FH - Studiengang für Informationstechnik und System-Management Salzburg

ITS

Übungen in Spezielle Softwaretechnologien

Protokoll

Gegenstand der Übung gemäß Anleitung:

Verkehrssimulation

Version: 1

Datum der Übung: 01.03.2017

Datum der Abgabe: 13.07.2017

Autoren: Christopher Wieland, Martin Wieser, Stephanie Kaschnitz, Hannes Kleiner, Andreas Lippmann

Inhaltsverzeichnis

Tabellenverzeichnis

1 Aufgabenstellung

Die Aufgabenstellung beinhaltet eine mikroskopische Simulation von Fahrzeuge, sowohl PKW als auch LKW, und Ampelanlagen. Ein weiterer wichtiger Punkt stellt das zusammenhängende Verkehrs-/Straßennetz dar. Weiters ist es möglich ungeregelte und geregelte Kreuzungen darzustellen. Ein wichtiger Faktor ist die Parametrisierbarkeit der Verkehrsteilnehmer und der Lichtsteueranlage.

Die Lichtsteueranlage wird über einen eigenen Prozess geregelt. Die verfügt die Applikation über eine grafische Darstellung sowie über eine einfache Benutzerschnittstelle.

In weiterer Folge sollen nun die Simulationen der verschiedenen Gruppen untereinander kommunizieren können. Das heißt, dass Fahrzeuge bei einer Straße einer Simulation ausfahren und dann in der anderen Simulation über eine Straße wieder einfahren können. Dem Benutzer wird ermöglicht beliebige Hindernisse auf der Straße per Mausklick ein-/auszuschalten.

2 Gesamtübersicht und Idee

Für die Umsetzung der Aufgabenstellung wurden die Aufgaben in sechs große Bereiche gespalten: GUI, Verkehrsnetz, Verkehrsteilnehmer, Verkehrsregeln, Ampelsteuerung und Rabbit MQ. Zu beachten ist der Punkt, dass die Ampelsteuerung extern zur Verfügung gestellt werden musste. Um eine Kommunikation zwischen Ampelsteuerung und Verkehrssimulation zu erreichen, wurde mit Windows Communication Foundation (WCF) gearbeitet. Hierbei ist die Ampelsteuerung der Server, welcher den Dienst der Ampel zur Verfügung stellt und die Verkehrssimulation der Client, welcher den Status der Ampeln abgefragt.

2.1 GUI

In der hier vorliegenden Architektur wurde die GUI als eigene Komponente entwickelt. Grund hierfür war eine entkoppelte Benutzeroberfläche von der Simulationslogik anzubieten. Die GUI stellt lediglich die einzelnen Elemente der Simulation dar und gibt den Input des Benutzers an die darunterliegenden Logikkomponenten weiter.

2.1.1 Klasse „Verkehrsteilnehmer“

Die Klasse Verkehrsteilnehmer kümmert sich um die korrekte Darstellung aller Autos/LKWs im Straßennetz. Mit Hilfe der bereitgestellten Funktion können Verkehrsteilnehmer hinzugefügt, gelöscht und ihre Position updatet werden.

2.1.2 Klasse „Ampel“

Die Klasse Ampel setzt die Ampeln an geregelte Kreuzungen und bietet Funktionen an mit denen der Status der Ampeln verändert werden kann (Rot, Grün, Gelb, Blinkend)

2.1.3 Klasse „Obstacle“

Diese Klasse übernimmt das Einzeichnen eines Hindernisses. Sie bietet dem Benutzer die Funktion über einen Mausklick ein Hindernis einzufügen

2.2 Verkehrsnetz

Das Verkehrsnetz ist eine Kernkomponente in der die Straßeneigenschaften für die Fahrzeuge ausgewertet und weitergegeben werden. In den Unterpunkten werden die Hauptklassen des Verkehrsnetzes beschrieben.

2.2.1 Klasse „EnvironmentHandler“

Die Klasse EnvironmentHandler dient zum Aufbau und der Aufbereitung der Informationen des Spielfeldes.

Wichtige Informationen dabei sind:

- Ampeln
- Regeln
- Hindernisse
- Straßeneigenschaften

Durch die Konfiguration der geregelten Kreuzungen im Config-file (env_config.json) wird das Spielfeld automatisch erstellt. Die restlichen Elemente wie Straßen, ungeregelte Kreuzungen oder Gras resultieren aus dieser Konfiguration.

Nach dem Aufbau können die Informationen für die Fahrzeuge abgefragt werden.

2.2.2 Klasse „ObstacleHandler“

Die Klasse ObstacleHandler wird dazu verwendet die hinzugefügten Hindernisse zu verwalten. Die Hindernisse können von der GUI über das Interface I_ENV_GUI eingetragen werden.

2.2.3 Klasse „Env_Ampelhandler“

Die Klasse Env_Ampelhandler regelt die Kreuzungen und wie diese zusammenspielen. In dieser Klasse werden die Ampeln initialisiert und je nach Konfiguration als 3er oder 4rer Kreuzungen verknüpft. Danach können diese mit einer ID abgefragt werden.

2.2.4 Klasse „Streetelem“

Die Klasse Streetelem stellt die „tiles“ des Spielfeldes dar. Die Straßenelemente beinhalten die benötigten Informationen vom Untergrunde auf dem das Auto fährt.

2.2.5 Klasse „StreetInfo“

Die Klasse StreetInfo wird dazu verwendet die informationen der Straße für die Autos aufzubereiten und ist somit ein wichtiger bestandteil der Simulation. Mit dieser Klasse werden die nötigen infos bzg Straßentyp, mögliche Hindernisse und der Ampeln weitergegeben. Die Streetinfo beinhaltet NICHT die Auswertung von Positionen anderer Fahrzeuge.

2.3 Verkehrsregeln

Die Verkehrsregeln beinhalten allgemeine Regeln, welche bei ungeregelten und geregelten Kreuzungen eingehalten werden müssen.

2.3.1 Klasse „Schilder“

Diese Klasse wird benötigt um eine gewisse Anzahl und die verschiedenen Typen der Schilder zu generieren. Zurückgegeben wird eine Liste der Schilder.

2.3.2 Klasse „AllgemeineVerkehrsregeln“

Diese Klasse ist für die allgemeinen Verkehrsregeln zuständig. Hierfür liefert sie eine Liste mit allen Verkehrsregeln zurück.

2.4 Verkehrsteilnehmer

Die Komponente Verkehrsteilnehmer umfasst die Verwaltung von PKWs und LKWs sowie deren Verhalten und Positionen.

2.4.1 Klasse „TrafficObject“

Ein einfache Klasse die einen einzelnen Verkehrsteilnehmer und dessen Eigenschaften repräsentiert.

2.4.2 Klasse „TrafficHandler“

Diese Klasse verwaltet alle Verkehrsteilnehmer, fügt neue hinzu und aktualisiert in regelmäßigen Abständen den Standort aller Verkehrsteilnehmer.

2.5 Ampelsteuerung

Die Ampelsteuerungskomponente hat die Aufgabe, Ampeln nach Belieben zu erstellen, aktiv/inaktiv zu schalten, Statusphasen manuell anzupassen und automatisch umzuschalten, wenn die Phasen jeweils vorüber sind. Realisiert wurde diese Komponente mit 2 Klassen:

2.5.1 Klasse „Ampeln“

Diese Klasse symbolisiert eine einzelne Ampel. Mithilfe dieser Klasse ist das Erstellen von Ampeln ermöglicht worden. Funktionen und Variablen zu dieser Klasse sind in Abschnitt 3.5.1 ersichtlich.

2.5.2 Klasse „Ampelsteuerung“

Die Klasse Ampelsteuerung ist für das Verhalten der Ampeln verantwortlich. Hier wird ebenso der WCF Server gestartet. In Kapitel 3.5.2 werden die verwendeten Funktionen und Variablen beschrieben.

2.6 RabbitMQ

RabbitMQ wird verwendet um zwischen den Gruppen die ausfahrenden Autos weiterzuleiten.

2.6.1 Klasse „RabbitMQHandler“

Der RabbitMQHandler wird verwendet um Transaktionen zu senden und zu empfangen.

2.6.2 Klasse „RemoteTransaction“

Hierbei handelt es sich um eine Transaktion, welche sich aus den Attributen CarId, GroupId, CarType, Speed, Timestamp und Errorcode zusammensetzen.

3 Dokumentation der Funktionalität der Verkehrssimulation

In diesem Sektor werden alle Funktionen aller Komponenten erklärt. Ebenso wird darauf eingegangen, wie alle Komponenten miteinander Funktionieren.

3.1 GUI

Die Komponente „GUI“ besteht aus den Klassen: Verkehrsteilnehmer, Ampel und Obstacle (siehe Kapitel 2.1), sowie den Interfaces: IVerkehrsnetz und IObject. In diesem Abschnitt wird genauer auf diese eingegangen.

3.1.1 Funktionen der Klasse Verkehrsteilnehmer

Die folgende Liste gibt die Funktionen der Klasse Verkehrsteilnehmer wieder und dem zugehörigen „ObjectHandler“ wieder.

Funktion	Erklärung
Verkehrsteilnehmer()	Konstruktor: Erzeugt entweder ein Auto oder einen LKW
update(int x, int y, int type, int direction)	Aktualisiert die Position/Typ/Ausrichtung der Verkehrsteilnehmers
getShape()	Gibt die Form (Rechteck, Quadrat) wieder
setColor(Color c)	Setzt die Farbe (Differenzierung zwischen den Gruppen)
getID()	Gibt die Id wieder
enter(obj sender, arg e)	Erkennt wenn der Verkehrsteilnehmer angeklickt wurde

leave(obj sender, arg e)	Erkennt wenn der Mausklick beendet wird
conaoleMsg(obj sender, arg e)	Schreibt eine Log-Massage mit der ID und Position des angeklickten Elements

Tabelle 1: Funktionen der GUI Klasse Verkehrsteilnehmer

Die Attribute hierzu sind:

```
int id;
Shape shp;
int Xpos;
int Ypos;
int direction;
```

3.1.2 Funktionen der Klasse ObjectHandler

Um die Darstellung der Verkehrsteilnehmer im Verkehrsnetz zu verwalten, muss ein Objekt Handler hinzugefügt werden. Dieser kümmert sich um alle Fahrzeuge/Hindernisse, die sich zurzeit auf der Map (Canvasoberfläche) befinden.

- `public bool addCarObject(int x, int y, int id);`

  Fügt ein Auto an den X/Y Koordinaten ein und liefert im Erfolgsfall ein „true“ zurück.

- `public bool addLKWObject(int x, int y, int id, int dir);`

  Fügt ein LKW an den X/Y Koordinaten ein.

- `public bool updateCarwithID(int x, int y, int id, int dir);`

  Aktualisiert die Position und Richtung eines Autos mit einer bestimmten Id.

- `public bool removeObject();`

  Entfernt das Objekt mit der zugehörigen ID von der Map.

- `public bool addObstacle(obj sender, Arg e);`

  Fügt ein Hindernis an dem angeklickten Punkt im Canvas ein.

3.1.3 Funktionen der Klasse Ampel

Hier geht es lediglich um den Farbwechsel der Ampeln und das Einfügen an den geregelten Kreuzungen des Verkehrsnetzes. Die folgende Liste gibt die Funktionen der Klasse Ampel und dem zugehörigen Handler „AmpelHandler“ wieder.

Funktion	Erklärung
Ampel(double x, double y, int dir, int id)	Konstruktor: Erstellt eine Ampel aus 4 Elementen mit bestimmter Position, ID und Ausrichtung
getObjId() / setObjId(int id)	Liefert die Id der Ampel zurück / Setzt die Id neu
getXPos() / setXPos(int x)	Liefert die x-Position der Ampel zurück / Setzt die x-Position neu
getYPos() / setYPos(int y)	Analog zur x-Postion
getGreenCircle() / setGreenCircle(Shape)	Liefert den grünen Kreis der Ampel / Setzt die ihn neu
getYellowCircle() / setYellowCircle(Shape)	Liefert den gelben Kreis der Ampel / Setzt die ihn neu
getRedCircle() / setRedCircle(Shape)	Liefert den roten Kreis der Ampel / Setzt die ihn neu
getObjShape / setObjectShape(shape)	Liefert den Rahmen der Ampel / Setzt die ihn neu

Tabelle 2: Funktionen der GUI Klasse Ampel

Die Attribute hierzu sind:

```
int status;
Shape redCircle;
Shape yellowCircle;
Shape greenCircle;
```

```
Shape shape;
int id;
bool green, yellow, red;
int xPos;
int yPos;
int direction;
```

3.1.4 Funktionen der Klasse AmpelHandler

Um die Darstellung der Ampeln im Verkehrsnetz zu verwalten, muss ein Ampel Handler hinzugefügt werden. Dieser kümmert sich um die korrekte Darstellung der Ampeln und des jeweiligen Status.

- `publi bool addTrafficLight(int x, int y, int id, int dir);`

  Fügt eine Ampel an den X/Y Koordinaten mit bestimmter Ausrichtung ein und liefert im Erfolgsfall ein „true“ zurück.

- `publi void setGreen(int id);`

  Setzt die Ampel mit der übergegbenen Id auf grün.

- `publi void setYellow(int id);`

  Setzt die Ampel mit der übergegbenen Id auf gelb.

- `publi void setRed(int id);`

  Setzt die Ampel mit der übergegbenen Id auf rot.

- `publi void setNext(int id);`

  Prüft welche Farbe die Ampel mit der Id anzeigt und schaltet auf die nächst-logische Farbe.

- `publi void setStatus(int id, int status);`

  Setzt die Farbe der Ampel nach einem übergebenen Status.

3.1.5 Funktionen der Klasse Obstacle

Die Klasse „Obstacle“ erzeugt ein Hindernis. Da, für die wenigen Funktionen der Klasse ein separater Handler keinen Sinn machen würde, wurden die Funktionen im „ObjectHandler“ implementiert

Funktion	Erklärung
Obstacle(int id, int x, int y)	Konstruktor: Erzeugt ein neues Hindernis mit einer Id und Position im Canvas
getShape()	Gibt die Form des Hindernisses wieder

Tabelle 3: Funktionen der GUI Klasse Obstacle

Die Attribute hierzu sind:

```
int id;
Shape shp;
int Xpos;
int Ypos;
```

3.1.6 Funktionen der Interfaces IObject

Dieses Interface dient als Schnittstelle zur Komponente: Verkehrsteilnehmer. Die folgende Tabelle zeigt die Funktionen des Interfaces:

Funktion	Erklärung
addCarObject(int x, int y, int id)	Fügt ein Auto der GUI hinzu
addLKWObject(int x, int y, int id, int dir)	Fügt ein LKW der GUI hinzu
updateCarwithID(int x, int y, int id, int dir);	Aktualisiert die Position in der GUI
removeObject(int id)	Löscht ein Verkehrsteilnehmer

Tabelle 4: Funktionen des GUI Interface IObject

3.1.7 Funktionen der Interfaces IVerkehrsnetz

Dieses Interface dient als Schnittstelle zur Komponente: Verkehrsnetz. Die folgende Tabelle zeigt die Funktionen des Interfaces:

Funktion	Erklärung
getGreenLight() / setGreenLight(Shape)	Liefert den grünen Kreis der Ampel / Setzt die ihn neu
getGreenLight() / setGreenLight(Shape)	Liefert den gelben Kreis der Ampel / Setzt die ihn neu
getRedLight() / setRedLight(Shape)	Liefert den roten Kreis der Ampel / Setzt die ihn neu
getXPos() / setXPos(int x), **getYPos() / setYPos(int y),**	Liefert die Position der Ampel zurück / Setzt die Position neu
getObjId() / setObjId(int id)	Liefert die Id der Ampel zurück / Setzt die Id neu

Tabelle 5: Funktionen des GUI Interface IVerkehrsnetz

3.2 Verkehrsnetz

Das Verkehrsnetz das in Kapitel 2.2 beschrieben ist besteht aus einigen Klassen. Diese werden hier kurz näher beschrieben. Zudem werden noch die Interfaces für die GUI und die Verkehrsteilnehmer sowie weitere Hilfsklassen kurz beschrieben.

3.2.1 Methoden der Klasse EnvironmentHandler

- `public EnvironmentHandler(Canvas mycanvas, ref GUI.AmpelHandler _ah, ref IAmpelService _trafficlight)`
  Konstruktor für den Environmenthandler

- `private void LoadJson()`
  Lädt das JSON Object aus dem Config-file.

- `private int calcAmpelCnt()`
  Berechnet die Anzal der benötigten Ampeln.

- `private void LoadEnvironment()`
  Aufbau des Spielfeldes.

- `private void addThreeGuiLights(int nopath,int xpos,int ypos)`
  Aufbau von 3er Kreuzungen.

- `private void rotateElement(int cnt, int xpos, int ypos)`
  Ausrichtung von 3er Kreuzungen.

- `public void printEntryPoints()`
  Ausgabe der Entrypoints in die Konsole.

- `private void addEntryPoints(int xpos, int ypos)`
  Hinzufügen der Entrypoints von 4rer Kreuzungen.

- `private void addEntryPoints(int xpos, int ypos, int nopath)`
  Hinzufügen der Entrypoints von 3rer Kreuzungen.

- `private void addSolution(int xpos, int ypos)`
  Aufbau der Resultierenden Straßen aus einer 4er Kreuzung.

- `private void addSolution(int xpos, int ypos, int nopath)`
  Aufbau der Resultierenden Straßen aus einer 3er Kreuzung.

- `public Streetelem addObject(int x, int y, int type)`
  Erstellen des Straßenelementes “Streeetelem” zum hinzufügen zur Liste.

- `public Streetelem addObject(int x, int y, int type, int off)`
  Erstellen des Straßenelementes “Streeetelem” zum hinzufügen zur Liste.

- `public void UpdateGUIAmpeln()`
  Updaten der Ampeln auf der GUI und im Ampelhandler

- `public bool isObstacleInMyWay(int x, int y)`
  Überprüft of bein Hindernis auf dieser Position ist

- `public EnvElement.StreetType getStreetType(int x, int y)`
  Gibt den Straßentyp auf der entsprechenden Position zurück

- `public Streetelem getStreetElement(int x, int y)`
  Gibt das Straßenelement auf der entsprechenden Position zurück.

- `public int getAmpelID(int x, int y)`
  Gibt die AmpelID auf der entsprechenden Position zurück

- `public List<EntryPoint> getEnvironmentEntries()`
  Gibt die Liste mit den Entrypoitns zurück

- `public void addObstacle(int startx, int starty, int endx, int endy)`
  Fügt ein Obstacle zur Liste vom Environmenthandler hinzu.

- `public bool isOutside(int x, int y)`
  Gibt zurück ob sich das fahrzeug ausserhalb des Spielfeldes befindet.

- `public int getNeededEnvironmentRules(int x, int y)`
  Gibt den Straßentyp zurück.

- `public void setAmpelanlage(bool val)`
  Schaltet die Ampelanlagen ein/Aus. Hier wird auf die Ampelsteuerung zugegriffen.

- `public StreetInfo getNeededStreetRules(int x, int y)`
  Holt die Straßeninformation für die entsprechende Position

- `public List<Obstacle> getObstacles()`
  Gibt die Liste der Hindernisse zurück.

- `private void getEnvType(object sender, MouseEventArgs e)`
  Gibt den Typ der “Tile” in die Konsole aus.

3.2.2 Methoden der Klasse ObstacleHandler

- `public ObstacleHandler()`
  Konstruktor für den ObstacleHandler.

- `public List<Obstacle> getObstacles()`
  Rückgabe der Liste der Hindernisse.

- `public bool checkObstacles(int x, int y)`
  Fragt ab ob die Position ein Obstacle betrifft.

- `public bool inArea(int x, int y, Obstacle obs)`

Überprüft kollision der Koordinaten und des Obstacles.

- `public void addObstacle(Obstacle obs)`
  Fügt ein Hindernis zur liste hinzu.

3.2.3 Methoden der Klasse Env_Ampelhandler

- `public Env_Ampelhandler(int ampelcnt, JObject obj)`
  Konstruktor des Env_Ampelhandlers.

- `public Kreuzung getKreuzung(int id)`
  Gibt die Kreuzung mit gewünschter id zurück.

- `internal void setOffline(bool val)`
  Setzt die Ampeln offline.

- `internal void updateKreuzungen()`
  Updated den Status der Kreuzungen

- `internal void printStatus()`
  Gibt den Status in die Konsole aus.

- `internal void updateAmpel(int ampelid, int mycase)`
  Updatet eine einzelne Ampel, unabhängig von der Kreuzung

Zudem warden mit dem Ampelhandler die Kreuzungen implementiert die die Status der 3 bzw 4 Ampeln beinhalten.

3.2.4 Methoden der Klasse Streetelem

- `public Streetelem(int _x, int _y, int _dir, int _type, int ampelid)`
  Konstruktor des Straßenelementes.

- `public Image getImage()`
  Gibt das Image des Straßenelementes zurück.

- `public void enter(object sender, EventArgs e)`
  MouseEnter function für die Visuelle darstellung. Opacity wird verändert

- `public void leave(object sender, EventArgs e)`
  MouseEnter function für die Visuelle darstellung. Opacity wird verändert.

- `public int getAmpelID()`
  Gibt die Ampelid zurück, zum verknüpfen mit der Kreuzung

- `public void updateType(StreetType type)`
  Updatet den Straßentyp. (Bsp bei der resultierenden Straßenlage beim erstellen des Spielfeldes)

- `private void initImgType()`
  Initialisieren des Images des Elementes.

- `public StreetType getStreetType()`
  Rückgabe des Straßentypes.

- `public void printStreetType(object sender, EventArgs e)`
  Ausgabe des Straßentypes in die Konsole.

- `public int getRotation()`
  Rückgabe der Rotation für die 3er Kreuzung oder den Straßen.

In der Klasse Streetelem gibt es zudem noch weitere experimentelle Methoden die für ein dynamisches verändern der Straßenlage gedacht sind.

3.2.5 Attribute der Klasse StreetInfo

Die Klasse StreetInfo ist die Straßeninformation für die Autos die periodisch abgerufen wird. Diese Klasse beinhaltet Getter/Setter für die folgenden Attribute:

```
int type                  - Straßentyp
int layout                - Ausrichtung
int ampelstatusUp         - Status der Ampel -> Norden
int ampelstatusDown       - Status der Ampel -> Süden
int ampelstatusLeft       - Status der Ampel -> Osten
int ampelstatusRight      - Status der Ampel -> Westen
double steigungHorizontal - Not Implemented
double steigungVertical   - Not Implemented
```

3.2.6 Methoden des Interfaces I_ENV_GUI

Angebotene Schnittstelle für die GUI

Funktion	Erklärung
void addObstacle(int startx, int starty, int endx, int endy)	Fügt über die GUI ein obstacle hinzu
void setAmpelanlage(bool val)	Schaltet über die GUI die ampelanlagen ein/aus

Funktionen des Verkehrsnetz Interfaces I_ENV_GUI

3.2.7 Methoden des Interfaces I_ENV_VKTeilnehmer

Angebotene Schnittstelle für die Verkehrsteilnehmer

Funktion	Erklärung
StreetInfo getNeededStreetRules(int x, int y)	Gibt die Straßeninformation zurück
List<Obstacle> getObstacles()	Gibt die Obstacle Liste zurück
List<EntryPoint> getEnvironmentEntries()	Gibt die Entrypoints zurück (für das spawnen der Farhzeuge)
int getNeededEnvironmentRules(int x, int y)	Gibt Typen der Straße zurück
bool isObstacleInMyWay(int x, int y)	gibt zurück ob sich ein Hindernis in diesem bereich befindet
bool isOutside(int x, int y)	Gibt zurück ob sich ein Auto mit der übergebenen Position aus dem Spielfeld bewegt.

Funktionen des Verkehrsnetz Interfaces I_ENV_VKTeilnehmer

3.3 Verkehrsregeln

Die Verkehrsregeln setzen sich aus den Klassen „Schilder“ und „AllgemeineVerkehrsregeln“ zusammen. Außerdem bieten sie die Schnittstelle „IVerkehrsregeln“ an.

3.3.1 Funktionen der Klasse „AllgemeineVerkehrsregeln“

Diese Klasse liefert eine Liste mit allen Verkehrsregeln zurück. Im Anschluss wird auf jede Funktion dieser Klasse erläutert.

- `public List<int> getAllgemeineVerkehrsregeln()`

  Diese Funktion füllt zuerst die Liste mit der Hilfsfunktion „fillList“. Danach gibt sie diese Liste zurück.

- `private List<int> fillList()`

  Diese Funktion füllt die Liste mit den allgemeinen Verkehrsregeln, welche Integer Werte sind. Zuvor wurde definiert welcher Integer-Wert für welche Regel gilt.

3.3.2 Funktionen der Klasse „Schilder“

Diese Klasse erstellt Schilder und gibt diese in einer Liste zurück. Nachfolgend werden die benötigten Funktionen erläutert.

- `public List<Schilder> createSchilder(int _anzahl, int _type)`

Diese Liste erstellt je nach mitgegebenen „_anzahl“ Parameter die Anzahl der Schilder des Typs „_type“ und gibt diese zurück.

3.3.3 Funktionen des Interfaces „IVerkehrsregeln“

Die Verkehrsregeln bieten dem Verkehrsnetz die Schnittstelle IVerkehrsregeln an.

Methode	Kurzbeschreibung
createSchilder	Liefert eine Liste an Schildern zurück, je nach mitgelieferter Anzahl und Typ der Schilder.
getAllgmeineVerkehrsregeln	Liefert eine Liste mit den Allgemeinen Verkehrsregeln zurück.

Tabelle 6: Funktionen der Verkehrsregeln

3.4 Verkehrsteilnehmer

Die Komponente „Verkehrsteilnehmer“ bietet folgende Funktionen zur Verwaltung der Verkehrteilnehmer an:

3.4.1 Funktionen von „TrafficHandler“

- `private TrafficHandler(ref ITeilnehmer _eb, ref IObject _oh, ref RabbitMQ.RabbitMQHandler _mqhandler)`

  Privater Konstruktor nach Singleton Design-Pattern. Die Übergabeparameter sind Interfaces auf andere benötigte Komponenten.

- `public static TrafficHandler getInstance(ref ITeilnehmer _eb, ref IObject _oh, ref RabbitMQ.RabbitMQHandler _mqhandler)`

  GetInstance-Funktion nach Singleton Design-Pattern. Die Übergabeparameter sind Interfaces auf andere benötigte Komponenten.

- `public static TrafficHandler getInstance()`

  GetInstance-Funktion nach Singleton Design-Pattern. Nur zum Aufruf für schon instanziierte Klasse.

- `public void updateAll()`
  Dies ist die Funktion zum regelmäßigen Abarbeiten der gesamten Hintergrunglogik. Die umfasst folgende Schritte:
  - Erfassen auf welche Position sich die Verkehrsteilnehmer als nächstes bewegen würden
  - Erfassen ob dies möglich ist (dies kann z.B. nicht möglich sein durch andere Verkehrsteilnehmer im Weg, Ampeln oder recht-vor-links-Vorfahrtsregeln)
  - Erkennen von Verkehrsteilnehmer, die in einem „Deadlock“ feststecken (z. B. 4-armige Kreuzung, bei der an allen armen ein Geradeausfaher ankommt) und Verteilen von Sonderfahrprivilegien an einzelne Verkehrsteilnehmer um einen solchen „Deadlock“ aufzulösen.

- Aufspüren von Fahrzeugen, die sich sehr lange (100 Updates) nicht bewegt haben (vermutlich durch Unfall) und Entfernen von diesen.
- Aktualisierung der Position der Verkehrsteilnehmer, die fahren können.
- Entfernen von Verkehrsteilnehmern (aus der Logik) die die Simulation verlassen
- Versenden von Verkehrsteilnehmern, die die Simulation verlassen, an andere Gruppe:
  - 10%-Change, dass versendet wird
  - verlässt nach Norden -> an Gruppe 1
  - verlässt nach Osten -> an Gruppe 2
  - verlässt nach Süden -> an Gruppe 4
- Hinzufügen von Verkehrsteilnehmern wenn die tatsächliche Anzahl an Verkehrsteilnehmer die gewünschte Anzahl unterschreit. Die Wahrscheinlichkeit, dass ein neues Fahrzeug erzeugt wird, entspricht folgender Formel:

  $p = 1 - (\frac{\#Verkehrteilnehmer(tatsächlich)}{\#Verkehrteilnehmer(gewünscht)})^2$
- Empfangen von Verkehrsteilnehmern von anderen Gruppen

- `public void createNewVerkehrsteilnehmer(int x, int y, int speed, int typ, int direction, int nextDirection)`

  Hierbei handelt es sich um eine Funktion zum Hinzufügen eines neuen Verkehrsteilnehmers. Die Parameter x und y geben die Position des Verkehrsteilnehmers an. Speed entspricht der Geschwindigkeit des Verkehrsteilnehmer (1 bis 5). Mit typ wird entsprechend dem Enum `TrafficObject.Fahrzeugtyp` angegeben, ob es sich um einen PKW oder LKW handelt. Mit direction und nextDirction wird entsprechend dem Enum `TrafficObject.Dir` angegeben, in welche Richtung der Verkehrsteilnehmer fährt und im nächsten Streckenabschitt fahren möchte.

- `public void addCarToEntryPoint(EntryPoint entrypoint, int typ, int speed = 5, int moveForward = 0)`

  Hierbei handelt es sich um eine alternative Funktion (vgl. `createNewVerkehrsteilnehmer`) zum Hinzufügen eines neuen Verkehrsteilnehmers. Anstatt einer xy-Postion wird ein Eingangspunkt „entrypoint“ (Ein-/Ausfahrt) angegeben. Wenn entrypoint null ist, wird der erste entrypoint gewählt. Direction

und NextDirection wird von der Funktion ermittelt und kann daher nicht angegeben werden. Mit moveForward kann ein Verkehrsteilnehmer erst ein Stück weiter vorne in seiner Fahrrichtung gespawnt werden.

- `public void removeVerkehrsteilnehmer(int id)`
  Hierbei handelt es sich um eine Funktion zum Entfernen eines Verkehrsteilnehmers mit einer bestimmten Id.

- `private int checkIfTilesAreEmpty(int startX, int startY, int destX, int destY, int requesterId, Boolean ignoreStandingCars)`
  Hierbei handelt es sich um eine Funktion, die überprüft,ob sich andere Verkehrsteilnehmer sich in einem bestimmten Bereich der Simulation befinden. startX, startY, destX und destY spannen das Rechteck (2 Eckpunkte). auf, in dem überprüft wird. Mit requesterId wird die Id des Fahrzeugs, für das überprüft wird, sodass dieses ignoriert wird. Falls nicht für ein Fahrzeug überprüft wird, kann dieser Parameter auf -1 gesetzt werden. Mit ignoreStandingCars kann gesetzt werden, ob sich nicht bewegende Fahrzeuge ignoriert werden (wird für „Vorrangverzicht“ genutzt). Der Rückgabewert entspricht der Anzahl der gefunden Fahrzeuge.

- `private Boolean checkIfCanDrive4Way(TrafficObject obj, StreetInfo info)`
  Mit dieser Funktion wird überprüft, ob ein Fahrzeug (`TrafficObject obj`) in eine Kreuzung einfahren darf/kann. Hierzu müssen Informationen über die Kreuzung angegeben werden (`StreetInfo info`).

- `private Boolean checkIfCanDrive4WayWithoutTrafficLight (TrafficObject obj)`
  Mit dieser Funktion wird überprüft, ob ein Fahrzeug (`TrafficObject obj`) in eine ungeregelte Kreuzung einfahren darf/kann.

- `private Boolean checkIfCanDriveWithTrafficLight(TrafficObject obj, int trafficLightColour)`
  Mit dieser Funktion wird überprüft, ob ein Fahrzeug (`TrafficObject obj`) in eine gerelegte Kreuzung einfahren darf/kann, wenn seine Ampel auf Grün gesetzt ist.

- `private Tuple<int,int> getNextRoadTileXY(TrafficObject obj)`

Mit dieser Funktion wird errechnet wo sich ein Fahrzeug als nächstes befinden würde, wenn es fahren kann. Diese x,y-Koordinate wird in einem `Tuple<int,int>` zurückgegeben.

- `private Boolean checkIfIntersectionEntryIsEmpty(int x, int y, int direction)`
  Mit dieser Funktion wird überprüft ob an einer Kreuzung ein Fahrzeug aus einer bestimmten Richtung (`int direction`) kommt.(x,y müssen einer Position entsprechen, die auf dem Streckenabschnitt (100x100-Tile) dieser Kreuzung zu finden ist, üblicherweise Position des abfragenden Fahrzeugs). **Wichtig:** Mit direction wird angegeben in welche Richtung solch ein Fahrzeug kommen würde, also müsste auf ein Auto, das von rechts kommt, mit „Left"-direction überprüft werden.

- `private int getStreetRegion(int x, int y)`
  Diese Funktion ermittelt anhand xy-Koordinaten auf welchen Teil eines Straßenabschnitt mit Kreuzung man sich befindet:
  - `NormalStreet = 0` normale, gerade Straße
  - `IntersectionAhead = 1` Kreuzungseingangsbereich
  - `Intersection = 2` Kreuzungsbereich
- `private Boolean checkIfObstacleAhead(int y, int x, int direction, int speed)`
  Diese Funktion überprüft ob sich ein oder mehrere Hindernis(se) in einer bestimmten Richtung (direction) in einer bestimmten Reichweite (speed) befindet.

- `private Boolean checkIfCanPassObstacle(int y, int x, int direction)`
  Diese Funktion überprüft, wenn ein Hindernis im Weg ist, ob dieses mit fahren auf die linke Fahrspur umfahren werden kann.

- `public void updateCarAmount(int cars)`
  Diese Funktion wird zum Ändern der Zielmenge an Fahrzeuge verwendet.

- `public void updateTruckRatio(int percent)`
  Diese Funktion setzt den Zielwert an LKWs fest (in Prozent).

- `private int getCurrentTruckRatio()`
  Diese Funktion ermittelt das aktuellen prozentuellen Anteil an LKWs.

3.4.2 Attribute von „TrafficHandler“

```
static TrafficHandler instance = null;
int targetNumberOfCars = 20;
int truckratio = 0;
List<TrafficObject> trafficobjs;
ITeilnehmer eb;
IObject oh;
RabbitMQ.RabbitMQHandler mqhandler;
RabbitMQ.RabbitMQHandler.RemoteTransaction remoteTransaction;
Random rng = new Random();
int id_number;
```

3.4.3 Attribute von „TrafficHandler“

```
int Id
int X
int Y
int NextX
int NextY
int Speed
int Direction
int NextDirection
Boolean MayDrive
int Typ
int PassingObstacleStatus
int WaitingCounter
```

3.4.4 Enums von „TrafficHandler“

- `public enum Dir{Up  =0, Left = 1, Down = 2, Right = 3 };`
  Diese Richtung entsprechend Nord, West, Süd, Ost
- `public enum Fahrzeugtyp { Car = 0, Truck = 1};`
  Fahrzeugtypdefinition ob PKW oder LKW
- `public enum PassingObstStatus { RightSide = 0, GoingWrongSide = 1, WrongSide = 2, GoingRightSide = 3, TurnAround = 4};`
  Angabe wie ein Auto gerade auf der Straße fährt.

3.5 Ampelsteuerung

Die Ampelsteuerung besteht aus 2 Klassen, siehe Kapitel 2.5.1 und 2.5.2 sowie deren Methoden und Variablen. In diesem Abschnitt wird genauer auf diese eingegangen.

3.5.1 Funktionen der Klasse Ampel

Damit die Phasendauer einer Ampel, der Ausfall einer Ampel oder die ID abgefragt werden kann, wurden Getter und Setter sowie variablen in der Klasse angelegt:

Funktion	Erklärung
getStatus()	Holt Ampelstatus
getRotPhase()	Gibt Zeit der Rotphase wieder

getGelbPhase()	Gibt Zeit der Gelbphase wieder
getGruenPhase()	Gibt Zeit der Grünphase wieder
getDefect()	Gibt Defekt der Ampel an
getID()	Holt ID der Ampel
getSekundenzähler()	Erhält Sekundenzähler
setSekundenzähler(int value)	Setzt den Sekundenzähler
setRotPhase(int value)	Setzt die Rotphasendauer
setGelbPhase(int value)	Setzt die Gelbphasendauer
setGruenPhase(int value)	Setzt die Grünphasendauer
setDefect(int value)	Setzt eine Ampel Defekt
setStatus(int value)	Setzt den Status einer Ampel
setID(int value)	Setzt bei Erstellung die ID der Ampel

Tabelle 7: Funktionen der Getter und Setter - Ampelsteuerung

Die Variablen hierzu sind:

```
int sekundenzähler = 0;
int Status;
int ID;
bool defect = false;
int rotphase = 3;
int gelbphase = 1;
int gruenphase = 3;
```

3.5.2 Funktionen der Klasse Ampelsteuerung

Damit die Ampelsteuerung für die Verkehrssimulation verfügbar ist und per WCF erreichbar ist, muss folgendes hinzugefügt werden:

```
[ServiceBehavior(Name = "Ampelsteuerung", InstanceContextMode = InstanceContextMode.Single)]
```

Hiermit gibt man die Ampelsteuerung unter dem Namen „Ampelsteuerung“ am Localhost bekannt und lässt somit den Zugriff auf diese zu. Weiters werden folgende Funktionen benötigt:

- `private void StartServer();`

  Diese Funktion erstellt einen ServiceHost, welcher von der Verkehrssimulation zum Erstellen einer Verbindung benötigt wird. Weiters wird gewartet, bis von der Verkehrssimulation die Anzahl der benötigten Ampeln geliefert wird. Die Ampeln werden erstellt und ein Timer zum Schalten der Phasen wird pro Ampel gestartet.

- `private Task HandleTimer();`

  Dieser Timer inkrementiert den Sekundenzähler aus der Klasse „Ampeln“ und erzeugt so einen Phasenwechsel.

- `public int getAmpelStatus(int ampelid);`

  Diese Funktion erhält die eine AmpelID. Diese ID zeigt auf eine Ampel und gibt den Status dieser Ampel zurück. Wird „0“ anstelle der AmpelID mitgegeben, so erhält man den Status aller erstellten Ampeln.

- `public string getAmpelAusfall(int ampelid)`

  Diese Funktion gibt Information darüber, ob eine Ampel ausgefallen ist aufgrund der mitgegebenen AmpelID.

- `public void setAmpelAusfall(int ampelid)`

  Mit setAmpelAusfall ist man in der Lage, eine Ampel aufgrund dessen ID Auszuschalten.

- `public void setAmpelOn(int ampelid);`

  Hier kann man eine Ampel mit mitgegebener ID wieder einschalten, sofern sie ausgeschaltet ist.

- `public void setAmpelStatus(int ampelid, int neuerStatus);`

  Hier kann man den Status einer Ampel manipulieren bzw. angeben, ob eine Ampel Rot, Gelb, oder Grün ist.

- `public string getRotPhase(int ampelid);`

  Hier erhält man aufgrund der ID die Zeit der Rotphase wieder.

- `public string getGelbPhase(int ampelid);`

  Hier erhält man aufgrund der ID die Zeit der Gelbphase wieder.

- `public string getGruenPhase(int ampelid);`

  Hier erhält man aufgrund der ID die Zeit der Grünphase wieder.

- `public void setRotPhase(int ampelid, int zeit);`

  Hier kann man aufgrund der ID die Zeit der Rotphase einstellen.

- `public void setGelbPhase(int ampelid, int zeit);`

  Hier kann man aufgrund der ID die Zeit der Gelbphase einstellen.

- `public void setGruenPhase(int ampelid, int zeit);`

  Aufgrund der ID der Ampel wird hier die Zeit der Grünphase eingestellt.

- `public void setAmpelAnzahl(int anzahl);`

  Diese Funktion wird zu Beginn ausgeführt, um die Anzahl der Ampeln zu erstellen. Diese Funktion wird von der Verkehrssimulation verwendet.

- `public List<Ampeln> factory(int anzahl);`

  Diese Funktion erstellt die Ampeln.

- `static void Main();`

  In der Main wird die Ampelsteuerung ausgeführt.

  Alle Funktionen werden über das Interface „`IAmpelService`“ der Verkehrssimulation zur Verfügung gestellt.

3.5.3 Variablen der Klasse Ampelsteuerung

Damit die Funktionen miteinander Funktionieren, wurden folgende Variablen benötigt:

```
ServiceHost host;
 bool _serverRunning = false;
 bool run = false;
 public static List<Ampeln> Trafficlights = new List<Ampeln>();
 public static int Anzahl = 0;
 static Timer Ampeltimer;
```

3.6 RabbitMQ

Diese Komponente wurde mit der Klasse RabbitMQHandler und RemoteTransaction realisiert.

3.6.1 Funktionen der Klasse RabbitMQHandler

Die Funktionen dieser Klasse sind für das Senden und Empfangen benötigt.

- `public void Send(RemoteTransaction transaction, String group)`

  Diese Funktion ist für das Senden der Fahrzeuge in der Verkehrssimulation verantwortlich. Der 2. Parameter gibt den Gruppennamen an, an welcher das Fahrzeug gesendet wird.

- `public  void Receive()`

  Diese Funktion wird für das Empfangen anderer Fahrzeuge anderer Gruppen verwendet.

3.6.2 Variablen der Klasse RabbitMQHandler

Damit die RabbitMQ Klasse auf die Funktionen der Klasse RemoteTransaciotn zugreifen kann und eine Verbindung zu unserem Verkehrsnetz hergestellt werden kann wurden folgende Variablen verwendet:

```
RemoteTransaction remoteTransaction;
ITrafficHandler th;
```

3.6.3 Funktionen der Klasse Remotetransaction

Die Funktionen dieser Klasse sind für die Transaktionen zwischen den Gruppen zuständig.

- `public static long GetCurrentUnixTimestampMillis()`
  Diese Funktion wird verwendet um den aktuellen Timestamp in Millisekunden zu erhalten.
- `public RemoteTransaction(double speed, string cartype, int errorcode = 0)`

  Mithilfe dieser Funktion wird eine Transaktion erstellt. Hierzu werden die Parameter „speed“, „cartype“ und „errorcode“ verwendet.

- `public string GetStatus()`

  Diese Funktion liefert den Status zurück. Hierzu werden die Errorcodes ausgelesen und in einen lesbaren Text umgewandelt.

3.6.4 Getter und Setter der Klasse Remotetransaction

Die nachfolgende Tabelle zeigt die Getter und Setter dieser Klasse.

Funktion	Erklärung
`public Guid CarId { get; set; }`	Holt und gibt die CarId
`public Guid GroupId { get; set; }`	Holt und gibt die GroupId
`public String CarType { get; set;}`	Holt und gibt die CarType
`public double Speed { get; set; }`	Holt und gibt die Geschwindigkeit
`public long Timestamp { get; set; }`	Holt und gibt den Timestamp
`public int Errorcode { get; set; }`	Holt und gibt den Errorcode

Tabelle 8: Funktionen der Getter und Setter der RabbitMQ Klasse RemoteTransaction

4 Zusätzliche externe Komponenten

Als zusätzliche Komponenten wurden NewtonSoftJSON und ein Rabbit MQ Server verwendet. Die externe JSON Komponente wurde für die Generierung des Verkehrsnetzes verwendet und der Rabbit MQ Server, damit die Kommunikation zwischen den anderen Gruppen hergestellt werden konnte.

5 Zusammenfassung und Ausblick

Die Funktionalitäten konnten fast zur Gänze implementiert bzw. umgesetzt werden. Das Erstellen von Hindernissen in einer Kreuzung wurde aufgrund der Komplexität außen vorgelassen.