

Qualitätssicherung

ROSE (Road System Editor)
Edit Highway Roads Easily



Planung: Jannes Wagner
Entwurf: Yannik Sproll
Implementierung: Philipp Seidel
Qualitätssicherung: Cristian Gorun
Abnahme: Max Schweikart

Betreuung: Erik Burger, Jelle Kübler und Marvin Baumann

Institut für Verkehrswesen
Fakultät für Bauingenieur-, Geo- und Umweltwissenschaften
Karlsruher Institut für Technologie

März 2022

Inhaltsverzeichnis

1 Einleitung	5
2 Behobene Bugs	6
2.1 RoseSortedBox kann durch Iterator geändert werden	6
2.2 Group kann durch Iterator unbemerkt geändert werden	6
2.3 Der Vergleich von HighwaySegments mit anderen Arten von Segmenten liefert eine falsche Sortierung	7
2.4 Violations bleiben bestehen nachdem die dazugehörigen Segmente gelöscht wurden	8
2.5 SelectableAttributes zeigen einen Platzhalter anstelle des anzuseigenden Wertes an	9
2.6 SpeedLimits unterstützen nur Integerwerte nicht aber "Tunnel", "SBA" und "Ohne"	9
2.7 ComboBoxen werden nicht übersetzt wenn die Anzeigesprache geändert wird	10
2.8 Fehlender Editor für Gruppennamen	10
2.9 Suchfunktion im HierarchyPanel funktioniert nicht	11
2.10 Selektion von Segmenten wird im HierarchyPanel nicht richtig angezeigt.	11
2.11 ConnectionViews werden beim Redo falsch gezeichnet	12
2.12 Connections werden in Commands nicht für Undo/Redo zwischengespeichert	13
2.13 Kompatibilitätskriterien erzeugen jeden Verstoß zweimal	13
2.14 Attribut "Längsneigung" ist mit falschem Datentyp gespeichert	15
2.15 Zuletzt geöffnete Projekte werden mehrmals gespeichert	16
2.16 Eingabefelder für INTEGER und FRACTIONAL-Attribute können nicht geleert werde	16
2.17 Falsche Fehlermeldungen für gescheiterten import und export von Kriterien .	17
2.18 Das CriteriaWindow öffnet sich im Hintergrund	18
2.19 Draggen von ElementViews im HierarchyPanel erzeugt inkonsistente View-Zustände	18
3 Nicht behobene Bugs	19
3.1 Richtungspfeil wird auf Basissegmenten manchmal nicht gezeichnet	19
3.2 Attributeditor skaliert mit dem Grid	19
3.3 Probleme mit dem verbinden von Connectoren	20
3.4 SegmentViews können sich gegenseitig überdecken	20
3.5 Gelegentlich auftretende NullPointerExceptions beim Verbinden von Straßensegmenten	21
3.6 JavaFx und die Weiterleitung von Events	22
3.7 Schnelles bewegen von Segmenten bewegt den Hintergrund	22
3.8 JavaFx und das Auslösen von DragEvents	23
4 Modultests	24

5 Globale Testfälle	26
5.1 Programm ausführen	26
5.2 Straßensegment platzieren (Doppelklick)	26
5.3 Straßensegment platzieren (Drag and Drop)	26
5.4 Attribute eintragen	26
5.5 Straßensegmente verbinden (Basisstraßensegment)	27
5.6 Das Projekt speichern	27
5.7 Das Straßennetz in das FREEVAL-Format exportieren	27
5.8 Das Straßennetz in das SUMO-Format exportieren	27
5.9 Selektierte Straßensegmente verschieben	27
5.10 Straßensegmente verbinden (Einfahrt)	28
5.11 Duplizieren von Straßensegmenten	28
5.12 Undo und Redo	28
5.13 20 mal rückgängig machen und wiederherstellen	29
5.14 Projekt laden	29
5.15 Sprache zur Laufzeit einstellen	29
5.16 Vollständigkeitskriterien prüfen	29
5.17 Kompatibilitätskriterien auswerten	29
5.18 Straßensegment löschen	30
5.19 Bulk-Edit	30
5.20 Anzahl der Fahrstreifen ändern	30
5.21 Attribute auf der Editorfläche anzeigen	30
5.22 Undo durch Tastenkombination bedienen	30
5.23 Kompatibilitätskriterium bearbeiten	30
5.24 Plausibilitätskriterien ex- und importieren	31
5.25 Zur Fehlerquelle springen	31
5.26 Sicherungskopien	31
5.27 Tastenkombinationen einsehen	31
5.28 Elemente suchen	31
5.29 Länge und Anzahl von Messwert-Zeitintervallen konfigurieren	31
5.30 Straßensegment drehen	32
5.31 Zoomen in der Editorfläche	32
5.32 Gruppieren von Straßensegmenten	32
5.33 Messwerte eintragen	32
5.34 Element in andere Gruppe verschieben	32
6 Optimierungen	33
6.1 Bulk edit	33
6.2 Die Richtung von Segmenten wird überprüft	33
6.3 Beim Einfügen eines Straßensegmentes in die Editorfläche per DragAndDrop wird das Segment an der Mausposition eingefügt	34
6.4 Straßensegmente zeigen Richtungspfeile an	34
6.5 Fehlermeldungen beim Import- und Export von Plausibilitätskriterien	35
6.6 Root-Group für das RoadSystem	35

7 Nutzertests	37
7.1 Einstieg	37
7.2 Erstellung eines Straßennetzes	38
7.3 Bulk Edit	38
7.4 Speichern, Exportieren und Laden	39
7.5 Fehler suchen und korrigieren	39
7.6 Allgemeines Feedback	40
7.7 System Usability Scale	40
8 Entwurfsänderungen bezüglich des Stands der Implementierungsphase	42
8.0.1 Neuer Konstruktor für RoseProjectController	42
8.0.2 Verwendung einer Logging-Bibliothek	42
8.0.3 Objekt-Datentypen statt primitiven Datentypen für Attribute	42
8.0.4 Neue Methode getIsSegmentSelected für HierarchyController	43
8.1 Neue Methode getRoadSystem für RoadSystemPanel	43
8.2 Neue Methode getZoomSetting für RoadSystemPanel	43
8.2.1 Neuer Konstruktorparameter für SegmentView	43
8.2.2 Neuer Konstruktorparameter für ElementTreeCell	43
8.2.3 Neuer Konstruktorparameter für DragSegmentEndCommand	43
8.2.4 Neuer Konstruktorparameter für DragStreetSegmentCommand	44
8.2.5 Neue Klasse LocalizedComboBox hinzugefügt	44
8.2.6 Methode getRootElements durch Methode getRootGroup ersetzt in RoadSystem	44
8.2.7 Neuer Konstruktorparameter für CreateStreetSegmentCommand	44
9 Codequalität	45
9.1 Reliability	45
9.2 Security	45
9.3 Security Review	45
9.4 Maintainability	46
9.5 Kennzahlen für Codequalität	46
Glossar	47

1 Einleitung

In diesem Dokument halten wir unsere Strategie zur Verbesserung der Qualität von "ROSE" fest. Das umfasst das Vorgehen bei Unit-Tests und GUI-Tests, sowie die dabei angestrebten Ziele. Dabei erörtern wir auch in wieweit wir diese Ziele erreicht haben.

Des Weiteren dokumentieren wir hier alle gefundenen Bugs. Dazu zählen die behobenen Bugs, sowie die nicht behobenen Bugs. Auch Bugs, die wir bereits während der Implementierungsphase gefunden haben werden hier aufgeführt. Für jeden Bug beschreiben wir dessen Wirkung, analysieren die Ursache und stellen in den meisten Fällen einen Lösungsansatz vor.

Eine Beschreibung, sowie unsere Auswertung der von uns durchgeführten Nutzertests werden hier auch vorgestellt. Darin analysieren wir das Feedback unserer Tester und stellen die Methodik unseres Nutzertests vor.

Auch vorgenommene Optimierungen am Code und an anderen Aspekten, wie z.B. der Benutzbarkeit von "ROSE" beschreiben wir hier.

Alle Änderungen am Entwurf, die wir im Zuge der Bugfixes und Optimierungen vorgenommen haben, dokumentieren wir hier ebenfalls. Diese Änderungen werden, analog zum Implementierungsbericht, in einer kurzen textuellen Beschreibung festgehalten.

Das hier beschriebene Vorgehen zur Qualitätssicherung bezieht sich auf die im "ROSE" Pflichtenheft spezifizierten Maßnahmen. Das gilt insbesondere für GUI-Tests, welche die globalen Testfälle abdecken sollen, sowie für Nutzertests.

2 Behobene Bugs

In diesem Abschnitt werden alle behobenen Bugs dokumentiert. Die Dokumentation eines Bugs umfasst eine Beschreibung der Wirkung des Bugs und eine Analyse der Ursache des Bugs. Ebenso wird beschrieben, durch welche Änderungen das Problem behoben wurde und wie sicher gestellt wurde dass das Problem tatsächlich gelöst wurde. Zusätzlich wird festgehalten wie der Bug entdeckt wurde.

2.1 RoseSortedBox kann durch Iterator geändert werden

Beschreibung: Die Schnittstelle `SortedBox<T>` spezifiziert eine geordnete Datenstruktur, die nach ihrer Erzeugung nicht mehr geändert werden kann, d.h. es sollen keine Elemente mehr hinzugefügt oder entfernt werden können. Die Methode `RoseSortedBox<T>#iterator()` gibt einen `Iterator<T>` zurück, über deren `remove()`-Methode Elemente der sortierten Box entfernt werden können.

Analyse: Die Implementierung `RoseSortedBox<T>` setzt die Speicherung der enthaltenen Elemente als Unterklasse von `RoseBox<T>` mithilfe einer versteckten `List<T>` um. `RoseSortedBox<T>#iterator()` gibt dabei den modifizierbaren Iterator der Liste zurück, obwohl die Implementierung von `RoseBox<T>#iterator()` bereits einen schreibgeschützten `Iterator<T>` zurückgibt.

Lösung: Die Implementierung von `RoseSortedBox<T>#iterator()` wurde entfernt, so dass wieder der unmodifizierbare Iterator der Oberklasse zurückgegeben wird.

Getestet durch:

- Der Modultest `RoseSortedBoxTest#testIteratorDoesNotSupportRemove()` prüft exemplarisch, dass bei der Ausführung von `remove()` auf `RoseBox<T>#iterator()` eine Ausnahme erzeugt wird und kein Element aus der Box entfernt wurde.

Entdeckung: Dieses Fehlverhalten wurde beim Schreiben von Modultests zur Erhöhung der Zeilenüberdeckung entdeckt (siehe 4).

2.2 Group kann durch Iterator unbemerkt geändert werden

Beschreibung: Gruppen (`Group`) können weitere Elemente (`Element`) enthalten. Der Inhalt einer Gruppe kann über die Methoden `addElement(Element)` und `removeElement(Element)` geändert und über die Methoden `getElements()` und `iterator()` ausgelesen

werden. Die Methode `iterator()` gibt dabei einen `Iterator<Element>` zurück, über deren `remove()`-Methode Elemente der der Gruppe entfernt werden können. Über diese Entfernung werden die `SetObserver<Element, Element>`, die die Gruppe beobachten, jedoch nicht benachrichtigt.

Analyse: Die Implementierung von `Group` setzt die Speicherung der enthaltenen Elemente mithilfe einer versteckten `List<Element>` um. `Group#iterator()` gibt dabei den modifizierbaren Iterator der versteckten Liste zurück. Ein Aufruf von `remove()` auf dem `Iterator<Element>` entfernt direkt ein Element aus der versteckten Liste, ohne dass die Gruppe etwas davon mitbekommt. Daher löst die Gruppe auch kein Benachrichtigungen der Form `notifyRemoval(Element)` an ihre Beobachter aus.

Lösung: `Group#iterator()` gibt nicht mehr den Iterator der versteckten Liste zurück, sondern den Iterator von `getElements()`. Da `getElements()` bereits eine schreibgeschützte Kopie der enthaltenen Elemente zurückgibt, ist auch deren Iterator schreibgeschützt.

Getestet durch:

- Der Modultest `GroupTest#testIterator()` prüft exemplarisch, dass bei der Ausführung von `remove()` auf `Group#iterator()` eine Ausnahme erzeugt wird und kein Element aus der Gruppe entfernt wurde.

Entdeckung: Dieses Fehlverhalten wurde beim Schreiben von Modultests zur Erhöhung der Zeilenüberdeckung entdeckt (siehe 4).

2.3 Der Vergleich von HighwaySegments mit anderen Arten von Segmenten liefert eine falsche Sortierung

Beschreibung: Um eine deterministische Zeichenreihenfolge im View-Subsystem zu ermöglichen, erbt die `Segment`-Schnittstelle von `Comparable<Segment>`. `HighwaySegment` ist eine Oberklasse von allen aktuell unterstützten Segmenten, es könnte aber in einer Erweiterung der Anwendung weitere Implementierungen von `Segment` geben, die nicht von `HighwaySegment` erben. Wie der Vergleich von anderen Segmenten mit `HighwaySegments` implementiert wird, wird dabei den anderen Implementierungen von `Segment` überlassen. Vergleicht man nun ein `HighwaySegment` mit einem anderen `Segment`, gibt `HighwaySegment#compareTo(Segment)` einen falschen Vergleichswert zurück, der nicht der vom anderen `Segment` spezifizierten Sortierung entspricht.

Analyse: Die Implementierung `HighwaySegment#compareTo(Segment)` berücksichtigt zwar die Sortierung, die vom anderen Segment vorgegeben wird, jedoch wird diese mit dem falschen Vorzeichen übernommen. Für zwei vergleichbare Java-Objekte sollte bei

einer ordentlichen Implementierung gelten: `object1.compareTo(object2) == -object2.compareTo(object1)`

Lösung: Das fehlende Vorzeichen wird ergänzt.

Getestet durch:

- Der Modultest `HighwaySegmentTest#testCompareToNonHighwaySegment()` prüft exemplarisch mithilfe eines *Mocks*, dass `HighwaySegment#compareTo(Segment)` die Sortierung von anderen `Segment`-Implementierungen umgekehrt übernimmt.
- Der Modultest `HighwaySegmentTest#testCompareToHighwaySegment()` prüft exemplarisch, dass ein Vergleich zwischen zwei `HighwaySegments` nach wie vor aus der Erstellungsreihenfolge der Objekte hervorgeht.

Entdeckung: Dieses Fehlverhalten wurde beim Schreiben von Modultests zur Erhöhung der Zeilenüberdeckung entdeckt (siehe 4).

2.4 Violations bleiben bestehen nachdem die dazugehörigen Segmente gelöscht wurden

Beschreibung: `PlausibilityCriterion` sollten bemerken wenn nicht mehr gegen sie verstößen wird. Dies ist auch der Fall wenn ein Segment, das für einen Verstoß verantwortlich war entfernt wurde. Beim entfernen eines Segments bleiben allerdings die `Violations` bestehen.

Analyse: Weder `Violation` noch `PlausibilityCriterion` kann das `RoadSystem` beobachten. Die Logik zum entfernen eines Segments ist in der `notifyRemoval` Methode des `CompatibilityCriterion` implementiert, diese wird aber nur aufgerufen wenn ein Segment aus einer `Group` entfernt wird und nicht wenn dieses gelöscht wird.

Lösung: Das `GraphRoadSystem` benachrichtigt beim entfernen von `Segmenten` deren `Subscriber`. `AbstractCriterion` überprüft jetzt in jedem `notifyChange()` Aufruf, ob das betroffene `Segment` noch Teil des `RoadSystems` ist und entfernt, falls nicht, alle `Violations` die das `Segment` betreffen.

Entdeckung: Dieses Fehlverhalten wurde beim Ausführen beispielhafter Anwendungsfälle entdeckt.

2.5 SelectableAttributes zeigen einen Platzhalter anstelle des anzugegenden Wertes an

Beschreibung: Wenn der `AttributeEditor` eines `Segments` geöffnet wird zeigen `SelectableAttribute`s immer einen Platzhalter “_” an, auch wenn ein Wert gesetzt wurde.

Analyse: `SelectableAttribute` bezieht an keiner Stelle den aktuellen Wert aus dem `model` und setzt deswegen immer den standard Platzhalter bis ein neuer Wert ausgewählt wird. Die Werte im `model` sind korrekt, werden aber nicht in die `view` übertragen.

Lösung: Es wurde eine neue `selectWithoutListener(T)` Methode hinzugefügt die genutzt werden kann um das ausgewählte `Attribute` in der `LocalizedComboBox` zu ändern ohne wieder eine Änderung im `model` zu verursachen. Diese Methode wird nun im Konstruktor und in der `notifyChange()` Methode genutzt um den aktuellen stand des `models` in die `view` zu übertragen.

Getestet durch: Da es sich hierbei um eine Änderung handelt die nur in `view` wirksam ist, wird dies nicht durch Modultests abgedeckt.

Entdeckung: Dieses Fehlverhalten wurde beim ausführen beispielhafter Anwendungsfälle entdeckt.

2.6 SpeedLimits unterstützen nur Integerwerte nicht aber “Tunnel”, “SBA” und “Ohne”

Beschreibung: Beim auswählen von `SpeedLimit` als `Attribute` für ein `CompatibilityCriterion` kommt es zu Fehlern und es findet keine sinnvolle Validierung statt.

Analyse: `SpeedLimit` wurde früher als Integer modelliert. Wie später bemerkt wurde müssen aber auch “Tunnel”, “SBA” und “Ohne” als gültige Werte akzeptiert werden. Dies erfordert die Ummodellierung zu einem Enum und eine neue Möglichkeit diese zu validieren. Es wurde nur die Ummodelierung zum Enum vorgenommen wodurch es zu Fehlern kommt, falls `SpeedLimit` als `Attribute` ausgewählt wird.

Lösung: `SpeedLimits` enthalten nun zusätzlich einen Integerwert der, falls gültig, ihre Geschwindigkeit repräsentiert. Es wurde ein neuer `ValidationType LESS_THAN_SPEED_LIMIT` hinzugefügt der in der Lage ist diesen Wert auszulesen und zu vergleichen.

Getestet durch: Im Modultest `SpeedlimitLessThanTest` wird die neue `ValidationStrategy` mit und ohne Diskrepanz getestet.

Entdeckung: Dieses Fehlverhalten wurde beim Ausführen beispielhafter Anwendungsfälle entdeckt.

2.7 ComboBoxen werden nicht übersetzt wenn die Anzeigesprache geändert wird

Beschreibung: Die im Attributeditor und im Kriterieneditor verwendeten ComboBoxen übersetzen ihre angezeigten Inhalte beim ändern der Annzeigesprache nicht. Das bedeutet, dass weder der Text der selektierten Option noch die Einträge der Auswahlliste nach dem Ändern der Sprache mitübersetzt werden.

Analyse: ComboBoxen verwenden ListCells um die selektierte Option und die Einträge der Auswahlliste darzustellen. Eine ListCell gehört immer zu einer Option und enthält den zugehörigen Text. Diese ListCells sind außerhalb der ComboBox nicht zugreifbar. Deshalb kann der Text der ListCells beim Ändern der Anzeigesprache nicht neu gesetzt werden.

Lösung: Die ComboBoxen verwenden eine modifizierbare Fabrikmethode, um ListCells zu erzeugen. Diese Fabrikmethode wurde so verändert, dass ListCells auch außerhalb der ComboBoxen gespeichert werden. Beim Ändern der Anzeigesprache wird dann auf die gespeicherten ListCells zugegriffen und ihr Text wird modifiziert.

Getestet durch:

- Das korrekte Übersetzen der ComboBoxen wird durch eine manuelle Verifikation zur Laufzeit getestet.

Entdeckung: Dieses Fehlverhalten wurde während einer Fehlersuche zur Laufzeit des Programms entdeckt.

2.8 Fehlender Editor für Gruppennamen

Beschreibung: Im HierarchyPanel sollte durch einen Doppelklick auf eine GroupView der angezeigte Gruppenname editierbar werden.

Analyse: Da die GroupView im HierarchyPanel auf einer ElementTreeCell liegt, wird der Doppelklick abgefangen und öffnet stattdessen Unterelemente der TreeView.

Lösung: Da der Doppelklick nicht verfügbar ist, öffnet sich bei einem Rechtsclick auf die GroupView ein Kontextmenü. In diesem Kontextmenü kann der Nutzer auswählen, dass

er den Gruppennamen bearbeiten möchte. Der Editor für den Gruppennamen öffnet sich in einem Dialogfenster.

Getestet durch:

- Das korrekte Auftauchen des Editors für Gruppennamen wird durch eine manuelle Verifikation zur Laufzeit getestet.

Entdeckung: Dieses Fehlverhalten wurde während einer Fehlersuche zur Laufzeit des Programms entdeckt.

2.9 Suchfunktion im HierarchyPanel funktioniert nicht

Beschreibung: Bei einer Eingabe in die `SearchBar` werden nicht alle `Elemente` ausgeblendet, von deren Name die Eingabe kein Präfix ist.

Analyse: Die Kindelemente in der `ElementTreeItems` im `TreeView` des `HierarchyPanel`s werden in die Collection `ElementTreeItem#getChildren`, anstatt an die Collection `ElementTreeItem#getInternalChildren` eingefügt. Da die Collection `ElementTreeItem #getChildren` nicht filterbar ist, werden entsprechende `Elemente` nicht ausgeblendet.

Lösung: Die Kindelemente werden in die Collection `ElementTreeItem#getInternalChildren` eingefügt

Getestet durch:

- Das korrekte Ausblenden der `ElementTreeItems` wird durch eine manuelle Verifikation zur Laufzeit getestet.

Entdeckung: Dieses Fehlverhalten wurde während einer Fehlersuche zur Laufzeit des Programms entdeckt.

2.10 Selektion von Segmenten wird im HierarchyPanel nicht richtig angezeigt.

Beschreibung: Wenn eine `ElementTreeCell` eines Segments im `HierarchyPanel` ausgeblendet und wieder eingeblendet wird, wird ihr Selektionsstatus nicht mehr korrekt angezeigt.

Analyse: Beim Wiedereinblenden einer `ElementTreeCell` eines `Segments` im `HierarchyPanel` wird die `SegmentView` der `ElementTreeCell` neu erzeugt. Der Selektionsstatus wird der `SegmentView` ausschließlich über Observer-Callbacks der Controller mitgeteilt. Ein solcher Callback wird beim Neuerzeugen der `SegmentView` nicht ausgelöst.

Lösung: Die Methode `HierarchyController#getIsSegmentSelected` wurde hinzugefügt. Damit kann eine `ElementTreeCell` beim Neuerzeugen einer `SegmentView` den Selektionsstatus des zugehörigen `Segments` abfragen.

Getestet durch:

- Das Anzeigen des korrekten Selektionsstatus wird durch eine manuelle Verifikation zur Laufzeit getestet.

Entdeckung: Dieses Fehlverhalten wurde während einer Fehlersuche zur Laufzeit des Programms entdeckt.

2.11 ConnectionViews werden beim Redo falsch gezeichnet

Beschreibung: Modifiziert eine Aktion die rückgängig gemacht werden kann ein verbundenes `Segment`, dann zeichnet der zugehörige Redo die `ConnectionView` falsch ein.

Analyse: Beim Zeichnen der `ConnectionView` bei einem Redo, werden die Positionen der Endpunkte der `ConnectionViews` aus den `ConnectorViews` ausgelesen. Zu diesem Zeitpunkt sind die Positionen der Endpunkte der `ConnectionViews` noch nicht korrekt bzw. veraltet. Deshalb werden die `ConnectionViews` falsch gezeichnet.

Lösung: Die Daten zum Zeichnen der `ConnectionViews` werden direkt von den entsprechenden `Connectoren` eingelesen.

Getestet durch:

- Das korrekte Zeichnen der `ConnectionViews` bei Redos wird durch eine manuelle Verifikation zur Laufzeit getestet.

Entdeckung: Dieses Fehlverhalten wurde während einer Fehlersuche zur Laufzeit des Programms entdeckt.

2.12 Connections werden in Commands nicht für Undo/Redo zwischengespeichert

Beschreibung: Bei den folgenden Nutzeraktionen werden die **Connections** der betreffenden **Segmente** beim Redo nicht wiederhergestellt.

- Segment duplizieren
- Segment löschen
- Segment verschieben
- Connector verschieben
- Group löschen

Analyse: Die **Commands**, die diese Nutzeraktionen kapseln, speichern die **Connections** nicht zwischen und können sie deshalb nicht wiederherstellen.

Lösung: Das ZwischenSpeichern und Wiederherstellen von **Connections** wurde ergänzt oder von den **Controllern** in die **Commands** verschoben.

Getestet durch:

- Das korrekte Verhalten in den oben genannten Beispielen wurde manuell überprüft,

Entdeckung: Dieses Fehlverhalten wurde während einer Fehlersuche zur Laufzeit des Programms entdeckt.

2.13 Kompatibilitätskriterien erzeugen jeden Verstoß zweimal

Beschreibung: Ein Kompatibilitätskriterium (**CompatibilityCriterion**) prüft bei Verbindungen (**Connection**) von Straßensegmenten (**Segment**), ob die aneinander angeschlossenen Enden Eigenschaften haben, die miteinander kompatibel sind (siehe auch: Pflichtenheft). Falls die Enden einer Verbindung nicht kompatibel sind, wird ein Verstoß (**Violation**) erzeugt, in dem das Kriterium selbst und die verstoßenden Straßensegmente referenziert werden. Verbindet man jedoch zwei inkompatible Enden, so zeigt ROSE in der Problemübersicht zwei Verstöße an, bei denen die verstoßenden Segmente in unterschiedlicher Reihenfolge genannt werden.

Analyse: Die Anzeige in der Problemübersicht ist korrekt, vom Kompatibilitätskriterium werden tatsächlich zwei Verstöße erzeugt. Dies kommt daher, dass Kompatibilitätskriterien über das hinzufügen von Verbindungen durch die Straßensegmente informiert werden. Da an jeder Verbindung von Straßensegmenten zwei Straßensegmente beteiligt sind, wird das Kompatibilitätskriterium zweimal über die neue Verbindung informiert und prüft sie

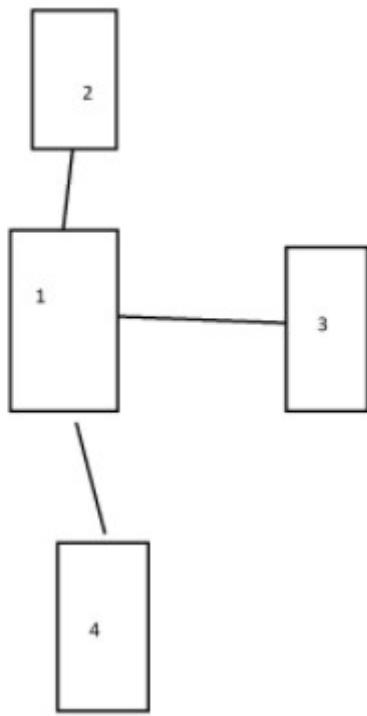


Abbildung 1: Im Bild werden vier Segmente dargestellt. Segment 1 konfliktiert mit den Segmenten 2, 3 und 4. Wenn die Segmente in der Reihenfolge 4, 3, 2, 1 geprüft werden, wurden bei der Überprüfung von 4, 3 und 2 bereits Verstöße für alle genannten Konflikte erzeugt. Bei der Prüfung von Segment 1 ist es daher falsch, Verstöße für die Konflikte zu erzeugen.

zweimal. Vor der Erzeugung eines Verstoßes wird zwar kontrolliert, ob ein solcher Verstoß bereits existiert, jedoch spielt dabei die Reihenfolge der Segmente im Verstoß eine Rolle. Somit erzeugt das Kompatibilitätskriterium mehrere Verstöße, in denen die Segmente in unterschiedlicher Reihenfolge genannt werden. Das Problem wird in Grafik 1 exemplarisch visualisiert.

Lösung: Vor der Erzeugung eines Verstoßes prüft das Kompatibilitätskriterium, ob es aus der Perspektive des jeweils anderen beteiligten Segmentes einen Verstoß bereits erzeugt hat. Ist dies der Fall, so wird kein neuer Verstoß erzeugt.

Entdeckung: Dieses Fehlverhalten wurde beim Schreiben von Modultests zur Erhöhung der Zeilenüberdeckung entdeckt (siehe 4).

2.14 Attribut “Längsneigung” ist mit falschem Datentyp gespeichert

Beschreibung: Jedes implementierte Straßensegment (Unterklassen von `HighwaySegment`) unterstützt das Attribut “Längsneigung” (`AttributeType#SLOPE`). Nach dem Entwurf ist die Längsneigung zwar ein Prozentwert und wird als gebrochene Zahl (`DataType.FRACTIONAL` bzw. `double`) repräsentiert, jedoch erlauben die entsprechenden Getter, Setter und `AttributeAccessors` nur Ganzzahlen.

Analyse: Die Attribute und Methoden, die zur Speicherung der Längsneigung verwendet werden, haben den Datentyp `double`. Dies ist eine Inkonsistenz in der Modellierung der Klassen.

Lösung: Der Datentyp des Attributs `HighwaySegment#slope` so wie die entsprechenden Getter und Setter werden auf den Datentyp `double` geändert. Ebenso wird der Datentyp von `SerializedSegment#slope` auf `double` geändert. In den Tests werden die Eingaben entsprechend als Fließkommawerte formuliert und Ausgabeprüfungen um eine erlaubte Diskrepanz ergänzt. Im sonstigen Quellcode müssen keine Anpassungen vorgenommen werden, da die Typprüfung von Attributen ohnehin immer über den `DataType` des entsprechenden `AttributeAccessors` durchgeführt wurden und der modellierte Datentyp nun zum Java-Datentyp passt.

Getestet durch:

- `CriterionGuiTest#testValidateCompletenessCriterion`

Entdeckung: Dieses Fehlverhalten wurde beim Schreiben von Modultests zur Erhöhung der Zeilenüberdeckung entdeckt (siehe 4).

2.15 Zuletzt geöffnete Projekte werden mehrmals gespeichert

Beschreibung: Die Pfade (Path) der zuletzt geöffneten Projekte zählen zu den projektübergreifenden Anwendungsdaten und werden im RoseApplicationDataSystem gespeichert. Jedes Projekt soll maximal einmal darin genannt werden. Das RoseApplicationDataSystem erkennt jedoch für manche relativen Pfade (wie bspw. ./directory/file.rose.json und directory/./directory/file.rose.json) nicht, dass diese Pfade die selbe Datei referenzieren und speichert beide Projektpfade ab.

Analyse: Die Menge der Pfade zuletzt geöffneter Projekte wird als Set<Path> gespeichert, um Duplikate auszuschließen. Ein Path modelliert in Java einen Pfad und nicht eine Datei, daher sind die oben beschriebenen Path-Instanzen nicht äquivalent.

Lösung: Die Pfade werden vor dem Hinzufügen zum Set<Path> zu absoluten Pfaden umgewandelt und anschließend normalisiert. Durch die Normalisierung werden alle relativen Komponenten im Pfad aufgelöst.

Getestet durch:

- Der Modultest RoseApplicationDataSystemTest#testAddRecentProjectPathAgainIsIgnored prüft exemplarisch, ob verschiedene Pfade zur selben Datei nur einmal hinzugefügt werden.

Entdeckung: Dieses Fehlverhalten wurde beim Schreiben von Modultests zur Erhöhung der Zeilenüberdeckung entdeckt (siehe 4).

2.16 Eingabefelder für INTEGER und FRACTIONAL-Attribute können nicht geleert werden

Beschreibung: Die Eingabe von Attributwerten vom Datentyp DataType#INTEGER und DataType#FRACTIONAL funktioniert mithilfe eines Texteingabefeldes. Es ist nicht möglich, den Inhalt dieses Textfeldes zu löschen, um eine andere Zahl einzugeben.

Analyse: Die Textfelder werden durch Unterklassen von TextFieldAttribute verwaltet. Kann eine Eingabe nicht in einen gültigen Wert eingelesen werden, so wird sie verworfen und der Inhalt des Textfeldes wird auf den letzten gültigen Wert zurückgesetzt. Insbesondere ist ein leeres Eingabefeld ungültig und wird daher zurückgesetzt.

Lösung: Ungültige Eingaben werden nicht sofort verworfen, sondern nicht gespeichert und stattdessen in roter Farbe hervorgehoben. Sobald eine gültige Eingabe vorliegt, kann der Wert übernommen und wieder in schwarzer Farbe dargestellt werden.

Getestet durch:

- Im Rahmen eines Nutzertests wurde das korrigierte Verhalten getestet.
- `TestAttributeEditor#testEnterAttribute`

Entdeckung: Dieses Problem wurde im Rahmen des Nutzertests entdeckt und bemängelt (siehe 7).

2.17 Falsche Fehlermeldungen für gescheiterten im- und export von Kriterien

Beschreibung: Falls beim Im- und Export von Kriterien ein Fehler auftritt, soll dem Nutzer eine Fehlermeldung angezeigt werden. Die Fehlermeldung ist aber identisch zu der beim Im- oder Export von einem Projekt und spricht ausdrücklich davon, dass ein Projekt nicht geladen (importiert) oder gespeichert (exportiert) werden konnte.

Analyse: Es gibt nur die `ErrorTypen` Import und Export. Diese waren speziell für das Projekt gedacht was aber durch die Bezeichnung nicht eindeutig war. Daher wurden sie auch für Fehler beim Import und Export benutzt.

Lösung: Die `ErrorTypes` `IMPORT_ERROR` und `EXPORT_ERROR` wurden zu `PROJECT_IMPORT_ERROR` bzw `PROJECT_EXPORT_ERROR` umbenannt. Es wurden die neuen `ErrorTypes` `CRITERIA_IMPORT_ERROR` und `CRITERIA_EXPORT_ERROR` hinzugefügt, sowie neue Einträge in den Sprachdateien, die jetzt im Fehlerfall ausgegeben werden.

Getestet durch:

- Im Testfall `testExportShowErrorOnFailure` der Test Klasse `RosePlausibilityControllerTest` testet der Test den Fall das beim Kriterien export etwas fehlschlägt.
- In der Test Klasse `RosePlausibilityControllerTest` testet der Test `testExportShowErrorOnFailure` den Fall das beim Project export etwas fehlschlägt.

Entdeckung: Dieses Fehlverhalten wurde während einer Fehlersuche zur Laufzeit des Programms entdeckt.

2.18 Das CriteriaWindow öffnet sich im Hintergrund

Beschreibung: Öffnet man das CriteriaWindow, um Kriterien zu editieren, sollte es sich immer im Vordergrund öffnen (d.h. sichtbar sein und im Fokus des Betriebssystems stehen). Es öffnet sich aber häufig hinter dem Hauptfenster des Programms. Außerdem wird es, falls es hinter dem Hauptfenster liegt, bei erneutem Öffnen auch nicht in den Vordergrund gesetzt.

Analyse: Es wird an keiner Stelle festgelegt welches Fenster den Fokus hat oder welches im Vordergrund stehen soll.

Lösung: Es wurde ein `toFront()`-Aufruf auf der `Stage` aller Fenster des Programms hinzugefügt.

Getestet durch:

- Es wurde manuell geprüft, dass sich die Fenster im Vordergrund öffnen.
- Für das korrekte Funktionieren der GUI-Tests ist es notwendig, dass die Fenster im Vordergrund stehen. Daher wird durch jeden GUI-Test auch die Lösung dieses Problems getestet.

Entdeckung: Dieses Fehlverhalten wurde während einer Fehlersuche zur Laufzeit des Programms entdeckt.

2.19 Draggen von ElementViews im HierarchyPanel erzeugt inkonsistente View-Zustände

Beschreibung: Beim Ändern der Hierarchie von Elementen im HierarchyPanel wird das gleiche Segment mehrfach angezeigt.

Analyse: Die `ElementTreeCells` des `TreeViews` im `HierarchyPanel` können von JavaFx wiederverwendet werden. Beim Wiederverwenden einer `ElementTreeCell` werden die Subscriptions von `ElementTreeCell` nicht richtig gelöst. Auf diese Weise werden in der View falsche Observer-Callbacks empfangen, die einen inkonsistenten View-Zustand verursachen.

Lösung: Die Subscriptions von `ElementTreeCell` werden beim Wiederverwenden gelöst und neu gesetzt.

3 Nicht behobene Bugs

In diesem Abschnitt werden jene Bugs dokumentiert, die wir nicht behoben haben. Die Dokumentation eines Bugs umfasst eine Beschreibung der Wirkung des Bugs sowie, soweit bekannt, eine Analyse der Ursache des Bugs. Auch ein möglicher Lösungsansatz für den Bug wird kurz vorgestellt. Zusätzlich wird festgehalten wie der Bug entdeckt wurde.

3.1 Richtungspfeil wird auf Basissegmenten manchmal nicht gezeichnet

Beschreibung: Wenn die beiden Connectoren eines Basensegments direkt übereinander platziert werden verschwindet der Richtungspfeil des Segments. Der Pfeil wird erneut gezeichnet sobald die Connectoren nicht mehr direkt übereinander liegen.

Analyse: Der Pfeil wird mit Hilfe einer JavaFx ImageView realisiert, auf diese wird ein Rotate Objekt angewandt um den Pfeil wie benötigt auszurichten. Erhält das Rotate Objekt den Winkel 0 als Eingabe so wird der Pfeil nicht gezeichnet.

Lösungsansatz: Aktuell ist kein sauberer Lösungsansatz bekannt, da es sich um einen Fehler des JavaFx Rotate Objekts zu handeln scheint. Eine Möglichkeit das Problem zu Beheben wäre den Eingabewert vor der Nutzung zu überprüfen und wenn nötig minimal abzuändern, da der Fehler scheinbar nur bei einer Eingabe von 0 auftritt.

Entdeckung: Dieser Fehler wurde bei der Implementierung des Richtungspfeils bemerkt.

3.2 Attributeditor skaliert mit dem Grid

Beschreibung: Die Größe des Attributeditors ist der Größe des Grids unterworfen. Dies führt dazu, dass die angezeigten Attribute manchmal nur schwer lesbar sind.

Analyse: Der Attributeditor wird auf dem Grid platziert, da die Positionierung von der Position des zugehörigen Straßensegmentes abhängt. Als Kindobjekt des Grids wird er mit diesem skaliert.

Lösungsansatz: Der Attributeditor muss entweder neu skaliert und positioniert werden sobald das Grid neu skaliert wird oder komplett von diesem gelöst werden. Dies würde die korrekte Positionierung des Editors allerdings stark erschweren.

Entdeckung: Dieses Fehlverhalten wurde beim ausführen beispielhafter Anwendungsfälle entdeckt.

3.3 Probleme mit dem verbinden von Connectoren

Beschreibung: Werden zwei Connectoren übereinander gezogen um sie zu verbinden, passiert dies in einigen Fällen nicht und die Connectoren bleiben eingefärbt, bis sie mit der Maus berührt werden.

Analyse: Dieses Problem teilt seine Ursache mit anderen Problemen und ist deswegen in §3.6 zusammengefasst. Im Falle der Verbindungen bedeutet dies, dass das Event in dem der Mausklick beendet wird nicht beim eigentlich gezogenen Connector, sondern beim stationären Connector ankommen kann. In diesem Fall kann keine Connection erzeugt werden da der Connector gar nicht bereit ist verbunden zu werden und nicht auf dieses event wartet.

Lösungsansatz: Dieses Problem teilt seine Lösung mit anderen Problemen und ist deswegen in §3.6 zusammengefasst.

Entdeckung: Dieses Fehlverhalten wurde beim ausführen beispielhafter Anwendungsfälle entdeckt.

3.4 SegmentViews können sich gegenseitig überdecken

Beschreibung: Große Diagonale Base Segmente können andere Segmente überdecken. Die überdeckten Segmente können nicht mehr ausgewählt oder verschoben werden kann.

Analyse: Dieses Problem teilt seine Ursache mit anderen Problemen und ist deswegen in §3.6 zusammengefasst. Für dieses Problem ist relevant, dass diagonale Base Segmente eine sehr große Pane haben, die ganze Segmente überdecken kann. In diesem Fall lösen klicks auf ein verdecktes Segment keine Events mehr auf diesem auf. Die einzige Lösung ist das verdeckende Base Segment so zu verschieben das die Pane nicht mehr über dem verdeckten Segment liegt.

Lösungsansatz: Dieses Problem teilt seine Lösung mit anderen Problemen und ist deswegen in §3.6 zusammengefasst.

Entdeckung: Dieses Fehlverhalten wurde beim ausführen beispielhafter Anwendungsfälle entdeckt.

3.5 Gelegentlich auftretende NullPointerExceptions beim Verbinden von Straßensegmenten

Beschreibung: Beim Verbinden von zwei ConnectorViews kann es vereinzelt zu NullPointerExceptions kommen.

Analyse: Die stationäre ConnectorView überdeckt die Bewegte. Dabei wird das Event, welches das Ende des DragAndDrop-Vorgangs markiert von dieser abgefangen und kommt somit nicht bei der bewegten ConnectorView an. Das Resultat ist der Versuch einen nicht bestehenden DragAndDrop-Vorgang auf der stationären ConnectorView zugehen. Siehe §3.6.

Lösungsansatz: Dieses Problem teilt seine Lösung mit anderen Problemen und ist deswegen in §3.6 zusammengefasst. Kurzfristig haben wir eine Überprüfung eingesetzt, welche den Zugriff auf das resultierende `null`-Objekt verhindert. Dies führt aber dazu, dass trotzdem keine Verbindung entsteht, es wird lediglich die Ausnahme verhindert.

Entdeckung: Dieses Fehlverhalten wurde beim ausführen beispielhafter Anwendungsfälle entdeckt.

3.6 JavaFx und die Weiterleitung von Events

Beschreibung: Dieser Bug manifestiert sich in unterschiedlichen Situationen. Grundlegend besteht das Problem darin, dass Zeichenobjekte in der Editorfläche Eingaben durch den Nutzer scheinbar ignorieren. Beispiele hierfür sind, dass Verbindungen zwischen Segmenten nicht eingezeichnet werden oder dass Segmente scheinbar grundlos Mausklicks ignorieren.

Analyse: JavaFx ordnet Zeichenobjekte in einer Baumstruktur an, dabei hat jedes Zeichenobjekt genau ein Elternobjekt sowie beliebig viele Kindobjekte. Events werden dabei immer auf dem aktuell als oberstes gezeichnete Zeichenobjekt ausgelöst. Wird ein Event (z.B. ein Mausklick) auf einem Zeichenobjekt ausgelöst so wird die entsprechende Methode zur Behandlung des Events auf dem Zeichenobjekt aufgerufen. Bietet das Zeichenobjekt keine passende Methode oder beendet das Objekt das Event nicht richtig so wird das Event anschließend an das Elternobjekt des Zeichenobjekts weitergeleitet. In unserem Fall ist das Elternobjekt das `Grid`. Kindobjekte dessen sind die `SegmentViews` sowie die `ConnectionViews`. `SegmentViews` halten außerdem `ConnectorViews` als Kindobjekte. Da Events immer an das entsprechende Elternobjekt weitergeleitet werden und nicht an ggf. überdeckte weitere Kindobjekte des `Grids` kann es passieren, dass Events, die eigentlich für Segmente gedacht waren, an das Grid weitergeleitet und dann dort beendet werden. Beispielsweise kann ein `BaseSegment` andere Segmente mit seinem transparenten Pane überdecken und so alle Mausklicks auf diese blockieren.

Lösungsansatz: Der sauberste Ansatz um dieses Problem zu beheben wäre es die Verwaltung der Events von JavaFx zu erweitern. In diesem Fall müsste das `Grid` als Elternobjekt überprüfen an welche seiner Kindobjekte dieses Event hätte gehen können und es an dieses weiterleiten. Dies erfordert das halten einer Liste von Kindobjekten pro Event, um sicherzustellen das keine Endlosschleifen entstehen. Des weiteren müsste zur Bestimmung der potenziellen Ziele für das Event eine geometrische Überprüfung stattfinden.

3.7 Schnelles bewegen von Segmenten bewegt den Hintergrund

Beschreibung: Wird ein Segment zu schnell bewegt kann es passieren, dass das Segment nur ein Stück bewegt, dafür aber der Hintergrund verschoben wird.

Analyse: Dieses Problem hat seine Ursache in einem Problem von JavaFx und wird deshalb in §3.8 erklärt. In diesem Fall kann es sein, dass das Event solange hängt bis sich die Maus außerhalb der `SegmentView` befindet. Daraufhin bekommt der Hintergrund also das `Grid` das Event, was dazu führt dass sich der Hintergrund bewegt.

Lösungsansatz: Eine mögliche Lösung, die funktionieren könnte (ohne das eigentli-

che Problem zu beheben), wäre es, eine Flagge einzuführen die per Callback von der `SegmentView` auf dem `Grid` gesetzt wird. So könnte durch das laufende Drag Event selbst eine Flagge gesetzt werden, die die Funktionalität des JavaFx-Calls, der den Beginn eines Drag-Events signalisiert, übernimmt.

3.8 JavaFx und das Auslösen von DragEvents

Beschreibung: In JavaFx soll, falls ein Drag Event beginnt, die Methode `onMouseDragged` ausgeführt werden bevor die Methode `onDragDetected` ausgeführt wird. Dies passiert manchmal allerdings nicht in dieser Reihenfolge.

Daher kann es vorkommen, dass ein Objekt das bewegt werden soll noch nicht dazu bereit ist und den event Aufruf weitergibt.

Analyse: In unserem Fall kann dies dazu führen das Events verzögert ausgeführt werden was zu diversen Fehlern führen kann, da normalerweise erwartet wird, dass ein `DragDetected` Event ausgelöst wird bevor ein anhaltendes `MouseDragged` Event startet.

4 Modultests

Um auszuwerten, welche Teile unseres Quellcodes durch Modultests geprüft werden, messen wir die Zeilenüberdeckung. Dabei schränken wir den Fokus der Zeilenüberdeckung auf das Controller-/ und auf das Model-Subsystem ein und schließen von der Überdeckung die Pakete aus, die ausschließlich Messwertfunktionalität implementieren, da diese nur Signaturen und Dokumentation entworfener Klassen enthalten, die im Rahmen dieses PSE-Projektes nicht implementiert wurden.

Wir haben uns selbst eine Zeilenüberdeckung von 95 % als Ziel gesetzt. Die verbleibenden 5 % setzen wir als Spielraum an, einerseits für untestbaren Code und andererseits für schwer testbaren Code.

Untestbarer Code kann in Java beispielsweise im Zusammenhang mit Zugriffen auf das Dateisystem entstehen: Wenn unser Programmcode zunächst prüft, ob die Anwendung auf eine Datei zugreifen kann und direkt im Anschluss auf die Datei zugreift, muss im Programmcode dennoch eine Ausnahmebehandlung für `IOExceptions` angegeben werden, damit das Programm korrekt kompiliert werden kann. Eine solche Ausnahme können wir jedoch nicht automatisiert auslösen, ohne neue Abstraktionen in unserem Programmcode einzufügen.

Da wir in der Implementierungsphase mit 270 Testfällen die meisten Klassen bereits grundlegend geprüft haben, sind wir mit 81 % Zeilenüberdeckung in die Qualitätssicherungsphase gestartet. In der Qualitätssicherungsphase haben wir durch das Hinzufügen von 170 weiteren Modultests und durch das Erweitern zahlreicher bestehender Testfälle die Zeilenüberdeckung auf 94 % erhöht. Tabelle 1 schlüsselt die Testüberdeckung durch Modultests nach Paketen auf. Die Zeilenüberdeckung beträgt am Ende der Qualitätssicherungsphase in der Hälfte der Pakete 100 % und in allen Paketen über 80 %.

Zwar ist diese Erhöhung beträchtlich kleiner als die Zeilenüberdeckung, die wir bereits in der Implementierungsphase erreicht haben, jedoch ist zu beachten, dass insbesondere bei der Quellcodeüberdeckung das *Paretoprinzip* greift. Denn im Rahmen der Implementierungsphase haben wir hauptsächlich Modultests geschrieben, die die Funktionalität grundlegend überprüfen, indem sie einzelne Funktionen mit dafür typischen Werten ausführen und das Ergebnis beispielsweise anhand des Rückgabewertes auf Plausibilität prüfen, nicht aber unbedingt auf Korrektheit überprüfen. Dabei werden die meisten Zeilen der aufgerufenen Methoden ausgeführt, nicht jedoch werden Zeilen aufgerufen, die der Behandlung von Randfällen dienen. Um genau diese Randfälle im Rahmen der Qualitätssicherungsphase zu testen, müssen oft komplizierte Ausgangsbedingungen geschaffen werden, die im Vergleich zu den Tests mit üblichen Eingaben nur wenige Zeilen mehr abdecken. Zudem ist in solchen Fällen auch eine detailliertere Auswertung der Testergebnisse erforderlich, um die getestete Funktionalität auch tatsächlich zu überprüfen.

Da nach der “Softwaretechnik 1”-Vorlesung Bugs besonders häufig in solchen Randfällen auftreten, ist es uns wichtig solche Randfälle ausführlich zu prüfen. Diese Bemühungen haben auch Früchte getragen, beispielsweise wurden beim Schreiben der Modultests die folgenden Bugs gefunden:

- `RoseSortedBox` kann durch `Iterator` geändert werden (siehe §2.1)
- `Group` kann durch `Iterator` unbemerkt geändert werden (siehe §2.2)
- Der Vergleich von `HighwaySegments` mit anderen Arten von Segmenten liefert eine falsche Sortierung (siehe §2.3)
- Kompatibilitätskriterien erzeugen jeden Verstoß zweimal (siehe §2.13)
- Attribut “Längsneigung” ist mit falschem Datentyp gespeichert (siehe §2.14)
- Zuletzt geöffnete Projekte werden mehrmals gespeichert (siehe §2.15)

Paket	Überdeckung	
	Zeilen	Zweige
edu.kit.rose.controller	100 %	n/a
edu.kit.rose.controller.application	100 %	100 %
edu.kit.rose.controller.attribute	92 %	83 %
edu.kit.rose.controller.command	100 %	100 %
edu.kit.rose.controller.commons	90 %	81 %
edu.kit.rose.controller.hierarchy	95 %	87 %
edu.kit.rose.controller.navigation	100 %	n/a
edu.kit.rose.controller.plausibility	100 %	57 %
edu.kit.rose.controller.project	94 %	70 %
edu.kit.rose.controller.roadsystem	83 %	70 %
edu.kit.rose.controller.selection	100 %	75 %
edu.kit.rose.infrastructure	100 %	100 %
edu.kit.rose.infrastructure.language	84 %	50 %
edu.kit.rose.model	90 %	73 %
edu.kit.rose.model.plausibility	100 %	100 %
edu.kit.rose.model.plausibility.criteria	97 %	82 %
edu.kit.rose.model.plausibility.criteria.validation	100 %	88 %
edu.kit.rose.model.plausibility.violation	90 %	90 %
edu.kit.rose.model.roadsystem	98 %	89 %
edu.kit.rose.model.roadsystem.attributes	100 %	n/a
edu.kit.rose.model.roadsystem.elements	99 %	95 %
Total	94 %	80 %

Tabelle 1: Quellcodeüberdeckung durch Modultests

5 Globale Testfälle

In diesem Abschnitt wird die Überdeckung der, im Pflichtenheft spezifizierten, globalen Testfälle dokumentiert. Für jeden globalen Testfall wird angegeben, ob er abgedeckt wird, und falls ja, durch welche Art von Test er abgedeckt wird. Ein solcher Test kann ein GUI-Test, ein Nutzertest oder eine manuelle Verifikation sein.

Zur Umsetzung der GUI-Tests haben wir eine Kombination aus JUnit und dem Testframework TestFx verwendet.

5.1 Programm ausführen

Wird abgedeckt durch: `TestProgramStart#testStartProgram`

Der Testfall wurde erfolgreich abgedeckt: ✓

5.2 Straßensegment platzieren (Doppelklick)

Wird abgedeckt durch: `TestSegmentBox#testSetStreetSegmentDoubleClick`

Der Testfall wurde erfolgreich abgedeckt: ✓

5.3 Straßensegment platzieren (Drag and Drop)

Wird abgedeckt durch: `TestSegmentBox#testSetStreetSegmentDragAndDrop`

Der Testfall wurde erfolgreich abgedeckt: ✓

5.4 Attribute eintragen

Wird abgedeckt durch: `TestAttributeEditor#testEnterAttribute`

Der Testfall wurde erfolgreich abgedeckt: ✓

5.5 Straßensegmente verbinden (Basisstraßesegment)

Wird abgedeckt durch: SegmentConnectTest#testConnectBaseSegment

Der Testfall wurde erfolgreich abgedeckt: ✗

Begründung: Der TestFx Roboter bewegt das Mauszeiger zu schnell und deshalb tritt der 3.6 Bug ein. Damit kann der Roboter nicht korrekt die Segmente verschieben. In normalem Anwendung gibt, trotzdem, dieses Problem nicht. Das Test wurde auch mit Nutzertest erfolgreich getestet.

5.6 Das Projekt speichern

Wird abgedeckt durch: SaveProjectGlobalTestCase#testSaveProject

Der Testfall wurde erfolgreich abgedeckt: ✓

5.7 Das Straßennetz in das FREEVAL-Format exportieren

Wird abgedeckt durch: YamlExportStrategyTest#testExportProject und ExportProjectGlobalTestCase#testExportProjectToFreevalYaml

Der Testfall wurde erfolgreich abgedeckt: ✓

5.8 Das Straßennetz in das SUMO-Format exportieren

Wird abgedeckt durch: ExportProjectGlobalTestCase#testExportProjectToSumoNetXml

Der Testfall wurde erfolgreich abgedeckt: ✗

Begründung: Der Test existiert, aber die Unterstützung des SUMO-Exportformats wurde nicht implementiert.

5.9 Selektierte Straßensegmente verschieben

Wird abgedeckt durch: TestSegmentManipulation#testMoveSelectedSegments

Der Testfall wurde erfolgreich abgedeckt: ✗

Begründung: Der TestFx Roboter bewegt das Mauszeiger zu schnell und deshalb tritt der 3.6 Bug ein. Damit kann der Roboter nicht korrekt die Segmente verschieben. In normalem Anwendung gibt, trotzdem, dieses Problem nicht. Das Test wurde auch mit Nutzertest erfolgreich getestet.

5.10 Straßensegmente verbinden (Einfahrt)

Wird abgedeckt durch: SegmentConnectTest#testConnectEntrySegment

Der Testfall wurde erfolgreich abgedeckt: ✗

Begründung: Der TestFx Roboter bewegt das Mauszeiger zu schnell und deshalb tritt der 3.6 Bug ein. Damit kann der Roboter nicht korrekt die Segmente verschieben. In normalem Anwendung gibt, trotzdem, dieses Problem nicht. Das Test wurde auch mit Nutzertest erfolgreich getestet.

5.11 Duplizieren von Straßensegmenten

Wird abgedeckt durch: TestSegmentManipulation#testDuplicateSegment

Der Testfall wurde erfolgreich abgedeckt: ✗

Begründung: Der TestFx Roboter bewegt das Mauszeiger zu schnell und deshalb tritt der 3.6 Bug ein. Damit kann der Roboter nicht korrekt die Segmente verschieben. In normalem Anwendung gibt, trotzdem, dieses Problem nicht. Das Test wurde auch mit Nutzertest erfolgreich getestet.

5.12 Undo und Redo

Wird abgedeckt durch: TestRoadSystemTools#testUndoRedo

Der Testfall wurde erfolgreich abgedeckt: ✓

5.13 20 mal rückgängig machen und wiederherstellen

Wird abgedeckt durch: TestRoadSystemTools#test20TimesUndoRedo

Der Testfall wurde erfolgreich abgedeckt: ✓

5.14 Projekt laden

Wird abgedeckt durch: GUI-Test LoadProjectGlobalTestCase#testLoadProject und Modultest RoseExportStrategyTest#testReImportProject

Der Testfall wurde erfolgreich abgedeckt: ✓

5.15 Sprache zur Laufzeit einstellen

Wird abgedeckt durch: MenuTest#testChangeLanguage, Nutzertest

Der Testfall wurde erfolgreich abgedeckt: ✓

5.16 Vollständigkeitskriterien prüfen

Wird abgedeckt durch: CriterionGuiTest#testValidateCompletenessCriterion

Der Testfall wurde erfolgreich abgedeckt: ✓

5.17 Kompatibilitätskriterien auswerten

Wird abgedeckt durch: CriterionGuiTest#testValidateCompatibilityCriterion

Der Testfall wurde erfolgreich abgedeckt: ✗

Begründung: Der TestFx Roboter bewegt das Mauszeiger zu schnell und deshalb tritt der 3.6 Bug ein. Damit kann der Roboter nicht korrekt die Segmente verschieben. In normalem Anwendung gibt, trotzdem, dieses Problem nicht. Das Test wurde auch mit Nutzertest erfolgreich getestet.

5.18 Straßensegment löschen

Wird abgedeckt durch: TestSegmentManipulation#testDeleteSegment

Der Testfall wurde erfolgreich abgedeckt: ✓

5.19 Bulk-Edit

Wird abgedeckt durch: TestAttributeEditor#testBulkEdit

Der Testfall wurde erfolgreich abgedeckt: ✓

5.20 Anzahl der Fahrstreifen ändern

Der Testfall wurde erfolgreich abgedeckt: ✗

Begründung: Die Breite der Straßensegmente wird nicht dynamisch angepasst.

5.21 Attribute auf der Editorfläche anzeigen

Der Testfall wurde erfolgreich abgedeckt: ✗

Begründung: Auf Straßensegmenten werden keine Attribute dargestellt.

5.22 Undo durch Tastenkombination bedienen

Wird abgedeckt durch: TestRoadSystemTools#testUndoShortcut

Der Testfall wurde erfolgreich abgedeckt: ✓

5.23 Kompatibilitätskriterium bearbeiten

Wird abgedeckt durch: CriterionGuiTest#testEditCriteria

Der Testfall wurde erfolgreich abgedeckt: ✓

5.24 Plausibilitätskriterien ex- und importieren

Wird abgedeckt durch: CriterionGuiTest#testImportExportCriterion

Der Testfall wurde erfolgreich abgedeckt: ✓

5.25 Zur Fehlerquelle springen

Wird abgedeckt durch: CriterionGuiTest#testJumpToViolation

Der Testfall wurde erfolgreich abgedeckt: ✓

5.26 Sicherungskopien

Wird abgedeckt durch: RoseProjectControllerTest#testBackup, Nutzertest

Der Testfall wurde erfolgreich abgedeckt: ✓

5.27 Tastenkombinationen einsehen

Wird abgedeckt durch: MenuTest#testShortcutsDisplay

Der Testfall wurde erfolgreich abgedeckt: ✓

5.28 Elemente suchen

Wird abgedeckt durch: TestHierarchy#testElementSearch

Der Testfall wurde erfolgreich abgedeckt: ✓

5.29 Länge und Anzahl von Messwert-Zeitintervallen konfigurieren

Der Testfall wurde erfolgreich abgedeckt: ✗

Begründung: Die Messwertfunktionalität wurde nicht implementiert.

5.30 Straßensegment drehen

Wird abgedeckt durch: TestSegmentManipulation#testRotateSegment

Der Testfall wurde erfolgreich abgedeckt: ✓

5.31 Zoomen in der Editorfläche

Wird abgedeckt durch: TestRoadSystemTools#testZoom

Der Testfall wurde erfolgreich abgedeckt: ✓

5.32 Gruppieren von Straßensegmenten

Wird abgedeckt durch: TestHierarchy#testGroupStreetSegments

Der Testfall wurde erfolgreich abgedeckt: ✓

5.33 Messwerte eintragen

Der Testfall wurde erfolgreich abgedeckt: ✗

Begründung: Die Messwertfunktionalität wurde nicht implementiert.

5.34 Element in andere Gruppe verschieben

Wird abgedeckt durch: TestHierarchy#testMoveElementsToGroup

Der Testfall wurde erfolgreich abgedeckt: ✓

6 Optimierungen

In diesem Abschnitt werden die an "ROSE" vorgenommenen Optimierungen vorgestellt. Unter "Optimierung" verstehen wir hierbei eine Ergänzung bzw. eine Verbesserung des Funktionsumfangs bezüglich des Zustands von "ROSE" am Ende der Implementierungsphase. Für jede Optimierung wird das damit angestrebte Ziel, eine Analyse der Ausgangssituation sowie die implementierte Lösung vorgestellt. Zusätzlich wird angegeben, welche Tests das neu implementierte Verhalten überprüfen.

6.1 Bulk edit

Ziel: Um die Nutzbarkeit der Anwendung zu erhöhen, sollen die Attribute von mehreren Segmenten zeitgleich bearbeitet werden können.

Analyse: Bei größeren Straßennetzen kann es oft sinnvoll sein von vielen Segmenten gleichzeitig Attribute wie die Höchstgeschwindigkeit oder die Fahrstreifenanzahl zu ändern. Dies würde erfordern jedes der Segmente einzeln auszuwählen und das Attribut zu ändern was schnell zu Fehlern führen kann.

Lösung: Es wurde ein Bulk Editor hinzugefügt der es ermöglicht die geteilten Attribute der aktuell ausgewählten Segmente zu bearbeiten.

Getestet durch: Der `SetBulkAttributeAccessorCommandTest` testet das Ändern von Attributen per Bulk Editor und das Verhalten wenn diese Änderungen rückgängig gemacht und wiederhergestellt werden (Undo/Redo).

6.2 Die Richtung von Segmenten wird überprüft

Ziel: Um Fehlern beim erstellen von Straßennetzen vorzubeugen soll kontrolliert werden ob Segmente mit den korrekten Ein- bzw. Ausfahrten verbunden sind.

Analyse: Die Connectoren von Verbindungen zu überwachen ist mit einem `CompatibilityCriterion` nicht möglich da diese Attribute verbundener Segmente überwachen und nicht die verbundenen `Connectoren`.

Lösung: Es wurde eine neue Art von `PlausibilityCriterion` hinzugefügt. Dieses `ConnectorCriterion` kontrolliert die `Connections` eines Segmentes und ob die verbundenen `Connectoren` zu einander passende Typen haben. Zum Beispiel soll ein `entry Connector` nur mit `exit` oder `Ramp exit Connector` verbunden sein)

Getestet durch: Der ConnectorCriterionTest überprüft in mehreren Szenarien ob dies korrekt funktioniert.

6.3 Beim Einfügen eines Straßensegmentes in die Editorfläche per DragAndDrop wird das Segment an der Mausposition eingefügt

Ziel: Um die Benutzbarkeit zu erhöhen sollen Straßensegmente beim loslassen der linken Maustaste an der aktuellen Mausposition in die Editorfläche eingefügt werden.

Analyse: Das Einfügen der Straßensegmente wird über einen zugehörigen Command verwaltet, dieser fügt Segmente allerdings nur in die Mitte der Editorfläche ein.

Lösung: Der zugehörige Command wurde erweitert, sodass er neben seiner vorherigen Form nun auch mit einer Position konstruiert werden kann. Diese wird dann anstelle der Mitte der Editorfläche verwendet um das Segment zu platzieren. Die aktuelle Mausposition kann dann vorher in der view bestimmt werden.

Getestet durch: Das Einfügen der Straßensegmente an der korrekte Mausposition wurde durch eine manuelle Verifikation zur Laufzeit getestet.

6.4 Straßensegmente zeigen Richtungspfeile an

Ziel: Straßensegmente sollen durch einen Richtungspfeil klar machen welche Connector en ihre Eingänge bzw. Ausgänge markieren.

Analyse: Einfahrten und Ausfahrten werden durch Bilddateien dargestellt. Basissegmente dynamisch gezeichnet.

Lösung: Durch editieren der entsprechenden Bilddateien lassen sich die Ein- / und Ausfahrten anpassen. Die Basissegmente zeichnen ab sofort eine ImageView an ihre Mitte, welche entsprechend rotiert wird um einen richtig ausgerichteten Richtungspfeil anzuzeigen.

Getestet durch: Das Anzeigen der Richtungspfeile wurde durch eine manuelle Verifikation zur Laufzeit getestet.

6.5 Fehlermeldungen beim Import- und Export von Plausibilitätskriterien

Ziel: Der Nutzer soll darüber informiert werden, wenn der Import von oder der Export in eine Kriteriendatei fehlschlägt.

Analyse: Die Methoden zum Importieren und Exportieren von Plausibilitätskriterien haben bereits einen `boolean`-Rückgabewert, an dem der Erfolg der Operation abgelesen werden kann. Es gibt jedoch noch keine Fehlernachricht für Fehler beim Import oder Export von Plausibilitätskriterien.

Lösung: Die beiden Fehlermeldungen werden sowohl im `ErrorType` enum als auch in den Übersetzungsdateien hinzugefügt. Der `RosePlausibilityController` lässt im Fehlerfall beim Import oder Export von Plausibilitätskriterien über den `Navigator` eine entsprechende Fehlermeldung anzeigen.

Getestet durch:

- Der Modultest `RosePlausibilityControllerTest#testImportShowsErrorOnFailure`
 - prüft exemplarisch, ob beim Importieren einer nicht existenten Datei die passende Fehlermeldung angezeigt wird.
- Der Modultest `RosePlausibilityControllerTest#testExportShowsErrorOnFailure`
 - prüft exemplarisch, ob beim Exportieren in eine nicht existente Datei die passende Fehlermeldung angezeigt wird.

6.6 Root-Group für das RoadSystem

Ziel: Für einen Programmierer oder Maintainer soll der Aufwand das `HierarchyPanel` zu warten bzw. zu modifizieren verringert werden. Dies soll erreicht werden indem ein einfacher Zugriff (einschließlich benötigter Callbacks) auf die Elemente des Roadsystems ermöglicht wird, die in der internen Baumstruktur an oberster Stelle liegen.

Analyse: Wird ein `Element` einer `Group` hinzugefügt oder aus einer `Group` entfernt, werden diese Änderungen über Observer-Callbacks an andere Komponenten kommuniziert. Ist ein `Element` in einer `Group` und wird darum entfernt ist es in keiner `Group` mehr. Das bedeutet, dass der Remove-Callback ausgelöst wird. Da das `Element` aber keiner anderen `Group` hinzugefügt wird, gibt es keinen Add-Callback mehr. Damit könnten andere Komponenten in einen inkonsistenten Zustand kommen.

Lösung: Dem Rodsystem wird eine Root-Group hinzugefügt. Diese Root-Group enthält alle neu erzeugten Elemente, sowie alle Elemente die in keiner anderen Group enthalten

sind. Damit ist jedes **Element** immer in einer **Group** und alle Add- und Remove-Callbacks werden korrekt ausgelöst.

7 Nutzertests

In diesem Abschnitt stellen wir die von uns durchgeführten Nutzertests vor und analysieren deren Resultate. Die hier beschriebenen Nutzertests setzen den "Benutzer-Betatest" um, der im Pflichtenheft im Abschnitt "Sonstige Tests" beschrieben ist. Das Ziel unserer Nutzertests ist es die Robustheit und die Benutzbarkeit von ROSE in einer produktionsähnlichen Umgebung zu testen.

Damit die Bewertungen der Nutzer im Rahmen der Nutzertests aussagekräftig und vergleichbar sind haben wir alle Tester die gleiche Aufgabenstellung bearbeiten lassen. Diese Aufgabenstellung ist so gestaltet, dass sie die meisten Szenarien und Anwendungsfälle abdeckt, für die wir die benötigte Funktionalität implementiert haben. Während die Tester die Aufgabenstellung bearbeiteten haben wir sie beobachtet, um selbst einen Eindruck der Benutzbarkeit zu gewinnen. Auch dieser Eindruck wird in diesem Abschnitt festgehalten. Nach der Bearbeitung der Aufgabenstellung haben wir die Tester zur Benutzung von ROSE befragt. Die Ergebnisse der Befragung stellen wir hier ebenfalls vor.

Im Folgenden werden die einzelnen Bestandteile des Nutzertests vorgestellt. Dabei wird jeweils die Methodik erläutert und der Bezug zu Anwendungsfällen, Szenarien und globalen Testfällen hergestellt. Zusätzlich werden für jeden Bestandteil die Antworten der Befragung und unsere Beobachtungen angegeben.

7.1 Einstieg

Fragestellung: Zu Beginn wurde dem Nutzer kurz vorgestellt das "ROSE" ein Programm zum Bauen von Straßennetzen ist. Außerdem bekam der Nutzer eine kurze Einleitung. Der Nutzer bekam eine konkrete Aufgabenstellung die er umsetzen soll. Zu dieser Aufgabenstellung gab es jedoch keine Anleitung, um so zu kontrollieren wie Intuitiv die Bedienung ist und an welchen Stellen eine Anleitung erforderlich ist.

Zuerst sollte der Nutzer das Programm öffnen und die Sprache auf Deutsch umstellen.

Getestete Szenarien und Anforderungen: Diese Aufgabe deckt die funktionale Anforderung /FA6010M/ ab, sowie den Teil des Szenarios 10.1.2 SSprache und Export" der die Spracheinstellung betrifft.

Antworten und Beobachtungen: Das Programm hat sich in jedem Test ohne Probleme öffnen lassen. Die Spracheinstellung hat in jedem Test gut funktioniert und wurde sofort von allen Teilnehmern gefunden.

7.2 Erstellung eines Straßennetzes

Aufgabestellung: Die Nutzer bekamen eine Abbildung eines Stücks der A40, dass mehrere Ein- und Ausfahrten sowie Standard-Straßenstücke behinhaltet. Die Abbildung wurde durch Informationen bezüglich Tempolimit, Länge der Abschnitte und Spurenanzahl ergänzt.

Die Aufgabe war es zunächst, dieses Straßennetz in ROSE nachzubauen, alle Segmente korrekt zu verbinden und die Länge der einzelnen Abschnitte einzustellen.

Getestete Szenarien und Anforderungen: Diese Aufgabe deckt die funktionalen Anforderungen /FA1010M/ bis /FA1050W/, /FA2010M/ bis /FA2065M/, /FA2080M/, /FA2090M/, /FA2150W/ und /FA6010M/ ab. Des weiteren deckt sie den Teil von Szenario 10.1.1 Einfügen von Straßensegmenten und Speichernäb, der nicht mit dem Speichern und Laden zusammenhängt.

Antworten und Beobachtungen: Das Einfügen von Segmenten per Drag-and-Drop wurde von allen Teilnehmer schnell entdeckt. Einige Teilnehmer haben auch das Einfügen von Segmenten per Doppelklick gefunden.

Das Verbinden, welches nur möglich ist wenn man den **Connector** eines Segments bewegt, war weniger intuitiv und wurde nicht immer schnell gefunden. Alternativ versuchten einige Nutzer zuerst Segmente einfach nah aneinander zu platzieren um diese zu verbinden. Es wurde angemerkt, dass das Verbinden fehlerbehaftet ist und nicht jedes mal funktioniert. Dies wurde in §3.3 ausführlich dokumentiert.

Der Attributeditor wurde von allen Nutzern schnell gefunden und der Doppelklick, der ihn öffnet, als intuitiv bezeichnet.

7.3 Bulk Edit

Aufgabestellung: Als nächster Schritt war die Aufgabe für alle erstellten Segmente per Mehrfachselektion und Bulk Edit die Fahrstreifenanzahl, das Tempolimit und ob sie in einem Ballungsraum liegen einzustellen. Außerdem sollte für Teilmengen des Netzes die Steigung angepasst werden.

Getestete Szenarien und Anforderungen: Diese Aufgabe testet die funktionalen Anforderungen /FA2080M/ und /FA2160W/.

Antworten und Beobachtungen: Das Mehrfachselektieren von Segmenten per Strg + Mausklick war der erste Gedanke aller Nutzer.

Das öffnen des Bulk Editors mit Strg + Doppelklick war den Nutzern allerdings nicht sofort klar und es wurde ein Rechtsklick oder Doppelklick ohne Strg versucht.

Nachdem den Nutzern bekannt gemacht worden war wie der Bulk Editor zu öffnen ist

war die Bedienung einfach und schnell möglich.

7.4 Speichern, Exportieren und Laden

Aufgabestellung: Als nächstes sollten die Nutzer das gebaute Straßennetz abspeichern und dann in das FREEVAL-YAML-Format exportieren. Anschließend sollten sie ein vorbereitetes Projekt in das Programm laden.

Getestete Szenarien und Anforderungen: Diese Aufgabe deckt den Teil des Szenarios 10.1.1 „Einfügen von Straßensegmenten und Speichern“ bei dem es um das Speichern und Laden eines Projektes geht. Außerdem den Exportteil des Szenarios 10.1.2 „Sprache und Export“. Die Aufgabe deckt die funktionalen Anforderungen /FA4010M/, /FA4040M/, /FA5010M und /FA5020M/ ab.

Antworten und Beobachtungen: Die Funktionalität zum Speichern und Laden wurde von allen Nutzern sofort gefunden und als intuitiv wahrgenommen.

7.5 Fehler suchen und korrigieren

Aufgabestellung: In ein geladenes Straßennetz sind einige Fehler eingebaut worden. Zum einen sind Straßensegmente in falscher Richtung verbunden und zum anderen wechselt die Fahrstreifenanzahl von verbundenen Segmenten von 3 auf 6 was noch keinen Verstoß hervorruft. Diese Fehler werden dem Nutzer als Verstoß angezeigt.

Die nächste Aufgabe ist es nun einen der Verstöße bezüglich der Richtung zu beheben. Außerdem soll ein Plausibilitätskriterium erstellt werden welches Sicherstellt das die Anzahl von Fahrstreifen zwischen zwei verbundenen Segmenten sich maximal um zwei verändert. Dies führt zu neuen Verstößen von denen der Nutzer auch einen beheben soll.

Getestete Szenarien und Anforderungen: Diese Aufgabe deckt die funktionalen Anforderungen /FA3030M/ bis /FA3070W/ ab.

Das Szenario 10.1.3 „Plausibilitätskriterien“ wird von dieser Aufgabe abgedeckt. Außerdem wird das Szenario 10.1.4 „Zoomen und Verschieben der Editorfläche“ abgedeckt da es hier notwendig ist das Zoomen und Verschieben zu nutzen und auch Straßensegmente zu rotieren und zu bewegen.

Antworten und Beobachtungen: Die Verstoßübersicht wurde von allen Nutzern schnell gefunden. Auch das Springen zu den am Verstoß beteiligten Segmenten per Doppelklick wurde schnell ausprobiert. Dadurch konnte schnell einer der Verstöße behoben werden.

Der Kriterieneditor wurde auch sofort gefunden und konnte meistens schnell bedient werden. Etwas unintuitiv war die Auswahl der betroffenen Segmente des Kriteriums, die mit Strg + Mausklick erfolgen muss. Hier hatten manche Nutzer einen einfachen Klick erwartet.

Dennoch waren alle Nutzer schnell dazu in der Lage das Kriterium zu konfigurieren und anschließend einen Verstoß dagegen zu beheben.

7.6 Allgemeines Feedback

Allgemein wurde die Anwendung als übersichtlich und intuitiv bezeichnet. Gerade die simple Bedienung per Maus, die zusätzlich durch Tastenkürzel ergänzt werden kann, wurde gelobt. Als größtes Problem wurde das manchmal fehlerhafte Verbinden von Segmenten genannt. Außerdem skaliert der Attribut Editor mit der Editorfläche was von einigen Nutzern als unintuitiv wahrgenommen wurde. Ansonsten wurden alle Eingaben vom Programm korrekt behandelt und fehlerhafte Eingaben sofort markiert. Von den Nutzern, die Straßennetze bauen müssen, wurde der Vergleich zu Vissim gezogen, was als professionelle Software natürlich einen deutlich größeren Funktionsumfang bietet. Dennoch wurde "ROSE" als Alternative um schnell und einfach Straßennetze zu bauen, die dann in Vissim verfeinert werden können in Betracht gezogen. Vor allem aber für das FREEVAL-YAML-Format wurde "ROSE" als aktuell beste Möglichkeit bezeichnet um die Eingabe von Straßennetzen in einer Tabelle abzulösen.

Es wurden einige Verbesserungsvorschläge genannt, wie die Einführung einer Tabelle (wie in Vissim) um schnell alle Attribute der Segmente einsehen zu können. Auch das Einführen von Standard IDs für Segmente wurde vorgeschlagen, da neu erstellte Segmente in der Hierarchieübersicht aktuell nicht zu unterscheiden sind, bis sie einen neuen Namen bekommen. Allgemein wurde eine Erweiterung der Funktionalität der Hierarchieübersicht vorgeschlagen um dort auch auf den Attribut- und Bulk-Editor zugreifen zu können. Auch ein Import für OSM-Daten wurde vorgeschlagen was allerdings nicht im Rahmen von "ROSE" liegt, da viele der Daten in ROSE verloren gehen würden und auch keine maßstabsgetreuen Straßennetze dargestellt oder exportiert werden können.

Um ein allgemeines und einfaches Feedback für "ROSE" zu bekommen haben wir zusätzlich eine Auswertung durch den *Net-Promoter-Score* vorgenommen und dabei eine Punktzahl von 66 erzielt. Damit haben wir unser Ziel von 33 Punkten erreicht.

7.7 System Usability Scale

Am Ende der Fragenrunde sollten die Nutzer jeweils eine Note (1 - stimme gar nicht zu, 10 - stimme voll zu) zu sieben Aussagen vergeben. Diese Aussagen stammen aus dem SUS(System Usability Scale), welcher ein Bild über die Nutzbarkeit liefert. Wir haben

SUS gewählt, da es :

- ein Industriestandard ist
- Werkzeuge bietet, mit welchen Nutzbarkeit in Zahlen bewertet werden kann
- eine sehr einfach zu verwaltende Skala für die Teilnehmer hat
- valide ist - es kann effektiv zwischen nutzbaren und unnutzbaren Systemen unterscheiden

Als Erfolgskriterium haben wir festgelegt, dass alle Ergebnisse jeweils im besten Drittel liegen sollen. Also maximal 3 für negative Aussagen und mindestens 7 für positive Aussagen.

	Nutzer 1	Nutzer 2	Nutzer 3	Durchschnitt
1) Ich empfinde das System als unnötig komplex	2	2	3	2,33
2) Ich empfinde das System als einfach zu nutzen.	9	8	8	8,33
3) Ich finde, dass es im System zu viele Inkonsistenzen gibt.	3	1	3	2,33
4) Ich kann mir vorstellen, dass die meisten Leute das System schnell zu beherrschen lernen.	8	10	7	8,33
5) Ich empfinde die Bedienung als sehr umständlich.	3	4	2	3
Ich habe mich bei der Nutzung des Systems sehr sicher gefühlt.	9	6	7	7,33
Ich musste eine Menge Dinge lernen, bevor ich mit dem System arbeiten konnte	1	2	1	1,33

Wie hier ersichtlich wird, konnten wir unser Ziel erreichen und haben allgemein erfreuliche Bewertungsergebnisse erzielen können.

8 Entwurfsänderungen bezüglich des Stands der Implementierungsphase

In diesem Abschnitt werden die Entwurfsänderungen am finalen Stand der Implementierungsphase von "ROSE" festgehalten. Das umfasst Änderungen an öffentlichen Signaturen und hinzugefügte öffentlich sichtbare Klassen. Die an den Implementierungsdetails vorgenommenen Änderungen werden hier nicht beschrieben. Diese sind in den Kapiteln zu behobenen Bugs und Optimierungen beschrieben.

8.0.1 Neuer Konstruktor für RoseProjectController

Dem `RoseProjectController` wird ein neuer Konstruktor `public RoseProjectController (StorageLock storageLock, Navigator navigator, Project project, ApplicationDataSystem applicationDataSystem, long backupInterval, Path backupDirectoryPath)` hinzugefügt. Im Gegensatz zum bestehenden Konstruktor `public RoseProjectController (StorageLock storageLock, Navigator navigator, Project project, ApplicationDataSystem applicationDataSystem)` kann über diesen Konstruktor die automatische Erstellung von Sicherungskopien durch den Sicherungsordner und das Sicherungsintervall konfiguriert werden. Diese Änderung ist notwendig, um die automatische Sicherung des geöffneten Projektes automatisch und effizient testen zu können, da die entsprechenden Testfälle ansonsten fünf Minuten auf eine Sicherung warten müssten.

8.0.2 Verwendung einer Logging-Bibliothek

Es wird die Logging-Bibliothek logback eingebunden und alle Ausgaben an die Konsole werden nun über die *Simple Logging Facade* ausgegeben. Dieser Änderung ist notwendig, um (insbesondere im Rahmen der Nutzertests) ausführliche Fehlermeldungen zu speichern.

8.0.3 Objekt-Datentypen statt primitiven Datentypen für Attribute

Die Datentypen einiger Java-Attribute wurden geändert. Siehe entsprechende Bugs.

8.0.4 Neue Methode `getIsSegmentSelected` für HierarchyController

Dem Interface `RoseHierarchyController` wurde die Methode `boolean getIsSegmentSelected (Segment segment)` hinzugefügt. Sie erlaubt es den Selektionsstatus eines `Segments` nicht nur über das Observer-Pattern mitgeteilt zu bekommen, sondern ihn explizit abzufragen. Diese Änderung war notwendig, da im `HierarchyPanel` manche Objekte den Selektionsstatus eines `Segments` als Konstruktorparameter entgegen nehmen.

8.1 Neue Methode `getRoadSystem` für RoadSystemPanel

Diese Methode sichert die Testbarkeit für GUI Tests.

8.2 Neue Methode `getZoomSetting` für RoadSystemPanel

Diese Methode sichert die Testbarkeit für GUI Tests.

8.2.1 Neuer Konstruktorparameter für SegmentView

Dem Konstruktor `SegmentView(LocalizedTextProvider translator, Segment segment, HierarchyController controller)` wurde ein Parameter vom Typ `RoadSystemController` hinzugefügt. Diese Änderung wird benötigt, da die Methode `RoadSystemController# deleteStreetSegment(Segment segment)` zum Löschen eines `Segments` über das `HierarchyPanel` benötigt wird.

8.2.2 Neue Konstruktorparameter für ElementTreeCell

Dem Konstruktor `public ElementTreeCell(HierarchyController hierarchyController, LocalizedTextProvider translator)` wurde ein Parameter vom Typ `RoadSystemController` und ein Parameter vom Typ `Map<Element, ElementTreeCell>` hinzugefügt. Diese Änderung wird benötigt wegen 8.2.1 und 2.19.

8.2.3 Neuer Konstruktorparameter für DragSegmentEndCommand

Dem Konstruktor `public DragSegmentEndCommand(ReplacementLog replacementLog, MovableConnector segmentEnd, Movement translation)` wurde ein Parameter von

Typ `RoadSystem` hinzugefügt. Diese Änderung hat den Grund, dass beim Redo des `DragSegmentEndCommands` ggf. eine `Connection` konstruiert werden muss.

8.2.4 Neue Konstruktorparameter für `DragStreetSegmentCommand`

Dem Konstruktor `public DragStreetSegmentsCommand(ReplacementLog replacementLog, Project project, List<Segment> segments, Movement movement)` wurde ein Parameter vom Typ `Set<Connection>` und ein Parameter vom Typ `Connector` hinzugefügt. Diese Änderung wird benötigt, da der `DragStreetSegmentCommand` bei Undo und Redo `Connections` konstruieren muss.

8.2.5 Neue Klasse `LocalizedComboBox` hinzugefügt

Die Klasse `LocalizedComboBox` wurde hinzugefügt, um den Code zur Lokalisierung einer `ComboBox` zu kapseln.

8.2.6 Methode `getRootElement` durch Methode `getRootGroup` ersetzt in `RoadSystem`

Die Methode `RoadSystem#getRootElement` wurde hinzugefügt wegen 6.6. Da die Methode `RoadSystem#getRootElement` nicht mehr benötigt wird, wurde sie entfernt.

8.2.7 Neuer Konstruktorparameter für `CreateStreetSegmentCommand`

Es wurde ein Konstruktor mit dem Position parameter angelegt, um die Positionierung eines neu erstellten Straßensegments übergeben zu können. Wird benötigt für 6.3.

9 Codequalität

In diesem Abschnitt werden die Resultate der Qualitätsprüfung des Quellcodes von "ROSE" vorgestellt.

Um die Qualität des Quellcodes von "ROSE" sicherzustellen haben wir das Tool "SonarQube" eingesetzt. Damit haben wir eine statische Codeanalyse des Quellcodes von "ROSE" durchgeführt. SonarQube vergibt Bewertungen zwischen A und E in den Kategorien

- Reliability
- Security
- Security Review
- Maintainability

Dabei ist A die beste und E die schlechteste Bewertung. Des Weiteren werden von SonarQube spezielle Kennzahlen für Codequalität berechnet.

Die Ergebnisse dieser Analyse werden im folgenden vorgestellt.

9.1 Reliability

In dieser Kategorie hat "ROSE" die Bewertung A erhalten, da SonarQube keine potenziellen Bugs gefunden hat. Zuvor hatte SonarQube fünf potenzielle Bugs erkannt, die wir jedoch beheben konnten.

9.2 Security

In dieser Kategorie hat "ROSE" die Bewertung A erhalten, da SonarQube keine potenziellen Schwachstellen finden konnte.

9.3 Security Review

In dieser Kategorie hat "ROSE" die Bewertung A erhalten, da SonarQube keine Codestellen die einen manuellen Sicherheitsreview benötigen (sog. Security-Hotspots) gefunden hat.

9.4 Maintainability

In dieser Kategorie hat "ROSE" die Bewertung A erhalten. Allerdings hat SonarQube in dieser Kategorie 430 "Code-Smells" entdeckt. Dies konnten wir auf 288 "Code-Smells" reduzieren.

9.5 Kennzahlen für Codequalität

Die folgenden Kennzahlen für Codequalität, berechnete SonarQube für "ROSE":

- 1.0% Codeduplizierung gemessen über 11.000 Codezeilen
- 8 duplizierte Codeblöcke
- 432 Unit-Tests

Glossar

Mock Ein Mock-Objekt ist eine Attrappe für eine Instanz einer gegebenen Klasse oder Schnittstelle. In ROSE wird das Framework Mockito verwendet, um solche Attrappen für beliebige Klassen und Schnittstellen zu erstellen. Mockito ermöglicht es zudem, Interaktionen mit den Attrappenobjekten auszulesen und zu analysieren. 8

Net-Promoter-Score Der Net-Promoter-Score ist ein Indikator für die Zufriedenheit von Personen bezüglich eines Produkts. Dabei wird eine Person gefragt: "Wie wahrscheinlich ist es auf einer Skala von 0 bis 10, dass sie das Produkt weiterempfehlen werden?". Um den Net-Promoter-Score zu berechnen wird dann der prozentuale Anteil von Personen zwischen 0 und 7 vom prozentualen Anteil der Personen zwischen 9 und 10 abgezogen. Die resultierende Zahl ist der Net-Promoter-Score. 40

Paretoprinzip Das Paretoprinzip bezeichnet die Beobachtung, dass mit 20 % der Gesamtaufwandes die ersten 80 % des Ergebnisses erzielt werden können und im Gegenzug die letzten 20 % des Ergebnisses 80 % der Arbeit erfordern. 24

Simple Logging Facade Die Simple Logging Facade ist eine Abstraktion von Logging-Bibliotheken, die eine Schnittstelle nach dem Entwurfsmuster "Fassade" bereitstellt. Die Verarbeitung der Log-Nachrichten wird von Logging-Bibliotheken wie logback oder log4j übernommen.. 42