DHBW Stuttgart Campus Horb Vorlesung „Programmieren I + II“ 2014

Programmieraufgabe: Routenplaner

Von: Julius Mahlenbrey, Simon Stehle, Sebastian Penhouet und Robin Kinting

Vorwort

Die Aufgabe dieses Projekts war es, mittels des Algorithmus von Dijkstra auf den Daten des deutschen Autobahnnetzes einen Routenplaner zu implementieren.

Die Aufgabe wurde mit dem vorliegenden Programm vollständig gelöst. Darüber hinaus wurden einige weitere zusätzliche Funktionen implementiert, die komplette Liste aller Features folgt hier:

Features des Programms

- Mittels einer grafischen Benutzeroberfläche kann auf Basis von OpenStreetMap-Daten eine Route von einem Autobahnknoten zu einem anderen innerhalb eines Landes berechnet werden.
- Zur Berechnung benötig das Programm verschiedene Daten in Form von XML-Dateien, die es bei Bedarf automatisch herunterlädt und erzeugt. Während das Programm ausgeführt wird, sind diese Dateien vom Programm gesperrt, können also nicht vom Benutzer oder einem anderen Programm gelöscht werden.
- Beim Start des Programms wird ein Splash-Fenster angezeigt, das den Fortschritt bei der Prüfung der Programmvoraussetzungen anzeigt. Voraussetzungen für den Start des Programms sind eine Internetverbindung und die Verfügbarkeit des Datenverzeichnisses für die benötigten XML-Dateien.
- Die Eingabe des Starts und Ziel wird mit Vorschlägen, die nach Ähnlichkeit sortiert sind, automatisch vervollständigt.
- Zur Berechnung stehen zwei Algorithmen zur Verfügung:
    - Dijkstra-Algorithmus
    - Vereinfachter AStar-Algorithmus (ohne Heuristik)
- Bei der Berechnung kann wahlweise die schnellste oder die kürzeste Route zum Ziel berechnet werden. Geschwindigkeiten werden, wenn nicht anders in den OSM-Daten angegeben, mit 120 km/h auf Autobahnen und 60 km/h auf Ausfahrten angenommen.
- Die vom Programm berechnete Route ist einmal in Form einer simplen Liste dargestellt, in der die verschiedenen Zwischenstationen vom Start zum Ziel aufgelistet sind.
- Daneben wird eine grafische Aufbereitung des Ergebnisses geboten. Die Berechnete Strecke, sowie Start- und Zielknoten werden grafisch auf einer interaktiven OSM-Karte dargestellt.
- Über Programmeinstellungen kann die Farbe dieser Markierungen festgelegt werden.
- Alle im Programm getätigten Einstellungen werden in einer Settings-Datei im Datenverzeichnis gespeichert.
- Das Datenverzeichnis enthält außerdem die drei XML-Dateien, die die OSM-Daten enthalten.
- Das Datenverzeichnis befindet sich im Benutzerverzeichnis des aktuellen Benutzers unter (user.home)/DHBW-Routenplaner-2014
- Darüber hinaus ist es mit dem Programm möglich, Routen nicht nur in Deutschland, sondern auch in weiteren freigegebenen Ländern zu berechnen. Dazu werden die Daten des entsprechenden Landes herunter geladen und bleiben dann parallel nutzbar. Die Auswahl des Landes ist im Programm möglich.
- Zu guter Letzt bietet das Programm eine Update-Funktion für die vorhandenen OSM-Daten an, mit der die aktuellen Routen-Daten von OpenStreetMap heruntergeladen werden, und die Basis-Daten für die Berechnungen neu erzeugt werden.

Aufbau des Codes

Der Programmcode unseres Routenplaners ist auf mehrere Packages mit teilweise weiteren Unterpackages aufgeteilt.

Package-Struktur des Programms:

- Stamm-Package: *de.dhbw.horb.routePlanner*
    - *data*
    - *evaluation*
        - *aStar*
        - *dijkstra*
    - *ui*
        - *img*

Erläuterungen zu den einzelnen Packages

- Direkt im Stammverzeichnis liegen Klassen, die generelle Hilfsmethode bereitstellen und Programmkonstanten definieren
    - Das *data*-Package enthält Klassen, die für Erstellung, Verwaltung und Zugriff auf die dem Programm zugrunde liegenden XML-Dateien zuständig sind.
    - Im *evaluation*-Package liegen in Unterpackages die Klassen der beiden implementierten Algorithmen.
        - Im *aStar*-Package ist die Funktionalität des AStar-Algorithmus implementiert.
        - Im *dijkstra*-Package ist die Funktionalität des Dijkstra-Algorithmus implementiert.
    - Das *ui*-Package enthält alle Klassen und Nicht-Java-Dateien, die für das Anzeigen und die Funktion der grafischen Benutzeroberfläche benötigt werden.
        - Im *img*-Package liegen Bilddateien, die für die Funktion der eingebundenen Webseite nötig sind.

Genaueres zu einzelnen Klassen / Dateien

- In der Klasse **Constants** werden alle wichtigen Programmkonstanten verwaltet.
- Die Klasse **SupportMethods** bietet verschiedene Hilfsmethoden, die in mehreren anderen Klassen verwendet werden.
- Die Klasse **OverpassDownloader** lädt aktuelle OpenStreetMap-Daten aus dem Internet herunter und erzeugt daraus eine XML-Datei.
  Für den Download wurde die folgende Abfrage verwendet um alle XML Daten innerhalb eines Gebiets, die im Zusammenhang mit Autobahnen stehen, zu erhalten:

```
[timeout:3600];
area[name="Land"]->.a;
(way(area.a)[highway="motorway"];>;
way(area.a)[highway="motorway_link"];>;);
out;
```

- Die Klasse **JDomGraphDataCreator** erzeugt die beiden XML-Dateien mit den aufbereiteten Daten der einzelnen Kreuzungen (Nodes_[LAND].xml) und Autobahnabschnitten (Routes_[LAND].xml).
- Die Klasse **SettingsManager** übernimmt die Verwaltung der XML-Datei, in der die Programmeinstellungen gespeichert werden.
- Die Klasse **XMLFileManager** ist für die Verwaltung der XML-Dateien zuständig, welche die Routendaten enthalten. Dazu gehört auch das Sperren der Dateien, während sie in Benutzung sind.
- Die **.fxml-Dateien** im *ui*-Package repräsentieren das Layout sowie das Aussehen und den Aufbau der GUI.
- Die **Controller**-Klassen im *ui*-Package stellen die Funktionalität der GUI sicher.
- Die Klasse **UIEvaluationInterface** bereitet das Ergebnis einer Routenberechnung durch einen der beiden Algorithmen für die GUI auf.
- Die Dateien **overpass.html** sowie **OpenLayers.js** werden für das Anzeigen der Karte inklusive Markieren von Wegen und Knoten auf der Karte verwendet.

Verwendete Fremdbiliotheken und Technologien

- Unsere Anwendung wurde mit der Java-Version 8 Update 05 erstellt. Sie ist aktuell nur auf Windows getestet und für eine Bildschirmauflösung von mindestens 1280 x 720 Pixel optimiert.
- Zur Erstellung von Unit-Tests wurde **JUnit 4** verwendet.
- Mittels **ControlsFX** können auf einfache Art und Weise modale Popup-Fenster erstellt werden. In unserem Programm kommt das beispielsweise bei der Anzeige eines Berechnungsfehlers oder eines Info-Fensters zum Einsatz.
- Wir haben die GUI mit **JavaFX** umgesetzt. Von Vorteil waren hier neben einem hübschen Design die einfache Gestaltung des Layouts, sowie eine große Menge an bereits vorhandenen und einfach zu nutzenden Komponenten wie Fortschrittsbalken und einem eingebetteten Browser.
- Für das Lesen bzw. Auswerten der XML Dateien wird, wegen des schnellen Zugriffs, StAX verwendet. Dies ist in Java 8 bereits enthalten. Das Schreiben bzw. Speichern von XML Dateien ist mit **JDOM** allerdings wesentlich komfortabler und gleichzeitig ohne Performanceeinbußen möglich. Daher haben wir für diese Aufgabe auf diese Bibliothek zurückgegriffen.

- **OpenLayers** wurde gemeinsam mit der **Overpass-API** eingesetzt, um die grafische Anzeige der berechneten Route zu ermöglichen. Dabei wird in einen in der GUI eingebetteten Browser eine eigens angepasste HTML-Datei (overpass.html) geladen, der mittels Java und Javascript die berechneten Daten übergeben werden. Mit Hilfe der OpenLayers.js, die aufgrund ihrer Größe auch im Programm selbst eingebunden ist, können dann eine Kartenansicht geladen sowie die berechneten Wege markiert werden.

Bekannte Fehler / Probleme

- Da unser Programm mit dynamischen Daten arbeitet, die durch ein Update jederzeit vom Benutzer aktualisiert werden können, ist es besonders schwierig bis gar unmöglich vorherzusagen, welchen Inhalt die XML-Dateien mit den Route-Daten haben werden. Dadurch wird es auch äußerst schwierig Unit-Tests für die betroffenen Teile unseres Programms zu erstellen. Wir haben uns schließlich dagegen entschieden, für besagte Teile Unit-Tests zu schreiben, da diese durch die nötige Erstellung von statischen Testdaten und deren Einbindung ins Programm zu schwierig gewesen wäre.
- Ein bekanntes Problem unseres Programms ist es, dass das Anzeigen des Kartenhintergrund sehr lange dauern kann. Der Kartenhintergrund besteht aus „Kacheln“, kleinen Bilddateien, die von einem Dienst namens „Mapnik“ zur Verfügung gestellt werden. Gerade während der Abendzeit haben wir schon öfters beobachtet, dass das laden des Kartenhintergrunds, also der einzelnen Kacheln mitunter sehr lange dauern kann. Das liegt an der langsamen Datenübertragung der Bilddaten zwischen Mapnik-Web-Service und unserem Programm. Vermutlich sind die Mapnik-Server von Zeit zu Zeit ausgelastet und reagieren erst verspätet auf die Anfrage unseres Programms. Da die JavaFX-Komponente, in der die Webseite geladen wird, keinen Caching-Mechanismus bereitstellt, bleibt dieses Problem vorerst ungelöst.
- Zudem kommt es zu einer Warnung auf Konsolenebene, wenn man beispielsweise im Programm ein anderes Land auswählt und den darauffolgenden Dialog bestätigt. Bei der Warnung scheint es sich um ein generelles JavaFX-Problem zu handeln, das nichts Spezielles mit unserer Applikation zu tun hat. Diese Vermutung wird durch die Tatsache, dass die Warnung in einer Standart-Java8-Datei auftritt nochmals gestärkt. Somit ist es uns aktuell auch nicht möglich, diese Warnung zu verhindern. Allerdings hat die Warnung auch keinerlei festellbaren Einfluss auf die Funktion unseres Programms.

Ausblick auf mögliche weitere Features

- Speichern und Laden der Fenstergröße bei Programmneustart.
- Korrekte Fehlerbehandlung durch für den Benutzer ersichtliche Fehlermeldungen.
- Mehrsprachig durch Auslagern und Laden der Bezeichnungen über eine entsprechende Klasse.