BERUFSAKADEMIE | MANNHEIM
university of cooperative education | staatliche studienakademie

Fachrichtung Informationstechnik

Studienarbeit

TraffSpot

Simulation einer Verkehrsflussoptimierung mit autonomen Verkehrsteilnehmern in frei definierbaren Verkehrsräumen

Mannheim, Mai 2005

RiSc Tec

Andreas Richter
102731
TIT02BNS

Nico Schröder
121576
TIT02BNS

Betreuer:
Dipl.-Inform. Markus Stöbe

Kurzbeschreibung

In Zeiten zunehmenden Verkehrs durch erhöhte Mobilität der Menschen ist eine Optimierung der Verkehrsflüsse unabdingbar. Die sinnvolle Koordinierung der Verkehrsteilnehmer an Kreuzungen ist dabei ein elementarer Bestandteil dieser Arbeit.

TraffSpot soll ein Hilfsmittel werden, um Verkehrsflüsse in auf das Wesentliche reduzierten Umgebungen testen zu können. Dabei geht es darum, Ampelschaltungen an beliebigen Kreuzungen besser zu gestalten, so dass mehr Fahrzeuge diese passieren können und die Wartezeiten auf Grünphasen reduziert werden.

Abbildung 1: Screenshot von TraffSpot im Simulationsbetrieb

Entstanden ist eine Software, die genau diesen Anforderungen entspricht. Es können durch den Benutzer erstellte Szenarien als XML-Datei eingelesen werden, die in der Software so im Speicher organisiert werden, dass sich Fahrzeuge frei auf dem erstellten Graphen bewegen können. TraffSpot arbeitet mit OpenGL, damit die Verkehrsströme sichtbar gemacht werden können und so die Effizienz der implementierten Ampelalgorithmen auf den ersten Blick sichtbar wird.

In TraffSpot wurden auch unterschiedliche Algorithmen zur Beschaltung von Kreuzungen implementiert. Neben zwei Algorithmen, die auf unterschiedliche Weise die Anzahl der Fahrzeuge zur Schaltentscheidung benutzen, dient der statische Ansatz – das Schalten nach Zeit – als Referenz für die Auswertung. Dabei hat sich gezeigt, dass der klassische Schaltvorgang bei gleich belasteten Straßen gar nicht so schlecht ist, wie man denken könnte. Auch kann man mit neuen Entscheidungsroutinen nicht immer ein besseres Ergebnis erzielen.

Abstract

In times of growing human mobility, optimization of traffic flows is necessary. Efficient coordination of traffic participants at intersections is a fundamental part of this work.

TraffSpot is a tool to test traffic flows by means of reduced models of road intersections. Its goal is to improve the traffic light algorithm at arbitrary intersections in order to allow more cars to pass the crossroads and to shorten waiting times for green light.

Figure 2: Screenshot of TraffSpot

A software according to these requirements was developed. It is able to read user-built scenarios as XML files and organizes them internally for simulation of freely traveling vehicles. TraffSpot uses OpenGL for visualization of traffic flows, facilitating easy assessment of the efficiency of implemented traffic light algorithms.

TraffSpot contains different algorithms to switch traffic lights at intersections. In addition to the standard algorithm with fixed timing, there are two dynamic strategies which take the actual number of cars into account. The three algorithms were compared. The standard algorithm performed better than expected, for example at intersections of equally busy streets. Under certain conditions, new strategies do not improve the results.

Eidesstattliche Erklärung

Hiermit versichern wir, dass wir unsere Studienarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben.

Ort, Datum	Unterschriften der Verfasser

Vorwort

Die hier vorgestellte Studienarbeit beschäftigt sich mit der Verbesserung von Verkehrsflüssen. Diese Themenwahl liegt zu einem Teil darin begründet, dass sich auch das Deutsche Zentrum für Luft- und Raumfahrt e.V. in der Helmholtzgemeinschaft (kurz DLR) damit beschäftigt – das Institut für Verkehrsführung und Fahrzeugsteuerung ist unter anderem der Arbeitsplatz eines Studienarbeitsautoren – zum anderen sieht man jeden Tag auf dem Weg zur Arbeit, dass Verkehrsflüsse optimiert werden könnten.

Der Aufbau einer Simulation als Software ist jedoch nicht so schnell und einfach zu realisieren. So möchten wir uns für die tatkräftige Hilfe von vielen Seiten bedanken, speziell bei Herrn Florian Weichenmeier von GEVAS software, der uns im Nachhinein in unserem Lösungsweg bestätigt hat, sowie Herrn Prof. Joachim Schmidt, der uns mit seinem nicht nachlassenden Interesse an der Studienarbeit immer noch etwas mehr Motivation verschafft hat. Auch Markus Stöbe, unser Studienarbeitsbetreuer, soll hier nicht ohne Dankesgruß bleiben, da er uns jederzeit mit Rat und Tat zur Seite stand.

Inhaltsverzeichnis

Abbildungsverzeichnis

Formelverzeichnis

Abkürzungsverzeichnis

CSV	Comma Separated Value
DTD	Document Type Definition
GPL	General Public License
GUI	Graphical User Interface
ID	Identifikator oder Identifikationsnummer
OpenGL	Open Graphics Library
Mac OS	Apple Macintosh Betriebssystem
Qt	Klassenbibliothek zur Programmierung graphischer Benutzeroberflächen
XML	Extensible Markup Language

1 Einleitung

Der moderne Mensch ist mobil und arbeitet daran, immer mobiler zu werden. So hat er früh angefangen, über weite Strecken hinweg Waren auszutauschen und irgendwann begann er, auch nicht mehr direkt am Arbeitsort zu wohnen. Je mehr Menschen unterwegs sind, desto mehr Aufwand muss man in die Koordinierung der Verkehrsströme legen, damit nicht ein gnadenloses Chaos entsteht.

Chaos entsteht immer dann, wenn sich Verkehrswege auf gleicher Höhe kreuzen – also einen Knotenpunkt bilden. Nun kommt es darauf an, wie stark frequentiert die Straßen sind. Je frequentierter, desto mehr Mühe macht die Koordination der Kreuzung.

Sind nur wenige Verkehrsteilnehmer unterwegs, so reichen einfache Regeln wie *Rechts vor Links*. Sind allerdings mehr Fahrzeuge auf den Straßen, ist es angebracht, einen Mechanismus für diese Aufgabe zu finden: Die Lichtsignalanlage, landläufig auch Ampel genannt. Ein vorher definierter Algorithmus schaltet Lichtsignale für jede Spur und koordiniert somit das gefahrlose Benutzen der Kreuzung. Der nicht zu vernachlässigende Betrachtungspunkt bei dieser Problemlösung ist der Ampelalgorithmus. Dieser sollte möglichst auf die Verkehrssituation der Kreuzung angepasst sein.

Abbildung 3: stumme Helfer im Wildwuchs – Ampelanlagen außer Rand und Band[1]

1 Kunstinstallation mit Traffic Lights in London

2 Bereits existierende Lösungen

Wie bereits erläutert wurde, ist das Problem der Verkehrsflussoptimierung ein schon etwas älteres. So wurden bereits mehrere Projekte entwickelt, die sich auch mit dem Ansatz der Verkehrsoptimierung beschäftigen. Im Folgenden werden nun drei thematisch verwandte Projekte vorgestellt:

2.1 NONSTOP

Von der Firma GEVAS software /1/ wird bereits seit sieben Jahren das Programm NONSTOP entwickelt. NONSTOP ist mit dem Ziel entwickelt worden, nicht einfach nur eine Abbildung der Realität zu schaffen, sondern auch Netzsteuerungsverfahren exakt emulieren zu können. So können auf schnellstem Wege Verkehrsnetze neu angelegt und verändert werden und die Simulation durch offene Schnittstellen mit eigenen Erweiterungen bestückt werden. Des Weiteren bietet NONSTOP zum Beispiel die Möglichkeit, Fußgänger zu berücksichtigen, den öffentlichen Personennahverkehr mit zu integrieren und bietet schnelle und individuell einstellbare Fahrverhalten der Verkehrsteilnehmer.

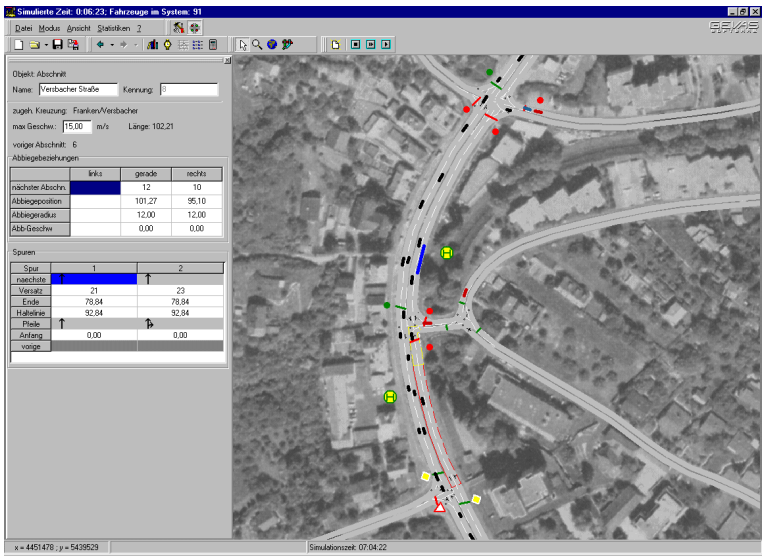


Abbildung 4: Verkehrsnetzsteuerung mit NONSTOP

Die Verkehrsnetze werden über so genannte GIS-Daten[2] eingefügt. Das sind Daten, die geographische Informationen über Strecken – nichts anderes als Kantenstützpunkte – enthalten. Da solche Informationen käuflich zu erwerben sind, ist es ein Leichtes, neue und aktuelle Straßennetze ohne großen Aufwand als Grundlage nutzen zu können.

Für die Ampelsteuerung kommt das Modul namens TRENDS-Kern zum Einsatz. Dabei handelt es sich um eine Implementierung für eine lokale Steuerung, die bereits in vielen Am-

2 Geoinformationssystem

peln in deutschen Städten ihren Einsatz findet. Das genaue Vorgehen des Algorithmus bleibt Erfolgsgeheimnis von GEVAS software. Für die Netzsteuerung kommt ein Modul des Kooperationspartners TransVer zum Einsatz, das nachfolgend kurz beschrieben wird.

2.2 BALANCE

Von TransVer /2/ stammen mehrere Module zur Ampelsteuerung, so auch das bereits angedeutete BALANCE. BALANCE ist hauptsächlich für die Steuerung ganzer Verkehrsnetze ausgelegt und kann optimale Signalzeiten ermitteln. Es benötigt allerdings trotzdem eine lokale Steuerung der Ampelanlagen. TransVer bietet die eigene Lösung namens MikroBALANCE an. Die Steuerung ist dabei frei skalierbar. Dadurch könnten auf Netzebene Steuerungsgrößen wie Umlaufzeit oder Freigabezeitaufteilung optimiert werden. In Sekundenschnelle kann dabei auf neue Situationen reagiert werden, ob nun über stochastische Messdaten oder zum Beispiel auf öffentlichen Personennahverkehr-Priorisierung.

Die Systeme BALANCE als auch NONSTOP sind soweit fortgeschritten, dass sie bereits in diversen Städten eingesetzt werden. So fand in Hamburg, sowie in München ein groß angelegter Pilotversuch zur Optimierung der Verkehrsflüsse statt. Die Ergebnisse sind so zufriedenstellend, dass man in Hamburg bis 2010 die meisten der Ampeln auf die neue Schaltung umstellen will /3/.

2.3 SUMO

Die „Simulation of Urban MObility“ ist ein weiteres Projekt, das sich mit Verkehrsflüssen beschäftigt /4/. Hierbei handelt es sich um ein open source-Projekt, an dem das Institut für Verkehrsforschung des DLR Berlin und das Zentrum für Angewandte Informatik Köln zusammenarbeiten. Das Ziel von SUMO ist es, allen ein Simulationstoolkit zur Verfügung zu stellen, die vor haben, eine Simulationsaufgabe realisieren zu wollen. Damit der Fokus auf die eigentliche Forschungsarbeit gerichtet bleiben kann, soll eine Funktionensammlung entwickelt werden, die das Entwerfen eines eigenen Simulationkernes überflüssig machen soll.

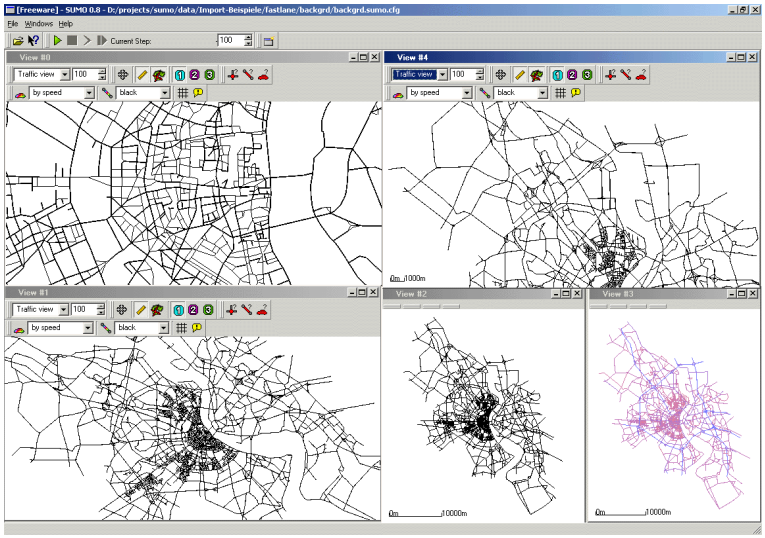


Abbildung 5: ein mit SUMO erstelltes Verkehrsnetz

SUMO bedient sich dabei mehrerer anderen Projekte, so besteht SUMO also eigentlich aus einer Vielzahl kleinerer Projekte. Somit ist es möglich, ein und die selbe Simulation frei aus einem makroskopischen in einen mikroskopischen Zustand zu zoomen – also sowohl ganze Netze, wie zum Beispiel eine Großstadt, können simuliert werden, als auch die physikalischen Vorgänge innerhalb eines bestimmten Pkw analysiert werden.

Es stellt sich nun die Frage, warum SUMO nicht Grundlage für TraffSpot geworden ist. Dies liegt darin begründet, dass man sich der Herausforderung stellen wollte, eine eigene Simulation zu entwickeln.

3 Aufgabenstellung

TraffSpot soll ein Synonym für die langatmige Umschreibung „Simulation einer Verkehrsflussoptimierung mit autonomen Verkehrsteilnehmern in frei definierbaren Verkehrsräumen“ sein. Dieser Titel beinhaltet bereits mehrere Arbeitspakete:

Es soll eine Simulation erstellt werden, in der man vorher erstellte Strecken möglichst ohne großen Aufwand testen kann. Des Weiteren soll der Verkehrsfluss mit jener Simulation sichtlich verbessert werden, dass heißt, es soll ein neuer Ansatz für eine Schaltlogik implementiert werden.

Die Software soll sich in diesem Projekt möglichst um alles kümmern, dass heißt, dass der Nutzer sich keine Gedanken um die Umsetzung einer Simulation machen muss. Die einzige ihm aufgebürdete Arbeit beinhaltet das Planen der Teststrecke und das Festhalten aller wichtigen Parameter in einer Datei. In dieser vom Benutzer geschriebenen Datei sollen die Straßenzüge festgelegt werden und Angaben, wo welche Art von Fahrzeugen entstehen. Die Software erstellt daraus eine Welt und lässt die Simulation bei beliebiger Geschwindigkeit ablaufen. In einer grafischen Ausgabe sollen die Abläufe sichtbar gemacht werden. Dies dient lediglich der Veranschaulichung der Vorgänge. Währenddessen sollen statistische Daten der Fahrzeuge und Kreuzungen erhoben werden, mit denen später eine Auswertung der Effizienz der Ampelalgorithmen durchgeführt werden kann. Nachdem die Simulation vollständig oder teilweise durchlaufen wurde, sollen die gesammelten statistischen Daten als Datei abgespeichert werden können.

Nicht aus dem Fokus zu verlieren ist aber auch die eigentliche Verkehrsflussoptimierung, die Simulationssoftware soll, wie bereits geschrieben, lediglich Mittel zum Zweck sein. Allerdings ist es ohne eigentliche Simulation schwer, einen Verkehrsfluss zu optimieren, so ist also beides eng verknüpft. Die Implementierung des Standardschaltverhaltens nach einer bestimmten Zeit ist dabei Referenz für alle neu zu erdenkenden Schaltungen.

Ein weiteres wichtiges Kriterium für TraffSpot soll die Plattformunabhängigkeit sein. Da beide Autoren dieser Studienarbeit auf unterschiedlichen Plattformen arbeiten[3], ist es natürlich auch ein Ziel, auf beiden Plattformen die Simulation durchführen zu können.

3 Mac OS X und Microsoft Windows XP

4 Software

TraffSpot soll, wie in der Aufgabenstellung bereits definiert, plattformunabhängig realisiert werden. Dazu wurde auf die bewährte Kombination von Qt und C++ zurückgegriffen.

Qt[4] ist eine C++-Klassenbibliothek der Firma Trolltech /5/ zur Programmierung portabler GUI-Programme. Mit Qt ist es möglich, schnell und einfach Programme mit grafischer Benutzeroberfläche zu erstellen. Die spätere Wartung wird dahingehend verbessert, als das keine Callback-Handler-Funktionen wie bei anderen Toolkits benutzt werden, sondern das so genannte Signal-Slot-Prinzip. Auf diese Weise können Objekte miteinander kommunizieren. Qt bietet des Weiteren auch Zusatzfunktionen, wie das Einbinden von OpenGL-Fenstern oder einem einfachen XML-Parser. Auch Dateizugriffe und andere Operationen sind in Qt implementiert.

Qt hat ein sehr einfaches Lizenzvergabesystem: So steht Qt in der aktuellen Version 3.3.4[5] für Linux, Embedded Linux und Mac OS X frei zur Verfügung, wenn die mit Qt entwickelten Programme unter die General Public License (GPL) gestellt werden. Das beinhaltet, dass das Programm zu jedem Zweck frei benutzt und geändert werden darf, sowie, dass der Quellcode zugänglich gemacht werden muss. In der kommenden Version Qt 4 wird es auch eine freie Version für Windows geben. Will man mit durch Qt entwickelte Programme eine Kommerzialisierung anstreben, wird eine kostenpflichtige Comercial-Lizenz fällig. Für die Forschung und Lehre gibt es des Weiteren auch kostenlose Lizenzen, so wird TraffSpot auch mit einer Education-Lizenz entwickelt. Bekannte kommerzielle Programme, die Qt benutzen, sind zum Beispiel der Opera-Webbrowser, das Bildbearbeitungsprogramm Photoshop Album von Adobe oder die KDE-Entwicklungsumgebung.

Die Entscheidung für die objektorientierte Programmiersprache C++ ist indirekt bereits durch die Entscheidung für Qt gefallen. Als Alternative steht Java mit eigenem Fensterklassen Swing und AWT zur Debatte. Die Entscheidung gegen Java basiert lediglich darauf, dass bereits gute Erfahrungen mit Qt und C++ gesammelt wurden[6]. Des Weiteren hält sich das Vorurteil über die schlechtere Performance von Java gegenüber C++ standhaft und da davon auszugehen ist, dass die Grafik der Simulation und die dahinter stehenden Daten einen nicht zu unterschätzenden Aufwand erreichen können, fiel die Entscheidung auf C++.

Eine objektorientierte Sprache hat den Vorteil, dass ein solches Programm mit Hilfe von Zusatzbibliotheken – wie zum Beispiel MPICH – parallelisierbar ist und die Last einer umfangreichen Simulation auf mehrere Computer auslagerbar wird.

4 Qt steht für Quasar toolkit. Quasare sind die dauerhaft am stärksten leuchtenden Objekte im Universum.
5 Stand: Mai 2005
6 RhiNotiz (http://rhinotiz.db-nico.de) – Ein Programm zur Notizverwaltung

TraffSpot wurde nach klassischem Wasserfallmodell in einem 3-Schichten-Modell, wie Abbildung 6 zeigt, entwickelt. Dieses Modell umfasst eine Darstellungsschicht, eine Anwendungsschicht und eine Datenhaltungsschicht. Umschlossen werden diese drei Schichten von einer Fehlerbehandlung, welche sich durch alle Schichten zieht.

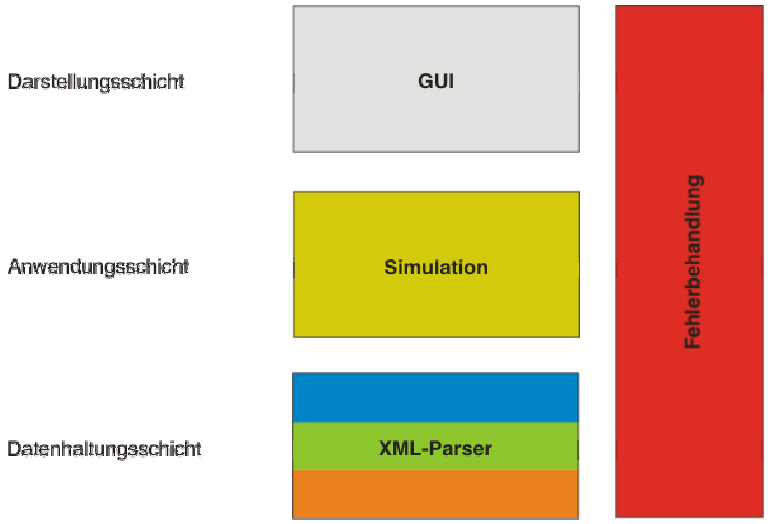


Abbildung 6: 3-Schichten-Modell von TraffSpot

Den drei Schichten sind folgende Bestandteile aus TraffSpot zugeordnet: Zum Ersten gibt es die eigentliche Simulation in der Anwendungsschicht, die alle wichtigen Klassen zum Simulieren der autonomen Verkehrsteilnehmer im Verkehrsraum definiert und implementiert. Als zweiter großer Bestandteil ist die GUI zu betrachten. Auf ihr wird die Simulation dargestellt und sie ist die Schnittstelle, mit der ein Benutzer den ablaufenden Prozess beeinflussen kann. Als letzten Block ist der XML-Parser zu betrachten. Hier wird ein vom User erstelltes Szenario eingelesen und im Hintergrund werden in die Listen gefüllt. Im Folgenden werden nun die drei Bestandteile und deren Funktion näher beschrieben.

4.1 Simulation

Wenn man sich nun an die Aufgabe wagt, die Verkehrsflüsse auf Kreuzungen zu optimieren, ist es schwer, dies in der Realität auszuprobieren. Deswegen bedient man sich der Simulation, die viele Vorteile aufweist. Hauptsächlicher Grund, die Problematik virtuell zu lösen, ist die finanzielle Seite. Es ist in einer simulierten Welt einfacher, Parameter zu ändern, beliebige Verkehrsräume zu erstellen oder beliebig viele unterschiedliche Verkehrsteilnehmer miteinander fahren zu lassen. Würde man dies in der Realität nachstellen wollen, wäre ein immenser finanzieller und zeitlicher Aufwand damit verbunden. In einer Simulation kann man schnell drei Kreuzungen hintereinander schalten. Auch Extremsituationen – und das ist

der eigentliche Vorteil – können nachgestellt und getestet werden, ohne Menschen zu gefährden. Auch ist es einfacher, neue Konzepte ausprobieren zu können.

4.1.1 Architektur

Die Simulation setzt sich aus der Sicht der Programmiersprache aus unterschiedlichen Klassen zusammen. Diese Klassen interagieren auf verschiedene Weise miteinander. Abgeleitet werden die Klassen aus der Realität. So gibt es *Spuren*, auf denen sich *Fahrzeuge* bewegen. Mehrere Spuren können zu einer *Kreuzung* zusammengefasst werden. Die Erstellung der Fahrzeuge in der *Simulation* wird von *Generatoren* übernommen. Auch ohne graphische Oberfläche ist die Simulation dank der Schichtenarchitektur eigenständig lauffähig und modular einsetzbar.

In der Simulation werden nur gerade Strecken verwendet, da das Hauptaugenmerk auf den Ampelschaltungen liegen soll. In einem solchen Fall sind die Kurven vor und nach Kreuzungen irrelevant. Es wird ein Graph implementiert, dessen Knoten die Straßen darstellen. Dies ist ein ungewöhnlicher Ansatz, da in der Regel die Kreuzungen auch die Knoten darstellen. In der Studienarbeit wurde dieser Ansatz gewählt, da auf diese Art und Weise eine Klasse *Spur* existieren kann und mit einfachen Mitteln mehrere Spuren zu einer Kreuzung zusammengefasst werden können. Die einzelnen Spuren werden in die Knoten verlagert und können über die Kanten des Graphen beliebig verbunden werden. Auf diese Weise lassen sich sowohl gerade Strecken, als auch Kreuzungen beliebiger Art und Anzahl von Spuren, wie in Abbildung 7 deutlich wird, realisieren.

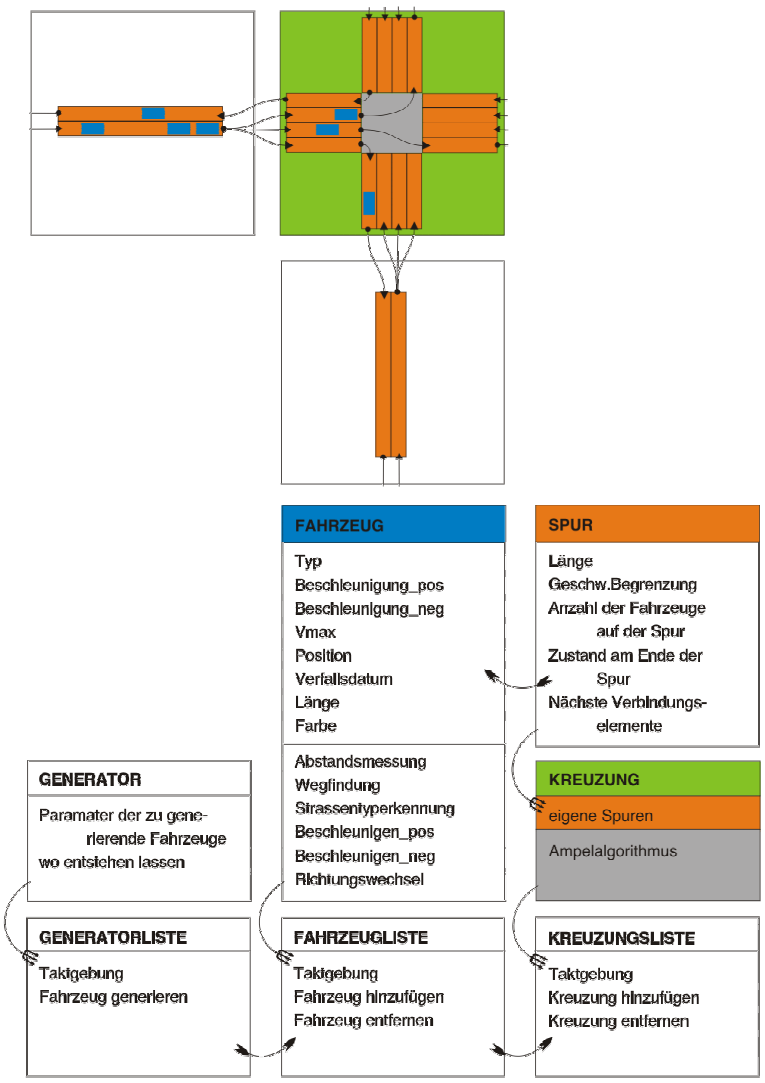


Abbildung 7: Simulationskonzept in TraffSpot

Die Klasse *Spur* wurde mit verschiedenen Eigenschaften (Attributen) versehen. Zu ihnen zählen unter anderem die Länge der Spur in Meter, eine allgemeine Geschwindigkeitsbeschränkung in km/h, die aktuelle Anzahl der Fahrzeuge auf dieser Spur und ihr Endzustand: Offen, die Ampel steht auf grün, oder geschlossen, die Ampel steht auf Rot. Die Hauptaufgabe der *Spur* besteht darin, ein Verbindungselement darzustellen, mit dem es möglich ist, einen gerichteten Graphen aufzubauen. So entsteht am Ende ein verknüpfter Straßenverlauf. Eine Spur enthält aus Verwaltungsgründen eine kleine Liste der auf ihr befindlichen Fahrzeuge. Dieser Umstand macht es einem Fahrzeug in der Simulation möglich, seinen Vorgänger abzufragen und eine Entfernungsmessung durchzuführen.

Spricht man von Spuren in einem Verkehrsnetz, so kommt man zwangsläufig auch zu Knotenpunkten. Diese Knotenpunkte werden in der Simulation durch *Kreuzungen* definiert.

Ihre Aufgabe besteht darin, auf die einzelnen Spuren einer Kreuzung Zugriff zu gewähren, um auf ihnen eine Ampelschaltung durchführen zu können.

Die Verkehrsteilnehmer werden durch die Klasse *Fahrzeug* spezifiziert. Die Objekte dieser Klasse können dabei unterschiedlicher Ausprägung sein, zum Beispiel:

- Personenkraftwagen (Pkw)
- Lastkraftwagen (Lkw)
- Busse
- Motorräder

Die Fahrzeuge unterscheiden sich nur in der Ausprägung ihrer Eigenschaften von einander. Ein Lkw ist im Falle der Simulation ein sehr langes Fahrzeug, mit geringem Beschleunigungsvermögen und geringer Höchstgeschwindigkeit, ein Motorrad hingegen kürzer und spritziger in der Beschleunigung. Dieses Vorgehen ermöglicht eine Gleichbehandlung aller Fahrzeuge in der Simulation. In dieser Klasse wurde sehr stark Bezug zur Realität genommen. Die Verkehrsteilnehmer in der Simulation sollen sich auch in das reale Leben übertragen lassen. Aus diesem Grund besitzen Fahrzeuge Sensoren, Aktoren und Eigenschaften. Mit den Sensoren wird die Umwelt analysiert. Dazu gehört die Abstandsmessung zum Vordermann oder zur Kreuzung, eine Wegfindung (per Fahrtenbuch oder Zufallsgenerator) und eine Straßentyperkennung (Höchstgeschwindigkeit, Möglichkeiten des Abbiegens). Aktoren reagieren auf Grund der Messdaten aus den Sensoren. Hierzu gehört das Beschleunigen, Bremsen oder der Richtungswechsel. Bedingt werden die Aktoren durch die Eigenschaften eines Fahrzeuges. Zu denen zählen der Typ (Pkw, Lkw, Bus, Motorrad), die Beschleunigung in positiver und negativer Richtung, die Höchstgeschwindigkeit und die Länge des Fahrzeuges. Neben diesen statischen Daten eines Fahrzeuges werden auch dynamische Daten (Telemetriedaten) gespeichert, z.B. die aktuelle Geschwindigkeit, der nächste Richtungswechsel, der zurückgelegte Weg (Kilometerzähler), die gefahrenen Zeit und die aktuell befahrene Spur. Auf Grund der Aktoren eines Fahrzeuges kann man es in der Realität mit einem Fahrzeug mit Automatikgetriebe vergleichen. Es besitzt sozusagen ein Gaspedal (positive Beschleunigung), ein Bremspedal (negative Beschleunigung) und ein Lenkrad (Richtungsänderung).

Um die Fahrzeuge an bestimmten Stellen in die Simulation einsetzten zu können, bedarf es der Generatoren, deren Hauptaufgabe darin besteht, neue Fahrzeuge in die Fahrzeugliste einzutragen und ihnen Eigenschaften mitzugeben. Die Eigenschaften können für einen Generator beliebig festgelegt werden. So ist es möglich, nur Fahrzeuge eines Typs zu generieren oder zufällig auf den Fahrzeugtyp abgestimmte Attribute setzen zu lassen. Über die Generatoren ist es möglich, die Simulation nach eigenem Ermessen zu steuern. Es können

bestimmte Fahrzeugtypen vorrangig erzeugt werden oder ihnen ein bestimmter Weg durch die Verkehrswelt bei der Erzeugung vorgegeben werden.

Das Fortschreiten der Zeit wird in der Simulation durch einen übergeordneten Taktgeber realisiert. Um mit Hilfe dieses Taktgebers, wie in Abbildung 8 ersichtlich, der Verkehrswelt mitteilen zu können, dass die Zeit um eine bestimmte Einheit fortgeschritten ist, müssen die Kreuzungen, Fahrzeuge und Generatoren in einer Liste organisiert werden. An dieser Stelle kann Vererbung und Polymorphie aus der objektorientierten Welt der Programmiersprachen sehr gut zu einer einheitlichen Ansteuerung der Listen verwandt werden. Es wird eine Oberklasse (*ListenDaten*) erzeugt, welche nur die für die Organisation in der Liste wichtigen Eigenschaften besitzt. So enthält diese Klasse eine ID zum Wiederfinden der Objekte in der Liste und ein Flag, welches den Zustand eines Objektes zwischen aktiv und inaktiv wechseln lässt. Von dieser Klasse werden die Klassen *Kreuzung*, *Fahrzeug* und *Generator* abgeleitet. Eine Klasse *Liste* verwaltet Objekte des Typs *ListenDaten*. Somit ist es möglich, in eine Liste sowohl Kreuzungen, Fahrzeuge und Generatoren einzupflegen.

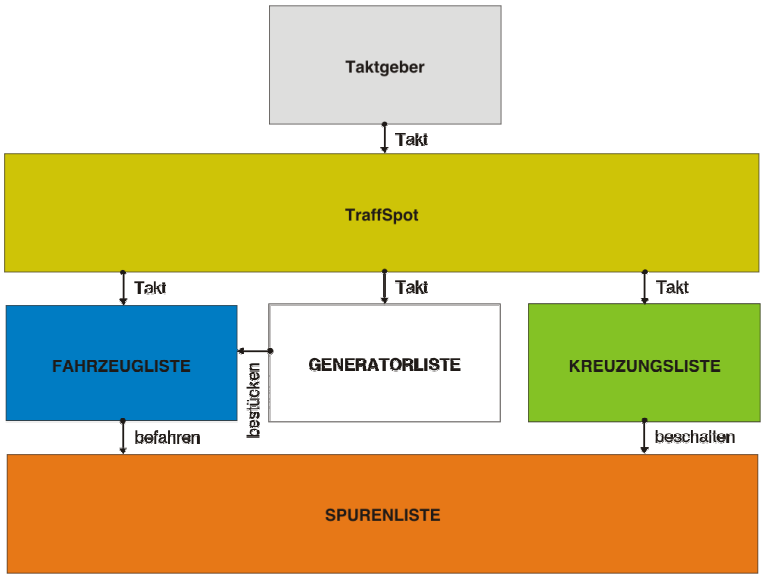


Abbildung 8: Modell des Antaktens in TraffSpot

Die Klasse *ListenDaten* besitzt eine virtuelle Funktion *takt()*, welche von jeder Unterklasse implementiert werden muss. Eine Kreuzung entscheidet an dieser Stelle, ob eine Änderung der Ampelschaltung vorgenommen werden muss. Die Fahrzeuge bewegen sich nach Aufruf der Funktion um eine bestimmte Strecke nach vorn. Die Generatoren entscheiden, ob ein neues Fahrzeug generiert werden soll. Um eine gewisse Übersicht in der Liste zu wahren, werden in der Simulation drei verschiedene Objekte der Klasse *Liste* instanziert und ihnen

nur Daten eines Typs übergeben. Die drei Listen werden nacheinander zu einer Aktion angeregt. Dabei geht die Liste alle Elemente in ihr durch und regt wiederum aktive Objekte zu einer Aktion an.

4.1.2 Aktionen

Jede Aktion wird in einer bestimmten Zeitspanne ausgeführt. Diese Zeitspanne wird auch Reaktions- oder Entscheidungszeit genannt. Dies ist die Zeit, die ein Verkehrsteilnehmer benötigt, um sich für eine neue Vorgehensweise zu entscheiden (z.B. weiterhin bremsen oder wieder beschleunigen). In der Simulation kann dieser Wert zwischen 10 und 100 Millisekunden vom Benutzer frei eingestellt werden. Diese Spannen wurden gewählt, weil Verkehrspsychologen diese Werte als Grundlage nutzen.

Die Aktion eines Fahrzeuges ist gegenüber den Aktionen der Kreuzungen und Generatoren um einiges komplexer. Ein Fahrzeug misst mittels seiner 'Sensoren' den Abstand zum Vordermann. Dieser Abstand wird als Hindernissabstand gewertet. Kann vor ihm kein weiteres Fahrzeug detektiert werden, so wird die folgende Kreuzung betrachtet. Ist das Ende der Spur geschlossen (z.B. rot an der Ampel), so ist der Abstand bis zum Ende der Spur der Hindernissabstand. Wird ein offenes Ende erkannt, so wird auf der nächsten Spur nach einem möglichen Hindernis gesucht, solange bis ein Hindernis gefunden wurde oder die natürliche Grenze des vorausschauenden Fahrens überschritten wurde. Der Hindernissabstand wird im Folgenden mit dem Bremsweg des Fahrzeuges verglichen, welcher sich bei einer sofortigen Vollbremsung aus der aktuellen Geschwindigkeit ergeben würde.

$$s_{bremsweg} = \frac{v^2_{momentan}}{a_{negFzg}}$$

Formel 1: Formel für die Bremswegberechnung

Ist der Hindernisabstand geringer als der doppelte Bremsweg, so muss eine Bremsung eingeleitet werden. Zu vergleichen ist diese Situation mit dem Mindestabstand eines Fahrzeuges auf einer realen Straße. Wird dieser zu klein, so bremst der Verkehrsteilnehmer ab, bis der Sicherheitsabstand wieder ein vorgeschriebenes Maß erreicht hat:

$$s_b[m] = v[\frac{km}{h}]/2$$

Formel 2: Richtwert für den Sicherheitsabstand

Dem Fahrzeug in der Simulation ist es möglich, das Maß der Bremsung an den Abstand anzupassen. So kann zwischen einer halben Bremsung und einer Vollbremsung stufenlos gewählt werden:

$$a_{bremsen} = n * a_{negFzg} \qquad n \in \mathbb{R},\ 0{,}5 \leq n \leq 1$$

Formel 3: Grad der Bremsung

In der Simulation errechnet ein einfacher Algorithmus diesen Wert, in der Realität basiert dieser auf der Einschätzung des Fahrers. Ist der Abstand zu einem Hindernis größer als der doppelte Bremsweg, so kann bei noch nicht Erreichen der Höchstgeschwindigkeit von Spur und Fahrzeug weiter beschleunigt werden. Bei einem Wechsel auf eine Spur mit geringerer Höchstgeschwindigkeit wird die aktuelle Geschwindigkeit automatisch der neuen Höchstgeschwindigkeit angepasst.

Wechselt ein Fahrzeug die Spur – z.B. an einer Kreuzung – so wird eine Betrachtung angestellt, welche Spur bei einem weiteren Spurwechsel als nächstes befahren werden soll. Dabei gibt es zwei Möglichkeiten: Hat der Benutzer ein Fahrtenbuch für das Fahrzeug definiert, so wird überprüft, welcher Richtungswechsel als nächstes in diesem vorgesehen ist. Ist es nicht möglich, diesen Richtungswechsel durchzuführen, wird der nächste gewählt, bis ein möglicher Richtungswechsel gefunden wurde. Ohne die Definition eines Fahrtenbuches oder nach einem abgefahrenen Fahrtenbuch ist es dem Zufall überlassen, welche Richtung als nächstes eingeschlagen wird. Auf jeder Spur wurde für alle drei Richtungen eine Wahrscheinlichkeit[7] definiert. Der Zufallsgenerator ermittelt auf Basis dieser Wahrscheinlichkeiten eine neue Richtung. Somit ist es möglich, stark und schwach befahrene Spuren durch Variation der Abbiegewahrscheinlichkeit zu erzeugen.

Die Aktion einer Kreuzung umfasst eine Prüfung der seit der letzten Schaltung abgelaufenen Zeit und eventuell der Anzahl über die Kreuzung gefahrener Fahrzeuge. Der eingesetzte Ampelalgorithmus entscheidet danach, ob eine neue Schaltung eingestellt werden muss. Dabei wird auch entschieden, wie beschaltet werden muss. Dazu im Kapitel 5 Verkehrsflussoptimierung mehr.

Ein Generator agiert ähnlich einer Kreuzung auf Zeitebene. Es wird bei jedem Takt die abgelaufene Zeit seit der letzten Fahrzeugerzeugung überprüft. Ist diese so weit fortgeschritten, dass die Erstellzeit erreicht wird, muss ein neues Fahrzeug erzeugt werden. Dazu werden voreingestellte Werte genutzt, oder ein Zufallsgenerator eingesetzt.

7 Bei Nichtexistenz einer Spur wird die Wahrscheinlichkeit p = 0 definiert.

Durch alle Schichten der Software zieht sich eine Fehlerbehandlung. Dazu wurde eine Klasse *Fehler* entworfen und implementiert. Tritt an einer beliebigen Stelle des Programms ein Fehler auf (z.B. falsche Parameter, ein Fahrzeug besitzt keine Spur, etc.), wird eine Instanz der Klasse *Fehler* erzeugt. Dieser Instanz kann eine Fehlerbeschreibung und eine optionale Fehlernummer mitgegeben werden. Der Fehler wird durch alle Schichten bis zur Benutzerschicht gereicht und kann an dieser Stelle dem Benutzer dargestellt werden.

4.1.3 Statistiken

Kreuzungen und Fahrzeuge erzeugen während ihrer Aktionen nebenläufig Statistiken. Diese Statistiken beinhalten bei den Fahrzeugen die gefahrenen Kilometer und die gefahrene Zeit. Die Kreuzungen führen Zählungen der über sie gefahrenen Fahrzeuge durch. Dabei werden die unterschiedlichen Fahrzeugtypen getrennt gezählt und nach den von ihnen eingeschlagenen Richtungen aufsummiert. Diese Statistiken können vom Benutzer während oder am Ende der Simulation in mehreren CSV-Dateien abgelegt werden. Öffnen kann diesen Dateityp jedes gängige Tabellenkalkulationsprogramm. Es ist dem Benutzer möglich, in der Tabellenkalkulation verschiedene Berechnungen anzustellen und Diagramme zu erzeugen. Aus diesem Grund fiel die Wahl des Ausgabemediums auf CSV-Dateien.

4.1.4 Ampelalgorithmen

Die Simulation besitzt verschiedene Ampelalgorithmen und stellt diese dem Benutzer zur Verfügung. Diese Algorithmen sind in der Software hart codiert. Trotzdem ist es problemlos möglich, weitere Ampelalgorithmen hinzuzufügen. Dazu müssen verschiedene Änderungen in der Klasse *Kreuzung* (*Kreuzung.cpp/h*) vorgenommen werden. Erstens muss eine Funktion mit dem Namen des Ampelalgorithmus in die Klasse integriert werden (*Kreuzung.h*):

```
class Kreuzung : public ListenDaten
{
   ...
   private:

      void algorithmusZeitZeit( double zeitintervall );
      void algorithmusZeitMenge( double zeitintervall );
      void algorithmusStopp( double zeitintervall );
      void algorithmusMengeMenge( double zeitintervall );
      void algorithmusNeu( double zeitintervall );
   ...
};
```

Der Parameter *zeitintervall* gibt die vom Benutzer gewählte Reaktionszeit der Verkehrsteilnehmer an. In der neu angelegten Funktion besitzt man über folgende Syntax Zugriff auf die unterschiedlichen Spuren, welche an der Kreuzung beteiligt sind (*Kreuzung.cpp*):

```
spurenListe_[ausrichtung-1][abbiegerichtung-1] ...
```

Dabei kann *ausrichtung* die Werte *NORD*, *SUED*, *OST* oder *WEST* annehmen. Der *abbiegerichtung* werden die Werte *LINKS*, *MITTE* und *RECHTS* zugeschrieben.

Das zweidimensionale Array *spurenListe_* beinhaltet Zeiger auf die Spuren in der Kreuzung. Existiert eine Richtung nicht, so befindet sich darin ein Zeiger auf *NULL*. Mit diesem Array ist es möglich, die Zustände auf den Spuren abzufragen und zu beeinflussen. Es ist somit dem Nutzer möglich, einen eigenen Algorithmus zu entwerfen und zu testen. Nach dem Umsetzen der letzten Programmierhürde steht der Algorithmus mit einer neuen ID (*n*) nach dem Kompilieren von TraffSpot, dem Nutzer zur Verfügung (*Kreuzung.cpp*):

```
void Kreuzung::setzeAmpelalgorithmus( int id ){
   if( id > 0 && id < n+1 ){
      ...
   }
}

void Kreuzung::takt( double zeitintervall ){
   try{
      switch( idAmpelAlgorithmus_ ){
         case 1: algorithmusZeitZeit( zeitintervall ); break;
         case 2: algorithmusZeitMenge( zeitintervall ); break;
         case 3: algorithmusStopp( zeitintervall ); break;
         case 4: algorithmusMengeMenge( zeitintervall ); break;
         case n: algorithmusNeu( zeitintervall ); break;
      }
   }
   catch( Fehler *fehler ){
      throw( fehler );
   }
}
```

4.2 Graphische Oberfläche

Die Simulation kann auch ohne eine graphische Benutzeroberfläche benutzt werden. Da in der heutigen Zeit der graphischen Betriebssysteme das Fehlen einer so genannte GUI vom Benutzer als störend empfunden wird, wurde auch TraffSpot mit einer GUI ausgestattet. Die GUI hat für den Benutzer verschiedene Vorteile, zum einen wird ein Programm transparenter, da für den Benutzer die Funktionen des Programms leichter ersichtlich werden. Zum anderen kann gerade bei einer Simulation der Nutzer 'live' das Geschehen im Simulationsuniversum verfolgen und leichter Einfluss darauf nehmen.

Die Simulation ohne GUI wird über eine Steuerklasse und deren Methoden gesteuert. Diese Vorgehensweise lässt sich mit dem Signal-Slot-Prinzip von Qt sehr gut verbinden. Es wird eine Oberfläche erzeugt, die ein Objekt der Klasse *TraffSpot* (Steuerklasse) instanziert. Die Knöpfe und Regler auf dem Hauptfenster (*MainWindow*) werden mit den Methoden der Steuerklasse verbunden und können somit die Simulation beeinflussen. Vergleichbar ist dies mit einer Marionette, die über ein mit Fäden verbundenes Holzkreuz gesteuert wird.

Um den Nutzer einen Einblick in die Simulation zu gewähren, wurde ein OpenGL-Widget in Qt erstellt. Dieses Widget kann auf dem Hauptfenster platziert werden und dominiert die GUI. Die Funktionalität des Widget findet sich in den Methoden seiner Klasse (*GLWirefame*) wieder. So ist es möglich, Spuren, Kreuzungen und Fahrzeuge durch OpenGL-Funktionen darzustellen. Die benötigten Daten werden von der Steuerklasse abgefragt und die erhaltenen Listen dargestellt. Implementiert wurde eine Maussteuerung, um im Simulationsuniversum navigieren und zoomen zu können. Dadurch kann das Geschehen aus verschiedenen Blickwinkeln betrachtet werden. Zur Umsetzung der Maussteuerung und des Widgets wurden die Beispiele /6/ für OpenGL-Widgets von Trolltech zu Rate gezogen.

4.3 XML-Parser

Der Parser hat eine primäre Aufgabe: Er muss die vom Benutzer in einer XML-Datei hinterlegte Welt in eine für die Software verständliche und benutzbare Speicherstruktur bringen.

Damit man gültige XML-Dateien erstellen kann, wurde für TraffSpot eine eigene Dokumenttypdefinition (DTD) entwickelt. In diesem Dokument wird die Struktur einer XML-Datei festgelegt. Folgende Elemente sind erforderlich: Spuren, Kreuzungen und Generatoren. In jedem Spuren-Element wird festgehalten, wo dieses liegt und welche Eigenschaften es hat. Des Weiteren können innerhalb der Spur bis zu drei Nachfolger definiert werden. In den Kreuzungs-Elementen wird festgelegt, welche Spuren zur jeweiligen Kreuzung gehören. Außerdem kann darin der zu benutzende Ampelalgorithmus festgelegt werden. In den Generatoren-Elementen werden alle nötigen Daten für jeden Fahrzeuggenerator hinterlegt, zum Beispiel, auf welcher Spur die Fahrzeuge entstehen sollen. Über ein im Generator befindliches Fahrzeug-Element können die Eigenschaften der zu generierenden Fahrzeuge festgelegt werden. Ein Fahrzeug kann, wenn es nicht nach Zufall bestimmt über den Parcour fahren soll, Fahrtenbucheinträge erhalten, in denen der Benutzer festlegen kann, in welche Richtung das Fahrzeug bei einer Kreuzung fahren soll. Weitere Informationen lassen sich im noch folgenden Abschnitt über die Benutzung finden, sowie in der *TraffSpot_1.0.dtd* im Anhang oder der beigefügten Beispiel-XML-Datei, welche auch im Anhang zu finden ist.

Die Implementierung des XML-Parsers bedient sich der Qt-Klasse *QXmlDefaultHandler*. Mit Hilfe der geerbten Methoden ist es möglich, ein XML-Dokument zu parsen und dabei auszuwerten /7/. Die XML-Datei wird in drei Durchläufen ausgewertet: Im ersten Durchlauf werden die Spuren gezählt. Im zweiten Durchlauf wird die größte ID, die eine Spur besitzt, festgestellt. Diese Angaben werden benötigt um das Feld der Spuren (Spurenarray) in der richtigen Größe initialisieren zu können. Nebenbei wird ein weiteres Hilfsfeld erstellt, das für die Verknüpfung der Spuren am Ende des Einleseprozesses von Nöten ist.

In das Spurenarray werden im dritten Durchlauf nun nach und nach alle Spuren eingehängt. Die Eigenschaften der Spuren sind zu diesem Zeitpunkt bereits in den Objekten gespeichert. Die Nachfolger einer Spur werden erst einmal nur als ID festgehalten. Parallel dazu wird die Position der Spur im Spurenarray im Hilfsarray an die Stelle seiner ID geschrieben (ähnlich einer Hashtable).

Des Weiteren werden beim Parsen auch Kreuzungen und Generatoren festgestellt. Diese werden über die Methode *datenHinzufuegen(ListenDaten *listenDaten)* in die Kreuzungsliste, beziehungsweise in die Generatorenliste eingehängt. Da sowohl die Klasse *Generator* als auch die Klasse *Kreuzung* von der Klasse *ListenDaten* erben, ist diese einfache Implementierung möglich.

Während des dritten Durchlaufs werden nicht nur die Attribute der Elemente ausgelesen und in den Objekten abgespeichert, es wird auch im Rahmen der Programmierung überprüft, ob in den Attributen richtige Werte enthalten sind. Ist dem einmal nicht so, wird ein Fehler generiert und über die Simulation an die GUI weitergereicht, in dem steht, bei welchem Element der fehlerhafte Eintrag gefunden wurde. Das Parsen der XML-Datei wird daraufhin abgebrochen.

Nachdem nun das dreimalige Durchlaufen der XML-Datei erfolgreich beendet wurde, liegt ein Spurenarray mit allen Spuren vor. Diese sind allerdings noch voneinander völlig losgelöst. Um diesen Umstand zu ändern, kommt das Hilfsarray ins Spiel. Das Spurenarray wird nun sequentiell durchlaufen und von jeder Spur die möglichen maximal drei Nachfolger ermittelt. Nun kann über die Nachfolger-ID das Objekt der Spur über das Hilfsarray erreicht werden, da die Position der Spur im Spurenarray im Hilfsarray abgelegt wurde. Nun werden die Spuren durch Absichern der Zeiger miteinander verbunden. Nach und nach entsteht so im Speicher ein Graph des gesamten Szenarios. Der Einstiegspunkt eines Fahrzeuges ist der Zeiger auf die aktuelle Spur. Über die Funktion *gibNaechsteSpur(int richtung)* erhält das Fahrzeug den Zeiger auf den Nachfolger und kann somit problemlos den Parcours befahren.

Nach Beenden des Verknüpfen der Spuren wird das Hilfsarray gelöscht, das Spurenarray wird allerdings von der Simulation für die Darstellung in OpenGL benötigt.

4.4 Tutorial

In diesem Abschnitt soll kurz gezeigt werden, wie TraffSpot benutzt werden kann. Als erstes muss ein Straßenverlauf erstellt werden. Dieser wird von dem Nutzer mittels einer XML-Datei dem Programm mitgegeben. Die XML-Datei muss zur Zeit allerdings noch von Hand erstellt werden.

4.4.1 Aufbau einer XML-Datei

Eine XML-Datei beinhaltet folgende Elemente:

```
<welt version="1.0"> ... </welt>
```

Im Welt-Element stehen alle anderen Elemente, die ein Szenario beschreiben. Dazu gehören:

```
<spur>
  <nachfolger />
</spur>

<kreuzung />

<generator>
  <fahrzeug />
  <fahrtenbucheintrag />
</generator>
```

In der Praxis hat sich die oben stehende Reihenfolge in organisatorischer Sicht bewährt. Die Elemente benötigen allerdings noch Attribute. Die Spur benötigt eine ID, die Länge in Meter, die Ausrichtung – also ob die Straße von oben gesehen horizontal (*HOR*) oder vertikal (*VER*) verläuft – die Richtung der Spur – ob sie von oben gesehen von links nach rechts geht (*LTOR*), oder entgegengesetzt (*RTOL*), bzw. von oben nach unten (*DOWN*) oder eben von unten nach oben (*UP*) – schließlich die Koordinaten für x, y und z und natürlich die erlaubte Höchstgeschwindigkeit. Ein Eintrag könnte wie folgt aussehen:

```
<spur id="200" laenge="500" ausrichtung="HOR" richtung="RTOL"
      x="500" y="1100" z="0" maximaleGeschwindigkeit="80" />
```

Um einer Spur Nachfolger mitzugeben, können in einem Spurenelement bis zu drei Nachfolgerelemente angeführt werden. Diese beinhalten jeweils die ID der nachfolgenden Spur, die Abbiegerichtung (*LINKS*, *MITTE*, *RECHTS*) und die Wahrscheinlichkeit, mit der auf den Nachfolger gewechselt wird. Eine Spur mit Nachfolgern kann in der XML-Datei folgendermaßen aussehen:

```
<spur id="201" laenge="500" ausrichtung="HOR" richtung="LTOR"
      x="0" y="1060" z="0" maximaleGeschwindigkeit="80">
  <nachfolger id="101" richtung="LINKS"
              wahrscheinlichkeit="0.3" />
  <nachfolger id="102" richtung="MITTE"
              wahrscheinlichkeit="0.3" />
  <nachfolger id="103" richtung="RECHTS"
              wahrscheinlichkeit="0.3" />
</spur>
```

Einem Kreuzungselement werden die an der Kreuzung beteiligten Spuren, sowie der zu verwendende Ampelalgorithmus und eine ID für die Kreuzung mitgegeben:

```
<kreuzung id="1" ampelalgorithmus="1"
     westenLinks="101" westenMitte="102" westenRechts="103"
     suedenLinks="111" suedenMitte="112" suedenRechts="113"
     ostenLinks="121"  ostenMitte="122"  ostenRechts="123"
     nordenLinks="131" nordenMitte="132" nordenRechts="133" />
```

Soll zum Beispiel eine T-Kreuzung nur Spuren aus Norden, Westen und Süden beinhalten, so werden die Attribute *ostenLinks*, *ostenMitte* und *ostenRechts* einfach weggelassen.

Zum Schluss sollten noch die Generatoren definiert werden: Dabei wird dem Generator eine ID zugeordnet, die ID der zu befüllenden Spur, die Pause zwischen dem Generieren von zwei Fahrzeugen in Sekunden und die Zahl der zu generierenden Fahrzeuge:

```
<generator id="2001" spurID="201" erstellzeit="5"
           fahrzeugAnzahl="500" />
```

Im obigen Fall wird 500 Mal alle fünf Sekunden zufällig ein Fahrzeug generiert. Will man nur einen bestimmten Fahrzeugtyp generieren lassen, gibt man dem Generator noch ein Fahrzeug-Element mit:

```
<generator id="2001" spurID="201" erstellzeit="5"
           fahrzeugAnzahl="500">
  <fahrzeug typ="PKW" laenge="4.1" lebensdauer="100" rot="0"
            gruen="128" blau="64" beschleunigungPositiv="6.2"
            beschleunigungNegativ="-7.5"
            hoechstgeschwindigkeit="180" />
</generator>
```

In diesem Beispiel erstellt nun der selbe Generator wie eben nur dunkelgrüne Pkws, die 4,1 Meter lang sind, 100 Spurwechsel lang existieren, mit 6,2 m/s² und mit -7,5 m/s² beschleunigen bzw. bremsen und maximal 180 km/h schnell fahren können. Soll das Fahrzeug so lange fahren, bis es irgendwann an einer Spur ohne Nachfolger verschwindet, muss die Lebensdauer auf -1 gesetzt werden. Der Fahrzeugtyp kann zwischen *PKW*, *LKW*, *BUS* und *BIKE* gewählt werden. Die Befüllung der restlichen Parameter mit sinnvollen Daten bleibt dem Benutzer überlassen.

Soll nun das Fahrzeug auch noch eine bestimmte Route abfahren, so kann man beliebig viele Fahrtenbucheinträge mitgeben:

```
<generator id="2001" spurID="201" erstellzeit="5"
           fahrzeugAnzahl="500">
  <fahrzeug typ="PKW" laenge="4.1" lebensdauer="100" rot="0"
            gruen="128" blau="64" beschleunigungPositiv="6.2"
            beschleunigungNegativ="-7.5"
            hoechstgeschwindigkeit="180" />
  <fahrtenbucheintrag abbiegerichtung="MITTE" />
  <fahrtenbucheintrag abbiegerichtung="RECHTS" />
  <fahrtenbucheintrag abbiegerichtung="LINKS" />
  <fahrtenbucheintrag abbiegerichtung="LINKS" />
  ...
</generator>
```

Jeder Eintrag gilt pro befahrener Spur. Ist bei einer Kreuzung die angegebene Abbiegerichtung nicht existent, so wird wie sonst auch durch Zufall entschieden, in welche Richtung das Fahrzeug abbiegt.

4.4.2 Szenario planen und umsetzen

Nachdem der Aufbau der XML-Datei und die benötigten Daten bekannt sind, kann der eigentliche Szenarioaufbau beginnen. Dabei geht man wie folgt vor: Zuerst wird der Straßenverlauf skizziert. Dabei sollte jede Spur einer Straße und einer Kreuzung einzeln eingezeichnet werden. In die Spuren trägt man nun gewünschte Maximalgeschwindigkeiten ein, und vergibt systematische IDs, um später die Spuren bei der Auswertung schneller wiederfinden zu können. Als nächstes werden die Längen an die Spuren geschrieben. Dabei hat sich ein Abstand zwischen den Spuren von ca. 20 Metern als optisch guter Abstand ergeben. Eine Spur selbst ist standardmäßig zwei Meter breit. Danach sollten die Ursprünge jeder einzelnen Spur eingezeichnet werden. Dabei ist die Ausrichtung und Richtung jeder einzelnen Spur ausschlaggebend. Die Abbildung 9 verdeutlicht dies. Der blaue Punkt ist dabei der Ursprung einer Spur und dessen Koordinaten sind in der XML-Datei anzugeben:

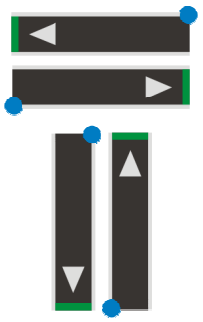


Abbildung 9: Zeichenrichtungen

Danach sollten die Spuren verbunden werden. Als nächstes können die Kreuzungen eingetragen werden. Jede der beteiligten Spuren erhält die Kennung L, M oder R um später das Editieren der XML-Datei zu beschleunigen. Als letztes werden die Generatoren an die Spuren eingefügt, auf denen neue Fahrzeuge entstehen sollen. Es sollte eine Zeichnung ähnlich der nachfolgenden Grafik entstehen:

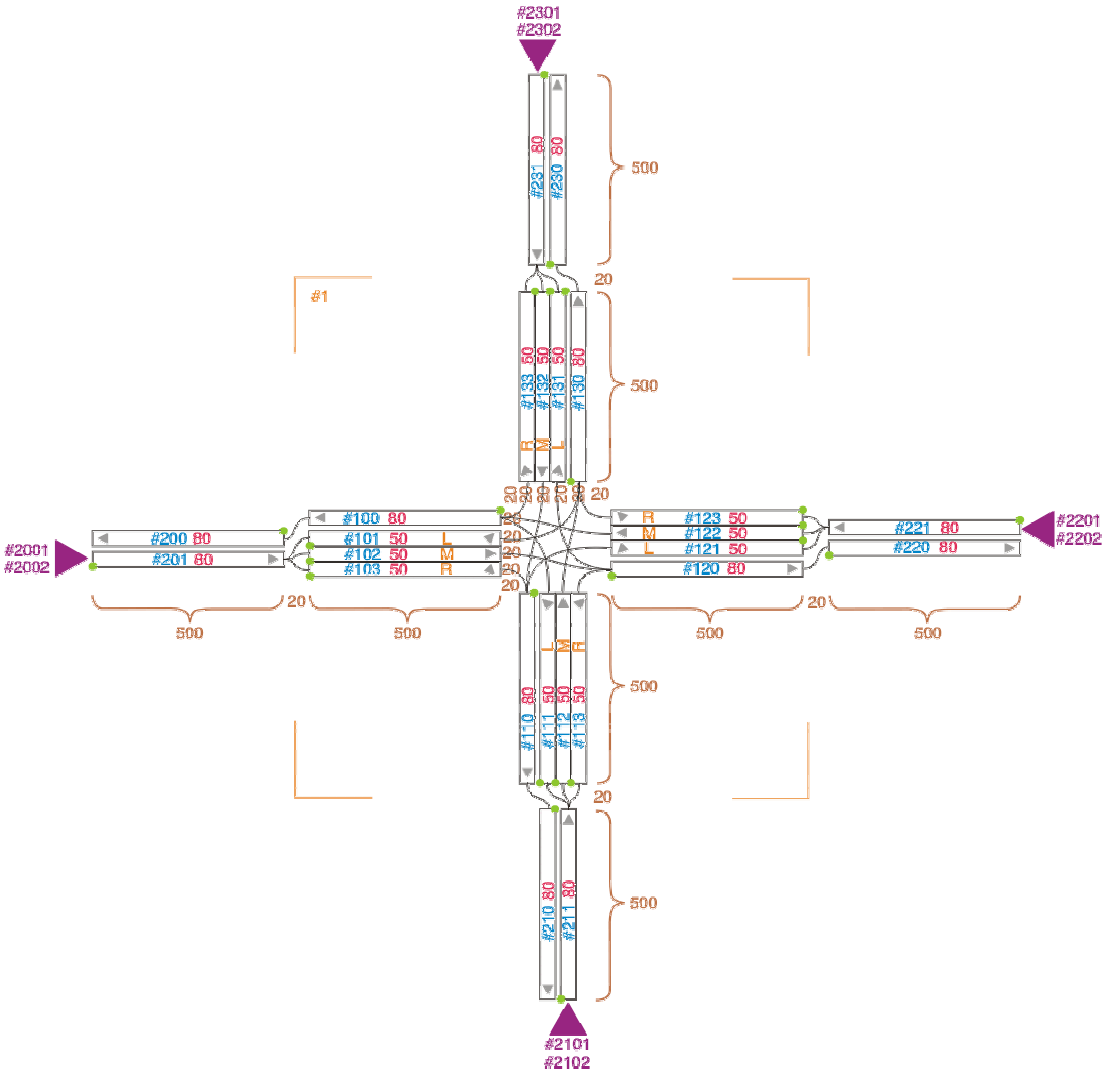


Abbildung 10: schematische Darstellung der im Anhang befindlichen Beispieldatei

Nun bleibt einem nichts anderes übrig, als alle Daten, die auf dem Papier stehen, in eine XML-Datei zu schreiben, eine Assistenz gibt es dafür noch nicht. Zu beachten ist, dass jeder Koordinatenpunkt auch von Hand berechnet werden muss – bei großen Strecken leider keine triviale Aufgabe. In der Anlage befindet sich das obige Beispiel als XML-Datei beigefügt (*strassenkreuz_statisch_zeitzeit.xml*).

Ist die XML-Datei nun fertig, kann sie mittels des W3C-Validators unter *http://validator.w3.org/* geprüft werden. Wird kein Fehler beanstandet, so stehen die Chancen nicht schlecht, dass TraffSpot sie einließt. Die vorletzte Prüfinstanz ist TraffSpot selbst. Bei den meisten Attributen wird geprüft, ob Werte darin stehen. Sind keine enthalten, so beendet TraffSpot das Einlesen der XML-Datei mit Ausgabe der Fehlerursache. Die letzte Prüfung ist das generierte Modell in TraffSpot selbst. Sieht es irgendwie verschoben aus, so sollte

von Hand in der XML-Datei nachgebessert werden – Erfahrungen haben gezeigt, dass dies bei großen Szenerien immer der Fall ist.

4.4.3 Simulation durchführen

Sind auch diese kleinen Verbesserungen erledigt, steht der Durchführung der Simulation nichts mehr im Wege. Dafür muss nur auf den Play-Knopf im Bereich *Simulationskontrolle* gedrückt werden.

Abbildung 11: Screenshot von TraffSpot im Simulationsbetrieb

Danach kann die Simulation pausiert oder gestoppt werden. Hat man die Simulation pausiert und drückt nochmals Play, so läuft die Simulation an der Stelle weiter, an der sie pausiert wurde. Ist die Simulation gestoppt worden und es wird auf Play gedrückt, so wird die Simulation von neuem gestartet.

Im Bedienfeld *Reaktionszeit* kann – wie der Name schon sagt – die Reaktionszeit der Verkehrsteilnehmer eingestellt werden. Im Bereich *Simulationszeit* wird die abgelaufene Zeit angezeigt. Daneben befindet sich eine so genannte Spinbox, in der eingestellt werden kann, wie schnell die Simulation ablaufen soll: Von Echtzeit (1x) bis hin zu 100 facher Beschleunigung.

Sind schlussendlich alle Fahrzeuge über den Parcours gefahren, so können die statistischen Werte, die nebenbei gesammelt wurden, abgespeichert werden. Dafür muss im Bereich *Werkzeuge* der Export-Knopf gedrückt werden. Es werden dann die Fahrzeug- und die Kreuzungsstatistiken in jeweils eine CSV-Datei gespeichert.

5 Verkehrsflussoptimierung

Um die Fahrzeuge sicher über eine Kreuzung fahren zu lassen, bedarf es verschiedener Regeln. Die einfachste Regel ist die *Rechts vor Links*-Regel. Darauf folgt eine Regelung des Verkehrsflusses durch Verkehrszeichen wie zum Beispiel Vorfahrt beachten oder das Stoppschild. Die komplexeste Art den Verkehrsfluss an einer Kreuzung zu regulieren, ist eine Lichtsignalanlage (kurz Ampel). Jede Spur an der Kreuzung wird mit einer Ampel ausgestattet. Dabei können auch mehrere Spuren zusammengefasst werden (zum Beispiel Rechtsabbieger und Geradeausfahrer). Wann und wie eine der Spuren ein Signal erhält, wird durch Ampelalgorithmen auf einer bestimmten Entscheidungsgrundlage gewählt. Die erste Entscheidung ist, zu welchem Zeitpunkt eine Spur beschaltet wird. Dies kann nach einem bestimmten Zeitpunkt oder bei einer bestimmten Menge an Fahrzeugen in dieser Spur geschehen. Wie lange eine Spur ein bestimmtes Signal beibehält, ist das Ergebnis der zweiten Entscheidungsmöglichkeit. Auch diese kann in zwei Richtungen unterschieden werden. Die Entscheidung kann auf Basis der abgelaufenen Zeit oder der von der Spur ausgefahrenen Menge an Fahrzeugen getroffen werden. Auf Grund dieser Aspekte werden die Ampelalgortihmen in der vorliegenden Studienarbeit in vier Hauptkategorien unterteilt:

- zeitorientiert / zeitorientiert
- zeitorientiert / mengenorientiert
- mengenorientiert / zeitorientiert
- mengenorientiert / mengenorientiert

Die im Folgenden vorgestellten Schaltungen sind alle lokale beziehungsweise dezentrale Schaltungen. Es ist möglich, ein ganzes Verkehrsnetz zu beschalten, so wie es zum Beispiel NONSTOP macht. Im Rahmen von TraffSpot wurde sich bewusst für eine lokale Schaltung entschieden. Möchte man ein ganzen Netz beschalten, so muss dieses erst einmal nachgebildet werden und es wird eine Steuerzentrale benötigt, die den Aufwand meistern kann. Benutzt man eine dezentrale Steuerung, so muss der Algorithmus nur in jede Ampelhardware geladen werden. Eine manuelle Anpassung an die aktuelle Kreuzung sollte nicht nötig sein, da die neuartigen Algorithmen so zu entwickeln sind, dass sie sich dynamisch auf die Kreuzungssituation anpassen können. Eine kostenaufwändige Zentralsteuerung entfällt dabei.

5.1 zeitorientiert / zeitorientiert

Unter die Rubrik zeitorientiert / zeitorientiert zählt der statische Ampelalgorithmus, der nach einem bestimmten Zeitplan schaltet. Im Zeitplan wird festgehalten, wann eine bestimmte Spur ein Signal bekommen soll. Da das Verhalten für jeden Zeitpunkt genau vorhersagbar ist, zählt dieser Algorithmus nicht zu den spekulativen Algorithmen. Der Vorteil

liegt in einer sehr einfachen technischen Umsetzung, denn für die Ampelschaltung wird nur ein Taktgeber mit einem Demultiplexer[8] benötigt.

Diese Art von Algorithmus wurde als erster Schritt in die Software TraffSpot implementiert. Es zeigt sich, dass er für eine Kreuzung, auf welcher alle Spuren ungefähr gleichstark befahren werden, akzeptable Werte liefert. An Kreuzungen mit unterschiedlicher Belastung kann der Algorithmus durch Verschieben der Zeitachsen an die Begebenheiten angepasst werden. Allerdings ist dies nur in begrenzten Maßen möglich und bedarf eines manuellen Eingriffs.

5.2 zeitorientiert / mengenorientiert

Wird die Ampelanlage, auf welcher der statische Ampelalgortihmus läuft, um Sensoren für die Verkehrsflussmessung der Spuren über die Ampel ausgestattet, können diese Sensoren genutzt werden, um einen Mindestdurchfluss von Fahrzeugen zu gewährleisten. Sie dienen zur Unterstützung des Zeitplanes, da dieser an veränderte Umgebungsbedingungen angepasst werden kann.

Dieser Ansatz wurde als zweites implementiert. Nach den Tests zeigte sich eine Verschlechterung gegenüber dem statischen Ampelalgorithmus. Dies liegt darin begründet, dass durch den Sensor nur die abfahrenden Fahrzeugen gezählt werden. Es kann auf Basis dieser Daten eine Mindestanzahl von Fahrzeuge festgelegt werden, die über die Kreuzung fahren sollen. Es wird allerdings immer eine maximale Wartezeit an der Kreuzung mitgeführt, die verhindern soll, dass an einer Ampel zu große Wartezeiten durch eine schwach befahrene Spur entstehen. Diese Ampelschaltung kann unter günstigen Bedingungen und Einstellungen maximal einen Verkehrsfluss in gleicher Höhe des statischen Ampelalgorithmus bringen.

Mit dem Algorithmus kann auch das Problem einer sehr stark belasteten und einer unbelasteten Spur an einer Kreuzung nicht gelöst werden. Denn nach einem bestimmten Zeitpunkt wird die unbelastete Spur auf Grün geschaltet, auch wenn sich in dieser Spur überhaupt kein Fahrzeug befindet. Diese Grünphase bleibt für die maximale Wartezeit bestehen.

5.3 mengenorientiert / zeitorientiert

Weitere Sensoren können für die Messung der Anzahl der Fahrzeug in den Spuren vor einer Ampel genutzt werden. Wird eine bestimmte Anzahl Fahrzeuge überschritten, dann wird eine Spur auf ein Signal gesetzt. Dieses Signal bleibt mindestens für einen Zeitabschnitt Δt bestehen. Danach wird wieder eine Mengenmessung der Fahrzeuge vor den einzelnen Ampeln durchgeführt und die Schaltung dementsprechend vorgenommen.

8 Auch DEMUX genannt. Dies ist ein Bauteil der Digitaltechnik, mit dem ein Eingangssignal auf eines von mehreren Ausgangssignalen gelegt werden kann.

Diese Art Algorithmus wurde in TraffSpot nicht umgesetzt. In der Theorie zeigt sich schon, dass dieser Algorithmus Schwächen hat, die unter Umständen den Verkehrsfluss komplett zum Erliegen bringen könnten. Es könnte vorkommen, dass die Ampel erst bei einer bestimmten Menge an Fahrzeugen in einer Spur schaltet, wobei der Füllgrad der Spur erst nach längerem Zeitraum erreicht werden kann. Somit kann es zu langen Wartezeiten kommen. Durch eine manuelle Einstellung könnte dieses Risiko minimiert werden. Allerdings sind dann keine großen Vorteile gegenüber dem statischen Ampelalgorithmus zu erkennen.

5.4 mengenorientiert / mengenorientiert

Alle schon erwähnten Sensoren an einer Ampel eingesetzt, ergeben eine Ampelschaltung, welche durch mengenorientierte / mengenorientierte Algorithmen beschaltet werden kann. Es wird ausgewertet, wie viele Fahrzeuge in einer Spur stehen oder fahren. Wird ein bestimmter Wert überschritten, so kann eine Spur auf Grün gesetzt werden. Diese Schaltung bleibt bestehen, bis eine weitere bestimmte Anzahl von Fahrzeugen über die Kreuzung gefahren ist. Dieser Algorithmus wurde in TraffSpot integriert. Durch diesen Algorithmus kann eine unnötige Beschaltung von leeren Spuren mit Grün verhindert werden. Es zeigte sich allerdings auch, dass durch eine schlechte Annahme der Fahrzeuganzahl wieder extrem lange Wartezeiten entstehen können. Es muss ein sinnvolles Verhältnis zwischen der Anzahl Fahrzeuge vor der Ampel und Fahrzeuge, die über die Ampel fahren, ermittelt werden. Dazu dient das Dynamisieren der vorgestellten Algorithmen.

5.5 dynamisieren

Mit Hilfe von Speichern in Ampeln können die Algorithmen dynamischer gestaltet werden. So können die Verkehrsflüsse analysiert werden und so eine Verschiebung der Mengen- und/oder Zeitachsen bewirkt werden. Der Algorithmus wäre selber in der Lage, auf die Auslastung der Spuren zu reagieren und könnte ohne manuelle Anpassungen auf allen Kreuzungen eingesetzt werden. Es würde eine kurze Eingewöhnungszeit für den Algorithmus benötigt, bis dieser optimal auf die Erfordernisse angepasst ist. Allerdings ist diese Zeit vermutlich sehr gering im Verhältnis zu den Einsparungen an Zeit nach der Eingewöhnung.

Ein solcher Algorithmus wäre in der Lage, auf Verkehrslagenänderungen schnell, automatisch und effizient zu reagieren. Beispiele für Verkehrslagenänderungen können im täglichen Berufsverkehr Umleitungen auf Grund von Baustellen oder Unfällen sein. Die Ampeln sind in der Lage, auf die Veränderungen ohne Eingreifen der Verkehrsbetriebe zu reagieren und den Verkehrsfluss optimal einzustellen.

Ein erster Ansatz wurde in TraffSpot umgesetzt. Dieser Algortihmus ist ein dynamisierter mengenorientierter / mengenorientierter Ansatz. Es wird die Anzahl der Fahrzeuge vor den Ampeln analysiert. Darauf basierend wird eine Reihenfolge der Grünphasen auf den Spuren

ermittelt (die Spur mit den meisten Fahrzeugen bekommt als erstes Grün), leere Spuren werden nicht betrachtet und erhalten somit nicht zwingend Grün. Beim Aufstellen der Schaltreihenfolge muss das Zusammenspiel zwischen den Spuren berücksichtigt werden (z.B. alle Rechtsabbieger bekommen auf einmal Grün). Als letzter Schritt werden den Schaltschritten die Menge an Fahrzeuge angefügt, die während des Schrittes über die Kreuzung fahren müssen. Ist diese Anzahl Fahrzeuge über die Kreuzung gefahren, so wird der nächste Schaltschritt vollführt. Wurden alle Schaltschritte durchgeführt, wird eine neue Betrachtung angestellt und ein neuer Schaltplan aufgestellt.

In den Versuchen zeigte sich eine Effizienzsteigerung gegenüber dem statischen Ampelalgorithmus. Erster Vorteil war das Auslassen der leeren Spuren. Als weiterer Vorteil konnte erkannt werden, dass eine Spur nur solange Grün bekommt, wie sich auch Fahrzeuge in der Spur befinden (neu einfahrende Fahrzeuge werden vernachlässigt). Zusammenfassend lässt sich sagen, dass der Algorithmus ähnlich einer Toillettenspülung wirkt. Alle Fahrzeuge, die sich zu einem bestimmten Zeitpunkt t auf der Kreuzung befinden, werden über die Ampel geschleust. Danach wird eine neue Betrachtung angestellt.

Weitere Erweiterungsmöglichkeiten sind in der Schaltreihenfolge zu sehen. So kann in die Reihenfolge eine Priorisierung eingefügt werden (ähnlich der Prozesspriorität beim Scheduler im Betriebssystem). Diese Priorisierung muss die Anzahl der Fahrzeuge mit der Wartezeit verknüpfen. Es wäre dann ein sinnvoller Schritt, nach jeder Einzelschaltung eine neue Betrachtung anzustellen. Beschaltet werden müsste immer die am oberen Ende der Prioritätenschlange befindliche Spur. In einer Matrix sollte abgelegt werden, welche Spuren mit welcher anderen gemeinsam beschaltet werden kann. So kann die Prioritätenschlange durchgegangen werden, bis zu der Spur, welche mit der ersten zusammenpasst. Dies wird bis zum Ende der Schlange durchgeführt. Danach kann eine reale Schaltung durchgeführt werden. Ein erster Ansatz für die Priortätenberechnung wäre es, die Anzahl Fahrzeuge mit der Wartezeit (in Schaltungen oder reale Zeit) zu multiplizieren. Somit würde eine schwach befahrene Straße einen Augenblick länger auf Grün warten müssen, als eine stark befahrene Straße. Der Gesamtdurchfluss sollte allerdings ansteigen. Auf Grund der Komplexität der vorliegenden Simulation konnte dieser Ansatz nur theoretisch erörtert werden.

6 Praxistest

Nachdem nun die neuen Ampelalgorithmen in der Theorie ihre Vor- und Nachteile bereits gezeigt haben, sollen diese auch in der Simulation bewiesen werden. Dafür gibt es drei Beispiele, die die Unterschiede gut zur Geltung bringen. Alle benutzten XML-Szenarien befinden sich auch im Anhang auf beigefügter CD.

6.1 Drei Kreuze

Als erstes Szenario wurde eine einfache Kreuzung für sich allein stehend drei mal implementiert, so dass drei Ampelalgorithmen unabhängig voneinander im Einsatz bewertet werden können.

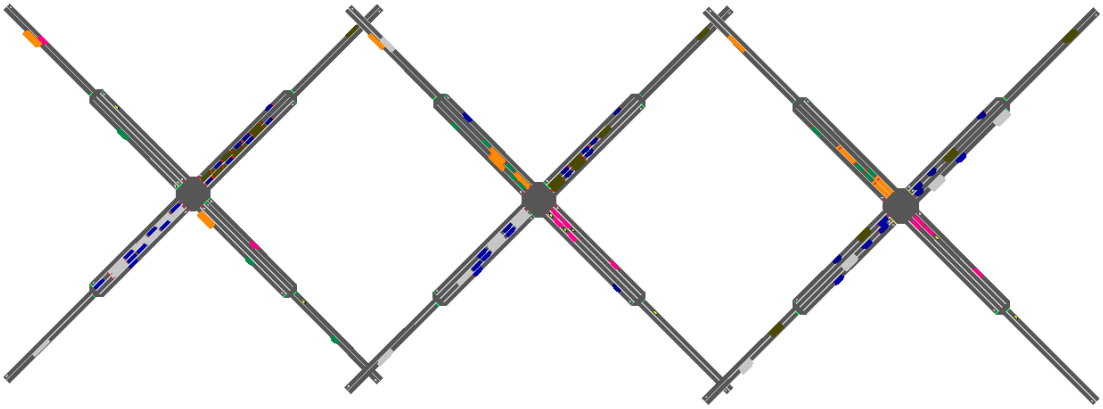


Abbildung 12: von links nach rechts: zeit / zeit, zeit / menge, menge / menge

Die Kreuzungen wurden so mit Generatoren bestückt, dass zur gleichen Zeit bei jeder Kreuzung der selbe Typ Fahrzeug generiert wird. Insgesamt lief die Simulation zehn Minuten[9]. Rein optisch bewertet scheint der zeitorientierte / mengenorientierte-Algorithmus (zeit / menge) am schlechtesten abzuscheiden, da sich auf der mittleren Kreuzung immer relativ wenig Fahrzeuge bewegen. Der klassische zeit / zeit-Ansatz scheint sich gut zu halten, da sich bei den langen Grünphasen auch später entstehende Rückstaus abbauen können. Die Generatoren liefern in diesem Beispiel so viele Fahrzeuge, dass sie irgendwann die Kreuzung überfluten. Wird nicht effizient geschaltet, stehen viele Fahrzeuge still. Der neuartige menge / menge-Algorithmus scheint die beste Wahl für diese Situation zu sein. Dort be-

9 Das Zeitlimit wird durch den Algorithmus gesetzt, bei dem als erstes keine Fahrzeuge mehr auf dem Parcours sind.

wegen sich immer Fahrzeuge und müssen relativ kurz auf Grün warten. Teilweise bestätigt werden diese Annahmen durch die erhobenen statistischen Daten:

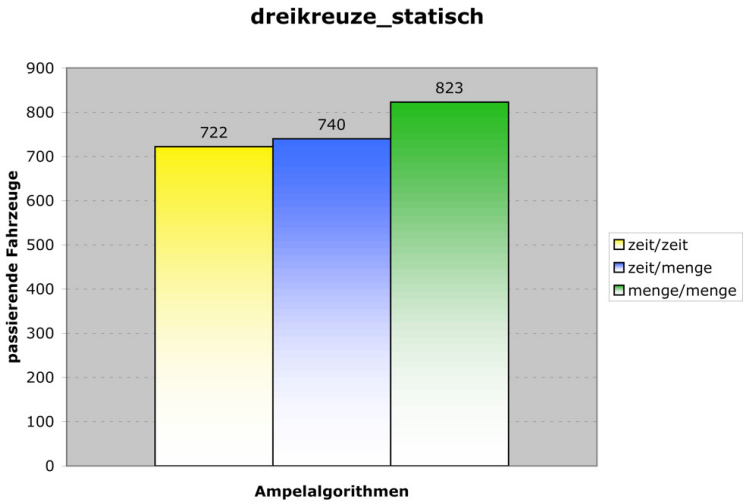


Abbildung 13: Auswertung der Ampelalgorithmen auf einer Kreuzung

Hier zeigt sich deutlich die Überlegenheit des menge / menge-Ansatzes. Im Vergleich zum Standardansatz zeit / zeit kann der neue Algorithmus in gleicher Zeit knapp 14% mehr Fahrzeuge über die Kreuzung lotsen (101 Fahrzeuge mehr). Bemerkenswert ist auch, dass die vorverurteilte zeit / menge-Schaltung am Ende gar nicht so schlecht da steht, als beobachtet. Sie schafft immerhin 18 Fahrzeuge mehr über die Kreuzung als der klassische Ansatz. Allerdings sind diese 2% keine wirkliche Verbesserung, da bei dieser Schaltung die Fahrzeuge häufiger anfahren und wieder anhalten müssen, bevor sie die Kreuzung hinter sich lassen können. Im folgenden wird diese Schaltung auch nicht weiter betrachtet, da sie kaum Potential zur Lösung der Problematik bietet.

6.2 Die Quadrate von Mannheim[10]

Als nächstes soll der menge / menge-Algorithmus in einem größeren Netz das vermutete bessere Schaltvermögen beweisen. Dazu wurde ein Netz aus zwölf Kreuzungen erstellt:

10 Eine Anlehnung an den quadratischen Stadtkern Mannheims, in dieser Stadt wurde auch die XML-Datei geschrieben.

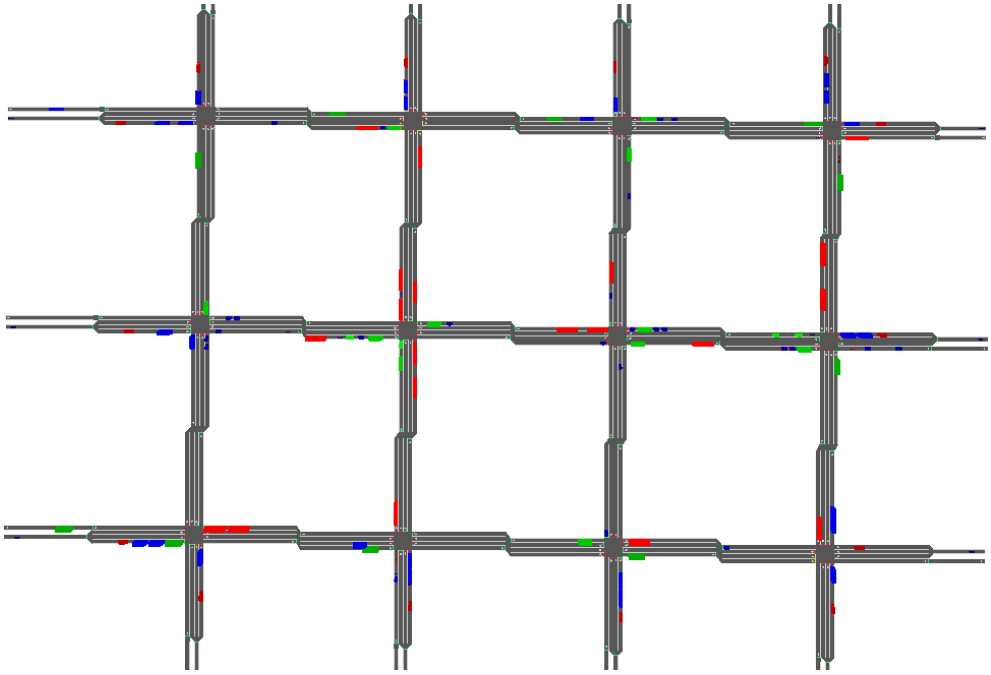


Abbildung 14: Straßennetz jeweils mit zeit / zeit bzw. menge / menge-Algorithmus beschaltet

Diese Simulation lief elf Minuten und 38 Sekunden. Die Generatoren sind in diesem Beispiel nicht auf Überlastung des Netzes eingestellt, sondern liefern nur einen konstanten Strom an Fahrzeugen. Die Auswertung zeigt auch hier wieder ganz deutlich, dass der neuartige Algorithmus eine höhere Durchsatzrate hat. Im Schnitt kann jede einzelne Kreuzung 42 Fahrzeuge mehr 'bearbeiten', wie die nachfolgende Auswertung zeigt, als der statische Ansatz, also ca. 11% mehr.

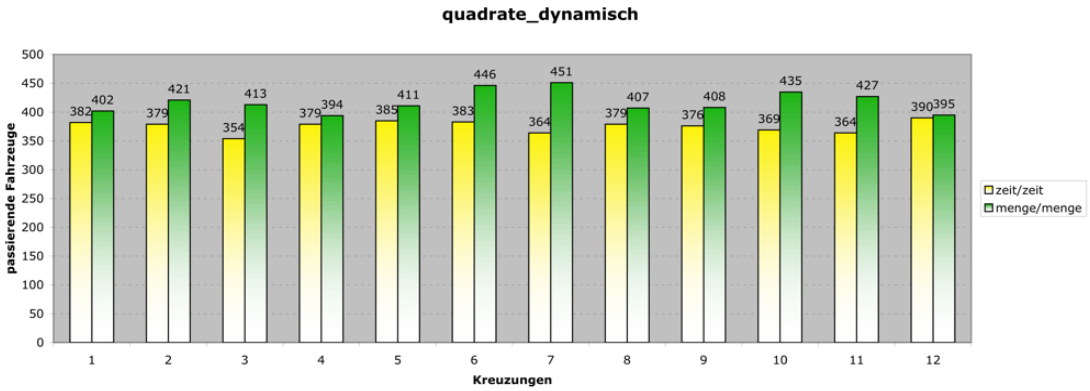


Abbildung 15: alle zwölf Kreuzungen im direkten Vergleich zu einander

Auch die zusammenfassende Statistik beider Durchläufe bestätigt das Ergebnis aus dem ersten Test: Der mengenorientierte / mengenorientierte Ansatz ist der klassischen Schaltung überlegen.

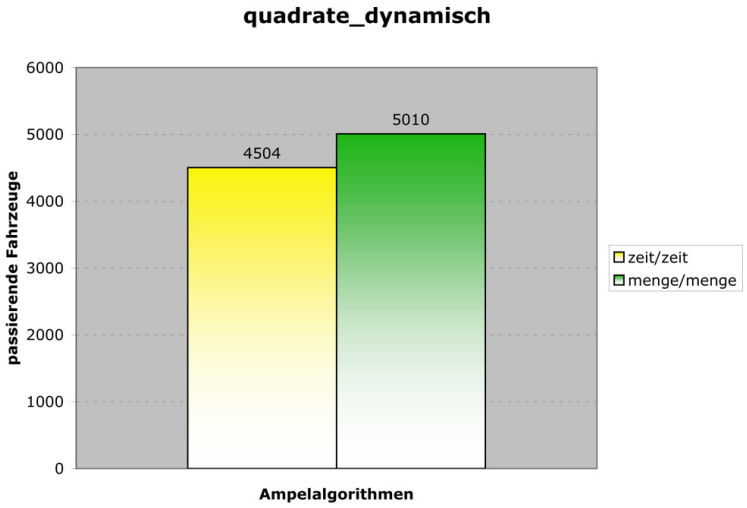


Abbildung 16: menge / menge-Algorithmus ist wieder effizienter

6.3 Der Kleinstadttest

Die bisherigen Test sind allerdings noch in einer nahezu idealen Umgebung abgelaufen. Von jeder Seite sind ungefähr gleich viele Fahrzeuge gekommen, alle Kreuzungen hatten somit gleich belastete Spuren. Dies findet man in der Realität nicht so häufig, deswegen soll es einen letzten Test geben. Testgelände ist die 'Kleinstadt'. Sie besteht aus drei Kreuzungen mit jeweils unterschiedlicher Anzahl an Spuren, die zusätzlich auch eine unterschiedliche Benutzungswahrscheinlichkeit aufweisen:

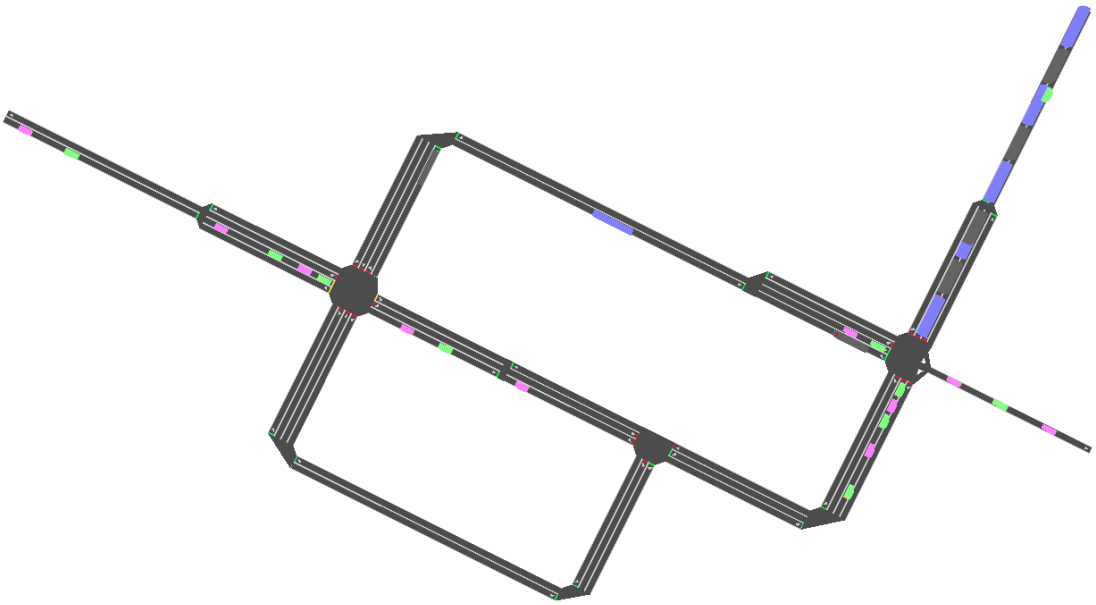


Abbildung 17: nicht symmetrisch aufgebautes Szenario

Auf der linken Seite befindet sich Kreuzung 1, die zum Beispiel von Osten kommend keine geradeaus führende Spur besitzt. Kreuzung 2 (in der Mitte der Darstellung) ist eine einfache T-Kreuzung ohne Spuren im Norden und Kreuzung 3 auf der rechten Seite hat im Osten nur eine abfließende Spur.

Um das Ergebnis vorweg zu nehmen, auch hier hat der menge / menge-Algorithmus triumphiert. In der Einzelbetrachtung liegen die Schaltungen nach knapp 18 simulierten Minuten im Schnitt 175 Fahrzeuge auseinander, die menge / menge-Entscheidung dirigiert 38% mehr Fahrzeuge über die Kreuzung, als der klassische Ansatz zu leisten vermag. Zum Vergleich lief der Test vollständiger Weise auch noch mit dem zeit / menge-Algorithmus. Das Ergebnis zeigt deutlich, dass dieser Ansatz keine Zukunft hat.

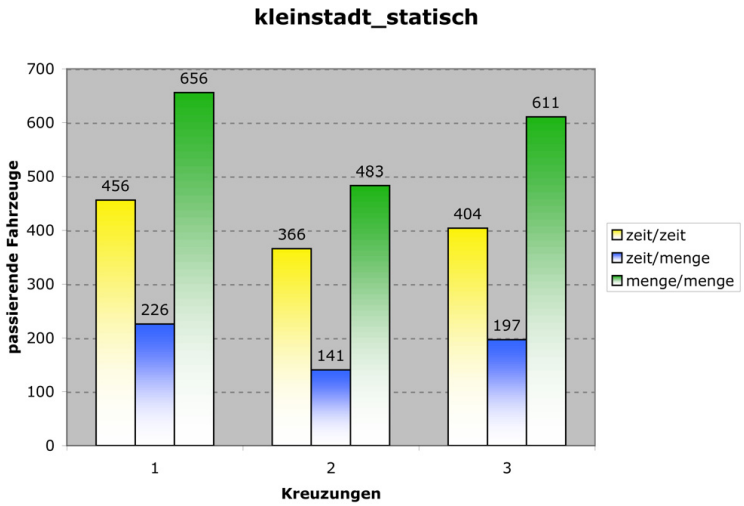


Abbildung 18: die Unterschiede der Schaltungen werden bei inhomogenen Verkehrsnetzen deutlicher

Summiert man alle Fahrzeuge, die die drei Kreuzungen zusammen koordinieren, fällt das Ergebnis noch drastischer aus. Die zeit / zeit-Schaltung kann 524 Fahrzeuge weniger über die Kreuzungen bringen und erreicht damit gerade einmal 70% der Leistung des menge / menge-Ansatzes. Das ist schon ein beeindruckendes Ergebnis:

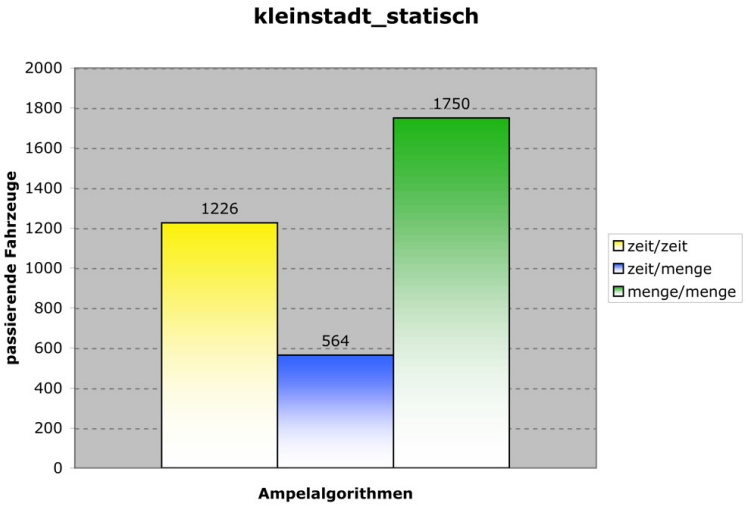


Abbildung 19: Klarer Favorit: Die mengenorientierte/mengen-orientierte Schaltung

Die Erklärung für dieses Phänomen ist relativ leicht. Der neue spektakuläre Ansatz überprüft jedes Mal die Anzahl der Fahrzeuge auf einer Spur. Sind einige Spuren nicht vorhanden, so werden diese auch nicht in die Berechnung einbezogen und somit nur beschaltet,

wenn sie im Rahmen eines neuen Schaltvorgangs sozusagen mit beschaltet werden könnten. Somit wird keine Zeit mit Grünphasen für Spuren verschwendet, die es eigentlich gar nicht gibt.

Der zeit / menge-Ansatz hat das Problem, dass er bereits nach einer bestimmten Anzahl von Fahrzeugen schaltet. Ist diese Anzahl zu gering eingestellt, erreicht man keinen hohen Durchfluss. Ist eine Spur Grün, aber kein Fahrzeug darauf, wird so lange gewartet, bis eine maximale Zeit vorbei ist. Währenddessen warten andere Fahrzeuge noch länger vor Rot.

Man muss allerdings auch eingestehen, dass der letzte Test nicht ganz fair abgelaufen ist. Der statische Ansatz wurde einfach auf die unterschiedlichen Kreuzungen projiziert. Eigentlich hätte dieser angepasst werden müssen, da zum Beispiel bei der T-Kreuzung andere Spurenfolgen gleichzeitig beschaltet werden könnten. Allerdings hat diese Benachteiligung auch einen Vorteil: Dies zeigt nämlich, dass der menge / menge-Algorithmus universell eingesetzt werden kann, so dass eine Nachkalibrierung von Menschenhand nicht nötig ist, wie sie bei der zeit / zeit-Schaltung unbedingt erforderlich wäre.

7 Ausblick

Bei der Entwicklung von TraffSpot haben sich schon an einigen Stellen mögliche Verbesserungen gezeigt, die allerdings in der Kürze der Zeit nicht umsetzbar sind. So wäre es prinzipiell schön, wenn die Simulation durch Klicken auf die Grafik direkt in der grafischen Benutzeroberfläche beeinflussbar wäre. Denkbar wäre hier das dynamische Wechseln der Ampelalgorithmen jeder einzelnen Kreuzung oder das Verhalten eines Fahrzeuggenerators während der Simulation zu ändern. Das Problem dabei ist allerdings, dass die Grafikausgabe direkt mit OpenGL realisiert wird. Eine Funktionalität, die zum Beispiel das angeklickte Objekt zurück liefern kann, wird von OpenGL nicht unterstützt. Eine mögliche Lösung ist das Benutzen eines Toolkits, wie zum Beispiel OpenSceneGraph /8/. Dadurch, dass OpenSceneGraph ein 3D-Toolkit ist, bietet es unter anderem die Möglichkeit, dieses oben beschriebene Klick-Ereignis umsetzen zu können. Auch gibt es einen Wrapper für OpenSceneGraph, der das Einbinden in ein Qt-Programm erleichtert. Allerdings ist das Wechseln der Zeichenmechanismen zu einem späten Zeitpunkt nicht mehr trivial. In diesem Fall müsste die GLWireframe-Klasse ersetzt werden. Auf Grund der Modularisierung sollte das später allerdings nicht allzu große Schwierigkeiten bereiten.

Des Weiteren hat sich gezeigt, dass das Erstellen von XML-Dateien ein langwieriger und verbesserungswürdiger Prozess ist. Jede Spur muss einzeln definiert werden und die Positionsdaten der Spur müssen vom Benutzer von Hand ausgerechnet werden. Es hat sich gezeigt, dass dieser mühsame Prozess stark fehleranfällig ist. Wünschenswert wäre hier ein Wizard, in dem man grafisch die Spuren zum Beispiel mittels Drag&Drop zurechtschieben könnte und über eine Eingabemaske die restlichen nötigen Daten einzutragen wären, zum Beispiel die ID der Spur oder die dort zu fahrende Höchstgeschwindigkeit. Ist man schließlich fertig, generiert das Hilfsprogramm eine XML-Datei mit diesem Szenario. Auch diese Problematik ist nicht einfach zu lösen, wie eben erwähnt, wird dafür ein eigenständiges Programm benötigt, dass den nicht geringen Funktionsumfang bereithält. Die Entwicklung dafür kann gut und gerne eine eigene Studienarbeit zum Thema haben.

8 Fazit

TraffSpot ist ein kleines schlankes Simulationswerkzeug. Es können durch den Benutzer erstellte Szenarien als XML-Datei eingelesen werden, die in der Software so im Speicher organisiert werden, dass sich Fahrzeuge frei auf dem erstellten Graphen bewegen können. TraffSpot arbeitet mit OpenGL, damit die Verkehrsströme sichtbar gemacht werden können und so die Effizienz der implementierten Ampelalgorithmen auf den ersten Blick sichtbar wird. Des Weiteren kann nun jeder mit TraffSpot auf einfache Weise neue Ampelschaltungen testen. Man kann über Funktionen im Quellcode neue Funktionalitäten zukünftiger Ampelanlagen nachbilden und somit neue Ideen ausprobieren.

Mit diesem neuen Hilfsmittel namens TraffSpot wurde ein neuer Ampelalgorithmus implementiert, der mit Hilfe von Sensoren die Anzahl der Fahrzeuge auf den Abbiegespuren feststellen kann. Mit Hilfe dieses Wissens ist die Schaltung in der Lage, bei der nächsten Grünphase so zu schalten, dass die meisten Fahrzeuge über die Kreuzung kommen. Im Test können die neuen Schaltungen über Messdatenerhebung mit dem klassischen Ansatz verglichen werden. So zeigte sich, dass der neue so genannten menge / menge-Algorithmus eine Leistungssteigerung von elf bis je nach Situation über 30% bringen kann. Ein Potential in dezentraler Ampelsteuerung ist also definitiv vorhanden und kann mit TraffSpot ermittelt werden.

9 Quellenverzeichnis

/1/ GEVAS software
http://www.gevas.de/Losungen/Page10773/NONSTOP/index.html
Kurzvorstellung der Software NONSTOP mit Auflistung diverser Features.
(zuletzt überprüft: 19.05.2005)

/2/ TransVer
http://www.transver.de/transver/de/office/F20000922143144471.htm
Beschreibung des Projekts BALANCE und Einblick in das Leistungsvermögen.
(zuletzt überprüft: 19.05.2005)

/3/ Hamburger Abendblatt
http://www.gevas.de/Aktuelles/index.html
Archivierter Zeitungsartikel über die Installation neuer Ampelsysteme in Hamburg.
(zuletzt überprüft: 19.05.2005)

/4/ SUMO
http://sumo.sourceforge.net/index.html
Startseite des Projekts SUMO mit weiterführenden Links.
(zuletzt überprüft: 19.05.2005)

/5/ Trolltech
http://www.trolltech.com/products/qt/index.html
Überblick über das Leistungsvermögen der aktuellen Qt-Version.
(zuletzt überprüft: 19.05.2005)

/6/ http://doc.trolltech.com/3.3/opengl-examples.html
An Hand von Beispielen wird die Funktion von OpenGL in Qt erklärt.
(zuletzt überprüft: 23.05.2005)

/7/ Helmut Herold:
Das Qt Buch: portable GUI-Programmierung unter Linux/Unix/Windows
SuSE-Press, 2001 ISBN 3-934678-76-9

/8/ OpenSceneGraph
http://www.openscenegraph.org
Startseite von OpenSceneGraph mit weiterführenden Links.
(zuletzt überprüft: 19.05.2005)

/9/ http://de.wikipedia.org
Wikipedia – Die freie Enzyklopädie

/10/ Herbert Schildt
C++ ENT-PACKT
MITP-Verlag, 2001 ISBN 3-8266-0731-7

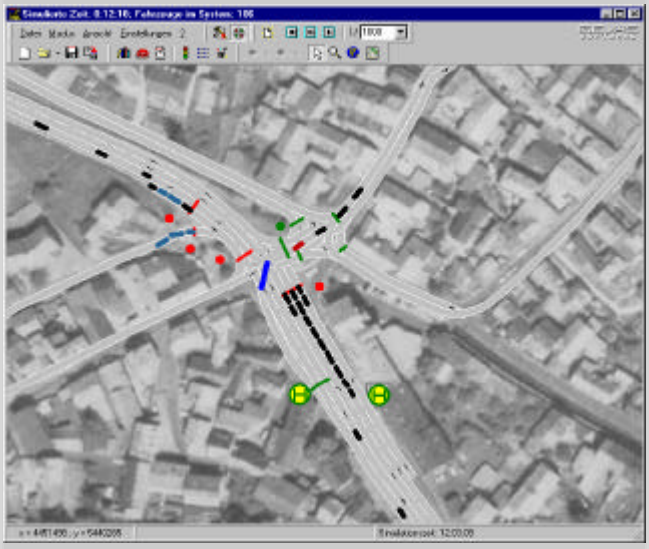
/11/ Hartmut Helmke, Rolf Isernhagen
Softwaretechnik in C und C++ – Das Lehrbuch
HANSER, 2001 ISBN 3-446-21683-9

/12/ Mason Woo, Jackie Neider, Tom Davis, Dave Shreiner
OpenGL Programming Guide (Third Edition)
Addison-Wesley, 2002 ISBN 0-201-60458-2

10 Anhang

- Flyer Nonstop
- Flyer Balance
- strassenkreuz_statisch_zeitzeit.xml
- TraffSpot_1.0.dtd
- Klassendiagramm Simulation
- Klassendiagramm XML-Parser
- Arbeitsplan
- Programm inklusive Dokumentation und Beispieldateien auf CD

Die Verkehrsfluss-Simulation zur Planung und Qualitätsanalyse von Steuerungen



Projektierungen und Funktionstests von Verkehrssteuerungen am Einzelknoten oder in Netzen - ob mit oder ohne Lichtsignal-anlagen - sind heute nicht mehr möglich ohne die Simulation des Verkehrsgeschehens. Nur sie stellt einen möglichst hohen Qualitätsstandardder Steuerungen sicher.

NONSTOP wurde entwickelt, um nicht nur die genaue Nachbildung der Knotenpunkt-steuerungen zu gewährleisten, sondern auch Netzsteuerungsverfahren exakt zu emulieren. Der schnelle und einfache Netzaufbau, umfassende Diagnose-möglichkeiten sowie die COM- und CORBA-Schnittstellen machen **NONSTOP** zum idealen Werkzeug des Verkehrsingenieurs.

The traffic flow simulation for planning and evaluating traffic control

Planning and operating tests of traffic control in road junctions or in traffic networks - whether with or without signal light systems - are today no longer possible without simulating traffic events. This guarantees a high standard of quality in traffic control.

***NONSTOP** was developed not only to ensure an exact reproduction of road junction control, but also to copy network control processes precisely. The quick and easy network construction and comprehensive opportunites for making a diagnosis make **NONSTOP** the traffic engineers 'ideal tool.*

NONSTOP

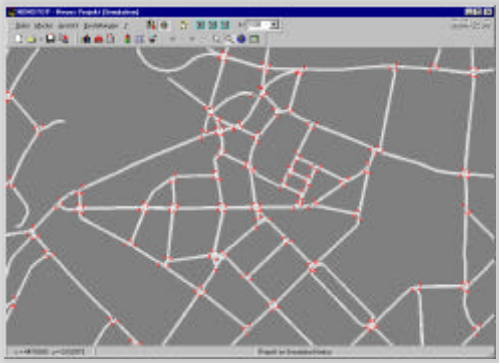
GEVAS SOFTWARE

OCIT-konform

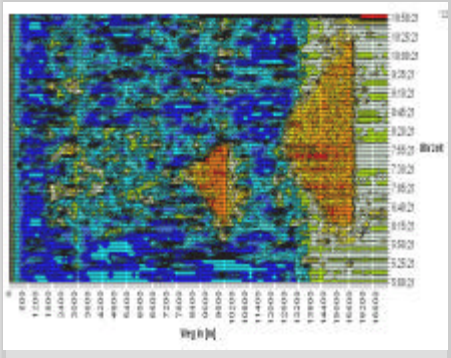
NONSTOP ist das vielseitige Tool zum Aufbau und zur Validierung von Verkehrsnetz-Steuerungssystemen. Die exakte Verkehrsabbildung, der schnelle Netzaufbau und die offenen Schnittstellen zu anderen Programmen und Verfahren machen **NONSTOP** zum unentbehrlichen und praktischen Helfer für den Planer.

Verkehrsabbildung und Netzaufbau

+ + + Berücksichtigung von Fußgängern + + + Beachtung von überstauten Knoten + + + exakte Behandlung von Links- und Rechtsabbiegern + + + realistisches Anfahr- und Abbremsverhalten + + + genaue Nachbildung des ÖPNV + + + basiert auf dem GIS-Standard der Firma ESRI™ + + + verkehrstechnische Versorgung in der komfortablen graphischen Oberfläche + + + intuitive Benutzerführung mit allen WINDOWS-Standards + + + schnelle und einfache Grundversorgung durch den Netzaufbau-Wizard + + + Steuerung der Lichtsignalanlagen durch den TRENDS-Kern oder durch Festzeitprogramme aus CROSSIG + + +



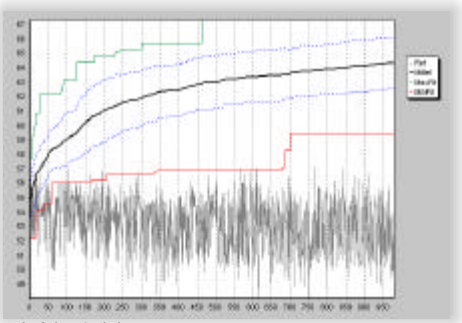
Schnittstellen und Auswertungsmöglichkeiten



+ + + Unterstützung der Kommunikations-Standards COM/DCOM und CORBA + + + Mehrplatz-Betrieb möglich + + + jede moderne Netzsteuerung kann NONSTOP online als Simulationsumgebung nutzen + + + alle Standard-Auswertungsmöglichkeiten + + + Zeit-Weg-Diagramme, Reisezeiten und Darstellung der geschalteten Signalzeiten etc. + + + Protokollierung aller verkehrlichen Ereignisse in einer Datenbank + + + Export und Weiterverarbeitung der Daten in Excel möglich + + + alle Daten für Detailauswertungen und weitergehende Analysen verfügbar + + +

Offline-Optimierung von Grünen Wellen im Netz

+ + + Berechnung optimaler Signalzeiten im Netz für festgelegte Verkehrssituationen + + + Optimierung mit genetischen Algorithmen + + + freie Festlegung der Bewertungsfunktion + + + die Ergebnisse dienen als Grundlage für die Erstellung Grüner Wellen + + +



Verlauf einer Optimierung
x-Achse: Optimierungsgenerationen; y-Achse: Gütewert
unteres Band: Ergebnisse des Referenz-Signalprogrammes
dicke Linie: Mittelwert der optimierten Signalprogramme

GEVAS SOFTWARE
Systementwicklung und Verkehrsinformatik GmbH

Leuchtenbergring 20
D-81677 München

software@gevas.de
www.gevas.de

Tel. +49 -(0)89 -25 55 97 -0
Fax +49 -(0)89 -25 55 97 -66

strassenkreuz_statisch_zeitzeit.xml

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE welt PUBLIC "-//RiSc Tec//DTD WELT 1.0//DE"
               "http://traffspot.db-nico.de/DTDs/TraffSpot_1.0.dtd">

<welt version="1.0">

<!-- Spuren
<spur id="" laenge="" ausrichtung="" richtung=""
      x="" y="" z="0" maximaleGeschwindigkeit="">
  <nachfolger id="" richtung="" wahrscheinlichkeit="" />
</spur>
-->

<spur id="100" laenge="500" ausrichtung="HOR" richtung="RTOL"
      x="1020" y="1120" z="0" maximaleGeschwindigkeit="80">
  <nachfolger id="200" richtung="MITTE" wahrscheinlichkeit="1.0"/>
</spur>

<spur id="101" laenge="500" ausrichtung="HOR" richtung="LTOR"
      x="520" y="1080" z="0" maximaleGeschwindigkeit="50">
  <nachfolger id="130" richtung="MITTE" wahrscheinlichkeit="1.0" />
</spur>

<spur id="102" laenge="500" ausrichtung="HOR" richtung="LTOR"
      x="520" y="1060" z="0" maximaleGeschwindigkeit="50">
  <nachfolger id="120" richtung="MITTE" wahrscheinlichkeit="1.0" />
</spur>

<spur id="103" laenge="500" ausrichtung="HOR" richtung="LTOR"
      x="520" y="1040" z="0" maximaleGeschwindigkeit="50">
  <nachfolger id="110" richtung="MITTE" wahrscheinlichkeit="1.0" />
</spur>

<spur id="200" laenge="500" ausrichtung="HOR" richtung="RTOL"
      x="500" y="1100" z="0" maximaleGeschwindigkeit="80" />

<spur id="201" laenge="500" ausrichtung="HOR" richtung="LTOR"
      x="0" y="1060" z="0" maximaleGeschwindigkeit="80">
  <nachfolger id="101" richtung="LINKS" wahrscheinlichkeit="0.3" />
  <nachfolger id="102" richtung="MITTE" wahrscheinlichkeit="0.3" />
  <nachfolger id="103" richtung="RECHTS" wahrscheinlichkeit="0.3" />
</spur>

<spur id="110" laenge="500" ausrichtung="VER" richtung="DOWN"
      x="1060" y="1020" z="0" maximaleGeschwindigkeit="80">
  <nachfolger id="210" richtung="MITTE" wahrscheinlichkeit="1.0" />
</spur>

<spur id="111" laenge="500" ausrichtung="VER" richtung="UP"
      x="1060" y="520" z="0" maximaleGeschwindigkeit="50">
  <nachfolger id="100" richtung="MITTE" wahrscheinlichkeit="1.0" />
</spur>

<spur id="112" laenge="500" ausrichtung="VER" richtung="UP"
      x="1080" y="520" z="0" maximaleGeschwindigkeit="50">
  <nachfolger id="130" richtung="MITTE" wahrscheinlichkeit="1.0" />
</spur>
```

```
<spur id="113" laenge="500" ausrichtung="VER" richtung="UP"
      x="1100" y="520" z="0" maximaleGeschwindigkeit="50">
  <nachfolger id="120" richtung="MITTE" wahrscheinlichkeit="1.0" />
</spur>

<spur id="210" laenge="500" ausrichtung="VER" richtung="DOWN"
      x="1080" y="500" z="0" maximaleGeschwindigkeit="80" />

<spur id="211" laenge="500" ausrichtung="VER" richtung="UP"
      x="1080" y="0" z="0" maximaleGeschwindigkeit="80">
  <nachfolger id="111" richtung="LINKS" wahrscheinlichkeit="0.3" />
  <nachfolger id="112" richtung="MITTE" wahrscheinlichkeit="0.3" />
  <nachfolger id="113" richtung="RECHTS" wahrscheinlichkeit="0.3" />
</spur>

<spur id="120" laenge="500" ausrichtung="HOR" richtung="LTOR"
      x="1140" y="1040" z="0" maximaleGeschwindigkeit="80">
  <nachfolger id="220" richtung="MITTE" wahrscheinlichkeit="1.0" />
</spur>

<spur id="121" laenge="500" ausrichtung="HOR" richtung="RTOL"
      x="1640" y="1080" z="0" maximaleGeschwindigkeit="50">
  <nachfolger id="110" richtung="MITTE" wahrscheinlichkeit="1.0" />
</spur>

<spur id="122" laenge="500" ausrichtung="HOR" richtung="RTOL"
      x="1640" y="1100" z="0" maximaleGeschwindigkeit="50">
  <nachfolger id="100" richtung="MITTE" wahrscheinlichkeit="1.0" />
</spur>

<spur id="123" laenge="500" ausrichtung="HOR" richtung="RTOL"
      x="1640" y="1120" z="0" maximaleGeschwindigkeit="50">
  <nachfolger id="130" richtung="MITTE" wahrscheinlichkeit="1.0" />
</spur>

<spur id="220" laenge="500" ausrichtung="HOR" richtung="LTOR"
      x="1660" y="1060" z="0" maximaleGeschwindigkeit="80" />

<spur id="221" laenge="500" ausrichtung="HOR" richtung="RTOL"
      x="2160" y="1100" z="0" maximaleGeschwindigkeit="80">
  <nachfolger id="121" richtung="LINKS" wahrscheinlichkeit="0.3" />
  <nachfolger id="122" richtung="MITTE" wahrscheinlichkeit="0.3" />
  <nachfolger id="123" richtung="RECHTS" wahrscheinlichkeit="0.3" />
</spur>

<spur id="130" laenge="500" ausrichtung="VER" richtung="UP"
      x="1100" y="1140" z="0" maximaleGeschwindigkeit="80">
  <nachfolger id="230" richtung="MITTE" wahrscheinlichkeit="1.0" />
</spur>

<spur id="131" laenge="500" ausrichtung="VER" richtung="DOWN"
      x="1100" y="1640" z="0" maximaleGeschwindigkeit="50">
  <nachfolger id="120" richtung="MITTE" wahrscheinlichkeit="1.0" />
</spur>

<spur id="132" laenge="500" ausrichtung="VER" richtung="DOWN"
      x="1080" y="1640" z="0" maximaleGeschwindigkeit="50">
  <nachfolger id="110" richtung="MITTE" wahrscheinlichkeit="1.0" />
</spur>
```

```
<spur id="133" laenge="500" ausrichtung="VER" richtung="DOWN"
      x="1060" y="1640" z="0" maximaleGeschwindigkeit="50">
  <nachfolger id="100" richtung="MITTE" wahrscheinlichkeit="1.0" />
</spur>

<spur id="230" laenge="500" ausrichtung="VER" richtung="UP"
      x="1080" y="1660" z="0" maximaleGeschwindigkeit="80" />

<spur id="231" laenge="500" ausrichtung="VER" richtung="DOWN"
      x="1080" y="2160" z="0" maximaleGeschwindigkeit="80">
  <nachfolger id="131" richtung="LINKS" wahrscheinlichkeit="0.3" />
  <nachfolger id="132" richtung="MITTE" wahrscheinlichkeit="0.3" />
  <nachfolger id="133" richtung="RECHTS" wahrscheinlichkeit="0.3" />
</spur>

<!-- Kreuzungen
<kreuzung id="" ampelalgorithmus=""
          nordenLinks="" nordenMitte="" nordenRechts=""
          ostenLinks="" ostenMitte="" ostenRechts=""
          suedenLinks="" suedenMitte="" suedenRechts=""
          westenLinks="" westenMitte="" westenRechts="" />
-->

<kreuzung id="1" ampelalgorithmus="1"
          westenLinks="101" westenMitte="102" westenRechts="103"
          suedenLinks="111" suedenMitte="112" suedenRechts="113"
          ostenLinks="121" ostenMitte="122" ostenRechts="123"
          nordenLinks="131" nordenMitte="132" nordenRechts="133" />

<!-- Generatoren
<generator id="" spurID="" erstellzeit="" fahrzeugAnzahl="">
     <fahrzeug typ="" laenge="" lebensdauer="" rot="" gruen=""
               blau="" beschleunigungPositiv=""
               beschleunigungNegativ="" hoechstgeschwindigkeit="" />
     <fahrtenbucheintrag abbiegerichtung="" />
</generator>
-->

<generator id="2001" spurID="201" erstellzeit="5"
           fahrzeugAnzahl="500">
  <fahrzeug typ="PKW" laenge="4.1" lebensdauer="100"
            rot="0" gruen="128" blau="64"
            beschleunigungPositiv="6.2" beschleunigungNegativ="-7.5"
            hoechstgeschwindigkeit="180" />
</generator>

<generator id="2002" spurID="201" erstellzeit="6"
           fahrzeugAnzahl="400">
  <fahrzeug typ="BUS" laenge="8" lebensdauer="100"
            rot="255" gruen="128" blau="0"
            beschleunigungPositiv="4.7" beschleunigungNegativ="-6.0"
            hoechstgeschwindigkeit="130" />
</generator>
```

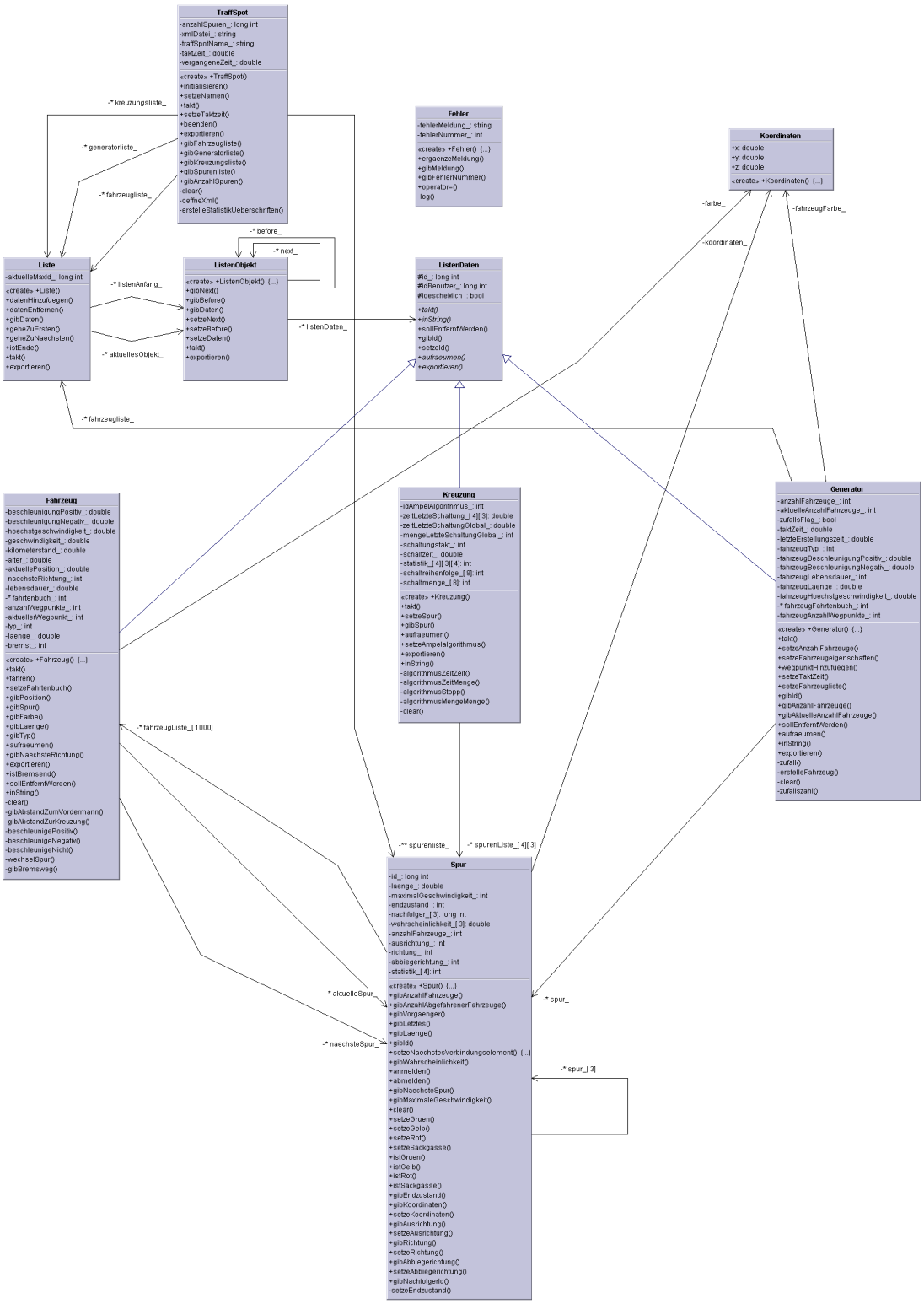
```
<generator id="2101" spurID="211" erstellzeit="5"
           fahrzeugAnzahl="500">
  <fahrzeug typ="PKW" laenge="4.7" lebensdauer="100"
            rot="0" gruen="0" blau="160"
            beschleunigungPositiv="6.5" beschleunigungNegativ="-7.6"
            hoechstgeschwindigkeit="200" />
</generator>

<generator id="2102" spurID="211" erstellzeit="6"
           fahrzeugAnzahl="400">
  <fahrzeug typ="LKW" laenge="7" lebensdauer="100"
            rot="192" gruen="192" blau="192"
            beschleunigungPositiv="4.1" beschleunigungNegativ="-8.0"
            hoechstgeschwindigkeit="90" />
</generator>

<generator id="2201" spurID="221" erstellzeit="5"
           fahrzeugAnzahl="500">
  <fahrzeug typ="PKW" laenge="4.5" lebensdauer="100"
            rot="255" gruen="0" blau="128"
            beschleunigungPositiv="6.3" beschleunigungNegativ="-7.1"
            hoechstgeschwindigkeit="160" />
</generator>

<generator id="2202" spurID="221" erstellzeit="6"
           fahrzeugAnzahl="400">
  <fahrzeug typ="BIKE" laenge="1.7" lebensdauer="100"
            rot="255" gruen="255" blau="0"
            beschleunigungPositiv="10.2"
            beschleunigungNegativ="-9.9"
            hoechstgeschwindigkeit="250" />
</generator>

<generator id="2301" spurID="231" erstellzeit="5"
           fahrzeugAnzahl="500">
  <fahrzeug typ="PKW" laenge="3.8" lebensdauer="100"
            rot="0" gruen="0" blau="128"
            beschleunigungPositiv="5.9" beschleunigungNegativ="-6.9"
            hoechstgeschwindigkeit="160" />
</generator>

<generator id="2302" spurID="231" erstellzeit="6"
           fahrzeugAnzahl="400">
  <fahrzeug typ="LKW" laenge="6" lebensdauer="100"
            rot="64" gruen="64" blau="0"
            beschleunigungPositiv="4.2" beschleunigungNegativ="-8.5"
            hoechstgeschwindigkeit="90" />
</generator>

</welt>
```

TraffSpot_1.0.dtd

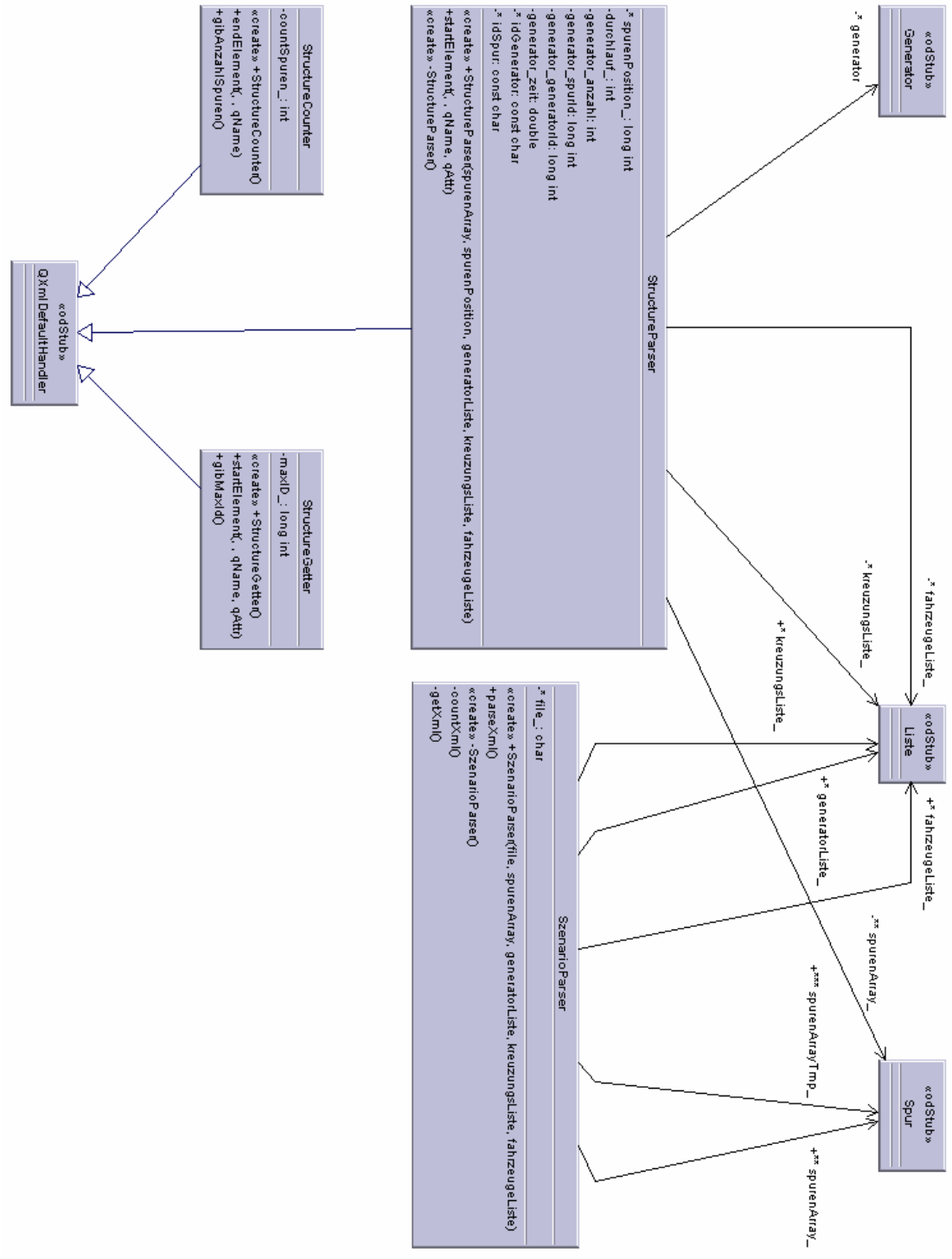
```
<?xml version="1.0" encoding="iso-8859-1" ?>

<!--  Autor: Andreas Richter
   Version: 1.0

   DTD zum Ueberpruefen von XML-Dateien, die Strassennetze fuer
   TraffSpot definieren sollen. Eine von TraffSpot zu verarbeitende
   XML-Datei MUSS gegenueber dieser DTD validierbar sein!
      (Dazu kann "Validate by File Upload" auf
       http://validator.w3.org/ benutzt werden)
-->

<!--Welt
   Fiktives Element, welches alle Bestandteile eines Szenarions
   beinhalten muss. Dieses Element ist nur der Uebersicht wegen
   in der DTD verankert. -->
<!ELEMENT welt (spur*, kreuzung*, generator*) >
<!ATTLIST welt version CDATA "1.0" >

<!--Spur
   Jede Spur schliesst drei oder weniger Nachfolger-Elemente ein -->
<!ELEMENT spur (nachfolger*) >
<!ATTLIST spur id CDATA #REQUIRED>
<!ATTLIST spur laenge CDATA #REQUIRED>
<!ATTLIST spur ausrichtung (HOR|VER) #REQUIRED>
<!ATTLIST spur richtung (LTOR|RTOL|UP|DOWN) #REQUIRED>
<!ATTLIST spur x CDATA #REQUIRED>
<!ATTLIST spur y CDATA #REQUIRED>
<!ATTLIST spur z CDATA #REQUIRED>
<!ATTLIST spur maximaleGeschwindigkeit CDATA #REQUIRED>

<!ELEMENT nachfolger EMPTY >
<!ATTLIST nachfolger id CDATA #REQUIRED>
<!ATTLIST nachfolger richtung (LINKS|MITTE|RECHTS) #REQUIRED>
<!ATTLIST nachfolger wahrscheinlichkeit CDATA #REQUIRED>

<!--Kreuzung
   Im Attribut 'ampelalgorithmus' wird ueber einen Zahlencode
   der gewuenschte Algorithmus zugewiesen. Alle anderen Attribute
   (ausser 'id', das ist die KreuzungsID) enthalten die IDs der
   beteiligten Spuren. -->
<!ELEMENT kreuzung EMPTY >
<!ATTLIST kreuzung westenLinks CDATA #IMPLIED>
<!ATTLIST kreuzung westenMitte CDATA #IMPLIED>
<!ATTLIST kreuzung westenRechts CDATA #IMPLIED>
<!ATTLIST kreuzung suedenLinks CDATA #IMPLIED>
<!ATTLIST kreuzung suedenMitte CDATA #IMPLIED>
<!ATTLIST kreuzung suedenRechts CDATA #IMPLIED>
<!ATTLIST kreuzung ostenLinks CDATA #IMPLIED>
<!ATTLIST kreuzung ostenMitte CDATA #IMPLIED>
<!ATTLIST kreuzung ostenRechts CDATA #IMPLIED>
<!ATTLIST kreuzung nordenLinks CDATA #IMPLIED>
<!ATTLIST kreuzung nordenMitte CDATA #IMPLIED>
<!ATTLIST kreuzung nordenRechts CDATA #IMPLIED>
<!ATTLIST kreuzung ampelalgorithmus CDATA #REQUIRED>
<!ATTLIST kreuzung id CDATA #REQUIRED>
```

```
<!--Generator
   Enthaelt der Generator ein Fahrzeug-Element, so werden aus-
   schliesslich Fahrzeuge mit angegebenen Parametern kreiert.
   Ist kein Fahrzeug-Element eingetragen, werden zufaellige Fahr-
   zeuge generiert. In beiden Faellen koennenn Fahrtenbuchein-
   traege vorgenommen werden. Je ein Eintrag gilt fuer eine
   Kreuzung. -->
<!ELEMENT generator (fahrzeug?, fahrtenbucheintrag*) >
<!ATTLIST generator fahrzeugAnzahl CDATA #REQUIRED>
<!ATTLIST generator erstellzeit CDATA #REQUIRED>
<!ATTLIST generator spurID CDATA #REQUIRED>
<!ATTLIST generator id CDATA #REQUIRED>

<!ELEMENT fahrzeug EMPTY >
<!ATTLIST fahrzeug typ (PKW|BUS|LKW|BIKE) #REQUIRED>
<!ATTLIST fahrzeug laenge CDATA #REQUIRED>
<!ATTLIST fahrzeug lebensdauer CDATA #REQUIRED>
<!ATTLIST fahrzeug beschleunigungPositiv CDATA #REQUIRED>
<!ATTLIST fahrzeug beschleunigungNegativ CDATA #REQUIRED>
<!ATTLIST fahrzeug hoechstgeschwindigkeit CDATA #REQUIRED>
<!ATTLIST fahrzeug rot CDATA #REQUIRED>
<!ATTLIST fahrzeug gruen CDATA #REQUIRED>
<!ATTLIST fahrzeug blau CDATA #REQUIRED>

<!ELEMENT fahrtenbucheintrag EMPTY >
<!ATTLIST fahrtenbucheintrag abbiegerichtung (LINKS|MITTE|RECHTS)
#REQUIRED>
```

Klassendiagramm Simulation



Klassendiagramm XML-Parser



Arbeitsplan

Oktober 2004 bis Juni 2005

07.12.2004 - Fertigstellung des theoretischen Simulationskonzeptes
20.12.2004 - Recherche im Themenfeld der Verkehrsflusoptimierung
- Framework (Grundgerüst) in C++

23.01.2005 - Implementierung der Graphen (Straßen)
13.02.2005 - Implementierung der Fahrzeuge
7. KW - Treffen in Braunschweig
06.03.2005 - Kombination beider Teile
20.03.2005 - Fertigstellung der Simulation mit rudimentären Ampelalgorithmus

- statischer Ampelalgorithmus
17.04.2005 - Zählungen für Statistiken zum Vergleich verschiedener Algorithmen

(2 Wo) - 2 verschiedene Ansätze für Ampelalgorithmen

(2 Wo) - Daten sammeln und auswerten

15.05.2005 - Abschluss praktischer Teil
- Dokumentation überarbeiten
    - Dokumentation für Code kompilieren
    - Statistiken auswerten
    - Schriftstücke überarbeiten

01.06.2005 - Abschluss des Projektes

Legende:
Meilensteine

Programm inklusive Dokumentation und Beispieldateien