

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

Projekt 1

Graph Viewer

Projektdokumentation

18.01.2014

Berger Stefan (bergs13)

Menzi Stephan (menzs2)

Gugelmann Christian (gugec1)

Inhalt

[1 Allgemeines 3](#_Toc377833648)

[1.1 Projektbeschreibung 3](#_Toc377833649)

[1.2 Projektorganisation 3](#_Toc377833650)

[2 Anforderungen 4](#_Toc377833651)

[2.1 Erstellen und Bearbeiten von Graphen 4](#_Toc377833652)

[2.2 Import/Export von Graphen 4](#_Toc377833653)

[2.3 Visualisierung von Graphen Algorithmen 4](#_Toc377833654)

[3 Umsetzung 5](#_Toc377833655)

[3.1 Klassendiagramm 5](#_Toc377833656)

[3.2 GUI 5](#_Toc377833657)

[3.3 Graph-, Vertex-, Edge-Formate 6](#_Toc377833658)

[3.3.1 Attribute Graph (GraphFormat.java) 6](#_Toc377833659)

[3.3.2 Attribute Vertex (VertexFormat.java) 6](#_Toc377833660)

[3.3.3 Attribute Edge (EdgeFormat.java) 6](#_Toc377833661)

[3.4 Graph Editor/Visualisierung Graph 6](#_Toc377833662)

[3.5 Import/Export 7](#_Toc377833663)

[3.6 Spezielles 7](#_Toc377833664)

[4 Arbeitsaufteilung 7](#_Toc377833665)

[5 Benutzerhandbuch 8](#_Toc377833666)

[5.1 Graph editieren 8](#_Toc377833667)

[5.2 Neuer Graph erstellen 9](#_Toc377833668)

[5.3 Graph speichern/laden 10](#_Toc377833669)

[5.4 Visualisierung von Graphen Algorithmen 10](#_Toc377833670)

[5.5 Custom Algorithm: Attribute zur Visualisierung setzen 11](#_Toc377833671)

# 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, Stefan Menzi, Christian Gugelmann |
| Projektbetreuung | Peter Schwaab |

# 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.

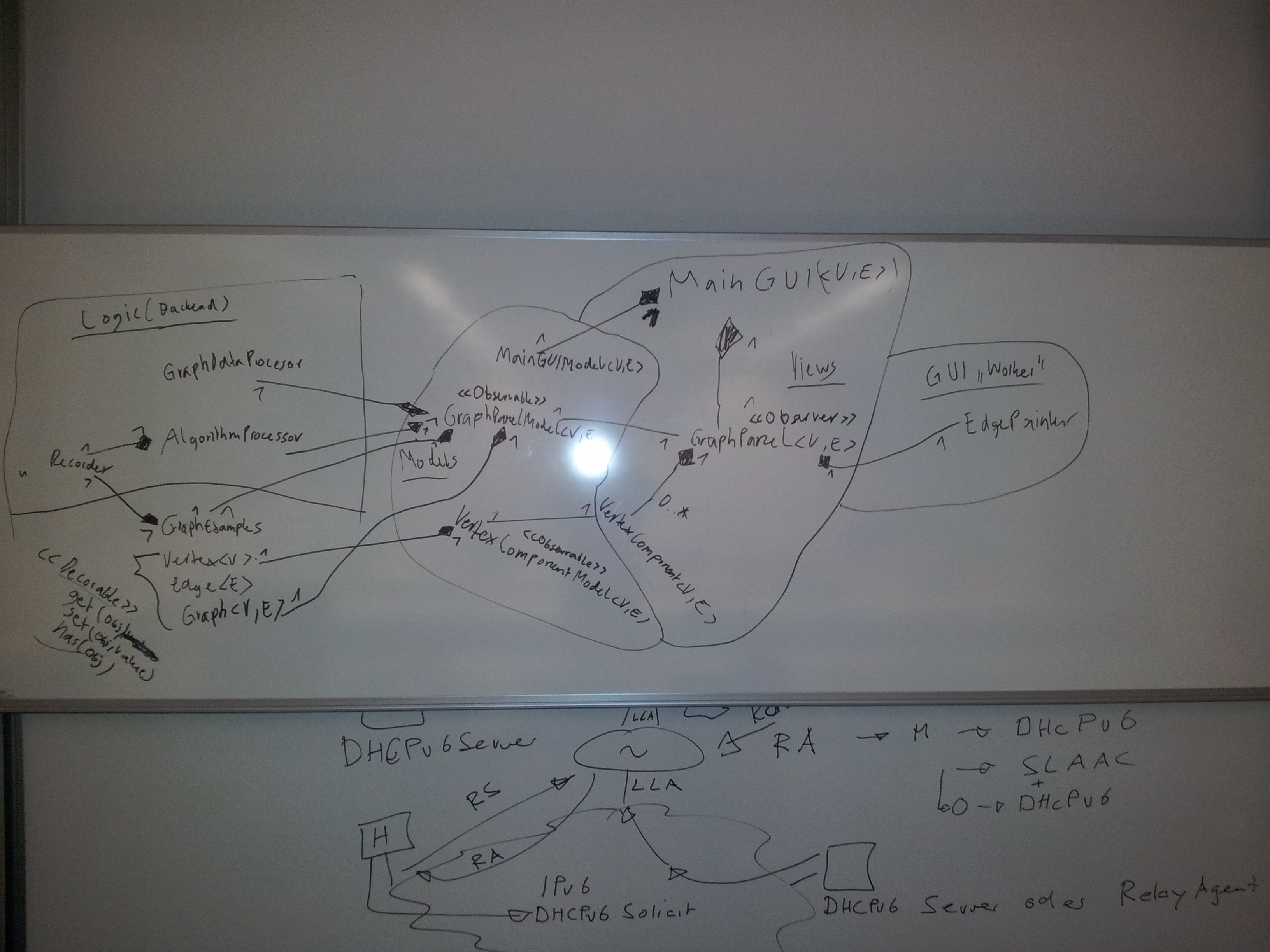
## Visualisierung von Graphen Algorithmen

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:

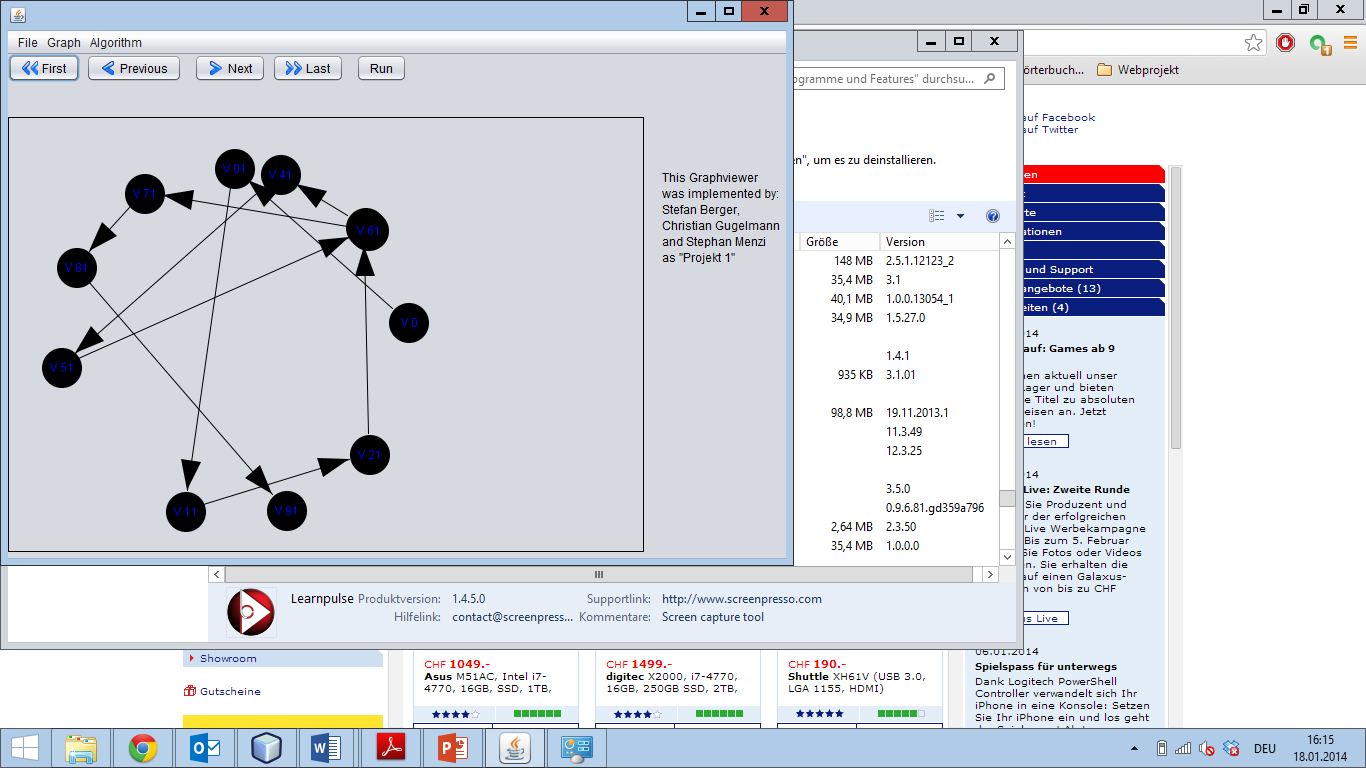
* Aktiver Eckpunkt/aktive Kante
* Besuchter Eckpunkt/besuchte Kante
* ....???!!!

# Umsetzung

## Klassendiagramm



## GUI



Das Haupt-GUI, ausser das zentrale GraphPanel zur anzeige des Graphen, wurde mittels NetBeans-GUI-Editor erstellt.

## Graph-, Vertex-, Edge-Formate

Ein Graph, Vertex und Edge besitzen verschiedene Attribute, welche jeweils in einer separaten Format-Klasse definiert sind.

### Attribute Graph (GraphFormat.java)

* Labels sichtbar/unsichtbar (isLabelVisible)
* directed/undirected (isDirected)
* Gewichtung (isWeighted)
* Aktive Farbe (activeColor)
* Farbe besuchtes Element (visitedColor)
* Farbe unbesuchtes Element (unvisitedColor)

### Attribute Vertex (VertexFormat.java)

* Position Mitte des Punktes (ceterPoint)
* Labe (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

Berger: Grundlegender Aufbau

### 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

Benutzte Patterns:

Observer Pattern

Memento Pattern

MVC Pattern...

# Arbeitsaufteilung

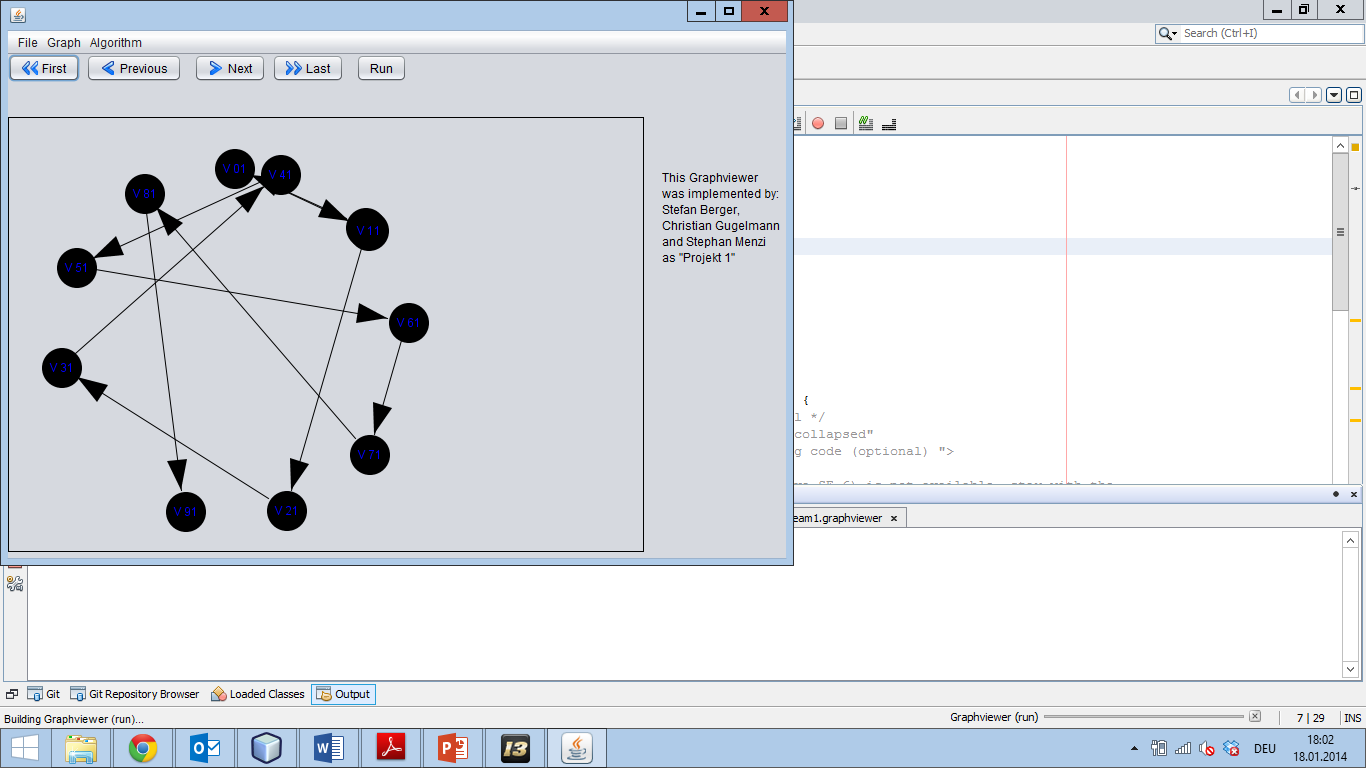
Das Erarbeiten des Grundaufbaus des Tools sowie die Definitionen eines Graphen, Vertex und Edges erfolgten im Team. Danach konnten die Arbeiten in drei Hauptkategorien aufgeteilt werden, welche unter den Teammitgliedern verteilt wurden:

* 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.

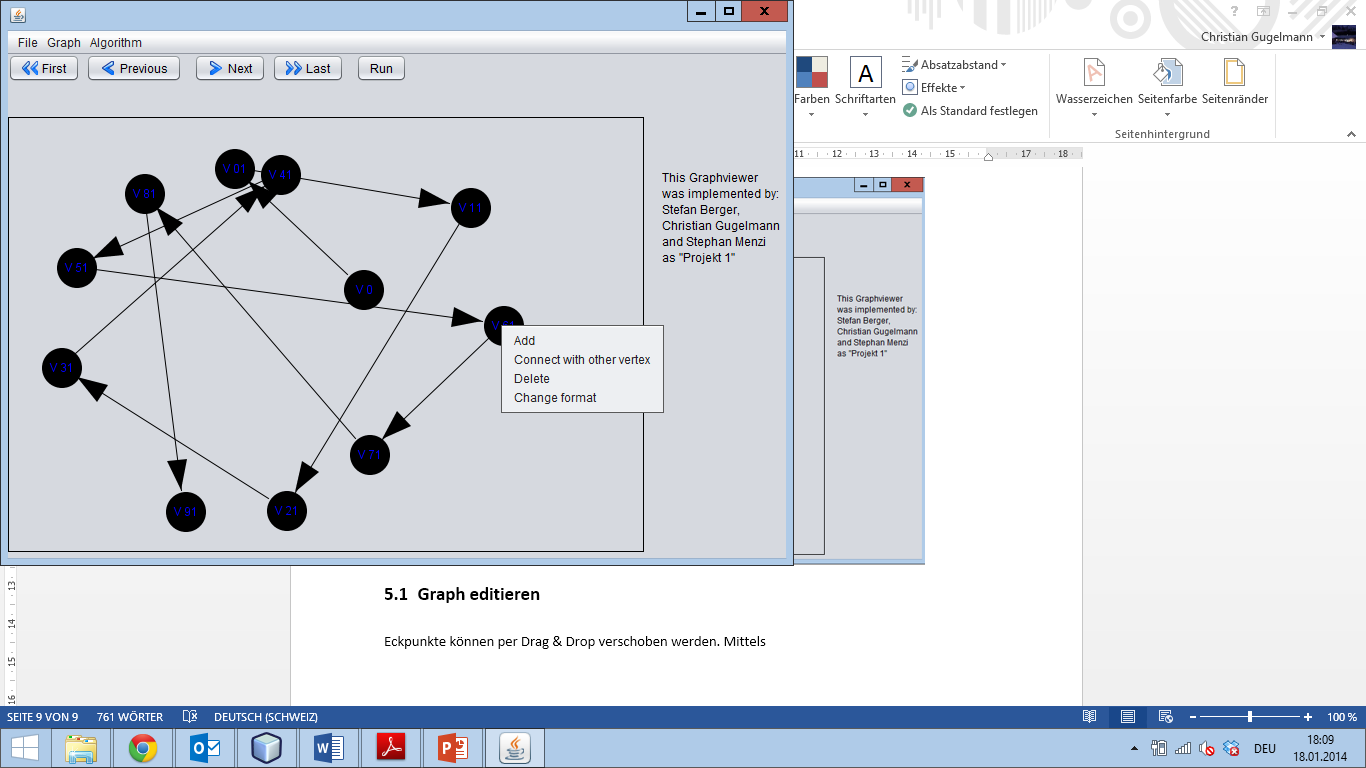
# Benutzerhandbuch

Beim Programmstart wird ein Default-Graph geladen und angezeigt:



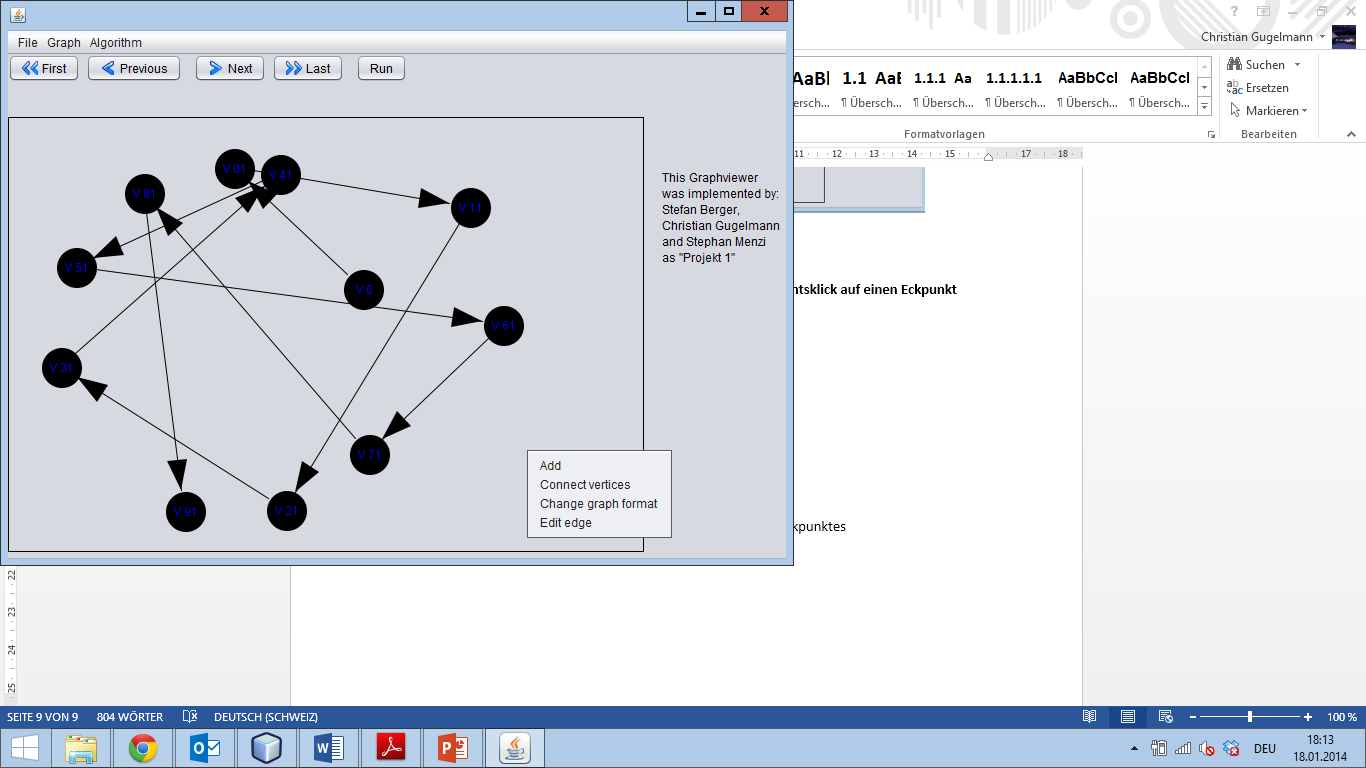
## Graph editieren

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

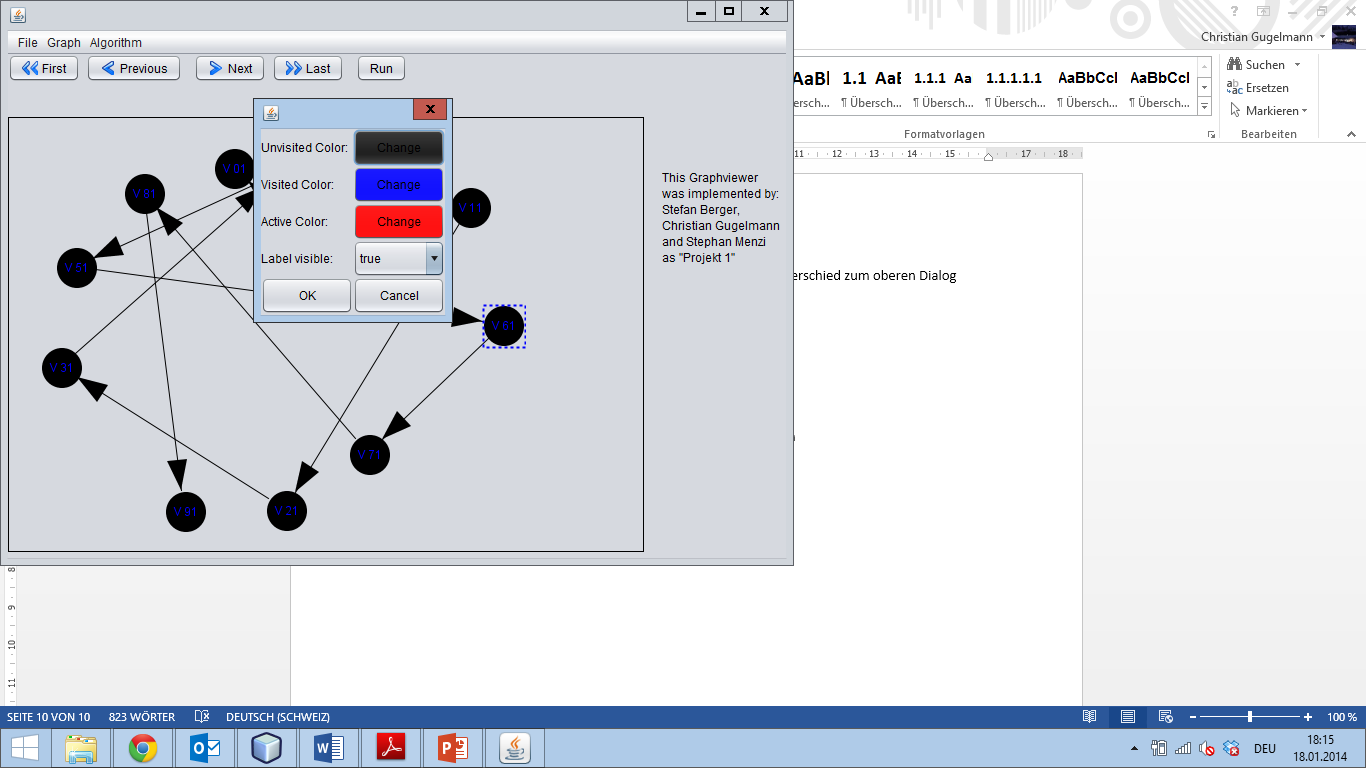


* Add: Neuer Eckpunkt hinzufügen
* Connect with other vertex: Verbinden von Eckpunkten
* Delete: Löschen des gewählten Eckpunktes
* Change format: Anpassen des Formats des gewählten Eckpunktes

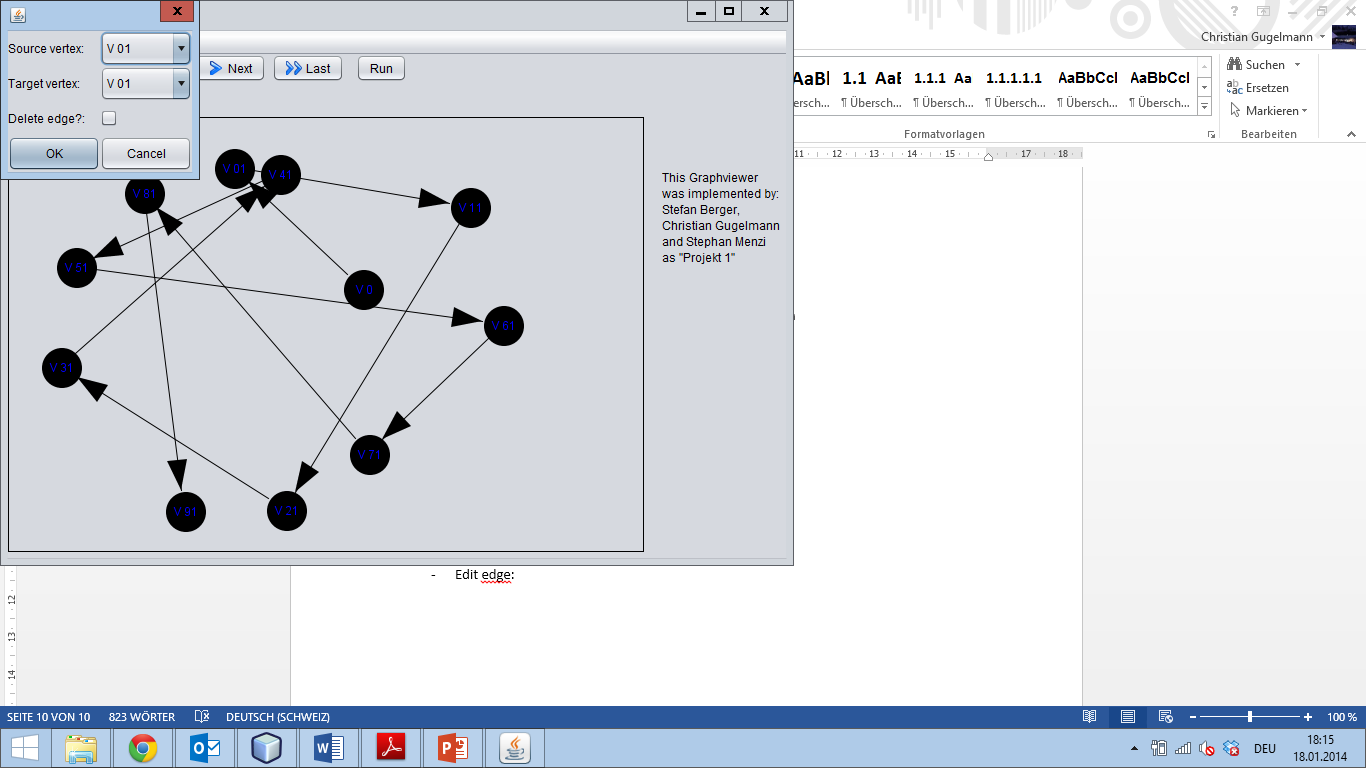
Mittels **Rechtsklick auf eine leere Stelle** im Fenster erscheint im Unterschied zum oberen Dialog folgender:



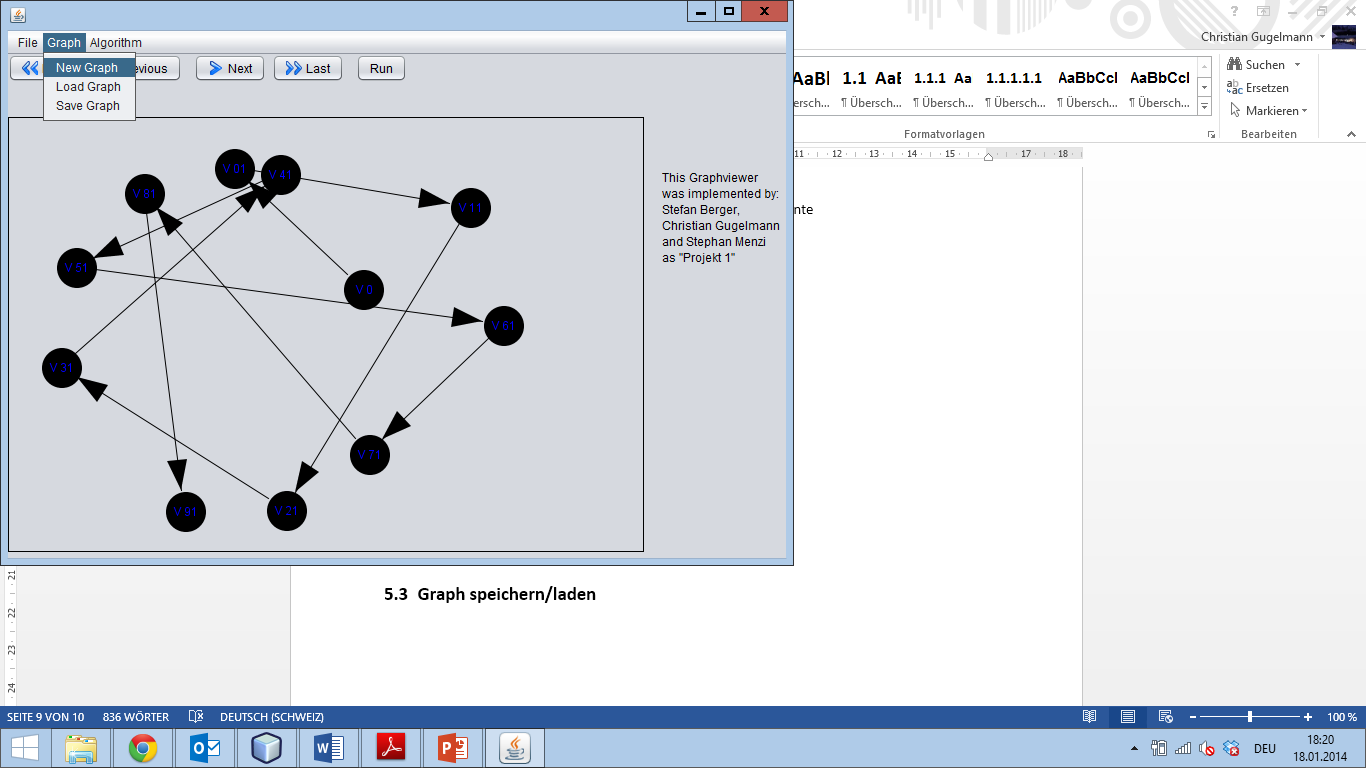
* Change graph format: Ändern des Formates des Graphen



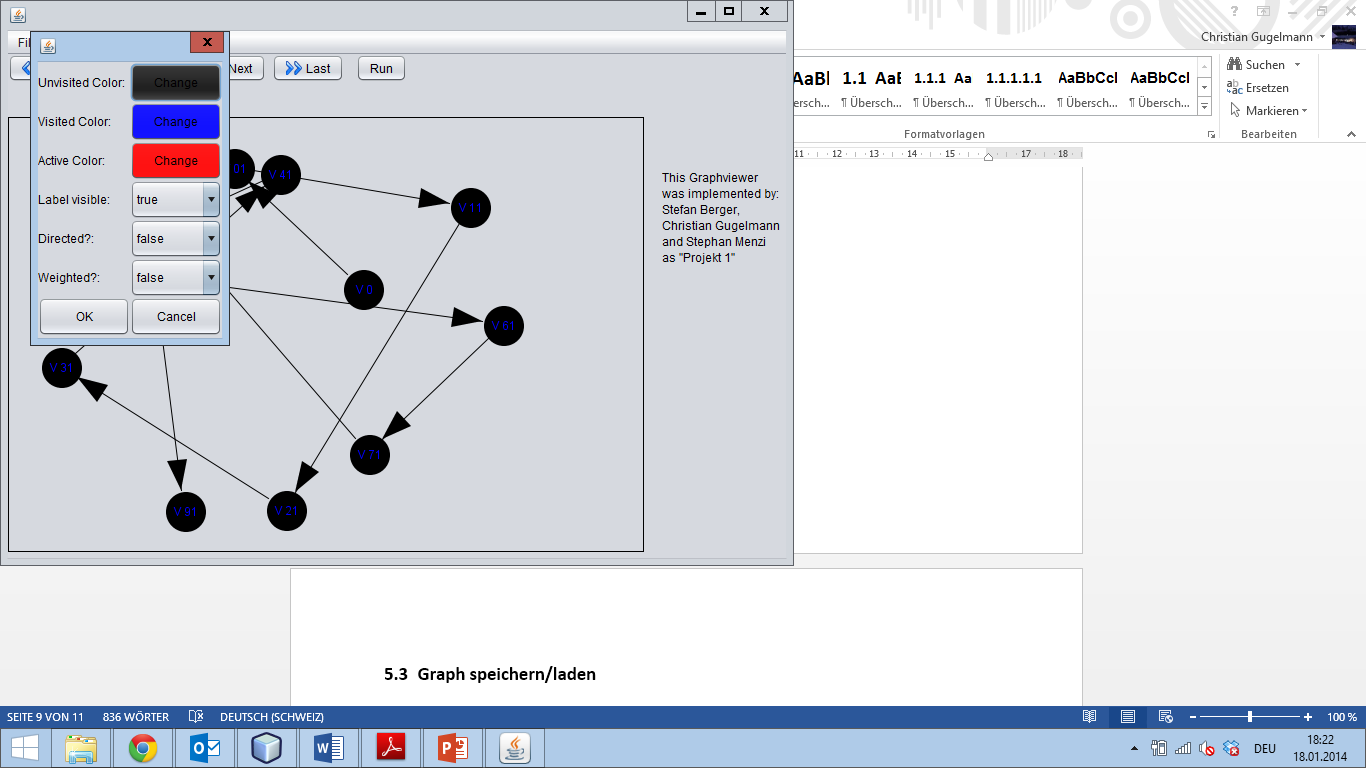
* Edit edge: Anpassen oder Löschen einer gewünschten Kante



## Neuer Graph 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.

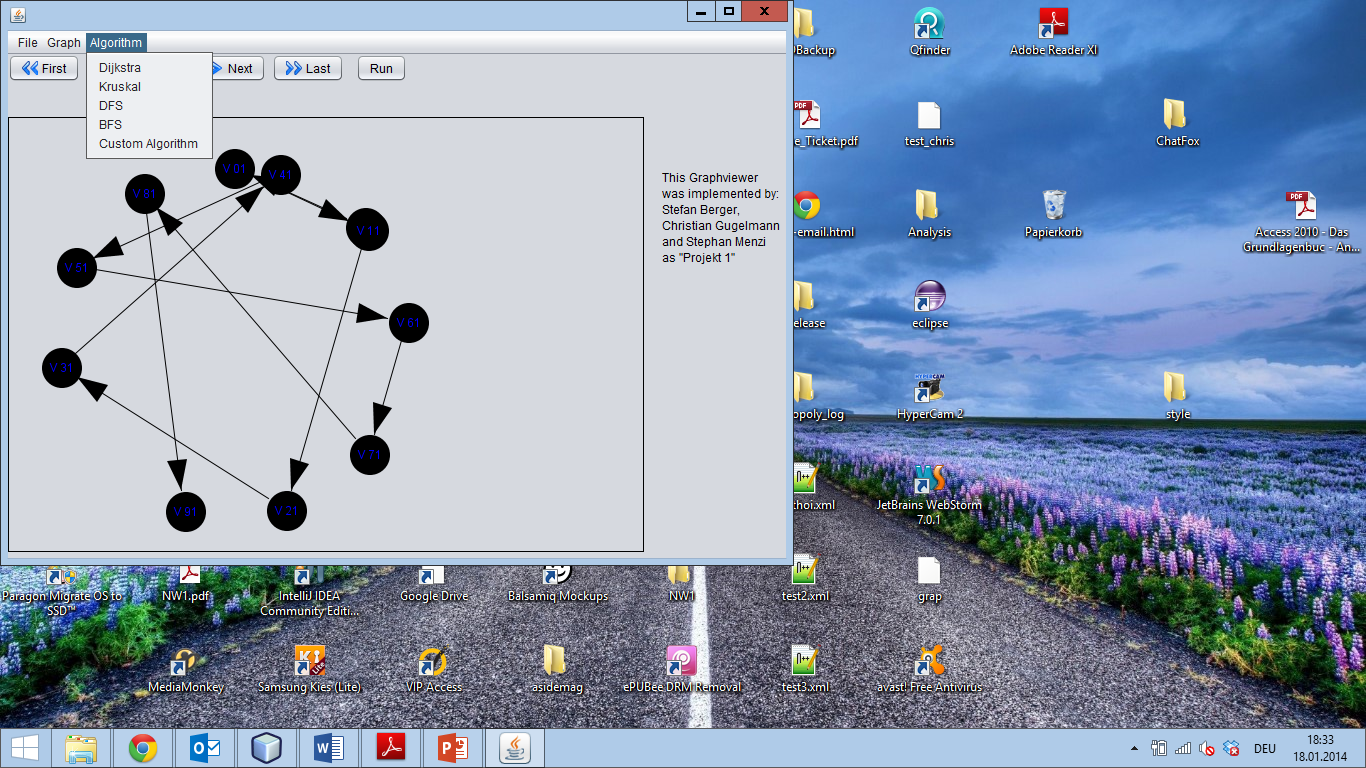


## Graph speichern/laden

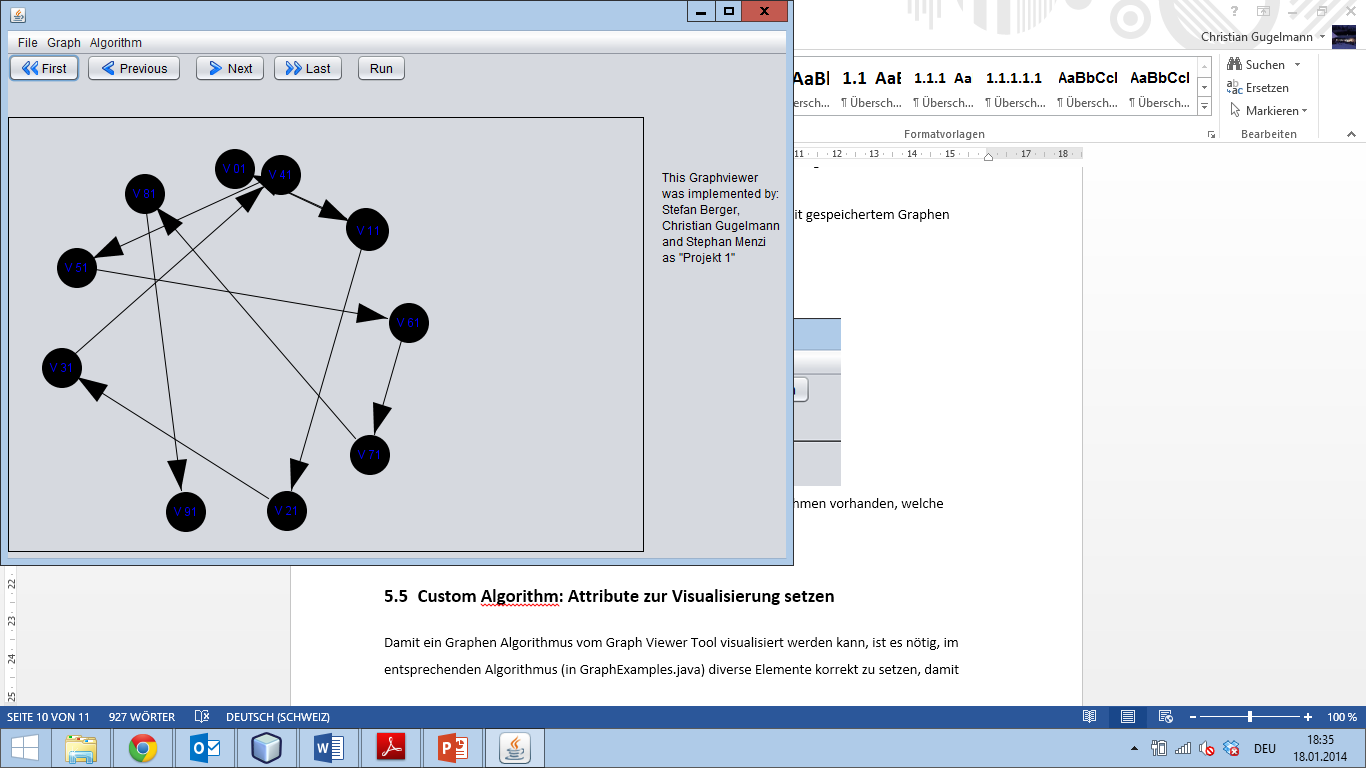
Unter dem Menüpunkt Graph:

* Save Graph: Öffnet Speichern Dialog, mit welchem der Pfad gewählt werden kann, wo der Graph abgespeichert werden soll.
* Load Graph: Öffnet Laden Dialog, mittels welchem File mit gespeichertem Graphen gewählt werden kann.

## Visualisierung von Graphen Algorithmen

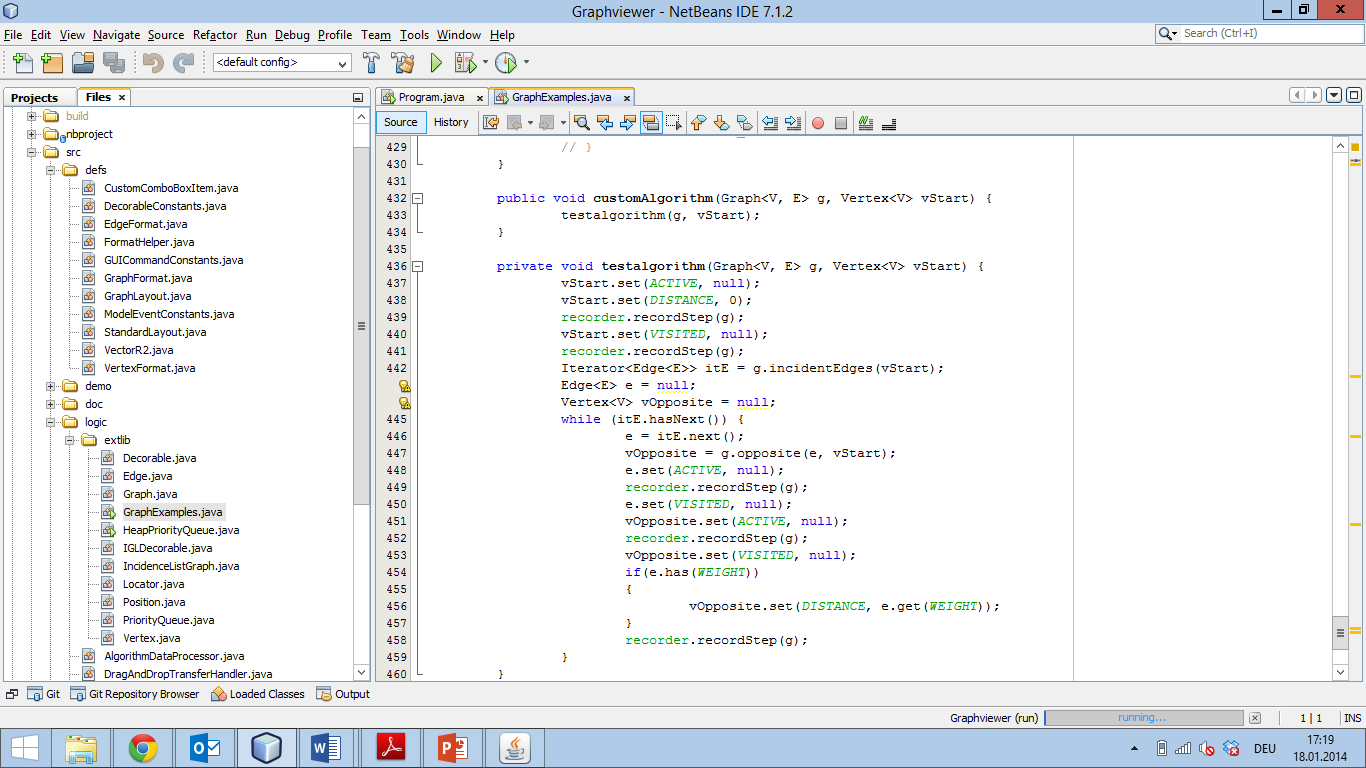


Unter dem Menüpunkt Algorithm sind vordefinierte Graphen Algorithmen vorhanden, welche visualisiert werden können. Nach dem Wählen des Startpunktes des Graphen kann der gewählte Algorithmus mittels den folgenden Buttons durchlaufen werden.



## Custom Algorithm: Attribute zur Visualisierung setzen

Damit ein Graphen Algorithmus vom Graph Viewer Tool visualisiert werden kann, ist es nötig, im entsprechenden Algorithmus (in GraphExamples.java) diverse Elemente korrekt zu setzen, damit diese vom Tool dargestellt werden können. 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 Attribute sind:

* ACTIVE
* DISTANCE
* VISITED
* WEIGHT