**Fortgeschrittenenprojekt**

**Wirtschaftsinformatik (SS 2016)**

zum Thema

Simulation des Fahrverhaltens an einer Ampelkreuzung

eingereicht bei

Prof. Dr. rer. nat. Jörg Philipp Müller,

# Dipl.-Inf. Philipp Kraus.

## Institut für Informatik

Technische Universität Clausthal

von

*Nina Trilck, Matrikelnummer:*

*Studienrichtung: Wirtschaftsinformatik*

*Nikita Maslov, Matrikelnummer:*

*Studienrichtung: Wirtschaftsinformatik*

*Tran Khac Dat, Matrikelnummer: 410401*

*Studienrichtung: Wirtschaftsinformatik*

Semester: SS 2016

Datum: 22.09.2016

Inhaltsverzeichnis

1. **Problemstellung** ………………………………………………………………...02

1. Das Thema wir gewählt haben ..........................................................................02

2. Die Motivation für dieses Thema ......................................................................02

3. Die Fragen mit diesem Thema sollten beantwortet werden ………………......03

1. **Lösungsideen und Anforderungen** …………………………………………....03

1. Die Lösungsideen wir diskutiert haben .............................................................03

2. Die Entscheidungen für Ideen ...........................................................................04

3. Gründe für diese Entscheidungen ......................................................................04

1. **Architektur und Entwurf** …………………………………………...................04
2. **Realisierung** ……………………………………………………………….…....05

1. Die Umsetzung (Algorithmus) der Idee ............................................................05

2. Die Probleme während der Umsetzung .............................................................05

1. **Evaluierung** …………………………………………………………………......06

1. Die Kriterien, um die Umsetzung zu überprüfen ..............................................06

2. Die Erwartung für die Ergebnisse .....................................................................06

3. Die Ergebnisse wir ermittelt haben ...................................................................07

4. Die Interpretation dieser Ergebnisse ………………………………………….07

1. **Diskussion und Fazit**……………………………………………………………08
2. Dinge, die wir persönlich gelernt haben ……………………………………..08
3. Was würden wir im nächsten Mal anders machen? …………………………09

**I. Problemstellung**

1. Das Thema wir gewählt haben

- Simulation des Fahrverhaltens an einer Ampelkreuzung

2. Die Motivation für dieses Thema

- Vorerst Grundmodell

- Stau ist ein Problem

- Anhand der Simulation Bewertung potenzieller Stausituationen

- Erweitern die Funktionen in der Zukunft wie z.B: Anzahl der Autos, Parameter von Geschwindigkeit eingeben

3. Die Fragen mit diesem Thema sollten beantwortet werden

- Wie ist Verhalten der Fahrzeuge in einer Kreuzung?

- Wann kann ein Stau auftauchen?

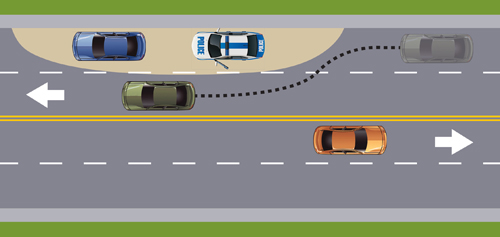
- Wovon ist ein Stau abhängig?

**II. Lösungsideen und Anforderungen**

1. Die Lösungsideen wir diskutiert haben

- Car-following model (Nagel-Schreckenberg-Modell): Das ist ein theoretisches Modell, um [Straßenverkehr](https://de.wikipedia.org/wiki/Stra%C3%9Fenverkehr) zu simulieren. Mit diesem Modell kann man das Phänomen: [Stau aus dem Nichts](https://de.wikipedia.org/wiki/Stau_aus_dem_Nichts) (auch Phantomstau – ein Verkehrsstau, der ohne objektive Überlastung des Verkehrswegs entsteht) erklären. Wenn man Sicherheitsabstände zwischen Autos nicht einhalten, dann kann Stau passieren.

- Car lane changing model: Mit diesem Modell kann man seine Autobahn wechseln. Aber es ist auch komplizieter im unseren Projekt zu verwenden.

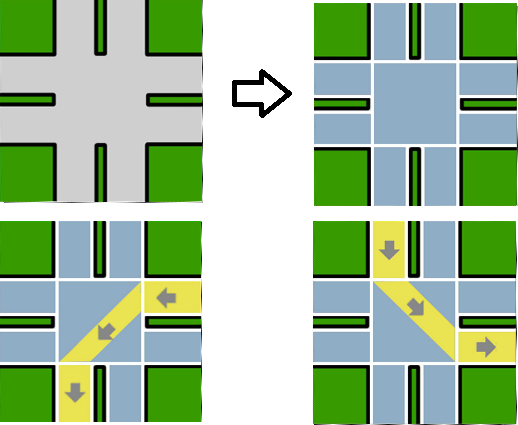


*Quelle: www.servicenl.gov.nl.ca*

- Über Multi-Threading und ThreadPool haben wir auch diskutiert. Mit diesen Methoden können wir Nebenläufigkeit verwenden, um die Autos und auch die Ampeln gleichzeitig zu steuern.

- Wir haben auch über Java Swing und Libgdx gedacht, um die Grafik zu darstellen.

- Schließlich haben wir die ersten Ideen über Fahrverhalten diskutiert:



Zuerst haben wir eine normale Kreuzung. Dann verteilen wir sie in die kleineren blauen Bereiche, damit wir die Fahrverhalten einfacher simulieren können.

2. Die Entscheidungen für Ideen

Wir haben entscheidet, um diese folgende Methode auszuwählen:

- Multi-Threading, ThreadPool

- Car-following model Nagel-Schreckenberg-Modell

- Libgdx, um Grafik zu darstellen

3. Gründe für diese Entscheidungen

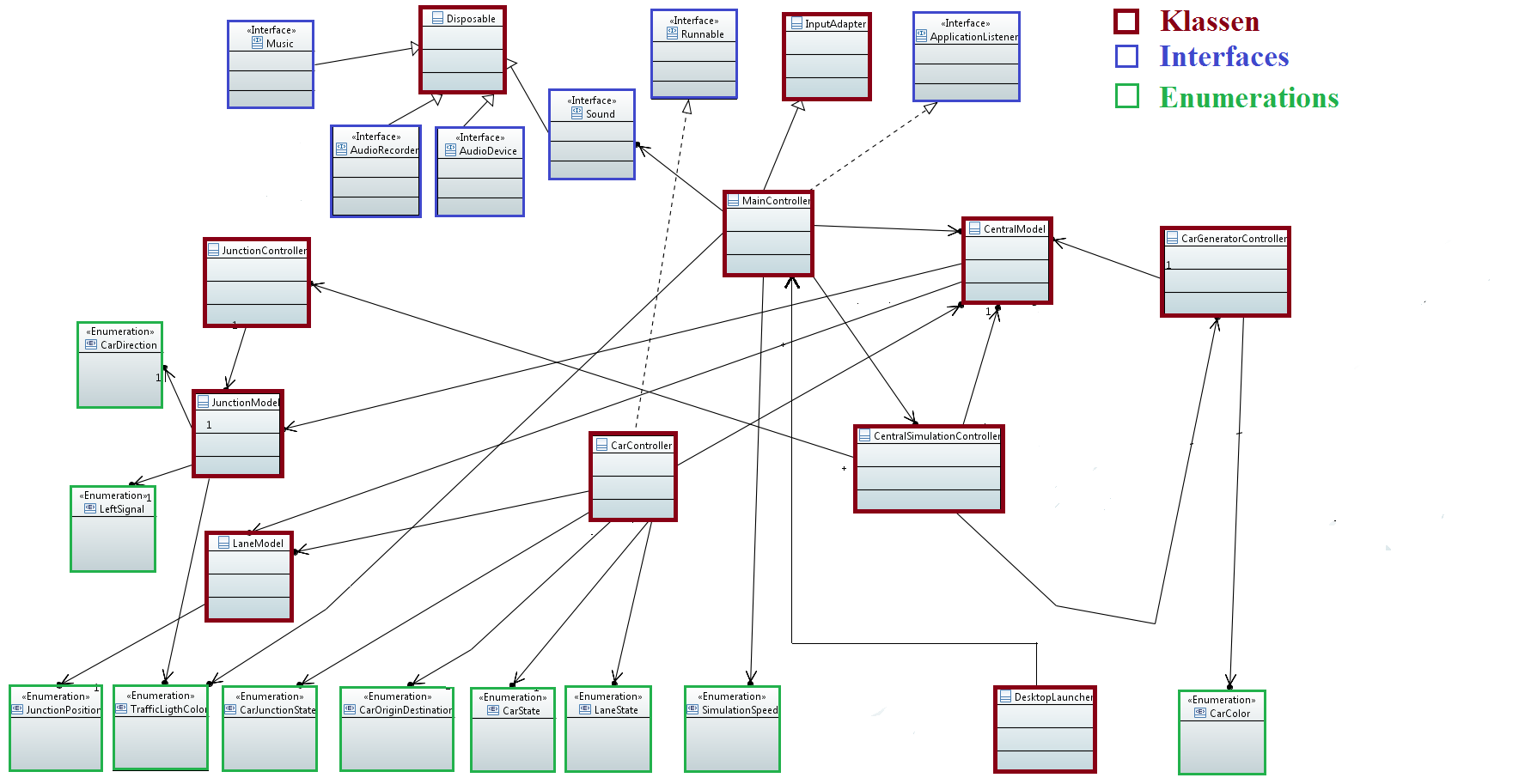
- Multi-Threading, ThreadPool

- Car-following model (Nagel-Schreckenberg-Modell):

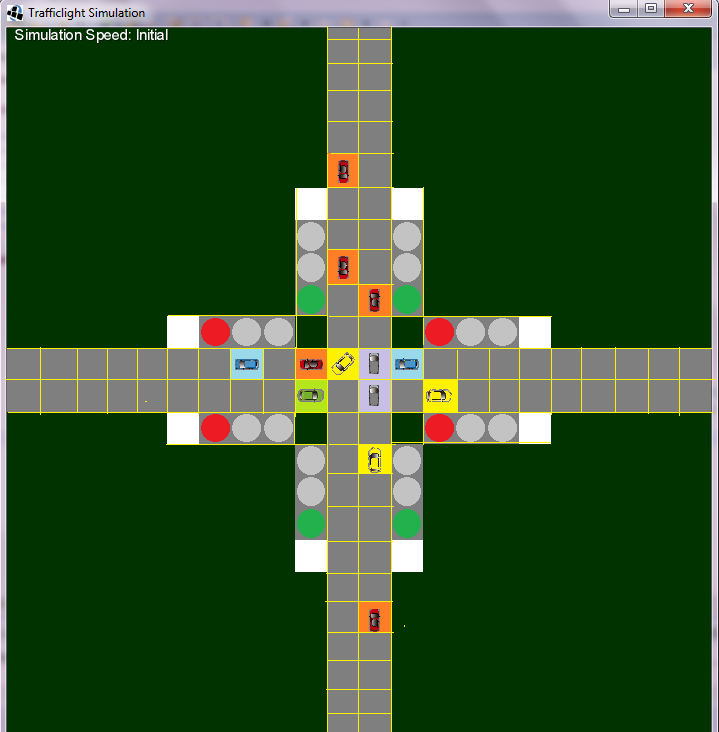
- Java Swing ist kompliziert zum Verwenden und hat eine schlechte Wirkung auf die Laufzeit -> Wählen Libgdx aus.

**III. Architektur und Entwurf**

1. Wir haben für unseren Projekt ein folgendes Klassendiagramm:

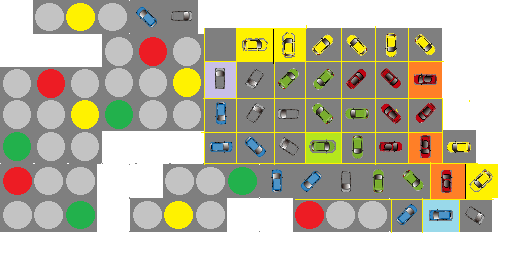


2. Mit diesen Abbildungen erklären wir den Prozess, um die Objekte zu visualisieren:

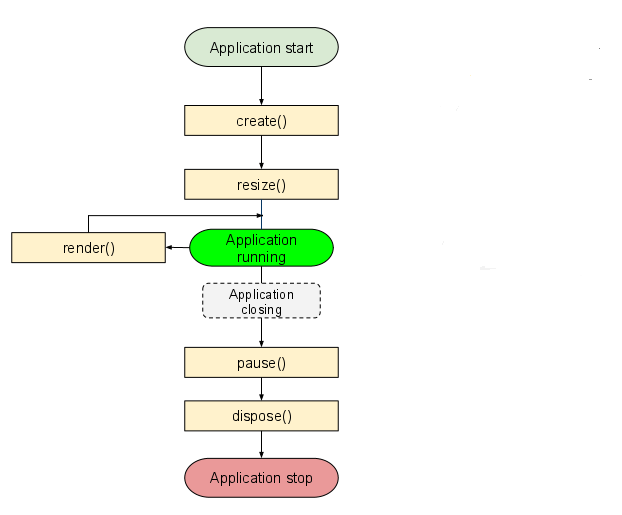


- Wir darstellen die gesamte Kreuzung wie ein Grid, in dem Autos fahren können. Wie zum Beispiel hier ist ein RedCarSouth Auto , das aus Norden kommen oder aus Süden verlassen kann, oder hier Street  ist ein einfacher Teil von der Straße, oder NorthTrafficGreen  ist eine Ampel mit der Farbe Grün im Norden.

- Wie können wir diese Bilder für unser Programm verwenden? Zuerst müssen wir alle Bilder für alle Möglichkeiten, um Fahrverhalten zu darstellen, erstellen. Dann benutzen wir das Tool Texture Packer, um alle diese Bilder zusammenzufassen und wir bekommen ein großes folgendes Bild (NewSimPics.atlas):



Dieses große Bild enthält alle wichtigen Informationen über die kleineren Bilder innerhalb wie zum Beispiel: Namen, Koordinaten.



Mit dieser Informationen können wir mit dem Method create() in diesem Aktivitätsdiagramm alle kleinen Bilder im großen Bild NewSimPics einmal laden. Dann rufen wir das Method render () wiederholt an. Mit diesem Method verwenden wir jedes kleine Bild, setzen wir sie in einer richtigen, passenden Reihenfolge, damit wir die Animationen auf der Straße und die Fahrverhalten richtig simulieren können.

**IV. Realisierung**

1. Die Umsetzung (Algorithmus) der Idee

Wir beobachten die Funktionen von der folgenden wichtigen Klassen:

class CarController : Zentrale ControllerKlasse für das Auto. Steuert die Bewegung des Autos innerhalb der in der Klasse CentralModel spezifizierten Datenstruktur.

class CarGeneratorController: Klasse für Autogenerierung. Erzeugt Autos mit zufälliger Farbe, zufälliger Herkunft und zufälligem Reiseziel und zufälliger Geschwindigkeit.

class CentralModel: Alle für die Visualisierung relevanten Daten sind in dieser Klasse enthalten und können mit Get Methoden abgefragt werden

class CentralSimulationController: Steuert und erzeugt alle anderen im logischen Teil der Simulation genutzten Controller Klassen.

class DesktopLauncher: Launcher Class für Programm laufen

class JunctionController: Controllerklasse für die Kreuzung. Aendert abhängig von einem internen Counter die Farbe der Ampeln.

class JunctionModel: Modellklasse für die Kreuzung. Enthält Informationen über die Farbe der 4 Ampeln, den Zustand der LinksAbbiegen Signale, ein Feld das angibt, wo sich gerade welches Auto auf der Kreuzung befindet und ein Feld das die Ausrichtung aller Autos auf der Kreuzung enthaelt.

class LaneModel: Modellklasse für Lane. Enthält Informationen über Länge der Lanes, Die Postion der Autos auf der Lane und die Position der Ampel.

class MainController: Main Class für Objekte und Sound durch Libgdx erstellen. Wir können auch das Tempo von Animation des Programmes durch die Tastatur steuern.

2. Die Probleme während der Umsetzung

- Multi-Threading Problem

- Abbiegen-Verhalten Problem

- Kollisionen-Vermeidung zwischen Autos Problem

- Factory-Method zum Erstellen Problem

- Leistung von Programm

**V. Evaluierung**

1. Die Kriterien, um die Umsetzung zu überprüfen

- Änderung der Konfigurationseigenschaften:

- SpawnRate

- LaneLength

- WaitingTime

2. Die Erwartung für die Ergebnisse

- Volle/ korrekte Funktionsweise der Simulation bleibt erhalten

3. Die Ergebnisse wir ermittelt haben

- Änderung der eingegebenen Werte beeinträchtigen die Simulation nicht

4. Die Interpretation dieser Ergebnisse

- Algorithmus der Simulation ist gut und effektiv umgesetzt worden

**VI. Diskussion und Fazit**

1. Dinge, die wir persönlich gelernt haben

- Realitätsnahes Arbeiten.

- Neue Erkenntnisse gewonnen: Github, Maven, LibGDX, Threads, YAML…

- Projektorganisation

2. Was würden wir im nächsten Mal anders machen?

- Es könnten die neuen Funktionen wie Kontrolle der Anzahl der Autos, Tempo der Autos messen, Stau vorwarnen… ergänzt werden.

- Es könnten intelligente Ampeln simuliert werden.