Architekturdokument

Projekt:

Street Sim

Matteo Bentivegna, Sandra Kiefer, Jan Ningelgen, Niklas Schlögel

12.06.2020
Version 1.0

Hochschule RheinMain

Sommersemester 2020

Versionshistorie

Version	Datum	Beschreibung
1.0	12.06.2020	Initiale Erstellung des Dokumentes

1. Einführung und Ziele

1.1 Aufgabenstellung

Zur Unterstützung der Verkehrswende soll ein kleines Simulationssystem programmiert werden. Dieser Verkehrssimulator ermöglicht es ein individuelles Verkehrsnetz aus verschiedenen Straßenabschnitten, wie zum Beispiel Gerade, Kurven und Kreuzungen, in einem grafischen Editor zu erzeugen und platzieren. Des Weiteren können Verkehrsteilnehmer, in Form von Autos, auf diesem Straßennetz platziert und simuliert fahren gelassen werden. Autos können durch ihre Geschwindigkeit individuell angepasst werden. Das Programm verfügt ebenfalls über eine einfache Kollisionserkennung zwischen den verschiedenen Verkehrsteilnehmern. Darüber hinaus sollen am Straßenrand Ampeln platziert werden, welche den Verkehrsfluss regulieren. Werden keine Ampeln platziert, wird der Verkehr über die in der Straßenverkehrs-Ordnung bekannten Vorfahrtsregeln reguliert. Zuletzt kann der Benutzer seinen individuellen Programmstand abspeichern und zu einem späteren Zeitpunkt wieder laden.

Dieses Projekt entsteht im Rahmen der Veranstaltung Softwaretechnik im Studiengang Medieninformatik der Hochschule RheinMain im Sommersemester 2020. Es ist ein Gruppenarbeit die zum Bestehen des Praktikums dient.

Bezogen wird sich auf das Anforderungsdokument, Version 1.2.[1]

[1] https://scm.mi.hs-rm.de/rhodecode/2020swt/2020swt01/StreetSim/files/095620d1751cf0c5998f60990053497beb5b32be/doku/Anforderungsspezifikationen.pdf

1.2 Qualitätsziele

Qualitätsziele werden in folgender Tabelle dargelegt. Sie sind in absteigender Reihenfolge priorisiert gelistet.

Qualitätsziele	**Mengengerüst/Messbarkeit**	**Begründung**
Platzierung von Verkehrsnetz-Bestandteilen	< 1s	Die Platzierung von Straßenabschnitten, Autos und Ampeln muss schnellstmöglich erfolgen, sodass der Benutzer kaum eine Verzögerung merkt. Dieser Anwendungsfall ist einer der Wichtigsten (siehe [1] – S.8) und daher ist es von unbedingter Notwendigkeit, dass dieser Vorgang möglichst verzögerungsfrei erfolgt.
Bedienung der Anwendung	pro Klick Wartezeit < 1s	Dieses Programm soll eine interaktive Anwendung werden. Daher muss es, um die Benutzerfreundlichkeit gewährleisten zu können, möglichst ohne Wartezeiten laufen, da sonst der Benutzer das Gefühl bekommen könnte, dass die Anwendung nicht richtig funktioniert und eventuell das Programm vorzeitig beendet.
Ladezeiten von Speicherständen	< 10s bei mittelgroßer Welt	Wird eine vorher erstellte und gespeicherte Welt geladen, erfordert dies nicht nur das Parsen des Speicherformats, sondern vor allem auch die erneute Instanziierung aller gewünschten Objekte. Die Anwendung erweckt durch den minimal gestalteten Aufbau den Eindruck, dass diese nicht viel Leistung benötigt. Das tut sie auch zur Laufzeit nicht. Hingegen ist das Laden jedoch sehr zeitaufwändig. Ziel ist es daher, den Benutzer nicht länger als 10s auf seine Welt warten zu lassen.

1.3 Stakeholder

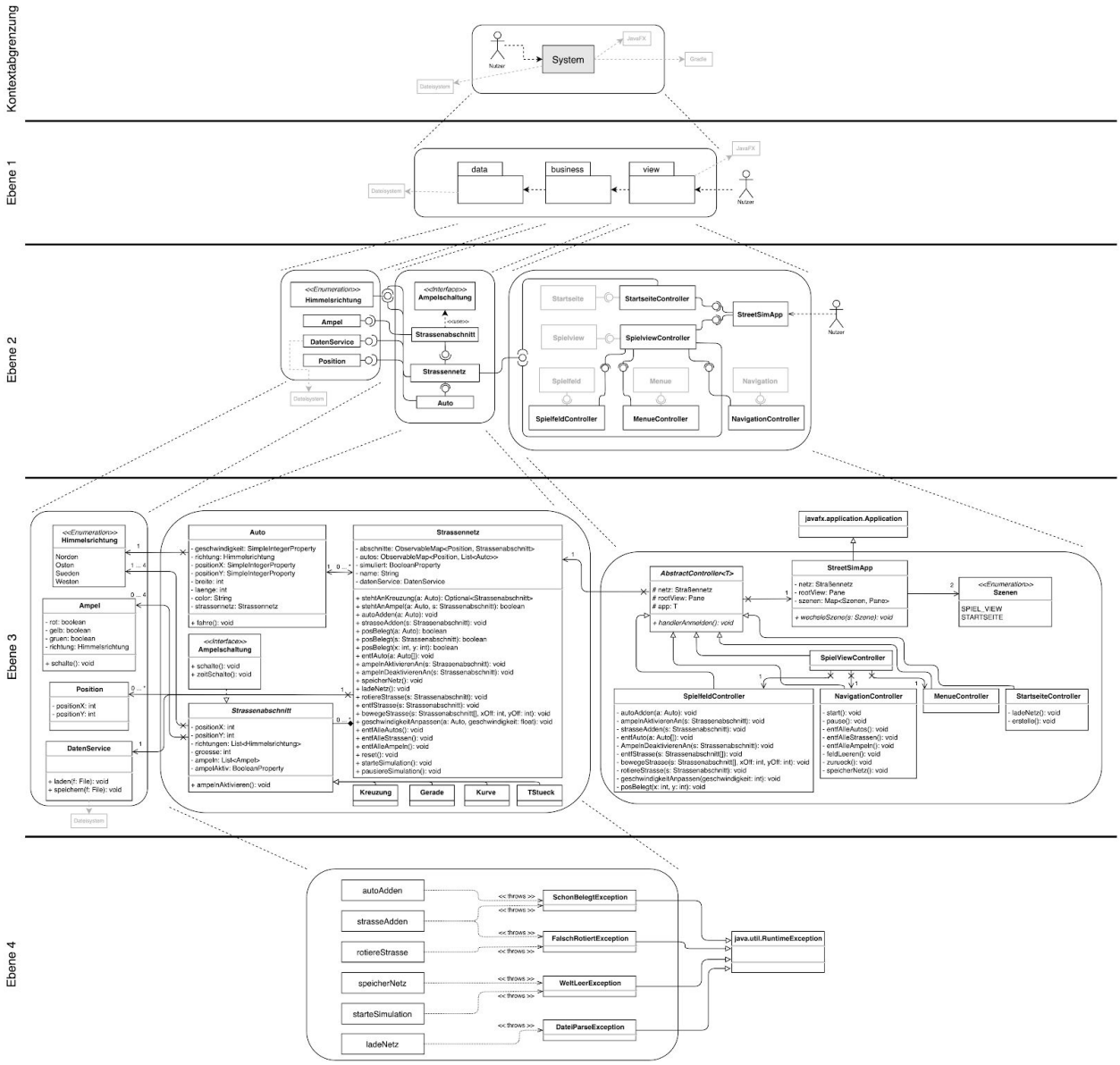
Rolle	Erwartungshaltung
Entwickler	Das Architekturdokument soll den Entwickler bei der Implementierung unterstützen. Entwickler sollen erfahren können, welche Anforderungen erfüllt werden müssen und wie die Struktur des Projektes umgesetzt werden soll.
Auftraggeber	Der Auftraggeber soll durch die Dokumentation prüfen können, ob seine Anforderungen erfüllt werden und das Projekt entsprechend seiner Erwartungen korrekt umgesetzt wird.
Nutzer	Der Nutzer wird insofern berücksichtigt, da auf ihn beispielsweise verschiedene Qualitätsziele abgestimmt sind.

2. Randbedingungen

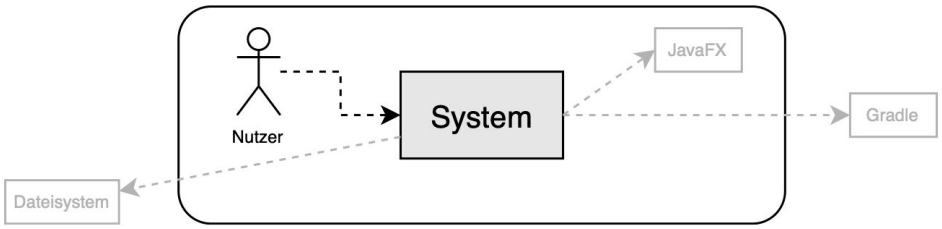
Randbedingung	Hintergrund
Frist 10.07.2020 18:00 CET	Die fristgerechte Bereitstellung der Applikation mit den essentiellen Funktionalitäten ist unverzichtbar. (Anforderungsdokument [1] Abschnitt 2.5 Prioritätsklassen I und II)
Implementierung in Java SE 11	Der Auftraggeber besteht auf die Verwendung von Java SE 11. (https://docs.oracle.com/en/java/javase/11/)
OpenJFX	Der Auftraggeber besteht auf die Verwendung von JavaFX https://openjfx.io/javadoc/11/
Verwendung von Gradle	Die Applikation muss unter Verwendung des Build Tools Gradle der Version 5.1 oder höher eingerichtet und entwickelt werden. (https://docs.gradle.org/5.1/release-notes.html)
Moderate Hardware	Die Applikation muss sich auf einem marktüblichen Standard-Laptop ausführen lassen.

3. Bausteinsicht

Hier wird die angestrebte Programmstruktur gezeigt. Die Darstellung wird je Ebene detaillierter und wird auf den folgenden Seiten ausführlicher beschrieben.



3.1 Gesamtsystem



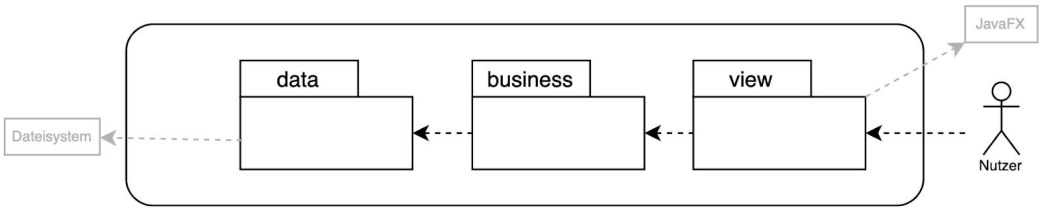
Beschreibung

Das System, welches der Nutzer benutzen soll, verwendet für die grafische Benutzeroberfläche das Framework JavaFX, zur Build-/Testautomatisierung und Paketverwaltung Gradle und zum Laden und Speichern von Daten das Dateisystem als benachbarte Systeme.

Enthaltene Bausteine

Name	**Verantwortung**
System	Vollständige Applikation, umfasst jegliche Funktionalitäten
JavaFX	Framework zur Realisierung der grafischen Benutzungsoberfläche
Gradle	Paketmanager zur Verwaltung von externen Dependencies und zur Build-/Testautomatisierung
Dateisystem	Speichern und Laden von Dateien

3.2 Ebene 1



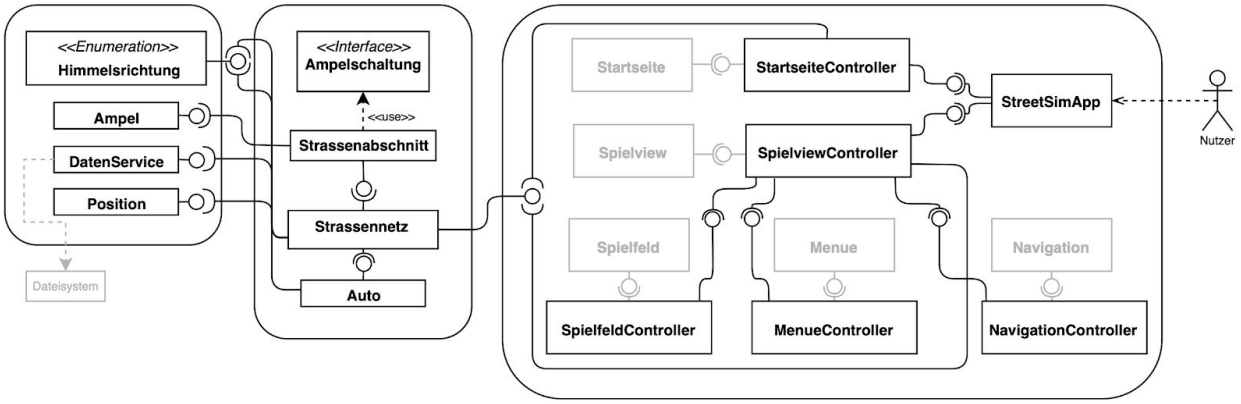
Beschreibung

Ebene 1 repräsentiert die 3-Schichten-Architektur des Systems, bestehend aus den Whiteboxen „data“, „business“ und „view“. So kann das System klar in verschiedene Abstraktionsebenen gegliedert werden und macht, für mögliche Grundsatz-Änderungen in ferner Zukunft, einzelne Schichten leichter austauschbar.

Enthaltene Bausteine

Name	**Verantwortung**
data	Applikationsschicht welche für das Verwalten der Dateien und reinen Daten-Klassen zuständig ist
business	Applikationsschicht welche für die Berechnungen zur Laufzeit, dem Ablauf der Anwendung und Weiteres zuständig ist
view	Applikationsschicht welche für die grafische Darstellung der aus der Business-Schicht resultierenden Informationen zuständig ist

3.2 Ebene 2



Beschreibung

Ebene 2 veranschaulicht die Beziehungen zwischen den einzelnen Bausteinen durch Komponentendiagramme. Der Nutzer greift lediglich durch eine zentrale Komponente der Benutzeroberfläche auf die Anwendung zu und kann somit mit dem System interagieren.

Enthaltene Bausteine

data

Name	Verantwortung
Himmelsrichtung	Abbildung der Himmelsrichtungen
Ampel	Abbildung einer einzelnen Ampel mit den passenden Zuständen (Rot, Gelb, Grün, An-/Ausgeschaltet)
DatenService	Speichern und Laden von Daten
Position	Abbildung der Position von den X und Y-Koordinaten

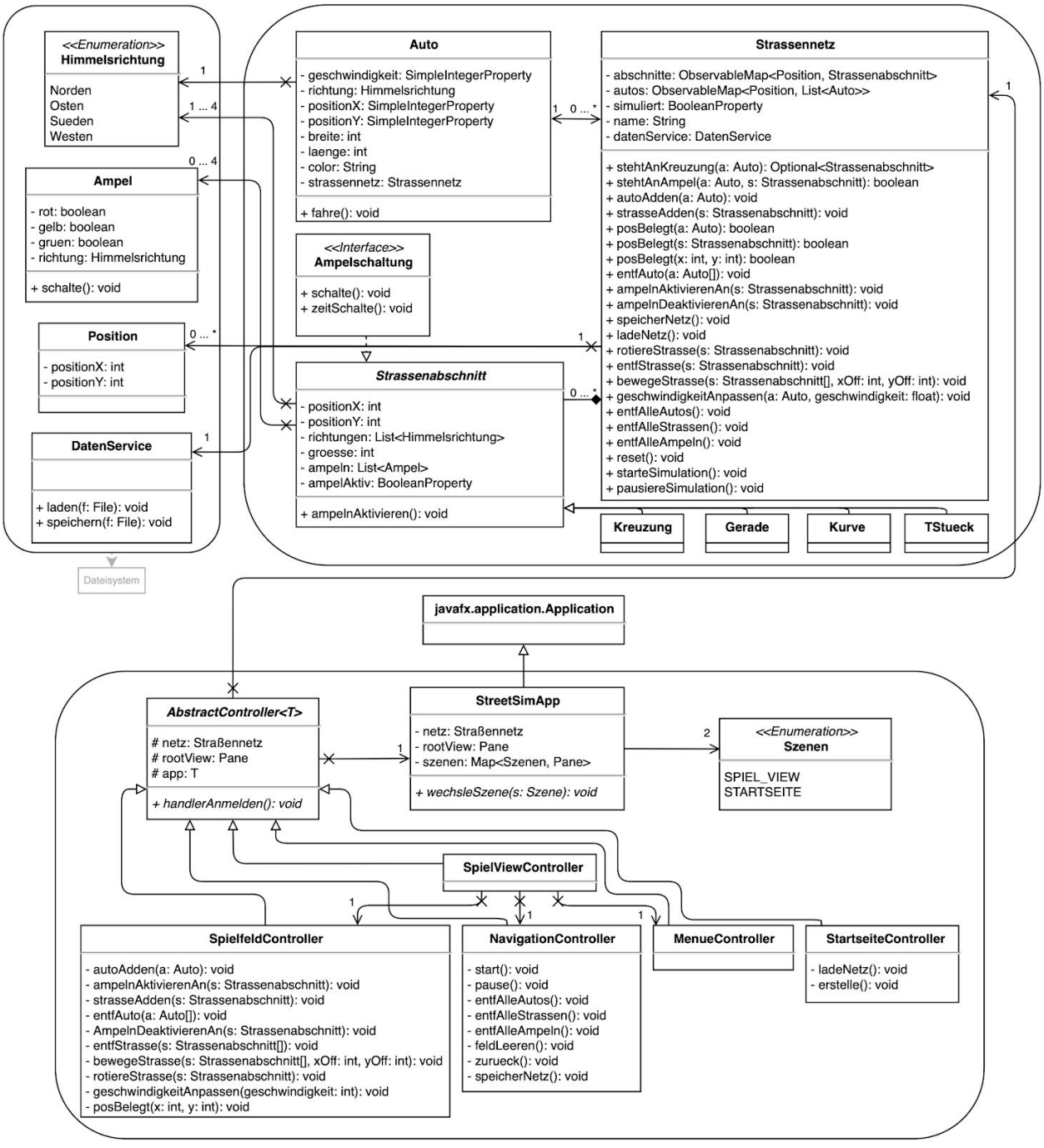
business

Name	Verantwortung
Ampelschaltung	Korrekte Schaltung mehrerer Ampeln
Strassenabschnitt	Verwaltung der Richtungen in welche ein Straßenabschnitt führt, verwendet Ampelschaltung für optionale Ampeln
Strassennetz	Verwaltung aller Straßenabschnitte und Autos sowie Großteil der Anwendungslogik und Schnittstelle für obere Schicht („view“)
Auto	In eine Himmelsrichtung sich fortbewegendes Objekt welches auf das Straßennetz, insbesondere Ampeln, reagiert

view

Name	Verantwortung
StreetSimApp	Verwaltung der gesamten Applikation
StartseiteController	Verwaltung von Aktionen auf der Startseite
SpielviewController	Verwaltung der folgenden drei Controller (SpilefeldController, MenueController, NavigationController)
SpielfeldController	Verwaltung von Aktionen auf dem Spielfeld
MenueController	Verwaltung von Aktionen im Menü
NavigationController	Verwaltung von Aktionen in der Navigationsleiste

3.3 Ebene 3


<<Enumeration>>
Himmelsrichtung
Norden
Osten
Sueden
Westen
Auto
- geschwindigkeit: SimpleIntegerProperty
- richtung: Himmelsrichtung
- positionX: SimpleIntegerProperty
- positionY: SimpleIntegerProperty
- breite: int
- laenge: int
- color: String
- strassennetz: Strassennetz
+ fahre(): void
Strassennetz
- abschnitte: ObservableMap<Position, Strassenabschnitt>
- autos: ObservableMap<Position, List<Auto>>
- simuliert: BooleanProperty
- name: String
- datenService: DatenService
+ stehtAnKreuzung(a: Auto): Optional<Strassenabschnitt>
+ stehtAnAmpel(a: Auto, s: Strassenabschnitt): boolean
+ autoAdden(a: Auto): void
+ strasseAdden(s: Strassenabschnitt): void
+ posBelegt(a: Auto): boolean
+ posBelegt(s: Strassenabschnitt): boolean
+ posBelegt(x: int, y: int): boolean
+ entfAuto(a: Auto[]): void
+ ampelnAktivierenAn(s: Strassenabschnitt): void
+ ampelnDeaktivierenAn(s: Strassenabschnitt): void
+ speicherNetz(): void
+ ladeNetz(): void
+ rotiereStrasse(s: Strassenabschnitt): void
+ entfStrasse(s: Strassenabschnitt): void
+ bewegeStrasse(s: Strassenabschnitt[], xOff: int, yOff: int): void
+ geschwindigkeitAnpassen(a: Auto, geschwindigkeit: float): void
+ entfAlleAutos(): void
+ entfAlleStrassen(): void
+ entfAlleAmpeln(): void
+ reset(): void
+ starteSimulation(): void
+ pausiereSimulation(): void
Ampel
- rot: boolean
- gelb: boolean
- gruen: boolean
- richtung: Himmelsrichtung
+ schalte(): void
<<Interface>>
Ampelschaltung
+ schalte(): void
+ zeitSchalte(): void
Position
- positionX: int
- positionY: int
Strassenabschnitt
- positionX: int
- positionY: int
- richtungen: List<Himmelsrichtung>
- groesse: int
- ampeln: List<Ampel>
- ampelAktiv: BooleanProperty
+ ampelnAktivieren(): void
DatenService
+ laden(f: File): void
+ speichern(f: File): void
Kreuzung
Gerade
Kurve
TStueck
Dateisystem
javafx.application.Application
AbstractController<T>
netz: Straßennetz
rootView: Pane
app: T
+ handlerAnmelden(): void
StreetSimApp
- netz: Straßennetz
- rootView: Pane
- szenen: Map<Szenen, Pane>
+ wechsleSzene(s: Szene): void
<<Enumeration>>
Szenen
SPIEL_VIEW
STARTSEITE
SpielViewController
SpielfeldController
- autoAdden(a: Auto): void
- ampelnAktivierenAn(s: Strassenabschnitt): void
- strasseAdden(s: Strassenabschnitt): void
- entfAuto(a: Auto[]): void
- AmpelnDeaktivierenAn(s: Strassenabschnitt): void
- entfStrasse(s: Strassenabschnitt[]): void
- bewegeStrasse(s: Strassenabschnitt[], xOff: int, yOff: int): void
- rotiereStrasse(s: Strassenabschnitt): void
- geschwindigkeitAnpassen(geschwindigkeit: int): void
- posBelegt(x: int, y: int): void
NavigationController
- start(): void
- pause(): void
- entfAlleAutos(): void
- entfAlleStrassen(): void
- entfAlleAmpeln(): void
- feldLeeren(): void
- zurueck(): void
- speicherNetz(): void
MenueController
StartseiteController
- ladeNetz(): void
- erstelle(): void

Beschreibung

Ebene 3 stellt die gesamte Architektur detailliert als Klassendiagramm dar. Inklusive den Beziehungen zwischen den einzelnen Klassen und für die Anwendungslogik relevanten Methoden, sowie Attribute.

Hinzu kommen die Realisierungen von abstrakten Klassen und Entitäten der Relationen.

Enthaltene Bausteine

data

Himmelsrichtungen

Aufzählungstyp welche die Ausrichtungsmöglichkeiten: Norden, Osten, Süden, Westen, für Straßenabschnitte und Autos definiert.

Ampel

Standardmäßiger Aufbau einer Ampel-Klasse mit den drei möglichen Zuständen rot, gelb und grün. Beinhaltet Funktion „schalte" mit der zwischen den verschiedenen Zuständen geschaltet werden kann.

Position

Konstruiert ein Positionsobjekt mit den übergebenen X und Y-Koordinaten und rundet die Werte ab, um sie zu den Startpunkten eines Rasters zuzuordnen. Position überschreibt die „equals" und „hashCode"-Methode, so dass Objekte korrekt voneinander unterschieden werden können und eindeutige Zugriffe als Key in einer Map (wird im Straßennetz benötigt) möglich sind.

DatenService

Bringt die Funktionalität zum Speichern und Laden von Entwürfen mit Hilfe der Methoden „laden(f: File)" und „speichern(f: File)". Die Methoden kriegen über den systemabhängigen Dateimanager die zu ladende oder speichernde Datei.

business

Auto

Definiert Auto-Objekte, welche durch die Funktion „fahre” eigenständig in die mögliche Himmelsrichtung fahren kann und auf Ampeln sowie weitere Verkehrsteilnehmer reagieren kann. Dafür wird für jedes Auto ein eigener Thread gestartet. Durch Properties werden Veränderungen an der Geschwindigkeit und Position des jeweiligen Autos für Listener sichtbar. Jedes Auto hat eine Referenz auf ein Straßennetz, um Abbiegemöglichkeiten und Ampeln zu erkennen. Des weiteren kann für jedes Auto eine eigene Länge, Breite und Farbe definiert werden.

Ampelschaltung

Ein Interface welches die Implementierung von „schalte” und „zeitSchalte” vorsieht.

Die Methode „schalte” soll alle Ampeln auf dem aktuellen Straßenabschnitt umschalten.

Die Methode „zeitSchalte” soll nach einem festgelegten Intervall alle Ampeln auf dem aktuellen Straßenabschnitt umschalten (führt „schalte“ aus).

Straßenabschnitt

Eine abstrakte Klasse, welche gemeinsame Funktionalitäten für verschiedene Straßenformen implementieren soll. Sie hält als Attribute seine Position im Spielfeld (als X und Y-Koordinaten), eine Liste von möglichen Richtungen in welche ein Auto fahren kann, die Größe des Abschnitts, eine Liste von Ampeln, die entsprechend der Realisierung der Unterklassen lang ist und ein BooleanProperty, welches kennzeichnet ob die Ampeln an dem Straßenabschnitt aktiviert und geschaltet werden sollen oder nicht.

Die Methode „ampelnAktivieren“ aktiviert die Ampeln in der Liste und die startet die „zeitSchalt“-Methode des Ampelschaltung-Interface.

Kreuzung

Eine Kreuzung ist eine Realisierung eines Straßenabschnitts. Sie hält maximal vier Ampel-Instanzen in der Liste und hat vier mögliche Straßenrichtungen.

Gerade

Eine Gerade ist eine Realisierung eines Straßenabschnitts. Sie hält maximal zwei Ampel-Instanzen in der Liste und hat zwei mögliche Straßenrichtungen.

Kurve

Eine Kurve ist eine Realisierung eines Straßenabschnitts. Sie hält maximal zwei Ampel-Instanzen in der Liste und hat zwei mögliche Straßenrichtungen.

TStueck

Ein T-Stück ist eine Realisierung eines Straßenabschnitts. Es hält maximal drei Ampel-Instanzen in der Liste und hat drei mögliche Straßenrichtungen.

Straßennetz

Das Straßennetz ist die zentrale Instanz des Models, welche sämtliche Logik implementiert und zur Kommunikation mit den Controllern durchgereicht wird.

Die Map „abschnitte“ dient zur Überprüfung, ob an einer gewissen Position schon ein Straßenabschnitt enthalten ist.

Die Map „autos“ dient zur Überprüfung, ob Autos an einer bestimmten Position platziert sind. Da mehrere Autos auf einer Position sein können (z.B. bei Gegenverkehr oder ganz dichtem Auffahren ohne Einhaltung der Abstandsregelung), ist der Value der Map eine Liste von Autos.

Das Attribut „simuliert“ kennzeichnet, ob aktuell die Simulation läuft (true) oder ob die Simulation pausiert ist (false).

Das Attribut „name“ kennzeichnet den Namen eines Straßennetzes. Dieses wird initialisiert, sobald ein Spielstand abgespeichert wird (als <Name>.json).

Der DatenService unterstützt das Straßennetz bei Dateisystemzugriffen wie Lade- und Speichervorgänge.

Die Methode „stehtAnKreuzung” gibt ein Straßenabschnitt zurück, falls das übergebene Auto an einem Straßenabschnitt steht, an dem es in eine andere Richtung abbiegen kann. Das Auto wählt dann zufällig aus den möglichen Abbiegerichtungen seine neue Fahrtrichtung.

Die Methode „stehtAnAmpel“ gibt zurück, ob ein Auto an einer Ampel steht, oder nicht.

Die Methode „autoAdden“ fügt ein Auto zum Straßennetz hinzu (zur autos-Map). Sofern schon ein Auto auf dem Straßennetz mit der selben Position und der selben Richtung existiert, wird eine „SchonBelegtException“ geworfen. Der Vorgang kann dann mit einer anderen Richtung oder Position wiederholt werden (siehe [1] – Anwendungsfall I-C1).

Die Methode „strasseAdden“ fügt ein Straßenabschnitt-Objekt zum Straßennetz hinzu (zur abschnitte-Map). Ist an der Position, an der das Straßennetz versucht wird zu positionieren, bereits ein anderer Straßenabschnitt platziert, wird eine „SchonBelegtException” geworfen. Ist ein weiterer Straßenabschnitt direkt neben dem Neuen und Letzteres bildet keinen Straßenfluss mit seinem direkten Nachbarn, wird eine FalschRotiertException geworfen, sofern der Nachbar ein offenes Straßenende hat (siehe [1] – Anwendungsfall I-B1).

Die Methoden „posBelegt“ prüfen, je nach Parameter, ob die Position des Parameters Auto oder Straßenabschnitt schon durch ein anderes gleiches Objekt belegt ist oder nicht. Die Methode mit den Parametern x und y dient zur schnellen Überprüfung einer freien Position ohne das Instanzen von Auto oder Straßenabschnitt erstellt werden müssen.

Die Methode „entfAuto“ entfernt beliebig viele Autos vom Straßennetz (siehe [1] – Anwendungsfall I-C2).

Die Methode „ampelnAktivierenAn“ aktiviert an einem gegebenen Straßenabschnitt die Ampeln und setzt dort das Attribut „ampelAktiv” auf true, wodurch die Methode „zeitSchalte“ gestartet wird, welche das automatische Schalten von Ampeln übernimmt (siehe [1] – Anwendungsfall II-B1).

Die Methode „ampelnDeaktiverenAn” bekommt einen Straßenabschnitt übergeben und setzt die BooleanProperty „ampelAktiv” im gegebenen Straßenabschnitt auf false um. Damit werden alle Ampeln auf dem Straßenabschnitt deaktiviert (siehe [1] – Anwendungsfall II-B2).

Die Methode „speicherNetz” speichert das aktuelle Straßennetz im Dateisystem mit dem gegebenen Namen (Attribut name) ab (siehe [1] – Anwendungsfall II-C1).

Die Methode „ladeNetz” versucht ein Straßennetz aus einer Datei zu laden. Falls die Datei nicht geparsed werden konnte wird eine Exception geworfen, die dem User mitteilt, dass die Datei nicht gelesen werden konnte (siehe [1] – Anwendungsfall II-C2).

Die Methode „rotiereStrasse” bekommt einen Straßenabschnitt übergeben und versucht diesen um 90 Grad im Uhrzeigersinn zu rotieren (entsprechend eine Himmelsrichtung weiter). Falls nach der Rotation keine Verknüpfung mit einem anderen Straßenabschnitt bestimmt werden konnte, wird eine Exception geworfen, die dem User mitteilt, dass die Rotation den Straßenlauf nicht fortführt (siehe [1] – Anwendungsfall I-B3).

Die Methode „entfStrasse” bekommt beliebig viele Straßenabschnitte übergeben um sie zu entfernen. Bevor jedoch die Straßenabschnitte entfernt werden muss noch überprüft werden ob sich Autos auf den jeweiligen Straßenabschnitten befinden. Falls ja, müssen erst alle Autos entfernt werden und im Anschluss wird der Straßenabschnitt entfernt (siehe [1] – Anwendungsfall I-B2).

Die Methode „bewegeStrasse” versucht beliebig viele Straßenabschnitte zu verschieben, wofür sie zu den Straßenabschnitten, zusätzlich noch den zu verschiebenden Wert (Offset) für die X und Y-Koordinaten bekommt. Bevor jedoch verschoben werden darf, muss geprüft werden ob sich Autos auf den jeweiligen Straßenabschnitten befinden. Falls ja, muss die Position dieser Autos ebenfalls angepasst werden (siehe [1] – Anwendungsfall I-B4).

Die Methode „geschwindigkeitAnpassen” passt die Geschwindigkeit eines Autos an, welches mit einem gegebenen float-Attribut übergeben wurde. Der Wert des float-Attributs kann in dem Intervall zwischen 0 und 1 liegen (siehe [1] – Anwendungsfall III-A).

Die Methoden „entfAlleAutos”, „entfAlleStrassen” und „entfAlleAmpeln” entfernen alle zu den Methoden gehörigen Elemente vom Straßennetz. Auch hier gilt: Erst müssen alle Autos und Ampeln entfernt werden, bevor alle Straßen entfernt werden können.

Die Methode „reset” dient dazu die Anwendung beim Verlassen der geladenen Welt in den Ausgangszustand zu setzen.

Die Methode „starteSimulation” setzt das Attribut „simuliert” auf true und startet somit die Simulation.

Die Methode „pausiereSimulation” setzt das Attribut „simuliert” auf false und pausiert somit die Simulation.

view

StreetSimApp

JavaFX-Applikation, die vom Nutzer gestartet wird. Hier findet sämtliche Interaktion mit dem System statt. Diese Klasse hält eine Straßennetz-Instanz, um sie den AbstractControllern (siehe unten) mitzugeben, speichert seine eigene aktuelle Root-Ansicht (Pane) und hält eine Map, die mit einem Szenen-Aufzählungstyp alle verfügbaren Ansichten hält, die über die Methode „wechsleSzene” gewechselt werden kann.

Szenen

Ein Aufzählungstyp, der die beiden möglichen Ansichten beschreibt: Startseite(STARTSEITE) und Spielfeld-Ansicht (SPIEL_VIEW).

AbstractController<T>

Eine abstrakte Klasse, die eine Straßennetz-Instanz, eine Pane, die dem Nutzer angezeigt wird und eine JavaFX-Application-Instanz (T entsprechend des Namen der Applikationsklasse) hält. Alle erbenden Controller sollen eine Methode „handlerAnmelden” implementieren, welche durch GUI-Aktionen entsprechende Methoden im Model (Straßennetz) anstoßen.

Die abgebildeten Methoden der folgenden Unterklassen (welche in der Methode „handlerAnmelden” GUI-Elemente angemeldet werden müssen) stoßen über die Straßennetz-Instanz gleichnamige Methoden im Model an.
(Genauere Beschreibung siehe ab S. 14)

StartseiteController

Eine Controller-Klasse, die von AbstractController erbt, welche für die Behandlung von Aktionen auf der Startseite zuständig ist.

Die Methode „ladeNetz" stößt die gleichnamige Methode in der business-Schicht an und lädt ein Straßennetz.

Die Methode „erstelle" erzeugt in der Startseite eine neue Straßennetz-Instanz und wechselt die View zu „SPIEL_VIEW".

SpielViewController

Eine Controller-Klasse, die von AbstractController erbt, die die Teilcontroller Spielfeld-, Navigation- und MenueController zusammenfasst, und somit über einen gemeinsamen Controller die Gesamtview und die gesamte Logik für alle drei Teilcontroller bereitstellt.

Die Gliederung ergibt sich aus den Wireframes des Anforderungsdokumentes (siehe [1] – S. 24), aus welchen ersichtlicherweise mindestens 3 Komponenten abgeleitet werden können.

SpielfeldController

Eine Controller-Klasse, die von AbstractController erbt, welche Interaktionen auf dem Spielfeld verarbeitet und an das Model weitergibt.

NavigationController

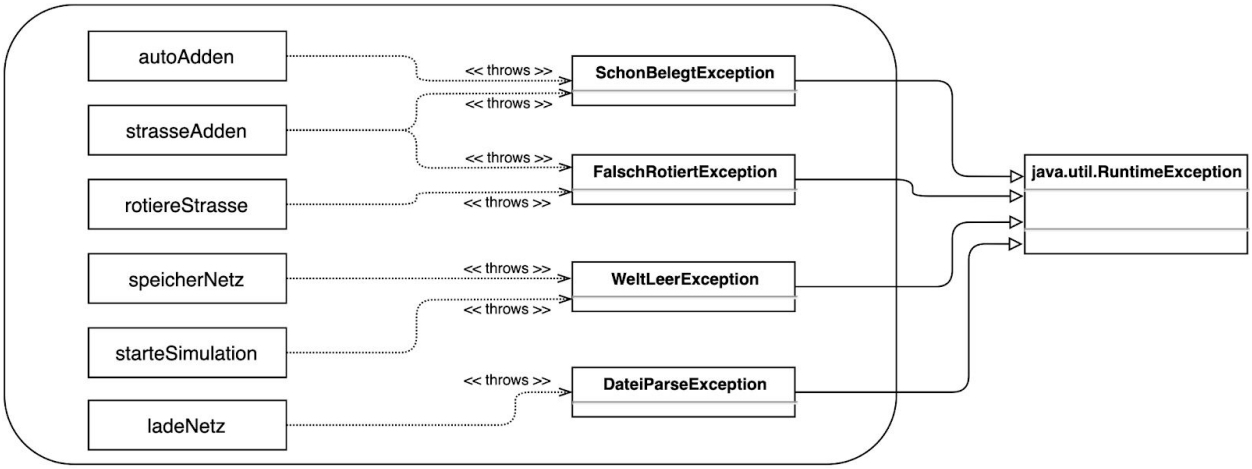
Eine Controller-Klasse, die von AbstractController erbt, welche Interaktionen in der Navigationsleiste (links unten laut Wireframes) verarbeitet und an das Model weitergibt.

Die Methode „zurueck" dient zum Wechsel der Spielview zur Startseite. Gleichzeitig wird die Welt geleert.

MenueController

Eine Controller-Klasse, die von AbstractController erbt, welche für die Behandlung von GUI Aktionen in dem Seitenmenü (rechts laut Wireframes) zuständig ist und selbst nicht mit dem Model kommuniziert.

3.4 Ebene 4



Beschreibung

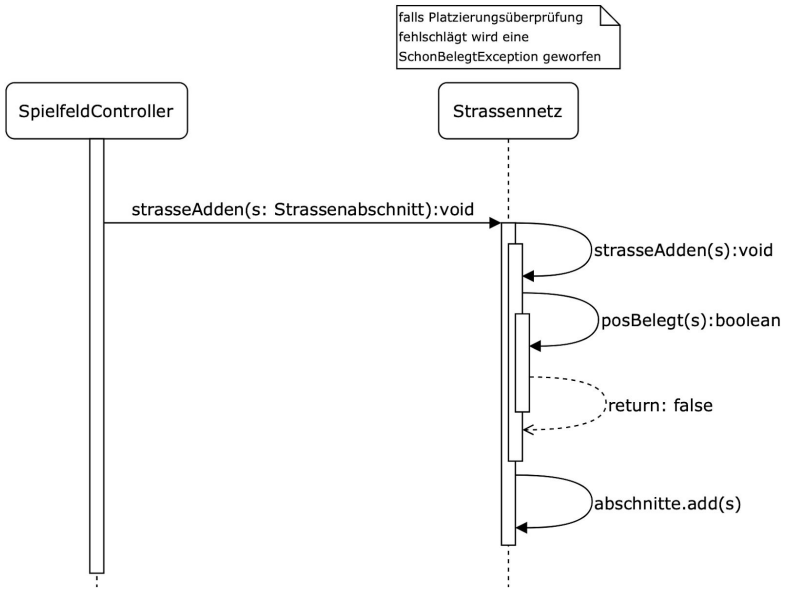
Ebene 4 betrachtet sich genauer die Methoden, die spezielle RuntimeExceptions werfen können.

Enthaltene Bausteine

SchonBelegtException	Wird beim Hinzufügen eines Straßenabschnitts oder Autos auf belegter Position geworfen.
FalschRotiertException	Wird beim Rotieren eines Straßenabschnitts geworfen, wenn die angrenzenden Straßenabschnitte nicht verknüpft werden können.
WeltLeerException	Wird beim Speichern eines Entwurfs oder beim Starten einer Simulation geworfen, wenn noch keine Elemente ins Straßennetz hinzugefügt worden sind (Straßennetz ist leer).
DateiParseException	Wird beim Laden eines Straßennetzes geworfen, falls eine korrupte/nicht parsebare Datei ausgewählt wurde.

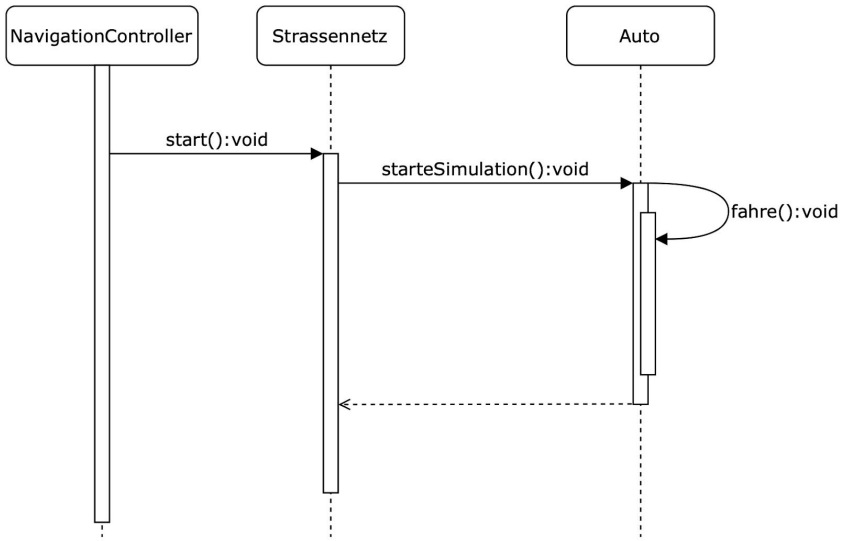
4. Laufzeitsicht

4.1 Neues Platzieren eines Straßenstücks auf einem leeren Feld mit Platzierungsüberprüfung



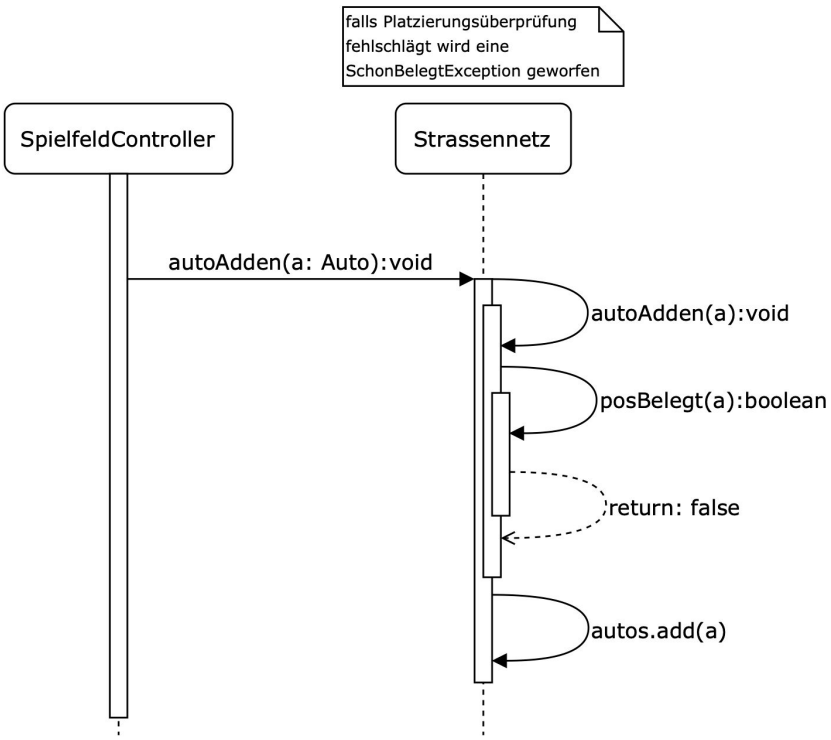
Sobald ein Straßenabschnitt platziert werden soll, wird bereits ein passendes Objekt des korrekten Untertyps mit entsprechender Position erstellt. Der SpielfeldController reicht dieses Objekt ins Straßennetz hinein, welches anhand der Position leicht herausfinden kann, ob sich bereits ein Abschnitt mit gleicher Position im Straßennetz befindet. Glückt diese Prüfung, bzw. existiert kein Abschnitt an selber Position, so wird der Abschnitt im Straßennetz entsprechend gespeichert. Da die Map, die die Straßenabschnitte verwaltet, „observable“ ist, wird der Controller in der UI entsprechend benachrichtigt und kann so die Ansicht anpassen.

4.2 Bewegen des Autos um einen Simulationsabschnitt weiter mit Überschreiten einer Straßenabschnitt-Grenze



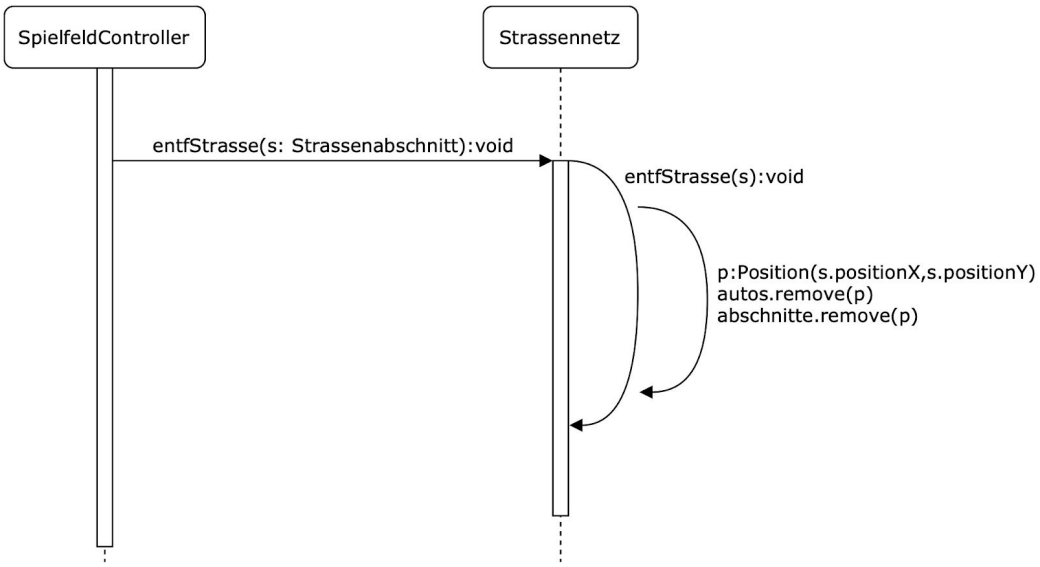
Bevor die Simulation überhaupt erst gestartet wird, werden für jedes hinzugefügte Auto ChangeListener auf deren PositionX und PositionY Properties gesetzt. Dies erlaubt es dem Straßennetz, nachdem die Simulation gestartet und in diesem Zuge jedes Auto mittels eines eigenen Threads herumfährt, bei jeder Änderung zu prüfen ob sich das Auto nun in einem anderen Abschnitt befindet. Dies geschieht durch das Mapping der Auto Koordinaten auf die passende „Ecke“. Dies ist die Position des zugehörigen Straßenabschnitts. Über den alten Wert lässt sich die alte Position ermitteln. Stimmen sie nicht überein, wird die Zugehörigkeit entsprechend angepasst. Da die Positionswerte des Autos „observable“ sind wird die GUI bei ändern der Werte benachrichtigt und kann so seine Anzeige anpassen.

4.3 Platzieren eines Autos mit notwendiger Überprüfung



Nachdem der Nutzer ein Auto ausgewählt und an gewünschter Stelle platziert hat, erfolgt eine Instanziierung mit jeweiligen Werten (Position) und die Instanz wird an das Straßennetz weitergereicht. Dieses Prüft nun anhand der Position ob sich an dieser Stelle ein Straßenabschnitt befindet und ob die Position zulässig ist. Sind die Bedingungen erfüllt wird das Auto dem Straßennetz hinzugefügt. Da die Map, die die Autos verwaltet „observable“ ist, wird die UI benachrichtigt, und kann somit das neue Element direkt anzeigen.

4.4 Löschen eines Straßenabschnitts auf dem ein Auto steht



Nachdem der Nutzer den zu löschenden Straßenabschnitt ausgewählt und entsprechende Aktion ausgeführt hat, wird der Abschnitt in das Straßennetz hereingereicht welches dann das Entfernen durchführt. Da die Positionen von Autos auf Eckpunkte von Straßenabschnitten gemappt werden um sie in Listen verwalten zu können, kann einfach die zu der Position des Abschnittes gehörende Liste geleert bzw. entfernt werden. Danach wird der Abschnitt entfernt, und da die Map, die die Straßenabschnitte verwaltet „observable“ ist, kann somit auch die Ansicht aktualisiert werden und das Straßenstück auch visuell entfernt werden.

Glossar

Begriff	Definition / Erklärung
directions	Himmelsrichtungen der Teilabschnitte für welche auf einem Straßenabschnitt eine Straße existiert Beispiele: Kreuzung ┼ (Norden, Osten, Süden, Westen), Rechtskurve ┌ (Süden, Osten)
Navigationsleiste	Die Navigationsleiste ist das kleine Menü, das sich unten links in der GUI befinden soll, worüber die Simulation gestartet/pausiert werden kann, das Spielfeld geleert werden kann und das Spiel gespeichert und/oder verlassen werden kann
Simulation	Der Vorgang in der Anwendung, in der automatisiert Autos fahren auf einem Straßennetz
Spielfeld	Das Spielfeld ist der Bereich, auf dem Elemente wie Straßenabschnitte, Autos und Ampeln platziert werden können und worauf die Simulation abläuft. Es ist der große mittlere Teil in der Benutzeroberfläche
Straßenabschnitt	Ein quadratischer normierter Abschnitt eines Straßennetzes
Straßennetz	Ein Netz aus verschiedenen Straßenabschnitten
Verkehrsteilnehmer (Auto)	In diesem Anwendungsfall zählen nur Autos als Verkehrsteilnehmer