Projekt 1: Graph-Visualisierung

Hinweis: Dieses Dokument erhebt keinen Anspruch auf Vollständigkeit und wird laufend an den Projektfortschritt angepasst.

# Projektbeschreibung

Es soll eine Software erstellt werden, welche Graphen darstellen kann, bzw. es erlaubt, Graphen aus einer Textdatei zu importieren und zu bearbeiten.

Gleichzeitig soll die Software zur animierten Visualisierung 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 für beliebige Graphen-Algorithmen eignen. Neue Graphen-Algorithmen sollen ohne grossen Aufwand hinzugefügt werden können.

# User Requirements

1. **Daten-Aufbereitung und Visualisierung:**
   * Es können aus einer Liste einige bereits vorhandene Graphen und Algorithmen zur animierten Visualisierung ausgewählt werden.
   * Der Benutzer kann einen beliebigen Graphen aus einer Datei in das Programm importieren und wieder entfernen.
   * Ein beliebiger Graph-Algorithmus kann ohne grossen Aufwand in das Programm eingebunden und wieder entfernt werden.
   * Der Graph wird durch Kreise (Knoten), Geraden (ungerichtete Kanten), Pfeile (gerichtete Kanten) und Beschriftungen (Knotenbezeichnungen, Kantenbezeichnungen und Kantengewichte) visualisiert.
2. **Graph-Animation:**
   * Für einen gewählten Algorithmus und Graphen kann der Benutzer eine animierte Visualisierung starten. Die Animation kann jederzeit gestoppt oder abgebrochen werden.
   * Der Benutzer kann auch Schritt für Schritt durch den sich in Ausführung befindenden Algorithmus klicken (vor und zurück in einem oder mehreren Schritten; an den Anfang und ans Ende).
3. **Graph-Bearbeitung:**
   * Die einzelnen Knoten des Graphen können in ihrer Position, Grösse, usw. beliebig verändert werden.
   * Es können neue Knoten und Kanten hinzugefügt werden.
   * Der neue Graph kann zur späteren Wiederverwendung gespeichert werden.

Bemerkung: Die Punkte 1 und 2 haben eine höhere Priorität als Punkt 3.

# System Requirements

Es folgt eine Beschreibung der Requirements aus technischer Sicht:

1. **Daten-Aufbereitung und Traversierung:**
   * Bereits importierte Graphen und eingebundene Algorithmen bleiben persistent im Programm erhalten.
   * Es stehen mindestens die folgenden bereits implementierte Algorithmen zur Verfügung:

* Dijkstra: Suchen des kürzesten Weges zwischen zwei Knoten in einem gerichteten und gewichteten Graphen
* Kruskal: minimaler Spannbaum berechnen
* Rekursive Tiefensuche
  + Ein neuer Graph kann aus einer GraphML-Datei importiert werden. GraphML ist ein auf XML basierendes Format zur Speicherung von Graphen. Der Input wird vor dem Import auf Korrektheit geprüft.
  + Es können gerichtete und ungerichtete sowie gewichtete und ungewichtete Graphen mit Einfach- und Mehrfachkanten verwendet werden.
  + Ein beliebiger Algorithmus kann eingebunden und verwendet werden. Voraussetzung ist, dass ein vorgegebenes Interface implementiert wird. Alle Operationen des Algorithmus müssen auf dem vom Interface vorgegebenen Graph-Objekt durchgeführt werden. Zudem muss (über Annotations) angegeben werden, für welchen Graphen-Typ der Algorithmus zulässig ist (gerichtet, ungerichtet, gewichtet, ungewichtet, Einfach- oder Mehrfachkanten. Die notwendigen Input-Parameter der Methode müssen ebenfalls spezifiziert werden können. Hinweis: Die genaue Spezifizierung des Interfaces erfolgt im Verlauf des Projektes.
  + Der Start- und Endknoten des Graphen muss bei manchen Algorithmen spezifiziert werden.
  + Nach der Wahl des Graphen und Algorithmus wird zuerst eine Vorberechnung der Daten für die Animation durchgeführt. Erst dann kann die Animation gestartet werden.

1. **User Interface und Grafik:**
   * UI Elemente für Datenaufbereitung: Neuen Graphen einbinden, Neuen Algorithmus einbinden, vorhandenen Graph oder Algorithmus auswählen, vorhandenen Graphen oder Algorithmus aus Liste entfernen, Vorberechnung der Daten starten.
   * UI Elemente für Traversierung: Animation starten, anhalten, abbrechen, Progressbar für Animationsfortschritt, Inkrement festlegen für Anzahl Schritte pro Klick, vor und zurück durch den Algorithmus gehen (schrittweise), an das Ende oder den Anfang der Animation springen, Zeitintervall zwischen einzelnen Animationsschritten festlegen.
   * Der Animationsfortschritt wird durch Farbänderungen der Knoten und Kanten verdeutlicht. Bei jedem Knoten können (abhängig vom gewählten Algorithmus) während der Animation Zwischenergebnisse angezeigt werden.
   * Für dieses Projekt wird das Java Universal Network/Graph Framework (JUNG) eingesetzt. Damit kann die Visualisierung und Bearbeitung des Graphen erheblich vereinfacht werden.

# Systemarchitektur

Es wird das Model-View-Controller (MVC) Architektur-Pattern verwendet. Zur Visualisierung der Graphen wird das Java Universal Network/Graph Framework (JUNG) eingesetzt. Die Systemkomponenten sind in Schichten unterteilt, wobei nur eine höher liegende Schicht direkten Zugriff hat auf eine darunterliegende Schicht (Schichtenrchitektur ). Auf Architektur-Ebene lassen sich grob die folgenden Systemkomponenten identifizieren.

* **View**: User Interface und Grafik Panel
* **Controller**: Listener-Interfaces und Listener-Klassen
* **Model**:
  + Graph (Erweiterung der vorhandenen JUNG-Framework-Klassen und Interfaces: Klassen für Graphen, Knoten, Kanten, …)
  + Daten-Aufbereitung
  + Graph-Traversierung

# Projektmanagement

Für dieses Projekt wird Scrum als Vorgehensmodell verwendet (stark vereinfachte Variante von Scrum, da nur zwei Personen am Projekt beteiligt sind). Es ist folgender Zeitplan vorgesehen:

* Projektplanung und Systemarchitektur (ca. 3 Wochen):
  + Einarbeitung in die Thematik (Graphen, Algorithmen, …)
  + Erstellung der Requirements Spezifikation
  + Design der Systemarchitektur
* Wöchentliche Sprint Zyklen (ca. 10 Wochen)
  + User stories wählen für nächsten Sprint
  + (Re-) Design der Klassenhierarchie, Interfaces definieren, Tests definieren, Implementation
  + Review, Anpassung der Planung
* Projektabschluss und Präsentation (restliche Zeit):
  + Code Refactoring
  + Systemtests
  + Dokumentation
  + Präsentation

Core stellt grundlegende Dienste zur Verfügung (Application-Service)

Core stellt Datenstruktur Graph zur Verfügung, Import und Speicherung von Graphen, führt Berechnungen durch, liefert als Resultat einen StepIterator

MVC:

* Model:
  + UI-Model
  + Data-Model
* View: bekommt Zustand vom UI-Model und Daten vom Data-Model
* Controller:
  + Führt Applikationslogik aus (Dienste, die von der Core-Schicht zur Verfügung gestellt werden)
  + Update des Models