Programmbericht des „Graph Visualizers“

Robin Kopitz
Hausarbeit Grafische Nutzerschnittstellen
rkopitz@hs-harz.de

Gliederung

1. Problembeschreibung / Motivation

In der Graphentheorie, wie es der Name schon sagt, werden Graphen benötigt, sowohl in der Lehre, Forschung oder der Entwicklung. Um einen Graphen in der Programmierung verarbeiten zu können, benötigt man ein bestimmtes Format, eines dieser Formate ist die Adjazenz Matrix. Ein Graph kann sehr viele ausgehende als auch eigehende Verbindungen haben, was das Erstellen einer solchen Adjazenz Matrix verkomplizieren kann. Dabei kann der erstellte „Graph Visualizer“ Abhilfe schaffen, dieser ermöglicht es dem Nutzer einen Graphen mit wenigen Mausklicks zu erstellen und über die Export Erweiterung in die gewünschte Adjazenz Form zu bringen. Für einen geübten Graphen Theoretiker mag das nicht zwangsläufig nötig sein, jedoch in der Lehre könnte dieses Programm Studenten helfen Graphen zu erstellen und die gewünschte Adjazenz Form ohne Probleme für weitere Berechnungen aufzustellen. Das unterstützt den Studenten bei der Übung und hilft ihm ein Verständnis für die Entwicklung, als auch die Verarbeitung erstellter Graphen zu erlangen. Für die Forschung oder Entwicklung müsste der „Graph Visualizer“ noch weiterentwickelt werden, um den benötigten Anforderungen gerecht zu werden, dazu jedoch später mehr in den Entwicklungsperspektiven.

2. Programmstruktur / Programmentwicklung

2.1. Entwicklungsprozess

Beim Entwicklungsprozess wurden die groben Kriterien des „Graphen Visualizer“ festgelegt. Folgende Themen wurden im Entwicklungsprozess durchdacht:

- Datenspeicherung
- Einstellungen / „Settings“
- ID-System
- Import / Export Erweiterung
- Oberflächendesign
- Visuelle Umsetzung eines Graphen in JavaFX
- Quellcode Design Aspekte

Nach der Festlegung der Rahmenkriterien der genannten Punkte konnte die Entwicklung beginnen. In den folgenden Unterpunkten werden diese Kriterien einmal ausführlich erläutert.

2.2. Datenspeicherung

Für die Datenspeicherung der einzelnen Elemente werden in diesem Fall Klassen genutzt, die jetzt genauer erläutert werden. Auf die Datenspeicherung der Import-/Export TXT Datei wird hier jetzt nicht eingegangen, da dies unter der entsprechenden Überschrift erläutert wird. Ein Graph besitzt grundsätzlich Knotenpunkte und einen Verbindungstyp dieser kann „ungerichtet“ oder „gerichtet“ sein. Ein Knotenpunkt kann mit einem Namen versehen werden und mehrere Vorgänger und Nachfolger haben. Ein Verbindungstyp geht von einem Knotenpunkt zu einem anderen oder über eine Schleife zu dem selbigen. Daraus resultiert, dass ein Verbindungstyp

grundsätzlich einen Start- und einen Endpunkt besitzt. Außerdem kann ein Verbindungstyp zusätzlich noch mit Kosten belegt werden. Diese Kosten müssen abgespeichert werden, um die Möglichkeit zu eröffnen, später Berechnungen mit diesen durchzuführen. Das sind die groben Informationen zu einem Graphen, es müssen jedoch noch für die Visualisierung einige Informationen gespeichert werden. Doch diese sind für die einzelnen Klassen unterschiedlich, daher werden sie jetzt separat in den einzelnen Unterpunkten erläutert.

2.2.1. Knotenpunkt eines Graphen

Wie bereits schon erwähnt kann ein Graph mehrere Nachfolger und Vorgänger aufweisen. Diese werden im „Graph Visualizer“ in zwei Array Listen abgespeichert. Diese Array Listen sind vom Typen „GraphNode“, welcher hier die jeweiligen Knotenpunkte darstellt. Die Array Listen haben den Namen „prev“ für previous (Vorgänger) und next (Nachfolger). Beide Array Listen werden dann dementsprechend über den Verbindungstypen aktualisiert. Ein Knotenpunkt weist außerdem einen Namen auf, dieser Name nimmt ungesetzt die Knoten ID an. Der Name wird in einem String in der Klasse „GraphNode“ abgespeichert. Wie eben erwähnt besitzt jeder erstellte Knoten eine Knoten ID, diese wird beim Erstellen des Knotenpunkts gesetzt, zum ID-System später mehr unter der entsprechenden Überschrift. Für den Export besitzt die Klasse noch eine Array List vom Typen „Connection“, um alle ausgehenden Verbindungen des einzelnen Knotens abzuspeichern. Für die Visualisierung müssen zusätzlich noch einige Informationen gesetzt werden. Die Darstellung eines Knotenpunkts wird über einen Kreis realisiert. Um diesen zu erstellen muss die Klasse zusätzlich noch die beiden Koordinaten X und Y für den Mittelpunkt speichern, als auch einen Radius. Im Anschluss muss der Kreis natürlich auch noch dem entsprechenden Layout hinzugefügt werden. Das heißt im Rahmen der Visualisierung fallen in Kurzform folgende Daten an:

- Kreis vom Typen Circle, um den Zugriff auf diesen nicht zu verlieren
- Koordinaten des Mittelpunkts X/Y
- Radius
- Layout

Für die Darstellung des Namens benötigt die Klasse außerdem noch ein Unterlayout und ein Label, um dem Knotennamen im Hauptlayout anzuzeigen. Innerhalb der Klasse „GraphNode“ werden Methoden zur Verfügung gestellt die für das Zusammenspiel der einzelnen Klassen notwendig sind. Dazu später mehr unter dem Punkt Klassendiagramme und Prozesse.

2.2.2. Verbindung zwischen Knotenpunkten

Es wurde bereits festgelegt, dass eine Verbindung „gerichtet“ oder „ungerichtet“ sein kann, jedoch grundsätzlich einen Start- / Endpunkt als auch Kosten aufweist. In Anbetracht dieser Vorrausetzungen und der eigentlich unerwünschten Differenzierung der einzelnen Verbindungstypen ist ein Oberklassensystem angebracht. Es handelt sich in beiden Fällen um eine Verbindung, deswegen wurde die Oberklasse „Connection“ geschaffen, welche abstrakt ist. Das ermöglicht es später die Differenzierungen zwischen den einzelnen Verbindungstypen geringer zu halten. Die Oberklasse „Connection“ weist die Variablen Start- / Endpunkt auf und enthält zusätzlich noch die Kosten, deren Label und ein Layout für die Visualisierung. Außerdem wurden abstrakte Methoden deklariert, welche in beiden Unterklassen definiert werden müssen. Diese beiden Klassen speichern nur Informationen zum Shape-Objekt und

dem Hauptlayout, sie tragen den Namen „BoundConnectionNodes“ für den „gerichteten“ Verbindungstypen und „UnboundConnectionNodes“ für den „ungerichteten“ Verbindungstypen. Die getrennte Implementierung der Verbindungen wurde angestrebt, da es sich hier um zwei verschiedene Verbindungstypen handelt und eine getrennte Implementierung sinnvoller erscheint, um Verwirrung nachfolgender Entwickler zu vermeiden. Außerdem erleichtert es die Verwaltung der Visualisierung von Verbindungen. Die „gerichtete Verbindung“ wird mit einem Pfeil realisiert, da sie nur in eine Richtung nutzbar dargestellt werden soll. Eine „ungerichtete Verbindung“ wird mit einer normalen Linie realisiert, da diese in beide Richtungen nutzbar dargestellt werden soll. Eine Schleife wird in beiden Klassen mit einem Halbkreis realisiert, eine Richtung erscheint hier nicht sinnvoll. Zur Implementierung des Pfeils und der Schleife später mehr unter dem Punkt Umsetzung eines Graphen in JavaFX. Das Zusammenspiel der beiden Klassen mit anderen wird wie auch bei der Klasse „GraphNode“ später im Punkt Klassendiagramme erläutert.

2.3. Einstellungen / Settings

Im „Graph Visualizer“ werden Einstellungen verwendet die den Ablauf des Programms verändern. Es wird ein Verbindungstyp gewählt, um einen gemischten Graphen mit beiden Verbindungen zu vermeiden. Außerdem wird in den Einstellungen die gewünschte Export Art gewählt. Man kann wählen zwischen Standard Export oder Additional Export die Einstellung entscheidet darüber ob später ein Import der exportierten Datei durchgeführt werden kann oder nicht. Wie der Export und Import funktioniert wird im Punkt Import / Export Erweiterung erläutert. Ohne gewählte Einstellungen kommt der Nutzer nicht zu dem Graph Pannel, sondern es wird eine Fehlermeldung ausgegeben. Die Einstellungen des Systems werden in einer Separaten Klasse abgelegt um die Einstellung während des gesamten Vorgangs beizubehalten.

2.4. ID-System

Es wurde ein ID-System, welches mit einer instanziierten Klassen ID arbeitet implementiert, da eine „Singelton Implementation“ nicht nötig ist. Die IDs werden nämlich nur im Graph Pannel verwendet. Es wird also im „NewGraphPannel“ eine Instanz der Klasse „GenereateSceneIds“ erstellt, welche es dem Pannel ermöglicht einzelne IDs für Knotenpunkte zu erstellen. Das hat den Vorteil, dass nach dem Wechsel zur Startseite die IDs wieder von Null an gezählt werden und somit über die ID exportiert werden kann. Bei einer „Singelton Implementation“ wäre das nicht der Fall. Warum das im Export ein Problem darstellt wird im Punkt Import / Export Erweiterung erläutert.

2.5. Import / Export Erweiterung

Die Import- und Export Erweiterung arbeitet eng mit den definierten Einstellungen des „Graph Visualizer“ zusammen. Wie bereits erwähnt kann die Export Art in den Einstellungen gesetzt werden. Hier wird unterschieden zwischen Additional Export, welcher für den Import im späteren Verlauf geeignet ist und dem Standard Export, welcher lediglich die Adjazenz Matrix beinhaltet. Warum wird das unterschieden, das liegt daran, das für den visuellen Import zusätzliche Informationen verarbeitet werden müssen, nämlich die Koordinaten und Kosten / Namen der einzelnen Verbindungen / Knoten. Das kann die Datei mit vielen Informationen

füllen, die der Nutzer eventuell nicht benötigt, daher bietet der „Graph Visualizer“ die Option auf Standard Export, falls der Nutzer keine Weiterverarbeitung der Daten wünscht. Natürlich beinhaltet der Additional Export auch die Adjazenz Matrix, diese verschiebt sich dadurch lediglich in der Datei nach unten.

Der Export nutzt einen String, um die in den Klassen abgelegten Daten in einer TXT Datei zu speichern. Dafür gibt es in der Klasse Export drei Methoden. Die Methode „writeToFile“ speichert den befüllten String in eine TXT Datei, an den vom „FileChooser SaveDialog“ ausgewählten Ordnersystem. Über die Einstellungsklasse „GraphPannelSettings“ wird nun zwischen den Export Arten unterschieden. Die Methode „addAddtionalNodeInformations“ wird nur dann ausgeführt, wenn auch die dafür vorgesehen Einstellung gesetzt wurde. Diese Methode nutzt alle abgelegten Daten aus der „GraphNode“ Klasse um die Informationen dem String anzuhängen. Im Anschluss wird in beiden Fällen die Methode „prepareAdjMatrix ausgeführt, welche sich darum kümmert über die „prev“ Knotenliste die Adjazenz Matrix zu generieren. Ist der String befüllt wird nur noch die bereits erläuterte Methode „writeToFile“ aufgerufen um die im String abgelegten Informationen abzuspeichern.

Im Import wird die Datei genau gegensätzlich aufgelöst. Über den „FileChooser openDialog“ wird die Datei ausgewählt und der Dateipfad wird an die Klasse Import übergeben. Innerhalb der Klasse wird dann die ausgewählte Datei in einen String Zeile für Zeile eingelesen und dann über die Methode „String.split“ und den im Export gesetzten Indizes wie zum Beispiel „Knotenname:“ und „Koordinate X:“ ausgelesen. Die Daten werden dann in lokalen Variablen gespeichert bis alle Notwendigen Informationen für einen Knoten eingelesen sind, danach wird der Knoten erstellt. Im Abbildung 14 wird die Struktur einer exportierten Datei gezeigt. Zum Schluss werden dann aus der Export Datei die ausgehenden Verbindungen eingelesen und die einzelnen Verbindungen erstellt. Im Anschluss hat der Nutzer den ehemalig exportierten Graphen wieder in seiner Ausgangslage zurück. Achtung die Datei darf aufgrund der genutzten Indizes dahingehend nicht verändert werden, da ein Import sonst nicht mehr möglich ist.

2.6. Oberflächendesign

Das Oberflächendesign ist schlicht gehalten um den Nutzer nicht von der Funktionalität abzulenken. Es wurden schlichte Grautöne genutzt, um grelle Farben oder ungewollte Signaltöne zu vermeiden.

2.6.1. Startseite

Für die Startseite wurde als Hauptlayout ein „BorderPane“ gewählt, da dieses „Pane“ eine Einteilung der Top-, Bot- und Center-Elemente erlaubt. Das Design der Oberfläche findet in der „Style.css“ statt. Zusätzlich wurden drei Google Schriftarten als CSS-Datei eingebunden. Eine Schriftart ist der Menüleiste zugeordnet, eine dem Standard Text und eine den Überschriften. Als Top-Element wurde eine Menüleiste gewählt, um den Nutzer einige Bedienungsaspekte zur Verfügung zu stellen. Über dieses Menü, welches mit einer „Vbox“ und „MenuBar“ erstellt wurde, erhält der Nutzer die Möglichkeit einen Graphen zu Importieren oder einen neuen Graphen zu erstellen. Da die Farbe der Top-Elemente eher dunkel gewählt wurde und die Schrift gut lesbar bleiben sollte, wurde die Schriftfarbe Weiß gewählt. Im Center-Element wurde ein „GridPane“ gesetzt, um die Einstellungen über „ToggleButtons“ ohne

Probleme anzuordnen. Die „ToggleButtons“ wurden einer „ToggleGroup“ zugeordnet, damit die Einstellungen für den Nutzer angenehmer zu treffen sind. Ein aktivierter „ToggleButton“ wird zu dem visuell als aktiviert dargestellt, das wird sowohl über den Text, als auch dem gesetzten Schatten realisiert. Das Unterfeld wird mit dem Bottom-Element abgeschlossen hier wurde ein „FlowPane“ rechtsbündig eingefügt, um den Nutzer zusätzlich zu der Menüleiste noch die Möglichkeit zu bieten, über den Button Next einen neuen Graphen zu erstellen.

2.6.2. NewGraphPannel

Für das „NewGraphPannel“ wurde ebenfalls ein „BorderPane“ gewählt, um die Oberfläche in Top-, Bot-, Left- und Center-Elemente einzuteilen. Im Top-Pannel befindet sich wie auf der Startseite eine „MenuBar“, um den Nutzer den Rücksprung zur Startseite als auch den Export seines Graphen zu ermöglichen. Das Left-Element dient als Seitenmenü, welches dem Nutzer die Steuerung zum Erstellen eines Graphen ermöglicht, er kann über den „Button“ Knoten im Center-Element Knotenpunkte erstellen und über einen der beiden Verbindungsbuttons eine Verbindung mit Klick auf die Knoten herstellten. Auch diese „Buttons“ befinden sich in einer „ToggleGroup“ um den Nutzer die Steuerung des Seitenmenüs zu vereinfachen. Auch hier ist über einen Schatten der geklickte „ToggleButton“ für den Nutzer deutlich zu erkennen. Es wurde eine „Vbox“ mit einem weißen Hintergrund als Center-Element gesetzt, um die „OnClick“ Funktionalität auf Basis der „Vbox“ zu ermöglichen. Im Bottom-Element befindet sich ein Status „Label“, in dem die Aktion des gewählten „ToggleButtons“ noch einmal in Schrift für den Nutzer dargestellt wird.

2.7. Umsetzung eines Graphen in JavaFX

Für die visuelle Umsetzung des Graphen wurden feste Größen eingebaut. Der Radius, die Strichstärken als auch die visuellen Elemente wurden so definiert. Die Farben der Knotenpunkte und Verbindungen wurden in einem schlichten Grau gehalten und transparenter gesetzt um die Namen, Kosten der einzelnen Knoten und Verbindungen hervorzuheben. Die Positionen von Namen- und Kosten „Label“ werden anhand der Koordinaten so berechnet, dass sie fast immer außerhalb der einzelnen Objekte liegen. Falls ein „Label“ jedoch mal über einen Knoten liegen sollte wird aufgrund der Transparenz die Sichtbarkeit der einzelnen Objekte das „Labels“ hervorgehoben, sodass die gesetzten Werte für den Nutzer immer erkennbar bleiben. Die Pfeil Klasse wurde im Internet zur Verfügung gestellt und minimal angepasst. (kn0412, 2020) Die Schleifen Klasse wurde mit einem Arc Shape konstruiert und als Halbkreis dargestellt. Bei der Erstellung eines Knotens werden die Koordinaten des geklickten Center Pannels an die „GraphNode“ Klasse mit den Hauptlayout übergeben und das erstellte „GraphNode“ Objekt wird der Array Liste aus der „GraphInformations“ Klasse angefügt, um den Zugriff auf das erstellte Objekt nicht zu verlieren. Über einen Rechtsklick auf den dabei erstellten Kreis, wird dem Nutzer ermöglicht, das Objekt durch ein „ContextMenu“ zu Löschen oder einen Namen zusetzen. Beim setzen des Namens öffnet sich ein Eingabe Feld, was es dem Nutzer ermöglicht diesen einzugeben. Der eingegebene Name wird dann in dem bereits dafür erstellten „Label“ in der „GraphNode“ Klasse gesetzt, zusätzlich wird auch der „String“, welcher für den Namen angedacht ist gesetzt. Jedes erstellte Knoten Objekt bekommt nun einen „OnClickListener“, um die Verbindungen erstellen zu können. Eine Verbindung kann nur von einem Knoten zu einem anderen oder zu sich selbst existieren. Über diese „OnClickListener“, wird dann die Verbindungserstellung geregelt. Dafür wird ein Klickzähler genutzt der Registriert, wann ein erster und wann ein zweiter Klick getätigt wurde.

Während dieses Vorgangs werden dann die Knoten Koordinaten gespeichert und eine Verbindung zwischen den beiden Knoten wird erstellt. Falls der Start- und Endknoten gleich sind wird eine Schleife dargestellt. Auch für die Verbindung wird ein „ContextMenu“ gesetzt, welches es dem Nutzer ermöglicht eine Verbindung zu Löschen oder die Kosten zu setzen. Beim Setzen der Kosten wird der eingegebene Wert in der zuständigen „Connection“ Klasse gesetzt und das erstellte Label genutzt. Dieser Prozess ist aufgrund der abstrakten Klassenstruktur nicht mit einem „Cast“ zu differenzieren. Die erstellte Verbindung wird der zuständigen Array Liste in den „GraphInformations“ angefügt, um den Zugriff nicht zu verlieren. Zum Löschen der gesetzten Informationen werden die internen Klassen Methoden genutzt, um alle durchgeführten Änderungen Rückgängig zu machen.

2.8. Quellcode Design Aspekte

Der Quellcode des „Graph Visualizers“ ist so designed, dass eine Änderung der Texte und der Style-Klassen, über eine Datei der Konstanten Klasse gesteuert wird. Außerdem ist durch das Abstrakte Klassenkonzept in den Verbindungen eine getrennte Aufbewahrung der Arten nicht notwendig, denn Funktionen die benötigt werden lassen sich über die Oberklasse „Connection“ ansteuern. Dadurch konnte eine Array Liste innerhalb der „GraphInformations“ eingespart werden. Zudem werden unnötige „Casts“ innerhalb der Programmierung erspart, was zu einem lesbaren Quellcode führt. Durch die separate Einstellungsklasse kann vom ganzen Programm auf die gesetzten Einstellungen zugegriffen werden, was unnötige lokale Variablen und Parameter in Methoden einspart. Das gleiche Prinzip wurde bei der „GraphInformations“ Klasse verwendet um die Übergabe als unnötige Parametrierung und lokale Variablen zu vermeiden. Das wiederum minimiert potentielle Fehlerquellen. Zusätzlich wurde eine „StageChanger“ Klasse implementiert, welche das Wechseln zwischen den einzelnen Stages der Startseite und des „NewGraphPannel“ erleichtert.

3. Klassendiagramme und Prozesse

Die nachfolgenden Klassendiagramme wurden in drei Teile unterteilt, um Unübersichtlichkeiten zu vermeiden Der erste Teil zeigt den Prozess auf der Startseite des „Graph Visualizers“ hier werden die Einstellungen gesetzt und die Startseiten „Stage“ innerhalb des „StageChangers“ gespeichert. Außerdem ruft die „Graph Visualizer“ Klasse Daten aus der Constant Klasse ab, um Text- und Style Elemente ordentlich darzustellen. Sobald die Einstellungen gesetzt sind kann eine neue Instanz der Klasse „NewGraphPannel“ erstellt werden, welche nach dem instanziieren Zugriff auf die Einstellungen hat und eine Instanz der Klasse „GenerateSceneIds“ erstellt, um den weiteren Verlauf für die Knoten Erstellung zu ermöglichen. Die „NewGraphPannel Stage“ wird in der „StageChanger“ Klasse gespeichert. Die Klasse „NewGraphPannel“ nutzt ebenfalls die Constants Klasse um Text- und Style Elemente darzustellen.

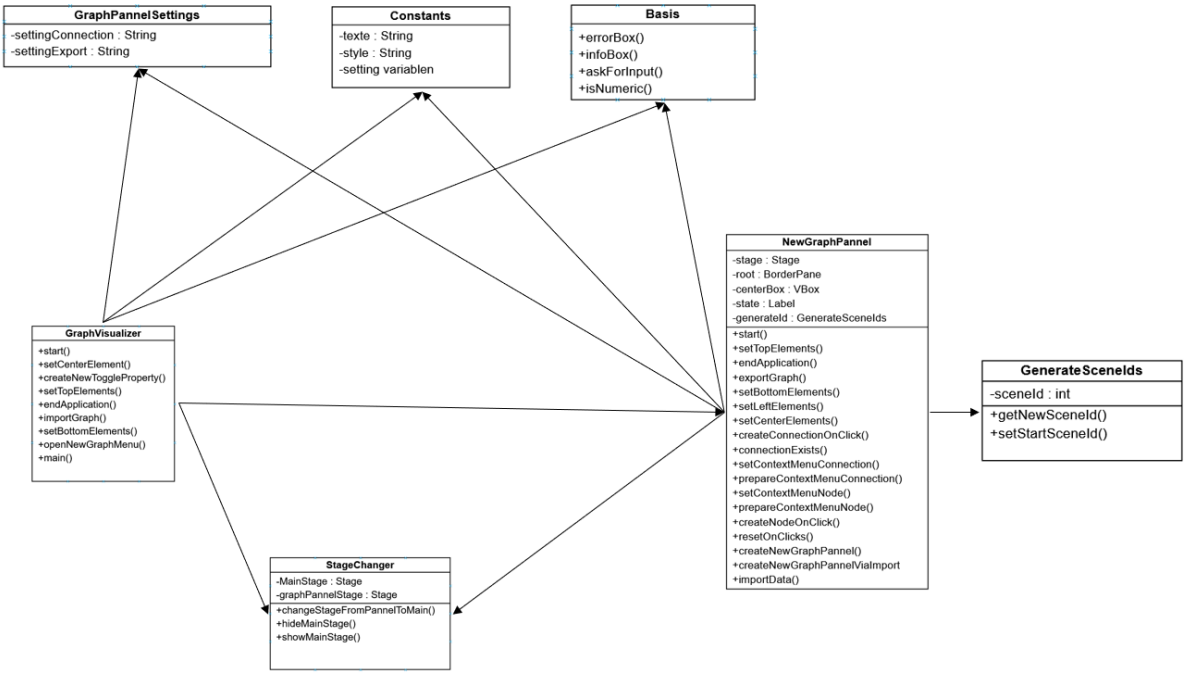

ABBILDUNG 1 - STARTSEITE / NEWGRAPHPANNEL

Im zweiten Diagramm wird die Erstellung von Knoten und Verbindungen gezeigt. Um eine Verbindung oder einen Knoten zu erstellen wird die benötigte Klasse einfach instanziiert. Damit die Informationen nicht verloren gehen, werden diese in der „GraphInformations“ Klasse abgelegt. Die Basis Klasse enthält die angezeigten Error-, Info- und Eingabeboxen, welche von den einzelnen Klassen verwendet werden um Fehler, Informationen oder Eingaben zu verwalten. Die Verbindungsklassen greifen außerdem auf die „GraphLoop“ Klasse zu, um eine Schleife visuell für den Nutzer darzustellen. Eine „gerichtete Verbindung“ muss zusätzlich noch den Pfeil realisieren können, deswegen greift sie auf die Arrow Klasse zu.

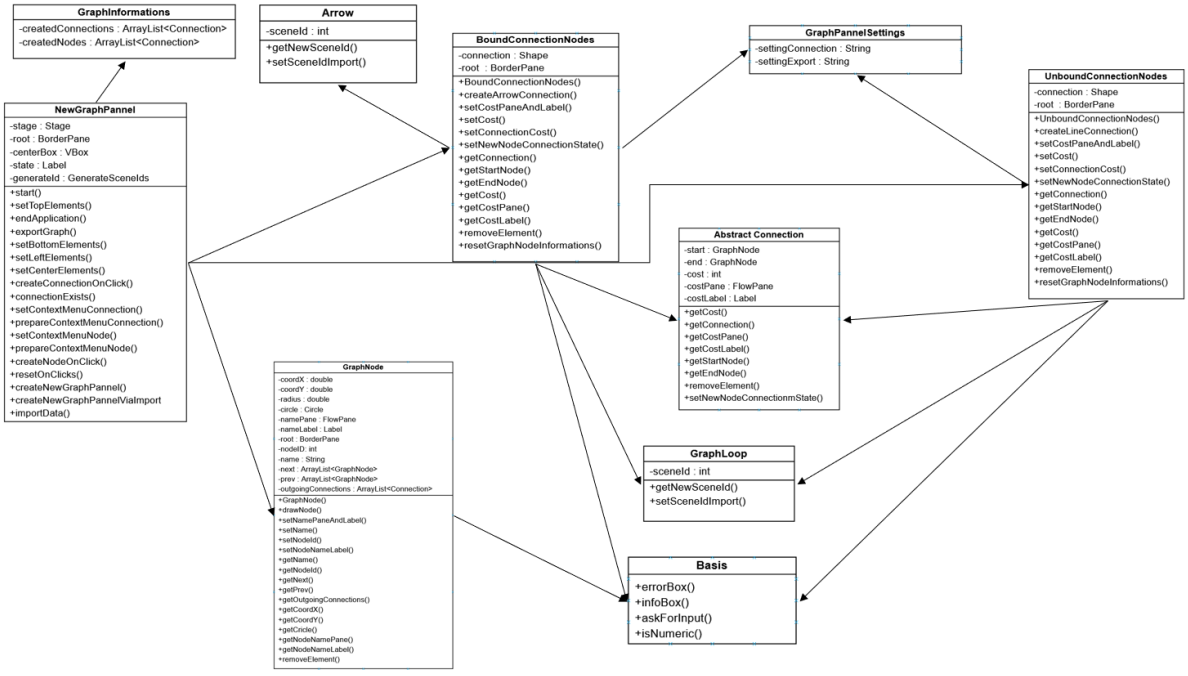

ABBILDUNG 2 - GRAPHEN ERSTELLUNG

Der dritte Teil der Klassendiagramme zeigt die Prozedur der Import / Export Erweiterung. Die Klasse „NewGraphPannel“ wählt die über die Einstellungen gesetzte Funktionalität. In den Klassen „ExportGraph“ und „ImportGraph“ werden dann die Graph Informationen verarbeitet und die „GraphPannelSettings“ überprüft. Dem Nutzer wird nach erfolgreicher Durchführung eine Rückmeldung, über die Basis Klasse generiert.

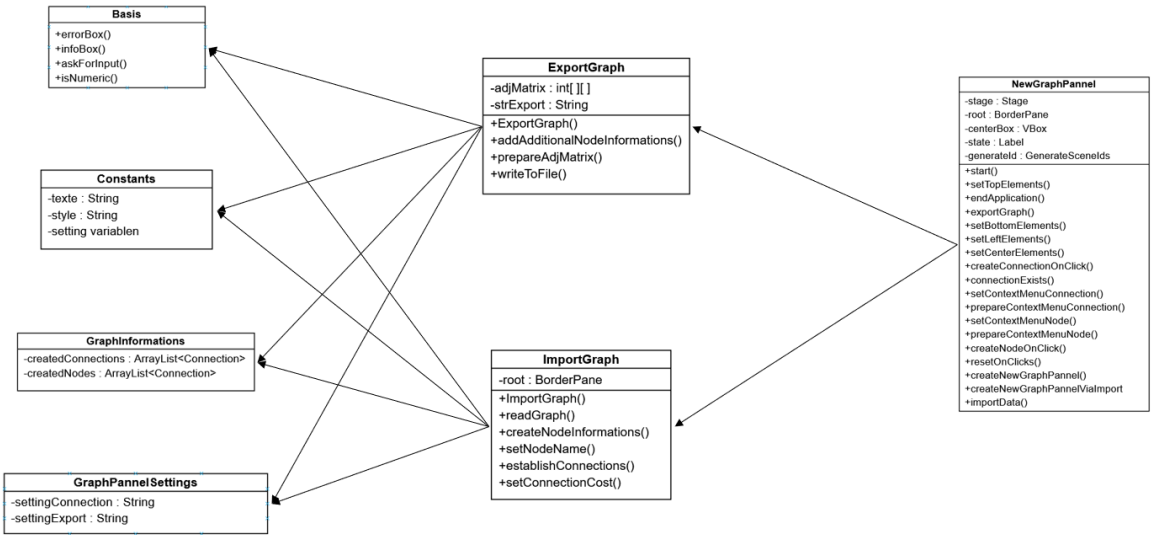

ABBILDUNG 3 - IMPORT / EXPORT ERWEITERUNG

4. Beschreibung der Oberfläche

4.1. Entwurf

Beim Entwurf der Oberfläche wurde auf ein strukturiertes, sauberes und einfaches Design geachtet, um für den Nutzer die Funktionalität in den Vordergrund zu stellen. Die Einstellungen wurden vorerst gering gehalten. Ein weiterer positiver Aspekt der Oberfläche ist die einfache und schnelle Möglichkeit einen Graphen zu erstellen.

4.2. Ablauf

Der Ablauf ist simpel, nach dem Start der Anwendung öffnet sich die Startseite, in der die einzelnen Einstellungen für den Graphen getroffen werden können. Hier kann der Nutzer wählen zwischen einem „gerichteten“ oder „ungerichteten“ Graphen und der gewünschten Export Art. Damit die Graphen Erstellung nicht ohne Einstellungen gestartet werden kann, wurde hierfür eine Überprüfung eingebaut. Die Startseite des Graphen ist auf Abbildung 4 zu sehen. Über das obere Menü kann der Benutzer einen Graphen importieren oder einen neuen Graphen erstellen.

Wurde der Menüpunkt „Neuen Graphen erstellen“ ausgewählt so öffnet sich ein Fenster in dem der Nutzer die Möglichkeit erhält Knotenpunkte und Verbindungen zur erstellen. Das Fenster für die Graphen Erstellung ist in Abbildung 7 zu sehen. Über das Seitenmenü kann dann die gewünschte Aktion gewählt werden und durch einen Klick an eine beliebige Position im mittleren Feld, kann der Nutzer dann einen Knoten erstellen. Möchte der Nutzer eine Verbindung erstellen, so muss er lediglich die gewünschten Punkt nacheinander auswählen. Ein erstellter Graph ist im Abbildung 10 zu sehen. Während der Erstellung hat der Nutzer über einen Rechtsklick, auf ein Item, die Möglichkeit ein Kontext Menü aufzurufen und das Item zu löschen oder den Namen / Kosten zusetzen. Das Kontext Menü ist in **Fehler! Verweisquelle konnte nicht gefunden werden.** zusehen. Ist der Nutzer mit der Erstellung seines Graphen fertig, so kann er diesen über die obere Menüleiste an einen gewünschten Speicherplatz exportieren. Diesen Prozess kann man in der Abbildung 11 sehen. Beim importieren des Graphen wird er aufgefordert eine Datei auszuwählen, wenn die ausgewählte Datei das richtige Format hat, wird der Graph importiert und der Nutzer kann diesen weiterbearbeiten. Das ist auf Abbildung 15 zusehen.

5. Entwicklungsperspektiven

Der „Graph Visualizer“ bietet viel Potential für nachfolgende Entwickler. Durch das relativ einfache design des Quellcodes können beispielsweise noch Algorithmen zur Graphen Berechnung ergänzt werden, so kann auf dem erstellten Graphen zum Beispiel ein „Dijkstra“ Algorithmus ausgeführt. Die Klasse „GraphNode“ ermöglicht durch ihre vielen Attribute die Umsetzung von Algorithmen. Diese müssen nur an das gegebene Format angepasst werden. Um den „Graph Visualizer“ für die Forschung und Entwicklung interessanter zu gestalten, könnte ein Scroll bares „Pane“ eingebunden werden, damit der Nutzer die Möglichkeit erhält größere Graphen zu designen. Für das Einbinden der Graphen Algorithmen wie „Dijkstra“

muss lediglich, das Seitenmenü und deren Kriterien angepasst werden, außerdem muss noch ein Erkennungsalgorithmus für Schleifen und negative Kosten eingeführt werden. Zusätzlich würde sich Beispielsweise noch eine separate Export Erweiterung anbieten, damit das Ergebnis gespeichert werden kann. Es könnte auch ein Färbungsalgorithmus eingeführt werden, welcher die einzelnen Knoten mit drei Farben markieren kann. Eine Weiterentwicklung des „Graph Visualizers“ kann nach Belieben vorgenommen werden.

6. Zusammenfassung

Das gezeigte Programm löst die oben genannte Problemstellung für kleinere bis mittlere Graphen und ist ideal für die Erstellung von Test Graphen zu lernzwecken. Durch sein einfaches und schlichtes Design, erleichtert es dem Nutzer die Bedienung. Über die per Klick erstellbaren Knotenpunkte und Verbindungen hat der Nutzer die Freiheit seine Graphen nach Belieben zu erstellen. Die Import- und Export Erweiterung ermöglicht es ihm sein Graphen mit anderen zu teilen, weiterzuverarbeiten oder über den Import noch zu ergänzen. Der Quellcode des Programms bietet außerdem Möglichkeiten für Erweiterungen, die in den Entwicklungsperspektiven beschrieben wurden. Die in der Graphentheorie zu beachtenden Voraussetzungen wurden im Programm umgesetzt. Das Arbeiten mit dem „Graph Visualizer“ ist einfach und schnell, da die benötigten Graph Informationen nicht direkt in Multidimensionalen Arrays abgelegt werden, um Schleifendurchläufe zu vermeiden. Das verwenden eines abstrakten Klassen Konzepts bei den Verbindungstypen erspart das unterscheiden der einzelnen Verbindungsarten in vielen Prozessen ein, was den Quellcode lesbarer gestaltet. Die Einstellungsklasse ermöglicht es die gesetzten Einstellungen vom ganzen Programm aus verarbeiten zu können. Durch den Implementierten „StageChanger“ ist ein Wechsel zwischen den beiden Layouts ohne Probleme möglich. Über die Nutzung der bereits implementierten Formen, war es möglich einen Pfeil und eine Schleife visuell darzustellen. Dank der eigebunden CSS-Datei konnten die „Strings“ für die Styles eingespart werden. Durch die „Constants“ Klasse können alle Texte editiert werden, wodurch der Entwickler viel Zeit bei der Bearbeitung einsparen kann.

7. Anhänge

ABBILDUNG 4 - STARTSEITE

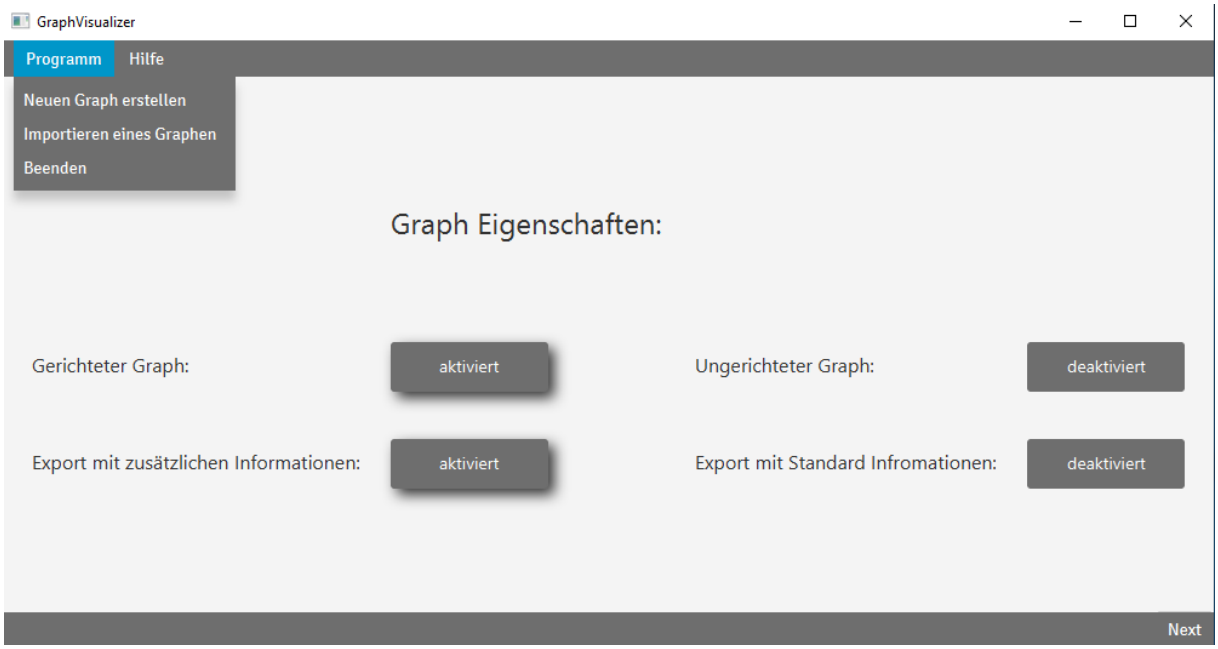

ABBILDUNG 5 - STARTSEITE MENÜ

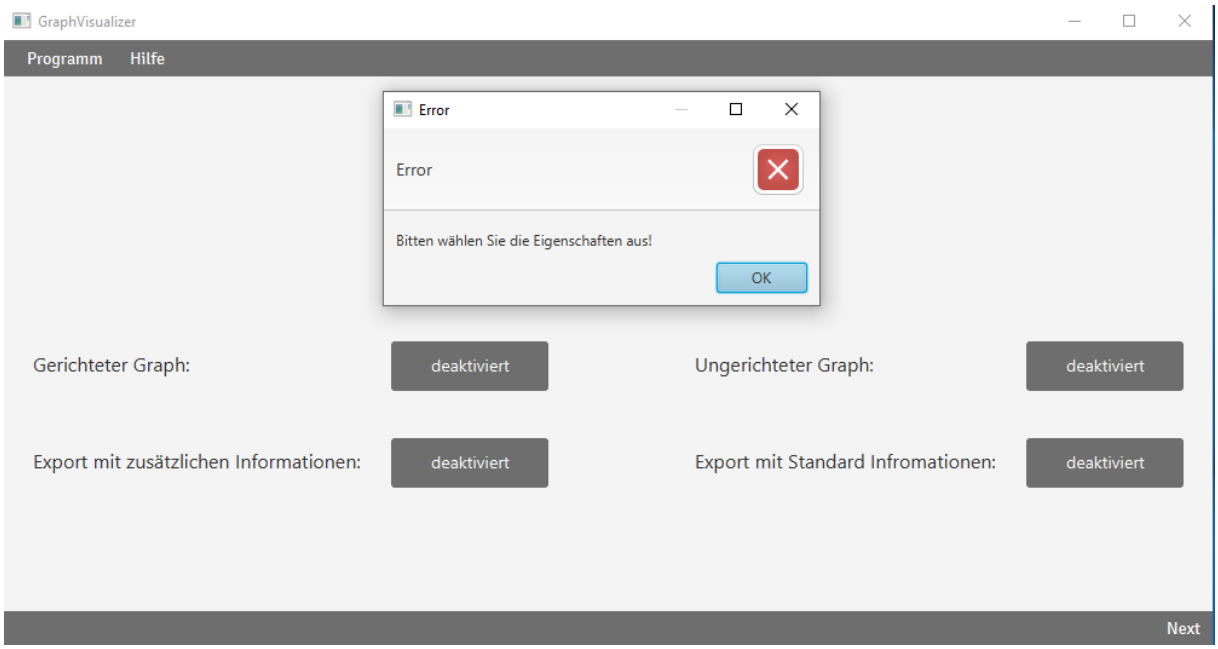

ABBILDUNG 6 - FEHLER OHNE GESETZTE EINSTELLUNGEN

ABBILDUNG 7 - NEWGRAPHPANNEL

ABBILDUNG 8 - KNOTEN ERSTELLUNG AUSGEWÄHLT

ABBILDUNG 9 - NEWGRAPHPANNEL MENÜ

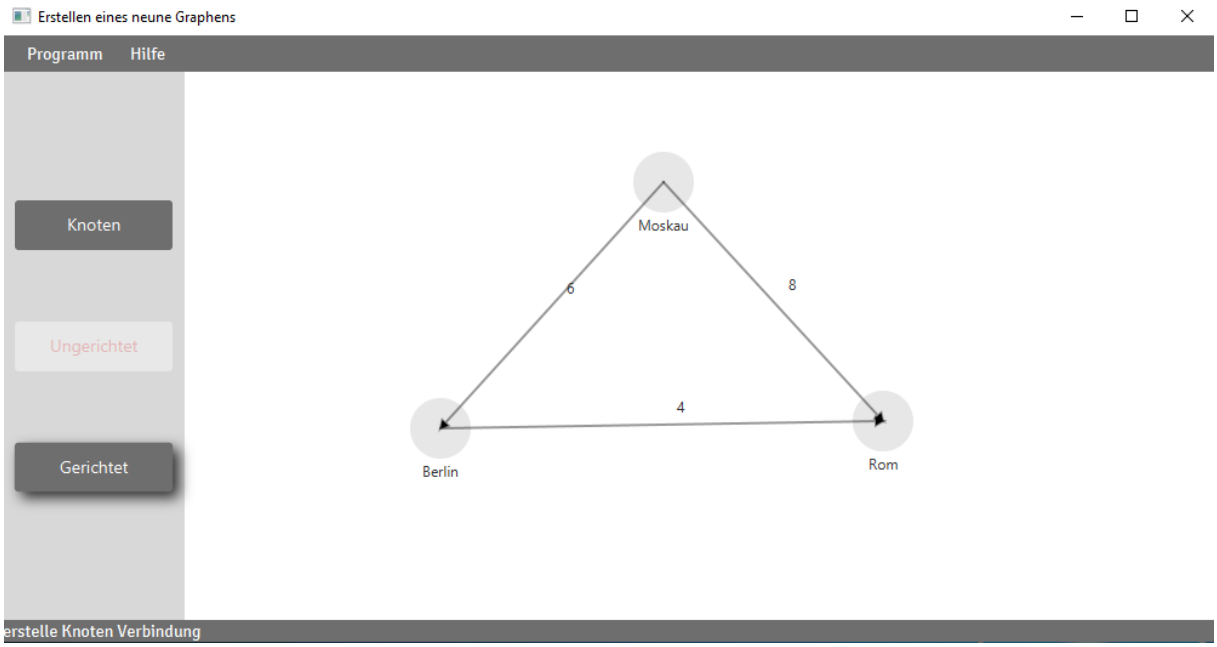

ABBILDUNG 10 - ERTSELLTER GRAPH

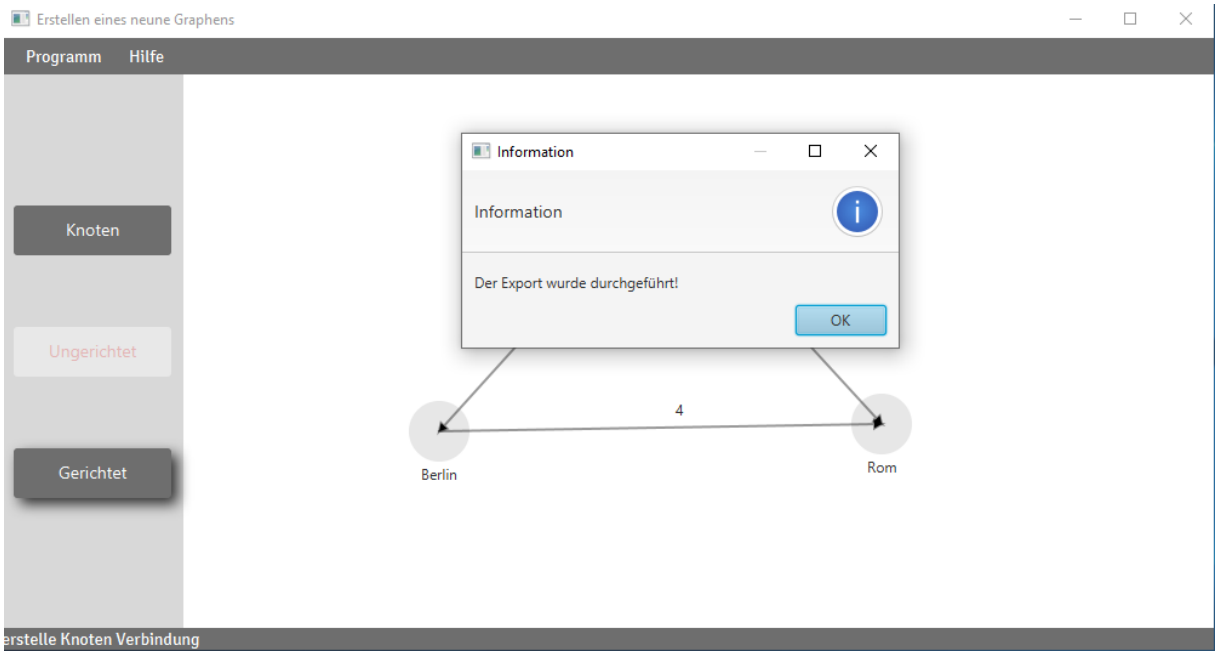

ABBILDUNG 11 - EXPORT DES ERSTELLTEN GRAPHEN

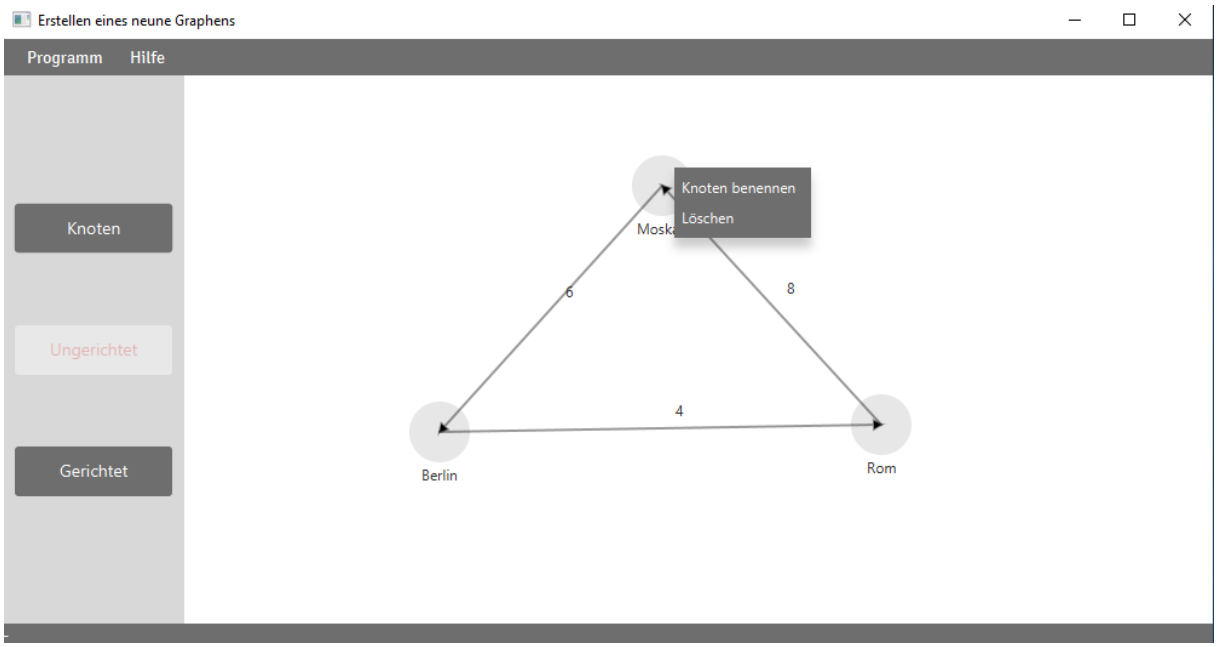

ABBILDUNG 12 - ITEM KONTEXT MENÜ

```
Adjazenz Matrix:

0 1 1
0 0 1
0 0 0
```

ABBILDUNG 13 - STANDARD EXPORT IN TXT FILE

```
Zusaetzliche Node Informationen:

------------------------------------------------------------
Knotenname: Moskau
KnotenID: 1
Koordinate X: 546.0
Koordinate Y: 131.0
Nachfolger Knoten:
        Knotenname: Berlin | Id: 2
        Knotenname: Rom | Id: 3


Vorgaenger Knoten:
        Keine Vorgaenger gefunden!

Ausgehende Verbindungen:
 bound Verbindung: startKnotenId: 1 | endKnotenId: 2 | Kosten: 6
 bound Verbindung: startKnotenId: 1 | endKnotenId: 3 | Kosten: 8
------------------------------------------------------------

------------------------------------------------------------
Knotenname: Berlin
KnotenID: 2
Koordinate X: 386.0
Koordinate Y: 361.0
Nachfolger Knoten:
        Knotenname: Rom | Id: 3


Vorgaenger Knoten:
        Nachfolgende Knotenname: Moskau | Id: 1

Ausgehende Verbindungen:
 bound Verbindung: startKnotenId: 2 | endKnotenId: 3 | Kosten: 4
------------------------------------------------------------

------------------------------------------------------------
Knotenname: Rom
KnotenID: 3
Koordinate X: 728.0
Koordinate Y: 373.0
Nachfolger Knoten:
        Keine Nachfolger gefunden!

Vorgaenger Knoten:
        Nachfolgende Knotenname: Berlin | Id: 2 Nachfolgende Knotenname: Moskau | Id: 1

Ausgehende Verbindungen:
        Keine ausgehenden Verbindungen gefunden!
------------------------------------------------------------

Adjazenz Matrix:

0 1 1
0 0 1
0 0 0
```

ABBILDUNG 14 - ERWEITERTER EXPORT

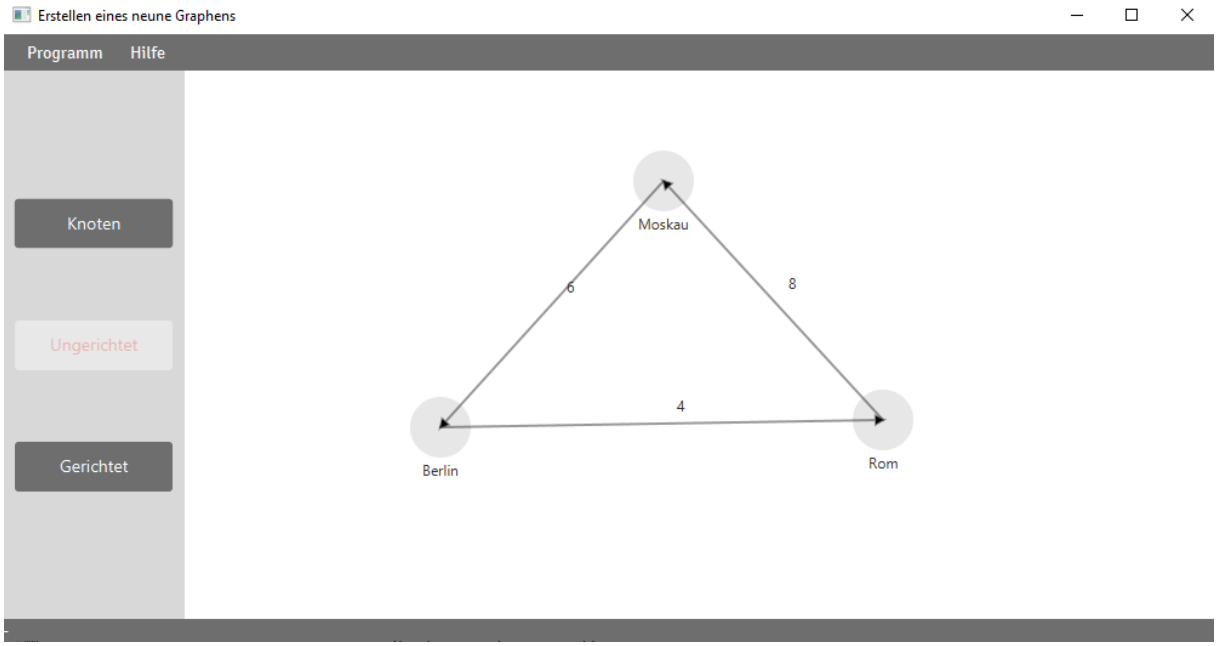

ABBILDUNG 15 - IMPORTIERTER GRAPH

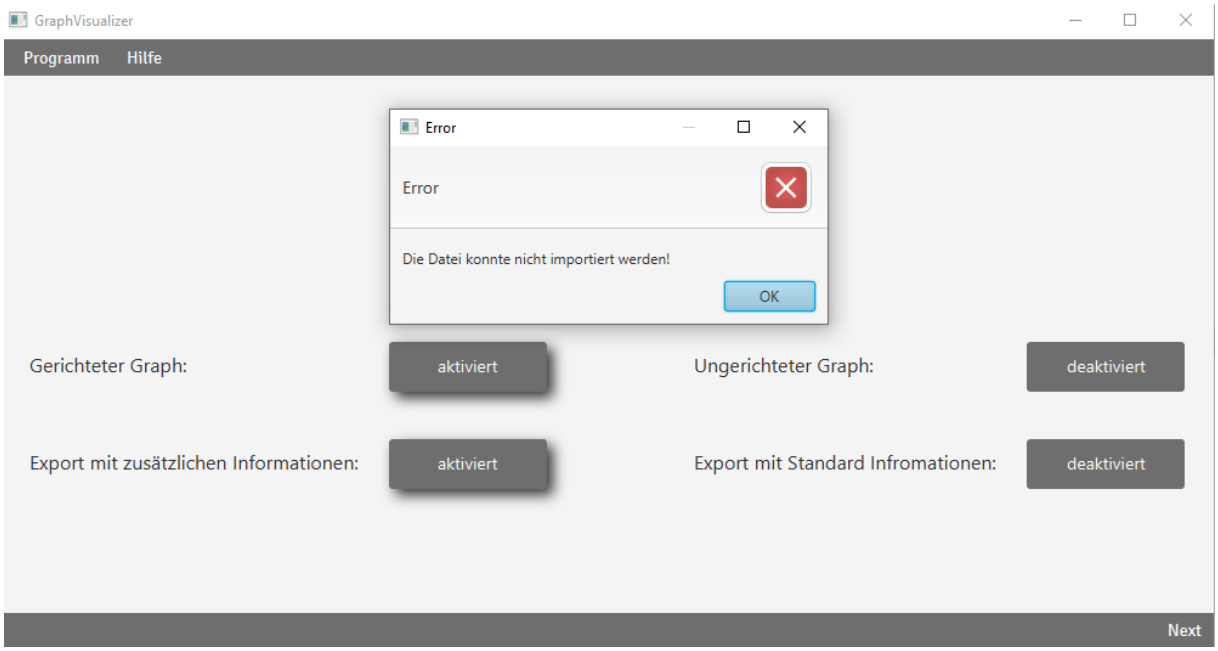

ABBILDUNG 16 - FEHLERHAFTER IMPORT

8. Quellen

kn0412. (22. 02 2020). *GitHub*. Von https://gist.github.com/kn0412/2086581e98a32c8dfa1f69772f14bca4 abgerufen

Visual Paradigm. (22. 02 2020). *Visual-Paradim Online Diagrams*. Von https://online.visual-paradigm.com/de/diagrams/ abgerufen

Wilhelm, M. (22. 02 2020). Basis Klasse.