SOFTWARE-ARCHITEKTUREN

Verkehrssimulation Dokumentation

durchgeführt am
Studiengang Informationstechnik & System–Management
an der
Fachhochschule Salzburg GmbH

vorgelegt von

Fabian Schörghofer, Andreas Reschenhofer, Lukas Altenhuber, Paul Riedl, Mike Thomas

Leiter des Studiengangs:	FH-Prof. DI Dr. Gerhard Jöchtl
Betreuer:	DI (FH) DI Roland Graf, MSc

Puch am, 12. Juli 2017

Inhaltsverzeichnis

1 Einführung

Im Abschnitt 1 wird dem Leser ein Überblick der Aufgaben des Verkehrssimulationsprojekts gegeben.

1.1 Aufgabenstellung

Im Rahmen dieser Übung soll eine Verkehrssimulation realisiert werden. Dabei sollen sich diverse Verkehrsteilnehmer, zum Beispiel Autos und Busse, entsprechend der üblichen Straßenverkehrsregeln in einem gegebenen Straßennetz bewegen. Der Anwender der Simulation soll die Möglichkeit haben sowohl Simulationsparameter als auch Straßennetze modifizieren zu können. Für die Einstellung der Simulationsparameter soll eine Editor Oberfläche erstellt werden. Über diese Oberfläche hat der Benutzer die Möglichkeit vor und während der Simulation, Parameter wie die maximale Geschwindigkeit der Fahrzeuge, Beschleunigung der Fahrzeuge, Einfahrtsrate der Fahrzeuge oder Ampelschaltzeiten anzupassen. Die Simulation soll in einer zwei- oder dreidimensionalen graphischen Oberfläche dargestellt werden. Der Benutzer soll aus diversen Kartentypen, welche persistent gespeichert sein sollen, zu Beginn der Simulation auswählen können.

In den weiteren Lehreinheiten wurden weitere Aufgaben definiert. So sollte eine gruppenübergreifende Kommunikation möglich sein, Autos sollen von einer Simulation in die nächste fahren können.

Eine weitere Aufgabenstellung war die Möglichkeit ein Hinderniss in die Fahrbahn zu platzieren. Dies sollte frei möglich sein (also zur Laufzeit). Ein Auto soll dieses Hindernis umfahren können und gleichzeitig eine Kollision mit einem anderen Auto verhindern.

2 Software-Architektur

In diesem Kapitel wird die Architektur in Form des Arc42-Templates sowohl grafisch als auch textuell dargestellt.

2.1 Randbedingungen

Die Verwendung der Programmiersprache C# und die Auslagerung der Logik für geregelte Kreuzungen sind als Vorgaben für die Realisierung der Verkehrssimulation gegeben.

Es sollen verschiedene Einstellungen zur Laufzeit geändert werden können, um die Simulation entsprechend zu strapazieren. Dazu gehören:

- Anzahl der maximalen Fahrzeuge während der Simulation
- Maximale Geschwindigkeit der Fahrzeuge
- Die Zeit in der neue Fahrzeuge erstellt werden
- Verhältnis zwischen PKW und LKW

Des Weiteren soll über eine vorab definierte Schnittstelle ein Austausch von Fahrzeugen innerhalb der Gruppen erfolgen können.

2.2 Kontextabgrenzung

Einschränkungen im Detailgrad sowie im Umfang der Implementierung wurde keine vorgegeben. Es soll nur die Aufgabe mit den vorgegebenen Randbedingungen erfüllt werden. Wie diese umgesetzt werden, ist dem Projektteam überlassen.

2.3 Lösungsstrategie

Da ein Teil der Projekt Mitarbeiter bereits Erfahrungen mit Unity gemacht haben, wurde überlegt, diesen auf für die Verkehrsimulation zu verwenden. Durch die Verwendung des Unity Editors, welche als Laufzeit- und Entwicklungsumgebung für Spiele dient, wird ein Großteil bereits von der von Unity zur Verfügung gestellten "Game Engineäbgenommen. Da Unity eine 2D bzw. 3D Game Engine zur Verfügung stellt wurde überlegt, welche von beiden die meisten Vorteile für eine Verkehrssimulation bringt. Da jedoch der Unterschied nur in der grafischen Darstellung liegt, was bedeutet das Logik von Autos, Straße, Ampeln usw. sowohl in 2D, als auch in 3D implementiert hätte werden müssen. Da eine Simulation mit 3D Modellen ansprechender aussieht, fiel die Entscheidung auf die 3D Implementierung.

Eine weitere Lösungsstrategie ist das Aufteilen der Aufgabenbereiche. Folgende Teilbereiche für die Implementierung wurden überlegt:

- Ampelsteuerung
- Erstellung von Straßen und Kreuzungen
- Logik und Verhalten in Fahrzeugen
- Straßenlogik (Waypoints)

2.4 Bausteinsicht

In diesem Abschnitt folgt die Beschreibung der Komponenten und in Abbildung 2.1 ist das Komponenten Diagramm zur Verkehrssimulation abgebildet.

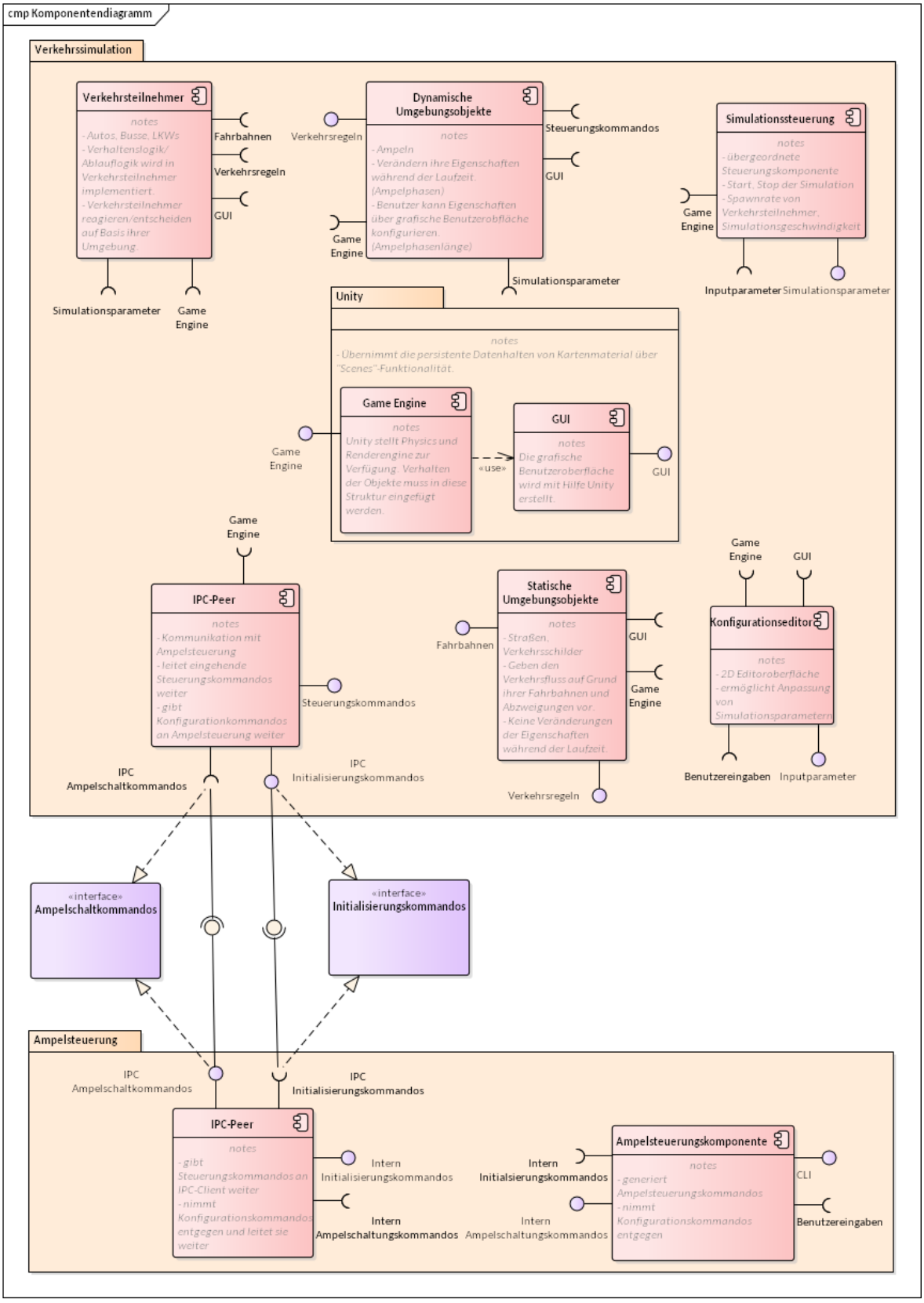

Abbildung 2.1: Komponenten Diagramm Verkehrssimulation

Verkehrssimulation und Ampelsteuerung
Sowohl Verkehrssimulation als auch Ampelsteuerung sind eigene Executables. Die Verkehrssimulation wird aus dem Unity-Projekt erzeugt und beinhaltet alle oben angegebenen Komponenten. Die Ampelsteuerung ist eine Konsolen Applikation, welche über die IPC-Komponenten mit der Verkehrssimulation kommuniziert.

Verkehrsteilnehmer
Zu diesen Komponenten gehören alle sich aktiv in der Simulation bewegenden Objekte, wie Autos, Busse und LKWs. Die einzelnen Komponenten bewegen sich autonom voneinander im Straßennetz fort. Die Komponenten scannen ihre Umgebung auf Fahrbahnen, Verkehrsregeln und andere Verkehrsteilnehmer. Auf Basis dieses Scans entscheiden sie ihre nächste Aktion, wie zum Beispiel Beschleunigen, Bremsen oder Abbiegen. Verkehrsteilnehmer können sich ausschließlich auf Fahrbahnen bewegen und beachten alle Verkehrsregeln innerhalb ihres Aktionsradius. Weiters benötigen sie Simulationsparameter, die über ihre maximale Geschwindigkeit, Beschleunigung und Bremsverhalten entscheiden. Verkehrsteilnehmer treffen ihre Entscheidungen bei jedem Renderdurchlauf der Game Engine. Die Verknüpfung der Verkehrsteilnehmer mit der GUI erfolgt abstrahiert durch Unity.

Dynamische Umgebungsobjekte
Dynamische Umgebungsobjekte sind Objekte, welche während der Simulationslaufzeit ihre Eigenschaften ändern, jedoch nicht ihre Position. Konkret sind dynamische Umgebungsobjekte Ampeln. Ampeln bieten den umliegenden Verkehrsteilnehmern ihren derzeitigen Status als Verkehrsregeln an, welche diese beachten müssen. Dynamische Elemente benötigen Simulationsparameter, welche ihre Schaltzeiten vorgeben. Konkret geben die Simulationsparameter die Phasenzeiten der Ampeln an. Weiters benötigen dynamische Umgebungsobjekte Steuerungskommandos um zwischen diversen Modi zu wechseln. Über Steuerungskommandos können Ampeln von automatischen Betrieb in einen inaktiven dauerhaft Gelb blinkenden Status gebracht werden. In jedem Renderzyklus der Game Engine wird auf neue Steuerungskommandos geprüft und falls nötig der vorgegebene Schaltvorgang eingeleitet. Die Verknüpfung der dynamischen Umgebungsobjekte mit der GUI erfolgt abstrahiert durch Unity.

Simulationssteuerung
Die Simulationssteuerung stellt eine übergeordnete Kontrollinstanz der Simulation dar. Sie legt globale Einstellungen für Simulationskomponenten zentral fest. Die Simulationssteuerung bietet Simulationsparameter für andere Simulationskomponenten an, welche diese abrufen können. Konkrete Simulationsparameter sind Schaltzeiten für Ampeln, Höchstgeschwindigkeiten für Verkehrsteilnehmer oder Spawnraten für neue Verkehrsteilnehmer. Die Simulationssteuerung benötigt Inputparameter, welche von außen die Simulationsparameter bestimmen. Die Aktualisierung der Simulationsparameter auf Basis der Inputparameter erfolgt bei jedem Renderzyklus der Game Engine.

Konfigurationseditor

Der Konfigurationseditor stellt eine zweidimensionale graphische Benutzeroberfläche dar, welche es dem Benutzer ermöglicht angebotene Inputparameter für die Simulation zu verändern. Vom Benutzer geänderte Inputparameter werden bei jedem Renderzyklus der Game Engine verarbeitet und entsprechend weiter geleitet.

IPC-Peer Verkehrssimulation
Der IPC-Peer Verkehrssimulation repräsentiert eine Instanz der Inter Process Communication zwischen den Executables Verkehrssimulation und Ampelsteuerung dar. Dieser IPC-Peer gibt Initialisierungskommandos an die Ampelsteuerung weiter. Über diese Kommandos werden die benötigte Anzahl an Ampeln mit den korrekten Initialisierungsparametern angelegt. Weiters werden Ampelschaltkommandos entgegen genommen und weiter geleitet um den Modus einer Ampel zu wechseln. Ein- und ausgehende Kommandos während in jedem Renderzyklus der Game Engine bearbeitet.

Statische Umgebungsobjekte
Statische Umgebungsobjekte repräsentieren Simulationskomponente, welche weder ihre Eigenschaften noch ihre Position während der Simulationslaufzeit verändern. Konkret sind Straßen und Verkehrsschilder statische Umgebungsobjekte. Sie bieten Fahrbahnen für die Verkehrsteilnehmer an, auf welchen diese sich bewegen können. Zusätzlich werden auch Verkehrsregeln vorgegeben, welche von den Verkehrsteilnehmer beachtet werden müssen. Die statischen Umgebungsobjekte werden von der Game Engine gerendert, jedoch sollte diese Komponente keine Logik besitzen. Die Verknüpfung der statischen Umgebungsobjekte mit der GUI erfolgt abstrahiert durch Unity.

IPC-Peer Ampelsteuerung
Die Komponente IPC-Peer Ampelsteuerung bildet die Gegenstelle der zuvor beschriebenen IPC-Peer Verkehrssimulation. Sie gibt Ampelschaltkommandos zur Gegenstelle weiter und nimmt Initialisierungskommandos entgegen. Diese Kommandos werden intern, innerhalb der Ampelsteuerung Executable an die Ampelsteuerungskomponente weiter gegeben.

Ampelsteuerungskomponente
Die Ampelsteuerungskomponente übernimmt die Verwaltung der einzelnen Ampelinstanzen. Anhand eingehender Initialisierungskommandos werden Ampelinstanzen mit den benötigten Initialisierungsparametern erstellt. Über das angebotene CLI kann der Benutzer Schaltbefehle absetzen, welche über Ampelschaltungskommandos weiter geleitet werden.

Message Queue
Die Message Queue übernimmt die Kommunikation mit den anderen Gruppen. Die Kommunikation mit den anderen Gruppen wird über einen externen Server abgewickelt, auf denen sich die Gruppen anmelden und ihre Queues abonnieren können. Ein Format, basierend auf JSON wurde zwischen den Gruppenteilnehmern definiert.

Als Protokoll wird dabei AMPQ verwendet, der dazugehörende Server, RabbitMQ implementiert diese Protokoll und ermöglicht die Übertragung von Nachrichten.

2.5 Laufzeitsicht

Die Ampelsteuerung wird beim Start der Verkehrssimulation initialisiert. Danach wird in regelmäßigen Abständen die Ampelsteuerung von der Verkehrssimulation gepollt und der Status der Ampeln abgefragt. Dieser Ablauf ist im Sequenz Diagramm in Abbildung 2.2 dargestellt.

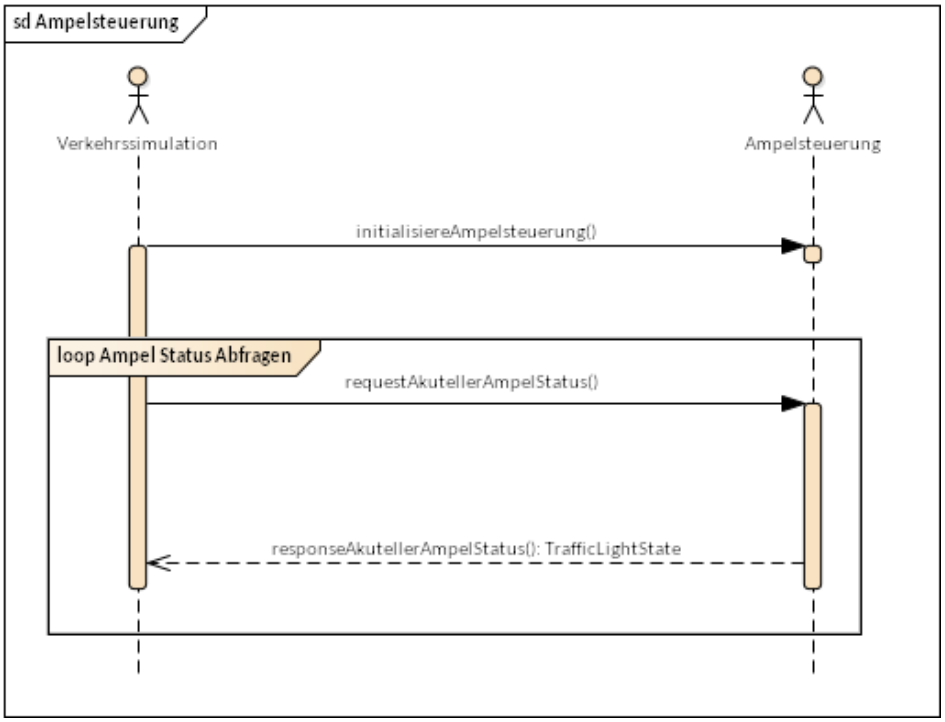

Abbildung 2.2: Sequenz Diagramm Verkehrssimulation - Ampelsteuerung

2.6 Verteilungssicht

Die Komponenten Ampelsteuerung und Verkehrssimulation sind auf zwei Geräte aufgeteilt. Die Ampelsteuerung läuft zentral auf einem Server und die Verkehrssimulation kann lokal auf einem PC ausgeführt werden. Die Verteilung ist im Deployment Diagramm in Abbildung 2.3 abgebildet.

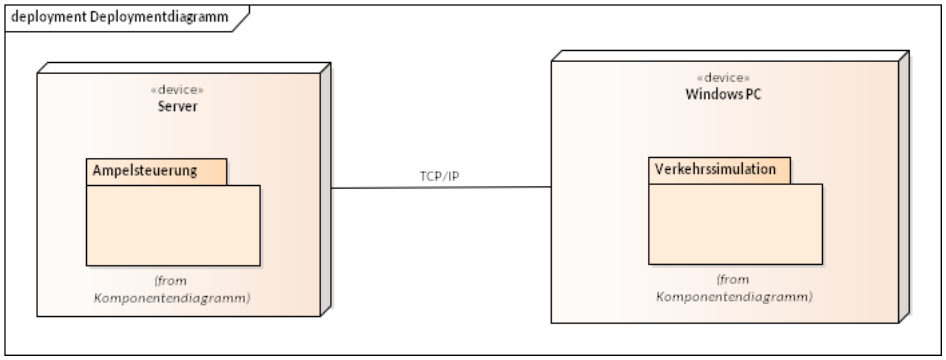

Abbildung 2.3: Deployment Diagramm der Verkehrssimulation

2.7 Querschnittliche Konzepte

Straßen und Kreuzungen werden als GameObjects abgespeichert. Dadurch wird ermöglicht, dass die erstellten Teilstücke wiederverwendet werden können. Somit kann die Erstellung der Welt per "Drag & Dropünd genaues Ausrichten erfolgen. Folgende Straßenstücke wurden in verschiedenen Längen erstellt:

- T-Kreuzung (mit Ampeln)
- X-Kreuzung (mit Ampeln)
- Spawn/Despan-Zone
- Gerade Teilstrecke (5m, 6m, 24m, 30m, 40m, 50m, 60m)
- 90 Grad Kurve

Die damit wiederverwendbaren Teilstücke beinhalten bereits die benötigten Kollider, welche für die Fahrzeuglogik verwendet werden, sowie Shaders zum Darstellen der Straßenstücke.

Auch das verwenden der Ampelsteuerung funktioniert über eine definierte Schnittstelle. Hierbei können einfach neue Kreuzungen mit Ampeln erstellt werden und die Ampelsteuerung regelt automatisch die neu erstellte Ampel.

2.8 Entwurfsentscheidungen

Wie in Kapitel 2.3 bereits kurz beschrieben wurde die Unity Engine als treibende Engine verwendet, da diese bereits viele Sachen implementiert, welche von uns nicht mehr berücksichtigt werden müssen (z.B. Handling wann wird welches GameObject angesprochen usw.). Ein weiterer Grund für die Entscheidung mit Unity war die schnelle und anschauliche Darstellung von 3D-Modellen, sowie die zahlreich vorhandenen Tutorials über Unity und die verschiedenen Möglichkeiten.

2.9 Qualitätsszenarien

Keine Vorhanden, da es sich hierbei nur um ein Studierendenprojekt handelt.

2.10 Risiken und technische Schulden

Da die Verkehrssimulation nun zur Genze mit der Unity-Engine funktioniert, kann des im Falle von Updates von Unity vorkommen, das API-Updates durchgeführt werden und diese nicht mehr mit den bereits eingebauten funktionierenden Funktionen übereinstimmen. Dieses führt daraufhin zu Mehraufwand, der berücksichtigt werden müsste.

3 Design

In diesem Kapitel werden die Implementierungen der einzelnen Komponenten sowie deren Schnittstellen zu anderen Komponenten dokumentiert.

3.1 Strassensystem

3.1.1 Ampeln

3.1.2 Strassen

3.1.3 Kreuzungen

3.2 Fahrzeuglogik

3.3 Ampelsteuerung

3.4 Hindernisse

Mit einem Klick der Maus sollen überall in der Welt Hindernisse platziert werden, welche anschließend von Fahrzeugen umfahren werden müssen. Durch einen weiteren Klick auf ein Hindernis soll dieses wieder gelöscht werden.

Unity ermöglicht das Erstellen von GameObjects zur Laufzeit. Durch den Klick auf die Welt können die Koordinaten des Klicks festgestellt werden und somit das GameObject, welches bereits beim Start der Verkehrssimulation geladen wurde, platziert werden.

Durch das Halten eines Buttons auf der Tastatur soll die Erstellung von Hindernissen aktiviert werden. Dies verhindert, dass der Benutzer unbeabsichtigt zu viele Hindernisse erstellt.

3.5 Aus- und Einfahren von Fahrzeugen anderer Gruppen

4 Implementierung

In diesem Kapitel werden die Implementierungen der einzelnen Komponenten sowie deren Schnittstellen zu anderen Komponenten dokumentiert.

4.1 Erstellung der Welt

4.2 Fahrzeugsteuerung

4.3 Erkennen und Reagieren auf Ampeln

4.4 Fahrzeug Kollisionserkennung

4.5 Ampelsteuerung

4.6 Erstellen und Löschen von Hindernisse

Um im Environment Hindernisse zur Laufzeit zu Erstellen, muss dem Terrain-GameObject ein Skript (Obstacles.cs) hinzugefügt werden. In diesem Skript findet die Abarbeitung des Inputs des Users statt.

Wie bereits in 3.4 erläutert wird zum Start der Verkehrssimulation das Hindernis geladen. Dies erfolgt über den "Load"Befehl.

```
private GameObject prefabLog;
void Start()
{
  prefabLog = Resources.Load("Rock", typeof(GameObject)) as GameObject;
}
```

Der in 3.4 beschrieben soll ein Button gedrückt gehalten werden um Hindernisse spawnen zu können. Dieser wird über einen KeyCode definiert. Im Skript wird nun in jedem Update Aufruf darauf gewartet, ob der definierte Button gedrückt und ein "MouseDownËvent vorkommt. Mit diesem Event kann die Position der Maus auf dem Bildschirm herausgefunden werden, jedoch stimmen diese nicht mit den Welt-Koordinaten überein. Deshalb muss hier eine Umwandlung durchgeführt werden, welche mit Hilfe eines Raycasts gelöst wurde. Durch das Instanzieren wird das neu erstellte Hindernis in der Welt platziert.

```
private KeyCode shiftLeft = KeyCode.LeftShift;
if(Input.GetMouseButtonDown(0) && Input.GetKey(shiftLeft)) //Left mouse
    button clicked
{
  Vector3 mousePosition = Input.mousePosition;
```

```
  var ray = Camera.main.ScreenPointToRay(mousePosition);
  RaycastHit hit;
  if(Physics.Raycast(ray, out hit, 1000f))
  {
    Vector3 position = hit.point;   Vector3 yOffset = new Vector3(0,
       1.5f, 0);
    position += yOffset;
    GameObject prefabInstance = Instantiate(prefabLog, position, new
       Quaternion()) as GameObject;
  }
}
```

Zum Löschen eines Hindernisses muss die rechte Maustaste in Kombination mit dem vorher definierten Button verwendet werden. Mittels "Destroy"wird anschließend das erkannte GameObject wieder von der Welt gelöscht.

```
if(Input.GetMouseButtonDown(1) && Input.GetKey(shiftLeft)) //Right mouse
    button clicked
{
  Vector3 mousePosition = Input.mousePosition;
  var ray = Camera.main.ScreenPointToRay(mousePosition);
  RaycastHit hit;
  if(Physics.Raycast(ray, out hit, 1000f))
  {
    Vector3 position = hit.point;
    GameObject collidedObject = hit.collider.gameObject;
    if(collidedObject.name.Equals("Rock(Clone)"))
    {
      Destroy(collidedObject);
    }
  }
}
```

4.7 Aus- und Einfahren von Fahrzeugen anderer Gruppen

Versionshistorie

Version	Datum	Autor(en)	Änderungen
0	03.04.2017	FS	Dokumentation erstellt (Vorlage: Martin Uray)
0.1	15.04.2017	MT	Komponentenbeschreibung
0.2	08.06.2017	AR	Erweiterung auf Arc42 Template

Autoren:

Andreas Reschenhofer (AR), Fabian Schörghofer (FS), Mike Thomas (MT)