

# **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

<b>1 Einführung</b>	<b>2</b>
1.1 Aufgabenstellung . . . . .	2
<b>2 Software-Architektur</b>	<b>3</b>
2.1 Randbedingungen . . . . .	3
2.2 Kontextabgrenzung . . . . .	3
2.3 Lösungsstrategie . . . . .	3
2.4 Bausteinsicht . . . . .	4
2.5 Laufzeitsicht . . . . .	8
2.6 Verteilungssicht . . . . .	9
2.7 Querschnittliche Konzepte . . . . .	10
2.8 Entwurfsentscheidungen . . . . .	11
2.9 Qualitätsszenarien . . . . .	11
2.10 Risiken und technische Schulden . . . . .	11
<b>3 Design</b>	<b>12</b>
3.1 Strassensystem . . . . .	12
3.1.1 Ampeln . . . . .	12
3.1.2 Straßen . . . . .	12
3.1.3 Kreuzungen . . . . .	13
3.1.4 Spawn . . . . .	14
3.2 Fahrzeuglogik . . . . .	14
3.3 Ampelsteuerung . . . . .	15
3.4 Hindernisse . . . . .	16
3.5 Aus- und Einfahren von Fahrzeugen anderer Gruppen . . . . .	16
<b>4 Implementierung</b>	<b>17</b>
4.1 Erstellung der Welt . . . . .	17
4.2 Fahrzeugsteuerung . . . . .	17
4.3 Erkennen und Reagieren auf Ampeln . . . . .	17
4.4 Fahrzeug Kollisionserkennung . . . . .	17
4.5 Ampelsteuerung . . . . .	17
4.6 Erstellen und Löschen von Hindernisse . . . . .	18
4.7 Aus- und Einfahren von Fahrzeugen anderer Gruppen . . . . .	19

# 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 Verkehrssimulation 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 Engine abgenommen. 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.

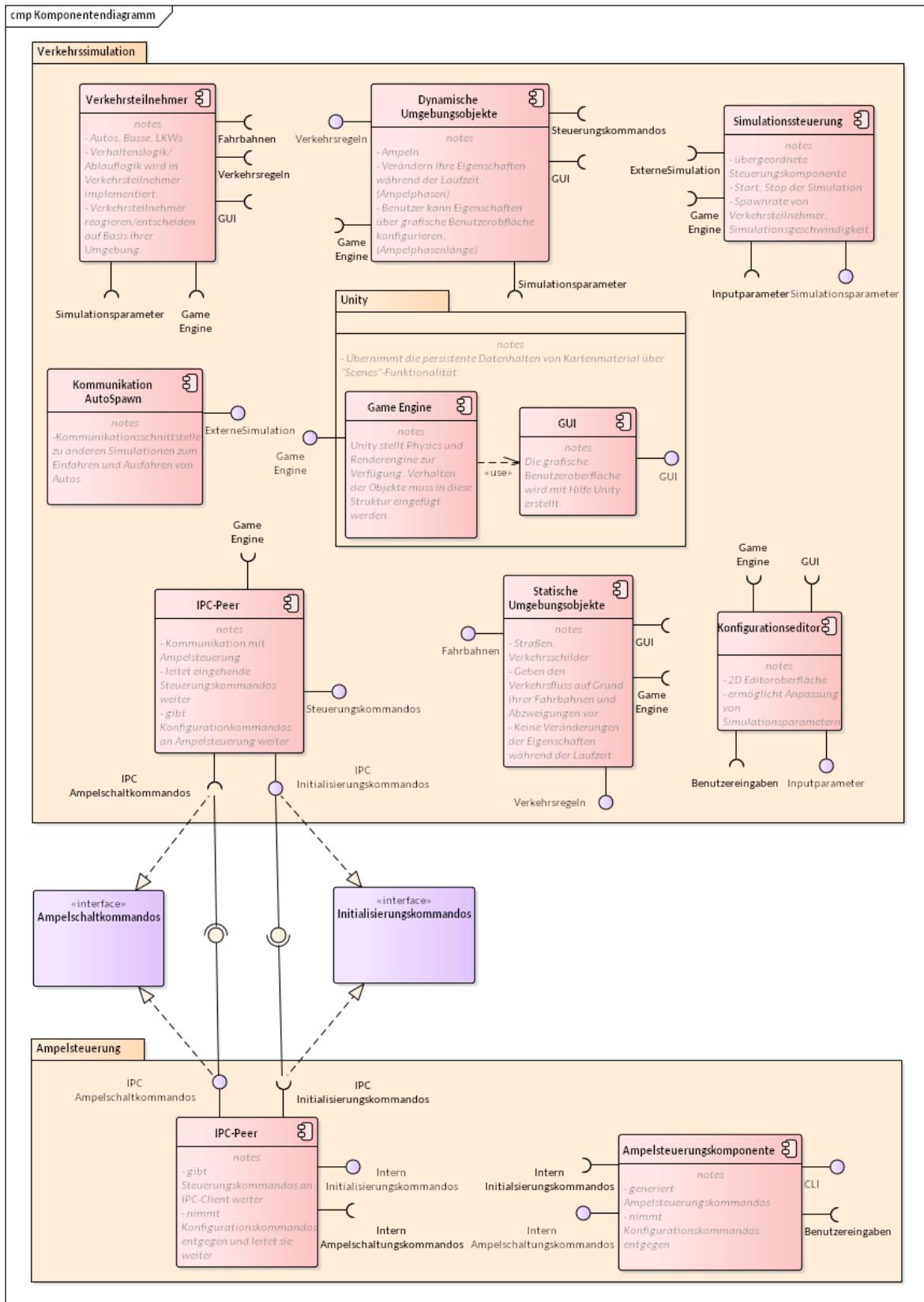


Abbildung 2.1: Komponenten Diagramm Verkehrssimulation

## **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 dazugehörige 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 gepolst und der Status der Ampeln abgefragt. Dieser Ablauf ist im Sequenz Diagramm in Abbildung 2.2 dargestellt.

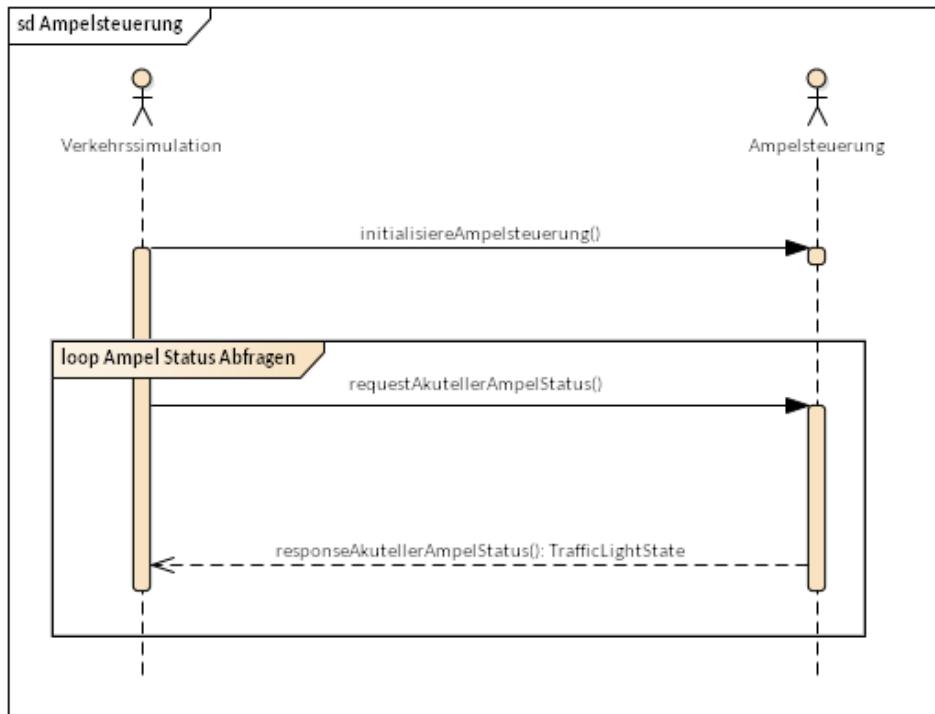


Abbildung 2.2: Sequenz Diagramm Verkehrssimulation - Ampelsteuerung

Bei der Kommunikation mit anderen Simulation zum Austausch von Fahrzeugen wird auf ein Messaging System gesetzt. Das bedeutet, dass ankommende Fahrzeuge über einen Asynchronen Push Aufruf empfangen werden und ausfahrende Fahrzeuge einen den Server gesendet werden (Abbildung 2.3).

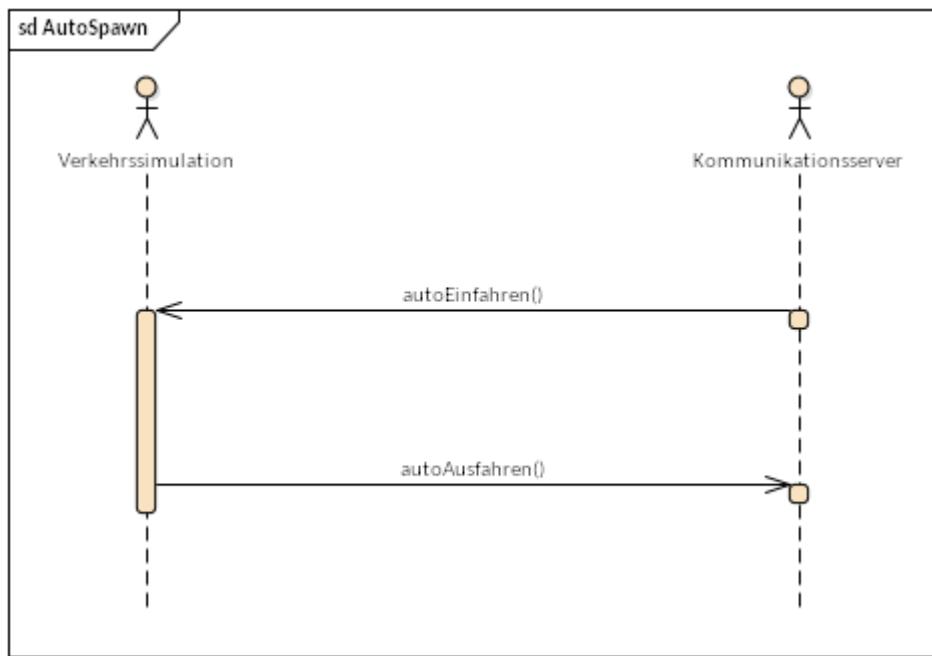


Abbildung 2.3: Sequenz Diagramm Verkehrssimulation - Ampelsteuerung

## 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. Weiters wird zur Kommunikation mit anderen Simulationen ein weiterer Server benötigt. Die Verteilung ist im Deployment Diagramm in Abbildung 2.4 abgebildet.

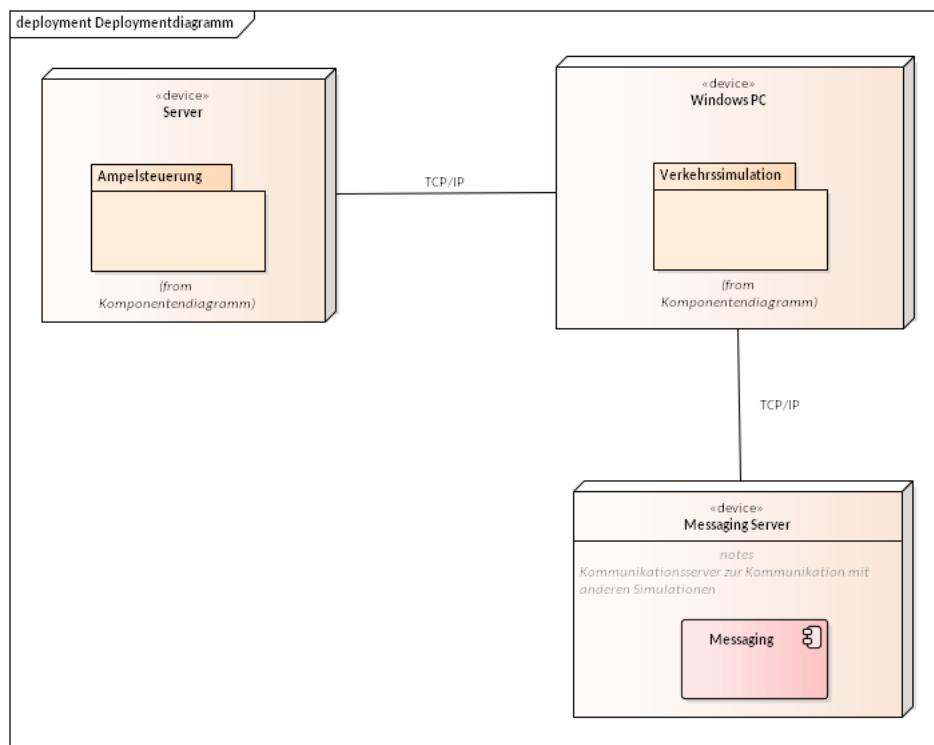


Abbildung 2.4: Deployment Diagramm der Verkehrssimulation

## 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 & Drop" und genaues Ausrichten erfolgen. Folgende Straßenstücke wurden in verschiedenen Längen erstellt:

- T-Kreuzung (mit Ampeln)
- X-Kreuzung (mit Ampeln)
- Spawn/Despawn-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

Die Ampeln wurden von uns als eigenständiges System geplant und umgesetzt. Um dieses Ziel zu erreichen erstellten wir zuerst auf Basis der Technologie TCP-Channel ein RemoteObject in Form einer eigenen DLL. Dieses RemoteObject definiert die Datenstrukturen, welche in weiterer Folge zwischen Client und Server geteilt werden. Als Serveranwendung erstellten wir eine Konsolenanwendung, welche wir auf einem öffentlich erreichbaren Linux Server mit Hilfe von Mono deployten. Als Clientgegenstellen für dieses System können verschiedene Unityinstanzen den Dienst der Serveranwendung verwenden. Sowohl am Client als auch am Server ist als DLL das RemoteObject hinterlegt. Instanzen des RemoteObjects können mit Hilfe am Server definierten Interfaces von den Clients erstellt und bearbeitet werden. Die somit erstellten Instanzen werden dann durch die TCP-Channel Technologie zwischen Client unser Server geteilt. Somit können Clients komfortabel mit ihrem lokalen Stub des RemoteObjects arbeiten ohne die Komplexität der IPC mitzubekommen.

Durch die zentrale Definition des RemoteObjects in Form einer DLL können die geteilten Datenstrukturen zentral erweitert und gewartet werden.

#### 3.1.2 Straßen

Das Design der einzelnen Straßenstücke wurde an den beiden Enden mit Collidern ausgestattet um somit ein durchgängiges Straßensystem erzeugen zu können, dem Autos folgen und dabei auf ihrer richtigen Spur bleiben.

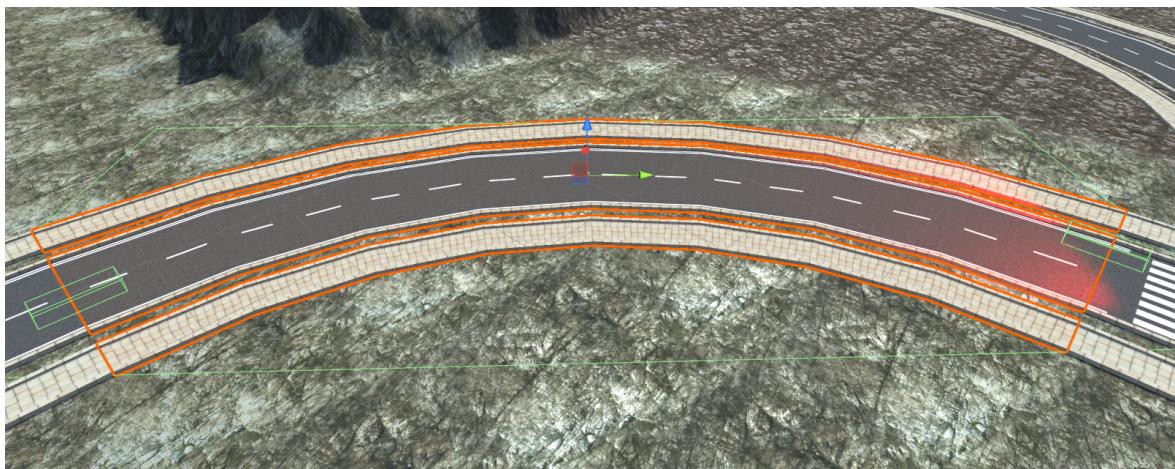


Abbildung 3.1: Collider an normaler Straße

### 3.1.3 Kreuzungen

Die auf den Kreuzungen hinzugefügten Collider dienen dem jeweiligen Straßenabschnitt als Endpunkt an welchem sich die Fahrzeuge entscheiden müssen in welche Richtung sie weiterfahren wollen.



Abbildung 3.2: Collider an Kreuzung

### 3.1.4 Spawn

Hier werden je nach Einstellung der Controls die Fahrzeuge gespawnt bzw despawnt, sobald sie mit dem in Abbildung 3.3 gezeigten Collider in Berührung kommen.

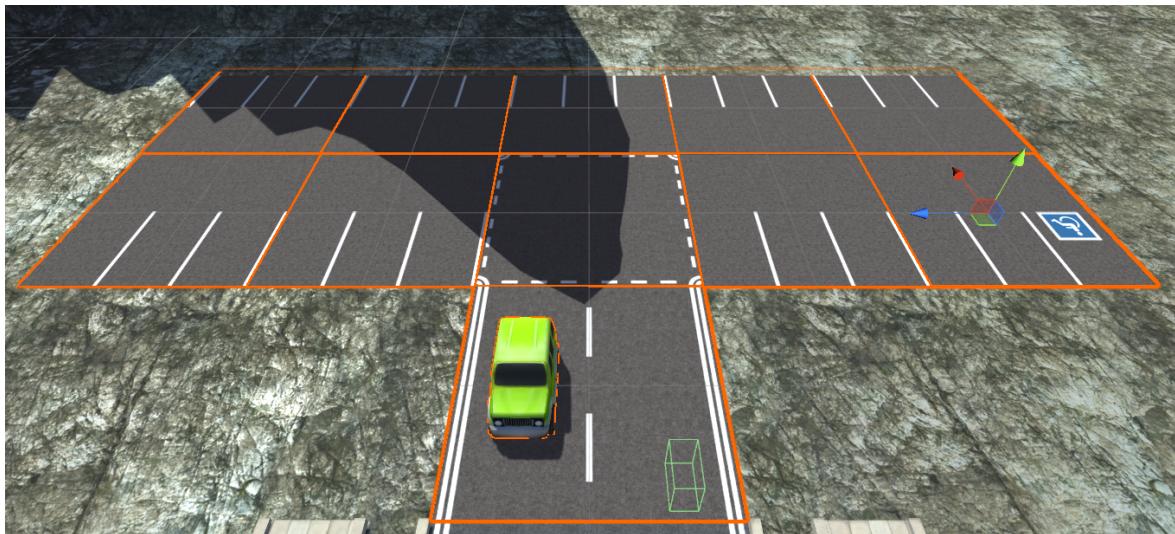


Abbildung 3.3: Spawning of vehicles

## 3.2 Fahrzeuglogik

Die Fahrzeuge implementieren alle ein Skript, welches für das Erkennen von Ampeln, Fahrzeugen und sonstiges Hindernissen zuständig ist. Dies ermöglicht eine völlige Kapselung und somit die Unabhängigkeit zur Straße und Kreuzung. Das bedeutet, dass Straßen und Kreuzungen nur sogenannte "Collider" zur Verfügung stellen, mit denen die Fahrzeuge interagieren können. Somit bleibt die gesamte Intelligenz in den Fahrzeugen.

Diese Collider werden je nach Geschwindigkeit des Fahrzeugs länger bzw. größer (wie in Abbildung 3.4 gezeigt) um somit auf Hindernisse wie andere Fahrzeuge, Kreuzungen und sonstiges frühzeitig reagieren zu können.

Das Erkennen und Verringern der Geschwindigkeit auf andere Fahrzeuge wird mittels Raycast erledigt. Die Lösung mit den Collidern war ein Problem, da sich die Fahrzeuge, nicht mehr bewegt haben, nachdem sie kollidiert sind.

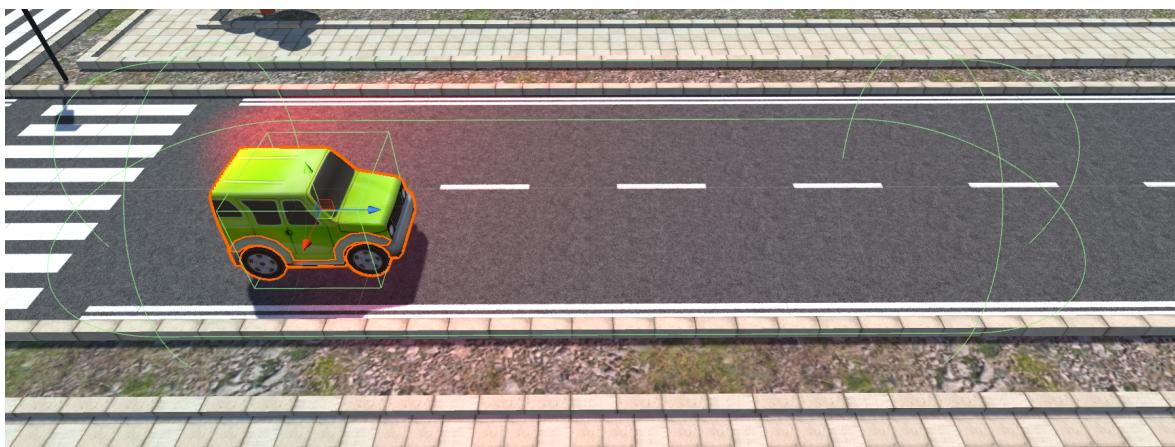


Abbildung 3.4: Collider an Fahrzeugen

### 3.3 Ampelsteuerung

Die zentrale Komponente der Ampelsteuerung ist das RemoteObject. Wie bereits erwähnt handelt es sich dabei um eine eigenständige Komponente, welche in Form einer DLL in diverse Projekte leicht eingebunden und zentral gewartet werden kann. Das RemoteObject repräsentiert die Datenstrukturen und Funktionalitäten, welche zwischen Client und Server mit Hilfe der TCP-Channel Technologie geteilt werden.

Das RemoteObject bietet die Möglichkeit eine beliebige Anzahl von Kreuzungen zu erstellen und zu verwalten. Die Logik für den Aufbau der IPC-Verbindung wird dabei direkt im Client oder Server hinterlegt. Mit Hilfe von definierte Interfaces hat der Client die Möglichkeit Kreuzungen mit drei oder vier Ampeln am Server-Skeleton anzulegen. Die Kreuzungen werden entweder mit einer festgelegten Standardkonfiguration oder mit den vom Entwickler angegeben Übergabeparametern angelegt.

Von bereits angelegten Kreuzungen können die Zykluszeiten der einzelnen Ampeln editiert werden. Weiters ist es möglich das RemoteObject zu reseten und somit den Initialzustand wiederherzustellen.

Die einzelnen Kreuzungen, welche am RemoteObject angelegt sind können separat gestartet werden. Beim Starten einer Kreuzung wird ein eigener Thread für jede Ampel der Kreuzung erzeugt in dem eine State machine für die Ampelschaltung abläuft. Dieser Thread läuft bis entweder die Serveranwendung terminiert wird oder die Kreuzung durch einen Reset vom Client gelöscht wird. Die State machine einer Ampel durchläuft die üblichen Stati und berücksichtigt dabei die Konfiguration der entsprechenden Ampel.

Mit Hilfe der ID einer Kreuzung und der Bezeichnung einer konkreten Ampel der Kreuzung kann der Status einer Ampel abgefragt werden.

Durch die Verwendung der TCP-Channel Technologie und der Definition des RemoteObjects in Form einer eigenen DLL wurde die Komplexität einer IPC sehr gut abstrahiert. Leider erfuhren wir während den Tests, dass die von uns verwendete Technologie sehr anfällig gegenüber schlechten Latenzzeiten ist. Dies wirkt sich in Form von Verbindungsproblemen zwischen Client und Server aus. Daher würde ich die Technologie TCP-Channel für IPC nur innerhalb eines lokalen Netzwerkes empfehlen.

### 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

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.

## 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

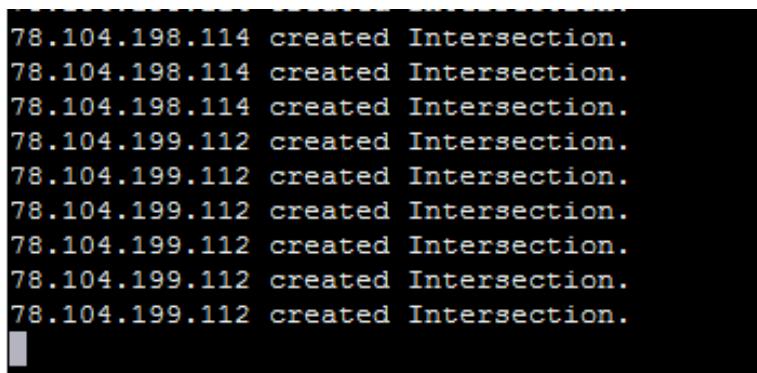
### 4.5 Ampelsteuerung

Die Ampelsteuerung wurde als eigene Applikation entwickelt. Die Ampelsteuerung kann grundsätzlich unabhängig von der Unity-Spielwelt laufen, auch die Ampeln schalten dementsprechend autonom, auch wenn die Simulation schon beendet sein sollte.

... evtl. mike wos

Die Ampelsteuerung ist auch mit Mono lauffähig. Mono ist eine C#-Implementierung für unixoide Betriebssysteme. So läuft die Ampelsteuerung auch mit Linux. Auch eine Kompilierung ist möglich, mit `xbuild solution.sln` wird eine ausführbare Datei erzeugt, die sowohl mit Linux als auch Windows lauffähig ist.

In Abbildung 4.1 ist die Ausgabe der Ampelsteuerung zu sehen. Da die Ampelsteuerung gut funktionierte, wurde in die Ausgabe weniger Zeit investiert, sodass hier nur die aufrufende IP ausgegeben wird.



```
78.104.198.114 created Intersection.  
78.104.198.114 created Intersection.  
78.104.198.114 created Intersection.  
78.104.199.112 created Intersection.
```

Abbildung 4.1: Ausgabe Ampelserver

## 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;
}
```

---

Listing 4.1: Laden des Hindernisses

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 "MouseDown" Event 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;
    }
}
```

---

Listing 4.2: Erstellen des Hindernisses

Zum Löschen eines Hindernisses muss die rechte Maustaste in Kombination mit dem vor-

---

her 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);
        }
    }
}
```

---

Listing 4.3: Zerstören des Hindernisses



Abbildung 4.2: Hindernis

## 4.7 Aus- und Einfahren von Fahrzeugen anderer Gruppen

Mit dem Plugin ‘Unity3D.Amqp’ (<https://github.com/CymaticLabs/Unity3D.Amqp>) für Unity ist es möglich einen RabbitMQ-Server direkt in Unity einzubinden.

Die Konfiguration der Serverdaten werden dabei direkt in den Menüeinstellungen von Unity vorgenommen. Die verwendete Konfiguration ist in Abbildung 4.3 zu sehen.

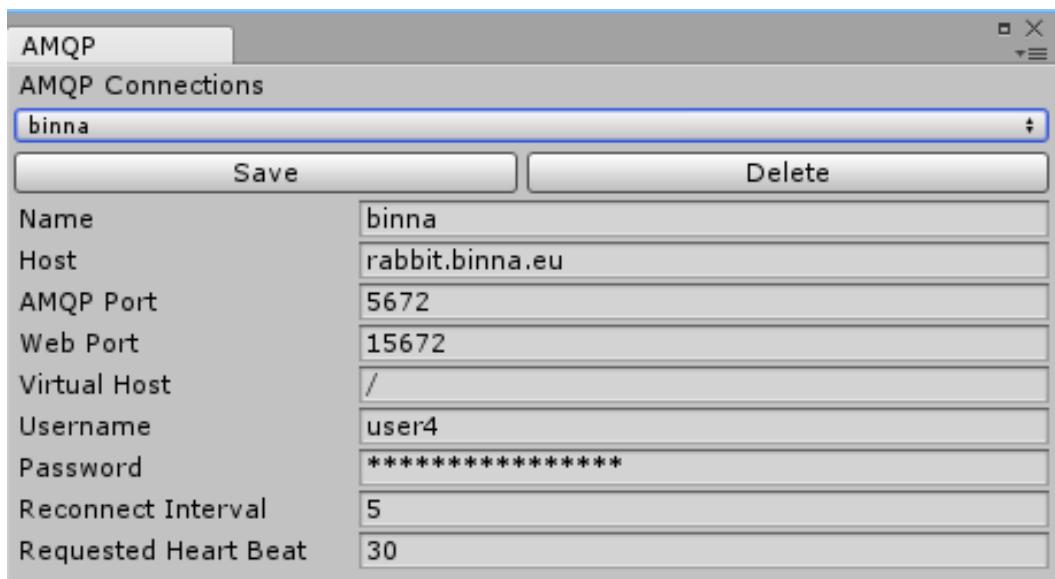


Abbildung 4.3: Einstellungen RabbitMQ

Jede Gruppe bekam einen eigenen Benutzer samt Passwort. Damit man Nachrichten empfangen kann, muss man sich auf eine ‘Queue’ subscriben.

Das Plugin stellt anschließend Methoden zur Verfügung. Eine solche Methode ist `OnMessageReceived()`. Damit lässt sich eine eingehende Nachricht an ein Objekt in der Spielwelt weitergeben. Das Objekt kann daraufhin reagieren, beispielsweise ein Auto erzeugen.

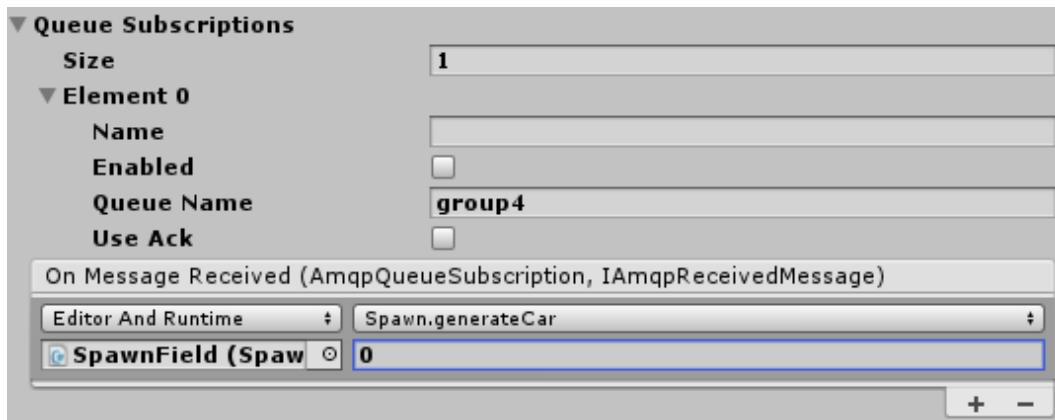


Abbildung 4.4: RabbitMQ Queue

In Abbildung 4.4 sieht man wie eine eingehende Nachricht beim Objekt `SpawnField` eine Methode aufruft.

## Versionshistorie

<b>Version</b>	<b>Datum</b>	<b>Autor(en)</b>	<b>Änderungen</b>
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
0.3	08.06.2017	AR	Hinzufügen der Messaging Komponente

Autoren:

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