routeKIT

Testdokument

7. März 2014

Kevin Birke
Felix Dörre
Fabian Hafner
Lucas Werkmeister
Dominic Ziegler
Anastasia Zinkina

betreut durch

Julian Arz
G. Veit Batz
Dr. Dennis Luxen
Dennis Schieferdecker

am

Karlsruher Institut für Technologie
Institut für Theoretische Informatik
Algorithmik II
Prof. Dr. Peter Sanders

Inhaltsverzeichnis

1 Einleitung

Dieses Dokument erläutert und protokolliert die Qualitätssicherung der Anwendung *routeKIT*. Es dokumentiert die gefundenen Fehler, die durchgeführten Unit-Tests, die Testüberdeckung und die durchgeführten Änderungen seit der Implementierungsphase.

routeKIT ist eine Anwendung zur Routenplanung; durch Verwendung von ↗Profilen kann sie dem Benutzer die optimalen ↗Routen für sein spezielles Fahrzeug angeben. Um die Routenberechnung zu beschleunigen, wird pro ↗Profil und ↗Karte eine zeitaufwändige ↗Vorberechnung durchgeführt.

2 Fehlerprotokoll

2.1 Kritische Fehler

/BG10/ Korrupte Indexdatei

Symptom Wenn die Indexdatei Karten oder Profile enthält, die nicht existieren, stürzt *routeKIT* beim Programmstart ab.

Ursache Es wird versucht, auf Karten/Profile, die `null` sind, zuzugreifen.

Maßnahme Karte oder Profil überspringen, wenn es `null` ist.

Status behoben

2.2 Funktionale Fehler

/BG20/ Fehlende Straßen

Symptom Im vergröberten Graph fehlen Straßen.

Ursache Die entsprechenden Kanten werden weggeworfen.

Maßnahme Ein falsches `break` entfernen, das nach der ersten Ein-Kanten-Straße das Erzeugen von Kanten abbricht.

Status behoben

/BG30/ Punkt-Projektion

Symptom Die Projektion von eingegebenen Koordinaten auf die nächstliegenden Kanten ist oft schief.

Ursache Die Projektion findet im Geokoordinatensystem statt und nicht nach der ↗Mercator-Projektion (also im ↗Slippy Map Tile (SMT)-Koordinatensystem).

Maßnahme Für die Abstandsberechnung ↗SMT-Koordinaten anstelle von Geokoordinaten benutzen.

Status behoben

/BG40/ Einbahnstraßen, Randfall

Symptom Eine Route auf einer einzelnen Kante, die nur in einer Richtung befahrbar ist, kann die Kante auch entgegen der Fahrtrichtung benutzen.

Ursache Es wird nicht verhindert, dass man in Einbahnstraßen entgegen der Fahrtrichtung losfährt.

Maßnahme Den ersten Schritt der Routenberechnung gesondert ausführen, um dies als Spezialfall zu behandeln.

Status behoben

/BG50/ Profilverwaltung: Warnung bei Enter

Symptom Wenn im Profilverwaltungsdialog Vorberechnungen gelöscht werden und er mit *Enter* geschlossen wird, erscheint keine Warnung. Beim Klick auf *OK* erscheint sie aber.

Ursache Der Code, der die Warnung erzeugt, liegt im Event-Handler des Buttons und nicht in der Methode der `ProfileManagerView`, die den Dialog schließt.

Maßnahme Den Code, der die Warnung erzeugt, in die `ok`-Methode der `ProfileManagerView` verschieben.

Status behoben

/BG60/ CLI-Hilfe Test

Symptom Der Test für die Ausgabe des Hilfetext für das Konsoleninterface schlägt auf Windows-Systemen mit einer `ArrayIndexOutOfBoundsException` fehl, obwohl er auf Linux-Systemen funktioniert.

Ursache Die Ausgabe des Hilfetexts wird in ein Bytearray geschrieben. Dieses wird mit der Größe des Referenzstrings alloziert. Da auf Linux nur ein Byte für den Zeilenumbruch genutzt wird und auf Windows zwei, läuft das Array über.

Maßnahme Den Referenzstring plattformabhängig mit den richtigen Zeilenumbrüchen erzeugen, damit er mit der Ausgabe exakt übereinstimmt.

Status behoben

/BG70/ Karten- und Profilverwaltung: Inkonsistenz

Symptom In der Kartenverwaltung ist ein Profil für die aktuelle Karte vorberechnet oder soll vorberechnet werden. Jetzt wird dieses Profil im Dialog für das Hinzufügen neuer Profile gelöscht. Das Profil ist weiterhin in der Liste der Karte vorhanden.

Ursache Beim Beenden des Dialogs werden gelöschte Profile nicht richtig erkannt und verarbeitet.

Maßnahme Weiteren Code hinzufügen, der alle gelöschten Profile ermittelt und sie direkt aus den Listen löscht.

Status behoben

/BG80/ Gelöschtes aktuelles Profil

Symptom Das einzige Nicht-Standard-Profil ist ausgewählt. In der Kartenverwaltung wird dieses Profil (im Hinzufügen-Dialog) gelöscht und der Dialog mit *OK* geschlossen. Beim erneuten Öffnen des Hinzufügen-Dialogs ist ein Standardprofil ausgewählt und die Eingaben sind nicht deaktiviert.

Ursache Der Dialog lädt das nicht mehr existierende aktuelle Profil und deaktiviert deshalb die Eingaben nicht. Das Profil fehlt aber in der Liste, wodurch ein anderes Profil ausgewählt ist, was nun ein Standardprofil ist.

Maßnahme Beim Löschen eines Profils oder einer Karte immer sicherstellen, dass die aktuelle Kombination nur aus nicht gelöschten Teilen besteht.

Status behoben

/BG90/ Falscher Infotext, wenn keine Route gefunden wird

Symptom Liegen Start- und/oder Zielpunkt außerhalb der Karte, d. h. `findNearestPointOnEdge` liefert `null`, so wird der Infotext „Route wird berechnet...“ angezeigt, obwohl keine Route existiert.

Ursache Zu Beginn der Routenberechnung wird `startCalculating` aufgerufen, wodurch „Route wird berechnet...“ angezeigt wird. In der Abbruchbedingung bei Start/Ziel `== null` wird die Routenberechnung abgebrochen, ohne `currentRoute` und `currentDescription` auf `null` zu setzen. Der Infotext bleibt unverändert auf „Route wird berechnet...“

Maßnahme In der Abbruchbedingung `currentRoute` und `currentDescription` auf `null` setzen, wodurch „Keine Route gefunden.“ angezeigt wird.

Status behoben

2.3 Benutzbarkeitsfehler

/BG100/ Wettlauf bei der Routenberechnung

Symptom Wenn während einer aktiven Routenberechnung eine weitere gestartet wird, kann es passieren, dass die erste Route angezeigt wird, obwohl Start und Ziel der zweiten ausgewählt sind.

Ursache Wenn die zweite Routenberechnung schneller ist, schreibt sie ihr Ergebnis zuerst und die erste Routenberechnung überschreibt dann anschließend die Route.

Maßnahme Den aktuellen Berechnungsthread speichern und die berechnete Route nur noch abschicken, wenn der berechnende Thread der aktuelle ist.

Status behoben

/BG110/ Zoom und Ziehen

Symptom Wenn die Karte während einer Ziehgeste gezoomt wird, verändert sich die Kartenansicht, sodass man auf einem komplett anderen Teil der Karte ist.

Ursache Während der Ziehgeste werden die Ursprungskoordinaten der Geste gespeichert. Durch das Zoomen werden nun die Koordinaten halbiert (oder verdoppelt), was sich aber nicht direkt auf die gespeicherten Koordinaten auswirkt. Wird nun die Ziehgeste fortgesetzt, werden diese halb (oder doppelt) so großen Koordinaten zurückgeschrieben, was den Sprung auf der Karte verursacht.

Maßnahme Zoomen während einer Ziehgeste verhindern.

Status behoben

/BG120/ Vergröberung nötig

Symptom Das Rendern der Kacheln ist auf kleineren Zoomstufen zu langsam.

Ursache Es werden zu viele feine Kanten gezeichnet.

Maßnahme Den Graph für die Anzeige vergröbern, sodass weniger Kanten gezeichnet werden müssen.

Status behoben

/BG130/ Routenberechnung: alle Kanten

Symptom Die Routenberechnung ist zu langsam, auch für kurze Routen.

Ursache Der Algorithmus fügt zu Beginn der Berechnung alle vorhandenen Kanten in einen Heap ein, obwohl die eigentliche Routenberechnung durch ↗Arc-Flags oder nah beieinander liegende Start- und Zielpunkte möglicherweise nur auf einen Bruchteil der Kanten zugreifen muss.

Maßnahme Den Heap „lazy“ initialisieren (nicht eingefügte Kanten sind „unendlich“ weit entfernt).

Status behoben

/BG140/ Vorberechnung: alle Kanten

Symptom Die Vorberechnung ist zu langsam.

Ursache Es werden für jeden der Dijkstra-Durchläufe am Anfang alle Kanten des Graphen in den Heap eingefügt, auch wenn später möglicherweise nie darauf zugegriffen wird.

Maßnahme Kanten nur noch bei Bedarf in den Heap einfügen.

Status behoben

/BG150/ Falscher Infotext während Routenberechnung

Symptom Während der Routenberechnung zeigt die Wegbeschreibungs-/Info-Box den Text „keine Route gefunden“ an.

Ursache Begonnene Routenberechnungen werden dem `RouteModel` nicht mitgeteilt, so dass die `MainView` nicht erfahren kann, dass gerade eine Berechnung läuft.

Maßnahme Methode `isCalculated` zum `RouteModel` hinzufügen und verwenden; siehe auch Abschnitt ↗7.

Status behoben

/BG160/ Kartenausschnitt zentriert bei Profilwechsel

Symptom Der sichtbare Kartenausschnitt wird auf das Zentrum der Karte verschoben, obwohl man nur ein anderes Profil ausgewählt hat.

Ursache Der Kartenausschnitt wird zentriert, sobald sich die aktuelle `ProfileMapCombination` ändert.

Maßnahme Den Kartenausschnitt nur zentrieren, wenn sich die aktuelle Karte Ändert.

Status behoben

/BG170/ Textfelder für Start und Ziel beim Profil-/Kartenwechsel

Symptom Wird die Karte oder das Profil geändert, bleiben in den Textfeldern für Start und Ziel die alten Koordinaten stehen.

Ursache Die Route wird zwar zurückgesetzt, aber die Textfelder werden nicht geleert.

Maßnahme Die Textfelder leeren, sobald die Karte oder das Profil geändert werden.

Status behoben

3 Protokoll der Unit-Tests

/UT10/ **Was wird getestet** `GPXExporter`

Beschreibung Es wird die GPX-Export-Funktion des Programms getestet.

Bestanden, wenn die erzeugte Datei ein gültiges XML-Dokument ist.

Status bestanden

/UT20/ **Was wird getestet** `HTMLExporter`

Beschreibung Es wird die HTML-Export-Funktion des Programms getestet.

Bestanden, wenn die erzeugte Datei ein gültiges XML-Dokument ist.

Status bestanden

/UT30/ **Was wird getestet** `History`

Beschreibung Es wird die Speicher- und Lade-Funktion des Verlaufs getestet.

Bestanden, wenn der Verlauf nach dem Speichern und anschließenden Laden die gleichen Einträge enthält wie davor.

Status bestanden

/UT40/ **Was wird getestet** `HistoryEntry`

Beschreibung Es werden die Methoden `toString` und `fromString` der Verlaufseinträge getestet.

Bestanden, wenn ein Verlaufseintrag nach Anwenden von `fromString` auf den von `toString` erzeugten String denselben Start- und Zielpunkt sowie Zeitstempel enthält wie davor.

Status bestanden

/UT50/ **Was wird getestet** `EdgeProperties`

Beschreibung Es werden die Getter von `EdgeProperties` und insbesondere `getMaxSpeed` getestet.

Bestanden, wenn die Getter die korrekten Werte zurückgeben und `getMaxSpeed`, je nach Straßentyp und übergebenem Profil, die korrekte Höchstgeschwindigkeit des gegebenen Fahrzeugtyps auf diesem Straßentyp liefert.

Status bestanden

/UT60/ **Was wird getestet** `Restriction`

Beschreibung Es wird für verschiedene Beschränkungen (`Height-`, `Width-`, `Weight-`, `VehicleType-`, `Multiple-`, `NoRestriction`) die `allows`-Methode für die Profile „PKW (Standard)“ und „LKW (Standard)“ getestet.

Bestanden, wenn bei jeder getesteten Beschränkung `allows` für die jeweiligen Profile die korrekte Ausgabe liefert.

Status bestanden

/UT70/ **Was wird getestet** `EdgeBasedGraph`

Beschreibung Es wird die Speicher- und Lade-Funktion von `EdgeBasedGraph` getestet.

Bestanden, wenn der kantenbasierte Graph nach dem Speichern und anschließenden Laden die gleichen Kanten und Abbiegemöglichkeiten enthält wie davor.

Status bestanden

/UT80/ **Was wird getestet** `Graph`

Beschreibung Es werden die Getter sowie die Speicher- und Lade-Funktion von `Graph` getestet.

Bestanden, wenn die Getter die korrekten Werte zurückgeben und der Graph nach dem Speichern und anschließenden Laden die gleichen Knoten und Kanten enthält wie davor.

Status bestanden

/UT90/ **Was wird getestet** `GraphIndex`

Beschreibung Es werden `findNearestPointOnEdge` und `getEdgesInRectangle` der beiden `GraphIndex`-Implementierungen (`Array-` und `Tree-`) getestet.

Bestanden, wenn die resultierenden Kantenmengen und die `PointOnEdge`s jeweils mit den erwarteten Werten übereinstimmen.

Status bestanden

/UT100/ **Was wird getestet** `TileCache` Cache

Beschreibung Es wird das Speichern von Kartenkacheln im `TileCache` getestet.

Bestanden, wenn eine Kachel innerhalb von 100 ms nach der ersten Anfrage im `TileCache` gespeichert wird.

Status bestanden

/UT110/ **Was wird getestet** `TileCache` Prefetch

Beschreibung Es wird das Prefetching (vorsorgliche Berechnung von benachbarten Kacheln, auch wenn diese noch nicht direkt angefragt wurden) des `TileCache` getestet.

Bestanden, wenn nach dem Anfragen einer Kachel innerhalb von 100 ms benachbarte Kacheln ebenfalls im `TileCache` gespeichert werden.

Status bestanden

/UT120/ **Was wird getestet** `TileCache` Cleanup

Beschreibung Es wird das Entfernen von Kacheln aus dem `TileCache` getestet.

Bestanden, wenn nach dem Anfragen von 50^2 anderen Kacheln die älteste gespeicherte Kachel nicht mehr im `TileCache` vorhanden ist.

Status bestanden

/UT130/ **Was wird getestet** `ArcFlags`

Beschreibung Es wird die Speicher- und Lade-Funktion der Arc-Flags getestet.

Bestanden, wenn nach dem Speichern und anschließenden Laden die gleichen Arc-Flags gesetzt sind wie davor.

Status bestanden

/UT140/ **Was wird getestet** `RouteModel`

Beschreibung Es wird das ↗Beobachter-Muster der `RouteModel`-Klasse getestet.

Bestanden, wenn die von einem `TestListener` gemessene Anzahl an ausgelösten Ereignissen an allen Stellen mit der erwarteten Anzahl übereinstimmt.

Status bestanden

/UT150/ **Was wird getestet** `Weights`

Beschreibung Es wird die Speicher- und die Lade-Funktion der Kantengewichte getestet.

Bestanden, wenn nach dem Speichern und anschließenden Laden die gleichen Kantengewichte gesetzt sind wie davor.

Status bestanden

/UT160/ **Was wird getestet** `MapEdge`

Beschreibung Es wird die `MapEdge`-Klasse getestet, die vom OSM-Parser verwendet wird.

Bestanden, wenn das Setzen der ID korrekt funktioniert und die Getter die richtigen Werte zurückgeben.

Status bestanden

/UT170/ **Was wird getestet** `OSMParser`

Beschreibung Es wird die Verarbeitung verschiedener Karteneigenschaften (verschiedene Abbiegetypen, Einbahnstraßen, Ampeln, Kreisverkehre, ...) sowie das Erkennen von fehlerhaften Eingabedaten (Eigenschaften ohne Name, Eigenschaften ohne Wert, fehlende Koordinaten, ungültige Koordinaten, ...) durch den OSM-Parser getestet.

Bestanden, wenn alle gültigen Strukturen korrekt in den Graph übernommen werden und ungültige Daten vom Parser erkannt werden und der Import dann mit einer Ausnahme abgebrochen wird.

Status bestanden

/UT180/ **Was wird getestet** `OSMWay`

Beschreibung Es wird die Verarbeitung verschiedener OSM-Tags (Straßentyp, Kreisverkehr, ...) durch die `OSMWay`-Klasse getestet.

Bestanden, wenn alle OSM-Tags auf die richtigen Strukturen im Graph abgebildet werden.

Status bestanden

/UT190/ **Was wird getestet** `ProfileMapCombination`

Beschreibung Es wird die Speicher- und die Lade-Funktion von `ProfileMapCombination` getestet.

Bestanden, wenn die `ProfileMapCombination` nach dem Speichern und anschließenden Laden das gleiche Profil und die gleiche `StreetMap` sowie, falls vorhanden, die gleiche abgeschlossene Vorberechnung (Kantengewichte, Arc-Flags, benötigte Zeit) enthält wie davor.

Status bestanden

/UT200/ **Was wird getestet** `TurnRestriction`

Beschreibung Es wird die Funktionalität von Abbiegebeschränkungen getestet.

Bestanden, wenn eine Abbiegebeschränkung ihre Start- und Zielkante korrekt zurückgibt und `allowsTurn` korrekt entscheidet, ob ein Abbiegevorgang auf eine bestimmte Kante erlaubt ist oder nicht.

Status bestanden

/UT210/ **Was wird getestet** `Profile`

Beschreibung Es werden `equals` und `clone` sowie die Setter, Getter und die Speicher- und Lade-Funktionen von `Profile` getestet.

Bestanden, wenn `equals` und `clone` erwartungsgemäß funktionieren, die Setter die verschiedenen Werte (Fahrzeugtyp, Höhe, Breite, ...) korrekt setzen, die Werte von Standardprofilen jedoch nicht verändern, die Getter die korrekten Werte zurückgeben und das Profil nach dem Speichern und anschließenden Laden die gleichen Werte enthält wie davor.

Status bestanden

/UT220/ **Was wird getestet** `FibonacciHeap`

Beschreibung Es werden die verschiedenen Operationen (Einfügen/Entfernen von Elementen, Ändern der Priorität) des Fibonacci-Heaps getestet.

Bestanden, wenn die `deleteMin`-Methode nach dem Hinzufügen/Entfernen von Elementen und Ändern von Prioritäten immer korrekt das Element mit der niedrigsten Priorität zurückgibt.

Status bestanden

/UT230/ **Was wird getestet** `Coordinates`

Beschreibung Es werden die Methoden `distanceTo`, `angleBetween` sowie `toString`/`toSMT` und `fromString`/`fromSMT` getestet.

Bestanden, wenn `distanceTo` und `angleBetween` die erwarteten Werte zurückgeben und eine Transformation und Rücktransformation mit den Konvertierungsmethoden die Testkoordinaten unverändert lässt.

Status bestanden

/UT240/ **Was wird getestet** `PointOnEdge`

Beschreibung Es werden der Konstruktor sowie die Getter von `PointOnEdge` getestet.

Bestanden, wenn der Konstruktor nur positive Kanten-IDs sowie Positionen zwischen 0 und 1 annimmt und die Getter die korrekten Werte zurückgeben.

Status bestanden

/UT250/ **Was wird getestet** `FileUtil`

Beschreibung Es werden `checkMapName` und `checkProfileName` getestet.

Bestanden, wenn `checkMapName`/`checkProfileName` zulässige Karten-/Profilnamen akzeptiert und unzulässige ablehnt.

Status bestanden

4 Protokoll des GUI-Tests

4.1 Neue GUI-Tests

Alle Testfälle mit kleineren Nummern wurden im Pflichtenheft beschrieben. Bei allen hier beschriebenen Testfällen gilt als Vorbedingung, dass *routeKIT* gestartet ist.

/**TF410**/ Über-Fenster öffnen

Vorbedingungen Alle Fenster außer dem Hauptfenster sind geschlossen.

Ablauf „routeKIT/Über...“ in der Menüleiste wird ausgewählt.

Erwartetes Ergebnis Das Fenster mit den Informationen über das Programm erscheint.

/**TF420**/ Über-Fenster schließen

Vorbedingungen Das Über-Fenster ist geöffnet.

Ablauf Die Tastenkombination $Alt + F4$ drücken.

Erwartetes Ergebnis Das Über-Fenster wird geschlossen.

/**TF430**/ Koordinaten vertauschen

Vorbedingungen Das Start- und das Zielfeld enthalten gültige Koordinaten.

Ablauf Der Start/Ziel-Tauschen-Button wird gedrückt.

Erwartetes Ergebnis Die Eingaben von Start- und Zielfeld werden vertauscht und die Markierungen auf der Karte werden neu gesetzt. Die Route zwischen Start- und Zielpunkt wird berechnet.

/**TF440**/ Ungültige Koordinaten

Vorbedingungen Das Profil ist für die aktuelle Karte vorberechnet.

Ablauf Ins Start- oder Zielfeld wird eine ungültige Koordinate eingegeben.

Erwartetes Ergebnis Das Feld färbt sich rot. Es wird keine Routenberechnung durchgeführt.

/**TF450**/ Wegbeschreibung

Vorbedingungen Das Start- oder das Zielfeld enthält eine gültige Koordinate.

Ablauf Die fehlende gültige Koordinate wird eingegeben.

Erwartetes Ergebnis Die Markierungen auf der Karte wird gesetzt. Die Route zwischen Start- und Zielpunkt wird berechnet. Die Wegbeschreibung wird in das Wegbeschreibungsfeld ausgegeben.

/**TF460**/ Kartenverwaltung-Fenster öffnen

Vorbedingungen Alle Fenster außer dem Hauptfenster sind geschlossen.

Ablauf Der Karte-Button wird gedrückt.

Erwartetes Ergebnis Das Fenster der Kartenverwaltung erscheint.

/**TF470**/ Profilverwaltung-Fenster öffnen

Vorbedingungen Alle Fenster außer dem Hauptfenster sind geschlossen.

Ablauf Der Profil-Button wird gedrückt.

Erwartetes Ergebnis Das Fenster der Profilverwaltung erscheint.

/**TF480**/ Einbahnstraße benutzen

Vorbedingungen Karte ist sichtbar, Profil ist vorberechnet (für die aktuelle Karte), Karte enthält eine Einbahnstraße.

Ablauf Als Start- und Zielpunkt werden Punkte auf der Einbahnstraße gewählt. Dabei ist Ziel weiter in Einbahnstraßenrichtung als Start.

Erwartetes Ergebnis Eine Route, die den Einbahnstraßenabschnitt enthält, wird gefunden.

/**TF490**/ Profil hinzufügen

Vorbedingungen Die Kartenverwaltung ist offen.

Ablauf Der Hinzufügen-Button wird gedrückt. Ein Profil wird ausgewählt und die Wahl wird drei mal bestätigt.

Erwartetes Ergebnis Die Kartenverwaltung wird geschlossen. Das Profil wird für die Karte vorberechnet.

/**TF500**/ Verlauf-Fenster öffnen

Vorbedingungen Alle Fenster außer dem Hauptfenster sind geschlossen.

Ablauf Der Verlauf-Button wird gedrückt.

Erwartetes Ergebnis Das Verlauf-Fenster wird geöffnet.

/**TF510**/ Vorberechnung für Profil entfernen

Vorbedingungen Profil ist für die Karte vorberechnet. Kartenverwaltung ist offen.

Ablauf Ein Profil wird ausgewählt. Der Entfernen-Button wird gedrückt.

Erwartetes Ergebnis Die Vorberechnung von dem Profil wird gelöscht.

/**TF520**/ Keine ↗Arc-Flags verwenden

Vorbedingungen Das Profil ist für die Karte vorberechnet. Die ↗Arc-Flags sind aktiviert.

Ablauf In der Menüleiste werden die ↗Arc-Flags abgewählt.

Erwartetes Ergebnis Es werden keine ↗Arc-Flags bei der Routenberechnung verwendet.

/**TF530**/ ↗Arc-Flags verwenden

Vorbedingungen Das Profil ist für die Karte vorberechnet. Die ↗Arc-Flags sind deaktiviert.

Ablauf In der Menüleiste werden die ↗Arc-Flags gewählt.

Erwartetes Ergebnis Es werden ↗Arc-Flags bei der Routenberechnung verwendet.

4.2 Ablauf des automatisierten Tests

Startbedingung: Eine Internetverbindung ist hergestellt. Die aktuelle Konfiguration (auch alle Profile und Karten) ist gelöscht. Bei allen Tests außer sich selbst erfüllt „Start“ eine Vorbedingung.

ID	Testname	Testnummer	bestanden	Vorbedingungen erfüllt in
1	Start[1]	/TF30/	✓	[2]
2	Öffnung des Über-Fenster	↗/TF410/	✓	[2]
3	Schließen des Über-Fenster	↗/TF420/	✓	↗2
4	Eingabe der Koordinaten	/TF150/	✓	
5	Auswahl des ersten Punkts	/TF80/ /TF100/	✓	
6	Auswahl des zweiten Punkts	/TF90/ ↗/TF450/	✓	
7	GPX-Export	/TF70/	✓	↗6
8	HTML-Export	/TF20/	✓	↗6
9	Vertauschen von Eingaben	↗/TF430/	✓	↗6
10	OSM-Kacheln	/TF200/	✓	
11	Eigene Kacheln	/TF210/	✓	↗10
12	Reinzoomen	/TF120/ /TF220/	✓	
13	Rauszoomen	/TF130/ /TF220/	✓	
14	Karte ziehen	/TF110/	✓	
15	Deaktivierung der↗Arc-Flags	↗/TF520/	✓	
16	Vertauschen von Eingaben	↗/TF430/	✓	↗9
17	Aktivierung der ↗Arc-Flags	↗/TF530/	✓	↗15
18	Ungültige Koordinaten	↗/TF440//	✓	
19	Öffnung der Kartenverwaltung	↗/TF460/	✓	↗3
20	Import einer Karte	/TF290/ /TF330/	✓	↗19
21	Vorberechnung	/TF10/	✓	↗20
22	Öffnung der Profilverwaltung	↗/TF470/	✓	↗20 (schließen)
23	Auswahl eines Profils	/TF240/	✓	
24	Vorberechnung	/TF10/	✓	↗23 (schließen)
25	Einbahnstraße in Fahrtrichtung	↗/TF480/	✓	↗24
26	Einbahnstraße gegen die Fahrtrichtung	↗/TF430/ /TF350/ /TF340/	✓	↗25
27	Öffnung der Profilverwaltung	↗/TF470/	✓	↗20 (schließen)
28	Neues Profil	/TF230/	✓	
29	Öffnung der Kartenverwaltung	↗/TF460/	✓	↗28 (schließen)
30	Hinzufügen von Profil zur Karte	↗/TF490/	✓	↗29
31	Öffnung des Verlaufs	↗/TF500/	✓	↗30 (schließen)
32	Laden aus Verlauf	/TF50/	✓	↗26
33	Öffnung der Kartenverwaltung	↗/TF460/	✓	↗32 (schließen)
34	Entfernung einer Vorberechnung für Profil	↗/TF510/	✓	↗33 und ↗24
35	Entfernung eines Profils	/TF250/	✓	↗28

[1] Die Konfiguration wird neu vom Server heruntergeladen. Damit wird gesichert, dass der Test mit der Karte „Regierungsbezirk Karlsruhe“ und dem dafür vorberechneten Standardprofil „PKW (Standard)“ gestartet wird, eigene Kacheln verwendet werden und die ↗Arc-Flags aktiviert sind.

[2] Neustart der Anwendung

36	Entfernung einer Karte	/TF310/	✓	↗20

Tabelle 1: Der Ablauf des automatisierten GUI Tests

4.3 Manuelle Tests

Die Tests /TF40/, /TF60/, /TF140/, /TF160/, /TF170/, /TF 180/, /TF190/, /TF260/, /TF270/, /TF280/, /TF300/, /TF320/ aus dem Pflichtenheft wurden manuell auf dem Commit `3c2b61df374375029a4fe853dce074f05488f9b6` am 07. 03. 2014 ausgeführt und wurden bestanden.
Die Tests in Tabelle ↗2 wurden auf dem Commit `680d84bff40fce53b812-6ed23e148572ca364474` am 07. 03. 2014 für die Karte „Regierungsbezirk Karlsruhe“ ausgeführt.

Testnummer	bestanden	Startkoordinate Zielkoordinate	Begründung
/TF360/	✓	49.007614 8.394907 49.00796 8.394949	links abbiegen verboten
/TF370/	✓	49.00889 8.395014 49.00988 8.3951645	Beschränkung nach Höhe (3,5 m), Profil LKW (Standard)
/TF380/	✓	49.00889 8.395014 49.00988 8.3951645	Beschränkung nach Höhe (3,5 m), Profil PKW (Standard)
/TF390/	✓	48.964893 8.613882 48.965534 8.613367	LKW-Verbot, Profil LKW (Standard)
/TF390/	✓	49.43081 8.691645 49.430744 8.692374	Bus-Verbot, erstelltes Profil Bus
/TF390/	✓	49.392246 8.783075 49.392353 8.784149	Motorrad-Verbot, erstelltes Profil Motorrad
/TF400/	✓	48.964893 8.613882 48.965534 8.613367	LKW-Verbot, Profil PKW (Standard)

Tabelle 2: Die manuell auf der einer Karte ausgeführten Tests.

Somit wurden alle Test aus dem Pflichtenheft (von /TF10/ bis /TF400/) durchgeführt und bestanden.

5 Testüberdeckung

Die Abdeckung des Quellcodes im "src"-Ordner (also ohne Testfälle) während dem Commit `6d239a2ab7402c7a469b48299c42e8d56f547884` ist in Tabelle ↗3.

Testteil	Instruction Coverage	Line coverage	Branch coverage
RobotUI	78.1%	79.8%	56.9%
Unit Tests	51.3%	51.1%	44.9%
Merged	87.0%	89.0%	66.3%

Tabelle 3: Die Quellcodeabdeckung der Tests nach verschiedenen Kriterien

Eine Aufteilung der "Instruction Coverage" des vereinigten Durchlaufs in die einzelnen Teilpakete ist in Abbildung ↗1 dargestellt.

routeKIT	85,9 %	30.620	5.020	35.640
src	87,0 %	24.974	3.719	28.693
edu.kit.pse.ws2013.routekit.map	85,1 %	5.529	968	6.497
edu.kit.pse.ws2013.routekit.controllers	82,7 %	4.490	939	5.429
ManagementActions.java	67,4 %	458	222	680
ProfileMapManager.java	78,9 %	763	204	967
MapManagerController.java	82,5 %	674	143	817
ProfileManagerController.java	85,0 %	533	94	627
CLI.java	89,4 %	630	75	705
MainController.java	90,6 %	627	65	692
ProfileManager.java	81,2 %	220	51	271
MapManager.java	80,5 %	198	48	246
TerminalCLI.java	88,1 %	275	37	312
EasterEggCLI.java	100,0 %	112	0	112
edu.kit.pse.ws2013.routekit.util	66,9 %	762	377	1.139
edu.kit.pse.ws2013.routekit.models	76,7 %	1.196	364	1.560
edu.kit.pse.ws2013.routekit.views	94,8 %	6.056	331	6.387
edu.kit.pse.ws2013.routekit.precalculatio	91,8 %	3.038	271	3.309
edu.kit.pse.ws2013.routekit.profiles	78,0 %	556	157	713
edu.kit.pse.ws2013.routekit.routecalculat	91,1 %	1.468	143	1.611
edu.kit.pse.ws2013.routekit.history	71,5 %	216	86	302
edu.kit.pse.ws2013.routekit.export	85,6 %	298	50	348
edu.kit.pse.ws2013.routekit.mapdisplay	97,8 %	1.362	30	1.392
(default package)	50,0 %	3	3	6

Abbildung 1: Die Instruction Coverage des gesamten Testlaufs.

Die teilweise eher niedrige Abdeckung der IO-lastigen Klassen soll hier anhand eines Ausschnittes aus `ManagementActions` (Abbildung ↗2) erklärt werden.

```
216                     calculator.doPrecalculation(combination, reporter);
217                 } catch (Exception e) {
218                     e.printStackTrace();
219                     continue;
220                 } finally {
221                     reporter.popTask("Führe Vorberechnung durch für '"
222                             + combination + "'");
223                 }
224                 reporter.pushTask("Speichere '" + combination + "'");
225                 try {
226                     profileMapManager.savePrecalculation(combination);
227                 } catch (Exception e) {
228                     e.printStackTrace();
229                     continue;
230                 } finally {
231                     reporter.popTask("Speichere '" + combination + "'");
232                 }
233             } finally {
234                 reporter.popTask("Führe Vorberechnung durch und speichere '"
235                         + combination.toString() + "'");
```

Abbildung 2: Ein Codeausschnitt aus `ManagementActions` mit der Abdeckung des gesamten Testlaufs.

Hier sieht man deutlich, dass die fehlende Abdeckung durch IO-Fehlerbehandlungscode verursacht wird. Da sehr viele Dateien in verschiedensten Situationen gespeichert und geladen werden müssen, haben wir keine Testfälle geschrieben, die explizit hier Dateien entfernen oder die Festplatte komplett füllen, sodass keine Datei mehr angelegt werden kann.

Diese Problematik wird an einer anderen Stelle, bei der Benutzung des Java-Konstrukts "try-with-Resources" noch deutlicher, wie man in Abbildung ↗3 erkennen kann.

```
47    public void save(File f) throws IOException {
48        try (RandomAccessFile raf = new RandomAccessFile(f, "rw");
49                FileChannel fc = raf.getChannel()) {
50            MappedByteBuffer mbb = fc.map(MapMode.READ_WRITE, 0,
51                    (1 + flags.length) * 4);
52            mbb.order(ByteOrder.BIG_ENDIAN).asIntBuffer().put(flags.length)
53                    .put(flags);
54            mbb.force();
55        }
56    }
57
58    public static ArcFlags load(File f) throws IOException {
59        try (FileInputStream fis = new FileInputStream(f);
60                DataInputStream dis = new DataInputStream(fis);
61                FileChannel fc = fis.getChannel()) {
62            int length = dis.readInt();
63            MappedByteBuffer mbb = fc.map(MapMode.READ_ONLY, 4, length * 4);
64            int[] flags = new int[length];
65            mbb.order(ByteOrder.BIG_ENDIAN).asIntBuffer().get(flags);
66            return new ArcFlags(flags);
67        }
```

Problems | Javadoc | Declaration | Console | History | Coverage | Call Hierarchy | Debug

Merged (07.03.2014 20:25:51)

Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
edu.kit.pse.ws2013.routekit.map	85,1 %	5.529	968	6.497
edu.kit.pse.ws2013.routekit.controllers	82,7 %	4.490	939	5.429
edu.kit.pse.ws2013.routekit.util	66,9 %	762	377	1.139
edu.kit.pse.ws2013.routekit.models	76,7 %	1.196	364	1.560
ProfileMapCombination.java	80,0 %	479	120	599
ArcFlags.java	51,3 %	119	113	232
ArcFlags	51,3 %	119	113	232
load(File)	49,6 %	56	57	113
save(File)	57,1 %	52	39	91
equals(Object)	0,0 %	0	12	12
hashCode()	0,0 %	0	5	5
ArcFlags(int[])	100,0 %	6	0	6
getFlag(int)	100,0 %	5	0	5
Weights.java	51,3 %	119	113	232
ProgressReporter.java	95,6 %	394	18	412
RouteModel.java	100,0 %	85	0	85
edu.kit.pse.ws2013.routekit.views	94,8 %	6.056	331	6.387
edu.kit.pse.ws2013.routekit.precalculatio	91,8 %	3.038	271	3.309
edu.kit.pse.ws2013.routekit.profiles	78,0 %	556	157	713
edu.kit.pse.ws2013.routekit.routecalculat	91,1 %	1.468	143	1.611
edu.kit.pse.ws2013.routekit.history	71,5 %	216	86	302
edu.kit.pse.ws2013.routekit.export	85,6 %	298	50	348
edu.kit.pse.ws2013.routekit.mapdisplay	97,8 %	1.362	30	1.392
(default package)	50,0 %	3	3	6

Abbildung 3: Ein Codeausschnitt aus `ArcFlags` mit der Abdeckung des gesamten Testlaufs.

Man kann erkennen, dass die nicht überdeckten Anweisungen dieser 2 Methoden in den try-Blöcken liegen. Es sind die Anweisungen, die dafür sorgen, dass die Ressourcen (`RandomAccessFile`, `FileChannel`, `FileInputStream`, `DataInputStream`) auch im Fehlerfall ordnungsgemäß geschlossen werden (und auch, dass die anderen Ressourcen geschlossen werden, wenn das Schließen einer anderen fehlschlägt). So sind allein im sichtbaren Programmcode 96 Befehle, die nicht überdeckt sind.

6 Statistik

Das System enthielt zum Commit `6d239a2ab7402c7a469b48299c42e8d56f547884` (07. 03. 2014) die in Tabelle ↗4 ersichtliche Anzahl an Zeilen.

	Quelltext	Testcode	Gesamt
alle Zeilen	14514	2767	17281
ohne Leerzeilen	13201	2363	15562
ohne leere Zeilen, imports und Kommentare	7388	1816	9204

Tabelle 4: Codestatistik. Mit `qualitätssicherung/stats.sh` erstellt.

Für die erste Zahl wurden einfach alle Zeilen gezählt. Wenn die Datei dabei mit einem Zeilenumbruch endet, wird dies nicht als zusätzliche Zeile gewertet.

Für die zweite wurden alle Leerzeilen entfernt.

Für die dritte wurden zudem alle import-Zeilen, alle Kommentare und alle Zeilen, die nur eine schließende geschweifte Klammer enthalten, entfernt.

7 Änderungen seit der Implementierung

Im Folgenden sind die gegenüber der Implementierung geänderten und neu hinzugefügten Klassen, Methoden und Attribute aufgeführt.

- `GraphIndex`
  - Die Klasse `GraphIndex` wurde in ein Interface geändert und die bisherige Implementierung in die Klasse `TreeGraphIndex` verschoben.
  - Eine weitere Implementierung von `GraphIndex`, `ArrayGraphIndex`, wurde geschrieben. Sie verwendet nur zwei `int`-Arrays und ist dadurch etwas schneller und lässt sich gut speichern/laden.
  - `Graph` verwendet `ArrayGraphIndex`.
  - `Graph` lädt `ArrayGraphIndex` von der Festplatte, statt sie neu aufzubauen, wenn die Dateien vorhanden sind. Dies beschleunigt den Start der Anwendung erheblich.
- ↗/BG150/
  - `RouteModel` erhält die neuen Methoden `startCalculating`, um den Beginn einer Routenberechnung zu melden, und `isCalculating`, um abzufragen, ob aktuell eine Routenberechnung stattfindet. (Eine Routenberechnung endet implizit bei Aufruf von `setCurrentRoute`.)
- `MainController`
  - Die Klasse `MainController` enthält nun eine Methode `setUseArcFlags` zum Abschalten der ↗Arc-Flags bei der Routenberechnung. Hierzu wird entweder eine Instanz von `ArcFlagsDijkstra` oder `Dijkstra` zur Routenberechnung verwendet.
- `ArcFlagsDijkstra`

– Die Klasse `ArcFlagsDijkstra` erbt nun von der neuen Klasse `Dijkstra`, welche den normalen ↗Dijkstra's Algorithmus implementiert und ergänzt diese mit den ↗Arc-Flags.