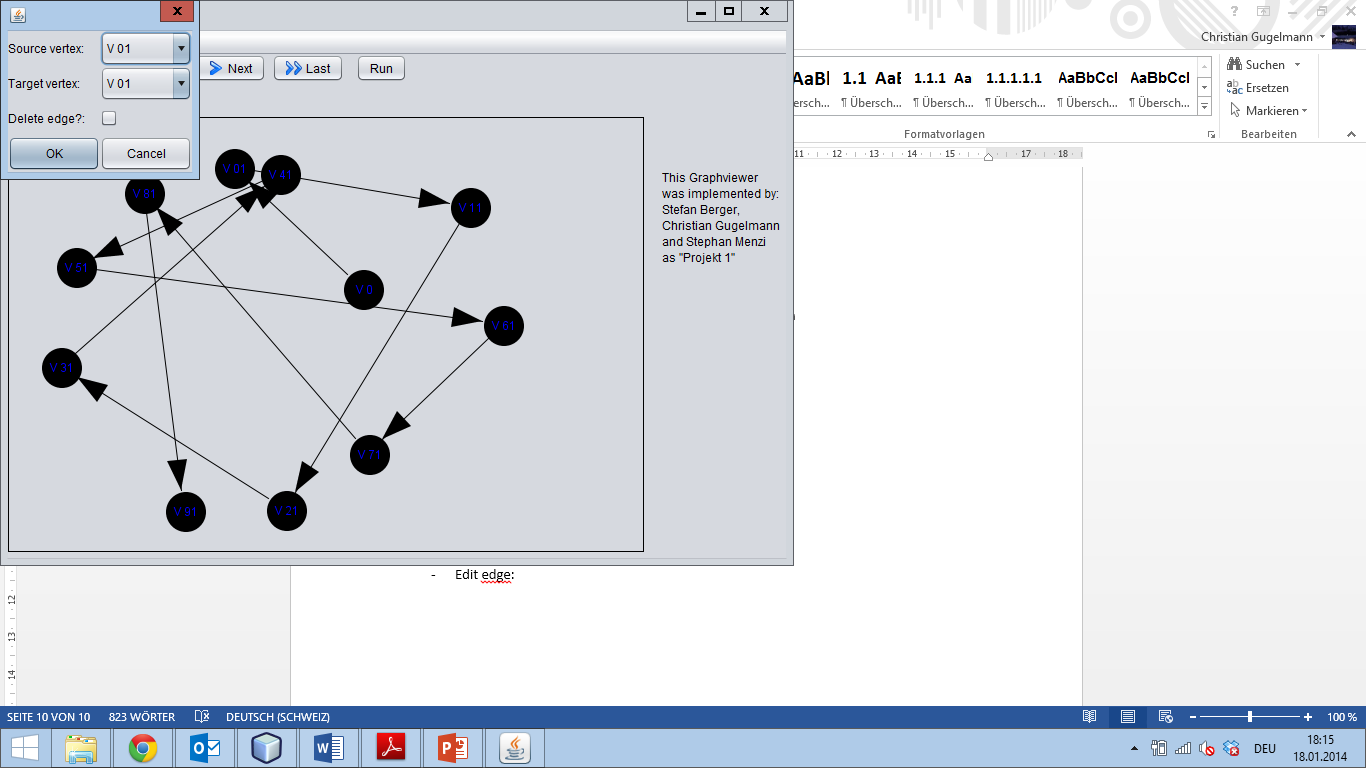
Mittels **Rechtsklick auf eine leere Stelle** im Fenster erscheint im Unterschied zum oberen Dialog folgender:



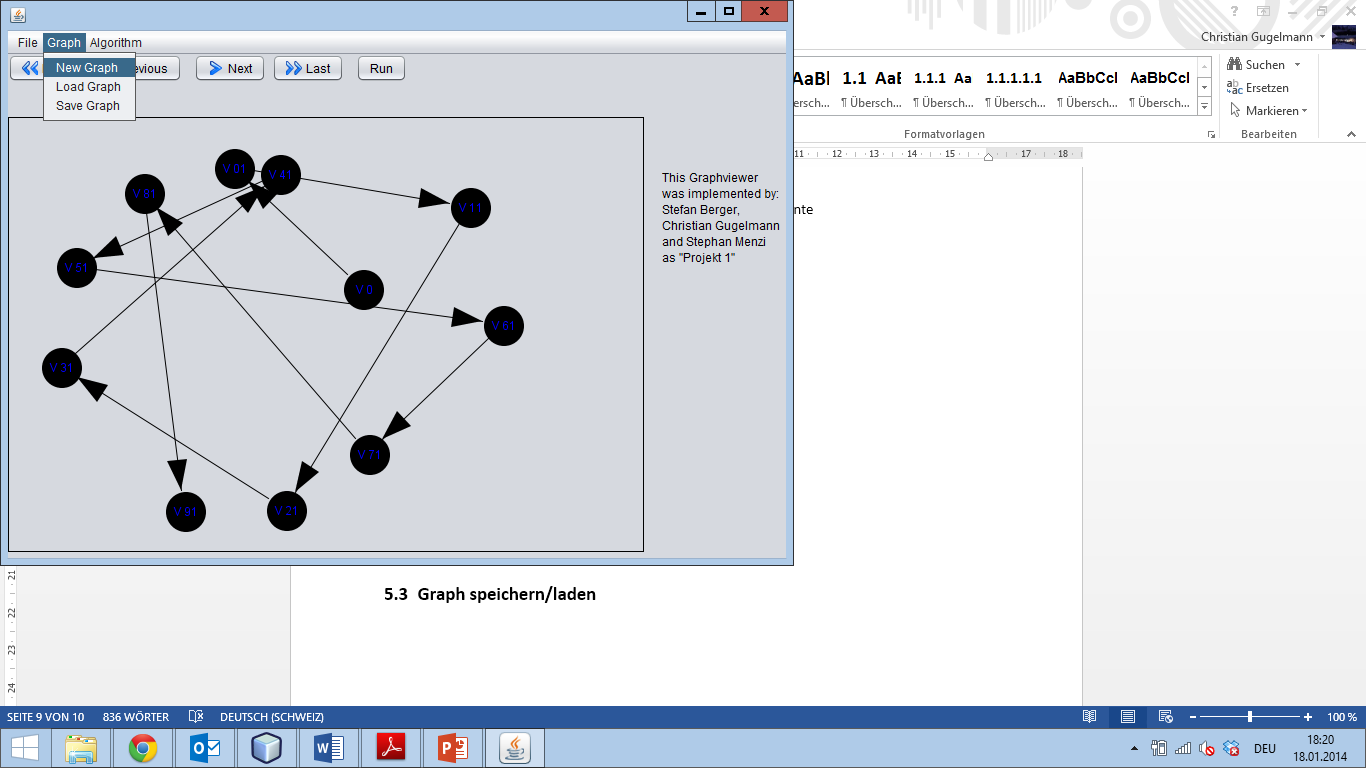
* Change graph format: Ändern des Formates des Graphen



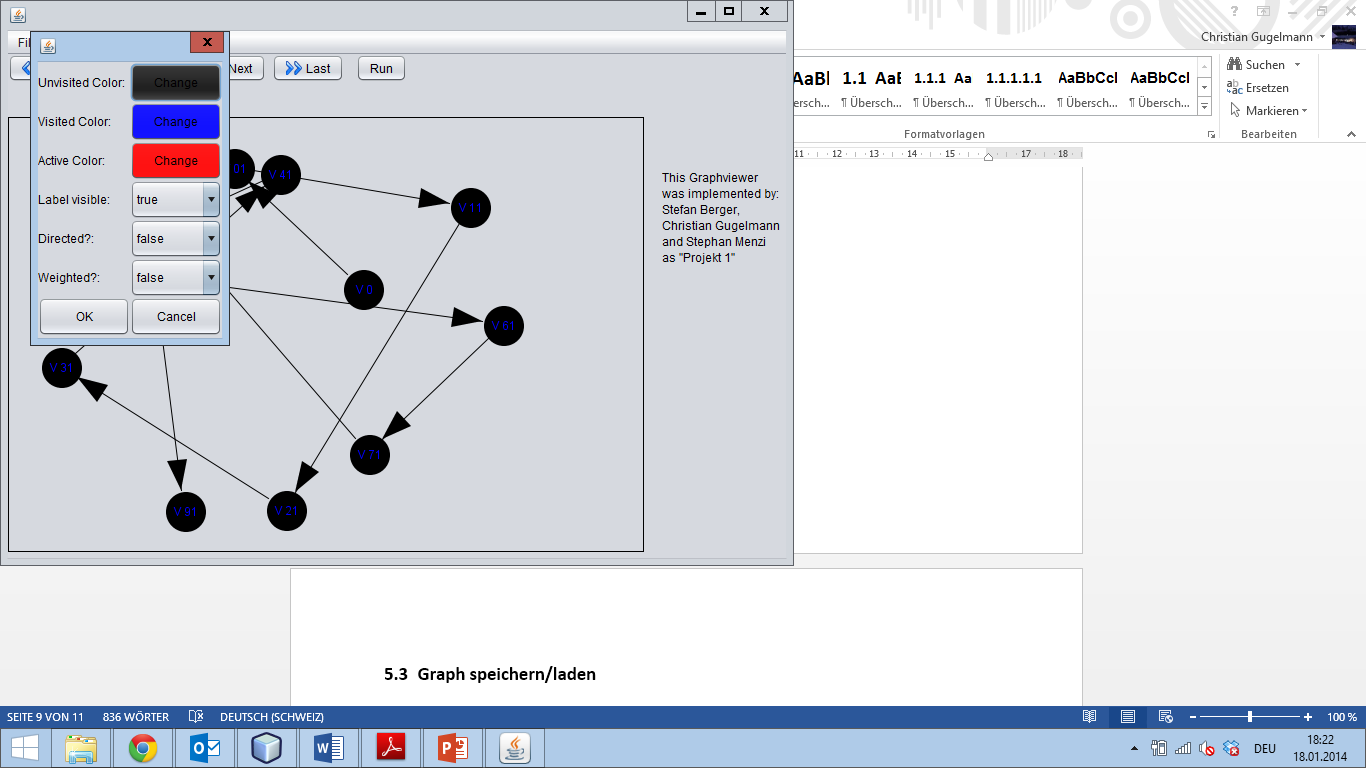
* Edit edge: Anpassen oder Löschen einer gewünschten Kante (Bei gewichteten Graphen kann das Gewicht der Kante ebenfalls angepasst werden)



## Neuen Graphen erstellen



Unter dem Menüpunkt Graph --> New Graph kann ein neuer Graph erstellt werden. Diesem müssen zuerst die Graph-Attribute gesetzt werden, bevor wie unter [**Graph editieren**](#_Graph_editieren)die Eckpunkte und Kanten hinzugefügt und bearbeitet werden können.

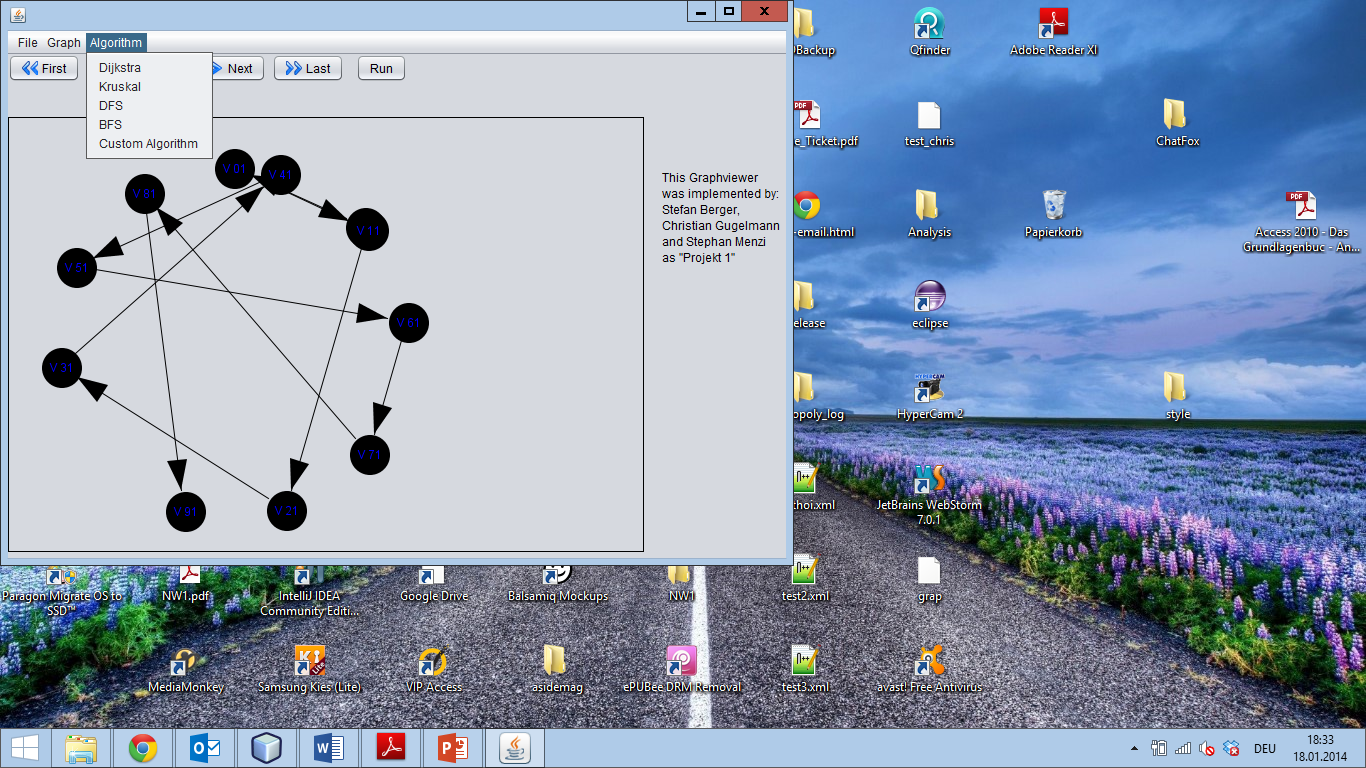


## Graph speichern/laden

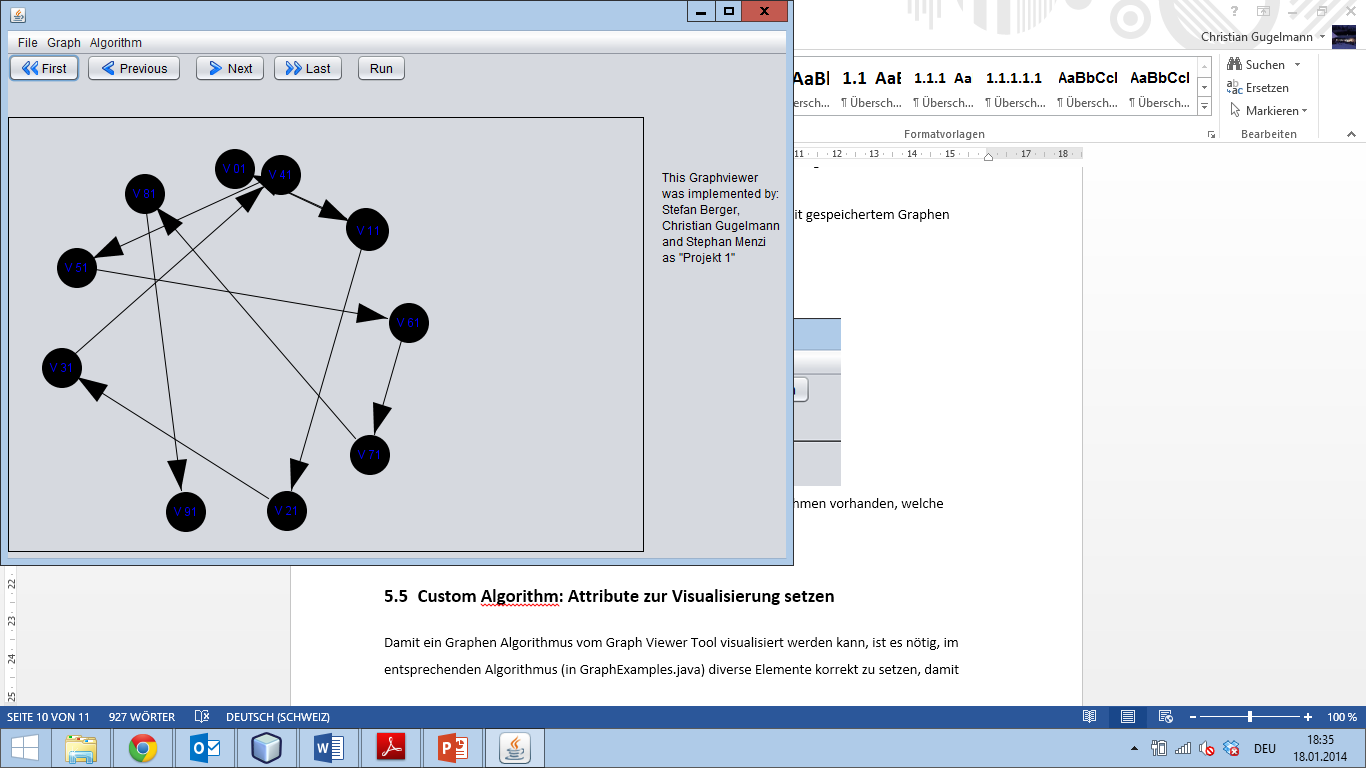
Unter dem Menüpunkt Graph:

* Save Graph: Öffnet Speichern Dialog, mit welchem der Pfad gewählt werden kann, wo der Graph abgespeichert werden soll (Angabe der Endung .xml wird empfohlen!).
* Load Graph: Öffnet einen Lade-Dialog, mittels welchem eine Datei mit gespeichertem Graphen gewählt werden kann.

## Visualisierung von Graphen Algorithmen

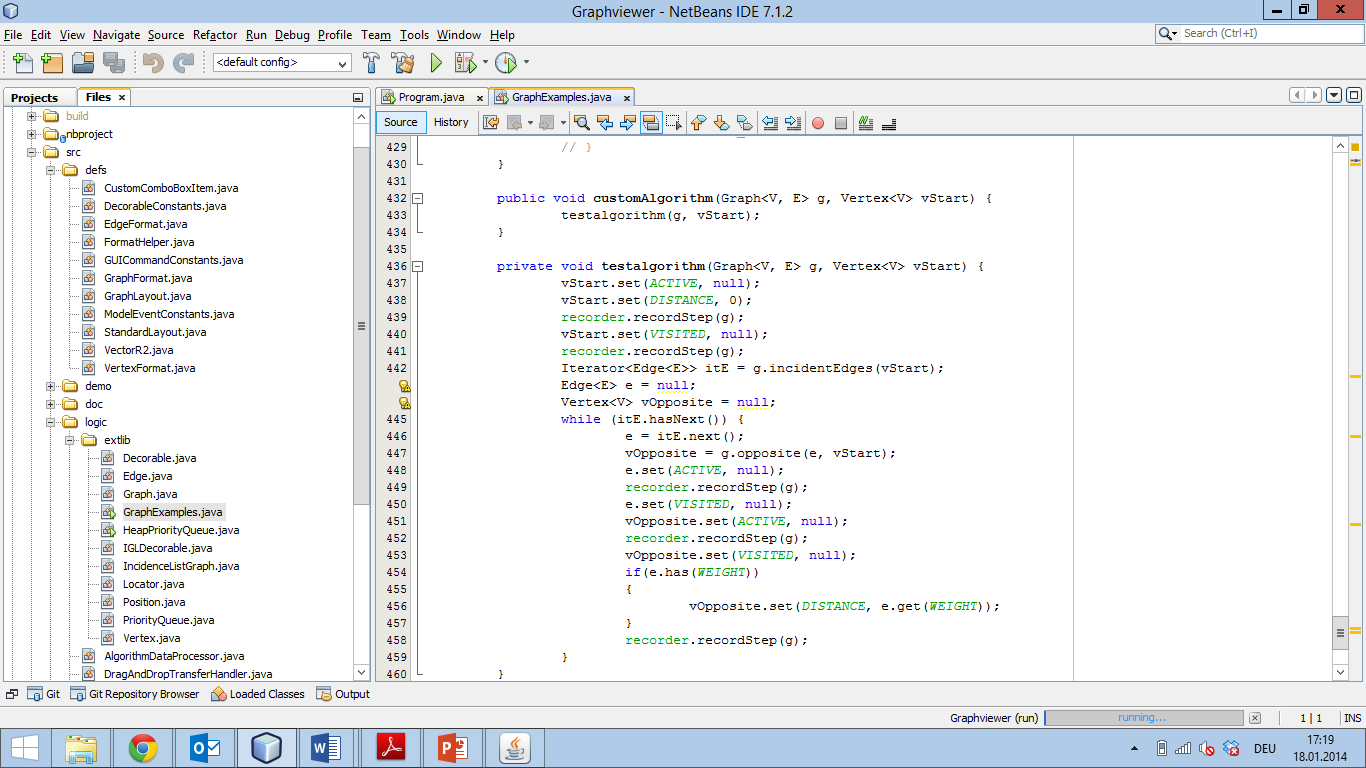


Unter dem Menüpunkt Algorithm sind vordefinierte Graphen Algorithmen vorhanden, welche visualisiert werden können. Nach dem Wählen des Startpunktes des Graphen kann der gewählte Algorithmus mittels den folgenden Buttons durchlaufen werden.



## Attribute zur Visualisierung für bestehende Algorithmen/den Custom Algorithm setzen

Damit ein Algorithmus über einen Graphen im Graph Viewer Tool visualisiert werden kann, muss man im entsprechenden Algorithmus (in GraphExamples.java) die gewünschten Decorable-Objekte korrekt setzen und die gewünschten Schritte jeweils mit dem Recorder aufzeichnen. Anhand eines kleinen Testalgorithmus soll dies illustriert werden:



**1**

**2**

|  |  |
| --- | --- |
| **1** | Zu jedem Vertex oder Edge wird das Attribut gesetzt:  <Vertex/Edge>.set(ATTRIBUT, WERT) |
| **2** | Der Graph im aktuellen Zustand wird im Recorder gespeichert: recorder.recordStep(g) |

**Mögliche**/Für die Visualisierung **reservierte** Attribute sind:

* **ACTIVE** //.set(ACTIVE, null), löscht wenn VISITED gesetzt das Objekt VISITED
* **DISTANCE** //.set(DISTANCE, 55.75) => Wir empfehlen Double-/Integer-Werte zu setzen!
* **VISITED** //.set(VISITED, null), löscht wenn ACTIVE gesetzt das Objekt ACTIVE
* **WEIGHT** //.set(WEIGHT, 75.75) => Wir empfehlen Double-/Integer-Werte zu setzen!

Die gesetzten Attribute zum Zeitpunkt des .recordStep(GRAPH) des Recorders werden dann entsprechend visualisiert.

Selbstverständlich ist es auch möglich mittels .destroy(ATTRIBUT), ein Attribut zu löschen oder mittels .has(ATTRIBUT) zu prüfen, ob ein solches gesetzt wurde.

Sollte man den Graphen nicht visualisieren wollen, spricht auch nichts dagegen diese Attribute sonst zu verwenden.