Abschlussprüfung Sommer 2020

Fachinformatiker Anwendungsentwicklung

Dokumentation zur betrieblichen Projektarbeit

Vernetzte Fahrzeuge in einer AR Simulation

Unfallsituation in einer Verkehrssimulation

Abgabedatum: München, den 26.05.2020

Prüfungsbewerber:

Felicitas Hank

Eschenstr. 64

85716 Unterschleißheim

Ausbildungsbetrieb:

MAGNA Telemotive GmbH

Frankfurter Ring 115a

80807 München

Inhaltsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

MTM ..... *Magna Telemotive*
AR ..... *Augmented Reality*
TSR ..... *Test Summary Report*
SDK ..... *Software Development Kit*

1 Einleitung

1.1 Projektumfeld

Die folgende Projektdokumentation beschreibt den Ablauf des betrieblichen Auftrags zur IHK Abschlussprüfung im Sommer 2020, durchgeführt im Rahmen der Ausbildung zur Fachinformatikerin für Anwendungsentwicklung. Der betriebliche Auftrag wurde beim Ausbildungsunternehmen (MAGNA Telemotive GmbH) ausgeführt.

Die Magna Telemotive AG, im nachfolgenden MTM genannt, ist eine Tochterdivision von Magna International Inc., welche seit Jahrzehnten für diverse OEMs in der Automobilbranche tätig ist.
Um den technischen Stand und die Innovationen von MTM auf dem Markt präsentieren zu können, sind Messen eine gute Möglichkeit insbesondere die Kompetenzen in den Bereichen Extended Reality (XR) und der Vernetzung von Automobilen dem potentiellen Kunden näher zu bringen.
Eine Augmented Reality (AR) Simulation, welche von der MTM Vorentwicklung umgesetzt wurde, wird zu diesem Zweck häufig auf Messen vorgestellt, hat dabei aber Verbesserungsbedarf gezeigt. Dabei kollidierten die Fahrzeuge immer wieder – diese Problematik wurde in einer anderen Projektarbeit behoben.
Eine weitere Verbesserung ist es, die Simulation für den Messestandbetreuenden intuitiver zu gestalten, damit man das volle Präsentationspotential nutzen kann.
Doch der größte Verbesserungspunkt ist, dass die Simulation bisher nur ohne weitere Vorkommnisse fahrende Fahrzeuge zeigt, und damit der Realität widerspricht.
Für ein realistisch wirkendes Szenario hat sich eine Unfallsituation als vielversprechend erwiesen, da dabei die Interaktion zwischen den individuellen Fahrzeugen eine Schlüsselrolle spielt.
Diese Erweiterung hat auch bei der Mutterdivision Magna Steyr Anklang gefunden, da diese sich seit der ersten Messedemonstration ein aussagekräftiges Szenario wünschten.

Diese Simulation wird an Hand eines 3D gedruckten Straßen – bzw. Kreuzungsmodell gezeigt.

1.2 Projektziel

Das Ziel des Projektes ist es, eine gesteigerte Erlebbarkeit der Simulation über ein von dem User auslösbares Szenario zu bieten, welches über eine physische Handlung (Setzen eines AR-Markers) sowie über ein Sprachkommando ausgelöst werden kann.
Die nutzerfreundliche Bedienbarkeit für den potentiellen Kunden sowie für den Mitarbeiter am Messestand soll auch verbessert und sichergestellt werden.

1.3 Projektbegründung

Der Primärnutzen der Erweiterung der Simulation ist es, die vorhandene Verkehrssimulation zu verbessern und intuitiv bedienbar zu machen, sowie die technischen Innovationen MTMs für Kunde und Konkurrenz sichtbar zu machen.
Dafür ist eine fehlerfreie und featurehaltige Präsentation unabdingbar, welche gut für den Präsentierenden zu bedienen ist um einen professionellen und souveränen Eindruck bei dem potentiellen Kunden zu hinterlassen.
Da die Unfallsimulation auch auf einen Vorschlag des Mutterkonzerns Magna Steyr eingeht, werden nicht nur Potentiale der vernetzen Automobile und die Fähigkeiten im Bereich XR gezeigt, sondern auch die Kooperation im Gesamtkonzern für technische Innovation.

Das Verfassen einer Testdokumentation und einer Vorlage für einen TSR (Test Summary Report) sorgt für einen qualitativen Erhalt der Simulation sowie professionelles Fehlermanagment, wodurch die Maintainability des Projektes und die Qualität sichergestellt ist.

1.4 Projektschnittstellen

Technische Schnittstellen:

Vuforia AR Library
Unity Framework
Watson Text-to-Speech von IBM

Das Projekt wurde durch den Creative Supervisor der Vorentwicklungsabteilung von Telemotive beauftragt und genehmigt sowie durch diesen mit Mitteln versorgt.

Die Anwendung wird von Messepersonal und Messebesuchern genutzt und muss daher intuitiv bedienbar sein.

Die spezifische Konfiguration soll nur durch das zuständige Implementierungsteam der MTM Vorentwicklung während der Umsetzung erfolgen. Bei der laufenden Anwendung sind keine Konfigurationsmöglichkeiten vorhanden noch gewünscht.

Das Projekt wird durch den Creative Supervisor der Vorentwicklung abgenommen.

1.5 Projektabgrenzung

Explizit ausgeschlossen vom Projektrahmen sind folgende Bereiche:

- Anpassung bzw. neu Entwicklung einer graphischen Nutzeroberfläche
- Graphische Modellierung der gerenderten Objekte
- 3D Druck des Simulationsmediums

2 Projektplanung

2.1 Abweichungen vom Projektantrag

Im Projektantrag wurde geplant, dass die Sprachumsetzung mit Google Cloud Text-to-Speech bzw. Speech-to-Text umgesetzt wird. Da aber in der Firma bereits Lizenzen für IBM Watson Speech-to-Text besitzt, wurde auf dieses Framework zurückgegriffen, da es einen ähnlichen Funktionsumfang aufweist.

Da sich der Testprozess als etwas umfänglicher als im Projektantrag erwiesen hatte, wurde etwas mehr Zeit für die Testphase verwendet wie ursprünglich geplant – jedoch war auch die Implementierungsphase weniger zeitintensiv wie erwartet, sodass sich diese etwa amortisierten.

2.2 Projektphasen

Der gesamte zeitliche Rahmen für die vollständige Umsetzung des Projektes war 70 Stunden. Bei Projektbeginn wurden diese auf die diversen Softwareentwicklungsphasen verteilt. In beiliegender Tabelle 1 ist die grobe Zeitplanung zu finden. Die präzisierte und konkretisierte Darstellung so wie die genauere Unterteilung der Projektphasen finden sich im Anhang A1 Detaillierte Zeitplanung.

Die in 2.1 Abweichungen vom Projektantrag beschriebenen zeitlichen Abweichungen sind durch -/+ Zeichen gekennzeichnet.

Projektphase	Zeitbedarf
Analysephase	8h
Entwurfsphase	8h
Implementierungsphase	38h - 2h
Test- und Abnahmephase	6h + 2h
Dokumentationsphase	10h
Gesamt	**70h**

Tabelle 1:Grobe Projektphasen

2.3 Ressourcenplanung

Die präzise Auflistung der genutzten Ressourcen ist im Anhang A2 Ressourcenauflistung zu finden. Diese beinhaltet die gesamte genutzte Software sowie die verwendete Hardware und die verwendete Arbeitszeit der Mitarbeiter. Alle für die Umsetzung genutzte Soft- und Hardware wurde bereits vor diesem Projekt finanziert oder ist als Open-Source Software kostenlos verwendbar, weshalb keine zusätzlichen Kosten für dieses Projekt entstanden sind.

2.4 Entwicklungsprozess

Für dieses Projekt wird mit einem agilen Entwicklungsprozess gearbeitet, welcher weitestgehend auf dem Scrum-Modell basiert. Dieser Entwicklungsprozess wurde auf Grund seines hohen Potentials für eine kontinuierliche Feedbackloop zwischen Entwickelndem und Stakeholder gewählt.

Scrum wird mit sogenannten Sprints iterativ durchgeführt – in diesem Projekt wurde der Zyklus von zwei wöchigen Sprints gewählt, welcher sich in der Praxis als vielversprechende

Iterationslänge bewiesen hat. Durch die Zeremonien wie dem Daily Stand Up und insbesondere dem Review wird viel Raum für Produkt Owner – Developer und Stackeholder-Developer Kommunikationen gegeben. Dies ermöglicht auch Anforderungsanpassungen in Rücksprache mit dem Stakeholder, was zur Konsequenz hat, das normal in der Entwurfsphase verwendete Zeit in die Implementierungsphase verlegt wurde um diesem Potential gerecht zu werden. Auf eine Retrospektive wurde auf Grund des kleinen Entwicklungsteams und dem daraus resultierenden niedrigeren Benefit einstimmig verzichtet und Verbesserungsvorschläge konstant im Daily Stand Up eingebracht.

Primärer Zweck des Daily Stand Up Meetings ist es, eine direkte Kommunikation zwischen Entwickler und Product Owner herzustellen, damit ein transparenter Gesamtüberblick auf das Projekt und des Fortschritts geworfen werden kann.

Das wöchentliche Review Meeting mit dem Stakeholder ermöglicht direktes Feedback zum aktuellen Entwicklungsstand und die flexible Anpassung der Anforderungen zum Projektl. Zusätzlich stellt es eine Art Vorabnahme da, welche den Arbeitsaufwand und die investierte Zeit in der finalen Abnahme verkürzt.

Daily Stand Up Meetings sollen als Austauschmöglichkeit bei möglichen Problemen genutzt werden und dazu dienen, einen Überblick über den Projektfortschritt zu erlangen.

Die typische Retrospektive als Scrumzeremonie, welche zur iterativen Verbesserung des Scrumteams von Sprint zu Sprint führen soll, ist auf Grund der geringen Teamgröße und der kurzen Projektdauer nicht gewinn – bzw. erkenntnisbringend, so wie dies bei einem klassischen Scrumprojekt der Fall wäre.

3 Analysephase

3.1 Ist-Analyse

Nach der Erstimplementierung und dem ersten Vorstellen auf Messen, wie in Punkt 1.1. beschrieben ist aufgefallen, dass die Interaktivität und Intuitivität der Simulation noch Verbesserungspotenzial aufweist. Des Weiteren war es kaum möglich den signifikanten Unterschied zwischen niedriger und hoher Fahrzeugvernetzung aufzuzeigen.

Die einzige vorhandene Interaktion war bis zu diesem Projekt, das Einstellen des Standes der Sonne, welches aber keinerlei Kompetenzen und Fähigkeiten von Automobilvernetzung und MTM aufzeigte.

Die gezeigte Simulation zeigte immer nur gleichfahrende Fahrzeuge, welche wenig mit einer realistischen Verkehrssituation geschweige denn mit einer realistischen Ausnahmesituation zu tun hatten.

3.2 Machbarkeitsstudie

3.2.1 Anforderungen Unfallszenario

Das Unfallszenario wird als wirtschaftlich realisierbar eingeschätzt. Da die Anbindung von der Unity Engine an Vuforia und die Konfiguration dafür schon von vorherigen Entwicklern gemacht wurde, entfällt dieser Arbeitsschritt.

Das Auslösen des Unfallszenarios soll über einen sog. Marker passieren oder durch ein von Watson Speech-to-Text unterstützten Sprachkommando erfolgen, wobei die Umsetzung so modular geschehen soll, dass Erweiterungen möglich sind.

Die Fahrzeuge sollen je nach Vernetzungsgrad unterschiedlich eine Rettungsgasse für ein explizites Rettungsfahrzeug bilden – dies wird über die Geschwindigkeit der Bildung der Rettungsgassen simuliert und ist daher mit einer Basisimplementierung des Szenarios gut parametrisierbar.

3.2.2 Anforderungen Sprachkommando für Szenenreset

Der Szenenrest ist ein Feature, bei welchem es von hoher Bedeutung ist, dass es absolut fehlerfrei funktioniert, da es sonst einen Reboot erfordern würde und damit einen Imageschaden während der Messepräsentation verursachen.

Trotz dieser Anforderungen ist dies realisierbar, da schon bei dem Unfallszenario der Watson Speech-to-Text von IBM verwendet wird, wodurch kein großer Mehraufwand besteht.

3.2.3 Anforderungen Testprozess

Der Testprozess stellt einen Mehraufwand da, welcher aber die Erhaltbarkeit und die Qualität des Projektes sichert.

Da der Testprozess nicht den Automobilprozesstandart ASPICE erfüllen muss, ist dieser für die Größe dieses Projektes realisierbar.

3.3 Wirtschaftlichkeitsanalyse

Da in den vorangegangen Punkten Mängel und nicht genutztes Potential der aktuellen Umsetzung der Verkehrssimulation festgestellt wurden und die Machbarkeitsstudie gezeigt hat, dass die Optimierung und Verbesserung realisierbar ist, soll gezeigt werden, dass die weitere Umsetzung auch wirtschaftlich ist.

3.3.1 Make or Buy-Entscheidung

Der primäre Nutzen der Simulation ist es bei Messen die Firmeneigene Kompetenz im Bereich XR zu zeigen sowie das Potential vernetzter Fahrzeuge ist es unabdingbar das Projekt selber umzusetzen.

3.3.2 Projektkosten

Im Folgenden sollen die anfallenden Projektkosten quantifiziert werden. Dabei werden die Faktoren Zeit, Personalkosten sowie Ressourcenkosten herangezogen. Die konkretisierten Personalkosten sind Firmeninterna und daher wird als Stundensatz für eine Auszubildende mit 8€ angesetzt und für einen festangestellten Project Engineer ein Stundensatz von 35€. In den Ressourcenkosten sind beispielsweise enthalten: Miete, Strom, Softwarekosten, Hardwarekosten sowie Lizenzkosten. Dafür wird eine Stundenpauschale von 30€ veranschlagt. Die eher hoch angesetzten Ressourcen kosten sind auf die recht hohen Lizenzkosten von der verwendeten Software Vuforia und Watson Speech-to- Text zurückzuführen.

In Tablle 2 wird die Kostenaufstellung verdeutlicht.

Vorgang	Zeitbedarf	Mitarbeiter	Personal	Ressourcen	Gesamt
Entwicklungskosten	67h	1x Azubi	536€	2010€	2546€
Abnahme	1h	1x Mitarbeiter 1x Azubi	43€	30€	73€
Agile-Meetings	2h	1x Azubi 1x Mitarbeiter	86€	60€	146€
				Gesamtkosten	**2765€**

Tabelle 2:Kostenaufstellung

3.3.3 Amortisationsdauer

Nach der Aufstellung der Projektkosten soll nun berechnet werden, wann sich die Entwicklungskosten amortisieren. Dieser Wert kann feststellen, ab welchen Zeitraum die Kosten der Entwicklung sich mit den Ersparnissen durch das Projekt ausgeglichen haben.

Für die Berechnung der Armortisationsdauer sind die Kosten und die Einsparung der Anwendung relevant, wobei die Kosten bereits in Kapitel 3.3.2. Projektkosten ermittelt wurden. Als Einsparung wird der Lizenzkostennachlass, welcher von der AR Library Vuforia für die Bereitstellung des Projektes zu Ausstellungszwecken. Diese Kosten werden nur teilweise als Ressourcenkosten berücksichtigt, da Vuforia von mehreren Projekten genutzt wird und dadurch keine direkten Kosten für dieses Projekt anfallen.

Der exakte Nachlass des Lizenzkostennachlasses fällt unter Geheimhaltung, sodass simplifiziert mit einem Nachlass von 80% gerechnet wird, welcher vom Stakeholder als realisitsch eingestuft wird, jedoch auch anmerkte, dass dieser Nachlass vertraglich auf 2 Jahre befristet ist.

Lizenzkosten: 40.000€ pro Jahr

Lizenzkostennachlass: 80% -> Ersparnis von 32.000€ pro Jahr

Dieses Abschlussprojekt stellt nur einen Teil des Gesamtprojektes dar, weshalb nicht der volle Lizenzkostennachlass als Ersparnis gewertet werden soll. Die erfolgte Erweiterung und Optimierung hat jedoch einen hohen Verbesserungsfaktor und wird daher mit 10% an den Ersparnissen gerechnet.

$$Ersparnisse = 32.000\,{}^{€}/_{Jahr} * 0.1 = 3200\,{}^{€}/_{Jahr}\,; Kosten = 2765€$$

$$Amortisationsdauer = \frac{Kosten}{Ersparnisse} = \frac{2765€}{3200\,{}^{€}/_{Jahr}} = 0{,}864\,Jahre \approx 45\,Wochen$$

Aus obenstehender Rechnung geht hervor, dass sich die Umsetzungskosten innerhalb einen Jahres (45 Wochen) rentieren. Da der vertraglich geregelte Lizenzkostennachlass für zwei Jahre läuft, ist die Amortisationsdauer erreicht.

3.4 Nicht monetärer Nutzen

Da das Projekt als Ausstellungsstück für eine Messepräsentation genutzt wird und daher nicht auf den Markt kommt, ist kein direkter monetärer Nutzen vorhanden.

Indirekt werden durch neue Aufträge Einnahmen generiert, welche aber nicht direkt auf dieses Projekt zurückzuführen sind.

Die Werbung neuer Kunden sowie der Erhalt von aktuellen Kunden, Anwerben von neuen Arbeitskräften, Aufmerksamkeit für die Firma sowie das Zeigen von mehreren Kompetenzen der Firma stehen im Vordergrund und bringen einen großen nicht monetären Nutzen welcher im gesunden Verhältnis zu den Ausgaben steht.

3.5 Anwendungsfälle

Die Simulation ist für Messepräsentationen konzipiert und soll daher für den Anwender eine einfache und intuitive Bedienung bieten.

Die geringe Menge an Interaktionsmöglichkeiten, damit der Nutzer nicht überfordert wird, sorgt folglich für ein überschaubares A3 Use-Case Diagramm.

3.6 Lastenheft

Das für die Umsetzung erforderliche Lastenheft wurde gemeinsam mit dem Stakeholder erstellt und ist mit allen erwarteten Anforderungen des Auftraggebers befüllt. Das Lastenheft ist auszugsweise im Anhang A4 zu finden.

4 Entwurfsphase

4.1 Zielplatform

Die primäre Zielplattform ist Andriod, obwohl als urspüngliche Zielplattform bei Projektstart das Apple iPad Pro war. Da bei iOS das deployen mit einem hohen Mehraufwand verbunden ist, das explizit auf einem Mac das Unitybuild gebaut werden muss und dabei auch noch xCode benötigt wird – diese zusätzlichen Arbeitsschritte entfallen bei einem Androidtarget, weshalb dieses vorher schon als Developmenttarget verwendet wurde. In Rücksprache mit dem Stakeholder wurde beschlossen, die Vorteile einer Androidzielplattform zu nutzen und dadurch ein iOS zu einer sekundären Zielplattform zu machen.

Das zu verwendende Framework, Unity, welches das am häufigsten verwendete Framework für AR Applikationen ist, wurde bereits vor Projektstart festgelegt. Unity eignet sich auf Grund seinen hohen Funktionsumfang und guter Kompatibilität mit anderen Programmes sehr gut für XR Anwendungen. In der Firma ist auch schon Wissen und Erfahrung mit Unity gesammelt worden sowie sind auch schon Lizenzen vorhanden, welche für dieses Projekt genutzt werden konnten.

Des Weiteren wird für die Umsetzung eines AR Projektes auch ein AR SDK benötigt. Da bei Projektstart Vuforia ausgewählt wurde, welches häufig verwendet wird, da es sich als sehr zuverlässig erwiesen hat und sich durch eine gute API sowie Anwendung auszeichnet.

Für das Erkennen der Sprachkommandos wird von IBM Watson Speech-to-Text verwendet, um dadurch die Keywords für die Sprachkommandos aufzuzeichnen und in einen „Text“ zu parsen, damit diese mit den ausgewählten Kommandos abgeglichen werden können.

Ursprünglich wurde das Google Speech-to-Text Framework vorgeschlagen, da aber die Firma bereits Watson Speech-to-Text Lizenzen besitzt wurde auf Grund des ähnlichens Funktionalitätsumfang auf dieses zurückgegriffen.

Als verwendete Programmiersprache wurde beim Start des ursprünglichen Projektes C# festgelegt, da diese sowohl von Vuforia als auch von Unity unterstützt wird. Aus diesem Grund wird die Auswahl hier nicht weiter ausgeführt.

4.2 Architekturdesign

Für die Umsetzung wird nach dem Model View Controller Muster implementiert. Der Model View Controller teilt Komponenten nach ihren Aufgabenbereichen auf. Die Modelkomponente ist für die Datenhaltung zuständig, der Controller hat die Aufgabe, dass Programm zu steuern und die Viewkomponente hat die Aufgabe, dass die visuelle Darstellung des Programmes erfolgt. Durch diese Aufteilung soll die Wartung, die Austauschbarkeit und das Testen der Einzelkomponenten gewährleistet sein.

Die Datenhaltung soll als eine C#-Klasse realisiert werden, welche von einer ebenfalls in C# geschriebenen Klasse verwaltet werden soll. Die View Komponente wird von Unity als Framework selber übernommen.

4.3 Entwurf der Benutzeroberfläche

Da in diesem Projekt keine Änderung der graphischen Nutzeroberfläche vorgesehen ist, ist hierfür auch kein Entwurf notwendig.

4.4 Geschäftslogik

4.4.1 Unfallszenario

Das Unfallszenario kann über zwei Wege ausgelöst werden – über ein Sprachkommando oder über einen AR Marker.

Bei dem Sprachkommando wird der Unfall an einem festgelegten Punkt ausgelöst.

Über ein Observerpattern, wo bei die Klasse ScenarioController KeywordObserver als Interface implementiert als Observer agiert und die Klasse SpeechInput welche das Subjekt darstellt. Es wird generalisiert der erkannte String reingepasst und alle Observer sind in der Lage über String.contains() die jeweiligen relevanten Keywords filtern.

Wenn das Keyword für das Unfallszenario erkannt wird, startet dies an einer festgelegten Position als Paramter, während wenn der AR Marker erkannt wird das Szenario an seiner nächsten validen Position auf einer Straße ausgelöst wird.

Die Fahrzeuge werden über ein Factorypattern über den CarSpawnController aus der Klasse VehicleFactory generiert, wobei auch Rettungsfahrzeuge und CrashDummy – Autos generiert werden können.

Die Informationen der Fahrzeuge sind in einer eigenen Datenhaltungsklassen hinterlegt.

Die Fahrzeuge werden durch den CarMovementController pro Frame bewegt und despawnt sowie durch den CarSpawnController gespawnt.

Bei dem Unfallszenario werden an Stelle des Markers (oder an der fixen Position) an der betroffenen Spur ein unsichtbares Fahrzeugobjekt generiert, welches für ein Verlangsamen bzw. einen Stillstand des Verkehrs auf dieser Spur sorgen. Das Rettungsfahrzeug, welches von hinten auf diese Spur fährt, hat eine Triggerbox, welche die restlichen Fahrzeuge dazu veranlasst, in einem gewissen Umkreis eine Rettungsgasse zu bilden. Die Größe dieser Triggerbox ist abhängig von der Vernetzung der Fahrzeuge, wodurch Fahrzeuge früher eine Rettungsgasse bilden, auch wenn das Rettungsfahrzeig noch weiter weg ist.

Im Anhang ist ein Klassendiagramm (A5) zu finden, dass die Klassen und deren Beziehungen für das Unfallszenario untereinander darstellt.

4.4.2 Szenenreset über Sprache

Das Neuladen der Unityszene und damit das Zurücksetzen aller Werte soll über ein Sprachkommando erfolgen.

Über das Observerpattern hört der GameController welcher KeywordObserver als Interface implementiert als Observer auf SpeechInput, welcher dann den Text auf das Keyword durchsucht. Durch die abstrakte Klasse KeywordObsever ist es möglich, das Pattern zu erweitern, sodass mehr Klassen SpeechInput subscriben oder Sprachkommandos in anderen Sprachen hinzugefügt werden können.

.

Bei dem Szenenreset wird das Tracking des Vuforia Markers unterbrochen und die Szene wird komplett neugeladen.

Der konkrete Reset geschieht dann im GameController, wo die aktuelle Szene (in diesem Fall Szene 0) neu geladen wird.

Im Anhang ist ein Klassendiagramm (A6) zu finden, dass die Klassen und deren Beziehungen für den Szenenreset und der Sprachsteuerung untereinander darstellt.

4.5 Maßnahmen zur Qualitätssicherung und Testing

Die Qualitätssicherung wird durch Tests gewährleistet, welche im Rahmen von diesem Projekt zu einem Testprozess erweitert werden sollen, aus welchem dann auch ein TSR hervorgehen soll. In diesem Testprozess sind Unittests, Integrationtests und manuelle Test enthalten um einen dreistufigen Testprozess zu erstellen.

4.6 Pflichtenheft

Das Pflichtenheft soll über individuelle Jiratickets umgesetzt werden, damit ein agiler Projektansatz gut realisiert werden kann.

5 Implementierungsphase

5.1 Implementierung der Spracherkennung

5.1.1 Implementierung der Datenstrukturen

Aufgrund des Umfangs des Projektes wurde für die Spracherkennung keine eigene Implementierung vorgenommen, sondern auf eine bereits in der Firma lizensierte Software IBM Watson Speech-To-text zurückgegriffen.

Da Watson Speech-to-Text über einen Cloudservice und durch eine Machine-Learning-AI aus gesprochener Sprache direkt Strings generiert ist die Datenhaltung für die Konvertierung nicht Teil dieses Projekts.

Die zu erkennenden Keywords sind in den jeweiligen KeywordObserver Klassen als Vergleichsstrings hinterlegt, dadurch könnten auch Keywords aus anderen Sprachen festgelegt und erkannt werden.

Es ist dabei möglich, dass mehrere Observer den SpeechInput subscriben bzw. unsubscriben können.

5.1.2 Implementierung der Geschäftslogik

Es mussten die Klasse GameController sowie das Interface KeywordObserver und die Klasse SpeechInput neu implementiert werden.

Der KeywordObserver dient als Interface dazu die Grundfunktionalität eines erkannten Sprachinputs an andere Klassen weiterzugeben, damit diese in der Lage sind, auf den SpeechInput Stream zu hören.

Watson Speech-to-Text erkennt gesprochene Sätze und generiert aus diesen einen InputString, welcher als ein String über einen RegEx oder über eine direkte Funktion im KeywordObserver durchsucht werden kann.

5.2 Implementierung des Unfallszenarios

5.2.1 Implementierung der Datenstrukturen

Zuerst wurden Datenstrukturen für das Unfallszenario implementiert. Dazu wurden die im Klassendiagramm dargestellten Klassen in der Programmiersprache C# umgesetzt und an die GameObjects in Unity angehängt, welche diese benötigen.

Der WorldMarker ist in der Unityszene als zweidimensionales Objekt zu finden und wird ab OnAwake() gesucht. Sobald der Marker von der Kamera der Hardware auf welcher die Software ausgeführt erkannt wird, was zur Voraussetzung hat, dass die Kamera auf den Marker gerichtet ist sowie passende Lichtverhältnisse vorhanden sind, beginnt das Rendern der Szene.

Alle Objekte in der Unityszene werden in Relation zu dem WorldMarker dargestellt was auch heißt, dass falls der Marker bewegt wird, sich alle Objekte in dieser Szene mit diesem bewegen. Voraussetzung dafür ist, dass alle Objekte das AR Marker Objekt als Parent haben, damit diese auch sich relativ dazu bewegen.

Durch den Unfall Vuforia Marker wird in Unity ein Objekt generiert und an der nächsten validen Stelle auf einer der Lanes (einem der nächsten ITween Pfade) platziert. Dieses Objekt stellt den Anker für ein CrashDummyVehicle da, welches von der Klasse VehicleObject und als Prefab so hinterlegt ist, dass es nicht sichtbar ist. Dieses CrashDummyVehicle sorgt dafür, dass die anderen simulierten Fahrzeuge ihre Geschwindigkeit verringern bzw. an der Stelle des Unfalls zum Stillstand kommen. Auf das CrashDummyVehicle wird noch eine Triggerbox gesetzt, welche auch Spuren, welche nicht direkt von dem Unfall betroffen sind verlangsamt im Bereich der Triggerbox. Diese Triggerbox hat nicht das EmergencyVehicleObject als Mittelpunkt sondern ist mit einem leichten Offset Richtung Fahrtrichtung versehen.

Das EmergencyVehicle Prefab ist mit einer Triggerbox versehen, welche alle GameObjects von der Klasse VehicleObject dazu bringt, mit einem Offset die Spur zu verlassen und dadurch eine Art Rettungsgasse zu bilden. Wie schnell die VehicleObjects diesen Offset umsetzen als auch die Größe dieser Triggerbox ist abhängig von dem Vernetzungswert, welcher im GameController hinterlegt ist.

5.2.2 Implementierung der Geschäftslogik

Nach der Umsetzung der Datenstrukturen wurde als nächstes der SzenarioController implementiert. Dieser wurde wie der GameController als auch der EmergencyVehicleController gemäß dem Singleton Pattern implementiert, um zu garantieren dass von jeder Controllerklasse nur eine Instanz existiert, damit unschöne Seiteneffekte vermieden werden.

Der SzenarioController hat die Aufgabe die Klasse CarSpawnController zu triggern und ihr die Position der Unfallstelle mitzugeben, damit diese die CarFactory korrekt triggern kann, sodass das CrashDummyVehicle an der Unfallstelle existierts und das EmergencyVehicle diese Position als Zielposition erhält. VehicleObjects werden vom CarMovementController auf den Spuren bewegt, während EmergencyVehicle GameObjects von dem EmergencyCarController gesteuert wird, wobei dieser von dem CarMovementController erbt.

Der SzenarioController ist auch für das Platzieren der Unfallobjekte zuständig – diese Platzierung kann auf zwei Arten bestimmt werden:

- Über die Position des UnfallMarkers
- Durch eine fixe Position

Der zweite Fall tritt dann ein, falls das Unfallszenario nicht über den UnfallMarker ausgelöst wird, sondern über das Sprachkommando, wodurch es nicht möglich ist sich eine valide Position durch den UnfallMarker zu berechnen, sodass diese Position von dem ScenarioController vorgegeben wird als Defaultposition.

Abgesehen von den Koordinaten der Unfallobjekte verhalten sich die Unfallszenarios gleich unabhängig von dem Trigger.

Sobald ein Trigger registriert wird, wird vom SzenarioController die Methode StartCrashScenario(Vektor3) aufgerufen welche im CarSpawnController die Methode spawnScenarioVehicles(LanePosition) triggert, wodurch in der Klasse VehicleFactory

zusätzlich zu den VehicleObjects auch das CrashDummyVehicle und das EmergencyVehicle aus den Prefabs instanziiert werden.

Das CrashDummyVehicle bekommt die Geschwindigkeit 0, wodurch alle folgenden VehicleObjects auf dieser Spur zum Stehen kommen.

Das Crashobjekt blockiert eine Lane, dadurch, dass es als Objekt der Klasse VehicleObject keine Geschwindigkeit und statisch an einer Position bleibt, bis ein Object der Klasse EmergencyVehicleObject die gleichen Koordinaten erreicht.

Das EmergencyVehicleObject wird ebenfalls wie das CrashObject über die CarFactory durch die Methode spawnEmergencyCar() generiert und wird ebenfalls durch den CarMovementController bewegt, hat jedoch andere Fahrtregeln wie die anderen VehicleObjects – so ist diese nicht von der Verlangsamung durch die CrashTriggerBox betroffen.

Das Prefab für das Rettungsfahrzeug weißt eine Triggerbox auf, welche etwa im Mittelpunkt des EmergencyVehicles ist – sobald sich ein anders VehicleObject sich im Bereich des Triggers befindet und dadurch die Methode OnTriggerEnter() aufruft wird die Geschwindigkeit des VehicleObjects reduziert und die Position ändert sich gelerpt zu einem Offset, damit die Spur für das EmergencyVehicle frei ist, wodurch die Fahrzeuge vor dem Rettungsfahrzeug die Rettungsgasse bilden. Sobald die Methode OnTriggerExit() für ein VehicleObject aufgerufen wird, kann dieses gelerpt an seine ursprüngliche Position auf der Spur zurückkehren

Bei OnTriggerEnter() wird im CarMovementController lerpOffset() aufgerufen wodurch über die Zeit die Position verschoben wird, während bei OnTriggerExit() lerpOffsetBack() gecalled wird, um die Rettungsgasse wieder zu schließen.

In Abhängigkeit von der Gesamtvernetzung der Fahrzeuge wird diese Triggerbox in ihrem Radius vergrößert sowie die Zeit, die lerpOffset() benötigt verkürzt, damit Fahrzeuge als früher über dieses Rettungsfahrzeug informiert gelten und damit schneller ihr Fahrverhalten anpassen.

Nachdem das EmergencyVehicle-Objekt die Koordinaten des Crashobjectes erreicht hat wird dieses aus der Szene entfernt und das EmergencyVehicle verlässt die Szene, in dem es der Lane bis ans Ende folgt.

Der Code für den ScenarioController (A7) als auch auszugsweise für den CarMovementController (A8) sind im Anhang zu finden.

5.3 Implementierung des Szenenresets

5.3.1 Implementierung der Datenstrukturen

Für den Szenenreset wird im GameController eine Funktion angelegt, welche die aktuell dargestellte Szene neu lädt – durch die Abstraktion der Klasse wird es ermöglicht das Resetsprachkommando auch zu nutzten, falls neue Szenen wie z.B. ein anderes Verkehrsnetz hinzugefügt werden.

5.2.2 Implementierung der Geschäftslogik

GameController erbt von der Klasse KeywordObserver und stellt damit selber einen Observer da, welcher InputSpeech als Subjekt subscribed um daraus das Keyword als Trigger zu identifizieren.

Das Keyword (in diesem Fall „reset“) löst dann die Methode resetScene() aus, welche dann die aktuell geladene Szene neu lädt inklusive aller Komponenten. Bei dem Neuladen geht auch erst einmal kurz das Tracking des Markers bzw. der Marker verloren, welches dann OnAwake() wieder neu gesucht wird.

Der Code für den GameController (A9) ist ebenfalls im Anhang zu finden.

6 Test- und Abnahmephase

6.1 Testprozess

Zur Qualitätssicherung und Absicherung wurde für dieses Projekt ein Testprozess eingeführt.

Dieser kann in drei Teststufen unterteilt werden

- Unittest
- Intergationtest
- Manueller Test

Der TSR bezieht sich ausschließlich auf den Integrationtest und die manuellen Tests welche als Regressionstests durchgeführt werden.

Ein TSR ermöglicht es einen hohen Standart bei Absicherungen und Tests zu erhalten sowie die bereits vorhandene Funktionalität zu erhalten. Bei einem Regrssionstests für den manuellen Test werden zuverlässig reproduzierbare Use Cases verwendet, welche den realistischen Abläufen entsprechen.

Es wurden während der Entwicklung kontinuierlich durch Deployment der zu testenden Software Integrationtests sowohl auf Android als auch auf iOs durchgeführt.

Der Integrationtest hat unterschiedliche Anforderungen abhängig von der Plattform – bei Android wurde nur die gebuildete APK Datei auf das Target deployed, während für ein iOS Gerät über xCode kompiliert werden muss und dann erst das Programm übertragen werden kann.

Diese wurde dokumentiert und auftretende Fehler sofort als Bugtickets festgehalten.

Bei den manuellen Tests gibt es eine Reihe an üblichen Usecases, welche von dem Tester ausgeführt und für rot (absolut fehlerhaft), gelb (z.B. Performanceprobleme) oder grün (alles lief reibungslos) im TSR markiert werden sollen.

Das Template des TSRs ist im Anhang A11 zu finden.

Die Integrationtests und manuellen Tests liefen zu Ende des Projektes reibungslos ab.

6.2 Abnahme

Da das Projekt in einem agilen Entwicklungsprozess durchgeführt wurde, war der Stakeholder über das Sprintreview immer auf dem aktuellen Stand der Entwicklung, welche es ganz nach dem agilen Ansatz ermöglichte, dass auf neue Anforderungen und Feedback sofort reagiert werden konnte.

Die finale Abnahme erfolgte Remote mit dem Stakeholder, wobei dieser die Software auf sein Androidsmartphone deployed wurde, was nicht den exakten Nutzbedingungen entspricht.

In der reduzierten Abnahme zeigte sich der Stakeholder zufrieden und es wurde eine Abnahme mit Normbedingungen geplant, welche aber nicht mehr in den Projektzeitraum fällt.

7 Dokumentation

Das Projekt wurde in einer Projektdokumentation, einer Entwicklungsdokumentation und einer Anwenderdokumentation festgehalten.

In der Projektdokumentation wurden die im Rahmen des Projektes durchlaufenen Phasen veschrieben.

Die Entwicklungsdokumentation erfolgte über Kommentare im Code und den vom Unityframework bereitgestellten Tooltips (siehe Codeausschnitte sowie A10).

Des Weiteren wurde eine Anwenderdokumentation verfasst, damit am Ende der Primärnutzer (Messestandbeauftragte) in der Lage ist, die neuen Features des Projektes zu kennen und nutzen zu können.

8 Fazit

8.1 Soll-/Ist-Vergleich

Im Gegensatz zum Projektantrag wurde die Sprachsteuerung nicht mit Google speech-to-text sondern mit Watson speech-to-text umgesetzt, was jedoch nichts an der Funktionalität beeinflusst hat.

Da der Aufwand für den Testprozess etwas höher ausfiel als erwartet, wurde dort mehr Zeit verwendet wie ursprünglich geplant – jedoch konnte auch die Softwareimplementierung etwas schneller durchgeführt werden.

Gesamthaft konnte das Projekt in dem geplanten Gesamtumfang durchgeführt werden.

Das Projekt konnte erfolgreich in einem 70h Zeitrahmen durchgeführt werden, was dem von der IHK vorgegebenen Zeitrahmen entspricht.

Projektphase	Soll	Ist	Differenz
Analysephase	8h	8h	0h
Entwurfsphase	8h	10h	0h
Implementierungsphase	38h	36h	-2h
Test- und Abnahmephase	6h	8h	+2h
Dokumentationsphase	10h	10h	0h
Gesamt	**70h**	**70h**	**0h**

Tabelle 3:Soll-Ist-Vergleich

8.2 Lessons Learned

Es wurden von der Auszubildenden eine Vielzahl an wichtigen Erfahrungen gesammelt.

Insbesondere wurde die Bedeutung einer genauen Voranalyse und einer realistisch durchgeführten Machbarkeitsstudie vermittelt.

Auch das Arbeiten mit mehreren third Party Softwarekomponenten und den daraus resultierenden Schwierigkeiten hat zu mehr Erfahrung der Auszubildenden geführt.

Gesamthaft kann das Projekt als ein großer Gewinn an Wissen für die Auszubildende beschrieben werden sowie auch für MTM ein Gewinn an neuer, verbesserter Software.

8.3 Ausblick

Es kam während der Entwicklung die Idee auf, auch weitere Szenarien zu entwickeln welche in die Simulation integriert werden können, wie z.B. eine Straßenblockade oder eine Polizeikontrolle.

Auch kann das Potential der Sprachsteuerung weiter genutzt werden um die AR Simulation noch interaktiver zu gestalten, wie z.B. die Geschwindigkeit zu erhöhen oder verschiedene Wetterlagen einzustellen.

Abschließend lässt sich sagen, dass die AR Simulation noch viel Potential bietet und es lukrativ ist, diese zu Erweitern.

Eidesstattliche Erklärung

Ich, Felicitas Hank, versichere hiermit, dass ich meine Dokumentation zur betrieblichen Projektarbeit mit dem Thema

Vernetzte Fahrzeuge in einer AR Simulation – Unfallsituation in einer Verkehrssimulation

Selbständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe, wobei ich alle wörtlichen und sinngemäßen Zitate als solche gekennzeichnet habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Unterschleißheim, den 26.05.2020

Felicitas Hank

Anhang

A1 Detaillierte Zeitplanung

Projektphase		
1. Analysephase	8h	
1.1 Ist Analyse		1h
1.2 Machbarkeitsstudie		4h
1.3 Wirtschaftlichkeitsanalyse		1h
1.4 Erstellen eines Use-Case Diagramms		0,5h
1.5 Erstellen eines Lastenhefts		1,5h
2. Entwurfsphase	8h	
2.1 Formulierung des Pflichtenhefts		1h
2.2 Entwurf des Unfallszenarios		4h
2.3 Entwurf des Sprachszenenresets		1,5h
2.3 Entwurf der Teststrategie		1,5h
3. Implementierungsphase	38h - 2h	
3.1 Implementierung des Szenenresets und Sprachkommando		6h
3.2 Implementierung der Datenstruktur		5h
3.3 Implementierung der Geschäftslogik		15h
3.4 Implementierung der Unittests		10h
4. Test- und Abnahmephase	6h + 2h	
4.1 Deployment auf Zielgerät		2h
4.2 Manuelle Tests und TSR		5h
4.3 Abnahme durch Auftraggeber		1h
5. Dokumentationsphase	10h	
5.1 Erstellung der Entwicklungsdokumentation		4h
5.2 Erstellung der Projektdokumentation		6h
Gesamt		**70h**

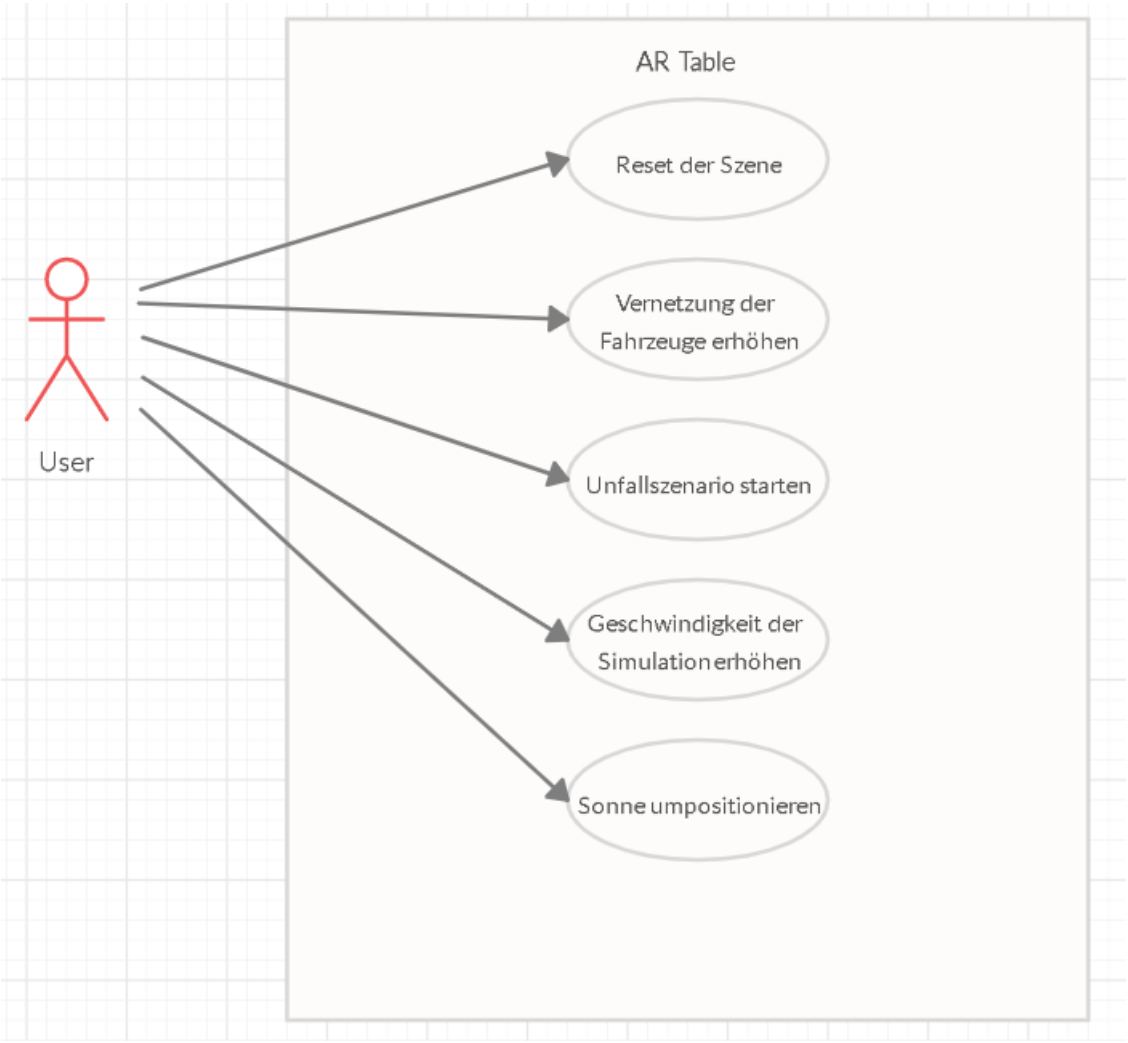
A2 Ressourcenauflistung

Hardware:
- HP Z-Book 15 (Entwicklungsrechner)
- IPad Pro (Zielgerät iOS)
- MacbookPro (ausschließlich für iOS benötigt)
- Samsung Tab A (Zielgerät Android)

Software:
- Unity Pro LTS 2018 (Entwicklungsumgebung)
- Vuforia Library (AR Library für Unity)
- XCode (benötigt für iOs)
- Android Building Toolkit für Unity (benötigt für Android)
- Watson speech-to-text von IBM (Keyworderkennung und für Sprachkommandos)
- Confluence (Dokumentation)
- Jira (Projektplanung)

Sonstiges:
- AR Marker für den Szenenanker
- AR Marker für das Unfallszenario
- Landschaftsmodell mit Straßen

A3 Use-Case Diagramm



A4 Lastenheft

Anforderungen:

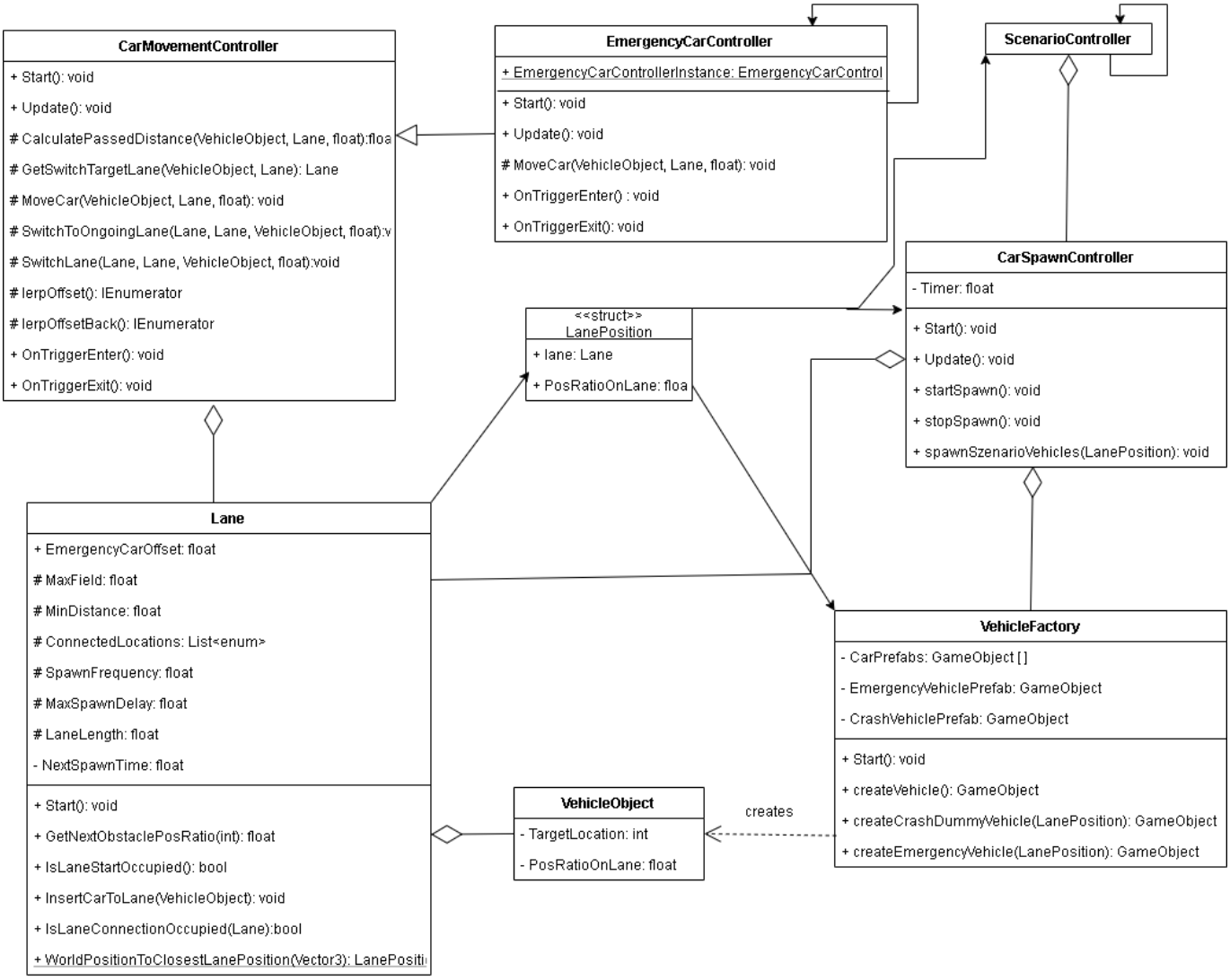
1. **Anforderungen an die Sprachsteuerung**
   - 1.1. Es soll Watson Speech-to-Text verwendet werden, da für diese Lizenzen bereits vorhanden sind
   - 1.2. Die Sprachkommandos sollen um neue Keywords erweiterbar sein
   - 1.3. Als Sprache soll zunächst Englisch verwendet werden
   - 1.4. Die Sprachsteuerung soll um andere Sprachen erweiterbar sein

[…]

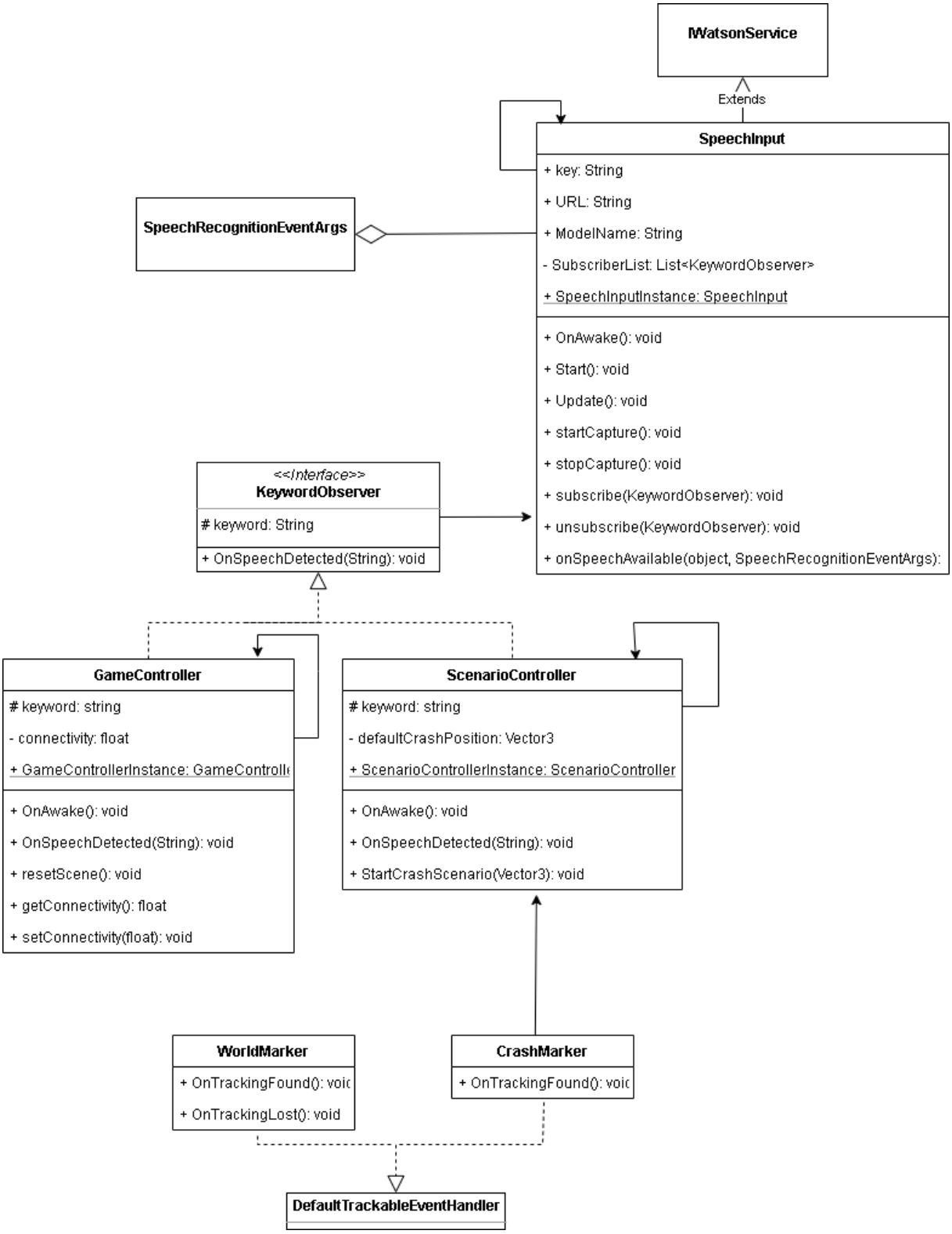
2. **Anforderungen an das Unfallszenario**
   - 2.1. Das Unfallszenario soll über ein AR Target auslösbar sein
   - 2.2. Das Unfallszenario soll über ein Sprachkommando auslösbar sein
   - 2.3. An der Position des AR Targets soll auf der nächsten Straße der Unfall ausgelöst werden
   - 2.4. Wenn das Unfallszenario über das Sprachkommando ausgelöst wird, soll an einer festen Stelle auf dem Straßenmodell der Unfall ausgelöst werden.
   - 2.5. Es soll ein Rettungsfahrzeug zu der Unfallstelle fahren
   - 2.6. Die anderen Fahrzeuge auf der Unfallspur sollen eine Rettungsgasse bilden
   - 2.7. Die Rettungsgasse soll nach Vernetzungsgrad der Fahrzeuge entsprechend schnell entstehen

[…]

A5 Klassendiagramm Unfallszenario



A6 Klassendiagramm Spracherkennung und Szenenreset



A7 ScenarioController

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

public class ScenarioController : MonoBehaviour, KeywordObserver
{
    [Tooltip ("The string for matching the trigger keyword")]
    protected string keyword = "crash";

    [Tooltip ("Instance of the Singleton ScenarioController")]
    public static ScenarioController ScenarioControllerInstance = null;

    private Vector3 defaultCrashPosition;


    /// <summary>
    /// Instanciating the ScenarioController
    /// </summary>
    public void OnAwake()
    {
        if (ScenarioControllerInstance != null)
        {
            Debug.Log("Multiple instances of ScenarioController. " +
            "Destroying duplicated ScenarioController: " + gameObject);
            Destroy(gameObject);
        }
        else
        {
            ScenarioControllerInstance = this;
        }
    }

    public void Start()
    {
        SpeechInput.SpeechInputInstance.subscribe(this);
        defaultCrashPosition = new Vector3(0, 0, 0);
    }

    /// <summary>
    /// Calls StartCrashScenario(), if the keyword is detected
    /// </summary>
    /// <param name="speechLog"></param>
    public void OnSpeechDetected(string speechLog)
    {
        speechLog = speechLog.ToLower();

        if (speechLog.Contains(keyword))

        {
            StartCrashScenario(defaultCrashPosition);
        }
    }

    /// <summary>
    /// Sets trigger for spawning of the crash vehicle and the emergency vehicle
    /// </summary>
    /// <param name="crashWorldPos"></param>
    public void StartCrashScenario (vector3 crashWorldPos)
    {
```

```
CarSpawnController.spawnScenarioVehicles(Lane.WorldPositionToClosestLanePosition(crash
WorldPos));
    }

}
```

A8 CarMovementController Ausschnitt

```
using System.Collections;
using UnityEngine;

public class CarMovementController : MonoBehaviour
{
    private float offset;    //offset that will be used in the movement function to
drive more left (negative) or right (positive) of the ITween path

    [Tooltip("the lane that this car drives on, assigned when car is spawned")]
    public Lane myLane;

    [Tooltip("the tag of the emergency lane trigger collider")]
    public string sirenCollider = "sirenCollider";



    […]

    /// <summary>
    /// On entering of the siren area of effect
    /// </summary>
    /// <param name="other"></param>
    private void OnTriggerEnter(Collider other)
    {
        Debug.Log("hearing siren and creating emergency lane");
        if (string.Equals(other.tag, sirenCollider)){
            StartCoroutine(lerpOffset());
        }
    }

    /// <summary>
    /// On exiting of the siren area of effect
    /// </summary>
    /// <param name="other"></param>
    private void OnTriggerExit(Collider other)
    {
        if (string.Equals(other.tag, sirenCollider))
        {
            StartCoroutine(lerpOffsetBack());
        }
    }

    /// <summary>
    /// calculating the offset to clear the path for the emergencyvehicle
    /// </summary>
    /// <returns></returns>
    public IEnumerator lerpOffset()
    {
```

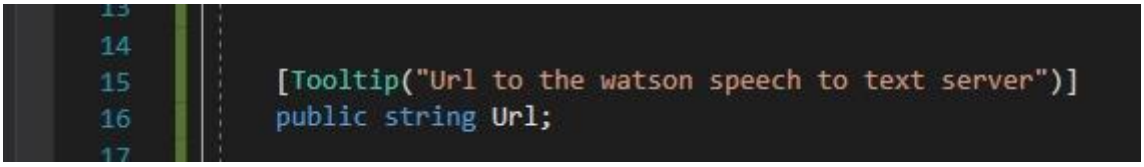
```
        float connectivity = GameController.GameControllerInstance.getConnectivity();
        float elapsedTime = 0f;

            while (elapsedTime < connectivity)
            {
            offset =  Mathf.Lerp(0f, -myLane.emergencyCarOffset, (elapsedTime /
connectivity));
            elapsedTime += Time.deltaTime;
            yield return null;
            }
        yield return null;
    }

    /// <summary>
    /// returning to default path pos
    /// </summary>
    /// <returns></returns>
    public IEnumerator lerpOffsetBack()
    {

        float connectivity = GameController.GameControllerInstance.getConnectivity();
        float elapsedTime = 0f;
        while (elapsedTime < connectivity)
        {
            offset = Mathf.Lerp(offset, 0f, (elapsedTime / connectivity));
            elapsedTime += Time.deltaTime;
            yield return null;
        }

        yield return null;
        offset = 0f;
    }
}
```

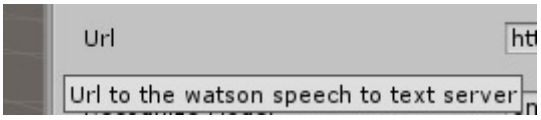
A9 GameController

```
using UnityEngine;
using UnityEngine.SceneManagement;
using System.Collections;
using System.Collections.Generic;

public class GameController : MonoBehaviour, KeywordObserver
{
    [Tooltip("The string for matching the trigger keyword")]
    protected string keyword = "reset";

    [Tooltip("Instance of the Singleton GameController")]
    public static GameController GameControllerInstance = null;

    private float connectivity = 0.5f;


    /// <summary>
    /// Instanciating the GameController
    /// </summary>
    public void OnAwake()
    {
        if (GameControllerInstance != null)
        {
            Debug.Log("Multiple instances of GameController. " +
            "Destroying duplicated GameController: " + gameObject);
            Destroy(gameObject);
        }
        else
        {
            GameControllerInstance = this;
        }
    }

    public void Start()
    {
        SpeechInput.SpeechInputInstance.subscribe(this);
    }

    /// <summary>
    /// Triggers the scene reset on keyword recognized
    /// </summary>
    /// <param name="speechLog"></param>
    public void OnSpeechDetected(string speechLog)
    {
       speechLog = speechLog.ToLower();

        if (speechLog.Contains(keyword))
        {
            resetScene();
        }
    }

    /// <summary>
    /// Reloads the scene to reset current scene
    /// </summary>
    public void resetScene()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
    }
```

```
    /// <summary>
    ///  Setter for the connectivity
    /// </summary>
    /// <param name="newConnectivity"></param>
    public void setConnectivity(float newConnectivity)
    {
        connectivity = newConnectivity;
    }

    /// <summary>
    /// Getter for the connectivity
    /// </summary>
    /// <returns></returns>
    public float getConnectivity()
    {
        return connectivity;
    }

}
```

A10Tooltips in Unity





A11 Auszug vereinfachter TSR

Test Summary Report Car2X

			Ergebnis	Anmerkungen
Integrationtest		**Beschreibung**	**Erwartetes Ergebnis**		
	iOS	Über xCode wird auf einem Mac gebaut und dann auf das iPad deployed	Anwendung startet korrekt auf Target auf	OK	
	Android	Build wird in Unity für Android gebaut und auf dem Tablet deployed	Anwendung startet korrekt auf Target auf	OK	
Regressionstests					
	Sonne verstellbar	Die Sonne wird über Touchinput mehrfach verstellt	Die Anwendung reagiert auf den Touchinput und die Sonne wird ohne Performanceeinbußen an der neuen Stelle angezeigt	OK	Bei zu schnellem Verstellen der Sonne sinkt die Framerate ab - dies wurde nur bei dem Androidgerät beobachtet
	Traking vom Worldmarker geht verloren	Nach erfolgreichen World Marker Tracking wird beabsichtigt die Kamera vom Marker abgewendet	Die Szene wird korrekt angezeigt in Relation zum Marker und bei Markerverlußt ist die Szene nach 5 sec nicht mehr sichtbar	OK	
	Szenenreset über Sprache	Die Szene wird durch das Keyword "Reset" auch in verschiedenen Geräuschkullisen zurückgesetzt	Die Szene lädt sich neu, auch unter schlechten Geräuschbedingungen	OK	