UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR MULTIMEDIALE
UND INTERAKTIVE SYSTEME

Direktor: Prof. Dr. rer. nat. Michael Herczeg

Ausbau der Software BeamNG zu Fahrsimulationsumgebung für Untersuchungen in der Nutzer-Energie-Interaktion

Development of the software BeamNG to a driving simulation environment for researches on the field of user-energy-interaction

Bachelorarbeit

im Rahmen des Studiengangs
Medieninformatik
der Universität zu Lübeck

vorgelegt von:

Pascal Stieglitz

ausgegeben und betreut von:

Univ.-Prof. Dr. rer. nat. Thomas Franke, Dipl.-Psych.

mit Unterstützung von:

Vivien Moll, M.Sc.

Lübeck, 14. September 2018

Kurzfassung

Die vorliegende Qualifikationsarbeit beschreibt den Ausbau der Software BeamNG.research zu einer Fahrsimulationsumgebung für Untersuchungen in der Nutzer-Energie-Interaktion. Zu Beginn wurde identifiziert, dass in den Bereichen Datenexport, energetische Simulation, Fahrzeug, experimentelle Kontrolle, UI-Mods und Strecke Anforderungen existierten, die nicht standardmäßig von der Software erfüllt werden. Alle gefundenen Anforderungen wurden in einem Anforderungsdokument gesammelt, welches im Laufe der Arbeit noch erweitert wurde. Es wurde eine Anleitung erstellt, die die Nutzung von BeamNG.research in allen zuvor genannten Bereichen erleichtern sollte. Im Bereich Datenexport wurde in der Anleitung die Nutzung einer für diesen Zweck bereitgestellten Mod erläutert. Dazu mussten die sogenannten Lua-Namen, unter denen die Parameter in BeamNG.research geloggt werden können, identifiziert werden. Außerdem wurde die Mod durch Kommentare bezüglich der Konfigurierbarkeit vereinfacht. Für die energetische Simulation wurde ein Live-Datenaustausch mit Matlab mittels File-based Interprocess Communication realisiert, damit diese zwecks hoher Präzision extern ablaufen kann. Des Weiteren wurde eine Möglichkeit gefunden, die Daten aus Matlab auch in BeamNG.research umzusetzen. Darüber hinaus wurde ein Fahrzeug mit elektrischem Antrieb integriert und die Kamera für ein realistisches Fahrgefühl angepasst. Auf Basis eines vorgegebenen Ablaufs eines potentiellen ersten Experiments wurde untersucht, wie diverse Anforderungen im Bereich der experimentellen Kontrolle gelöst werden konnten. Dazu wurden verschiedene, aufeinander abgestimmte Mods geschrieben, teilweise vorhandene Originaldateien modifiziert und die vorhandene Anleitung zur Erstellung von Szenarios um nützliche Hinweise ergänzt. In den Bereichen UI-Mods und Strecke wurden lediglich einige Hinweise zu UI-Mods in der Anleitung vermerkt, nachdem festgestellt wurde, dass UI-Mods bereits nahezu alle Anforderungen erfüllten. Abschließend wurde ein geplantes Experiment umgesetzt, um zu validieren, dass die wichtigsten Anforderungen sämtlicher Bereiche erfüllt wurden.

Im Rahmen dieser Arbeit wurde mit BeamNG.drive gearbeitet, sämtliche Ergebnisse sind jedoch auf BeamNG.research übertragbar. Während es sich bei BeamNG.drive um ein Simulationsspiel handelt, stellt BeamNG.research eine Forschungsvariante dar. Daher wird durch die Arbeit hinweg von BeamNG.research gesprochen.

Schlüsselwörter

BeamNG.research, Fahrsimulationssoftware, Nutzer-Energie-Interaktion

Abstract

This thesis describes the development of the software BeamNG.research to a driving-simulation environment for researches on the field of the user-energy-interaction. Initially, it was identified that there were requirements that potentially were not fulfilled by default. These requirements were found in the areas of exporting data, energy simulation, vehicle, experimental control, UI-mods and tracks. All identified requirements were collected in a requirements document which was further extended in the course of the work. A manual was created to facilitate the use of BeamNG.research across all the areas identified. Regarding the data export, instructions have been added to the manual for using a mod provided for this purpose. Therefore it was necessary to identify the corresponding names of the parameters. Furthermore the mod was simplified regarding the configuration of the mod using comments. In the scope of energy simulation a live data exchange between BeamNG.research and Matlab was realised via File-based Interprocess Communication in order to externalise the simulation for the purpose of a high precision. In addition, a way was to be found to actually use the data from Matlab in BeamNG.research. Furthermore an electrical vehicle was integrated in BeamNG.research and the camera was optimized for a realistic driving experience. On the basis of a predetermined sequence of a potential first experiment, it was investigated how various requirements of experimental control could be solved. For this purpose, several coordinated Mods were written and even some partially existing original files were modified. Furthermore, the manual on creating scenarios was extended by helpful hints. In the areas of UI-mods and driving route, only a few notes on UI-mods were mentioned in the manual, after it was determined that UI-mods already fulfilled nearly all requirements. Finally, a real experiment was realized using the results of this thesis to validate that all of the most important requirements were met.

During this thesis BeamNG.drive was used but all results are transferable to BeamNG.research. While BeamNG.drive is a simulation game, BeamNG.research represents the research variant. Therefore this document always refers to BeamNG.research.

Keywords

BeamNG.research, driving-simulation software, user-energy-interaction

Inhaltsverzeichnis

1 Einleitung

Die Elektrifizierung der Antriebstechnologie von Fahrzeugen ist ein wichtiger Schritt, der aktuell angegangen wird, um dem Klimawandel entgegen zu wirken (McCollum, Krey, Kolp, Nagai & Riahi, 2014). Doch die Reichweite eines Elektrofahrzeugs schreckt viele Menschen davon ab, eines zu kaufen. Neben der Technischen Reichweite eines Elektrofahrzeugs, muss auch die Psychologische Reichweite betrachtet werden (Franke, Neumann, Bühler, Cocron & Krems, 2012). Eine Veränderung des Fahrverhaltens eines Nutzers kann die Energieeffizienz beim Fahren steigern. Bessere Anzeigesysteme können also die Reichweite eines Fahrzeugs erhöhen, ohne die Antriebstechnik zu verändern. Solche effizienteren Anzeigesysteme, die die Reichweite erhöhen können, sollen am Institut für Multimediale und Interaktive Systeme (IMIS) im Rahmen von Forschungsschwerpunkten (z.B. Ecodriving), -projekten und Qualifikationsarbeiten entwickelt werden.

Für die Evaluation der entwickelten Anzeigesysteme kann ein Fahrsimulator einen erheblichen Mehrwert liefern, da Evaluationen eines Anzeigesystems während einer echten Fahrt aufwändiger und potentiell gefährlich sind (z.B. wenn getestet werden soll, inwieweit ein Anzeigesystem ablenkend ist) und eine Evaluation mit Fahrsimulator über bloße Befragungsstudien im Labor hinaus gehen und eine Erhöhung der Validität bieten. Deshalb wird am IMIS aktuell ein Fahrsimulator aufgebaut. An diesem sollen zukünftig verschiedene Untersuchungen in der Nutzer-Energie-Interaktion durchgeführt werden. Forschung auf dem Feld des Ecodriving wird bereits seit Jahren mit Fahrsimulatoren getätigt, z.B. von Hiraoka, Terakado, Matsumoto und Yamabe (2009). Hierbei lag der Fokus bisher darauf, wie möglichst energieeffizient gefahren werden kann. An anderen Universitäten werden teilweise Versuche mit einem Fahrsimulator geplant, wie sie so ähnlich auch am IMIS stattfinden könnten, z.B. entwarf Degirmenci 2018 eine gamifizierte App zum verbessern des Beschleunigungs- und Bremsverhaltens bezüglich des Energieverbrauchs, welche er im nächsten Schritt mit einem Fahrsimulator zu evaluieren plante.

In diesem Kapitel wird zunächst im Abschnitt 1.1 Ziele der Arbeit beschrieben, was die Ziele dieser Arbeit sind. Anschließend wird im Abschnitt 1.2 Stand der Technik der Stand der Technik

zu Beginn der Arbeit erläutert. Der Abschnitt ist wiederum in die Unterabschnitte 1.2.1 Fahrsimulator am IMIS, 1.2.2 Andere Fahrsimulatoren und 1.2.3 BeamNG.research und BeamNG.drive unterteilt. Zum Abschluss des Kapitels wird in Abschnitt 1.3 Vorgehensweise die Vorgehensweise der Software-Anpassungen dargestellt.

1.1 Ziele der Arbeit

Ziel der geplanten Experimente mit dem Fahrsimulator ist es, Veränderungen im Fahrverhalten (z.B. Beschleunigungsstärke) bzw. in daraus resultierenden Effizienzindikatoren (Verbrauch, Reichweite, Akkuladestand, etc.) zu untersuchen. Hierfür soll eine Software mit möglichst realistischer Fahrumgebung verwendet werden, um die Validität der Ergebnisse zu erhöhen. Die Computerspielindustrie produziert heutzutage Produkte mit hoher Realitätsnähe, insbesondere im graphischen Bereich. BeamNG.research ist eine Variation eines solchen Computerspiels, das im Gegensatz zur Spiele-Version BeamNG.drive für die Forschung gedacht ist (siehe Unterabschnitt 1.2.3 BeamNG.research und BeamNG.drive). In dieser Arbeit soll untersucht werden, ob BeamNG.research für Untersuchungen in der Nutzer-Energie-Interaktion genutzt werden kann.

Damit eine Software als Fahrsimulationssoftware im wissenschaftlichen Kontext genutzt werden kann, müssen verschiedene essentielle Funktionen vorhanden sein. In den folgenden Absätzen wird zwecks Darstellung der Ziele dieser Arbeit kurz auf die wichtigsten eingegangen. Eine ausführliche Analyse darüber, welche Funktionalitäten nötig und erstrebenswert sind, folgt in Kapitel 2 Abschnitt 2.3 Anforderungsanalyse.

Daten loggen: Für wissenschaftliche Untersuchungen ist es unerlässlich, dass verschiedene Daten wie z.B. die Geschwindigkeit geloggt und so anschließend in der Auswertung einbezogen werden können. Es muss also im Rahmen dieser Arbeit eine Möglichkeit gefunden werden, Daten in BeamNG.research zu loggen.

Daten austauschen: Es kann nötig sein, Daten in BeamNG.research während der Laufzeit zu manipulieren. Dies sollte sowohl per Hand als auch mittels externer Software möglich sein. Zu diesem Zweck soll im Rahmen dieser Arbeit eine Möglichkeit gefunden werden, Daten, die außerhalb von BeamNG.research erzeugt wurden, in BeamNG.research zu verwenden.

Fahrzeug: Es sollen alle wichtigen Anforderungen an ein Fahrzeug erfüllt werden, die in Kapitel 2 Unterabschnitt 2.3.3 Fahrzeug ermittelt werden. Insbesondere ist ein realistisches Elektrofahrzeug mit realistischem Energiemodell wichtig.

Automatischer Ablauf: Ein Experiment muss ohne Eingriffe des Versuchsleiters ablaufen können. Dazu ist es erforderlich, den Ablauf eines Experiments von Anfang bis Ende festlegen zu können.

UI: Anzeigesysteme müssen in BeamNG.research integriert werden, damit diese evaluiert werden können. Deshalb soll das Integrieren von neuen Anzeigesystemen vereinfacht werden, sodass dies ohne besondere Fachkenntnisse mit BeamNG.research (oder den zugrunde liegenden Sprachen) möglich ist.

Beim Erfüllen der aufgezählten Anforderungen ist ein besondere Fokus darauf zu legen, dass die Nutzer des Fahrsimulators nicht unbedingt Informatiker sind (siehe Kapitel 2 Abschnitt 2.4 Benutzeranalyse) und entsprechende Hilfestellungen bei technischen Schritten benötigen. Ziel ist es, möglichst viele Anforderungen zu realisieren, auch wenn dadurch vereinzelt bestimmte Anforderungen aus Zeitgründen nur prototypisch umgesetzt werden können.

1.2 Stand der Technik

Verschiedene Aspekte des Stands der Technik werden in diesem Abschnitt betrachtet. Zunächst wird in Unterabschnitt 1.2.1 Fahrsimulator am IMIS die Hardware, die im Fahrsimulator des IMIS Verwendung findet, genannt. In Unterabschnitt 1.2.2 Andere Fahrsimulatoren wird der Stand der Technik im Bereich Fahrsimulatoren dargestellt, wobei sich im Rahmen dieser Arbeit auf Fahrsimulationssoftware konzentriert wird. Abschließend werden in Unterabschnitt 1.2.3 BeamNG.research und BeamNG.drive die Softwares BeamNG.research und BeamNG.drive vorgestellt.

1.2.1 Fahrsimulator am IMIS

Der Fahrsimulator am IMIS befand sich zu Beginn dieser Arbeit noch im Aufbau. Die Beschaffung und der Aufbau der Hardware sind jedoch nicht Teil dieser Qualifikationsarbeit. Zum

Zweck einer besseren Reproduzierbarkeit der Versuche an dem Fahrsimulator folgt nun eine Auflistung der wichtigsten Anforderungen an die Hardware und der verwendeten Hardware.

Die Hardware soll das Cockpit eines Fahrzeugs simulieren. Dabei ist zu beachten, dass der Ein- und Ausstieg problemlos möglich ist. Außerdem soll das Field of View (FOV) ausreichend groß sein. Die dazu verwendeten Bildschirme müssen außerdem einen möglichst geringen Input Lag aufweisen.

Als Kern des Fahrsimulators diente zunächst ein Gaming Notebook, das MSI GT73EVR 7RE-838-DE Titan. Dieses verfügte über einen Intel Core i7-7700HQ Prozessor mit einer Taktgeschwindigkeit von bis zu 3.80 GHz. Außerdem war eine Nvidia GeForce GTX 1070 verbaut. Es verfügte über ein 17.3 Zoll großes Display, hatte 16 GB Arbeitsspeicher und 256 GB SSD und 1 TB HDD Speicherplatz. Als Betriebssystem wurde Windows 10 verwendet.

Neben dem Notebook wurde auch Hardware für mehr Realitätsnähe im Fahrgefühl verwendet. Zu diesem Zweck wurden die ClubSport Wheel Base V2.5, das ClubSport Lenkrad Classic und die ClubSport Pedale V3 invertiert, alle drei von Fanatec, angeschafft.

Gegen Ende der Bearbeitungszeit dieser Arbeit wurde weitere Hardware angeschafft, wobei der zuvor verwendete Laptop ersetzt wurde. Es wurde ein Rechner mit den folgenden Komponenten angeschafft: Mainboard: Asus TUF Z370-PRO GAMING S1151V2; Prozessor: Intel Core I7-8700K mit 3.70GHz und 12MB Cache; Arbeitsspeicher: 64GB aufgeteilt auf 4x 16GB DDR4 mit 2400MHz von der Marke Crucial; Grafikkarten: 2x MSI GEFORCE GTX 1080 TI mit jeweils 11GB GDDR5X, Festplatten: eine Samsung 860 EVO Basic 1TB SSD und eine Western Digital 2TB Purple 64MB SATA 6Gb/s. Außerdem wurden ein FK Gamesitz Silverstone Rennsimulator für Rennspiele schwarz, ein RS1 Frame Only Black der Marke RSEAT mit dem Product Code RS1FOB, ein Playseat TV Stand Pro – Basis und eine Playseat TV Stand Pro 3S – Erweiterung angeschafft. Als Bildschirme wurden 3 Samsung NU8000 (entspricht dem deutschen Modell NU8009) mit 55 Zoll und 4k Ultra HD Auflösung angeschafft.

1.2.2 Andere Fahrsimulatoren

Die Entscheidung, dass BeamNG.research am Fahrsimulator des IMIS verwendet werden soll, war vor Beginn dieser Arbeit bereits getroffen. Thema dieser Arbeit war ausschließlich die Software BeamNG.research (siehe Unterabschnitt 1.2.3 BeamNG.research und BeamNG.drive).

Trotzdem wird in diesem Unterabschnitt ein Auszug existierender Fahrsimulatoren und Softwares für Fahrsimulatoren vorgestellt. Zu den nachfolgend vorgestellten Fahrsimulatoren und Fahrsimulationssoftwares existierten in einem Dokument Notizen zu Vor- und Nachteilen, welche zur Wahl einer geeigneten Software herangezogen wurden. Die wichtigsten werden nachfolgend kurz erwähnt.

Carnetsoft Driving Simulator: Der Carnetsoft Driving Simulator (Driving simulator for research and training |, n.d.) ist eine Fahrsimulationssoftware, welche sowohl für die Forschung als auch für Fahrtraining entwickelt wurde und mit einer Empfehlung der zu nutzenden Hardware ausgeliefert wird. Die Software kostet 2500€ und ist darauf ausgelegt, psychologische Effekte beim Fahren untersuchen zu können. Laut des zuvor erwähnten Dokuments ist das Auslesen von Daten mit dieser Software nur mit einigem Aufwand möglich. Der Carnetsoft Driving Simulator wurde für verschiedene wissenschaftliche Untersuchungen verwendet, z.B. von Bock et al. (2018).

SimuAssist: SimuAssist Fahrschulsimulatoren (FAHRSCHUL SIMULATOREN, n.d.) ist eine Firma, die verschiedene (selbstproduzierte) Fahrsimulatoren zu Fahrtrainingszwecken verkauft. Die Simulatoren sind entsprechend den aktuellen EU-Richtlinien konzipiert worden. Laut des Dokuments ist die Auslesbarkeit von Daten ungewiss.

FOERST Fahrsimulator: FOERST Fahrsimulatoren (FOERST Fahrsimulatoren, n.d.) ist eine Firma, die sich auf den Verkauf und die Vermietung selbstproduzierter Fahrsimulatoren für die Kraftfahrerausbildung spezialisiert hat. Ob Daten ausgelesen werden können, ist laut des Dokuments unklar. Fahrsimulatoren von FOERST Fahrsimulator wurden bereits in wissenschaftlichen Untersuchungen verwendet, z.B. von Pavlou et al. (2015).

Vicom Editor: Der Vicom Editor (VicomEditor, n.d.) wurde vom TÜV ursprünglich entwickelt, um Bilder von Fahrszenen für die deutsche theoretische Fahrprüfung zu erstellen. Mittlerweile wird er jedoch auch für Fahrtraining verwendet. Der Vicom Editor wird neben der Software BeamNG.research auf dem Fahrsimulator des IMIS verwendet werden, war jedoch nicht Thema dieser Arbeit und wird daher nicht weiter betrachtet.

City Car Driving: City Car Driving (City Car Driving, n.d.) ist ein Computerspiel mit einem gewissen Simulationsanspruch. Die Physik ist allerdings laut des Dokuments relativ schlecht. Außerdem ist das Forum schlechter ausgebaut als das von BeamNG.drive. City Car Driving kostet $24.99.

1.2.3 BeamNG.research und BeamNG.drive

Die Software BeamNG.drive wurde von der BeamNG GmbH entwickelt. Die BeamNG GmbH ist ein 2013 gegründetes internationales indie (kurz für independent) Gaming Studio mit mehr als 30 Mitarbeitern weltweit (About Us – BeamNG, n.d.). BeamNG.drive wurde am 29.05.2015 auf Steam Greenlight (Steam Greenlight, n.d.) im Early Access veröffentlicht und kostet 16,99€ (Stand: 20.05.2018). Es wird von den Entwicklern als „dynamic soft-body physics vehicle simulator capable of doing just about anything“ (BeamNG.drive | BeamNG, n.d.) beschrieben.

Außer BeamNG.drive wurde von der BeamNG GmbH auch die Software BeamNG.research veröffentlicht. Diese ist eine Abwandlung von BeamNG.drive, die auf eine Nutzung in einem wissenschaftlichen Kontext ausgelegt ist (BeamNG.research – BeamNG, n.d.). BeamNG.research wurde erst im Verlauf der vorliegenden Arbeit veröffentlicht. Die erste Version wurde dahingehend untersucht, ob sie sich für den Fahrsimulator am IMIS besser eignet als BeamNG.drive und ob zu dem Zeitpunkt bereits an BeamNG.drive vorgenommene Modifikationen auf BeamNG.research übertragen werden können.

Vorteile der Software BeamNG.research sind die gute Grafik, die leichte Modifizierbarkeit und die hilfsbereite Community für das Spiel BeamNG.drive. Die detaillierte und hochauflösende Grafik (vgl. Abbildung 1) kann eine gute Immersion ermöglichen (Wilcox-Netepczuk, 2013). Mittels Modrepository lassen sich in BeamNG.research schnell und einfach Mods integrieren und aktivieren. Diese, in der Skriptsprache Lua geschriebenen, Mods können selbst erstellt werden. Alternativ kann auch eine in der Community verfügbare Mod verwendet werden, selbst wenn die Mod eigentlich für BeamNG.drive entwickelt wurde. Im BeamNG.drive Forum (BeamNG, n.d.) existiert ein ganzes Unterforum für verschiedene Mods und Mod Support.

Abbildung 1: Fahrszene in BeamNG.research

Da der Fokus der Software nicht auf der Testung von UIs sondern der Physiksimulation und dem autonomen Fahren liegt, ergeben sich einige Herausforderungen, um BeamNG.research als Fahrsimulationssoftware für Untersuchungen in der Nutzer-Energie-Interaktion nutzen zu können. Die verschiedenen Probleme werden in Abschnitt 1.1 Ziele der Arbeit bzw. Kapitel 2 Abschnitte 2.2 Erste Kurzanalyse und 2.7 Analyse der Möglichkeiten in BeamNG.research genannt.

Im Laufe dieser Arbeit wurden zwei Telefonkonferenzen mit dem Geschäftsführer der BeamNG GmbH und weiteren Mitarbeitern geführt. Bei einer dieser Telefonkonferenzen wurde u.a. erläutert, dass Mods auf BeamNG.research nur auf einer sogenannten „unlimited Version“ funktionieren, diese wurde kurz vor Ende dieser Arbeit zur Verfügung gestellt. Deshalb wurde im Rahmen dieser Arbeit zunächst mit BeamNG.drive gearbeitet. Später wurden sämtliche Ergebnisse auf BeamNG.research unlimited übertragen. Wie bereits erwähnt wird in dem vorliegenden Dokument mit einigen, jeweils kurz begründeten, Ausnahmen von BeamNG.research gesprochen, weil es die Forschungsvariante ist.

1.3 Vorgehensweise

Die Anpassungen von BeamNG.research wurden nach einem Prozess entwickelt, der an das User Centered Design (UCD, nach Norman & Draper, 1986) angelehnt wurde. Dabei wurde je-

doch ein zusätzlicher Bearbeitungsschritt zwischen der Konzeption und der Evaluation einer Lösung eingebaut, in dem die technische Umsetzbarkeit in BeamNG.research validiert wurde. Konnte diese Analyse die Umsetzbarkeit nicht bestätigen, wurde durch einen Rücksprung auf den vorherigen Bearbeitungsschritt eine neue Lösung basierend auf den neuen Erkenntnissen konzipiert. Außerdem wurde entgegen des klassischen UCD in jeder Iteration nur ein Bereich betrachtet. Nach der Planung des Prozesses und einer Kontextanalyse wurden ab der Anforderungsanalyse mehrere Iterationen für die verschiedenen Bereiche durchlaufen. Dies ermöglichte kürzere Iterationen, bei denen Analyse und Konzeption (und Realisierung) eines Bereiches näher beieinander lagen. Die dafür benötigte Feedbackfrequenz wurde durch wöchentliche Treffen mit zukünftigen Benutzern (konkret dem Betreuer dieser Arbeit, einer Doktorandin, die den Fahrsimulator für ihre eigene Forschung zu nutzen plante, und ein bis zwei für den Aufbau des Fahrsimulators zuständige wissenschaftliche Hilfskräften) ermöglicht. Ein Rückschritt zur Kontextanalyse war wie im klassischen Modell möglich, falls die Evaluationsergebnisse dies erforderten. Abbildung 2 zeigt das nach DIN EN ISO 9241-210 (International Organization for Standardization [ISO], 2010) visualisierte Modell mit den Abwandlungen in rot.

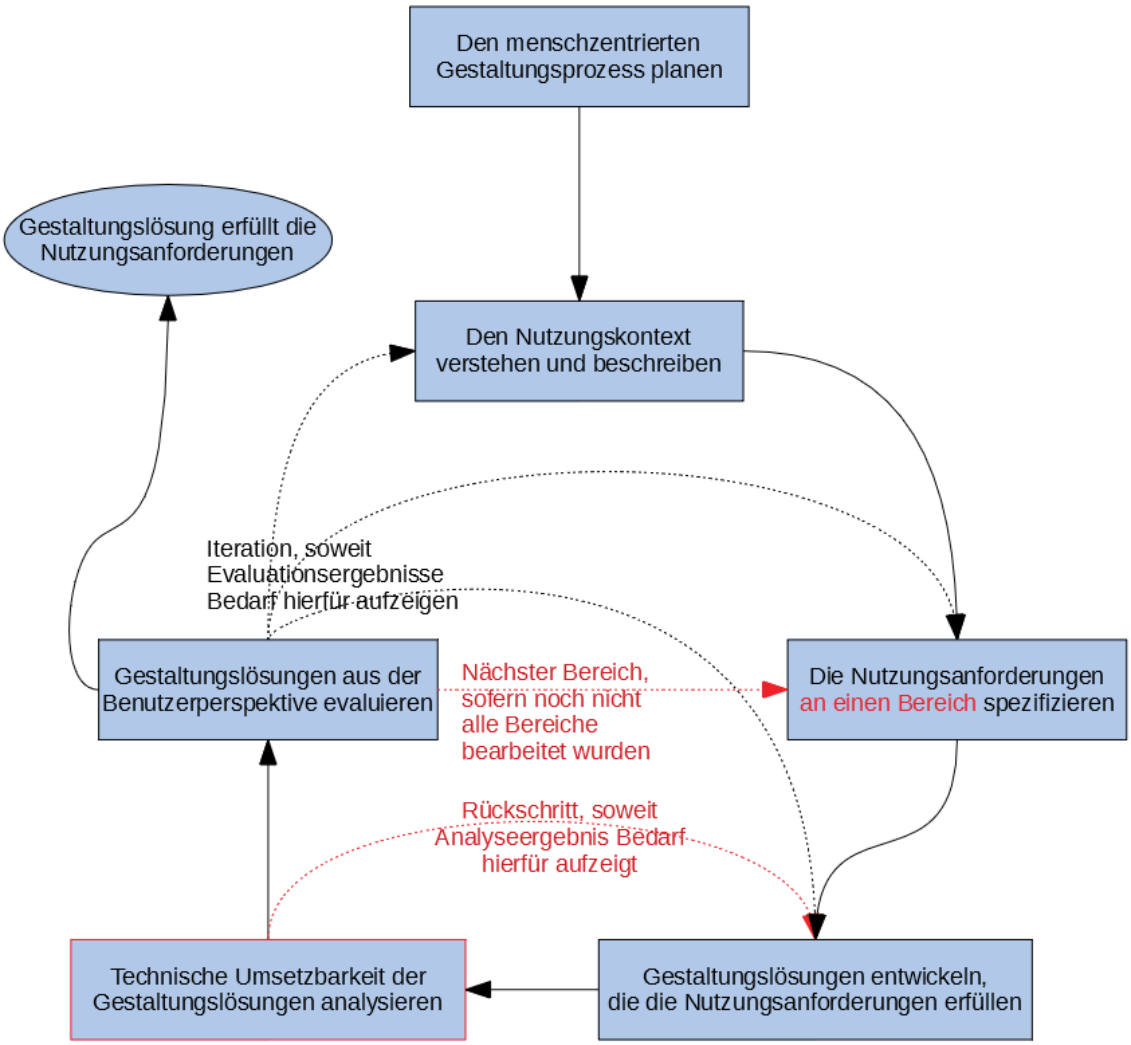

Abbildung 2: Abgewandeltes UCD nach DIN EN ISO 9241-210, Abwandlungen in rot

In Kapitel 2 werden verschiedene Analysen aufgeführt. Diese umfassen eine Erste Kurzanalyse, die im Rahmen der Erstellung des Exposé durchgeführt wurde. Weiterhin wurde für jeden der sechs identifizierten Bereiche Datenexport, energetische Simulation, Fahrzeug, experimentelle Kontrolle, UI-Mods und Strecke eine Anforderungsanalyse erstellt. Darüber hinaus wurden eine Benutzeranalyse, eine Kontextanalyse, eine Organisationsanalyse und für jeden der erwähnten sechs Bereiche eine Analyse der Möglichkeiten in BeamNG.research erstellt. Die verwendeten Analyseverfahren werden ebenfalls in diesem Kapitel beschrieben. Kapitel 3 beschreibt die Konzeption von Lösungen für die zuvor identifizierten Anforderungen, wobei die Abschnitte den zu betrachtenden Bereichen entsprechen, mit der Ausnahme der Strecke, der kein Abschnitt gewidmet wurde, da diese nicht im Rahmen der vorliegenden Arbeit umgesetzt wurde (siehe Kapitel 2 Unterabschnitt 2.3.6 Strecke). In Kapitel 4 wird die Realisierung der konzipierten Lösungen beschrieben, ebenfalls sortiert nach den verbleibenden fünf Bereichen. Im

5. Kapitel wird eine abschließende Evaluation durchgeführt, um zu validieren, dass die wichtigsten Anforderungen umgesetzt wurden. Abschließend werden in Kapitel 6 diese Arbeit zusammengefasst, offene Punkte erläutert und ein Ausblick möglicher weiterer Schritte gegeben.

2 Analyse

Wie bereits in Kapitel 1 Abschnitt 1.1 Ziele der Arbeit erwähnt, musste zunächst ermittelt werden, welche Anforderungen an eine Fahrsimulationssoftware bestanden. Zwar wurde zum Zweck der Erstellung des Exposés zu dieser Arbeit bereits eine erste kurze Analyse durchgeführt (siehe Abschnitt 2.2 Erste Kurzanalyse), diese war jedoch nicht ausreichend, um diese gesamte Arbeit darauf zu stützen.

In diesem Kapitel werden zunächst in Abschnitt 2.1 Analyseverfahren die verwendeten Analyseverfahren beschrieben. Die Ergebnisse werden in den Abschnitt 2.3 Anforderungsanalyse, 2.4 Benutzeranalyse, 2.5 Kontextanalyse und 2.6 Organisationsanalyse dargestellt. Abschließend werden in Abschnitt 2.7 Analyse der Möglichkeiten in BeamNG.research die Möglichkeiten in BeamNG.research erläutert.

2.1 Analyseverfahren

Die Informationen, auf die sich die Abschnitte 2.3 Anforderungsanalyse, 2.4 Benutzeranalyse, 2.5 Kontextanalyse und 2.6 Organisationsanalyse beziehen, wurden mittels verschiedener Analyseverfahren gewonnen. In diesem Abschnitt werden diese Verfahren erläutert.

Persönliche Gespräche: In diversen Gesprächen mit verschiedenen Personen (Kleingruppen von maximal fünf Personen) wurden einige für die Analysen wichtige Informationen ermittelt. Bei den Personen handelte es sich um den Betreuer dieser Arbeit, eine Doktorandin, die den Fahrsimulator für ihre eigene Forschung zu nutzen plante, und zwei wissenschaftliche Hilfskräfte, die für den Aufbau des Fahrsimulators zuständig waren. Ein großer Teil der Gespräche fand im Rahmen eines wöchentlichen Meetings statt. Zu diesen Gesprächen existieren keine Protokolle, auf die verwiesen werden könnte. Viele Ergebnisse, die durch andere Analyseverfahren erzielt wurden, begannen mit in einem Gespräch genannten Informationen. Falls die Ergebnisse dieses zusätzlichen Verfahrens protokolliert wurden, wurde auf dieses statt auf das

Gespräch verwiesen. Weitere Unklarheiten und offen gebliebene Fragen wurden teilweise mittels E-Mail geklärt.

Interviews: Es wurden Interviews mit zwei Teilnehmern durchgeführt. Die Interview-Teilnehmer waren beide Doktoranden, die für ihre Forschung den Fahrsimulator zu nutzen planten. Da sich die Transkripte nicht ausreichend stark anonymisieren lassen ohne die wichtigen Informationen zu zensieren, wurden die Interviewtranskripte nicht dem Anhang beigefügt. Es wird in diesem Dokument trotzdem auf Interviewteilnehmer 1 und Interviewteilnehmer 2 verwiesen.

Vorhandene Dokumente: Vor Beginn der Kurzanalyse (siehe Abschnitt 2.2 Erste Kurzanalyse) und damit der Tätigkeiten im Rahmen dieser Arbeit, wurden bereits Dokumente mit verschiedenen Anforderungen oder Ähnlichem von verschiedenen Personen erstellt. Diese wurden in die Analysen miteinbezogen. Meist wurden bereits existente Dokumente jedoch nur als Grundlage für ein anderes Analyseverfahren genutzt, da sie teilweise nicht detailliert genug ausgearbeitet oder noch nicht eindeutig verständlich waren.

Iterativ ausgearbeitete Dokumente: Durch das iterative Ausarbeiten von Dokumenten mit ständigem Feedback durch verschiedene Personen, meist den Betreuer dieser Arbeit oder eine Doktorandin, wurden Informationen bis auf eine feine Detailstufe herausgearbeitet. Die finalen Versionen von so entstandenen Text- und Tabellendokumenten, die in den Analysen referenziert wurden, wurden dem Anhang beigefügt.

Eines dieser iterativ ausgearbeiteten Dokumente ist besonders relevant und wird deshalb hier kurz erläutert. Es wurde ein Anforderungsdokument nach Kapitel 7.3 und 9.2-9.3 des Buchs *100 Minuten für Anforderungsmanagement* von Grande (2014) erstellt. Dabei wurde die Beschreibung als „Anforderung“ und der Status als „aktueller Stand“ bezeichnet. Von den von Grande aufgezählten möglichen sonstigen Attributen wurden „Priorität“, „Quelle“, „Ziel“ und „Typ“ (als Kurzform von „Anforderungstyp“) verwendet. Außerdem wurde der „Version“ ein Datum hinzugefügt, wann die Anforderung zuletzt aktualisiert wurde. Des Weiteren wurden Spalten für folgende Informationen ergänzt: Bereich der Anforderung, Relevanz der Anforderung für das erste mit dem Fahrsimulator geplante Experiment (welches zu Beginn dieser Arbeit noch nicht konkretisiert aber absehbar war), Status der standardmäßigen Erfüllung von BeamNG.research, Umsetzbarkeit der Anforderung und weitere Anmerkungen bzw. Details. Dieses Dokument sollte nach Abschluss dieser Arbeit fortgeführt werden und enthielt auch An-

forderungen, die den Rahmen dieser Arbeit überschritten hätten. Die zum Zeitpunkt der Fertigstellung dieser Arbeit aktuellste Version des Anforderungsdokuments kann im Anhang unter Anhang A: Anforderungsdokument gefunden werden.

2.2 Erste Kurzanalyse

Für die Erstellung des Exposés zu dieser Arbeit wurde zunächst eine kurze erste Analyse durchgeführt, in der anhand von bereits existenten Dokumenten und persönlichen Gesprächen die Anforderungen an eine Fahrsimulationssoftware grob geklärt wurden. Außerdem wurde identifiziert, welche dieser Anforderungen BeamNG.research standardmäßig erfüllt und welche im Rahmen dieser Arbeit nachträglich hinzugefügt werden mussten. Die Anforderungen wurden in zwei Gruppen eingeteilt, abhängig davon, wie leicht die Funktionen genutzt werden können sobald sie existieren. Dadurch sollte identifiziert werden, für welche Funktionalitäten Hilfsmittel wie z.B. Anleitungen benötigt werden. Die nachfolgend erläuterten Ergebnisse dieser ersten Analyse werden in Abbildung 3 grafisch dargestellt.

Die Anforderungen an eine Fahrsimulationssoftware wurden in vier Kategorien eingeteilt. Die erste dieser Kategorien beinhaltete Anforderungen an ein Szenario oder an die verwendete Karte. Als standardmäßig erfüllte Anforderungen in dieser Kategorie wurden verschiedene Strecken mit Kleinstadt, Landstraße, Wald und Wiese oder Kreuzungen, Kurven und gerader Strecken identifiziert. Auch Gefälle, Steigung und ebene Strecken waren erfüllte Anforderungen in dieser Kategorie. Des Weiteren wurden Bushaltestellen ohne Haltebucht und leichter Verkehr als benötigt und vorhanden identifiziert. Auf der nicht standardmäßig erfüllten Seite dieser Kategorie wurden deutsche Schilder und Texturen, Bushaltestellen mit Haltebucht, Kreisverkehre, starker Verkehr, Passanten und Radfahrer sowie funktionierende Ampeln erkannt.

In der zweiten Kategorie wurden Anforderungen an das zu verwendende Fahrzeug gesammelt. Als standardmäßig erfüllt wurden ein Auto mit Handyhalterung im Cockpit und die Möglichkeit Sondertüren (z.B. eine Bustür) zu öffnen identifiziert. Noch nicht erfüllte Anforderungen dieser Kategorie waren die Antriebstechnologie (es wurden Elektro- und Hybridautos benötigt) und ein fahrbarer Bus.

Als dritte Kategorie wurden Schnittstellen betrachtet. Zwar war ein Datenexport z.B. als .csv Datei nicht standardmäßig vorhanden, doch bereits vor der Kurzanalyse wurde von einem BeamNG Mitarbeiter eine Mod zur Verfügung gestellt, die dies ermöglichte. Daher wurde der Datenexport als vorhanden gewertet. Weiterhin war nötig und möglich, das UI im Cockpit zu ändern und das UI auf einem externen Bildschirm (z.B. einem Tablet) anzuzeigen. Nicht vorhanden aber nötig war eine Möglichkeit für einen Live-Datenaustausch mit einem anderen Programm, am besten mittels UDP-Port. Weiterhin musste noch geklärt werden, ob Parameter manipulieren werden können. Für einige angedachte Experimente war es außerdem wichtig, dass ein Fahrprofil oder Fahrverlauf eingelesen werden kann.

Die vierte Kategorie sammelte Parameter, welche exportiert, manipuliert oder mit einem anderen Programm ausgetauscht werden sollten. Als auslesbar wurden die Parameter Geschwindigkeit, Gas- und Bremspedalstellung, Fahrzeuggewicht, Drehzahl und Tankfüllung bzw. Energiestand identifiziert. Beschleunigung, Reichweite, gefahrene Kilometer, Rekuperation (zurückgewonnene Energie), Momentanverbrauch des Motors, Streckenbeschaffenheit, Wind und Wetter und sonstige Verbraucher wurden als nötige Parameter identifiziert, die standardmäßig nicht auslesbar sind.

Die Anforderungen an die Szenarien/Strecken und das Fahrzeug wurden als einmaliger Aufwand klassifiziert. Nach einem Erstellen einer geeigneten Strecke und eines geeigneten Fahrzeugs sollte die Anwendung dieser keine Fachkenntnisse benötigen. Für die Schnittstellen hingegen, die die Nutzung der verschiedenen Parameter einschließt, wurde zu der Einschätzung gelangt, dass die Nutzung dieser Fachkenntnisse oder eine Hilfestellung benötigen werden.

Abbildung 3: Erfüllte und nicht erfüllte Anforderungen an BeamNG.research für Untersuchungen in der Nutzer-Energie-Interaktion nach Stand der Kurzanalyse

2.3 Anforderungsanalyse

Im Rahmen der Anforderungsanalyse wurden mittels der in Abschnitt 2.1 Analyseverfahren vorgestellten Verfahren diverse Anforderungen an eine Fahrsimulationssoftware identifiziert. Diese ließen sich in die Kategorien Datenexport, Energetische Simulation und IPC, Fahrzeug, Experimentelle Kontrolle, UI-Mods und Strecke einteilen. In den folgenden Unterabschnitten werden die Anforderungen innerhalb der jeweiligen Kategorie erläutert, inklusive eines Verweises, woher diese Informationen stammen. Die Bereiche und die Anforderungen innerhalb der Bereiche wiesen jeweils eine unterschiedliche Priorität auf. Anforderungen, die nur für besonders komplexe Experimente relevant sind, hatten eine geringere als solche, die selbst für Experimente benötigt werden.

2.3.1 Datenexport

Dass der Datenexport eine essentiell wichtige Funktionalität war, war bereits vor Beginn dieser Arbeit bekannt. Bestätigt wurde das u. a. in beiden Interviews. Interessanter war, welche Parameter ausgelesen werden müssen. Um das herauszufinden wurde zunächst eine Liste der benötigten Parameter per E-Mail eingeholt. Diese Liste wurde anschließend iterativ in einem Dokument (siehe Anhang B: Anleitung) erweitert und verfeinert, bis schließlich die folgenden Parameter als notwendig identifiziert waren:

- Zeit
- Geschwindigkeit
- Beschleunigung
- Momentane/r und maximale/r Tankfüllung bzw. Energiestand
- Momentanverbrauch
- Gas- und Bremspedalstellung
- Restreichweite
- Drehzahl des Motors

- Position und Richtung des Fahrzeugs
- Gefahrene Distanz
- Gefahrene Route
- Fahrzeuggewicht
- Zustand der Zündung
- Leistung des Motors
- Batteriespannung
- Sonstige Verbraucher
- Streckenbeschaffenheit

Innerhalb dieser Liste existierten unterschiedliche Prioritäten, sortiert von oben höchste Priorität nach unten niedrigste Priorität. So wurden die Zeit, Geschwindigkeit, Beschleunigung, die Pedalstellungen, der Energiestand und der Momentanverbrauch als am wichtigsten identifiziert, wobei man die Beschleunigung und den Momentanverbrauch auch aus der Geschwindigkeit bzw. dem Energiestand und der Zeit berechnen könnte.

2.3.2 Energetische Simulation und IPC

Die Genauigkeit der energetischen Simulation wurde als essentiell wichtig identifiziert. Dies wurde u. a. in beiden Interviews betont. Zeitlich parallel zu dieser Arbeit wurde ein Skript für Matlab unter Verwendung von Simulink entwickelt, welches die Energiesimulation extern übernehmen sollte. Interviewteilnehmer 1 wies von sich aus auf das Skript hin, Teilnehmer 2 fand den Vorschlag die Energiesimulation auszulagern gut, solange sie präzise sei.

Bezüglich der energetischen Simulation bestand an eine Fahrsimulationssoftware also die Anforderung, dass mittels Interprocess Communication (IPC) Daten mit einem anderen Programm (in diesem Fall Matlab) ausgetauscht werden können mussten. Da die Daten in Echtzeit berechnet werden, musste der Austausch so schnell wie möglich arbeiten.

2.3.3 Fahrzeug

Bezüglich des Fahrzeugs wurden verschiedene Anforderungen identifiziert. In beiden Interviews wurde deutlich, dass sowohl ein Fahrzeug mit Elektro- als auch ein Fahrzeug mit Hybridantrieb benötigt wurden. Ein Fahrzeug mit Elektroantrieb wurde dabei als am wichtigsten identifiziert. Bezüglich des Modells sagte Interviewteilnehmer 1 im Interview aus, lediglich ein möglichst repräsentatives Auto zu benötigen. Ein repräsentatives Auto mit Elektroantrieb wurde auch in persönlichen Gesprächen als höchste Priorität in Sachen Fahrzeugauswahl bestätigt. Interviewteilnehmer 2 wünschte sich verschiedene Modelle für ein wenig Auswahl, insbesondere benötigte er auch einen elektrischen Bus.

Ebenfalls wichtig bei einem Fahrzeug war laut beiden Interviewteilnehmern, dass das Fahrzeug inklusive des Cockpits realistisch aussieht. Auch eine realistische Kamerapositionierung und daraus resultierende Perspektive wurde in persönlichen Gesprächen als wichtig identifiziert.

2.3.4 Experimentelle Kontrolle

Ein wichtiger Punkt bezüglich der Experimentellen Kontrolle ist es, dass Experimente ohne Eingriffe des Versuchsleiters ablaufen sollen, um Versuchsleitereffekte zu vermeiden. Deshalb ist es nötig, dass eine erstellte Sequenz von Fahrszenen inklusive der Anweisungen, was getan werden soll, automatisch ablaufen kann.

Dazu wurde zunächst der Ablauf eines potentiellen Experiments geklärt. Zu diesem Zweck wurde mit einer Doktorandin, die Experimente mit dem Fahrsimulator durchzuführen plante, und dem Betreuer dieser Arbeit gemeinsam ein Dokument erarbeitet, welches den Ablauf einer ersten Version eines möglichen Experiments beschreibt. Dabei wurde auch auf einzelne Anforderungen eingegangen, die an den jeweiligen Stellen erkennbar wurden. Das Dokument kann unter Anhang C: Experimentalablauf eingesehen werden.

Eines der wichtigsten Elemente eines Experimentes, welches automatisch ablaufen muss, ist das Ein- oder Ausblenden von Anweisungen zu den richtigen Zeitpunkten. Auch Audioinstruktionen bei bestimmten Bedingungen sind denkbar. Dazu müssen Trigger definiert werden können. Diese sollen entweder von dem aktuellen Ort, der verstrichenen Zeit, aktuellen Geschwindigkeit oder gefahrenen Distanz anhängig sein. Außerdem sollen verschiedene Trigger kombiniert werden können, damit z.B. nachdem zwei Sekunden lang 70 km/h gefahren wurden eine

neue Anweisung erscheinen kann. Ebenfalls wichtig ist es, dass Anweisungen und Fahrsequenzen in einer zufälligen aber nachvollziehbaren Reihenfolge angezeigt bzw. durchgeführt werden können. Es kann auch nötig sein, dass die Simulation bei einem Trigger pausiert wird, damit z.B. eine Befragung des Teilnehmers eingeschoben werden kann. Damit Fahrsequenzen mit kontrolliert unterschiedlichen Bedingungen (z.B. einer anderen Anzeige) stattfinden können, muss eine Teleportation des Fahrzeugs zurück an den Startort der vorherigen Bedingung möglich sein. Für einige Experimente ist es außerdem nötig, die Startgeschwindigkeit des Fahrzeugs festlegen zu können, damit eine initiale Beschleunigung nicht vom Versuchsteilnehmer ausgeführt oder auch nur bemerkt wird.

Darüber hinaus soll ein Experiment repliziert werden können. Dies soll auch für Forscherteams anderer Institutionen mit deren eigenem Fahrsimulator möglich sein. Der Ablauf eines Experiments muss daher in möglichst wenigen Dateien festgelegt werden, damit diese zwecks Reproduktion veröffentlicht werden können. Außerdem soll ein Experiment möglichst einfach nachvollziehbar sein.

2.3.5 UI-Mods

UI-Mods sind Mods, die eine UI hinzufügen. In Unterabschnitt 2.7.1 Erläuterungen zu BeamNG.research wird etwas ausführlicher erläutert, was eine UI-Mod ist. In diesem Unterabschnitt wird zwischen zwei Nutzungsarten von UI-Mods unterschieden: Anzeigen und Instruktionen. Anzeigen sind während einer Fahrt sichtbar. Instruktionen sind zwischen zwei Fahrsequenzen sichtbar und blockieren die Sicht, sodass, während eine Instruktion angezeigt wird, eine Fahrt gar nicht möglich ist.

Bezüglich der UI-Mods wurde in beiden Interviews deutlich, dass verschiedene Anzeigen in Experimenten gegeneinander getestet werden sollen. Entsprechend Unterabschnitt 2.3.4 Experimentelle Kontrolle soll der Wechsel der Anzeigen in Abhängigkeit vom jeweiligen aktuellen Zustand des Experiments automatisch erfolgen.

Als essentiell wichtig wurde außerdem u.a. in beiden Interviews identifiziert, dass die Anzeigen leicht erkennbar und gut lesbar sein müssen.

Die direkte Einbindung der Anzeigen in dem Fahrzeugcockpit anstelle einer UI-Mod wurde in nachfolgenden persönlichen Gesprächen als optimal bezeichnet. Die leichte Erkennbarkeit und

Lesbarkeit stellt jedoch eine notwendige Bedingung dar, die die Einbindung einer Anzeige erfüllen soll.

Als wichtige Anforderungen an mittels UI-Mod dargestellten Anzeigen wurde außerdem identifiziert, dass Anzeigen eine beliebige Größe haben können sollen. Außerdem sollen Anzeigen transparent oder intransparent sein können und ein- und ausgeblendet werden können.

Ebenfalls entsprechend Unterabschnitt 2.3.4 Experimentelle Kontrolle sollten auch der Wechsel und das Ein- und Ausblenden der Instruktionen entsprechend des Zustand des Experiments automatisch erfolgen. Die Möglichkeit, in einer eingeblendeten Instruktion Eingaben tätigen zu können, damit z.B. Befragungen zwischen verschiedenen Fahrsequenzen automatisiert ablaufen können, wurde ebenfalls begrüßt.

2.3.6 Strecke

Entsprechend des ersten Anforderungsblocks aus Abschnitt 2.2 Erste Kurzanalyse wurden unterschiedliche Strecken mit verschiedenen Eigenschaften wie städtisch oder ländlich, kurvig oder gerade oder eben oder mit Steigung benötigt. Ebenfalls bestätigt wurde die Anforderung, dass ein deutsches Layout der Straße wichtig sei. Laut Interviewteilnehmer 1 sei dies wichtiger, je städtischer die Strecke sei, da bei einer Strecke innerhalb einer Stadt mehr Merkmale vorhanden seien, die vertraut (deutsch) oder fremd wirken können.

Eine einfache, möglichst gerade Strecke wurde von Interviewteilnehmer 1 bevorzugt. Dies wurde auch mehrfach in persönlichen Gesprächen bestätigt.

Aufgrund der Tatsache, dass die Erstellung einer Strecke eine nahezu reine „Fleißarbeit“ zu sein scheint (vgl. Unterabschnitt 2.7.7 Strecke) und in Anbetracht der begrenzten Zeit dieser Arbeit und des Umfangs der sonstigen zu erfüllenden Anforderungen, wurde die Erstellung einer geeigneten Strecke in die Verantwortung einer wissenschaftlichen Hilfskraft übergeben. Entsprechend wird der Bereich Strecke in den Kapiteln 3 und 4 nicht weiter betrachtet.

2.4 Benutzeranalyse

Ziel dieser Arbeit war es, BeamNG.research für den Fahrsimulator am IMIS nutzbar zu machen. Daher konnten zwei hauptsächliche Benutzergruppen identifiziert werden. Bei diesen handelt

es sich einerseits um Mitarbeiter des IMIS und andererseits um Studenten, die ihre Qualifikationsarbeit am IMIS schreiben. Grundsätzlich ist vorstellbar, dass die Nutzung des Fahrsimulators zukünftig auch für wissenschaftliche Mitarbeiter bzw. Studenten anderer Institute interessant sein könnte. Allerdings lässt sich über diese Personen keine Aussage treffen, weshalb angenommen wird, dass sie keine besonderen Anforderungen darstellen. In den folgenden Unterabschnitten 2.4.1 Mitarbeiter und 2.4.2 Bacheloranden und Masteranden wird auf die beiden identifizierten Benutzergruppen eingegangen.

2.4.1 Mitarbeiter

Die Benutzergruppe der Mitarbeiter umfasst Professoren, Doktoren, Doktoranden, wissenschaftliche Mitarbeiter und wissenschaftliche Hilfskräfte. Letztere sind Studierende und teilen daher einige Eigenschaften mit der zweiten Benutzergruppe. Es ist sogar möglich, dass eine Person gleichzeitig beiden Benutzergruppen angehört.

Die Mitarbeiter haben (mit Ausnahme der wissenschaftlichen Hilfskräfte) ein Studium abgeschlossen. Ein Teil der Mitarbeiter hat ein Psychologiestudium absolviert, ein anderer Teil ein Informatik- oder Medieninformatikstudium. Die Vorkenntnisse im Umgang mit Computersystemen sind also potentiell stark unterschiedlich. Es ist außerdem davon auszugehen, dass nicht alle Mitarbeiter Programmierkenntnisse besitzen. Insbesondere sind keine Kenntnisse mit der von BeamNG.research verwendeten Skriptsprache Lua anzunehmen. Des Weiteren ist anzunehmen, dass bei den Mitarbeitern keine Vorkenntnisse im Umgang mit Fahrsimulatoren oder der Software BeamNG.research bestehen.

Mitarbeiter haben Zugriff auf den Fahrsimulator und damit auch auf Dokumente zur Bedienung des Fahrsimulators, die auf diesem hinterlegt sind. Alle relevanten Dateien können also auf dem Rechner des Fahrsimulators hinterlegt werden.

Mitarbeiter wollen den Fahrsimulator für ihre eigene Forschung oder die Forschung Vorgesetzter (insbesondere bei den wissenschaftlichen Hilfskräften relevant) benutzen. Sie arbeiten also in einem wissenschaftlichen Kontext (siehe Abschnitt 2.6 Organisationsanalyse).

2.4.2 Bacheloranden und Masteranden

Wie bereits in Unterabschnitt 2.4.1 Mitarbeiter erwähnt, kann ein Studierender, der seine Qualifikationsarbeit schreibt, auch ein Mitarbeiter (nämlich eine wissenschaftliche Hilfskraft) sein, i.A. ist dies jedoch nicht der Fall.

Bei den Bacheloranden bzw. Masteranden handelt es sich in erster Linie um Studierende der Medieninformatik oder der Psychologie. Es ist jedoch auch denkbar, dass Studierende anderer Informatikstudiengänge den Fahrsimulator für ihre Qualifikationsarbeit nutzen werden. Die Vorkenntnisse im Umgang mit Computersystemen sind also auch bei dieser Benutzergruppe potentiell stark unterschiedlich. Auch Programmierkenntnisse können nicht bei allen Mitgliedern dieser Benutzergruppe als gegeben angesehen werden. Vorkenntnisse im Umgang mit einem Fahrsimulator, BeamNG.research oder Lua sind bei dieser Benutzergruppe nicht anzunehmen.

Bacheloranden bzw. Masteranden arbeiten ungefähr ein halbes Jahr mit dem Fahrsimulator, um ihn für ihre Qualifikationsarbeit zu nutzen. Ein Einarbeiten in den Fahrsimulator in angemessener Zeit ist für diese Benutzergruppe daher wichtig. Wie auch ein Mitarbeiter haben Bacheloranden bzw. Masteranden Zugriff auf den Fahrsimulator und somit auf hierfür hinterlegt Dokumente.

2.5 Kontextanalyse

In diesem Abschnitt wird der physische Kontext des Fahrsimulators analysiert. Der organisatorische Kontext wird in Abschnitt 2.6 Organisationsanalyse betrachtet.

Der Fahrsimulator ist meist im Multimedialabor des IMIS aufgebaut. Sowohl die Vorbereitung eines Experiments als auch die Durchführung können in diesem ausgeführt werden. Der Fahrsimulator ist jedoch mobil. Er kann jederzeit abgebaut und woanders verwendet werden. Daraus ergibt sich an die Hardware die Anforderung, dass die für alle bzw. die meisten Experimente erforderliche essentielle Hardware mobil sein soll. An die Software, die den Fokus dieser Arbeit darstellt, ergeben sich aus dem Kontext keine besonderen Anforderungen. Für eine Auflistung der verwendeten Hardware siehe Kapitel 1 Unterabschnitt 1.2.1 Fahrsimulator am IMIS.

2.6 Organisationsanalyse

Der Fahrsimulator am IMIS soll für wissenschaftliche Untersuchungen genutzt werden. Es ist wichtig, dass am Fahrsimulator durchgeführte Experimente reproduziert werden können, nicht nur am IMIS sondern auch von Personen außerhalb der Universität zu Lübeck mit deren eigenen Fahrsimulatoren. Es muss also möglich sein, mit der Fahrsimulationssoftware reproduzierbare Experimente vorzubereiten und durchzuführen. Experimente müssen nach dem Starten ohne Eingriffe des Versuchsleiters ablaufen können, um Versuchsleitereffekte zu vermeiden. Mit der Fahrsimulationssoftware müssen also automatisch ablaufende Experimente vorbereitet werden können. Damit die Ergebnisse möglichst realistisch sind, muss alles, was durch die Fahrsimulationssoftware dargestellt wird, so realistisch und genau wie möglich sein. Dies umfasst eine realistische Optik, realistische Fahrphysik, bei vorgegebenen Fahrszenen ein realistisches Fahrverhalten und eine realistische Darstellung von GUIs. Insbesondere die Auflösung und die Detailtiefe sind relevant um eine gute Immersion zu erzeugen (Wilcox-Netepczuk, 2013) und somit valide Ergebnisse zu erzielen. Auch die Werte von Parametern, die ein Versuchsteilnehmer sieht, wie z.B. den Momentanverbrauch, müssen realistisch sein. Siehe dazu auch Unterabschnitt 2.3.4 Experimentelle Kontrolle.

2.7 Analyse der Möglichkeiten in BeamNG.research

Bei vielen Anforderungen musste überprüft werden, ob diese standardmäßig von BeamNG.research erfüllt werden. Außerdem musste untersucht werden, welche Möglichkeiten bestanden, die noch nicht erfüllten Anforderungen umzusetzen.

Zunächst werden in Unterabschnitt 2.7.1 Erläuterungen zu BeamNG.research einige allgemeine Informationen zu BeamNG.research aufgeführt, die zu technisch für Kapitel 1 Unterabschnitt 1.2.3 BeamNG.research und BeamNG.drive sind. Anschließend werden in diesem Abschnitt die Ergebnisse der Analyse der Möglichkeiten in BeamNG.research sortiert nach den Unterabschnitten in Abschnitt 2.3 Anforderungsanalyse dargestellt. Eine für alle Bereiche relevante Information ist, dass alle Quellcodedateien von BeamNG.research einsehbar und modifizierbar sind. Theoretisch ist es also möglich, alle nur erdenklichen Modifikationen vorzunehmen, sofern das nötige Fachwissen vorhanden ist.

Die in diesem Abschnitt genannten Informationen wurden Primär mittels des Forums (BeamNG, n.d.), des Wikis (Streams, n.d.), Digging (also dem Durchsuchen der Quellcodedateien, um Informationen zu finden) und Trial and Error gefunden.

2.7.1 Erläuterungen zu BeamNG.research

In diesem Unterabschnitt werden zunächst die verschiedenen Arten von Mods in BeamNG.research kurz vorgestellt. Anschließend wird erläutert, was Streams sind. Abschließend werden die wichtigsten Instanzen in BeamNG.research erläutert und ihre Kommunikation untereinander dargestellt.

Die erste Mod ist ein **Fahrzeug**. Diese fügt ein Fahrzeug hinzu und sollte nicht mit einer Fahrzeugmod (siehe nächster Absatz) verwechselt werden. Ein Fahrzeug besteht hauptsächlich aus .jbeam Dateien, welche die Struktur des Fahrzeugs definieren.

Eine **Fahrzeugmod** ist eine einzelne Komponente, die zu einem Fahrzeug hinzugefügt werden kann. Diese kann die Optik des Fahrzeugs beeinflussen, dies muss aber nicht der Fall sein. Ein Beispiel für eine Fahrzeugmod ist der in Unterabschnitt 2.7.2 Datenexport erwähnte Datenlogger. Eine Fahrzeugmod besteht aus einer .lua Datei, in der die Funktionalität liegt, einer .json Datei, in der Metainformationen abgelegt werden, und einer .jbeam Datei pro Fahrzeug, in dem die Fahrzeugmod verwendet werden soll.

Ein **Szenario** ist eine vorgegebene Sequenz von Aufgaben, die der Fahrer lösen muss. Beispielsweise kann ein Szenario fünf Orte auf der Strecke vorgeben, die in einer bestimmten Reihenfolge abgefahren werden müssen. Ein Szenario besteht im Wesentlichen aus einer .prefab Datei, in der alle Objekte des Szenarios (wie z.B. Absperrungen und das zu nutzende Fahrzeug) aufgelistet sind, einer .json Datei, die Metainformationen enthält, und einer .lua Datei, die Funktionalitäten enthält.

Eine **Karte** fügt eine eine neue Karte mit darauf befindlichen Strecken hinzu. Karten wurden im Rahmen dieser Arbeit nicht näher untersucht, da die Kartenerstellung von einer wissenschaftlichen Hilfskraft durchgeführt wurde (siehe Unterabschnitt 2.3.6 Strecke).

Eine **UI-Mod** fügt eine Anzeige hinzu, in der Daten innerhalb von BeamNG.research während einer Fahrt visualisiert werden können. Mit Knöpfen und anderen Steuerelementen kann über UI-Mods potentiell auch der Ablauf einer Fahrt manipuliert werden. Im Wesentlichen besteht

eine UI-Mod aus einer Metainformationen enthaltenden .json Datei und einer .js Datei, in der neben JavaScript Code auch HTML und CSS verwendet werden können. Alternativ können .html und .css Dateien eingebunden werden.

Streams enthalten Datenwerte und sind im Fahrzeug (siehe Absatz zu den Instanzen) zu finden. Die meisten standardmäßig vorhandenen Streams werden bei jedem Physiktick (also jeder Aktualisierung der im Rahmen der Physik berechneten Werte) aktualisiert. Einige Streams (wie z.B. `electrics.values.throttleOverride`) nehmen nur dann einen anderen Wert als `nil` (die Luaversion von `null`, also nichts) an, wenn sie vom Nutzer (oder einer Mod) gesetzt werden. Dann beeinflussen sie die Fahrt, beispielsweise wird im Falle von throttleOverride der physische Pedal-Input verworfen und durch den gesetzten Wert ersetzt. Es ist möglich, einen eigenen Stream zu definieren. Diesen kann man entweder per Mod aktualisieren lassen, um immer einen aktuellen Wert abfragen zu können, oder bei einem fixen Wert belassen und dadurch als eine Art global verfügbare Variable nutzen.

Die relevanten **Instanzen** in BeamNG.research sind das Fahrzeug, die Game Engine (ge) und meist noch das Szenario und die UI-Mods. Das Fahrzeug kann mittels Vehicle-Lua manipuliert werden. Zum Fahrzeug gehören auch die Streams und eine eventuell verwendete Fahrzeugmod. Die ge und das Szenario verwenden normales Lua. Worin der Unterschied zwischen Lua und Vehicle-Lua besteht, konnte nicht festgestellt werden. In der Konsole werden die beiden Luas jedoch strikt getrennt.

Die ge und das Fahrzeug können mit den Befehlen `obj:queueGameEngineLua(‚...‘)` und `playervehicle:queueLuaCommand(‚...‘)` kommunizieren. Sowohl die ge als auch das Fahrzeug können mittels `scenario_scenario_name.functionName()` die Funktion mit dem Namen functionName des Szenarios scenario_name aufrufen. Das Szenario kann einen Befehl an das Fahrzeug mit der Funktion `helper.queueLuaCommandByName(‚scenario_player0‘,‘...‘)` übergeben. Eine UI-Mod kann mittels `bngApi.activeObjectLua(‚...‘)` einen Luabefehl an das Fahrzeug übermitteln. Mit einer UI-Mod kann (scheinbar) nicht direkt kommuniziert werden. Da eine UI-Mod Streams auslesen kann, kann mit dazu gedachten Streams indirekt mit einer UI-Mod kommuniziert werden. Für die Kommunikation zwischen Instanzen ist manchmal eine Verkettung von Kommunikationsbefehlen nötig (StinchinStein, 2016), z.B. muss das Szenario dem Fahrzeug befehlen, eine Funktion des Szenarios aufzurufen, weil das Szenario die Streams nicht direkt auslesen kann. Abbildung 4 visualisiert den Datenfluss zwischen den Instanzen.

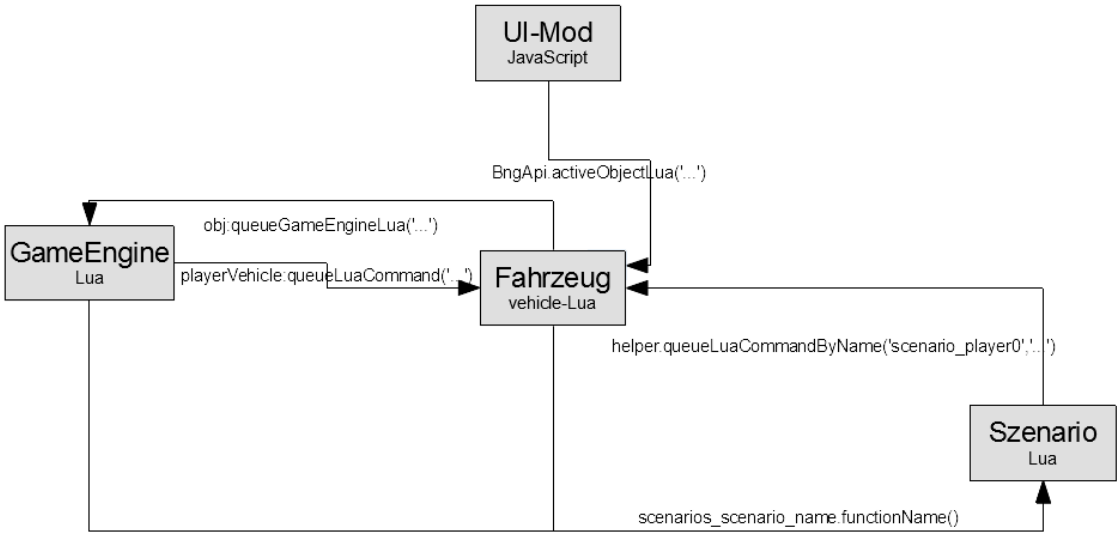

Abbildung 4: Die für diese Arbeit relevanten Instanzen in BeamNG.research und deren Kommunikation

2.7.2 Datenexport

Zunächst wurde die standardmäßig vorhandene UI-Mod „Log all available Streams“ getestet. Zum Beginn dieser Arbeit funktionierte diese Mod nicht. Wie bereits in Abschnitt 2.2 Erste Kurzanalyse erwähnt, wurde von einem BeamNG Mitarbeiter eine Mod zur Verfügung gestellt, die den Datenexport in eine .csv Datei umsetzte, weshalb die Mod „Log all available Streams“ nicht weiter betrachtet wurde (siehe Kapitel 6 Abschnitt 6.3 Ausblick). Im Bereich des Datenexports war daher hauptsächlich zu klären, welche der in Unterabschnitt 2.3.1 Datenexport ermittelten Parameter ausgelesen werden können und wie dies möglich ist. Parameter können geloggt werden, indem der Wert des Parameters in der Vehicle-Lua Umgebung abgefragt wird. In der Mod muss daher der Lua-Name eines Parameters angegeben werden.

Um den jeweiligen Lua-Namen in Erfahrung zu bringen, wurde für jeden Parameter zunächst der einfachste Weg ausprobiert. Dieser bestand darin, die Streams-Seite des BeamNG.drive Wikis (Streams, n.d.) zu nutzen, da hier viele auslesbare Parameter mit ihren jeweiligen Lua-Namen aufgelistet werden. Allerdings war die Auflistung nicht vollständig und die angegebenen Lua-Namen zum Teil in der Mod nicht funktionsfähig. Ob sie an anderer Stelle funktioniert hätten, wurde nicht überprüft, da es für diese Arbeit keine Relevanz hatte. Bei einigen der Parametern musste der angegebene Name nur leicht verändert werden, z.B. stand im Wiki häufig

‚electronics' obwohl ‚electrics' richtig wäre. Tabelle 1 listet die Parameter auf, deren Lua-Namen mit Hilfe des Wikis, und meist durch Ausprobieren, bestimmt wurden. Später wurde beim Durchsuchen verschiedener Dateien die Datei ‚\lua\vehicle\electrics.lua' gefunden, in der alle ‚electrics.values' gefunden werden können.

Parameter	**Lua-Name**
Geschwindigkeit	`electrics.values.wheelspeed`
Geschwindigkeit	`electrics.values.airspeed`
Gaspedalstellung	`electrics.values.throttle`
Bremspedalstellung	`electrics.values.brake`
Momentane relative Tankfüllung	`electrics.values.fuel`
Momentane absolute Tankfüllung	`electrics.values.fuelVolume`
Maximale Tankfüllung	`electrics.values.fuelCapacity`
Drehzahl des Motors	`electrics.values.rpm`

Tabelle 1: Über das BeamNG.drive Wiki ermittelte Parameter

Konnte der Lua-Name eines Parameters nicht mittels Wiki ermittelt werden, mussten andere Verfahren angewandt werden. Die in Tabelle 2 aufgelisteten Lua-Namen wurden mithilfe einer Kombination aus Beiträgen im BeamNG.drive Forum (BeamNG, n.d.), Austesten verschiedener Befehle in der Vehicle-Lua Konsole in BeamNG.research und der Durchsicht verschiedener BeamNG.research Dateien gefunden. Der Lua-Name des Zustands der Zündung wurde in der Datei ‚\lua\vehicle\controller\vehicleController.lua' gefunden. Die Lua-Namen der Energiestände wurden dank eines Hinweises auf Reddit (theseeker01, 2017) gefunden, bei diesen Parametern war jedoch ein Austesten mit der Konsole das Verfahren, das exakt die gewünschten Lua-Namen lieferte. Die Positions- und Ausrichtungsparameter wurden mittels Durchsicht einiger Dateien gefunden. Einen ersten Hinweis lieferte das Wiki. Daraufhin wurde die Datei ‚\lua\vehicle\sensors.lua' untersucht, in der ein Hinweis in Richtung der Datei ‚\lua\vehicle\guistreams.lua' gefunden wurde. In dieser wurden schließlich die Lua-Namen gefunden. Das Fahrzeuggewicht wurde auf Pastebin gefunden (Modified esc file for beamng drive, 2018) und musste nicht weiter angepasst werden.

Parameter	Lua-Name
Zustand der Zündung	`controller.mainController.engineInfo[18]`
Momentaner Energiestand	`energyStorage.getStorage('mainTank').storedEnergy` bzw. `energyStorage.getStorage('mainBattery').storedEnergy`
Maximaler Energiestand	`energyStorage.getStorage('mainTank').initialStoredEnergy` bzw. `energyStorage.getStorage('mainBattery').initialStoredEnergy`
X-Position	`obj:getPosition().x`
Y-Position	`obj:getPosition().y`
Z-Position	`obj:getPosition().z`
X-Ausrichtung	`obj:getDirectionVector().x`
Y-Ausrichtung	`obj:getDirectionVector().y`
Z-Ausrichtung	`obj:getDirectionVector().z`
Fahrzeuggewicht	`obj:calcBeamStats().total_weight`

Tabelle 2: Außerhalb des BeamNG.drive Wikis ermittelte Parameter

Tabelle 3 listet alle Parameter aus Unterabschnitt 2.3.1 Datenexport auf, für die kein Lua-Name gefunden wurde. Auf der Suche nach den Lua-Namen der Beschleunigung und des Momentanverbrauchs wurden UI-Mods untersucht, die standardmäßig in BeamNG.research vorhanden sind und diese Parameter darstellen. In den betreffenden UI-Mods wurden die Parameter jedoch nicht aus einem Stream ausgelesen sondern aus der Zeit und der alten und neuen Geschwindigkeit bzw. Restenergie berechnet. Es wurde daher davon ausgegangen, dass die Parameter Beschleunigung und Momentanverbrauch nicht direkt auslesbar sind sondern aus der Zeit und Geschwindigkeit bzw. Restenergie errechnet werden müssen.

Für die restlichen sieben Parameter wurde weder ein Lua-Name noch ein Hinweis darauf, dass kein Lua-Name existiert, gefunden. Vier der sieben Parameter sind der energetischen Simulation zuzuordnen. Da diese, wie in Unterabschnitt 2.3.2 Energetische Simulation und IPC erwähnt, ausgelagert werden sollte, ist zu vermuten, dass diese Parameter wahrscheinlich aus dem externen System auslesbar sind. Die gefahrene Route und Strecke sind durch die Position und Ausrichtung indirekt auslesbar. Da sie keine hohe Priorität hatten, wurde im Rahmen dieser Arbeit kein weiterer Aufwand darauf verwendet, eine direkte Möglichkeit zu finden. Die

Streckenbeschaffenheit wird von der externen Energiesimulation nicht betrachtet und hatte daher keine hohe Priorität.

Parameter	Auslesbarkeit
Beschleunigung	Nur indirekt
Batteriespannung	Unbekannt
Leistung des Motors	Unbekannt
Momentanverbrauch	Nur indirekt
Restreichweite	Unbekannt
Sonstige Verbraucher	Unbekannt
Gefahrene Route	Unbekannt
Gefahrene Strecke	Nur indirekt
Streckenbeschaffenheit	Unbekannt

Tabelle 3: Parameter, für die keine direkte Möglichkeit des Auslesens gefunden wurde

2.7.3 Energetische Simulation und IPC

Wie bereits in Unterabschnitt 2.7.2 Datenexport erwähnt, können Daten aus BeamNG.research mittels Mod in eine Datei geschrieben werden. Durch Austesten wurde bestätigt, dass so auch Daten aus einer Datei eingelesen werden können. Somit ist eine File-based IPC (mehr dazu in Kapitel 3 Abschnitt 3.2 Energetische Simulation und IPC) möglich. Mittels Forumsbeitrag (Stiegfried, 2018a) wurden die Dateien ‚\lua\vehicles\extensions\outgauge.lua' und ‚\lua\vehicles\extensions\outsim.lua' gefunden. Eventuell ermöglichen diese eine Socket-based IPC. Dies wurde jedoch nicht weiter untersucht. Die Gründe dafür sind die begrenzte Zeit dieser Arbeit und die Tatsache, dass die File-based IPC funktioniert.

Bei der File-based IPC war zunächst noch fraglich, wie gut die Performanz ist. Deshalb wurden sechs kurze Versuche durchgeführt. Diese kann man zwar nicht als repräsentativ bezeichnen, doch sie geben einen groben Eindruck, dass die Performanz der File-based IPC ausreicht. Für die Versuche wurde ein Matlabskript geschrieben, welches Daten aus BeamNG entgegennimmt und daraus berechnete Werte zurückgibt. Diese berechneten Werte haben den einzigen Nutzen, dass sie demonstrieren, dass die IPC inklusive in Matlab ausgeführter Berechnungen basierend auf den eingehenden Werten funktioniert. Nach dem ersten Versuch wurde das Skript noch etwas modifiziert um die Performanz der IPC zu verbessern. Später wurde noch eine dritte Version geschrieben, die die Performanz weiter verbesserte. Diese dritte Version

wurde jedoch aus Zeitgründen nicht ausführlich getestet, weil die Performanz nicht schlechter sein kann als in Version zwei (warum das so ist wird in den folgenden Absätzen erläutert) und die Performanz von Version zwei bereits ausreichend gut war. Die getestete Version zwei des Skripts kann unter Anhang G: dataExchange_v02.m gefunden werden.

Fünf der sechs Versuche wurden an einem privaten Gerät, nur der sechste wurde an dem in Kapitel 1 Unterabschnitt 1.2.1 Fahrsimulator am IMIS beschriebenen Laptop durchgeführt. Bei den Versuchen wurde eine IPC zwischen BeamNG.research und Matlab durchgeführt. Die Versuche dauerten unterschiedlich lange. Dabei wurde geloggt, wie häufig Matlab versucht hat, Daten einzulesen („Anläufe“), wie oft Matlab erfolgreich Daten einlesen konnte („Eingelesen“) und wie oft Matlab erfolgreich Daten ausgeben konnte („Ausgegeben“). Außerdem wurde geloggt, wie häufig keine Daten ausgegeben werden konnten wenn bereits bei dem vorherigen Anlauf keine Daten ausgegeben wurden („Aufeinanderfolgende Fehler“). Die Frequenz der IPC wurde mittels einer Pause zwischen zwei Anläufen geregelt. Aus den Spalten Anläufe und Ausgegeben wurde nachträglich der relative Fehler berechnet. Die Ergebnisse der Versuche können in Tabelle 4 eingesehen werden.

Schließt man Versuch eins aus, weil in diesem die weniger performante Version eins des Skripts verwendet wurde, erhält man als Ergebnis, dass für zwischen ca. 0.5% und 0.7% der Anläufe kein Wert ausgegeben wurde. Nur beim vierten Versuch traten aufeinanderfolgende Fehler auf. Zwar sind die Versuche nicht repräsentativ, lassen jedoch vermuten, dass die Performanz gut genug ist.

In der später erstellten Version drei des Skripts, die, aus zuvor genannten Gründen, nicht ausführlich getestet wurde, wurde die Pause ans Ende der Schleife verschoben. Dies hatte zur Folge, dass nur wenn kein Fehler auftrat eine Pause eingelegt wurde und im Falle eines Fehlers direkt ein neuer Versuch gestartet wurde. Da sich Version drei nur bei einem Fehler anders verhält als Version zwei, kann sich die Performanz im Vergleich zur Vorgängerversion nicht verschlechtert haben. Für den Fall, dass eine Pause definiert wird, wird die Performanz durch das dynamische auslassen der Pause bei einem Fehler erhöht. Wird keine Pause definiert sind Version zwei und drei quasi identisch.

Versuch Nr.	Skript-version	Gerät	Anläufe	Eingelesen	Ausgegeben	Rel. Fehler	Aufeinanderfolgende Fehler	Pause
1	1	Privat	2004	1991	1986	0.009	0	0.1s
2	2	Privat	2006	1996	1995	0.005	0	0.1s
3	2	Privat	20051	19906	19885	0.007	0	0.01s
4	2	Privat	153641	152671	152552	0.007	10	-
5	2	Privat	151455	150614	150499	0.006	0	-
6	2	Universität	100724	100134	100021	0.007	0	0.1s

Tabelle 4: Ergebnisse der Untersuchung der Performanz der File-based IPC

Laut Unterabschnitt 2.7.1 Erläuterungen zu BeamNG.research können Werte von einer Mod in Streams geschrieben werden. Diese Streams können dann von UI-Mods ausgelesen werden, wodurch es möglich ist, Werte aus Matlab (oder generell von außerhalb von BeamNG.research) darzustellen.

Es ist auch möglich, Werte, wie beispielsweise einen Pedal-Input, zu überschreiben, z.B. mit `electrics.values.brake=1` für die Bremse. Beim Gaspedal-Input (`electrics.values.throttle`) funktioniert dies jedoch nicht sehr gut, da das Fahrzeug nur mit schätzungsweise halber Kraft fährt wenn der Pedal-Input von einer Mod gesetzt wird. Dieses Problem lässt sich jedoch lösen, indem stattdessen ein überschreibender Wert (`electrics.values.throttleOverride`) genutzt wird.

Die Geschwindigkeit eines Fahrzeugs lässt sich in BeamNG.research nicht direkt festlegen. Es wurden zwei Möglichkeiten identifiziert, dieses Problem zu umgehen. Die erste besteht darin, in der Mod die aktuelle Geschwindigkeit abzufragen und bei einem Wert unterhalb der gewünschten Geschwindigkeit Gas zu geben. Bei diesem Lösungsansatz ergibt sich jedoch das Problem, dass ein unrealistisches Fahrverhalten entsteht wenn die Geschwindigkeit gehalten werden soll, da die Mod immer zwischen Gas und kein Gas wechselt. Die zweite Möglichkeit nutzt die BeamNG.research interne Extension (also ein Modul der Game Engine) ‚cruiseControl', welche standardmäßig per UI-Mod gesteuert werden kann. Dieser kann ein Geschwindigkeitswert übergeben werden, die Extension beschleunigt dann auf diese Geschwindigkeit und hält diese bei. Bei diesem Ansatz existierten zwei Probleme. Das erste war, dass standardmäßig nur Geschwindigkeiten ab 30km/h akzeptiert werden. Dies lässt sich jedoch mittels des Be-

fehls `extensions.cruiseControl.minimumSpeed=0` umgehen, welcher in der Datei ‚\ui\modules\apps\CruiseControl\app.js' gefunden wurde. Das zweite Problem war, dass beim Beschleunigen die Zielgeschwindigkeit um mehrere km/h überschritten wird, wodurch nicht mehr der gewünschte Geschwindigkeitsverlauf dargestellt wird. Dies ist jedoch nur bei zu großen Sprüngen in der Geschwindigkeit der Fall, und kann daher bei einer ausreichend hohen Auflösung der Datenpunkte (z.B. 100 Hz) ignoriert werden.

Zur Validierung, dass die Fahrt mittels vorgegebenen Geschwindigkeiten realistisch verläuft, wurden reale Fahrdaten, die von einem Mitarbeiter des IMIS zur Verfügung gestellt wurden, aus einer .csv Datei eingelesen. Zwei Probleme wurden dabei erkennbar. Erstens waren die Fahrdaten zu grob aufgelöst und wurden daher linear interpoliert. Zweitens fiel auf, dass das Fahrzeug bei langsamen Geschwindigkeiten (<10km/h) stark wackelt. Dies ist jedoch kein Problem, das nur eintritt wenn ein Geschwindigkeitsverlauf eingelesen wird, sondern auch wenn manuell langsam gefahren wird. Daher ist dieses Problem dem Bereich Fahrzeug zugeordnet.

Damit eine solche Fahrt mit eingelesenen Geschwindigkeitswerten pausiert werden kann, z.B. um eine Befragung des Versuchsteilnehmers einzuschieben, wurde eine Möglichkeit gesucht, Matlab innerhalb von BeamNG.research zu pausieren. Eine Pause lässt sich mit zwei zusätzlichen Zeilen Code in BeamNG.research realisieren, indem ein entsprechender Stream (z.B. `electrics.values.MatlabPause`) gesetzt und mit übertragen wird. Im verwendeten Matlabskript muss dann eine einfache if-Abfrage ergänzt werden, ob das Skript aktuell pausiert sein soll.

2.7.4 Fahrzeug

Fahrzeuge lassen sich in BeamNG.research mittels des in Kapitel 1 Unterabschnitt 1.2.3 BeamNG.research und BeamNG.drive bereits erwähnten Modrepository einfach hinzufügen. Fahrzeuge können selbst erstellt werden oder es gibt ein Unterforum extra für neue Fahrzeuge im BeamNG.drive Forum (Land | BeamNG, n.d.). Sowohl standardmäßig vorhandene als auch per Mod hinzugefügte Fahrzeuge können verändert werden.

Die Kamera kann hinsichtlich ihrer Positionierung im Fahrzeug, ihrer Rotation und des Field of View (FOV) angepasst werden. Dies kann einerseits in den Einstellungen oder in der Datei ‚\lua\ge\extensions\core\cameraModes\onboard.lua' geschehen. Andererseits kann die Kamera auch für ein bestimmtes Auto angepasst werden, dies ist jedoch sehr umständlich, da die

Werte im Fahrzeug mit verschiedenen, nicht offensichtlich erkennbaren Faktoren verrechnet werden, wodurch eine präzise Platzierung am gewünschten Ort aufwändig ist.

Eine Fahrzeugkonfiguration, eine .pc Datei, welche die im Fahrzeug verbauten Teile auflistet, kann auch eine Fahrzeugmod (siehe Unterabschnitt 2.7.1 Erläuterungen zu BeamNG.research) enthalten. Eine Fahrzeugkonfiguration kann entweder händisch durch Kopieren und Modifizieren einer vorhandenen Konfiguration oder in der Garage in BeamNG.drive erstellt werden. Die Funktion der Garage scheint es in BeamNG.research nicht zu geben.

Wie bereits in Unterabschnitt 2.7.3 Energetische Simulation und IPC erwähnt, wackelt das Fahrzeug bei niedrigen Geschwindigkeiten (<10km/h). Es wurde identifiziert, dass alle Fahrzeuge wackeln, jedoch unterschiedlich stark. Zunächst wurde vermutet, dass sich das Problem in den Dateien ‚coupe_suspension_F.jbeam' und ‚coupe_suspension_R.jbeam' lösen lässt, da diese die Federung des Fahrzeugs regeln. Später wurde in einer Telefonkonferenz mit Mitarbeitern der BeamNG GmbH die Vermutung geäußert, dass wahrscheinlich mehr Speichen im Rad nötig seien. Dies läge daran, dass Reifen aus Massepunkten und Federn definiert sind und die Anzahl der Massepunkte aus Leistungsgründen beschränkt ist. Mehr Speichen würden das Wackeln bei niedrigen Geschwindigkeiten verringern, jedoch die benötigte Rechenleistung etwas erhöhen und die Fahreigenschaften des Reifens bei hohen Geschwindigkeiten verschlechtern, dies sei bei 100km/h jedoch noch nicht relevant. In der .jbeam Datei der Reifen muss dazu der Wert ‚numRays' verändert werden. Dieser Hinweis wurde erst kurz vor Ende dieser Arbeit erhalten, weshalb diese wahrscheinliche Lösung noch nicht getestet wurde.

Kurz vor Ende dieser Arbeit wurde in einer Telefonkonferenz mit Mitarbeitern der BeamNG GmbH geäußert, dass eine elektrische Version des ETK800 geplant ist. Dadurch wären zukünftige Experimente am Fahrsimulator nicht auf ein Fahrzeug aus der Community angewiesen. Die Verbrauchsdaten des Fahrzeugs sollen dann an die Werte des für Matlab vorliegenden Energiemodells (siehe Unterabschnitt 2.3.2 Energetische Simulation und IPC) angepasst werden können. Dies wird jedoch erst nach Beendigung dieser Arbeit möglich sein und ist daher im Rahmen dieser Arbeit nicht weiter betrachtet worden.

2.7.5 Experimentelle Kontrolle

Der automatisierte Ablauf eines Experiments lässt sich mittels eines Szenarios realisieren. Mit einem Szenario ist es möglich, verschiedene Sequenzen in zufälliger Reihenfolge ablaufen zu

lassen, wobei die Randomisierung jedoch nicht standardmäßig implementiert ist. Indem Werte in selbst geschaffene Streams geschrieben werden (siehe Unterabschnitt 2.7.1 Erläuterungen zu BeamNG.research), kann der aktuelle Stand des Szenarios (und damit des Experiments) von allen Instanzen des Programms abgefragt werden. So können mittels UI-Mods Anweisungen in der vom Szenario entschiedenen Reihenfolge zu den im Szenario festgelegten Zeitpunkten dargestellt werden.

Im Optimalfall sollen alle Angaben, welche Anweisungen in welchem Durchlauf angezeigt werden sollen und welche Streams geloggt und mit Matlab ausgetauscht werden sollen, in einer einzigen Datei eingetragen werden, um das Reproduzieren (und Manipulieren) eines Experimentes zu vereinfachen. Zwar kann eine entsprechende Datei erstellt und auch eingelesen werden (für eine Möglichkeit, wie eine solche Datei aussehen könnte, siehe Anhang D: centralInformation.json), das Ausführen von eingetragenen Befehlen funktioniert jedoch häufig nicht. Ob ein Befehl funktioniert oder nicht hängt davon ab, welche Ressourcen er braucht, da das Ausführen eines Befehls aus einem String einen Funktionsaufruf benötigt, welcher sein Ergebnis nicht zurückliefert und keine lokalen Variablen der Instanzen von BeamNG.research verwenden kann. Dadurch können nur globale Befehle, die unabhängig von den Instanzen sind, ausgeführt werden. Es könnte möglich sein, eine Funktion zu schreiben, die einen Befehl aus einem String lokal ausführt, dies wurde jedoch aufgrund der in persönlichen Gesprächen identifizierten niedrigen Priorität zu Gunsten anderer Funktionalitäten nicht angegangen.

Die als Alternative verwendete Lösung hat zur Folge, dass die Informationen eines Experiments auf viele verschiedene Dateien aufgeteilt sind. Tabelle 5 listet die Dateien und Ordner auf, die als zur Reproduktion eines Experiments an einem anderen Rechner nötig identifiziert wurden, auf. Allerdings sind zum Erstellen/Modifizieren der relevanten Dateien gewisse Fachkenntnisse nötig, die laut Abschnitt 2.4 Benutzeranalyse nicht bei den Benutzern anzunehmen sind.

Art der Mod	Ordnerpfad/Datei	Anmerkungen
Fahrzeug	C:\Users\aFahrSim\Documents\BeamNG.drive\mods\unpacked	Der Ordner, der den Namen der Mod trägt, muss in dieses Verzeichnis kopiert werden.
Fahrzeugmod	C:\Users\aFahrSim\Documents\BeamNG.drive\mods\unpacked	Der Ordner, der den Namen der Mod trägt, muss in dieses Verzeichnis kopiert werden.
Szenario	C:\Users\aFahrSim\Documents\BeamNG.drive\mods\unpacked	Der Ordner, der den Namen der Mod trägt, muss in dieses Verzeichnis kopiert werden.
Karte	C:\Users\aFahrSim\Documents\BeamNG.drive\mods\unpacked	Der Ordner, der den Namen der Mod trägt, muss in dieses Verzeichnis kopiert werden.
UI-Mod	C:\Users\aFahrSim\Documents\BeamNG.drive\ui\modules\apps	Der Ordner, der den Namen der Mod trägt, muss in dieses Verzeichnis kopiert werden.
Kamera	C:\Program Files (x86)\BeamNG.drive\lua\ge\extensions\core\cameraModes\onboard.lua	Diese modifizierte Version einer originalen Datei muss kopiert werden, damit die Kamera korrekt positioniert ist.
Lenkradsperre	C:\Program Files (x86)\BeamNG.drive\lua\vehicle\input.lua	Diese modifizierte Version einer originalen Datei muss kopiert werden, damit das Lenkrad temporär deaktiviert werden kann
Bremssperre	C:\Program Files (x86)\BeamNG.drive\lua\vehicle\controller\vehicleController.lua	Diese modifizierte Version einer originalen Datei muss kopiert werden, damit Bremsen temporär deaktiviert werden kann
Einstellungen	C:\Users\aFahrSim\Documents\BeamNG.drive\settings	Dieser Ordner muss kopiert werden, damit diverse Einstellungen übernommen werden.

Tabelle 5: Für die Reproduktion eines Experiments erforderliche Dateien und Ordner

Generell müssen das Szenario und alle sonstigen verwendeten Mods aufeinander abgestimmt werden. Dies trifft insbesondere auf UI-Mods zu, die mittels bestimmter Streams automatisch gesteuert werden sollen.

Einige der Dateien, die zur Reproduktion eines Experiments nötig sind, sind nicht spezifisch für das Experiment. Der Ordner ‚settings' enthält die Einstellungen, die innerhalb von BeamNG.research getroffen wurden. Diese könnten alternativ auch händisch übernommen werden. Einige Dateien sind modifizierte Versionen von originalen in BeamNG.research vorhandenen Dateien, die zwecks Umsetzung bestimmter Anforderungen verändert wurden. In Kapitel 4 werden mehr Informationen hinsichtlich dessen, was in den Dateien jeweils verändert wurde, aufgeführt.

Gemäß Unterabschnitt 2.3.4 Experimentelle Kontrolle kann es nötig sein, die Simulation zu pausieren, beispielsweise um Instruktionen anzuzeigen. Dies kann mittels des Befehls `bullettime.togglePause()` getan und umgekehrt werden. Der Befehl wurde in der Datei ‚\lua\ge\bullettime.lua' gefunden, zusammen mit verschiedene anderen Funktionen für Pausen und Zeitlupen. Die Datei wurde bei der Durchsicht der Datei ‚\lua\ge\input_actions.json' gefunden.

2.7.6 UI-Mods

UI-Mods in BeamNG.research verwenden AngularJS. Außerdem finden plain HTML, CSS und JavaScript (JS) Anwendung. Im BeamNG.drive Wiki existiert eine durchaus gute Anleitung zur Erstellung einer UI-Mod (My first App, n.d.). Der AngularJS Anteil kann sehr gering gehalten werden, sodass Kenntnisse mit plain HTML, CSS und JS ausreichen, um eine UI-Mod zu erstellen.

Für UI-Mods kann -webkit CSS (jlnr & SJW, 2014) verwendet werden, wodurch Ein- und Ausblendungen der GUIs leicht realisiert werden können. Die (In-)Transparenz und die beliebige Größe lassen sich mittels plain CSS realisieren. Lediglich die Eingabe mittels Tastatur in eine UI-Mod ist nicht standardmäßig erfüllt. Eingaben in UI-Mods lassen sich allerdings über Dropdownmenüs oder andere Eingabeverfahren, die die Antwortmöglichkeiten vorgeben, realisieren.

2.7.7 Strecke

In BeamNG.research ist es möglich über das bereits in anderen Abschnitten erwähnte Modrepository Karten mit Strecken hinzuzufügen. Solche Karten können z.B. im Forum von BeamNG.drive (Terrains, Levels, Maps, n.d.) gefunden werden. Karten können auch selbst erstellt werden. In BeamNG.research existiert der sogenannte World Editor. Dieser ermöglicht es

u.a. eine vorhandene Karte zu modifizieren. Indem man eine vorhandene Karte kopiert und auf der Kopie alle unerwünschten Komponenten löscht, kann man eine eigene Karte von Grund auf erstellen. Im World Editor können gewöhnliche 3D-Modelle, wie man sie in 3D-Modellierungssoftware wie z.B. Blender erstellen oder online in diversen Foren finden kann, platziert werden. Die Erstellung einer Karte entsprechend der Anforderungen scheint insgesamt zwar aufwändig, aber, sofern man Vorerfahrung mit dem platzieren von 3D-Modellen hat, nicht kompliziert zu sein.

Da es relevant sein kann, Informationen wie z.B. die relative Position des Fahrzeugs im Bezug zur Mittellinie zu erhalten, kann es sinnvoll sein, die Strecke mittels Parametern zu generieren anstatt sie händisch zu erstellen. Dies könnte auch den Arbeitsaufwand verringern. Im Rahmen einer Telefonkonferenz mit Mitarbeitern der BeamNG GmbH kurz vor Ende dieser Arbeit wurden Dateien zur Verfügung gestellt, die zur Generierung einer Strecke nützlich sein könnten. Aus Zeitgründen wurden diese jedoch im Rahmen dieser Arbeit nicht mehr betrachtet.

3 Konzeption

Im Folgenden werden Lösungen für die in Kapitel 2 Abschnitt 2.3 Anforderungsanalyse identifizierten Anforderungen konzipiert. Für die Anforderungen jedes Unterabschnittes wurde ein eigener Abschnitt angelegt.

Nach der Entwicklung eines Konzeptes für eine der Anforderungen wurde die Machbarkeit Dieses in Kapitel 2 Abschnitt 2.7 Analyse der Möglichkeiten in BeamNG.research überprüft. Gegebenenfalls wurde die Konzeption anschließend entsprechend angepasst.

3.1 Datenexport

Entsprechend des Ergebnisses aus Kapitel 2 Unterabschnitt 2.3.1 Datenexport müssen verschiedene Parameter geloggt werden können. Die nötigen Lua-Namen wurden in Kapitel 2 Unterabschnitt 2.7.2 Datenexport ermittelt. Diese Lua-Namen müssen syntaktisch korrekt in der vorhandenen Mod verwendet werden, damit der Datenexport funktioniert. Doch laut Kapitel 2 Abschnitt 2.4 Benutzeranalyse sind keine Programmierkenntnisse und insbesondere keine Kenntnisse mit Lua bei den Benutzern anzunehmen. Deshalb sollte eine Möglichkeit konzipiert werden, wie die Daten ohne diese Kenntnisse geloggt werden können.

In einer schnell erstellbaren ersten Version einer Anleitung sollte detailliert erklärt werden, wie ein bestimmter Parameter geloggt werden kann und wie die entsprechende Zeile Quellcode in der Mod aussehen muss. Darüber hinaus sollte die Mod strukturiert werden, damit die Stellen, an denen Veränderungen vorgenommen werden müssen, leichter erkennbar sind. Die Anleitung sollte so formuliert werden, dass auch Nicht-Informatiker die Anweisungen verstehen, Fachvokabular sollte also weitestgehend vermieden oder, wo dies nicht möglich war, erläutert werden.

Des Weiteren sollte eine Datei geschrieben werden, die eine Mod für alle Fahrzeuge verfügbar macht, indem sie automatisch alle nötigen Ordner und Dateien erzeugt.

3.2 Energetische Simulation und IPC

Entsprechend des Ergebnisses aus Kapitel 2 Unterabschnitt 2.3.2 Energetische Simulation und IPC, dass eine Möglichkeit für eine IPC benötigt wurde, wurde für eine der beiden in Kapitel 2 Unterabschnitt 2.7.3 Energetische Simulation und IPC identifizierten Möglichkeiten eine mögliche, zunächst prototypische, Umsetzung konzipiert. Darauf basierend sollte später entschieden werden, welche der beiden Möglichkeiten final realisiert werden sollte.

Für die File-based IPC sollte eine Fahrzeugmod geschrieben werden, welche den aktuellsten Datensatz in einer .csv Datei speichern kann. Diese Mod sollte nach demselben Prinzip arbeiten, wie die Mod, die für den Datenexport zur Verfügung gestellt wurde (siehe Kapitel 2 Unterabschnitt 2.3.1 Datenexport). In einem Matlabskript sollten die Daten in einer festlegbaren Frequenz aus der .csv Datei eingelesen werden. Die daraus errechneten Werte sollte das Matlabskript anschließend in eine andere .csv Datei schreiben. Bei den Werte sollte es sich zunächst um Platzhalter-Werte handeln, bei einer Anbindung des Energiemodells sollten dann Werte für den Momentanverbrauch oder die Geschwindigkeit übergeben werden. Die Fahrzeugmod, die die Daten aus BeamNG.research in eine .csv Datei schreibt, sollte bei jedem Durchlauf auch die .csv Datei von Matlab einlesen. Die eingelesenen Werte sollten dann über BeamNG.research interne Datenstreams an das Fahrzeug und die GUI übergeben werden (vgl. Kapitel 2 Unterabschnitte 2.7.1 Erläuterungen zu BeamNG.research und 2.7.3 Energetische Simulation und IPC). Abbildung 5 visualisiert diesen konzipierten Datenfluss zwischen BeamNG.research und Matlab. Dabei stellen grüne Pfeile Daten dar, die vom Sender aktiv übergeben werden und blaue Pfeile Daten, die vom Empfänger ausgelesen werden. Bei dieser IPC findet also ein zeitlich asynchroner Datenaustausch statt. Um eine hohe Präzision zu erreichen könnte man die Frequenz in Matlab erhöhen, was natürlich mehr Rechenleistung beanspruchen würde. Vorteile dieser Variante wären eine simple Implementierung wodurch im Rahmen dieser Arbeit mehr Zeit für andere Anforderungen verfügbar wäre und die Möglichkeit Daten auch ohne Matlab an BeamNG.research zu übergeben, indem man die Datei ‚Output_Matlab.csv' per Hand manipuliert.

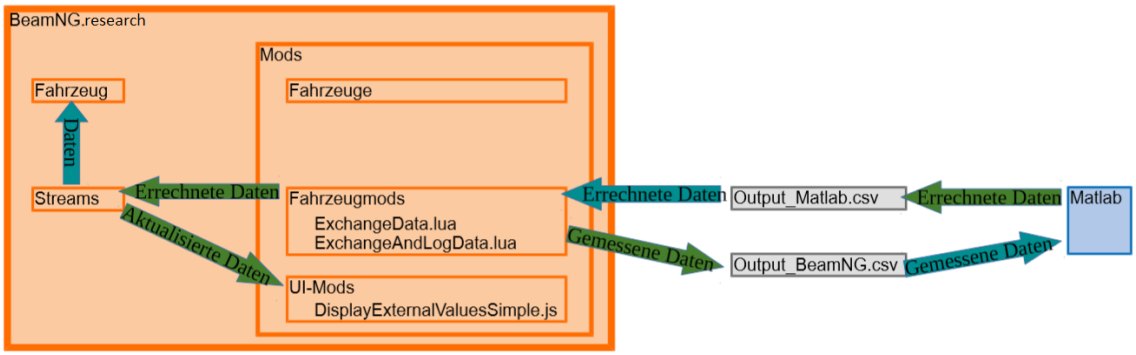

Abbildung 5: Visualisierung des Datenflusses zwischen BeamNG.research und Matlab mit File-based IPC

Die Möglichkeit einer IPC mittels UDP-Socket wurde nicht weiter untersucht (siehe Kapitel 6 Abschnitt 6.3 Ausblick), da die File-based IPC laut Kapitel 2 Unterabschnitt 2.7.3 Energetische Simulation und IPC ausreichend gut funktionierte und daher andere Anforderungen eine höhere Priorität aufwiesen.

Weiterhin sollte die in Abschnitt 3.1 Datenexport konzipierte Anleitung erweitert werden, sodass auch beschrieben wird, wie die Mod modifiziert werden muss, damit die IPC richtig funktioniert. Dazu sollte die nötige Mod ebenfalls gut strukturiert werden.

3.3 Fahrzeug

Entsprechend des Ergebnisses aus Kapitel 2 Unterabschnitt 2.3.3 Fahrzeug sollte ein Fahrzeug mit Elektroantrieb in BeamNG.research eingebunden werden. Dieses sollte basierend auf den Erkenntnissen aus Kapitel 2 Unterabschnitt 2.7.4 Fahrzeug nach Möglichkeit nicht selbst erstellt werden, sondern bereits vorab, z.B. von der Community, erstellt worden sein.

Die Kamerapositionierung des Fahrzeugs sollte so angepasst werden, dass sie ein realistisches Fahrgefühl liefert. Dies sollte mittels explorativem Manipulieren der Parameter geschehen.

Damit nach Abschluss dieser Arbeit noch ggf. weitere benötigte Fahrzeuge hinzugefügt werden können, sollte außerdem die in Abschnitt 3.1 Datenexport konzipierte Anleitung um ein Kapitel erweitert werden. In diesem sollten das Installieren eines Fahrzeugs und das Ändern der Kamerapositionierung sowie die Erstellung einer Fahrzeugkonfiguration erklärt werden.

3.4 Experimentelle Kontrolle

Entsprechend des Ergebnisses aus Kapitel 2 Unterabschnitt 2.7.5 Experimentelle Kontrolle sollte der in Kapitel 2 Unterabschnitt 2.3.4 Experimentelle Kontrolle beschriebene automatische Ablauf eines Experiments mittels eines Szenarios gelöst werden. Da die Erstellung eines Szenarios laut Kapitel 2 Unterabschnitt 2.7.5 Experimentelle Kontrolle Fachkenntnisse benötigt, die laut Kapitel 2 Abschnitt 2.4 Benutzeranalyse nicht anzunehmen sind, sollte die in den vorherigen Abschnitten konzipierte Anleitung um ein Kapitel zur Erstellung eines Szenarios erweitert werden. Aufgrund der Komplexität von Szenarios sollte dieses Kapitel jedoch so geschrieben werden, dass gewisse Programmierkenntnisse zur Umsetzung nötig sind. Der Grund dafür war, dass sich die Erstellung eines Szenarios nicht ausschließlich auf genau erläuterbare Zeilen beschränken ließ. Statt das Kapitel zur Erstellung von Szenarios selbst zu schreiben, sollte die existierende Anleitung (Nadeox1, 2015) um nützliche Hinweise ergänzt werden. Außerdem sollten die Anweisungen ins Deutsche übersetzt werden, da die restliche Anleitung ebenfalls auf Deutsch verfasst werden sollte.

Zusätzlich zu dem Kapitel zur Erstellung von Szenarios im Allgemeinen sollte der Anleitung auch ein Kapitel hinzugefügt werden, in dem mögliche Umsetzungen verschiedener Anforderungen, die nicht in jedem Experiment benötigt werden, erläutert werden sollten. Dies hatte den Grund, dass nicht wie für die Bereiche Datenexport, Energetische Simulation und IPC und Fahrzeug eine Mod erstellt werden konnte, die dann nur noch an das jeweilige Experiment angepasst werden musste.

Die im vorherigen Absatz genannten möglichen Umsetzungen verschiedener Anforderungen mussten zunächst entwickelt werden. Dies sollte, meist mittels Ausprobieren verschiedener denkbarer Lösungsansätze, im Rahmen der Realisierung in Kapitel 4 geschehen. Insbesondere das Setzen der Geschwindigkeit und das Teleportieren eines Fahrzeugs mussten ermöglicht werden.

3.5 UI-Mods

Wie in Kapitel 2 Unterabschnitt 2.7.6 UI-Mods beschrieben, waren bereits fast alle in Kapitel 2 Unterabschnitt 2.3.5 UI-Mods identifizierten Anforderungen an UI-Mods standardmäßig von BeamNG.research erfüllt. Lediglich die Erstellbarkeit von UI-Mods durch Nicht-Informatiker

war nicht erfüllt. Angesichts der Komplexität von Szenarios (vgl. Abschnitt 3.4 Experimentelle Kontrolle) und der Notwendigkeit, UI-Mods und Szenarios aufeinander abzustimmen (siehe Kapitel 2 Unterabschnitt 2.7.5 Experimentelle Kontrolle), wurde entschieden, dass es ausreicht, wenn Informatiker UI-Mods erstellen können. Deshalb sollte lediglich die in Kapitel 2 Unterabschnitt 2.7.6 UI-Mods bereits erwähnt Anleitung zur Erstellung von UI-Mods (My first App, n.d.) in der Anleitung der vorherigen Abschnitte referenziert werden. Außerdem sollte eine dummy Mod erstellt werden, die einfach um die gewünschten Elemente ergänzt werden kann. In der Anleitung sollte erklärt werden, wie die dummy Mod zu modifizieren ist.

4 Realisierung

In diesem Kapitel wird die Realisierung der in Kapitel 3 konzipierten Lösungen beschrieben. Auch dieses Kapitel ist in die Bereiche Datenexport, Energetische Simulation und IPC, Fahrzeug, Experimentelle Kontrolle und UI-Mods unterteilt. Wie in Kapitel 3 konzipiert, wurde eine Anleitung erstellt, die zu allen Bereichen Kapitel enthält. Die Anleitung kann unter Anhang B: Anleitung gefunden werden. Der Anleitung wurde auch ein Kapitel mit Tipps und Tricks hinzugefügt, in dem die in Kapitel 2 Unterabschnitt 2.7.2 Datenexport identifizierten Streams mit ihren Lua-Namen aufgelistet sind, die File-based IPC erläutert wird und nützliche Konsolenkommandos, Codezeilen und Dateien genannt und kurz beschrieben werden. Darüber hinaus wurde eine dummy Fahrzeugmod erstellt, die als Grundgerüst dienen kann, falls eine von den im Rahmen der Abschnitte 4.1 Datenexport und 4.2 Energetische Simulation und IPC erstellten Fahrzeugmods abweichende Funktionalität mittels Fahrzeugmod realisiert werden soll.

4.1 Datenexport

Entsprechend der Konzeption aus Kapitel 3 Abschnitt 3.1 Datenexport wurde die Mod zum Daten loggen strukturiert. Durch sinnvoll platzierte Leerzeilen wurde deutlich gemacht, was zusammen gehört. Außerdem wurden sogenannte Editblöcke eingefügt. Diese beginnen mit der Zeile

```
-- ###Editmarker XX###
```

wobei XX die Nummer des Editblocks ist. In der nächsten Zeile wurde erläutert, was in diesem Editblock getan werden soll. Meist folgen darauf Zeilen die ein Beispiel oder wichtige oder interessante Informationen liefern. In der darauf folgenden Zeile steht

```
-- ***Begin Editblock XX***
```

, wenige Zeilen später folgt eine Zeile

```
-- ***End Editblock XX***
```

. Diese Editblöcke sind so platziert worden, dass ein Benutzer ausschließlich innerhalb dieser Blöcke arbeiten muss. Dadurch soll dem Benutzer eine gewisse Sicherheit im Umgang mit der Mod geboten werden, weil er jederzeit weiß, dass er gerade an der richtigen Stelle in der Mod arbeitet. Der Quellcode der strukturierten Mod ist dem Anhang beigefügt (Anhang E: dataLogging.lua).

Ebenfalls entsprechend der Konzeption aus Kapitel 3 Abschnitt 3.1 Datenexport wurde die zu Beginn von Kapitel 4 erwähnte Anleitung erstellt. Diese beschreibt zunächst, dass zum Loggen der Daten die Mod angepasst werden muss und wo diese zu finden ist. Anschließend erläutert die Anleitung Schritt für Schritt, welche Anpassungen für welche Art von Daten vorgenommen werden müssen. Dabei wird bei jedem Arbeitsschritt, den der Benutzer durchführen muss, auf den entsprechenden Editblock verwiesen. Außerdem wurden häufig Beispiele eingefügt. Innerhalb der Erläuterungen wurden verschiedene Schriftfarben für verschiedene Informationsarten genutzt. Kommentare im Code wurden in grüner Schriftfarbe formatiert, da diese (zumindest mit Notepad++, welches in der Anleitung für die Bearbeitung der Mod empfohlen wird) auch im originalen Code grün sind. Die Namen eines Parameters wurden in blauer Schriftfarbe dargestellt, damit der Benutzer erkennen kann, was zum Parameter gehört und ab wo wieder „normaler“ Code beginnt. Variablen, also Objekte die auch anders benannt werden können als in der Anleitung beispielhaft getan, wurden in violetter Schriftfarbe formatiert. Jeglicher Code, der eins zu eins in der Mod wiedergefunden werden kann, wurde in der Schriftart Consolas geschrieben, damit der Benutzer weiß, dass er nach dieser Zeile Ausschau halten kann, um sich daran zu orientieren. Es wurde Consolas gewählt, weil dies die von den meisten einfachen Texteditoren (insbesondere auch die von Notepad++) verwendete Schriftart ist.

In einem weiteren Kapitel wurden die Parameter, die laut Kapitel 2 Unterabschnitt 2.7.2 Datenexport ausgelesen werden können aufgelistet. Die Parameter wurden dabei nach Gebieten aufgeteilt, nämlich Fahrphysik, Energiedynamik, Position und Sonstige, damit der Benutzer schnell die benötigten Parameter finden kann. Auch die Einheiten, in denen die Parameter geloggt werden, wurden in den Tabellen vermerkt, ebenso wie kurze Anmerkungen, sofern solche nötig waren. Auch wie innerhalb der Mod kleine Berechnungen durchgeführt werden können, um z.B. die Einheit umzurechnen oder einen nur indirekt aus zwei Parametern erhältlichen Parameter zu berechnen, wurde in der Anleitung erläutert.

Auch wie die Mod genutzt werden kann, sowohl durch manuelles Aktivieren als auch in einem Szenario (siehe Abschnitt 4.4 Experimentelle Kontrolle), wurde in einem Abschnitt in der Anleitung erläutert.

Außerdem wurde der Anleitung ein Kapitel hinzugefügt, in dem auf häufige Probleme beim Loggen der Daten eingegangen wird. Auch mögliche Lösungen für diese Probleme wurden an dieser Stelle erklärt. Entsprechend der Konzeption aus Kapitel 3 Abschnitt 3.1 Datenexport wurde weitestgehend auf Informatikvokabular verzichtet.

Weil Mods jeweils nur für bestimmte, explizit angegebene Fahrzeuge genutzt werden können (vgl. Kapitel 2 Unterabschnitt2.7.1 Erläuterungen zu BeamNG.research), wurde eine .bat Datei geschrieben, welche die Mod zum Loggen der Daten (und auch weitere, in folgenden Abschnitten erstellte Mods) für alle installierten Fahrzeuge konfigurierte, indem sie die entsprechenden Dateien erzeugt.

4.2 Energetische Simulation und IPC

Entsprechend der Konzeption aus Kapitel 3 Abschnitt 3.2 Energetische Simulation und IPC wurde eine Mod geschrieben, die eine File-based IPC ermöglicht. Dazu wurde eine Kopie der Mod zum Loggen von Daten (vgl. Abschnitt 4.1 Datenexport) um die Ausgabe einer zweiten .csv Datei mit dem Namen ‚output_BeamNG.csv' ergänzt. Die Mod wurde auf ähnliche Weise strukturiert wie die Mod zum Loggen der Daten (siehe Abschnitt 4.1 Datenexport). Die Anleitung zum Loggen der Daten wurde um einen Abschnitt zum Versenden und Empfangen von Daten mittels File-based IPC ergänzt. Entsprechend Kapitel 2 Abschnitt 2.4 Benutzeranalyse wurde auch dieser Abschnitt ohne Verwendung von Fachvokabular geschrieben.

Der Anleitung wurde außerdem ein Kapitel hinzugefügt, wie die File-based IPC funktioniert, da über simple Einstellungen in der Mod nicht alle sondern nur die häufigsten Nutzungen der erhaltenen Daten aktiviert werden können. In diesem zusätzlichen Kapitel wurde, teilweise unter Verwendung von Fachvokabular, beschrieben, wie die Mod funktioniert, damit ein Informatiker beliebige Änderungen vornehmen kann.

Während der Realisierung der Mod zum Datenaustausch wurde zunächst ein Matlabskript geschrieben, welches die Werte von BeamNG.research in einer einstellbaren Frequenz entgegennimmt und daraus berechnete dummy Werte zurückgibt. Später wurde dieses dummy Skript

genutzt, um die Anbindung des echten Energiemodells zu testen. Sowohl die Anbindung als auch die tatsächliche IPC funktionierten. Allerdings war das Energiemodell nicht auf eine Echtzeitnutzung ausgelegt, sondern erwartete sämtliche Werte zu Beginn der Berechnungen. Dieses Problem wurde zunächst temporär gelöst, indem die Simulation für jeden neuen Wert neugestartet wurde. Dies führte dazu, dass eine Frequenz von mehr als 1 Hertz nicht möglich war. Später sollte mittels einer Toolbox eine Echtzeitlauffähigkeit des Matlabskriptes getestet werden, dies geschah jedoch nicht mehr während dieser Arbeit.

4.3 Fahrzeug

Entsprechend der Konzeption aus Kapitel 3 Abschnitt 3.3 Fahrzeug wurden Elektrofahrzeuge aus dem BeamNG.drive Forum (Land | BeamNG, n.d.) gesucht. Es wurden fünf gefunden, von denen drei direkt als ungeeignet identifiziert wurden, weil sie kein repräsentatives Mittelklasse Fahrzeug darstellen, wie es laut Kapitel 2 Unterabschnitt 2.3.3 Fahrzeug benötigt wurde. Die ausgeschiedenen Fahrzeuge sind der Ibishu Condensa Electric (SushiPro, 2016), der Civetta Bolid-E (AdamB, 2016a) und der ETK K-series Electric (AdamB, 2016c). Der Ibishu 200EX (SushiPro, 2017) und der Hirochi Sunburst Electric (AdamB, 2016b) wurden näher betrachtet. Es wurde schließlich das Modell Ibishu 200EX gewählt, weil der Sound des Sunburst unrealistisch klingt.

Bezüglich der Kamera wurde entsprechend Kapitel 3 Abschnitt 3.3 Fahrzeug mit den Parametern herumgespielt, bis eine realistische Kopfposition gefunden war. Dabei wurden die identifizierten Werte notiert und zurückgesetzt. Dies wurde viermal wiederholt. Das Ergebnis war, dass dreimal die exakt gleichen Werte ermittelt wurden, und einmal nahezu identische Werte. Problematisch an der verwendeten ‚onboard.driver' Kamera war, dass die Kamera die Kopfbewegungen des Fahrers zu simulieren versucht und sich deshalb in Kurven nach links bzw. rechts dreht und bei einem starken Bremsen oder Beschleunigen nach vorne bzw. hinten wippt. Dies wirkt unrealistisch. Deshalb wurde nach einem vergeblichen Versuch, die Eigenbewegung der Kamera zu deaktivieren, stattdessen die Kamera ‚onboard.hood' verwendet. Diese soll eigentlich eine Kamera auf der Motorhaube simulieren und musste entsprechend weit nach hinten verschoben werden. Eine ausreichend starke Verschiebung ist nicht in den Einstellungen möglich. Deshalb wurde die Verschiebung in der Datei ‚\lua\ge\extensions\core\cameraModes\onboard.lua' vorgenommen. Bei einer Verschiebung von -0.341, -0.625, -0.065 und einer Rotation von 4.5, -4, 0 wurde die Kamera als realistisch eingeschätzt.

Der Anleitung wurden ein Abschnitt zum Hinzufügen eines Fahrzeugs und ein Abschnitt zum Modifizieren der Kamera hinzugefügt.

4.4 Experimentelle Kontrolle

Bevor richtig mit der Realisierung in dem Bereich der experimentelle Kontrolle begonnen werden konnte, musste eine Einarbeitung in Szenarios stattfinden. Diese wurde dadurch erschwert, dass das Programm gelegentlich den verwendeten Speicherort wechselte, wodurch Änderungen an der zuvor noch relevanten Datei plötzlich keinen Einfluss auf das Szenario hatten. Entsprechend der Konzeption aus Kapitel 3 Abschnitt 3.4 Experimentelle Kontrolle wurde der Anleitung ein Kapitel zur Erstellung eines Szenarios hinzugefügt. Dabei wurde das Tutorial aus dem Forum (Nadeox1, 2014) übersetzt übernommen und um wichtige Hinweise und nützliche Tipps ergänzt. Des Weiteren wurde ein dummy Szenario erstellt, welches als Grundgerüst für die Erstellung eines neuen Szenarios dienen kann.

Ebenfalls entsprechend der Konzeption aus Kapitel 3 Abschnitt 3.4 Experimentelle Kontrolle wurden Möglichkeiten gesucht, die in Kapitel 2 Abschnitt 2.3.4 Experimentelle Kontrolle identifizierten Anforderungen zu erfüllen. Zu diesem Zweck wurden mehrere Szenarios erstellt, in denen mittels Ausprobieren, teilweise unter Zuhilfenahme des Forums (BeamNG, n.d.), mögliche Umsetzungen der Anforderungen prototypisch realisiert wurden. Diese möglichen Umsetzungen wurden nach einem kurzen Feedback durch den Betreuer dieser Arbeit und eine Doktorandin, die Experimente mit dem Fahrsimulator durchzuführen plante, in die Anleitung aufgenommen.

4.4.1 Trigger

Nachdem die Erstellung eines Szenarios erlernt war, wurde als erste Anforderung ein Trigger angegangen, der an einem bestimmten Ort ein Ereignis auslöst. Dazu wurden zunächst die vielfach in originalen Szenarios verwendeten Waypoints betrachtet. Dabei wurde jedoch ein Bug entdeckt, durch den Waypoints manchmal sichtbar waren, obwohl ihr Attribut ‚hidden' auf 1 gesetzt wurde. Als Alternative wurden BeamNGTrigger bzw. Lua Trigger (in der .prefab Datei heißen sie BeamNGTrigger, im World Editor Lua Trigger) identifiziert. Diese wurden im Scenetree des World Editor unter Library→BeamNG gefunden, können außerhalb des World Editor

nicht sichtbar sein und funktionieren für ortsgebundene Events genauso gut wie bzw. aufgrund der Unsichtbarkeit sogar besser als Waypoints.

Wird ein Lua Trigger durchfahren, wird in der .lua Datei des Szenarios die Funktion `onBeamNGTrigger(data)` aufgerufen. In dieser kann abgefragt werden, welcher Lua Trigger durchfahren wurde und festgelegt werden, was bei jedem Lua Trigger jeweils geschehen soll.

Zeit-basierte Trigger konnten unter Zuhilfenahme der Luafunktion `os.time()` realisiert werden. Geschwindigkeits-basierte Trigger wurden mittels einer selbst geschriebenen Funktion `transmitSpeed(speed)` realisiert. Dies ist eine Funktion, die das Fahrzeug aufrufen kann, um dem Szenario die aktuelle Geschwindigkeit zu übermitteln. Verschiedene Arten von Triggern lassen sich problemlos miteinander kombinieren.

4.4.2 Fahrzeugkontrolle

Bezüglich der Fahrzeugkontrolle sollten laut Kapitel 3 Abschnitt 3.4 Experimentelle Kontrolle zwei wichtige Anforderungen betrachtet werden. Die erste der beiden war, dass das Fahrzeug an einen beliebigen Ort auf der Karte teleportiert werden können sollte. Nach einigem Austesten in der Konsole wurde der Befehl `vehicleSetPositionRotation(...)` gefunden. Dieser ermöglicht es, das Fahrzeug zu teleportieren und zu rotieren. Einer der benötigten Parameter ist die ID des Fahrzeugs, diese kann mittels `be:getPlayerVehicleID(0)` ermittelt werden (jojos38, 2017). Anfänglich zeigte das Fahrzeug ein inkonsistentes und nicht erklärbares Verhalten: Manchmal behielt es seine Geschwindigkeit und manchmal verlor es sie. Schließlich wurde identifiziert, dass die Geschwindigkeit verloren ging, wenn das Fahrzeug beim Teleportieren auch rotiert wurde. Dies wurde zunächst nicht erkannt, weil bei einem Aufruf des Befehls mit denselben Rotationsangaben wie beim vorherigen Ausführen des Befehls keine Rotation stattfindet, unabhängig davon, ob das Fahrzeug seit dem letzten Aufruf des Befehls manuell (also durch Fahren des Fahrzeugs) rotiert wurde.

Die zweite Anforderung die es im Bereich Fahrzeugkontrolle zu lösen galt, war, dass die Geschwindigkeit eines Fahrzeugs gesetzt werden können sollte. Dies wurde jedoch von einem Mitarbeiter der BeamNG GmbH als unmöglich eingeordnet (Stiegfried, 2018b). Da dies bereits vermutet wurde, wurde bereits vor der Antwort in dem Thread nach einem möglichen Workaround gesucht. Der gefundene Workaround wurde später auch im genannten Thread vorgeschlagen. Er beruht darauf, das Fahrzeug auf einer freien Fläche automatisch auf die ge-

wünschte Geschwindigkeit bringen zu lassen und es dann an den gewünschten Ort zu teleportieren, was, wie zuvor erläutert, möglich ist.

Dazu waren einige Punkte zu berücksichtigen. Das Beschleunigen des Fahrzeugs wird durch Setzen des Streams `electrics.values.throttleOverride = 1` umgesetzt, da das Fahrzeug so permanent einen Gasinput von 1 verwendet, unabhängig vom physischen Input. In der Datei ‚\lua\vehicle\controller\vehicleController.lua' wurde analog zum standardmäßig vorhandenem throttleOverride ein brakeOverride hinzugefügt. Dieser wird während der Beschleunigung auf 0 gesetzt, so dass unabhängig vom physischen Input ein Bremswert von 0 genutzt wird. Da die Teleportation keine Rotation durchführen darf, war es wichtig, das Fahrzeug vor beginn der Beschleunigung richtig zu rotieren. Dies stellte kein Problem dar. Außerdem musste verhindert werden, dass der Fahrer während der Beschleunigung lenkt. Dazu wurden in der Datei ‚\lua\vehicle\input.lua' in der Funktion `update(dt)` die Zeilen

```
if electrics.values.disableSteeringInput==1 then
      electrics.values['steering_input'] = 0
end
```

hinzugefügt. Mit Hilfe dieser Modifikation ist es möglich, ein manuelles Lenken zu verhindern, indem der Wert `electrics.values.disableSteeringInput = 1` gesetzt wird.

Um zu bestimmen, ob die gewünschte Geschwindigkeit erreicht ist, müssen das Szenario und das Fahrzeug miteinander kommunizieren. Es wurde dazu im Szenario eine Funktion geschrieben, welche vom Fahrzeug aufgerufen werden kann und die aktuelle Geschwindigkeit übermittelt. Sobald die gewünschte Geschwindigkeit erreicht ist, wird (in der ersten Version, siehe nächsten Absatz) die Simulation pausiert. Falls nicht genug Platz vorhanden sein sollte, um die gewünschte Geschwindigkeit zu erreichen, kann das Fahrzeug nach Erreichen des Endes der Beschleunigungsstrecke wieder an den Anfang dieser teleportiert werden.

Der Fahrer (= Versuchsteilnehmer) soll von dem Beschleunigungsvorgang nichts mitbekommen. Deshalb ist es nötig, die Sicht zu blockieren und den Sound für die Zeit zu deaktivieren. Zum Deaktivieren des Sounds wurde der Befehl `settings.setValue("AudioMasterVol",0,nil,nil)` mit Hilfe der Konsolenkommandos `dump(settings)` und `dump(settings.getValues())` gefunden, nachdem in den vielversprechenden Dateien ‚lua\ge\ge_utils.lua' und ‚lua\vehicle\sounds.lua' keine Lösung gefunden wurde. Dieser Befehl ermöglicht es, die Masterlautstärke zu setzen. Um die Sicht zu blockieren wurde eine UI-Mod

geschrieben, welche gleichzeitig die Zeit, die für die Beschleunigung benötigt wird, mit neuen Anweisungen, die der Versuchsteilnehmer lesen muss, überspielen kann. Weil UI-Mods nicht aktualisiert werden können während die Fahrt pausiert ist, wurde stattdessen mit dem Befehl `bullettime.set(1/1000)` die Simulationsgeschwindigkeit auf 1/1000 gesetzt. Je nach Gestaltung der Karte erhält der Versuchsteilnehmer mehrere Minuten bis Stunden Zeit um die Anweisungen zu lesen. Welcher Text in der UI-Mod angezeigt wird und ob diese überhaupt sichtbar sein soll, wird mittels selbstdefinierten Streams geregelt.

Beim Erreichen der gewünschten Geschwindigkeit wird in der UI-Mod ein Button zum Starten der Fahrt sichtbar. Dieser aktiviert den Sound wieder, blendet die UI-Mod aus, setzt die Simulationsgeschwindigkeit wieder auf 1 und löscht die Werte für throttleOverride, brakeOverride und disableSteeringInput, sodass der Versuchsteilnehmer fahren kann, als wäre das Fahrzeug lediglich auf die Geschwindigkeit gesetzt worden.

4.4.3 Sonstiges

Die zentrale Anforderung im Bereich experimentelle Kontrolle lautet, dass ein Experiment von Anfang bis Ende vollautomatisch ablaufen muss. Genau für solche Fälle sind Szenarios in BeamNG.research integriert. Allerdings ist in den Szenarios nicht angedacht, dass verschiedene Sequenzen nacheinander ablaufen sollen, sondern dass jedes Szenario genau eine Aufgabe für den Fahrer enthält. Dies ließ sich jedoch mittels einiger Kontrollvariablen realisieren. Diese speichern, welcher Durchlauf aktuell gefahren wird u.Ä.. Auch eine Teleportation zu Beginn einer Sequenz lässt sich leicht durchführen (siehe Unterabschnitt 4.4.2 Fahrzeugkontrolle).

Gemäß Kapitel 2 Unterabschnitt 2.3.4 Experimentelle Kontrolle sollte eine Möglichkeit gefunden werden, die Reihenfolge der Sequenzen zu randomisieren. Dies wurde realisiert, indem am Beginn der .lua Datei ein Table (eine Art Array in Lua) die Nummern der Sequenzen, die genutzt werden sollen, eingetragen werden können. Zwei weitere Tables wurden hinzugefügt, um Sequenzen angeben zu können, die in fester Reihenfolge vor den randomisierten Sequenzen bzw. danach abgespielt werden sollen. Zusätzlich wurde eine Funktion zum Randomisieren der Reihenfolge der Elemente eines Tables übernommen (Quelle: Uradamus, 2014), die zu Beginn eines Szenarios aufgerufen wird. Die Sequenzen werden dann in der Reihenfolge, in der die Nummern in dem randomisierten Table vorliegen, abgefahren. Die Sequenzen die so abgefahren werden sollen, müssen in einer .csv Datei hinterlegt werden.

Über die bereits erwähnten Kontrollvariablen, zu denen auch die Nummer der aktuell gefahrenen Sequenz und Informationen wie z.B. die aktuelle Zielgeschwindigkeit gehören, wird in für diesen Zweck erstellten UI-Mods entschieden, welcher Text und welche UI zu jedem Zeitpunkt jeweils sichtbar sein müssen.

Da in einem Szenario keine Fahrzeugmod (z.B. zum Loggen der Daten) aktiviert werden kann während das Szenario läuft, musste eine Fahrzeugkonfiguration (.pc Datei) geschrieben werden, welche den Logger standardmäßig integriert hat. Dazu wurde mit der Garage in BeamNG.drive eine entsprechende Konfiguration erstellt. Dieser Modus existiert extra für das Erstellen von Fahrzeugkonfigurationen, ist jedoch in BeamNG.research nicht vorhanden. Weil das Fahrzeug inklusive der Fahrzeugmod nicht die letzte beim Starten eines Szenarios initialisierte Instanz ist, müssen die geloggten oder ausgetauschten Parameter in einem pcall abgelegt werden. Ein pcall ist eine Luafunktion, welche versucht einen Codeabschnitt auszuführen, bei Fehlern aber nur die Funktion abbricht anstatt die Datei abstürzen zu lassen.

Damit in UI-Mods Eingaben getätigt werden können, wurden standardmäßig vorhandene UI-Mods untersucht. Das Ergebnis war, dass normale HTML Input Elemente genutzt und von dem JavaScriptteil der UI-Mod ausgelesen werden können. Dabei trat jedoch das Problem auf, dass der Tastaturinput nicht für die UI-Mod sondern für das eigentliche Programm (also z.B. als Gas) interpretiert wird. Eine Antwort im Forum (Stiegfried, 2018c) lieferte einen Hinweis, in welcher standardmäßig enthaltenen UI-Mod ein Beispiel für eine Texteingabe per Tastatur gefunden werden kann. Daraufhin wurde die dazu relevante Funktion aus der UI-Mod ‚procTrackUtil' übernommen. Sind die Eingaben getätigt, können diese problemlos in einen Stream geschrieben und so vom Datenlogger erfasst werden.

Gemäß der Analyse aus Kapitel 2 Unterabschnitt 2.3.4 Experimentelle Kontrolle sollte es möglich sein, eine automatische Fahrt durch die KI in eine manuelle Fahrt übergehen zu lassen. Zur Realisierung dieser Anforderung wurde in der UI-Mod ‚AIControl' die Funktionsweise der KI untersucht. Als Ergebnis wurden die folgenden Funktionen zur Steuerung der KI identifiziert und in einem prototypischen Szenario verwendet: `ai.setMode(„manual“)`, `ai.setSpeed(70/3.6)`, `ai.setSpeedMode(„limit“)`, `ai.setTarget(„BigRoard_29“)`, `ai.driveInLane(„on“)`.

Neben den zuvor adressierten Anforderungen, sollte auch eine Möglichkeit gefunden werden, Kollisionen zu deaktivieren, da in den Experimenten am IMIS keine Unfälle verursacht werden

sollen. Das Programm ist nicht darauf ausgelegt, sodass stattdessen eine Möglichkeit zur Kollisionsdetektion gesucht wurde, um eine Sequenz bei einem Unfall automatisch zu beenden und neu zu starten. Dazu wurde die UI-Mod ‚Vehicle Damage App', welche den aktuellen Zustand des Fahrzeug bezüglich potentieller Schäden darstellt, näher betrachtet. Dies lieferte jedoch nicht die gewünschte Information, weshalb die UI-Mod ‚Damage Counter' aus dem Forum (DrivKan, 2017) herangezogen wurde. In ihr wurden die Streamwerte stats.beams_broken und stats.beams_deformed gefunden. Über diese war es möglich, eine Schadensdetektion zu schreiben, welche bei einem durch einen Unfall verursachten Schaden am Fahrzeug das Szenario pausiert. Alternativ kann auch die Sequenz automatisch neu gestartet werden. Mittels Austesten wurden die Schadensgrenzen beams_broken>0 und beams_deformed>19 identifiziert. Bei diesen Werten wurde die Kollisionsdetektion innerhalb einer 20 minütigen Testfahrt mit rasantem Fahrstil nie ohne vorangegangenen Unfall aktiviert. Geringere Werte würden die Detektion empfindlicher machen und potentiell einen Unfall detektieren, obwohl nur rasant gefahren wurde. Bei höheren Werten würden ggf. auch leichte Unfälle als unfallfreie Fahrt klassifiziert werden. Zusätzlich zu der erstellten UI-Mod können in den Optionen von BeamNG.research Kollisionen mit sich selbst und anderen Fahrzeugen deaktiviert werden und bei der Erstellung einer Karte kann den platzierten Objekten durch setzen eines entsprechenden Attributes mitgeteilt werden, dass sie keine Kollisionsdetektion nutzen sollen. Dies hat zur Folge, dass das Objekt einfach durchfahren werden kann, ist jedoch für eine bereits existente Karte sehr aufwändig und wurde daher nur zu Testzwecken bei einigen wenigen Objekten getan.

4.5 UI-Mods

Gemäß der Konzeption aus Kapitel 3 Abschnitt 3.5 UI-Mods wurde in der Anleitung ein Kapitel für UI-Mods hinzugefügt. In diesem wird die Anleitung zur Erstellung einer UI-Mod aus dem Forum (My first App, n.d.) verlinkt.

Des Weiteren wurde eine dummy UI-Mod erstellt. Diese ist eine funktionierende UI-Mod, die jedoch nichts anzeigt und noch mit Inhalt gefüllt werden muss. Wie die dummy Mod verändert werden muss, damit sie tatsächlich genutzt werden kann, wurde ebenfalls in der Anleitung erläutert. Hierfür muss der Name der Mod an verschiedenen Stellen angepasst werden.

5 Evaluation

In diesem Kapitel wird die Evaluation der Ergebnisse dieser Arbeit beschrieben. Da entsprechend Kapitel 1 Abschnitt1.3 Vorgehensweise in regelmäßigen Treffen mit Vertretern der Benutzergruppe „Mitarbeiter“ Rückmeldungen zu Konzepten und Ergebnissen eingeholt wurde, war es nicht nötig, sämtliche Ergebnisse abschließend zu evaluieren. Dennoch wurde eine eher informelle Abschlussevaluation durchgeführt. Das Ziel dieser wird in Abschnitt 5.1 Ziel kurz erläutert. In Abschnitt 5.2 Vorgehen wird kurz aufgezeigt, wie das zuvor vorgestellte Ziel erreicht wurde. Abschließend wird das Ergebnis der Evaluation in Abschnitt 5.3 Ergebnisse dargestellt.

5.1 Ziel

Laut Kapitel 1 Abschnitt 1.1 Ziele der Arbeit war das Ziel der Arbeit, die wichtigsten Funktionalitäten umzusetzen, die ein Fahrsimulator für Untersuchungen in der Nutzer-Energie-Interaktion bieten muss. Ziel der Evaluation war deshalb zu validieren, dass alle wichtigsten Anforderungen erfüllt wurden und nun Experimente mit dem Fahrsimulator am IMIS durchgeführt werden können. Mit dieser Evaluation sollte also einerseits validiert werden, dass in Kapitel 2 Abschnitt 2.3 Anforderungsanalyse alle wichtigsten Anforderungen identifiziert wurden, und andererseits, dass die Konzeption aus Kapitel 3 und insbesondere auch die darauf aufgebaute Realisierung aus Kapitel 4 diese Anforderungen ausreichend gut erfüllen.

5.2 Vorgehen

Um das in Abschnitt 5.1 Ziel erläuterte Ziel zu erreichen, wurde ein reales Experiment vorbereitet. Die Anweisungen, wie das Experiment ablaufen soll, sowie die Texte und andere nicht Fahrsimulator-spezifische Informationen zum Experiment wurden von der Doktorandin bereitgestellt, die auch an den regelmäßigen Treffen teilnahm. In den nachfolgenden Unterabschnitten 5.2.1 Ablauf des Experiments und 5.2.2 Umsetzung werden der Ablauf des Experiments und die Umsetzung unter Zuhilfenahme der Ergebnisse dieser Arbeit beschrieben.

5.2.1 Ablauf des Experiments

In dem Experiment sollen Teilnehmer zunächst einen Fragebogen außerhalb von BeamNG.research ausfüllen. Anschließend sollen die Teilnehmer allgemeine Instruktionen zur Aufgabenstellung erhalten. Als nächstes sollen die Teilnehmer eine Testfahrt absolvieren, um sich mit der Strecke und der Aufgabenstellung vertraut zu machen. Hierbei werden nach einem Kilometer Instruktionen zu Beschleunigungs- und Bremsmanövern gegeben. Daraufhin sollen vier Sequenzen folgen, die vier verschiedene experimentelle Bedingungen repräsentieren. In der ersten sollen die Teilnehmer die Instruktion erhalten, die über Instruktionen angekündigten Beschleunigungs- und Bremsmanöver wie üblich auszuführen. Im zweiten Durchlauf sollen die Teilnehmer die Manöver möglichst energieeffizient fahren. In der dritten und vierten Sequenz soll dazu jeweils eine von zwei Anzeigen eingeblendet werden, die den Teilnehmer dabei unterstützen soll, energieeffizient zu fahren. Welche der beiden Anzeigen zuerst getestet wird, soll zufällig entschieden werden. Die verschiedenen Start- und Zielgeschwindigkeiten der Manöver sollen in einer .csv Datei hinterlegt werden und in jeder Sequenz in einer neu randomisierten Reihenfolge durchgeführt werden. Die Länge der .csv Datei soll veränderbar sein, ohne dass dies einen Einfluss auf den grundsätzlichen Ablauf des Experiments hat. Jede Sequenz soll aus zwei Blöcken bestehen, wobei in jedem Block jeweils alle Zeilen der .csv Datei genutzt werden sollen. Während des gesamten Experiments sollen Daten geloggt werden. Bei diesen Daten handelt es sich um die Zeit, Position, Geschwindigkeit, Momentanverbrauch, Informationen dazu, welche Sequenz, welcher Block, welche Zeile und zugehörige Spalte (Start- oder Zielgeschwindigkeit) der .csv Datei gerade gefahren wird und ob Anzeige A oder B gerade angezeigt wird.

5.2.2 Umsetzung

Für das Experiment wurde ein Szenario erstellt. Zum Darstellen der Aufgabenstellung wurde eine UI-Mod geschrieben, das Darstellen der aktuellen Zielgeschwindigkeit wird von einer weiteren, dafür erstellten, UI-Mod übernommen (siehe Abbildung 6). Eine dritte UI-Mod dient als Platzhalter für die zu testenden Anzeigen A und B. Um die gefahrene Distanz zu messen, wurden in der Mod zum Loggen der Daten die Variable `distanceTravelled` definiert und folgende Zeilen ergänzt:

```
distanceTravelled = distanceTravelled + electrics.values.wheels-
peed*(timer-timerOld)

electrics.values.distanceTravelled = distanceTravelled
```

. Im Szenario wurde die Funktion `setKmCompleted()` hinzugefügt. Bevor der Kilometer in der Testfahrt gefahren wurde, fragt das Szenario bei jeder Aktualisierung der Werte die gefahrene Distanz des Fahrzeugs ab. Sobald ein Kilometer gefahren wurde, setzt die Funktion die entsprechende Kontrollvariable im Szenario und in dem Stream `electrics.values`, sodass die Anzeige mit der Geschwindigkeitsanweisung sichtbar wird und für alle nachfolgenden Sequenzen von Anfang an ist. Die darzustellenden Geschwindigkeiten werden zu Beginn des Szenarios aus einer .csv Datei eingelesen. Wird diese um weitere Zeilen ergänzt oder werden vorhandene gelöscht, muss im Szenario lediglich eine Zeile angepasst werden. Zu Beginn jedes Blocks wird eine zufällige Reihenfolge erstellt, in der die Zeilen abgefahren werden.

Abbildung 6: UI-Mod zur Darstellung der Geschwindigkeitsanweisung

Sobald die Kontrollvariable zum Anzeigen der Geschwindigkeitsanweisung gesetzt wurde, fragt das Szenario bei jeder Aktualisierung der Werte die Geschwindigkeit des Fahrzeugs ab. Wird die geforderte Geschwindigkeit (auf Integer gerundet) registriert, startet ein zwei Sekunden Timer. Wird eine Geschwindigkeit von +/- 5km/h um die geforderte Geschwindigkeit eingehalten, wird die nächste Geschwindigkeit angezeigt. Über weitere Kontrollvariablen wird geregelt, welche Zeile und Spalte gefahren wird und wann ein Block und wann eine experimentelle

Bedingung abgeschlossen ist. Nachdem alle Zeilen der .csv Datei abgefahren wurden, beginnt der nächste Block. Die Anzahl der Blöcke pro Bedingung kann im Szenario mit einer Konstanten festgelegt werden. Wenn eine experimentelle Bedingung abgeschlossen ist, wird über die UI-Mod, die die Aufgabenstellung darstellt, die neue Aufgabenstellung der nächsten Bedingung dargestellt. Es wurde eine Funktion des Szenarios an die B-Taste des Lenkrad gebunden, sodass mit besagter Taste der Bildschirm mit der Aufgabenstellung geschlossen und die nächste Sequenz begonnen werden kann. Ob zuerst Anzeige A oder B angezeigt wird, wird zu Beginn des Szenarios zufällig entschieden. Ob eine der Anzeigen aktuell angezeigt werden soll, wird mittels Kontrollvariablen entschieden. Die .lua Datei des Szenarios kann unter Anhang F: example_scenario.lua gefunden werden. Die weiteren Dateien sind lediglich auf der CD hinterlegt, die diesem Dokument beiliegt.

Die Mod zum Loggen der Daten wurde so angepasst, dass die meisten der geforderten Daten geloggt werden. Lediglich der Momentanverbrauch kann nicht geloggt werden, siehe Kapitel 6 Abschnitt 6.2 Offene Punkte.

5.3 Ergebnisse

In Unterabschnitt 5.2.2 Umsetzung ist erkennbar, dass, mit Ausnahme des Momentanverbrauchs, alle laut Unterabschnitt 5.2.1 Ablauf des Experiments benötigten Anforderungen unter Zuhilfenahme der Ergebnisse dieser Arbeit gelöst werden können und das beschriebene Experiment umgesetzt werden konnte. Von der Doktorandin wurde bestätigt, dass das Experiment gut umgesetzt wurde und für die geplante wissenschaftliche Untersuchung genutzt werden kann.

Aus diesem Ergebnis lässt sich schließen, dass die wichtigsten Anforderungen an BeamNG.research als Fahrsimulationsumgebung für Untersuchungen in der Nutzer-Energie-Interaktion erfüllt wurden (mit Ausnahme des Momentanverbrauchs) und somit das Ziel dieser Arbeit erreicht wurde.

6 Zusammenfassung und Ausblick

Dieses Kapitel bildet den Abschluss dieser Arbeit. In Abschnitt 6.1 Zusammenfassung wird kurz zusammengefasst, was im Rahmen dieser Arbeit erreicht wurde. In 6.2 Offene Punkte werden Aufgaben benannt, die während dieser Arbeit hätten bearbeitet werden sollen, jedoch noch nicht wurden. Dabei ist zu beachten, dass das Ziel dieser Arbeit nicht notwendigerweise die Erfüllung aller Anforderungen war, da der Aufwand sich im Voraus nur schlecht abschätzen ließ. In Abschnitt 6.3 Ausblick wird ein kurzer Ausblick gegeben, was, abgesehen von den offenen Punkten, noch am Fahrsimulator im Bereich Software getan werden kann, um zukünftig auch komplexere Experimente zu ermöglichen.

6.1 Zusammenfassung

Ziel dieser Arbeit war es, Experimente im Bereich der Nutzer-Energie-Interaktion am Fahrsimulator des IMIS zu ermöglichen. Dazu wurde zunächst ein Anforderungsdokument erstellt, welches auch nach der anfänglichen Anforderungsanalyse noch erweitert wurde und schließlich 82 Anforderungen umfasst, die in neun verschiedene Bereiche unterteilt sind. In dieser Arbeit wurden die wichtigsten Anforderungen angegangen, wobei sie nach den fünf Bereichen Datenexport, energetische Simulation und IPC, Fahrzeug, experimentelle Kontrolle und UI-Mods kategorisiert wurden.

Im Bereich Datenexport wurden die nötigen Parameter samt deren zum Loggen nötigen Lua-Namen gesammelt. Außerdem wurde die Mod, die das Loggen der Daten übernehmen sollte, strukturiert und eine Anleitung geschrieben, wie die Mod modifiziert werden muss um die gewünschten Parameter zu loggen.

Im Bereich der energetischen Simulation und IPC wurde eine Möglichkeit für eine File-based IPC konzipiert und realisiert. Dazu wurde eine Fahrzeugmod geschrieben, die Daten in eine .csv Datei schreibt und aus einer anderen einliest. Außerdem wurde ein entsprechendes Gegenstück in Matlab prototypisch umgesetzt. Die Konfiguration dieser Mod wurde ebenfalls in der

Anleitung erläutert. Des Weiteren wurden Geschwindigkeitswerte einer realen Fahrt eingelesen und umgesetzt.

Es wurde ein repräsentatives Elektrofahrzeug installiert, welches von einem Mitglied der BeamNG.drive Community erstellt worden war. Außerdem wurde die Kamera in eine realistische Kopfposition verschoben und die Eigenbewegungen zur Simulation der Kopfbewegungen des Fahrers deaktiviert.

Im Bereich der experimentellen Kontrolle wurde zunächst mittels eines iterativ ausgearbeiteten Dokuments der Ablauf eines möglichen Experiments erarbeitet. Anschließend wurden diverse Funktionalitäten realisiert, wobei der automatische Ablauf eines Experiments im Fokus stand. Dazu wurden eine Möglichkeit für die Teleportation des Fahrzeugs und ein Workaround zum Festsetzen der Startgeschwindigkeit identifiziert. Außerdem wurde erarbeitet, wie Ereignisse (wie z.B. das Einblenden einer Instruktion) an diverse Trigger (Zeit, Ort, Geschwindigkeit, gefahrene Distanz) gekoppelt werden können. Es wurde auch ermöglicht, die Sicht zu blockieren und den Sound zu deaktivieren, z.B. während Anweisungen dargestellt werden. Es wurde eine Möglichkeit aufgezeigt, mittels Kontrollvariablen den gesamten Ablauf eines Experiments zu steuern. Weiterhin wurde eine Kollisionsdetektion geschrieben, um einen Versuch bei einem Unfall neu zu starten.

Im Bereich der UI-Mods wurde festgestellt, dass hier lediglich der Zusammenhang von UI-Mods und anderen Instanzen in BeamNG.research erläutert werden musste. Außerdem wurden diverse UI-Mods für andere Bereiche erstellt.

6.2 Offene Punkte

Es wurde im Rahmen dieser Arbeit noch nicht geschafft, das unnatürliche Wackeln des Fahrzeugs bei niedrigen Geschwindigkeiten zu beseitigen. Es wurde jedoch ein Hinweis gegeben, dass die zu geringe Anzahl der Speichen im Rad verantwortlich sei (siehe Kapitel 2 Unterabschnitt 2.7.4 Fahrzeug). Basierend auf diesem Ergebnis muss das Wackeln des Fahrzeugs noch auf ein akzeptables Maß reduziert werden.

Außerdem konnte das Matlabskript, welches die Energiesimulation extern übernehmen sollte, noch nicht in einer Weise angebunden werden, dass es tatsächlich zur Laufzeit verwendet werden könnte. Dazu müssen wahrscheinlich Matlab Toolboxen zur Echtzeitanwendung von Simu-

link Simulationen untersucht werden, was im Rahmen dieser Arbeit aus zeitlichen Gründen nicht mehr realisiert wurde.

Informationen zu einem Experiment müssen nach aktuellem Stand in diversen verschiedenen Dateien hinterlegt werden, damit sie von BeamNG.research genutzt werden können. Es wurde zeitlich nicht geschafft, sämtliche Angaben zu einem Experiment an einem Ort zu bündeln und die Informationen von dort auch zu nutzen.

Es wurde noch nicht untersucht, wie UIs ins Cockpit integriert werden können. Aus Zeitgründen und aufgrund der besseren Sichtbarkeit und Lesbarkeit wurde zunächst nur untersucht, wie UIs in Form von UI-Mods dargestellt werden können.

Vor Beginn dieser Arbeit war nicht geplant, Szenarios für die Experimente zu nutzen. Dies erwies sich im Laufe der Arbeit jedoch als nötig. Szenarios sollten ursprünglich aufgrund der vergleichsweise hohen Komplexität vermieden werden. Dass nun doch Szenarios verwendet werden mussten, hatte zur Folge, dass nicht sämtliche Schritte zur Vorbereitung eines Experiments ausreichend detailliert erläutert werden konnten, dass auch Nicht-Informatiker ohne Hilfe ein Experiment erstellen könnten.

6.3 Ausblick

Das Anforderungsdokument enthält noch viele ungelöste Anforderungen, die im Rahmen dieser Arbeit aus Zeitgründen gar nicht angegangen werden sollten, weil sie eine geringere Priorität aufweisen. Zur Weiterentwicklung des Fahrsimulators am IMIS ist es jedoch nötig, auch diese Anforderungen noch zu realisieren.

In dieser Arbeit wurde eine File-based IPC realisiert. Es wurde jedoch auch eine potentielle Möglichkeit für eine IPC mittels UDP-Socket identifiziert. Diese könnte näher untersucht werden.

Zu Beginn dieser Arbeit funktionierte die Mod „Log all available Streams“ nicht. Dies hat sich kurz vor Ende dieser Arbeit geändert. Die Mod eignet sich zwar wahrscheinlich nicht für Untersuchungen am IMIS, liefert allerdings viele Daten inklusive Herkunft, von denen vielleicht einige noch interessant sein könnten. Die Daten dieser Mod könnten also mal untersucht werden.

Vielleicht könnten so weitere Parameter identifiziert werden, von denen aktuell unbekannt ist, ob sie direkt auslesbar sind.

Das Energiemodell liegt aktuell nur in Matlab vor. Es wäre jedoch denkbar, das Energiemodell zukünftig in BeamNG.research zu integrieren, um so die Latenz zu verbessern, indem die IPC (für die energetische Simulation) überflüssig gemacht wird. Dazu könnten z.B. die Verbrauchsdaten des elektrischen ETK800, der während dieser Arbeit noch nicht existierte, siehe Kapitel 2 Unterabschnitt 2.7.4 Fahrzeug, an das Energiemodell aus Matlab angepasst werden.

In dieser Arbeit wurde identifiziert, dass UI-Mods mit HTML, JavaScript und CSS Kenntnissen ohne Probleme erstellt werden können. Basierend auf dieser Erkenntnis müssen noch Anzeigen erstellt werden, die in Experimenten gegeneinander getestet werden können.

Die Position des Fahrzeugs lässt sich feststellen und loggen. Es könnte untersucht werden, wie die relative Position des Fahrzeugs zur Straße ermittelt werden kann, um zu messen, wie stark ein Versuchsteilnehmer von der Mittellinie abweicht oder ob er vielleicht sogar die Straße verlässt. Dazu könnte es hilfreich sein, eine parametrische Generierung einer Strecke anzugehen (sieh Kapitel 2 Unterabschnitt 2.7.7 Strecke).

Es könnte recherchiert oder ggf. untersucht werden, ob eine dynamische Veränderung des FOV in Abhängigkeit von der aktuellen Geschwindigkeit zu einer besseren Immersion führt. Falls ja könnte dies noch umgesetzt werden, da diese Funktion nicht standardmäßig in BeamNG.research vorhanden ist.

Im Rahmen dieser Arbeit wurde aufgrund der Komplexität von Szenarios keine Möglichkeit realisiert, Szenarios, und damit Experimente, von Nicht-Informatikern erstellbar zu machen. Es wäre denkbar, ein Tool zu entwickeln, in welchem man mittels Eingabefeldern wie z.B. Dropdownmenüs oder Zahlenfeldern die Parameter eines Experiments vorgeben könnte. Dieses Tool könnte dann ein passendes Szenario erstellen, in dem nur noch die ggf. zu testenden Anzeigen in Form von .html Dateien hinzugefügt werden müssten. Ein solches Tool zu entwickeln wäre jedoch wahrscheinlich ziemlich aufwändig, eventuell aufwändig genug um eine vollständige Qualifikationsarbeit zu füllen, würde die Vorbereitung von Experimenten jedoch stark vereinfachen.

Abbildungen

Tabellen

Quellen

Literatur

Bock, O., Drescher, U., Janouch, C., Haeger, M., van Winsum, W. & Voelcker-Rehage, C. (2018). *An experimental paradigm for the assessment of realistic human multitasking.* London: Springer-Verlag London Ltd.. doi: 10.1007/s10055-018-0342-7

Degirmenci, K. (2018). *Toward a Gamified Mobile Application to Improve Eco-Driving: A Design and Evaluation Approach*. New Orleans, LA: Americas Conference on Information Systems (AMCIS 2018).

Franke, T., Neumann, I., Bühler, F., Cocron, P. & Krems, J.F. (2012). Experiencing Range in an Electric Vehicle: Understanding Psychological Barriers. *Applied Psychology*, (S. 368-391).

Grande, M. (2014). *100 Minuten für Anforderungsmanagement Kompaktes Wissen nicht nur für Projektleiter und Entwickler* (2nd ed.). Wiesbaden: Springer Fachmedien Wiesbaden GmbH. doi: 10.1007./978-3-658-06435-8

Hiraoka, T., Terakado, Y., Matsumoto, S. & Yamabe, S. (2009). *Quantitative Evaluation of eco-driving on fuel consumption based driving simulator experiments*. 16th World Congrss on Intelligent Transport Systems.

International Organization for Standardization. (2010). *Human-centred design for interactive systems* (ISO Standard No. 9241-210).

McCollum, D., Krey, V., Kolp, P., Nagai, Y. & Riahi, K. (2014). Transport electrification: A key element for energy system transformation and climate stabilization. *Climatic Change* (S. 651-664).

Norman, D.A. & Draper, S.W. (1986). *User Centered System Design.* Hillsdale: Lawrence Erlbaum Associates.

Pavlou, D., Papadimitriou, E., Antoniou, C., Papantoniou, P., Yannis, G., Golias, J. & Papageorgiou, S.G. (2015). *Driving Behaviour of Drivers with mild Cognitive Impairment and Alzheimer's Disease: A Driving Simulator Study.* Washington: Proceedings of the 94th Annual meeting of the Transportation Research Board.

Wilcox-Netepczuk, D. (2013). Immersion and Realism in Video Games – The Confused Moniker of Video Game Engrossment. The 18th International Conference on Computer Games. doi: 10.1109/CGames.2013.6632613

Weblinks

About Us – BeamNG [Website]. (n.d.). Retrieved from https://beamng.gmbh/about-us/

AdamB. (2016a, January 22). Civetta Bolid-E (Electric) [Mod]. Retrieved August 13, 2018, from https://www.beamng.com/resources/civetta-bolid-e-electric.1648/

AdamB. (2016b, August 7). Hirochi Sunburst Electric [Mod]. Retrieved August 13, 2018, from https://www.beamng.com/resources/hirochi-sunburst-electric.1220/

AdamB. (2016c, December 16). ETK K-series Electric [Mod]. Retrieved August 13, 2018, from https://www.beamng.com/resources/etk-k-series-electric.1675/

BeamNG [Forum]. (n.d.). Retrieved from https://www.beamng.com/forums/

BeamNG.drive | BeamNG [Website]. (n.d.). Retrieved from https://www.beamng.com/

BeamNG.research – BeamNG [Website]. (n.d.). Retrieved from https://beamng.gmbh/research/

City Car Driving – Car Driving Simulator, PC Game [Website]. (n.d.). http://citycardriving.com/

Driving simulator for research and training | [Website]. (n.d.). Retrieved from https://cs-driving-simulator.com/research-driving-simulator/

DrivKan. (2017, October 27). Damage Counter [Mod]. Retrieved August 1, 2018, from https://www.beamng.com/resources/damage-counter.2921/

FAHRSCHUL SIMULATOREN – FAHRSCHUL SIMULATOREN vom Fahrrad bis zum LKW bieten wir alle an und die perfekte traffic school software [Website]. (n.d.). Retrieved from https://fahrschulsimulator.com/

FOERST Fahrsimulatoren | Simulatoren für Lkw und Pkw [Website]. (n.d.). Retrieved from https://www.fahrsimulatoren.eu/de/

jlnr & SJW. (2014, April 24). WebKit extensions [Blog post]. Retrieved from https://developer.mozilla.org/de/docs/Web/CSS/CSS_Referenz/Webkit_Extensions

jojos38. (2017, July 8). How to get vehicle ID through vlua? [Forum post]. Retrieved from https://www.beamng.com/threads/how-to-get-vehicle-id-through-vlua.42591/

Land | BeamNG [Forum]. (n.d.). Retrieved from https://www.beamng.com/forums/land.44/

Modified esc file for beamng drive that solves EV an problem [Source Code]. (2018, May 5). Retrieved from https://pastebin.com/Q01LPqxR

My First App [Wiki page]. (n.d.). Retrieved July 22, 2018, from https://wiki.beamng.com/My_First_App

Nadeox1. (2015, July 14). Introduction to scenarios creation [Forum post]. Retrieved from https://www.beamng.com/threads/introduction-to-scenarios-creation.14700/

Steam Greenlight [Website]. (n.d.). Retrieved from https://steamcommunity.com/greenlight?l=german

Stiegfried. (2018a, May 11). Is there a possibility for message or socket based interprocess communication? [Forum thread]. Retrieved from https://www.beamng.com/threads/is-there-a-possibility-for-message-or-socket-based-interprocess-communication.54715/

Stiegfried. (2018b, June 15). Set the Speed (without acceleration) [Forum thread]. Retrieved from https://www.beamng.com/threads/set-the-speed-without-acceleration.55418/

Stiegfried. (2018c, June 26). Enter text into a <input type="text"> element in a ui-mod [Forum thread]. Retrieved from https://www.beamng.com/threads/enter-text-into-a-input-type-text-element-in-a-ui-mod.55667/

StinchinStein. (2016, May 11). A way to get a value from System Lua, T3D Lua, GameEngine Lua? [Forum thread]. Retrieved from https://www.beamng.com/threads/a-way-to-get-a-value-from-system-lua-t3d-lua-gameengine-lua.25139/

Streams [Wiki page]. (n.d.). Retrieved May 21, 2018, from https://wiki.beamng.com/Streams

SushiPro. (2016, December 19). Ibishu Condensa Electric Drive w/ Overdrive [Mod]. Retrieved August 13, 2018, from https://www.beamng.com/resources/ibishu-condensa-electric-drive-w-overdrive.1682/

SushiPro. (2017, March 21). Ibishu 200EX Electric Drive w/ Overdrive [Mod]. Retrieved August 13, 2018, from https://www.beamng.com/resources/ibishu-200ex-electric-drive-w-overdrive.2028/

Terrains, Levels, Maps [Forum]. (n.d.). Retrieved from https://www.beamng.com/forums/terrains-levels-maps.27/

theseeker01. (2017, January 22). Here's how to add fuel to normal cars and change the wind speed limit. How do I recharge a car thats a EV tho? [Forum post]. Retrieved from https://www.reddit.com/r/BeamNG/comments/5penc0/heres_how_to_add_fuel_to_normal_cars_and_change/

Uradamus. (2014, April 10). A simple array shuffle function for Lua implementing the Fisher-Yates shuffle. [Source Code]. Retrieved from https://gist.github.com/Uradamus/10323382

VicomEditor [Website]. (n.d.). Retrieved from https://www.vicomeditor.de/

Software

BeamNG GmBH (2015). *BeamNG.drive*

BeamNG GmBH (2018). *BeamNG.research*

Blender Foundation (1994). *Blender*

Google Inc. (2009). *AngularJS*

LabLua (1993). *Lua*

Don Ho and other Contributers (2018). *Notepad++*

The MathWorks (2018). *MATLAB R2018a*

TÜV | DEKRA arge tp 21 GbR (2007). *Vicom Editor*

Abkürzungen

CSS	Cascading Style Sheet
FOV	Field of View
GE	Game Engine
GUI	Graphical User Interface
HTML	Hypertext Markup Language
IMIS	Institut für Multimediale und Interaktive Systeme
IPC	Interprocess Communication
JS	JavaScript
Mod	Modifikation
UCD	User Centered Design
UI	User Interface

Anhänge

Inhaltsverzeichnis der Anhänge

Anhang A: Anforderungsdokument

Nachfolgend ist eine konvertierte Version der Exceltabelle eingefügt. Diese entspricht der zum Abschluss dieser Arbeit aktuellsten Version. Die aktuelle Version des Anforderungsdokuments kann, sofern eine Zugriffsrecht vorliegt, in der IMIS-Cloud unter ‚\FahrSim\02_Software\Anforderungen' eingesehen werden.

ID	Eine eindeutige ID
Bereich	Der Bereich in den die Anforderung gehört
Name	Ein eindeutiger Name
Anforderung	Beschreibung der Anforderung mit implizitem Abnahmekriterium
Version und Datum	Version der Anforderung (ganzzahlig) und Datum der letzten Änderung
Priorität	Von 1 bis 5 wobei 1 die höchste Priorität ist
Notwendig für Experiment 1	Hinweis ob die Anforderung für das erste geplante Experiment relevant ist; Diese Spalte könnte nach Abhalten des ersten Experiments irrelevant werden
Standardmäßig erfüllt	Angabe, ob BeamNG.drive die Anforderung standardmäßig erfüllt
Aktueller Stand	Aussage darüber, ob die Anforderung bereits angegangen wurde und Wenn ja wie weit die Umsetzung ist
Von uns umsetzbar	Angabe ob die Anforderung ohne zu große Probleme von uns umgesetzt Werden kann (statt von den Entwicklern von BeamNG.drive)
Ziel/Grund	Der Grund warum diese Anforderung überhaupt existiert
Typ	Funktional, Qualität oder aufgrund einer Randbedingung
Quelle	Wo die Anforderung herstammt
Details/Anmerkungen	

Dieses Anforderungsdokument wurde erstellt basierend auf den Kapiteln 7.3 und 9.2-9.3 des Nachfolgend referenzierten Buches:

Grande, M. (2014). 100 Minuten für Anforderungsmanagement Kompaktes Wissen nicht nur für Projektleiter und Entwickler (2nd ed.). Wiesbaden: Springer Fachmedien Wiesbaden GmbH. Doi:10.1007/978-3-658-06435-8

ID	Bereich	Name	Anforderung	Version und Datum	Priorität
001	Fremdverkehr	AI-Fahrzeuganzahl	Es kann eine beliebe Anzahl an Fahrzeugen herumfahren	v1 04.06.18	
002	Fremdverkehr	AI-Fahrzeugroute	Fahrzeuge fahren eine vorgegebene Route	v1 04.06.18	
003	Fremdverkehr	Fußgänger	Es können Fußgänger herumlaufen	v1 04.06.18	
004	Fremdverkehr	Radfahrer	Es können Radfahrer herumfahren	v1 04.06.18	
005	Fremdverkehr	StVO	Andere Verkehrsteilnehmer halten sich an die StVO	v1 04.06.18	
006	Strecke	Deutsche Strecke	Es gibt deutsche Schilder, Poller, etc. auf einer Strecke	V2 26.06.18	1
007	Strecke	Gerade Strecke	Es gibt eine lange gerade ebene Strecke	v1 04.06.18	1
008	Strecke	Kurvige Strecke	Es gibt eine ebene Strecke mit Kurven	v1 04.06.18	
009	Strecke	Haltestellen mit Haltebucht	Es existieren Bushaltestellen mit Haltebucht auf einer Strecke	v1 04.06.18	
010	Strecke	Ampeln	Es gibt funktionierende Ampeln auf einer Strecke	v1 04.06.18	
011	Fahrzeug	Elektrofahrzeug	Es gibt ein repräsentatives Elektrofahrzeug	v1 04.06.18	1
012	Fahrzeug	Hybridfahrzeug	Es gibt ein repräsentatives Hybridfahrzeug	v1 04.06.18	
013	Fahrzeug	Elektrobus	Es gibt einen Elektrobus	v1 04.06.18	
014	Fahrzeug	Energetische Präzision	Die energetische Simulation ist präzise	v1 04.06.18	
015	Fahrzeug	Realistisches Fahrgefühl	Es besteht ein realistisches Fahrgefühl	v1 04.06.18	1
016	Fahrzeug	Realistisches Cockpit	Fahrzeuge haben ein realistisches Cockpit	v1 04.06.18	1
017	Fahrzeug	Realistische Kamera	Eine onBoard Kamera ist eine realistische Kopfposition	V2 26.06.18	1
018	Fahrzeug	Mehrere Fahrzeugmods	Es können mehrere Fahzeugmods aktiv sein	v1 04.06.18	
019	Schnittstellen	Datenlogging	Daten können geloggt werden	v1 04.06.18	1
020	Schnittstellen	IPC	IPC mit Matlab ist möglich	v1 04.06.18	
021	Schnittstellen	Externe Werte anzeigen	Werte aus Matlab können angezeigt werden	v1 04.06.18	
022	Schnittstellen	Externe Werte verwenden	Werte aus Matlab können verwendet werden	v1 04.06.18	
023	Schnittstellen	Externe GUIs	GUIs können auf einem externen Bildschirm angezeigt werden	v1 04.06.18	5

ID	Bereich	Name	Anforderung	Version und Datum	Priorität
024	Schnittstellen	Geschwindigkeitsverlauf	Ein Geschwindigkeitsverlauf kann eingelesen werden	v1 04.06.18	1
025	Experimentelle Kontrolle	Einstellungen exportieren	Alle Einstellungen können exportiert werden	v1 04.06.18	2
026	Experimentelle Kontrolle	Einstellungen importieren	Einstellungen können aus einem File übernommen werden	v1 04.06.18	
027	Experimentelle Kontrolle	Experimente erstellen	Es kann ein Experiment erstellt werden	V2 12.07.18	1
028	Experimentelle Kontrolle	Automatischer Ablauf	Ein Experiment läuft ohne Eingreifen des Versuchsleiters ab	V2 12.07.18	1
029	Experimentelle Kontrolle	Instruktionen darstellen	Es können Instruktionen dargestellt werden	v1 04.06.18	1
030	Fahrzeugkontrolle	Fahrzeug teleportieren	Das Fahrzeug kann an einen beliebigen Ort gesetzt werden	V2 26.06.18	1
031	UIs	Beliebige GUIs	Beliebige GUIs können erstellt und genutzt werden	v1 04.06.18	
032	UIs	GUIs im Cockpit	GUIs können im Cockpit eines Fahrzeugs integriert werden	v1 04.06.18	
033	UIs	GUI Erkennbarkeit	GUIs sind leicht erkennbar	v1 04.06.18	1
034	Fahrzeugkontrolle	Startgeschwindigkeit setzen	Die Startgeschwindigkeit des Fahrzeugs kann gesetzt werden	V2 26.06.18	
035	Fahrzeugkontrolle	Geschwindigkeit setzen	Die Geschwindigkeit eines Fahrzeugs kann gesetzt werden	V3 26.06.18	
036	Experimentelle Kontrolle	Sound deaktivieren	Der Sound kann temporär deaktiviert werden	V2 26.06.18	1
037	Fahrzeugkontrolle	Route vorgeben	Ein Fahrzeug fährt eine vorgegebene Route mit vorgegebenem Fahrverhalten	v1 04.06.18	
038	UIs	Audioinstruktionen	Audioinstruktionen können abgespielt werden	v1 04.06.18	
039	UIs	GUI Transparenz	GUIs können transparent und intransparent sein	v1 04.06.18	1
040	Trigger	Orts-Trigger	An einem bestimmten Ort passiert etwas	v1 04.06.18	1
041	Trigger	unsichtbare Trigger	An einem bestimmten, für den Teilnehmer nicht erkennbaren Ort passiert etwas	v1 04.06.18	1
042	Experimentelle Kontrolle	Kontrollübergabe	Eine automatische kann in eine manuelle Fahrt übergehen	V2 26.06.18	1
043	Experimentelle Kontrolle	Simulation einfrieren	Simulation kann einfrieren und fortgeführt werden	v1 04.06.18	1
044	Experimentelle Kontrolle	Einfrieren per Trigger	Simulation kann per Trigger einfrieren und fortlaufen	v1 04.06.18	1
045	UIs	GUI fade-In und Out	GUIs können langsam ein oder ausgeblendet werden	V2 12.07.18	1
046	Trigger	Geschwindigkeits-Trigger	Bei über/unterschreiten einer bestimmten Geschwindigkeit passiert etwas	v1 04.06.18	1

ID	Bereich	Name	Anforderung	Version und Datum	Priorität
047	UIs	Eingabe GUIs	Es können Werte in GUIs eingegeben werden	V2 26.06.18	1
048	Experimentelle Kontrolle	Randomisierung	Die Reihenfolge bestimmter Events kann randomisiert werden	V2 12.07.18	1
049	Experimentelle Kontrolle	Speicherort	Die Dateien eines Teilnehmers werden in einem eigenen Ordner gespeichert	v1 04.06.18	
050	Trigger	Distanz-Trigger	Nach einer bestimmten Distanz passiert etwas	v1 04.06.18	1
051	Trigger	Zeit-Trigger	Nach einer bestimmten Zeit passiert etwas	V2 12.07.18	1
052	Trigger	kombinierbare Trigger	Trigger können kombiniert werden und abhängig sein	v1 04.06.18	1
053	UIs	GUI Platzbedarf	GUIs können den ganzen oder einen Teil des Bildschirm einnehmen	v1 04.06.18	1
054	Schnittstellen	Loggingfrequenz	Die Loggingfrequenz kann beliebig gewählt werden	v1 04.06.18	
055	Parameter	Parameter Geschwindigkeit	Die Geschwindigkeit kann ausgelesen werden	v1 04.06.18	1
056	Parameter	Parameter Pedalstellungen	Die Stellung von Brems- und Gaspedal kann ausgelesen werden	v1 04.06.18	1
057	Parameter	Parameter Tankfüllung	Die Tankfüllung (Momentan, Maximal) kann ausgelesen werden	v1 04.06.18	5
058	Parameter	Parameter Drehzahl	Die Drehzahl des Motors kann ausgelesen werden	v1 04.06.18	
059	Parameter	Parameter Zündung	Der Zustand der Zündung kann ausgelesen werden	v1 04.06.18	
060	Parameter	Parameter Energiestand	Der Energiestand (momentan, maximal) kann ausgelesen werden	v1 04.06.18	
061	Parameter	Parameter Position	Die momentane Position des Fahrzeugs kann ausgelesen werden	v1 04.06.18	
062	Parameter	Parameter Ausrichtung	Die momentane Ausrichtung des Fahrzeugs kann ausgelesen werden	v1 04.06.18	
063	Parameter	Parameter Fahrzeuggewicht	Das Gewicht des Fahrzeugs kann ausgelesen werden	v1 04.06.18	
064	Parameter	Parameter Beschleunigung	Die momentane Beschleunigung des Fahrzeugs kann ausgelesen werden	v1 04.06.18	1
065	Parameter	Parameter Spannung	Die Spannung der Batterie kann ausgelesen werden	v1 04.06.18	
066	Parameter	Parameter Leistung	Die Leistung des Motors kann ausgelesen werden	v1 04.06.18	
067	Parameter	Parameter Momentanverbrauch	Die Momentanverbrauch kann ausgelesen werden	v1 04.06.18	
068	Parameter	Parameter Restreichweite	Die Restreichweite kann ausgelesen werden	v1 04.06.18	

ID	Bereich	Name	Anforderung	Version und Datum	Priorität
069	Parameter	Parameter Verbraucher	Der Verbrauch durch sonstige Verbraucher kann ausgelesen werden	v1 04.06.18	
070	Parameter	Parameter Distanz	Die gefahrene Distanz kann ausgelesen werden	v1 04.06.18	1
071	Parameter	Parameter Route	Die gefahrene Route kann ausgelesen werden	v1 04.06.18	
072	Parameter	Parameter Beschaffenheit	Die Beschaffenheit der Strecke kann ausgelesen werden	v1 04.06.18	
073	Parameter	Parameter relative Position	Die Position relativ zu Objekten wie der Mittellinie kann ausgelesen werden	v1 04.06.18	
074	Parameter	Parameter Zeit	Die Zeit seit Beginn des Versuchs/der Fahrt kann ausgelesen werden	v1 04.06.18	1
075	Strecke	Strecken Settings	Es gibt Stadt, Wald und Wiesen Strecken	v1 04.06.18	1
076	Strecke	Kreuzungen	Es gibt eine Strecke mit Kreuzungen (und Kreisverkehr)	v1 04.06.18	
077	Parameter	Parameter Wetter	Das Wetter und der Wind können ausgelesen werden	v1 04.06.18	
078	Fremdverkehr	Fahrgäste	Passanten steigen in den Bus ein	v1 04.06.18	
079	Fahrzeug	Fahrzeug wackelt nicht	Das Fahrzeug darf nicht unnatürlich viel wackeln	V2 12.07.18	1
080	Fahrzeug	Keine Kamera Eigenbewegung	Die Kamera wippt nicht vor und zurück und dreht sich nicht entsprechend der Fahrt	V2 12.07.18	1
081	Strecke	Streckengröße	Es gibt beliebig große Karten (bzw. solche können erstellt werden)	V1 26.06.18	
082	Experimentelle Kontrolle	Kollision deaktivieren	Die Kollision des Fahrzeugs mit Objekten der Umgebung kann deaktiviert werden	V1 26.06.18	

ID	Notwendig für Experiment 1	Standardmäßig erfüllt	Aktueller Stand	Von uns umsetzbar	Ziel/Grund
001	Nein	Nein	Nicht angegangen	Nein	Realismus, Situationen herbeiführen
002	Nein	Nein	Nicht angegangen	Nein	Situationen herbeiführen
003	Nein	Nein	Nicht angegangen	Nein	Realismus, Situationen herbeiführen
004	Nein	Nein	Nicht angegangen	Nein	Realismus, Situationen herbeiführen
005	Nein	Nein	Nicht angegangen	Nein	Realismus
006	Ja	Teilweise	Eine Karte untersucht	Ja	Realismus/Vertrautheit
007	Ja	Nein	Nicht angegangen	Ja	Für Experimente benötigt
008	Nein	Nein	Nicht angegangen	Ja	Für Experimente benötigt
009	Nein	Nein	Nicht angegangen	Ja	Für Experimente benötigt
010	Nein	Ja	Standardmäßig erfüllt	-	Realismus, Situationen herbeiführen
011	Ja	Nein	Communitymod wird genutzt	Von Community	Für Experimente benötigt
012	Nein	Nein	Nicht angegangen	Evtl. von Community	Für Experimente benötigt
013	Nein	Nein	Nicht angegangen	Evtl. von Community	Für Experimente benötigt
014	Nein	Unbekannt, also Nein	Matlabskript wird entwickelt	Von anderer Uni Ja	Präzision
015	Ja	Hängt von Hardware ab	Hardware wird konfiguriert	Ja	Realismus
016	Ja	Ja	Standardmäßig erfüllt	-	Realismus
017	Ja	Nein	Erfüllt	-	Realismus
018	Nein	Nein	Nicht angegangen	Evtl.	Komfort
019	Ja	Nein	Teilweise erfüllt	Ja, mit Forum	Für Experimente benötigt
020	Nein	Nein	Filebased möglich, mit UDP zu prüfen	Filebased Ja, UDP vlt.	Präzision, Datenaustausch
021	Nein	Nein	Erfüllt	-	Für Experimente benötigt
022	Nein	Nein	Teilweise erfüllt	Evtl.	Für Experimente benötigt
023	Nein	Nein	Nicht angegangen	Notfalls mittels IPC Ja	Für Experimente benötigt

ID	Notwendig für Experiment 1	Standardmäßig erfüllt	Aktueller Stand	Von uns umsetzbar	Ziel/Grund
024	Ja	Nein	Sieht noch unrealistisch aus	Evtl.	Für Experimente benötigt
025	Ja	Nein	Nicht angegangen	Wahrscheinlich Ja	Reproduzierbarkeit
026	Nein	Nein	Nicht angegangen	Wahrscheinlich Ja	Reproduzierbarkeit
027	Ja	Ja	Standardmäßig erfüllt	-	Für Experimente benötigt, Reproduzierbarkeit
028	Ja	Ja	Standardmäßig erfüllt	-	Reproduzierbarkeit
029	Ja	Über UI-Mods Ja	Erfüllt	-	Für Experimente benötigt, Reproduzierbarkeit
030	Ja	Ja	Standardmäßig erfüllt	-	Für Experimente benötigt
031	Nein	Ja	Standardmäßig erfüllt	-	Für Experimente benötigt
032	Nein	Ja	Nicht untersucht wie	Wahrscheinlich Ja	Für Experimente benötigt
033	Ja	Im Allgemeinen Ja	Standardmäßig erfüllt	-	Für Experimente benötigt
034		Nein	Workaround existiert	nur der Workaround	Für Experimente benötigt
035		Nein	Workaround existiert	nur der Workaround	Für Experimente benötigt
036	Ja	Ja	Standardmäßig erfüllt	-	Für Experimente benötigt
037	Nein	Evtl.	Nicht angegangen	Nein	Für Experimente benötigt
038	Nein	Nein	Nicht angegangen	Evtl.	Für Experimente benötigt
039	Ja	Ja	Standardmäßig erfüllt	-	Für Experimente benötigt
040	Ja	Ja	Standardmäßig erfüllt	-	Für Experimente benötigt, Reproduzierbarkeit
041	Ja	Ja	Standardmäßig erfüllt	-	Für Experimente benötigt,Reproduzierbarkeit, Realismus
042	Ja	Ja	Standardmäßig erfüllt	-	Für Experimente benötigt
043	Ja	Ja	Standardmäßig erfüllt	-	Für Experimente benötigt
044	Ja	Ja	Standardmäßig erfüllt	-	Für Experimente benötigt, Reproduzierbarkeit
045	Ja	Ja	Standardmäßig erfüllt	-	Für Experimente benötigt
046	Ja	Ja	Standardmäßig erfüllt	-	Für Experimente benötigt, Reproduzierbarkeit

ID	Notwendig für Experiment 1	Standardmäßig erfüllt	Aktueller Stand	Von uns umsetzbar	Ziel/Grund
047	Ja	Halb, siehe Anmerkungen	Halb erfüllt	zweite Hälfte Nein	Für Experimente benötigt
048	Ja	Nein	Erfüllt	Ja	Für Experimente benötigt
049	Nein	Nein	Nicht angegangen	Ja	Komfort
050	Ja	Nein	Beantworteter Forumspost	Siehe Anmerkung	Für Experimente benötigt, Reproduzierbarkeit
051	Ja	Nein	Erfüllt	Ja	Für Experimente benötigt, Reproduzierbarkeit
052	Ja	Ja	Standardmäßig erfüllt	-	Für Experimente benötigt, Reproduzierbarkeit
053	Ja	Ja	Standardmäßig erfüllt	-	Für Experimente benötigt
054		Nein	Nicht angegangen	Evtl.	Für Experimente benötigt, Reproduzierbarkeit
055	Ja	Ja	Standardmäßig erfüllt	-	Für Experimente benötigt
056	Ja	Ja	Standardmäßig erfüllt	-	Für Experimente benötigt
057	Nein	Ja	Standardmäßig erfüllt	-	Für Experimente benötigt
058		Ja	Standardmäßig erfüllt	-	Für Experimente benötigt
059		Ja	Standardmäßig erfüllt	-	Für Experimente benötigt
060		Ja	Standardmäßig erfüllt	-	Für Experimente benötigt
061		Ja	Standardmäßig erfüllt	-	Für Experimente benötigt
062		Ja	Standardmäßig erfüllt	-	Für Experimente benötigt
063		Ja	Standardmäßig erfüllt	-	Für Experimente benötigt
064	Ja	Nein	Nicht direkt erfüllbar	-	Für Experimente benötigt
065		Unbekannt	Nicht angegangen	extern	Für Experimente benötigt
066		Unbekannt	Nicht angegangen	extern	Für Experimente benötigt
067		Nein	Nicht direkt erfüllbar	extern	Für Experimente benötigt
068		Unbekannt	Nicht angegangen	extern	Für Experimente benötigt

ID	Notwendig für Experiment 1	Standardmäßig erfüllt	Aktueller Stand	Von uns umsetzbar	Ziel/Grund
069	Nein	Unbekannt	Nicht angegangen	extern	Für Experimente benötigt
070	Ja	Unbekannt	Nicht angegangen	Nein	Für Experimente benötigt
071		Unbekannt	Nicht angegangen	Nein	Für Experimente benötigt
072	Nein	Unbekannt	Nicht angegangen	Nein	Für Experimente benötigt
073		Nein	Nicht angegangen	Nein	Für Experimente benötigt
074	Ja	Ja	Standardmäßig erfüllt	-	Für Experimente benötigt
075	Ja	Isoliert betrachtet Ja	Nicht angegangen	Ja	Für Experimente benötigt
076	Nein	Isoliert betrachtet Ja	Nicht angegangen	Ja	Für Experimente benötigt
077	Nein	Nein	Nicht angegangen	Evtl.	Für Experimente benötigt
078	Nein	Nein	Nicht angegangen	Nein	Realismus
079	Ja	Nein	Nicht angegangen	eher nein	Realismus
080	Ja	Nein	Erfüllt	Ja	Realismus
081		Nein (siehe Anmerkung)	angeschaut, keine Ergebnisse	Evtl.	Für Experimente benötigt
082		wahrscheinlich nein	Nicht angegangen	Nein	Für Experimente benötigt

ID	Typ	Quelle	Details/Anmerkungen
001	Qualität		
002	Funktionalität		
003	Funktionalität		
004	Funktionalität		
005	Funktionalität		Auch an Ampeln
006	Funktionalität	Experimentalablauf_v04.docx	Streckenanforderungen gelten häufig Anfoderungsübergreifend; auf einer zu kleinen Karte gibt es deutsche Poller
007	Funktionalität		Streckenanforderungen gelten häufig Anfoderungsübergreifend
008	Funktionalität		Streckenanforderungen gelten häufig Anfoderungsübergreifend
009	Funktionalität		Streckenanforderungen gelten häufig Anfoderungsübergreifend
010	Funktionalität		Streckenanforderungen gelten häufig Anfoderungsübergreifend
011	Funktionalität	Experimentalablauf_v04.docx	mit repräsentativ ist ein Durchschnittsauto gemeint
012	Funktionalität		mit repräsentativ ist ein Durchschnittsauto gemeint
013	Funktionalität		ideal wäre mit Sileo-Cockpit
014	Qualität		Wird nach aktuellem Stand nach Matlab ausgelagert
015	Qualität		
016	Qualität		
017	Qualität		Sollte wohl besser in eine Mod gewandelt werden (aktuell ist die Standardkamera überschrieben
018	Funktionalität		
019	Funktionalität	Experimentalablauf_v04.docx	Einige Parameter noch unklar, nämlich Gefahrene Strecke, Restreichweite, Leistung des Motors, Sonstige Verbraucher, Batteriespannung, Streckenbeschaffenheit
020	Funktionalität		
021	Funktionalität		
022	Funktionalität	Experimentalablauf_v04.docx	Geschwindigkeit ist noch problematisch; siehe Anforderung 035
023	Funktionalität		

ID	Typ	Quelle	Details/Anmerkungen
024	Funktionalität	Experimentalablauf_v04.docx	Problem: Geschwindigkeit lässt sich nicht direkt eingeben; siehe Anforderung 035
025	Funktionalität	Experimentalablauf_v04.docx	
026	Funktionalität	Experimentalablauf_v04.docx	
027	Funktionalität		Mittels Szenario → zu kompliziert für nicht-Informatiker
028	Funktionalität		Mittels Szenario
029	Funktionalität		
030	Funktionalität	Experimentalablauf_v04.docx	
031	Qualität		Dies sollten auch nicht-Informatiker machen können
032	Funktionalität		z.B. auch im Handyhalterung
033	Qualität		
034	Funktionalität	Experimentalablauf_v04.docx	Sonderfall von Anforderung 035; Fahrzeug beschleunigt auf Geschwindigkeit und wird dann teleportiert
035	Funktionalität	Experimentalablauf_v04.docx	Keine Beschleunigung nötig; Fahrzeug beschleunigt von alleine auf Geschwindigkeit und wird dann teleportiert
036	Funktionalität	Experimentalablauf_v04.docx	
037	Funktionalität	Experimentalablauf_v04.docx	vgl. Anforderung 024
038	Funktionalität	Experimentalablauf_v04.docx	
039	Funktionalität	Experimentalablauf_v04.docx	
040	Funktionalität		
041	Funktionalität	Experimentalablauf_v04.docx	Sonderfall von Anforderung 040
042	Funktionalität	Experimentalablauf_v04.docx	
043	Funktionalität	Experimentalablauf_v04.docx	
044	Funktionalität	Experimentalablauf_v04.docx	Sonderfall von Anforderung 043
045	Funktionalität	Experimentalablauf_v04.docx	
046	Funktionalität	Experimentalablauf_v04.docx	

ID	Typ	Quelle	Details/Anmerkungen
047	Funktionalität	Experimentalablauf_v04.docx	Eingabe per Tastatur funktioniert nicht, per Dropdown oder anklickbaren Pfeilen ja
048	Funktionalität	Experimentalablauf_v04.docx	Die Reihenfolge muss geloggt werden
049	Funktionalität	Experimentalablauf_v04.docx	
050	Funktionalität	Experimentalablauf_v04.docx	Definitiv umsetzbar, fraglich ist dabei aktuell noch die Präzision
051	Funktionalität	Experimentalablauf_v04.docx	
052	Funktionalität	Experimentalablauf_v04.docx	z.B. wenn das dritte mal die Geschwindigkeit erreicht wurde
053	Funktionalität	Experimentalablauf_v04.docx	
054	Funktionalität	Experimentalablauf_v04.docx	Sonderregel für Anforderung 019
055	Funktionalität		
056	Funktionalität		
057	Funktionalität		
058	Funktionalität		
059	Funktionalität		
060	Funktionalität		
061	Funktionalität		
062	Funktionalität		
063	Funktionalität		
064	Funktionalität		
065	Funktionalität		
066	Funktionalität		
067	Funktionalität		
068	Funktionalität		

ID	Typ	Quelle	Details/Anmerkungen
069	Funktionalität		
070	Funktionalität		
071	Funktionalität		
072	Funktionalität		
073	Funktionalität		
074	Funktionalität		
075	Funktionalität	Allgemeine_Anforderungen_v02.odp	Streckenanforderungen gelten häufig Anfoderungsübergreifend
076	Funktionalität	Allgemeine_Anforderungen_v02.odp	Streckenanforderungen gelten häufig Anfoderungsübergreifend
077	Funktionalität	Allgemeine_Anforderungen_v02.odp	
078	Funktionalität	Exposé_Stieglitz_V05.odt	
079	Qualität		
080	Qualität		
081	Funktionalität		Nur die Karte ohne alles ist „unendlich“ groß, lässt sich aber nicht bearbeiten
082	Funktionalität		Damit ein Experiment nicht neugestartet werden muss wenn man die Leitplanke streift o.ä.

Anhang B: Anleitung

Nachfolgend ist eine .jpg Version der Anleitung beigefügt. Die aktuelle Version der Anleitung kann, sofern ein Zugriffsrecht vorliegt, in der IMIS-Cloud unter ‚\FahrSim\02_Software\Anleitung' eingesehen werden.

Anleitung zur Nutzung der Software BeamNG.research

Dieses Dokument ist entstanden im Rahmen der Bachelorarbeit „Ausbau der Software BeamNG zu Fahrsimulationsumgebung für Untersuchungen in der Nutzer-Energie-Interaktion“ von Pascal Stieglitz. Es wurde mit BeamNG.drive Version „Alpha v0.13.0.4“ erstellt. Es sollte modifiziert werden, um die Angaben in dem Dokument auf dem jeweils aktuellsten und korrekten Stand zu halten. In dem Fall wird darum gebeten, die Änderung im Changelog zu vermerken und das Dokument als neue Datei mit neuer Versionsnummer abzuspeichern.

Changelog

Autor	Datum	Änderungen	Anmerkungen
Pascal Stieglitz	08.2018	-	Initiale Version

Inhaltsverzeichnis

1 Bedienung der Software

1.1 Szenario starten

Um ein Szenario zu starten muss man auf *Spielen* klicken, gefolgt von *Szenario*. Anschließend muss das gewünschte Szenario ausgewählt werden.

Für Experimente ist es häufig empfehlenswert bereits auf *Start* zu klicken bevor der Teilnehmer ankommt, da so der Startbildschirm verschwindet und eventuell Vorbereitungen vom Programm getan werden müssen (z.B. das Fahrzeug auf die gewünschte Startgeschwindigkeit beschleunigen). Dies sollte jedoch nicht zu lange im voraus geschehen.

Durch das drücken der *R*-Taste (auf der Tastatur) kann das Szenario neugestartet werden. Abhängig davon, wie das Szenario programmiert wurde, hat dies entweder den selben Effekt als würde das gesamte Programm neugestartet werden oder setzt nur das Fahrzeug an den Anfang der Strecke zurück und startet den nächsten Durchlauf/die nächste Sequenz.

1.2 UI-Mods aktivieren oder deaktivieren

Um UI-Mods zu aktivieren oder zu deaktivieren muss während der Freeroammodus oder ein Szenario läuft *ESC* gedrückt und *Customize UI Apps* (siehe Abbildung 1) angeklickt werden. Dadurch öffnet sich das UI-Menü (siehe Abbildung 3). Dadurch werden auch alle bereits vorhandenen UI-Mods (siehe Abbildung 2) sichtbar, unabhängig davon ob sie eigentlich unsichtbar sein sollten.

Abbildung 1: Customize UI Apps

Abbildung 3: UI Menü

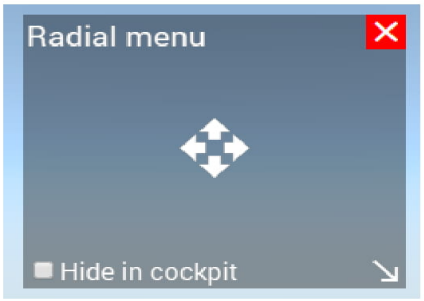

Abbildung 2: UI-Mod während das UI Menü offen ist

Um eine neue UI-Mod hinzuzufügen muss auf das Plus auf orangenem Hintergrund geklickt werden. Aus der sich öffnenden Liste verfügbarer UI-Mods muss die richtige ausgewählt werden. Anschließend taucht sie wie in Abbildung 2 auf.

Vorhandene UI-Mods können mittels des Pfeils unten rechts vergrößert oder verkleinert werden. Mit dem Kreuz oben rechts kann eine UI-Mod entfernt werden. Mit dem Symbol mit vier Pfeilen in der Mitte einer UI-Mod kann diese verschoben werden.

Mit der *ESC*-Taste oder einem Klick auf den Haken im UI-Menü kann das UI-Menü geschlossen werden. Mit dem Uhrensymbol im UI-Menü können alle UI-Mod Veränderungen auf Werkseinstellung zurückgesetzt werden. Dies ist für gewöhnlich nicht nötig.

UI-Mods bleiben auch nach einem Neustart von BeamNG.drive erhalten und sind für alle Szenarios gleich, für den Freeroammodus jedoch sind potentiell andere aktiv.

1.3 Die richtige Kamera wählen

Die Kamera sollte meist bereits richtig eingestellt sein. Für die meisten Experimente wird wahrscheinlich die Kamera *Onboard.Hood* benötigt. Man kann eine Kamera als Standardkamera

einstellen, indem man *ESC* drückt, auf *options* (siehe Abbildung 4)geht und den Reiter *Camera* (siehe Abbildung 5)auswählt.

Abbildung 4:
Options

Mit den Pfeilen (siehe Abbildung 5) kann die Reihenfolge der Kameras geändert werden. Besonders relevant ist welche Kamera an Position 1 steht, da diese die Standardkamera ist.

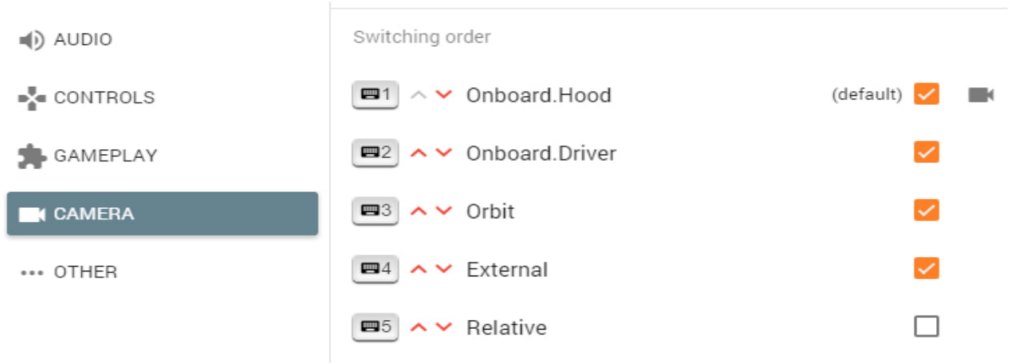

Abbildung 5: Kameramenü

Durch einen Klick auf das Zahlenfeld links neben den Pfeilen kann eine Kamera als aktuelle ausgewählt werden. Alternativ kann mit der C-Taste auf der Tastatur zwischen den Kameras gewechselt werden. Für Experimente empfiehlt es sich jedoch, die gewünschte Kamera als Standardkamera einzustellen, da während eines Durchlaufs auf die Standardkamera gewechselt werden könnte.

1.4 Fahrzeugmod für Datenlogging oder -austausch manuell aktivieren

Für Experimente empfiehlt es sich, die Mod für Datenlogging oder -austausch automatisch aktivieren zu lassen. Siehe dazu 3.1.1 Szenarios. In einem Szenario kann die nötige Fahrzeugmod nicht manuell aktiviert werden. Falls aus irgendwelchen Gründen die Fahrzeugmod zum Loggen oder für den Datenaustausch manuell aktiviert werden soll (z.B. zum testen ob diese funktioniert) kann dies wie folgt getan werden:

Zunächst muss der Freeroam-Modus von BeamNG.drive auf der gewünschten Karte an dem gewünschten Spawnpunkt gestartet werden. Außerdem muss das richtige Fahrzeug ausgewählt werden. Siehe dazu 1.5. Anschließend muss im Menü (geöffnet mit der *ESC*-Taste) an der linken Seite der Menüpunkt *Vehicle Config* (siehe Abbildung 6) ausgewählt werden. In dem standardmäßig ausgewählten Reiter *Parts* muss die Option *Addidtional Modification* auf *logging_data* bzw. *exchange_data* oder *exchange_and_log_data* (oder falls eine andere Fahrzeugmod verwendet werden soll diese) gewechselt werden.

Abbildung 6: Vehicle Config

1.5 Freeroam-Modus starten

Um den Freeroammodus zu starten muss man zunächst auf Spielen und dann auf Freeroam klicken. Anschließend müssen die richtige Karte und der richtige Spawnpunkt auf der Karte gewählt werden. Im Gegensatz zu einem Szenario muss im Freeroam evtl. noch das richtige Fahrzeug eingestellt werden. Um dies zu tun muss man Esc drücken und auf Vehicles (siehe Abbildung 7) klicken. Anschließend muss man das richtige Fahrzeugmodell und danach die richtige Fahrzeugkonfiguration wählen. Man kann an dieser Stelle außerdem die Farbe wechseln. Abschließend muss auf Replace Current geklickt werden. Es könnte noch nötig sein eine Fahrzeugmod zu aktivieren, siehe dazu 1.4.

Abbildung 7: Vehicle

1.6 Mögliche Probleme und Lösungen

1.6.1 Mod wird nicht angezeigt

Problem: Im Freeroam wird unter *Additional Modification* kein *logging_data* angezeigt.

Mögliche Lösung: Unter dem Dateipfad *D:\Dateien* die Datei *ConfigureBeamNG_vXX.bat* ausführen.

1.6.2 Es wird keine CSV-Datei erzeugt

Problem: Es wird keine .csv-Datei erzeugt.

Mögliche Lösung: Überprüfen ob in der Mod zum loggen der Daten nur in den Editblöcken gearbeitet wurde und ob alles in der richtigen Form eingetragen wurde (siehe 3.1.2). Falls kein Fehler gefunden werden kann, könnte es helfen, die Datei *dataLogging.lua* aus dem Dateipfad *D:\Dateien\logging\lua\vehicle\controller* in den Pfad *C:\Users\aFahrSim\Documents\BeamNG.drive\mods\unpacked\logging\lua\vehicle\controller* zu kopieren und die Änderungen neu vornehmen.

1.6.3 Fatal Vehicle Exception

Problem: Nachdem die Mod im Freeroam aktiviert wird bzw. wenn das Szenario startet, wird oben links 'Fatal Vehicle Exception' angezeigt und das Fahrzeug lässt sich nicht bewegen.

Mögliche Lösung: Überprüfen ob in der Mod zum loggen der Daten nur in den Editblöcken gearbeitet wurde und ob alles in der richtigen Form eingetragen wurde (siehe 3.1.2). Falls kein Fehler gefunden werden kann, könnte es helfen, die Datei *dataLogging.lua* aus dem Dateipfad *D:\Dateien\logging\lua\vehicle\controller* in den Pfad *C:\Users\aFahrSim\Documents\BeamNG.drive\mods\unpacked\logging\lua\vehicle\controller* zu kopieren und die Änderungen neu vornehmen.

2 Welche Dateien und Mods können wo gefunden und installiert werden

Dateien in den folgenden Ordnerpfaden bzw. die folgenden Dateien könnten für die Reproduktion eines Experiments relevant sein. Außerdem müssen neue Modifikationen der genannten Arten an diesen Orten abgelegt werden, damit sie genutzt werden können. Statt die Ordner für Fahrzeuge, Fahrzeugmods und Szenarios kann auch der Order kopiert werden, in dem diese liegen.

Art der Mod	Ordnerpfad/Datei	Anmerkungen
Fahrzeug	C:\Users\aFahrSim\Documents\BeamNG.drive\mods\unpacked	Der Ordner der den Namen der Mod trägt muss in dieses Verzeichnis kopiert werden.
Fahrzeugmod	C:\Users\aFahrSim\Documents\BeamNG.drive\mods\unpacked	Der Ordner der den Namen der Mod trägt muss in dieses Verzeichnis kopiert werden.
Szenario	C:\Users\aFahrSim\Documents\BeamNG.drive\mods\unpacked	Der Ordner der den Namen der Mod trägt muss in dieses Verzeichnis kopiert werden.
Karte	C:\Users\aFahrSim\Documents\BeamNG.drive\mods\unpacked	Der Ordner der den Namen der Mod trägt muss in dieses Verzeichnis kopiert werden.
UI-Mod	C:\Users\aFahrSim\Documents\BeamNG.drive\ui\modules\apps	Der Ordner der den Namen der Mod trägt muss in dieses Verzeichnis kopiert werden.
Kamera	C:\Program Files (x86)\BeamNG.drive\lua\ge\extensions\core\cameraModes\onboard.lua	Diese modifizierte Version einer original Datei muss kopiert werden, damit die Kamera korrekt positioniert ist.
Lenkradsperre	C:\Program Files (x86)\BeamNG.drive\lua\vehicle\input.lua	Diese modifizierte Version einer original Datei muss kopiert werden, damit das Lenkrad temporär deaktiviert werden kann (siehe 3.2.3).
Bremssperre	C:\Program Files (x86)\BeamNG.drive\lua\vehicle\controller\vehicleController.lua	Diese modifizierte Version einer original Datei muss kopiert werden, damit Bremsen temporär deaktiviert werden kann (siehe 3.2.3).
Einstellungen	C:\Users\aFahrSim\Documents\BeamNG.drive\settings	Dieser Ordner muss kopiert werden, damit diverse Einstellungen übernommen werden.

Tabelle 1: Ordnerpfade und Dateien für Mods und Einstellungen

3 Erstellen neuer Modifikationen/Modifizieren vorhandener Dateien

In 3.1 wird beschrieben, wie die verschiedenen Arten von Mods erstellt und verändert werden können, in 3.2 sind Tipps wie eine Liste Auslesbare Werte/Streams oder Nützliche Konsolenkommandos und Codezeilen zu finden.

3.1 Die verschiedenen Arten von Mods erstellen/verändern

3.1.1 Szenarios

Unter *D:\Dateien* kann ein *dummyScenario* gefunden werden, welches erweitert werden kann. Dieses beinhaltet eine *.prefab*, eine *.json*, eine *.jpg* und eine *.lua* Datei. Die *.jpg* dient als Vorschaubild im Auswahlfenster welches Szenario man starten möchte. Die *.json* enthält einige Metadaten,

relevant ist lediglich der Eintrag *prefabs*. Alle weiteren Einträgen dienen der Beschreibung des Szenarios oder können so belassen werden. Die *.lua* Datei enthält Lua-Code. Die *.prefab* Datei listet alle Objekte des Szenarios inklusive Informationen wie z.B. der Position des Objektes auf.

Die folgende Anleitung ist größtenteils eine erweiterte und ggf. korrigierte Übersetzung der Anleitung unter https://www.beamng.com/threads/introduction-to-scenarios-creation.14700/. Die ursprüngliche Anleitung wurde nicht von mir erstellt, lediglich die Übersetzung und einige zusätzliche Hinweise.

Als erstes muss ein Ordnerpfad entsprechend der gewünschten Karte wie folgt erstellt werden: *‚C:\Users\aFahrSim\Documents\BeamNG.drive\mods\unpacked\nameOfTheScenario\levels\name_of_the_map‘*. Für das *dummyScenario* auf der Karte *East Cost USA* würde der Ordnerpfad lauten: *‚C:\Users\aFahrSim\Documents\BeamNG.drive\mods\unpacked\dummyScenario\levels\east_coast_usa‘*. Der Levelname muss dem Namen der *.zip* der Karte entsprechen. Originalkarten liegen unter dem Pfad *‚C:\Program Files (x86)\BeamNG.drive\content\levels‘*, per Mod hinzugefügte unter *‚C:\Users\aFahrSim\Documents\BeamNG.drive\mods\unpacked‘*. Die *.prefab* Datei, die *.json* Datei, die *.lua* Datei im *scenarios* Ordner (inklusive des Ordners) müssen in den erstellten Ordnerpfad kopiert werden. Die Dateien und der Ordner dürfen umbenannt werden (müssen sie sogar, wenn bereits ein Szenario mit dem Namen *dummyScenario* existiert).

Während der Szenarioerstellung sollte die Karte nicht gespeichert werden. Falls versehentlich gespeichert wurde kann die Karte durch eine Kopie der Karte ersetzt werden. Eine solche sollte ggf. vor Beginn der Arbeiten erstellt werden.

Fahrzeug positionieren:

Zunächst muss das Szenario gestartet werden inklusive eines Klicks auf *Start*. Dann muss mit *J* das Spiel pausiert und mit *F11* der *Editor Mode* geöffnet werden. Als erstes muss muss im *Scene Tree* runtergescrollt werden. Unter *ScenarioObjectsGroup* muss das Szenario mit *Rechtsklick → Unpack Prefab* entpackt werden.

Hinweis: Änderungen die vorgenommen werden während das Szenario nicht entpackt ist werden nicht gespeichert.

Das Fahrzeug (*scenario_player0*) kann im *Scene Tree* ausgewählt werden. Oben links können im *Object Editor* (*F1*) die Tools für die Auswahl eines Objektes (die Auswahl im *Scene Tree* ist meist einfacher), Translation, Rotation und Skalierung ausgewählt werden. Mit diesen Tools oder den Textboxen in den Eigenschaften des Fahrzeug kann das Fahrzeug positioniert und rotiert werden. Andere Objekte können ebenfalls entsprechend bearbeitet werden.

Tipp: Positioniere das Fahrzeug mit den Tools nur grob in der Nähe des gewünschten Startortes, beende mit *J* die Pause und fahr das Fahrzeug an die gewünschte Stelle. Pausiere dann mit J erneut. So ist die nötige Präzision bei der Positionierung und insbesondere Rotation leichter erreichbar als mit ausprobieren der Werte.

Wenn das Fahrzeug richtig positioniert und rotiert ist, muss das Prefab mittels *Rechtsklick → Pack Prefab* wieder gepackt werden. Falls der Speicherort gewählt werden soll (soll er laut der Originalanleitung beim erstmaligen Packen immer, tatsächlich aber manchmal nicht) wähle einfach einen Pfad den du wieder findest. Anschließend kannst du das Szenario beenden.

Hinweis: Wird das Prefab nicht gepackt bevor das Szenario beendet wird, werden alle Änderungen verworfen.

Tipp: Die Änderungen werden nicht immer am selben Ort gespeichert. Manchmal ist es nötig die Dateien aus dem Dateipfad *‚C:\Users\aFahrSim\Documents\BeamNG.drive\levels‘* in den richtigen

Ordnerpfad zu verschieben. Auch an anderen Ordnerpfaden können die Änderungen gespeichert werden. Dass mal der eine und mal der andere Pfad verwendet wird ist wahrscheinlich ein Bug. Wenn also mal eine Änderung nicht übernommen wurde, überprüfe einfach ob ein zusätzlicher Ordnerpfad für das Szenario angelegt wurde.

Trigger platzieren:

Ein Trigger ist ein unsichtbares Objekt, welches die Funktion *onBeamNGTrigger(data)* auslöst wenn es durchfahren wird. Zunächst muss erneut das Szenario gestartet und entpackt werden. Dann muss im *Scene Tree* unter *Level* → *BeamNG* das Objekt *Lua Trigger* ausgewählt werden (siehe Abbildung 8). Mit einem Doppelklick wird ein Trigger erzeugt. Der Trigger kann wie das Fahrzeug verschoben, rotiert und (für das Fahrzeug war dies noch nicht wichtig) skaliert werden. Außerdem muss der Trigger umbenannt werden. Wenn mehrere Trigger den selben Namen hätten, gäbe es Probleme bei der Abfrage, welcher Trigger durchfahren wurde. Außerdem muss der Trigger im *Scene Tree* in das Szenario verschoben werden. Nachdem alle Trigger erzeugt, platziert, benannt und in das Szenario verschoben wurden, kann das Prefab wieder gepackt und das Szenario geschlossen werden.

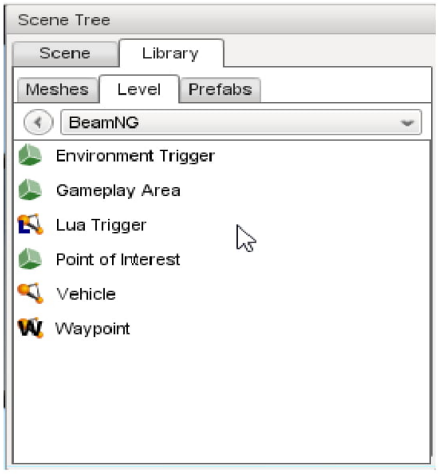

Abbildung 8: Scene Tree

Hinweis: Alle Objekte die Teil des Szenarios sein sollen müssen im *Scene Tree* in das Szenario verschoben werden, weil sie sonst zur Karte statt zum Szenario gehören würden.

Tipp: Verwende NICHT das Feld *luaFunction* das ein Trigger anbietet. Dies würde dazu führen, dass die Funktion *onBeamNGTrigger(data)* durch diesen Trigger nicht aufgerufen werden würde.

Objekte platzieren:

Das platzieren von Objekten verläuft genau wie das platzieren von Triggern.

An dieser Stelle endet die Anleitung unter https://www.beamng.com/threads/introduction-to-scenarios-creation.14700/. Die nun folgenden Erklärungen wurden selbst geschrieben.

Fahrzeug ändern:

Um das Fahrzeugmodell zu ändern muss in der der *.prefab* im Objekt *BeamNGVehicle(scenario_player0)* der Wert des Attributs *JBeam* geändert werden. Um die

Konfiguration zu ändern (z.B. um den Datenlogger zu nutzen) muss das Attribut *partConfig* angepasst werden. Die Farbe kann mittels des Attributs *color* verändert werden.

Funktionen in der .lua:

In der *.lua* sind bereits einige Funktionen vorhanden. Einige wurden einfach aus der *.lua* eines anderen Szenarios übernommen und können meist ignoriert werden (entweder weil sie unwichtig sind oder weil sie in jedem Szenario das selbe tun sollen). Es können beliebige neue Funktionen erstellt werden. Damit eine Funktion von außerhalb erreichbar ist (siehe 3.2.2), muss (vorzugsweise ganz unten) eine Zeile `M.functionName = functionName` hinzugefügt werden.

Hinweis: Nur Funktionen die mittels `M.functionName = functionName` exportiert werden, können von außerhalb aufgerufen werden. Funktionen die nur innerhalb der .lua verwendet werden sollen müssen nicht exportiert werden.

Sonstige Hinweise:

Es empfiehlt sich Trigger statt Waypoints zu nutzen, weil Waypoints aufgrund eines Bugs manchmal nicht unsichtbar gemacht werden können. Außerdem kann so kein ‚Du hast gewonnen/verloren' Bildschirm erscheinen wenn der letzte Waypoint durchfahren ist.

3.1.2 Fahrzeugmod editieren (zum Daten loggen oder austauschen)

Für Fahrzeugmods wurden zwei Abschnitte erstellt. Dieser geht darauf ein, wie in der Mod zum loggen von Daten bzw. austauschen mit Matlab die richtigen Parameter eingestellt werden können. Er wurde so formuliert, dass man (hoffentlich) auch ohne Programmierkenntnisse zum Ziel kommt.

Um die Mod konfigurieren zu können, muss die entsprechende Datei geöffnet werden. Exemplarisch wird die Mod zum loggen der Daten verwendet, für die Mod für den Datenaustausch oder Datenaustausch und logging müssen der Dateipfad und der Dateiname entsprechend angepasst werden. Es muss zum Dateipfad *C:\Users\aFahrSim\Documents\BeamNG.drive\mods\unpacked\logging\lua\vehicle\controller* navigiert werden (z.B. durch klicken auf den Link). Anschließend muss die Datei *dataLogging.lua* mit *Rechtsklick -> Edit with Notepad++* geöffnet werden.

In der Datei sind einige Zeilen, die mit zwei Strichen beginnen und einige ohne Striche. Zeilen mit zwei Strichen am Anfang sind Kommentare. In der Datei sind mehrere Kommentarblöcke, die mit einer Zeile `-- ###Editmarker xx###` beginnen. Auf diese Zeile folgt ein Kommentar, was in diesem Block editiert werden kann/darf. Anschließend folgen potentiell mehrere Zeilen, die mit `-- Example:`, `-- Important:` oder `-- Note:` beginnen. Diese liefern Beispiele, wichtige Hinweise oder eventuell interessante Hinweise. Die Nachfolgende Zeile lautet `-- ***Begin Editblock xx***`. Weiter unten folgt eine Zeile `-- ***End Editblock xx***`. Zwischen diesen beiden Kommentaren dürfen Zeilen hinzugefügt, gelöscht und verändert werden. Außerhalb dieser Editblöcke sollte man nur dann arbeiten, wenn man weiß was man tut.

Logging:

Die Zeit (gemessen in Sekunden mit 15 Nachkommastellen) wird immer geloggt (Ausnahme: exchangeData.lua, da diese nicht loggt). In 3.2.1 Auslesbare Werte/Streams werden die direkt verfügbaren Parameter aufgelistet. Um einen dieser Werte zu loggen, muss nur im Editblock 04 eine entsprechende Zeile existieren. Diese muss das Format

```
str = str..tostring((electrics.values.airspeed or 0) )..","
```

aufweisen. Das `electrics.values.airspeed` wird dabei durch den LUA-Namen (siehe Tabelle 2, 3, 4 und 5 in Kapitel 3.2.1 Auslesbare Werte/Streams) des gewünschten Parameters ersetzt. Die letzte Zeile in diesem Editblock muss auf ein `"\n"` statt auf ein `","` enden.

Daten senden:

Das Versenden von Daten mittels File-based Interprocess-Communication funktioniert ähnlich wie das Loggen der Daten (siehe vorherigen Absatz). Der einzige Unterschied ist, dass die zu übermittelnden Daten in Editblock 06 eingetragen werden müssen.

Daten empfangen:

Das Empfangen von Daten funktioniert etwas anders als das Senden. Je nachdem was mit den Daten getan werden soll, muss die Einspeisung anders durchgeführt werden.

Um Geschwindigkeitswerte einzulesen bzw. nicht einzulesen, müssen zwei Zeilen im Editblock 07 verändert werden. Um einen Geschwindigkeitswert einzulesen müssen die erste und die letzte Zeile dieses Editblocks `--` lauten. Damit keine Geschwindigkeitswerte eingelesen werden, muss die erste Zeile `--[[` lauten und die letzte `--]]`.

Hinweis: Abgesehen von den ersten beiden und der letzten Zeile des Editblock 07 ist der gesamte Block zum justieren der Bremsen beim automatischen Fahren. Durch herumprobieren mit den Parametern kann das Bremsverhalten angepasst werden.

Daten, die in einen Datenstream geschrieben werden sollen (z.B. damit sie in einer UI angezeigt werden können), müssen in Editblock 08 hinzugefügt werden. Dazu muss eine Zeile der Form `electrics.values.MatlabOne = tonumber(y)` erzeugt werden, wobei `electrics.values.MatlabOne` der Name des Datenstreams ist, in den der Wert geschrieben werden soll.

Hinweis: Die Mod ist darauf ausgelegt, dass nur ein Wert in einen Datenstream geschrieben werden soll. Falls mehrere Werte nach BeamNG importiert werden sollen, sind folgende Anpassungen der Mod notwendig: Pro zusätzlichem Wert muss der Bereich von `-- ***End Editblock 07***` bis `-- ***End Editblock 08***` kopiert und nach Editblock 08 eingefügt werden. Der Streamwert der geschrieben werden soll muss bei jedem Block entsprechend angepasst werden.

Indirekt verfügbare Parameter:

Einige Parameter, wie z.B. Beschleunigung oder auch Geschwindigkeit in km/h statt m/s, sind nicht direkt auslesbar, lassen sich aber aus auslesbaren Werten errechnen. Man kann diese Parameter nachträglich von einem anderen Programm oder direkt in der Mod errechnen lassen. Um einen Wert in der Mod direkt zu errechnen, kann im Editblock 04 statt eines Parameter eine Rechenoperation eingefügt werden, z.B. `electrics.values.wheelspeed*3.6` statt nur `electrics.values.wheelspeed`. Die Rechenoperationen +, -, * und / sind erlaubt.

Wenn für eine Berechnung nicht nur aktuelle sondern auch vorherige Werte benötigt werden, muss mit Variablen gearbeitet werden. Der vorherige Zeitpunkt kann immer verwendet werden, der Name der Variable lautet `timerOld`. Werden noch andere vorherige Werte benötigt, müssen zunächst entsprechende Variablen in Editblock 01 deklariert werden. Dazu ist eine Zeile der Form

```
local speedOld = 0
```

nötig, wobei `speedOld` in diesem Beispiel der Name der Variable ist. Variablennamen müssen einzigartig sein, dürfen nur Buchstaben, Zahlen und Unterstriche enthalten und sollten mit einem kleinen Buchstaben beginnen.

Im Editblock 02 muss eine Zeile

```
speedOld = 0
```

vorhanden sein.

Im Editblock 05 muss der alte Wert in die Variable geschrieben werden, dies geschieht mit einer Zeile der Form

```
speedOld = electrics.values.wheelspeed
```

wobei `electrics.values.wheelspeed` der zu speichernde Wert ist.

Wurde die Variable in den drei Editblöcken hinzugefügt, kann im Editblock 04 eine Mathematische Operation mit den Variablen ausgeführt werden. Diese könnte z.B.

```
(electrics.values.wheelspeed-speedOld)/(timer-timerOld)
```

lauten.

3.1.3 Fahrzeugmod erstellen

Eine Fahrzeugmod besteht aus einer .lua Datei die den Quellcode enthält und einer .jbeam Datei pro Fahrzeug in dem die Mod verwendet werden soll. Fahrzeuge, für die keine .jbeam Datei existieren, können die Mod nicht nutzen. Unter *D:\Dateien* kann eine *dummyMod* gefunden werden. Diese liefert die Grundstruktur einer Mod.

Unter *\dummyMod\vehicles* muss für jedes Fahrzeug ein Ordner mit dem Namen des Fahrzeugs (z.B. *200ex*) erstellt werden. In dem Ordner muss eine *.jbeam* Datei mit dem Namen der Mod liegen, beispielsweise *dummyMod.jbeam*. In dieser Datei muss die Mod für das Fahrzeug konfiguriert werden. Im folgenden wird der Text einer solchen *.jbeam* dargestellt. Die zu bearbeitenden Elemente werden farblich hervorgehoben und weiter unten erläutert. Elemente mit der gleichen Farbe müssen den gleichen Wert erhalten.

```
{
	"coupe_dummy_mod": {
		"information":{
			"authors":"Stiegfried",
			"name":"dummy_mod",
		},
		"slotType" : "coupe_mod",
		"controller": [
			["fileName"],
			["dummyMod", {}]
		],
	}
}
```

coupe ist der Name des Fahrzeugs für das die Mod mit dieser Datei konfiguriert wird.

dummy_mod ist der Name unter dem die Mod im Freeroammodus manuell aktiviert werden kann.

dummyMod ist der Name der Dateien und des Ordners.

Hinweis: Bei offiziellen Fahrzeugen sind der Name des Fahrzeug der in der .jbeam eingetragen werden muss und der Name des Fahrzeugs nach dem der Ordner benannt werden muss identisch. Bei Fahrzeugen die durch Mods hinzugefügt werden können sich die Namen unterscheiden. Z.B. muss beim 200ex in der .jbeam coupe eingetragen werden.

In der *.lua* Datei muss die Funktionalität der Mod umgesetzt werden. Für Auslesbare Werte und die Kommunikation mit anderen Instanzen (z.B. der GameEngine (ge)) siehe 3.2.

3.1.4 UI-Mod

Eine UI-Mod (oder auch App genannt) besteht im Kern aus zwei Dateien. Unter *D:\Dateien* kann eine *DummyApp* gefunden werden. In der *app.json* werden Metainfos eingetragen. Bearbeitet werden müssen hier nur die directive (der Ordnerpfad mit camelCase (also keine Leerzeichen oder sonstige Trennzeichenund immer den ersten Buchstaben eines Wortes groß mit Ausnahme des ersten)) und das domElement (der Ordnerpfad aber diesmal alles klein und mit Bindestrich getrennt). Die restlichen Attribute können theoretisch ignoriert werden.

Hinweis: Die Groß-/Kleinschreibung und die Trennung der einzelnen Wörter sind bei UI-Mods relevant. Der Ordner der UI-Mods beginnt als einziges Objekt mit einem großen Buchstaben.

Die app.js benutzt AngularJS (https://angularjs.org/). In der *app.js* wird die Funktionalität der UI-Mod umgesetzt. Außerdem können hier ein Template (also HTML) und CSS verwendet werden. HTML und CSS können entwerder in der *app.js* genutzt werden oder in externe Dateien ausgelagert werden. Dann müssen sie natürlich eingebunden werden.

In der *app.js* müssen in Zeile zwei und in der Funktion `scope.$on(‚destroy‘, function() {…` der Name der App in der entsprechenden Schreibweise eingetragen werden.

Zusätzlich zu der *.json*, *.js*, *.html* und *.css* kann noch eine *.png* hinzugefügt werden, die das Vorschaubild der App ist.

Die Einbindung von Streams wird in der *DummyApp* demonstriert. Eine Anleitung zur Erstellung von UI-Mods (inklusive der der Nutzung von Streams) kann unter https://wiki.beamng.com/My_First_App gefunden werden.

3.1.5 Fahrzeug

Das erstellen eines neuen Fahrzeugs wird in dieser Anleitung nicht beschrieben, wie man eine Fahrzeugkonfiguration erstellt um z.B. eine Fahrzeugmod standardmäßig zu nutzen (nötig für ein Szenario). Dazu existieren zwei unterschiedliche Wege, die auch kombiniert werden können. Die Möglichkeiten sind, eine Konfig in der Garage zu erstellen oder eine .pc Datei selbst zu erstellen bzw. eine vorhandene zu verändern.

Garage:

Die erste Möglichkeit eine Fahrzeugkonfiguration zu erstellen ist die Garage. Diese kann im Hauptmenü von BeamNG.drive geöffnet werden. Ob diese auch in BeamNG.research existiert ist zum Zeitpunkt der Erstellung dieser Anleitung ungewiss. Falls nicht muss Möglichkeit 2 genutzt werden.

In der Garage ist das in Abbildung 9 abgebildete Menü sichtbar, zu beginn ist der Reiter Vehicles ausgewählt. Links kann nun das Fahrzeugmodell und unten eine vorhandene Konfiguration als Vorlage ausgewählt werden. Unter Parts kann angepasst werden, welche Teile verwendet werden sollen. Unter Tune können verschiedene Eigenschaften verändert werden. Mit Paint lässt sich die Farbe des Fahrzeugs ändern. Unter My Carrs sind zuvor abgespeicherte Farzeugkonfigurationen einsehbar.

Abbildung 9: Menü in der Garage

Hinweis: Fahrzeugmods, welche Streams oder Objekte benötigen, die in der Garage nicht erzeugt werden, verursachen in der Garage Fehler. Es empfiehlt sich dann, die zweite Methode zu verwenden.

Wurde eine Fahrzeugkonfiguration mit der Garage erstellt, wurde die Datei ‚Documents\BeamNG.drive\vehicles\Fahrzeugname\Konfigname.pc' erstellt. Diese kann in das Fahrzeug verschoben werden.

Datei selbst erstellen:

Um eine Fahrzeugkonfiguration ohne die Garage zu erstellen, muss eine *.pc* Datei erstellt werden. Dazu empfiehlt es sich, eine vorhandene *.pc* Datei des Fahrzeugs zu kopieren und umzubenennen. Um eine Fahrzeugmod hinzuzufügen, muss eine Zeile ‚"Fahrzeugname_mod":"Fahrzeugname_modname",' hinzugefügt werden, z.B. ‚"coupe_mod":"coupe_exchange_and_log_data",' für das Fahrzeug ‚coupe' mit der Mod ‚exchange_and_log_data'.

Hinweis: Die einfachen Anführungszeichen (‚') gehören nicht zu der Zeile, sonder geben an, wo die Zeile beginnt und endet. Die doppelten Anführungszeichen („") hingegen gehören zu der Zeile und müssen übernommen werden.

Die fertige Datei muss in den Ordner des Fahrzeugs verschoben werden, sofern sie nicht bereits im Fahrzeug erstellt wurde.

3.1.6 Kamera

Für viele Experimente wird die Kamera *Onboard.Hood* geeignet sein. Ihr Vorteil der Onboard.Driver gegenüber besteht darin, dass sie nicht das Wippen und drehen des Kopfes simuliert. Allerdings kann das offset dieser Kamera in den Einstellungen nicht stark genug verändert werden um eine realistische Kopfposition darzustellen. Um die *Onboard.Hood* zu verändern muss die Datei *onboard.lua* (siehe 2) bearbeitet werden.

Das Field of View (fov) kann in Zeile 19 geändert werden, indem der Wert von `self.fov = self.fov + Wert` gesetzt wird.

Das Offset (also die Verschiebung) kann in den Zeilen 23-25 angepasst werden.

Die initiale Rotation der Kamera kann in Zeile 36 und 44 geändert werden.

Statt eine vorhandene Kamera zu modifizieren kann auch eine neue erzeugt werden. Wie dies gemacht werden kann wird in dieser Anleitung jedoch nicht erläutert.

3.2 Tipps und Tricks

3.2.1 Auslesbare Werte/Streams

Unter https://wiki.beamng.com/Streams ist eine Liste verfügbarer Streams einsehbar. Diese ist jedoch unvollständig und teilweise inkorrekt. Die in den Tabellen 2, 3, 4 und 5 enthaltenen Angaben sind auf Korrektheit überprüft.

Fahrphysik:

Parameter	LUA-Name	Einheit	Anmerkungen
Reifengeschwindigkeit	electrics.values.wheelspeed	m/s	
Luftgeschwindigkeit	electrics.values.airspeed	m/s	
Beschleunigung			Nicht direkt verfügbar
Gaspedalstellung	electrics.values.throttle		Von 0 bis 1
Bremspedalstellung	electrics.values.brake		Von 0 bis 1

Tabelle 2: Liste benötigter Fahrphysik-Parameter

Energiedynamik:

Parameter	LUA-Name	Einheit	Anmerkungen
Zustand der Zündung	controller.mainController.engineInfo [18]		0=Aus, 1=An
Momentane prozentuale Tankfüllung	electrics.values.fuel		Von 0 bis 1
Momentane Tankfüllung	electrics.values.fuelVolume	Liter	
Maximale Tankfüllung	electrics.values.fuelCapacity	Liter	
Momentaner Energiestand	energyStorage.getStorage('mainTank').storedEnergy	???	Einheit könnte kwh sein; statt mainTank muss bei Elektrofahrzeugen mainBattery genutzt werden
Maximaler Energiestand	energyStorage.getStorage('mainTank').initialStoredEnergy	???	Siehe Momentaner Energiestand
Batteriespannung			
Leistung des Motors			
Drehzahl des Motors	electrics.values.rpm	RPM	
Momentanverbrauch			Nicht direkt verfügbar
Restreichweite			
Sonstige Verbraucher			

Tabelle 3: Liste benötigter Energiedynamik-Parameter

Position:

Hinweis zur Benennung: Z ist die Höhe, X und Y spannen die Ebene auf.

Hinweis zu den Vektoren für die Ausrichtung: Ein Wert von 1 bedeutet, pro Einheit, den das Fahrzeug fährt, fährt es den Vektor-Wert in die entsprechende Richtung. X-Vektor = 1 bedeutet, dass das Fahrzeug entlang der X-Achse fährt, der X-Wert also pro gefahrener Einheit um 1 steigt, Y und Z Wert bleiben unverändert.

Parameter	LUA-Name	Einheit	Anmerkungen
Gefahrene Route			
Gefahrene Distanz			
X-Position	obj:getPosition().x	???	Einheit ist wahrscheinlich Meter (vom Ursprung entfernt)
Y-Position	obj:getPosition().y	???	Einheit ist wahrscheinlich Meter (vom Ursprung entfernt)
Z-Position	obj:getPosition().z	???	Einheit ist wahrscheinlich Meter (vom Ursprung entfernt)
X-Ausrichtung	obj:getDirectionVector().x		X-Vektor; Von -1 bis 1
Y-Ausrichtung	obj:getDirectionVector().y		Y-Vektor; Von -1 bis 1
Z-Ausrichtung	obj:getDirectionVector().z		Z-Vektor; Von -1 bis 1

Tabelle 4: Liste benötigter Positions-Parameter

Sonstiges:

Parameter	LUA-Name	Einheit	Anmerkungen
Fahrzeuggewicht	obj:calcBeamStats().total_weight	kg	
Streckenbeschaffenheit			

Tabelle 5: Liste sonstiger benötigter Parameter

3.2.2 Kommunikation zwischen Instanzen

In BeamNG.drive existieren verschiedene Instanzen, die für die Umsetzung einiger Funktionalitäten miteinander kommunizieren müssen. Die wichtigsten Instanzen sind das Fahrzeug (vehicle lua) und die Game Engine (ge lua). Außerdem stellen jede UI-Mod und ein Szenario jeweils eine eigene Instanz dar. Es existieren weitere Instanzen, diese scheinen jedoch minder relevant zu sein für Modifikationen wie wir sie brauchen.

In den folgenden Codezeilen steht … stellvertretend für einen vollwertigen Lua Befehl.

Mit `obj:queueGameEngineLua(‚...‘)` kann ein Befehl von dem Fahrzeug an die ge übergeben werden. Für die umgekehrte Richtung kann `playerVehicle:queueLuaCommand(‚...‘)` verwendet werden.

Von einem Szenario aus kann mit `helper.queueLuaCommandByName(‚scenario_player0‘,‘...‘)` ein Befehl an das Fahrzeug

gesendet werden. Befehle an die ge können einfach aus dem Szenario ausgeführt werden. Ein Befehl an das Szenario kann mit `scenario_demo_scenario.transmitSpeed('3')` ausgeführt werden, wobei `demo_scenario` der Name des Szenarios und transmitSpeed() eine Funktion in der demo_scenario.lua sind.

Hinweis: Um einen Befehl an ein Szenario auszuführen muss bei alten Versionen scenarios_name verwendet werden. Bei neueren Versionen muss stattdessen scenario_name genutzt werden.

Von einer UI-Mod können mit der *bngApi* Lua Befehle an das Fahrzeug übergeben werden. Um die *bngApi* zu nutzen muss diese zunächst in Zeile zwei einer UI-Mod eingebunden werden. Die Zeile könnte dadurch wie folgt aussehen:

```
.directive('blockVision', ['$log', 'bngApi', 'StreamsManager', function
($log, bngApi, StreamsManager) {
```

Anschließend kann im JavaScriptteil der UI-Mod der Befehl `bngApi.activeObjectLua(,...‘)` verwendet werden. Dieser Befehl an das Fahrzeug kann wiederum einen Befehl an eine andere Instanz enthalten.

Tipp: Häufig muss ein Befehl einen Umweg über eine andere Instanz gehen. Beachte dabei, dass evtl. ein \ vor bestimmte Zeichen wie z.B. , gesetzt werden muss, damit diese nicht auf der falschen Ebene des Befehl gewertet werden. Durch eine mehrfache Weitergabe eines Befehls können Abschnitte wie z.B. `,\‘\\\‘...\\\‘\‘‘` entstehen.

Es können (so weit ich das sehe) keine Befehle direkt an eine UI-Mod übergeben werden. Allerdings kann man indirekt Befehle in der UI-Mod ausführen lassen, indem man einen Stream in electrics.values definiert (siehe 3.2.3) und in der UI-Mod einen Befehl hinterlegt, der ausgeführt werden soll wenn der entsprechende Stream gesetzt ist.

Tipp: Erzeuge in der UI-Mod eine Variable pro Befehl, der durch einen Stream getriggert werden soll, um zu tracken, ob der Befehl bereits ausgeführt wurde. So kann verhindert werden, dass er zu oft ausgeführt wird. Den Stream mit der *bngApi* wieder zu löschen würde mehrere Ticks dauern und daher nicht das gewünschte Ergebnis erzielen.

Befehle an das Fahrzeug können von einer Fahrzeugmod normal ausgeführt werden (bei einigen Befehlen muss ein `obj:` davor hinzugefügt werden) und Streams normal ausgelesen werden. Mit einer Fahrzeugmod kann indirekt mittels Streams kommuniziert werden (vgl. UI-Mod, siehe 3.2.3)

Die direkten Kommunikationen sind in Abbildung 10 dargestellt.

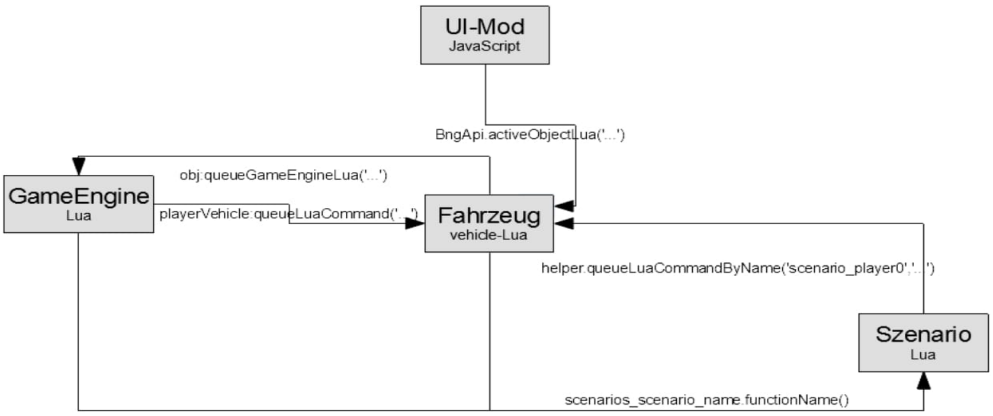

Abbildung 10: Kommunikation der Instanzen in BeamNG.drive

3.2.3 Nützliche Konsolenkommandos und Codezeilen

Der vielleicht nützlichste Konsolenbefehl ist `dump()`. Besonders `dump(extensions)` oder `dump(electrics.values)` ist oft nützlich. Mit `dump()` können einige Werte und Funktionsnamen ausgelesen werden, die mittels `print()` nicht angezeigt werden (tatsächlich habe ich keine einzige Information mit `print()` gefunden). Dieser Befehl ist sowohl im ge-Lua als auch im vehicle-Lua nützlich.

Ebenfalls sehr oft nützlich ist die Tatsache, dass man eigene Streams definieren kann. Dadurch kann man Variablen für alle Instanzen zugänglich machen und mit entsprechender Anpassung der Originaldateien sogar die Steuerung manipulieren (siehe nächster Punkt). Es empfiehlt sich, einen `electrics.values` zu definieren (Beispiel: `electrics.values.bla = 3`). Dies ist im vehicle-Lua möglich.

Mit `electrics.values.throttleOverride = x` kann der physische Input des Gaspedals verworfen und durch den Wert x ersetzt werden. Analog dazu wurde der Stream `electrics.values.brakeOverride` definiert. Durch eine Änderung in der entsprechenden Datei (siehe 2) kann nun auch der Bremsinput überschrieben werden. Mit `electrics.values.disableSteeringInput = 1` (ebenfalls in einer original Datei ergänzt; siehe 2) kann der Input des Lenkrads verworfen werden. Dies muss im vehicle-Lua geschehen.

Mit dem Befehl `vehicleSetPositionRotation()`, der 8 integer erwartet, kann das Fahrzeug teleportiert und rotiert werden. Wird die selbe Rotation wie beim letzten Teleport angegeben, wird das Fahrzeug nicht rotiert und behält seine Geschwindigkeit. Die Parameter sind (in dieser Reihenfolge) die ID des Fahrzeugs (siehe nächsten Punkt), die x-, y- und z-Koordinate, drei Rotationswinkel und einen Parameter, der immer auf 1 belassen werden kann. Dies ist ein Befehl der ge-Lua.

Mit `be:getPlayerVehicleID(0)` kann die ID des ersten Fahrzeugs ausgelesen werden, für andere Fahrzeuge muss die Zahl entsprechend angepasst werden. Dies ist ein Befehl der ge-Lua.

Die KI kann mit `ai.x` gesteuert werden, wobei x eine der Funktionen der KI ist. Die verschiedenen Funktionen der KI können mittels `dump(ai)` ausgelesen werden. Dies sind vehicle-Lua Befehle.

Mit `settings.setValue(„AudioMasterVol“, 0, nil, nil)` kann die Masterlautstärke auf 0 gesetzt werden, mit 1 entsprechend auf 1. Dies ist z.B. nützlich, um Instruktionen ohne

Umweltgeräusche zu präsentieren. Auch andere Einstellungen können mit dieser Funktion verändert werden. Dies ist ein ge-Lua Befehl.

3.2.4 Relevante Dateien

Zum finden von Funktionen, die über eine Taste bereits ausgeführt werden kann, ist die Datei ‚\lua\ge\input_actions.json' gut geeignet. In dieser werden die von den Tasten ausgelösten Events mit den Funktionen des Programms verknüpft.

In der Datei ‚\lua\vehicle\extensions\cruiseControl.lua' können die Befehle für die Cruise Control (Extension zum vorgeben der Geschwindigkeit) eingesehen werden.

In der Datei ‚\lua\ge\ge_utils.lua' können verschiedene Funktionen gefunden werden, die für die Fahrzeugkontrolle oder potentiell auch für Kollisionsdetektion nützlich sein können.

In der Datei ‚\lua\ge\bullettime.lua' können die Befehle für Pause und Zeitlupe gefunden werden. Außerdem könnten Änderungen beim eintreten einer Pause vorgenommen werden.

In der Datei ‚\lua\vehicle\electrics.lua' kann teilweise eingesehen werden, wo die Informationen in den electrics.values Streams herkommen. Auch andere Dateien in dem Pfad können dazu nützlich sein.

In der Datei ‚\lua\vehicle\controller\vehicleController.lua' wurde brakeOverride hinzugefügt (siehe 3.2.3). Hier können weitere Veränderungen vorgenommen werden, die den Umgang mit Inputs etc. manipulieren.

In der Datei ‚\lua\vehicle\input.lua' wurde disableSteeringInput hinzugefügt (siehe 3.2.3). Hier können weitere Änderungen vorgenommen werden, die den Umgang mit Inputs manipulieren.

Anhang C: Experimentalablauf

Grundinformationen:

Die wichtigsten Anforderungen werden unten abschließend noch einmal zusammengefasst

Bei jeder Anforderung steht in Klammern die ID der Anforderung im Anforderungsdokument

Rote Anforderungen sind nicht erfüllt, grüne sind erfüllt oder lassen sich (wahrscheinlich) leicht und schnell erfüllen

Abkürzungen: VL = Versuchsleiter

T = Teilnehmer

1 Vorbereitung eines Streckenszenarios

1 Erstelle oder wähle eine Strecke

a. Anforderung (006) /Wunsch: deutsche Strecke

2 Erstelle ggf. ein Auto (falls das nötige nicht vorhanden ist)

a. Anforderung (011)/Wunsch: Elektroauto

3 Konfiguriere die Mod exchange_and_log_data.lua (wird evtl. stattdessen in 6. erledigt)

a. Definiere, welche Parameter geloggt werden sollen

b. Definiere, welche Parameter an Matlab übergeben werden sollen

c. Definiere, was mit den Parametern von Matlab getan werden soll

4 Erstelle ein Szenario

a. Definiere Start-Spawn-Punkt für Streckenszenario

b. Definiere das Auto für Streckenszenario

    i. Definiere die Konfiguration des Autos

c. Definiere Waypoints

    i. Definiere, was an den Waypoints passiert

d. Definiere weitere Trigger

    i. Definiere, was bei den Triggern passiert

e. Definiere ggf. die Route der AI

f. Definiere Namen und Begrüßungstext des Szenarios

5 Erstelle oder wähle die nötigen GUI-Mods

6 Definiere allgemeine Einstellungen in settings.json

a. Anforderung (025) & (026): Einstellungen können aus Datei ausgelesen und dort eingespeichert werden

2 Vorbereitung bei Übernahme von einem anderen Rechner

1 Voraussetzungen:

a. BeamNG.drive & Matlab müssen installiert sein

b. Lenkrad und Pedale müssen installiert und konfiguriert sein

2 Hardware-Setup:

a. Laptop: MSI GT73EVR 7RE-838DE Titan

b. Lenkrad: ClubSport Lenkrad Classic von Fanatec

c. Wheelbase: ClubSport Wheel Base V2.5 von Fanatec

d. Pedale: ClubSport Pedale V3 invertiert von Fanatec

3 Folgende Dateien/Ordner müssen von Rechner A auf Rechner B übergeben werden:

a. Ordner ui in Dokumente\BeamNG.drive

    i. Alternativ: Unterordner Vorhang (und ggf. weitere)

b. Ordner mods in Dokumente\BeamNG.drive

    i. Alternativ: Unterordner exchangeAndLogging, Unterordner des Szenarios und Datei 200ex.zip (und ggf. Datei Streckenname.zip)

c. Datei settings.json in Dokumente\BeamNG.drive

d. Datei onboard.lua in lua\ge\extensions\core\cameraModes

e. Ggf. Datei Streckenname.zip in content\levels (evtl. stattdessen im Ordner Mods (siehe b) enthalten)

4 Vorbereitungen vor einem Versuchsdurchgang

1 VL startet Matlab (oder ggf. andere für ergänzende Funktionen nötige Zusatzprogramme)

2 VL startet das Matlabskript

3 VL startet BeamNG.drive

4 VL wählt „Szenario“ und startet das richtige Szenario

5 VL aktiviert ggf. UI-Mods (Vorhang, Instruktionen, etc. und zu testende UIs)

5 Übungsfahrt (z.B. hier 2x die gleiche Strecke abfahren)

6 T kommt an

7 Ggf. schriftliche Instruktion / Einführungsvideos / … jenseits von beamNG.drive

8 T erhält auf dem Bildschirm die Instruktion, sich zunächst durch freie Fahrt mit dem Simulator vertraut zu machen. Ein Button "Start" ist vorhanden.

9 T klickt den Button "Start" an

10 Der Instruktionsbildschirm verschwindet, Sound startet, Strecke ist zu sehen, Fahrzeug wird automatisch & sofort auf Startgeschwindigkeit gesetzt

a. Anforderung (034): Startgeschwindigkeit kann gesetzt werden

b. Anforderung (036): Sound kann temporär deaktiviert werden

11 Auf ersten zwei Kilometern fährt Fahrzeug vollautomatisch

a. Anforderung (037): Route und Fahrverhalten einer automatischen Fahrt können festgelegt werden

12 Nach 1,6 km erhält T Instruktion (Audio oder transparentes Overlay), ab Streckenelemente xy (z.B. „dem Baum da vorne“) die Kontrolle zu übernehmen

a. Anforderung (038): Audioinstruktionen können abgespielt werden

b. Anforderung(039): Transparente Overlays

c. Anforderung (041): unsichtbare Waypoints als Trigger

d. Anforderung (042): automatische Fahrt kann in manuelle Fahrt übergehen

13 Nach 3 km friert die Fahrt ein und eine Instruktion wird angezeigt

a. Anforderung (044): Simulation kann automatisch/per Trigger einfrieren

14 Anschließend wird das Fahrzeug wieder an den Start der Strecke gesetzt und fährt nochmal die gleiche Strecke

a. Variation: Fahrzeug wird an einen anderen Ort gesetzt

i. Anforderung (030): Fahrzeug kann zu beliebigem Zeitpunkt an beliebigen Ort gesetzt werden

15 Übungsfahrt wird beendet

a. Variation: Fahrt zu bestimmten Streckenpunkt und vorher Instruktion „Fahrzeug dort abstellen / Streckenpunkt durchfahren“

b. Ansatz: Ausblenden wenn an Streckenpunkt/Trigger erfüllt

i. Anforderung (045): Fade-Outs von UIs sind möglich

16 VL überprüft, ob Daten der freien Fahrt korrekt geloggt wurden

6 Experiment Teil 1

17 T erhält Anweisung, so effizient wie möglich von null auf 50 zu beschleunigen. Ein Countdown läuft von 15 abwärts, startet bei null die Sequenz und kann per Button pausiert werden.

18 T beschleunigt auf 50 km/h

19 Beim erreichen von 50 km/h wird der Instruktionsbildschirm im Verlauf von zwei Sekunden eingeblendet

a. Anforderung (045): Fade-Ins von UIs sind möglich

b. Anforderung (046): Geschwindigkeit als Trigger

20 T wird gebeten, die Energieeffizienz seiner Beschleunigung auf einer Skala von eins bis zehn zu bewerten. Dazu ist ein Textfeld vorhanden und ein Button zum Bestätigen der Eingabe.

a. Anforderung (047): Es können Werte eingegeben werden, die in einer Datei gespeichert werden

21 T gibt einen Wert ein und klickt auf den Button

7 Experiment Teil 2

22 T erhält die Instruktion im folgenden eine Beschleunigung anzuschauen, bei der er nur lenken muss, und anschließend die Effizienz zu beurteilen. Ein Countdown läuft von 15 abwärts, startet bei null die Sequenz und kann per Button pausiert werden.

23 Das Auto beschleunigt entsprechend der Werte in einer CSV-Datei von null auf 50

a. Anforderung (024): Geschwindigkeiten aus einer CSV-Datei können umgesetzt werden

    i. Alternativanforderung (022): Geschwindigkeiten von Matlab können umgesetzt werden

24 Beim erreichen von 50 km/h wird der Instruktionsbildschirm im Verlauf von zwei Sekunden eingeblendet

25 T wird gebeten, die Energieeffizienz der Beschleunigung auf einer Skala von eins bis zehn zu bewerten. Dazu ist ein Textfeld vorhanden und ein Button zum Bestätigen der Eingabe.

26 T gibt einen Wert ein und klickt auf den Button

8 Experiment Teil 3-X

27 (Abschnitt 7 wird wiederholt, bis alle vorgegebenen Sequenzen gefahren wurden. Die Reihenfolge ist zufällig und wird in einer Datei abgespeichert.)

a. Anforderung (048): Reihenfolgen können randomisiert werden

    i. Alternativ kann Matlab die Reihenfolge randomisieren

9 Ende

28 Auf dem Bildschirm ist eine Dankes-Floskel zu sehen mit dem Hinweis, dass der Simulatorteil des Experiments abgeschlossen ist

29 Ggf. Fragebögen etc. außerhalb von BeamNG.drive

30 T geht

31 VL beendet BeamNG.drive (oder geht zumindest ins Hauptmenü)

32 VL beendet Ausführung des Matlabskriptes

33 VL schiebt alle entstandenen Dateien (Logger-Daten, Datei mit den eingegebenen Werten, Reihenfolge der Sequenzen) in einen Ordner, der die Nummer des T hat

a. Optimal: Alle Dateien sind automatisch im selben Ordner

i. Anforderung (049): Bei jedem Durchlauf wird ein anderer Ordner verwendet

10 Grundelemente von Fahrsimulatorexperimenten

- Trigger

1. Streckenabschnitts-Trigger/Waypoints (wenn an bestimmten xy-Koordinaten, dann…) (040)

1. unsichtbare Streckenabschnittstrigger (041)

2. Gefahrene Distanz-Trigger (wenn x km gefahren, dann…) (050)

3. Zeit-Trigger (wenn x Minuten seit Start vergangen, dann…) (051)

4. Geschwindigkeits-Trigger (wenn x km/h über/unterschritten, dann…) (046)

5. Kombinierbar/Bedingbar (wenn zum zweiten mal an Position xy, dann... oder wenn min. x km und y min gefahren, dann…) (052)

- Spawning
    1. An beliebigen Punkten (030)
    2. Mit beliebiger Startgeschwindigkeit (034)
- Instruktionen
    1. Transparent, Intransparent (039)
    2. Vollbild, Partiell (053)
    3. Langsam ein- oder ausgeblendet (045)
    4. Audio (038)
    5. Mit & ohne Simulationspause/freeze (043), Sound kann deaktiviert werden (036)
    6. Instruktionsbildschirme können Eingaben entgegennehmen (047)
- Automatische Fahrzeugsteuerung
    1. Geschwindigkeit vorgeben (Fahrzeug beschleunigt genau auf diese) (024)
    2. Geschwindigkeit setzen (Fahrzeug hat plötzlich diese) (035)
    3. Geschwindigkeitsverlauf vorgeben (024) (alternativ Werte aus Matlab verwenden (022))
    4. Route vorgeben (037)
    5. automatische Fahrt kann in manuelle übergehen (042)
- Datenlogging
    1. Zu jedem definierten Intervall (frei definierbar) werden x Datenkanäle ausgelesen (019 & 054)
    2. Positionsdaten (absolute xy-Koordinaten, relativ zu Elementen (z.B Mittel/Seitenlinie)) (061 & 073)
    3. Dauer des Versuchs (gefahrene km, verstrichene Zeit) (070 & 074)
    4. Fahrdaten (Geschwindigkeit, Beschleunigung, Pedalstellungen) (055, 056, 064)

5. Energiedaten (Energiestand, Restreichweite, Momentanverbrauch) (060, 067, 068)
6. Fahrzeugdaten (Leistung des Motors, Spannung der Batterie (bei Elektroauto)) (065 & 066)
7. Alle geloggten Daten werden an einem Ort gespeichert (049)

- Allgemeines
  1. Einstellungen etc. können exportiert und importiert werden (025 & 026)
  2. Abschnitte werden in zufälliger Reihenfolge wiedergegeben (Reihenfolge wird geloggt) (048)
  3. Antriebs des Fahrzeuges (elektrisch und später auch hybrid) (011 & 012)
  4. Deutsche Strecke (006)
     1. mit verschiedenen Settings (007, 008, 009, 010)

Anhang D: centralInformation.json


```
{
      "scenario": {
            "map": "west_coast_usa",
            "finish": "WP2"
            "vehicle": {
                  "model": "200ex",
                  "config": "vehicles/200ex/300.pc",
                  "spawnpoint": "-683.240417 596.829956 111.999535",
                  "color": "1 0 0.823529005 0.405000001"
            },
            "waypoints": {
                  "waypoint1": {
                        "name": "WP1",
                        "position":       "-593.058533       624.168518
90.8574219",
                        "visible": "false",
                        "scale": "5 5 5"
                  },
                  "waypoint2": {
                        "name": "WP2",
                        "position":       "-631.208801       661.866882
91.5160217",
                        "visible": "false",
                        "scale": "5 5 5"
                  }
            }
```

```
},
"instructions": {
      "message1": {
            "in": {
                  "triggerType": "waypoint",
                  "triggerValue": "WP1",
                  "method": "fade",
                  "fadeValue": "3"
            },
            "text": "Bitte links abbiegen.",
            "out": {
                  "triggerType": "relativeTimer",
                  "triggerReference": "WP1",
                  "triggerValue": "7",
                  "method": "fade",
                  "fadeValue": "1"
            }
      }
},
"dataLogging": {
      "parameter1": "electrics.values.airspeed",
      "parameter2": "electrics.values.wheelspeed"
},
"dataExchange": {
      "BeamNGOutput": {
            "parameter1": "electrics.values.wheelspeed",
            "parameter2": "electrics.values.throttle"
```

```
            },
            "MatlabOutput": {
                  "parameter1": {
                        "type": "string",
                        "value": "speed"
                  },
                  "parameter2": {
                        "type": "lua",
                        "value": "electrics.values.MatlabOne"
                  }
            }
      },
      "camera": {
            "type": "onboard",
            "xOffset": "-0.081",
            "yOffset": "0.175",
            "zOffset": "-0.035",
            "fov": "5.4",
            "xRotation": "7",
            "yRotation": "-2",
            "zRotation": "0"
      }
}
```

Anhang E: dataLogging.lua

```
-- Note: The next three lines are for programmers only. You can ignore
them.

-- This Source Code Form is subject to the terms of the bCDDL, v. 1.1.

-- If a copy of the bCDDL was not distributed with this

-- file, You can obtain one at http://beamng.com/bCDDL-1.1.txt


local M = {}

M.type = "auxilliary"

M.relevantDevice = nil

local fh = nil


local timer = 0

local timerOld = 0

-- ###Editmarker 01###

-- add additional variables

-- Example: local speedOld = 0

-- ***Begin Editblock 01***

local speedOld = 0

-- ***End Editblock 01***


local function init()

    timer = 0

      timerOld = 0

      -- ###Editmarker 02###

      -- set additional variables to 0
```

```
      -- Example: speedOld = 0

      -- ***Begin Editblock 02***

      speedOld = 0

      -- ***End Editblock 02***


    if fh then fh:close() end

    fh = io.open(os.date("%Y.%m.%d-%H.%M.%S.csv"), "w")

      -- ###Editmarker 03###

      -- set the first line of the .csv file

      -- Important: Only edit the part between " and \n"

      -- Note: With -- you can disable this line to begin the .csv
with the first set of values

      -- ***Begin Editblock 03***

      fh:write(  "time,airspeed,wheelspeed,kmh,rpm,weight,fuel,accele-
ration\n")

      -- ***End Editblock 03***

end


local function updateGFX(dtime)

      timerOld = timer

    timer = timer + dtime


    if fh then

        local str = ""

        str = tostring(timer)..","

            -- ###Editmarker 04###

            -- set the values you want to log

            -- Important: Make sure to keep the format
```

```
            -- Important: End the last of these lines with "\n" but
all the other with ","

            -- Example: str = str..tostring((electrics.values.airspeed
or 0) )..","

            -- ***Begin Editblock 04***

        str = str..tostring((electrics.values.airspeed or 0) )..","

        str = str..tostring((electrics.values.wheelspeed or 0) )..","

            str = str..tostring((electrics.values.wheelspeed*3.6 or 0)
)..","

        str = str..tostring((electrics.values.rpm or 0) )..","

            str  =  str..tostring((obj:calcBeamStats().total_weight  or
0) )..","

        str = str..tostring((electrics.values.fuel or 0) )..","

            str   =   str..tostring(((electrics.values.wheelspeed-spee-
dOld)/(timer-timerOld) or 0) )..."\n"

            -- ***End Editblock 04***

        fh:write( str )

    end


      -- ###Editmarker 05###

      -- set variables that shall keep old values here

      -- Example: speedOld = electrics.values.wheelspeed

      -- ***Begin Editblock 05***

      speedOld = electrics.values.wheelspeed

      -- ***End Editblock 05***


end


M.init = init

M.reset = init
```

```
M.updateGFX = updateGFX


return M
```

Anhang F: example_scenario.lua

```
-- This Source Code Form is subject to the terms of the bCDDL, v. 1.1.
-- If a copy of the bCDDL was not distributed with this
-- file, You can obtain one at http://beamng.com/bCDDL-1.1.txt

local M = {}
--default scenario variables, no need to touch
local helper = require('scenario/scenariohelper')
local running = false
local playerWon = false
--these hold the values and the order in which the values are used
local values = {}
local seq = {}
--these variables are relevant for timing how long a certain speed is
held
local currentlyTiming = false
local currentlyTimed = 0
local timingStarted = 0
local timingStopped = nil
--configure which values of the csv shall be first, last and randomi-
zed
local seqStart = {}
local seqRandom = {1,2,3,4,5,6,7,8,9,10}
local seqEnd = {}
--configure how many runs (Bedingungen) and blocks per run to do
local maximumCounter = 5
local maximumBlock = 2
--these variables regulate the scenario
local currentSeqNumber = -1
local currentSeqState = 1
local currentCounter = 1
local currentBlock = 1
local triggered = 0
local aFirst = true
local blockFinished = true
local kmCompleted = false

--this function initializes the scenario
local function init()
      --decide by random whether display A or B is displayed first
      if math.random()<0.5 then
            aFirst = false
      end
      --read the values from a csv
      io.input("../../../../../../example.csv")
      local cancel = false
      local c = 1
      while not cancel do
```

```
            dump('do')
            local readVal = io.read(3)
            io.read(2)
            local readValTwo = io.read(3)
            io.read('*l')
            dump(readVal)
            dump(readValTwo)
            if readVal==nil then
                  cancel=true
            else
                  values[c] = {tonumber(readVal), tonumber(readValT-
wo)}
                  c = c+1
            end
      end
      --set the simulationspeed to 0.001
      bullettime.set(0.001)
end

-- this function shuffles the elements of the table seqRandom into the
table seq
-- and add seqStart and seqEnd to it
local function shuffle()
      --actual shuffeling
      math.randomseed(os.time())
      local output = { }
      local tmpTable = { }
      for i = 1, #seqRandom do
            table.insert(output, seqRandom[i])
      end
      for i = #seqRandom,1,-1 do
            local rand = math.random(#seqRandom)
            output[i], output[rand] = output[rand], output[i]
      end
      dump(seqStart)
      --write the correct sequence to seq
      seq = {}
      for k,v in ipairs(seqStart) do
            table.insert(seq,v)
      end
      for k,v in ipairs(output) do
            table.insert(seq,v)
      end
      for k,v in ipairs(seqEnd) do
            table.insert(seq,v)
      end
      dump('seq:')
      dump(seq)
end

--this function resets everything except for the controlvariables
local function resetSoft()
      --tell the UI a reset has happened
```

```
      --if the maximum amount of runs was alreadys done, block the vi-
sion and disable the vehicle
      if currentCounter == maximumCounter+1 then
            helper.queueLuaCommandByName('scenario_player0', 'elec-
trics.values.throttleOverride=0')
            helper.queueLuaCommandByName('scenario_player0', 'elec-
trics.values.brakeOverride=0')
            helper.queueLuaCommandByName('scenario_player0', 'elec-
trics.values.disableSteeringInput=1')
            settings.setValue("AudioMasterVol",0,nil,nil)
      else--else make sure the vehicle is enabled
            helper.queueLuaCommandByName('scenario_player0', 'elec-
trics.values.throttleOverride=nil')
            helper.queueLuaCommandByName('scenario_player0', 'elec-
trics.values.brakeOverride=nil')
            helper.queueLuaCommandByName('scenario_player0', 'elec-
trics.values.disableSteeringInput=nil')
            settings.setValue("AudioMasterVol",1,nil,nil)
      end


      ---Teleport
      vehicleSetPositionRotation(be:getPlayerVehicleID(0),288.963,
1799.71, 1.26251,1,1,1.1,1)
      vehicleSetPositionRotation(be:getPlayerVehicleID(0),288.963,
1799.71, 1.26251, 0, 0, 65, -64.6619)



      --TODO: Countdown

      --continue normal driving
      bullettime.set(1)
end

--this function starts the next block
--if maximum block for this run is hit, go to the next run
local function nextBlock()
      blockFinished = false
      --check if only block is increased or the run
      if currentBlock == maximumBlock then
            currentBlock=1
            currentCounter = math.min(currentCounter +1,maximumCoun-
ter+1)
      else
            currentBlock = currentBlock+1
      end
      --reset some of the controlvariables
      currentSeqNumber = 1
      currentSeqState = 1
      --shuffle
      shuffle()
      --tell the UI about the block increasment
```

```
      helper.queueLuaCommandByName('scenario_player0', 'elec-
trics.values.triggered='..triggered)
      helper.queueLuaCommandByName('scenario_player0', 'elec-
trics.values.currentSeqNumber='..currentSeqNumber)
      helper.queueLuaCommandByName('scenario_player0', 'elec-
trics.values.aFirst='..tostring(aFirst))
      helper.queueLuaCommandByName('scenario_player0', 'elec-
trics.values.currentCounter='..currentCounter)
      helper.queueLuaCommandByName('scenario_player0', 'elec-
trics.values.maximumCounter='..maximumCounter)
      helper.queueLuaCommandByName('scenario_player0', 'elec-
trics.values.currentBlock='..currentBlock)
      helper.queueLuaCommandByName('scenario_player0', 'elec-
trics.values.maximumBlock='..maximumBlock)
      helper.queueLuaCommandByName('scenario_player0', 'elec-
trics.values.targetSpeed='..values[seq[currentSeqNumber]][currentSeq-
State])
      helper.queueLuaCommandByName('scenario_player0', 'elec-
trics.values.currentRow='..seq[currentSeqNumber])
      helper.queueLuaCommandByName('scenario_player0', 'elec-
trics.values.currentColumn='..currentSeqState)
      helper.queueLuaCommandByName('scenario_player0', 'elec-
trics.values.bIsDisplayed='..tostring((currentCounter==4 and not
aFirst) or (currentCounter==5 and aFirst)))
      helper.queueLuaCommandByName('scenario_player0', 'elec-
trics.values.aIsDisplayed='..tostring((currentCounter==4 and aFirst)
or (currentCounter==5 and not aFirst)))
end

--this function is called by the e-key on the keyboard and the b-but-
ton of the steering wheel
--it starts the next block if the current one is done
local function attemptedReset()
      if not (currentCounter>maximumCounter) then
            if blockFinished then
                  nextBlock()
                  resetSoft()
            end
      end
end

--this function is called by the t-key on the keyboard
--it resets the current block with a new shuffle
local function forceReset()
      if not (currentCounter>maximumCounter) then
            currentBlock=currentBlock-1
            nextBlock()
            resetSoft()
      end
end

--this function is called by a hard-reset of the scenario (r-key)
local function reset()
```

```
    running = false
    playerWon = false

      init()

      helper.queueLuaCommandByName('scenario_player0', 'elec-
trics.values.currentCounter='..currentCounter)
      helper.queueLuaCommandByName('scenario_player0', 'elec-
trics.values.currentBlock='..currentBlock)
      helper.queueLuaCommandByName('scenario_player0', 'elec-
trics.values.maximumCounter='..maximumCounter)
      helper.queueLuaCommandByName('scenario_player0', 'elec-
trics.values.maximumBlock='..maximumBlock)
      helper.queueLuaCommandByName('scenario_player0', 'elec-
trics.values.currentSeqNumber='..currentSeqNumber)
end

local function fail(reason)
    reset()
    scenario_scenarios.finish({failed = reason})
end

local function success()
  local scenario = scenario_scenarios.getScenario()
  if not scenario then return end

  local finalTime = scenario.timer
  local minutes = math.floor(finalTime / 60);
  local seconds = finalTime - (minutes * 60);
  local timeStr = ''
  if minutes > 0 then
      timeStr = string.format("%02.0f:%05.2f", minutes, seconds)
  else
      timeStr = string.format("%0.2f", seconds) .. 's'
  end
  local result = {msg =
"scenarios.jungle_rock_island.drag_race.win.msg", timeStr}
  scenario_scenarios.finish(result)
  reset()
end

local function onRaceInit()
end

local function onRaceStart()
    reset()
    running = true
end

--this function is called by the vehicle thanks to the onRaceTick
function
--it recevies the speed and evaluates it
local function transmitSpeed(speed)
```

```
	--if currentSeqNumber<=0 there is currently no sequence running
an therefor no further testing is required
	if currentSeqNumber>0 then
		--if the timing is not stopped
		if timingStopped==nil then
			--but currently it is not timing
			if not currentlyTiming then
				--check if the current speed (floored) equals
the target speed
				if math.floor(speed*3.6)==values[seq[current-
SeqNumber]][currentSeqState] then
					--if so, start timing
					currentlyTiming = true
					currentlyTimed = 0
					timingStarted = os.time()
				end
			--if timing is not stopped but currently not timing
			else
				--if within a certain range (+/- 5) of the
target speed, sontinue timing
				if speed>(values[seq[currentSeqNumber]][cur-
rentSeqState]-5)/3.6 and speed<(values[seq[currentSeqNumber]][current-
SeqState]+5)/3.6 then
					currentlyTimed = os.time() - timingStar-
ted
				--if not within ght range of the target speed,
stop timing
				else
					currentlyTiming = false
				end
			end
			--if atleast two seconds were timed, do certain
stuff
			if currentlyTimed>=2 then
				dump('TRIGGGGGGERED'..speed*3.6)
				timingStopped = os.time()
				currentlyTimed=0
				--if the first column of a row was last used,
use the second now
				if currentSeqState==1 then
					currentSeqState=2
				else
					--else (column two was just reached)
check if final row was used
					if currentSeqNumber<#seq then
						--if not, use next row

	currentSeqNumber=currentSeqNumber+1
						--if old target-speed does not
equal new start-speed, use new startspeed else skip it
						if not (values[seq[currentSeqNum-
ber-1]][2]==values[seq[currentSeqNumber]][1]) then
							currentSeqState = 1
```

```
                                    end
                              --if final row was used
                              else
                                    --check if current block is the
maximum block, if so start next block
                                    if currentBlock<maximumBlock then
                                          nextBlock()
                                    --else tell the UI to block the
vision etc.
                                    else
                                          currentSeqNumber=-1
                                          bullettime.set(0.001)
                                          blockFinished = true
                                          helper.queueLuaCommandByNa-
me('scenario_player0',
'electrics.values.blockFinished='..tostring(blockFinished))
                                    end
                              end
                        end
                        --since a new value is to be displayed, update
a few values, including the versions for the UI

      helper.queueLuaCommandByName('scenario_player0',


 'electrics.values.currentSeqNumber='..currentSeqNumber)
                        triggered = triggered+1

      helper.queueLuaCommandByName('scenario_player0', 'elec-
trics.values.triggered='..triggered)
                        --if a new speed is to be driven, set the cor-
rect values
                        if currentSeqNumber>-1 then

      helper.queueLuaCommandByName('scenario_player0', 'elec-
trics.values.targetSpeed='..values[seq[currentSeqNumber]][currentSeq-
State])

      helper.queueLuaCommandByName('scenario_player0', 'elec-
trics.values.currentRow='..seq[currentSeqNumber])

      helper.queueLuaCommandByName('scenario_player0', 'elec-
trics.values.currentColumn='..currentSeqState)

      helper.queueLuaCommandByName('scenario_player0', 'elec-
trics.values.bIsDisplayed='..tostring((currentCounter==4 and not
aFirst) or (currentCounter==5 and aFirst)))
                              dump(((currentCounter==4 and not aFirst)
or (currentCounter==5 and aFirst)))


dump('electrics.values.bIsDisplayed='..tostring((currentCounter==4 and
not aFirst) or (currentCounter==5 and aFirst)))
```

```
      helper.queueLuaCommandByName('scenario_player0', 'elec-
trics.values.aIsDisplayed='..tostring((currentCounter==4 and aFirst)
or (currentCounter==5 and not aFirst)))
                        end

      helper.queueLuaCommandByName('scenario_player0', 'elec-
trics.values.currentSeqNumber='..currentSeqNumber)
                  end
            --reset the timer to work again for the next attempt
            else
                  if timingStopped+2<os.time() then
                        timingStopped=nil
                        currentlyTiming=false
                  end
            end
      end
end

--this function is called by the vehicle and tells the scenario that a
km was driven (and therefore instructions shall be shown)
local function setKmCompleted()
      kmCompleted=true
      helper.queueLuaCommandByName('scenario_player0', 'elec-
trics.values.showSpeed='..tostring(true))
      dump('kmCompleted')
end

--this function triggers on every raceTick
--it tells the vehicle to transmit either the speed or the distance
travelled
local function onRaceTick(raceTick)
      local scenario = scenario_scenarios.getScenario()
      if not scenario then return end
      if not kmCompleted then
            be:getPlayerVehicle(0):queueLuaCommand("if electrics.valu-
es.distanceTravelled>1000 then obj:queueGameEngineLua(\'scenario_exam-
ple_scenario.setKmCompleted()\') end")
      else
            be:getPlayerVehicle(0):queueLuaCommand("obj:queueGameEngi-
neLua(\'scenario_example_scenario.transmitSpeed(\'.. tostring(elec-
trics.values.wheelspeed) ..\')\')")
      end
end

local function onRaceWaypointReached(data, goal)
      if data.waypointName == 'WP1' then
            be:getPlayerVehicle(0):queueLuaCommand("electrics.valu-
es.WP1 = 1")
      end
end

local function onBeamNGTrigger(data)
```

```
end

local function onRaceResult(time)
    if playerWon then
        success()
    else
        fail("scenarios.jungle_rock_island.drag_race.fail.msg")
    end
    --helper.queueLuaCommandByName('aiCar', 'dragAi.reset()')
end

local function onScenarioRestarted(scenario)
   -- helper.queueLuaCommandByName('aiCar', 'dragAi.reset()')
end

M.onRaceStart = onRaceStart
M.onRaceInit = onRaceInit
M.onRaceTick = onRaceTick
M.onRaceResult = onRaceResult
M.onRaceWaypointReached = onRaceWaypointReached
M.onScenarioRestarted = onScenarioRestarted
M.transmitSpeed = transmitSpeed
M.onBeamNGTrigger = onBeamNGTrigger
M.attemptedReset = attemptedReset
M.skipBlock = skipBlock
M.forceReset = forceReset
M.setKmCompleted = setKmCompleted

return M
```

Anhang G: dataExchange_v02.m

```
%declare variables

%x=attempts

x = 0;

%y=successful read

y = 0;

%z=successful write

z = 0;

%c=repeated problems

c = 0;

%these variables help with the detection of repeated problems

gapOld =0;

marker = 0;

timer = 0;


%the IPC runs until shut down

while true

    %test if the last attemtp had a problem

    if x-z ~= gapOld

        %if it had and the one before had as well, increase c

        if marker==1

            c=c+1;

        end

        marker=1;

    else

        marker=0;

    end
```

```
    %calculate the gap between the attempts and the successful writes
    gapOld = x-z;
    x= x+1;
    try
        %try to read the csv from BeamNG
                                                                M       =       cs-
vread("C:\Users\astudent\Documents\BeamNG.drive\output_BeamNG.csv",0,0);
        pause(0.01);
        y= y+1;


        %try to write a csv file using the read data
         csvwrite("C:\Users\astudent\Documents\BeamNG.drive\output_MatLab.csv"
, [Speed(1,x);100+M(1,1)*0.25*mod(M(1,3),10)]);


        disp(M);
        z= z+1;


        %display the current stats
        disp("Pause: 0.01");
        disp("x=" + x + "; y=" + y + "; z=" + z + "; c=" + c);
    catch ME
    end
end
```

Erklärung

Ich versichere an Eides statt, die vorliegende Arbeit selbstständig verfasst und nur die angegebenen Quellen benutzt zu haben.

Unterschrift des Studierenden

Lübeck, Tagesdatum