DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg
Campus Friedrichshafen

Unreal Cup

STUDIENARBEIT

im Studiengang Informationstechnik

an der DHBW Ravensburg
Campus Friedrichshafen

von

Michael Kekeisen,
Michael Möbius,
Daniel Rapp,
Maximilian Schmitz

10.07.2015

Bearbeitungszeitraum	13.10.2014 – 10.07.2015
Matrikelnummer	4468889, 2459628, 6014276, 9322432
Betreuer	Prof. Erwin Fahr

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg
Campus Friedrichshafen

Eidesstattliche Erklärung

gemäß § 5 (3) der „Studien- und Prüfungsordnung DHBW Technik“ vom 22. September 2011.

Hiermit erkläre ich, dass ich die vorliegende Arbeit mit dem Titel

Unreal Cup

selbständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt, keine anderen als die angegebenen Hilfsmittel benutzt und wörtliche sowie sinngemäße Zitate als solche gekennzeichnet habe.

Immenstaad, den 10.07.2015

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg
Campus Friedrichshafen

Kurzfassung

Moderne Computerspiele bieten unterschiedlichsten Zielgruppen eine abwechslungsreiche Freizeitbeschäftigung. Unter der Vielzahl dieser Spiele existieren auch solche, die unter Verwendung von strategischen Elementen den Spieler herausfordern, mit verschiedenen Herangehensweisen ihren Erfolg zu maximieren.
UnrealCup bietet mit der Möglichkeit, eigene künstliche Intelligenzen für eine Fußballsimulation zu erstellen, eine Vertiefung dieser strategischen Elemente. Das Ziel für die Anwender ist es, ihr so zusammengestelltes Team in einem kompetitiven Wettstreit gegen Teams anderer Nutzer zum Sieg zu führen.
Neben der spielerischen Funktion hat UnrealCup außerdem den Effekt, dass die Anwender ihre Fähigkeiten in der Programmierung und KI-Entwicklung erweitern können.

Der UnrealCup-Simulator bietet für dieses Ziel neben moderner 3D-Grafik und einer komfortablen Schnittstelle zum Einbinden der KI alle wichtigen Funktionen und Regeln, um dem Nutzer eine realistische Simulation und ein packendes Erlebnis zu bieten. UnrealCup beinhaltet einen Entwurf für einen graphischen Editor, um auch Nutzern, die wenig Erfahrung mit Programmierung besitzen, die Möglichkeit zu bieten, ein Team nach ihren Wünschen zu erstellen.

Neben der Bereitstellung aller notwendigen Spielerfunktionen zum Erstellen einer leistungsfähigen KI wird auch dafür gesorgt, dass alle für Simulationszwecke sinnvollen Fußballregeln eingehalten werden.
Der UnrealCup-Simulator besitzt also Ähnlichkeiten zur Simulation League des bekannten RoboCups, hebt sich allerdings durch diverse Alleinstellungsmerkmale wie eine moderne Grafik und realitätsnahe Physik von vergleichbaren Projekten ab.

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg
Campus Friedrichshafen

Abstract

Modern video games offer a varying leisure-time activity for diverse target groups. Among the variety of video games there are games that challenge the player to maximize his success by offering the possibility to use strategic elements.
UnrealCup deepens those strategic parts of the game by allowing the user to create own artificial intelligence for his teams players. The aim for the user is to lead their assembled team to victory in competitive contests against other players. Aside of this playful function, UnrealCup allows its users to expand their skills in programming and AI-development.

The UnrealCup simulator offers, aside of modern 3D-graphics and a comfortable interface to integrate the AI, important functions and football rules to give a realistic simulation and a thrilling experience. UnrealCup also contains a prototype of a graphical WYSIWYG-Editor to enable users without lots of experience in programming to create their own team.

Besides offering all player functions needed to create an efficient AI the UnrealCup simulator also makes sure that all soccer rules which are reasonable in a simulation are considered. The Simulator has similarities with the simulation league of the known RoboCup, yet it stands out from similar projects with its modern graphics and realistic physics.

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg
Campus Friedrichshafen

Inhalt

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg
Campus Friedrichshafen

1 Einleitung

Simulationen wie die RoboCup Simulation League bieten Interessierten eine Fläche zur Entwicklung und Forschung an Künstlicher Intelligenz. Nur was hinter der Simulation an Aufwand für den reibungslosen Betrieb getätigt wird oder was an Technologien verwendet wird ist weitestgehend unbekannt.
Wir entschieden uns daher eine eigene Fußballsimulation auf die Beine zu stellen, setzen dabei aber auf aktuelle Technologien. Die Simulation setzt zum Beispiel auf moderne 3D Grafik, aber auch die Physik wird in drei Dimensionen simuliert. Damit beschränken sich Spielzüge nicht nur auf zwei Dimensionen wie es aktuell in einigen Simulationen der Fall ist. Das erlaubt eine wesentlich flexiblere Ausarbeitung von Strategien.

Das Thema der Echtzeit wird eine sehr große Rolle in der Entwicklung des Projektes spielen: Alle Spieler müssen ihre Skripte pseudo parallel ausführen können und dürfen sich gegenseitig nicht beeinflussen oder gar blockieren. Dafür muss auf erlerntes der Vorlesungen Echtzeit und Software Engineering zurückgegriffen werden.

Das fertige Projekt soll auch dem Spieler ein Gefühl für Logik vermitteln, die von der Aufmachung von RoboCup zuerst abgeschreckt waren. Damit wird die Zielgruppe erweitert und man erreicht auch Spieler im Casualbereich.

2 Aufgabenstellung

Das Ziel dieser Arbeit ist es, eine grafische Fußballsimulation zu erstellen, in welcher der Spieler eigene Logiken für sein Fußballteam hinterlegt und somit deren Verhalten beeinflusst. Dabei besteht ein Team aus jeweils 10 Feldspielern und einem Torwart, wobei jeder dieser Spieler seine eigene Logik haben kann. Während eines Fußballspiels werden die gängigen Regeln beachtet und bei einer Missachtung je nach Stärke des Vergehens geahndet. Innerhalb der Simulation werden die Bewegungen des Balls physikalisch korrekt simuliert. Die Spieler besitzen als Limitierung ein Ausdauersystem, welches verhindern soll, dass ein Spieler dauerhaft sprintet oder den Ball mehrfach in kurzer Zeit mit maximaler Kraft kickt.

Die Simulation soll komplett dreidimensional und mit animierten Fußballspieler gestaltet werden. Wobei der Aktionsumfang der Spieler zuerst auf ein Basisrepertoire beschränkt wird. Dazu gehört die Aktionen „Laufen“, „Schießen“ und „Ball annehmen“. Die Ausarbeitung der grafischen Komponente wird in der Priorität hinter den funktionalen Teil der Arbeit gestellt.

3 Stand der Technik

3.1 Einleitung

Die Arbeit baut auf unterschiedlichen Technologien auf, deren Verständnis für die Projektbearbeitung unumgänglich ist. Da die Simulation, wie sie in Kapitel 2 beschrieben ist, nicht im gegebenen Zeitrahmen von Grund auf Programmiert werden kann, muss eine Game Engine gewählt werden, die Grundlagen wie die 3D-Animation oder die Physik-Berechnungen bereitstellt. Des Weiteren muss eine Technologie gewählt werden, die die Steuerung der Charaktere über im Voraus erstellte Abläufe erlaubt.

In diesem Kapitel sollen die Technologien, die für die genannten Zwecke in Betracht gezogen wurden, kurz dargestellt werden und die Technologieentscheidung begründet werden

3.2 Game Engine

3.2.1 Grundsätzliches

Wie in Kapitel 3.1 angesprochen ist eine Game Engine notwendig, welche die Grundfunktionalitäten einer 3D-Simulation bereitstellt. Es existieren heutzutage mehrere solcher Frameworks, auf die im Folgenden kurz eingegangen werden soll.

3.2.2 Source Engine

Source Engine wird von Valve entwickelt und ist durch die weit verbreiteten Spiele, die mit ihr entwickelt wurden, bekannt. Dazu gehört neben Half Life 2 oder Counter Strike: Source auch Portal. Die Source Engine kann neben den Betriebssystemen Windows, Linux und MAC OS auch auf der Playstation 3 oder der Xbox360 verwendet werden. Neben der Grafik Engine beinhaltet dieses Framework auch eine Physik Engine. (vgl. [1])

Das Hauptproblem der Source Engine ist, dass sie zum Zeitpunkt dieser Arbeit nicht mehr Lizenziert werden kann und der Nachfolger erst auf der Game Developers

Conference 2015 angekündigt wurde. Die Source 2 Engine soll nach aktuellen Aussagen zukünftig komplett kostenlos angeboten werden. (vgl. [2])

3.2.3 Unity

Die Engine Unity wird von Unity Technologies entwickelt und unterstützt, wie die Source Engine, sowohl Grafik als auch Physikberechnungen. Ein großer Vorteil von Unity ist die große Anzahl der unterstützten Plattformen. Neben den Betriebssystemen Windows, Mac und Linux und den bekanntesten Konsolen (Playstation 3, Xbox360, Nintendo Wii) unterstützt Unity auch Smartphones (iOS, Android, Windows Phone) und weitere Plattformen. (vgl. [3])
Unity bietet zum Zeitpunkt dieser Arbeit eine Gratis-, Education- und eine Pro-Version an. Die Pro Version kann für 1500$ erworben werden, allerdings beinhaltet dieses Paket noch nicht alles. Z. B. der Android Pro Support muss für weitere 1500$ erworben werden. (vgl. [4])

3.2.4 Unreal Engine 4

Unreal Engine wird von Epic Games entwickelt und ist auf allen verbreiteten Computer-Betriebssystemen lauffähig. Neben diesen unterstützt es allerdings auch die neue Konsolengeneration mit Playstation 4 und Xbox One, sowie mobile Anwendungen mit iOS oder Android. (vgl. [5])
Seit der erstmaligen Verwendung der Unreal Engine im Computerspiel „Unreal“ im Jahr 1998 konnten viele, teilweise sehr bekannte, Spiele erfolgreich auf der Engine aufbauen (vgl. [6]). Die Unreal Engine 4, das aktuellste Framework der Reihe, wurde in Spielen wie Tekken 7 oder Kingdom Hearts 3 eingesetzt. (vgl. [7])
Die Engine konnte zunächst für 19€ pro Monat lizensiert werden. Anschließend entschloss sich Epic Games UE4 für Studenten und Hochschulen kostenlos zur Verfügung zu stellen (vgl. [8]). Während des Projektzeitraums entfiel die monatliche Gebühr schließlich generell, was die Verbreitung der Engine weiter fördert. Bei einem finanziellen Erfolg eines mit der Unreal Engine 4 erstelltem Produkts fordert Epic Games eine prozentuale Gewinnbeteiligung (5% nach 3000$ Freibetrag). (vgl. [5])

3.2.5 Verwendung im Projekt

Da die erwähnten Frameworks sich im benötigten Funktionsumfang kaum bis überhaupt nicht unterscheiden war die Auswahl der geeigneten Engine nicht ohne weiteres möglich. Eines der Hauptmerkmale, die für die Entscheidung relevant waren, war der Preis und die Lizensierung des Produkts.
Durch diese Betrachtung wurde schnell klar, dass die Source Engine für dieses Projekt nicht in Frage kommt, da hier momentan keine Möglichkeit zur Verwendung existiert. Die Source Engine kann nicht mehr lizenziert werden und die neue Version Ist noch nicht öffentlich verfügbar.
Ein weiterer wichtiger Punkt war die Erweiterbarkeit des Codes zum Einbinden der externen KI. Da ein solcher Ansatz natürlich nicht den Anforderungen an normale Spiele entspricht, bietet keine der von uns in Betracht gezogenen Frameworks eine vorgefertigte Lösung für dieses Problem.
Aus diesem Grund muss der Code der Game Engine um eine Schnittstelle erweitert werden, die die Befehle von der KI entgegennimmt. Bei der Unity Engine ist dies nicht möglich, da dieses Framework in der Gratis-Version keine Erweiterung des Quellcodes um Plugins zulässt.
Aus den oben genannten Gründen fiel die Wahl auf die Unreal Engine 4. Diese Engine ist sowohl unter Windows als auch Linux verwendbar und bietet die nötigen Funktionen plattformunabhängig an. Des Weiteren ermöglicht sie den kostenlosen Zugriff auf alle Funktionen, sogar den Quellcode, der Engine.

3.3 Scripting-Ansatz

3.3.1 Grundsätzliches

Da die Steuerung der Spieler, wie in der Aufgabenstellung erwähnt, von außerhalb der Simulation stattfindet, muss ein geeigneter Ansatz gewählt werden, über den eine programmierte KI in die Simulation eingespeist werden kann. Hierbei darf kein Neukompilieren der Simulation nötig werden.
Hierbei fiel die Wahl auf eine Skriptsprache, da diese typischerweise leichtgewichtig und einfach zu erlernen sind.

Alternativ wäre ein Client-Server Ansatz denkbar. Der Nachteil hierbei ist, dass für jeden Spieler ein eigener Client implementiert werden muss. Der Aufwand und die Komplexität liegen für Client Implementierung höher als beim Erarbeiten eines Skriptes zur Steuerung der Spieler.
Da die Erstellung einer künstlichen Intelligenz für die Spieler auch für Personen ohne größere Programmierkenntnisse möglich sein soll, ist eine Skriptsprache der geeignetere Ansatz.

3.3.2 LUA

LUA ist eine leichtgewichtige Skriptsprache und kann auf verschiedenen Plattformen eingesetzt werden. Dabei bietet LUA eine Menge von Basisfunktionen die zur Behandlung eines Problems erweitert werden können. Zu den Basisfunktionen gehören beispielsweise die üblichen mathematischen Befehle.
Der Aufbau von LUA ermöglicht das einfache Einbinden der Skriptsprache in größere Projekte. Das liegt darin begründet, dass LUA in reinem ANSI C Code geschrieben ist und die Bibliothek keine weiteren Abhängigkeiten besitzt (vgl. [9]).
Die erste Version von LUA stammt aus dem Jahr 1993. Entwickelt wurde sie von einem Team der Päpstlich Katholischen Universität in Rio de Janeiro.
Die Größe der LUA Bibliothek ist mit rund 120 Kilobyte sehr gering. Durch die Implementierung in C ist LUA eine schnelle Skriptsprache. Die meisten anderen Skriptsprachen können hier nicht mithalten.
Seit Version 5 steht LUA unter einer MIT Lizenz. Damit kann die Skriptsprache problemlos für Projekte verwendet und erweitert werden. (vgl. [10])
Zudem ist die Sprache leicht zu erlernen, da sie einen C ähnliche Syntax besitzt und es bereits sehr viele Bücher und Webseiten darüber gibt. Die Bibliothek selbst ist sehr gut getestet und wird in vielen Programmen, unter anderem in Videospielen wie Garrys Mod eingesetzt.

3.3.3 Alternative Skriptsprachen

Außer LUA stehen noch andere Skriptsprachen zur Verfügung. Bekannte und wichtige Vertreter sind JavaScript, Python oder Perl. Diese sind zwar mächtiger als LUA, aber auch dementsprechend komplexer.

Da das Augenmerk auf der einfachen Integration und der unproblematischen Verwendung lag, fiel die Entscheidung auf LUA. Hier treten, wie oben genannt, bei der Integration keine weiteren Abhängigkeitsprobleme auf. Außerdem genügen der Umfang und die Möglichkeiten von LUA für die Umsetzung des Projektes vollkommen.

3.4 Robo Cup

RoboCup ist ein Wettbewerb, bei dem es um Robotik und künstliche Intelligenz geht. Im Wettbewerb gibt es verschiedene Kategorien, in denen die Teams gegeneinander antreten. Als Gemeinsamkeit haben alle Kategorien, dass es darum geht Roboter autonom Fußball spielen zu lassen.

In den meisten Kategorien steht das Entwickeln und Konstruieren der Roboter im Vordergrund. Die Kategorien unterscheiden sich in der Größe und den Eigenschaften der Roboter, sowie in der Größe der Teams.

Zusätzlich zu den physikalischen Wettbewerben gibt es auch die Simulation League, in der die Programmierung der künstlichen Intelligenz im Vordergrund steht. Hier entfällt die Entwicklung und Konstruktion der Roboter und die Teams fokussieren ihre Anstrengungen auf die Weiterentwicklung der Intelligenz der Spieler (vgl. [11, p. 2f]).

Im Gegensatz zu den Kategorien, bei denen die Robotik im Vordergrund steht, ist die Simulation League in Bezug auf die Aufgabenstellung sehr interessant. In der RoboCup Simulation League werden zwei Teams mit jeweils bis zu elf Spielern simuliert. Die Simulation League gibt es in zwei Varianten: Eine 2D-Version und eine 3D-Version.

In der 2D-Variante beschränkt sich die Simulation auf Blickwinkel und eine simple zweidimensionale Physik. Somit sind hier keine komplexen, realistischen Spielzüge

möglich. Das mögliche Spielverhalten entspricht allerdings dem, der einfacheren Robotik-Kategorien.
Die 3D-Version bietet eine komplexere Physik Engine, welche entsprechend dreidimensionale Bewegungen des Balls ermöglicht. Dadurch wird das Spielverhalten realistischer. Die RoboCup Simulation League hat jedoch weiterhin die Simulation von Fußballrobotern im Vordergrund.
Technisch basiert die Umsetzung der Simulation League auf einer Client-Server-Architektur (vgl. [11, p. 11]). Der Server kontrolliert die Simulation und Implementiert die Regeln und die Physik. Die Spielerintelligenz wird von den Clients realisiert. Pro Spieler wird ein Client benötigt. Die Clients kommunizieren über UDP/IP mit dem Server und erhalten so Informationen über das Spiel und können den entsprechenden Spieler steuern.
Die Programmierung der Clients ist in verschiedenen Programmiersprachen möglich, wodurch die Gestaltung der Spielerintelligenz sehr flexibel gestaltet. Es ist möglich das Spielverhalten individuell für einzelne Spieler zu gestalten, aber auch das Zusammenspiel der Spieler im Team kann gestaltet werden. Dazu stehen Kommunikationsmechanismen zwischen den Spielern zur Verfügung.

Viele Ansätze aus der RoboCup Simulation League können für die Aufgabenstellung dieses Projekts übernommen werden. Zum einen ist die Kommunikation der Spieler untereinander ein interessanter und wichtiger Punkt für die Entwicklung komplexer Spielzüge. Auch die unabhängige Programmierung der Spielerlogik von einer Kontrollinstanz, die Regeln und Physik der Simulation sicherstellt, ist ein guter Ansatz um Flexibilität zu gewährleisten.

3.5 Künstliche Intelligenz

Künstliche Intelligenz beschäftigt sich mit Möglichkeiten Computer zu nutzen um komplexe Probleme zu lösen, die normalerweise menschliche Intelligenz zur Lösung voraussetzen. Als Teilgebiet der Informatik befasst sich die künstliche Intelligenz mit der Entwicklung intelligenter Computersysteme, die ein ähnliches Verhalten aufweisen wie die menschliche Intelligenz (vgl. [12, p. 13f]). Wobei es nicht direkt um

den Vergleich mit der menschlichen Denkweise geht, sondern darum mit den nachgebildeten menschlichen Fähigkeiten komplexe Problemstellungen zu lösen. Dabei handelt es sich um Probleme, die von klassischen Computerprogrammen nicht bewältigt werden können.

Das Feld der künstlichen Intelligenz ist breit aufgestellt und umfasst viele verschiedene Einsatzgebiete.
Ansätze der künstlichen Intelligenz sind zum Beispiel im Gebiet des Data Mining zu sehen. Mit der Technik soll die Wissensdarstellung und die Wissensverarbeitung unterstützt werden. Hier werden die Methoden verwendet um Muster und Regeln aus großen Datenmengen zu extrahieren. Diese Methoden können Informationen und Zusammenhänge finden, die für den Menschen so nicht erkennbar wären.
Auch im Bereich der Spracherkennung ist die künstliche Intelligenz vertreten. Hier geht es darum das Gesprochene zu verstehen und zu verarbeiten. Dazu muss der Computer in der Lage sein nicht nur die Syntaktik der Sprache zu verstehen sondern auch die Semantik. Das bedeutet der Computer muss ein ähnliches Verständnis für die Sprache besitzen wie der Mensch um sie zu verstehen. Die Darstellung der Sprache über ihre Grammatik ist hierzu ein anzutreffender Ansatz.
Ein weiteres, sehr bekanntes Beispiel für künstliche Intelligenz sind Computerspiele. In den meisten Computerspielen gibt es computergesteuerte Gegner oder Mitspieler. Diese müssen auf das Verhalten und die Strategie des Spielers reagieren können. Dazu müssen sie eigene Entscheidungen auf der Basis von bestimmten Umgebungswerten ableiten können.

Oft wird der Begriff künstliche Intelligenz auch mit maschinellem Lernen in Verbindung gebracht. Maschinelles Lernen stellt ein Teilgebiet der künstlichen Intelligenz dar, bei dem Computer auf Basis von vorhandenen Daten neues Wissen generieren. Das kann sowohl das automatisierte Treffen von Entscheidungen sein, als auch die Erkennung von Mustern und Zusammenhängen.
Dies wird in vielen Fällen mittels künstlicher neuronaler Netze versucht. Sie bilden menschliche Hirnstrukturen nach und lernen das gewünschte Verhalten durch die

Verarbeitung von Beispielen. Ähnlich dem menschlichen Gehirn greifen sie eine Art Erinnerungsvermögen zurück, wenn sie auf ähnliche Problemstellungen treffen (vgl. [12, p. 16]).

Das Thema künstliche Intelligenz hat Relevanz für diese Studienarbeit, da die Spieler basierend auf bekannten Informationen Aktionen ausführen sollen. Diese Informationen können die eigene Position, den Erschöpfungszustand, die Position des Balls, der Mitspieler und der Gegner sein. Aber auch Informationen über den Spielstand oder Beobachtungen über die gegnerische Strategie kommen in Frage. Auf Basis dieser Informationen können weitere Informationen abgeleitet werden und so Schlüsse über den Spielverlauf und die notwendige Strategie getroffen werden. Darauf aufbauend kann der Spieler Aktionen durchführen.

3.6 Zusammenfassung

Insgesamt kann gesagt werden, dass bereits technologische Grundlagen für die Umsetzung der Aufgabenstellung bestehen.

Der Einsatz einer Game Engine vereinfacht die Entwicklung der Simulationsumgebung deutlich. Die Umsetzung der gesamten Anforderungen wäre ohne ein solches Framework nicht möglich. Gerade die Entwicklung von Physik- und Grafiksimulation sind sehr komplex und zeitintensiv.

Eine Skriptsprache ermöglicht die unkomplizierte Entwicklung von künstlichen Intelligenzen für die Spieler. Das Gebiet der künstlichen Intelligenz wird in der Wissenschaft zunehmend untersucht. Es bestehen bereits viele Konzepte, die sich auf die Bedürfnisse der Aufgabestellung übertragen lassen.

Das Vorbild, die RoboCup Simulation League, enthält einige Ansätze, die für dieses Projekt geeignet sind.

Alles in Allem gibt es viele bereits vorhandene Technologien und Entwicklungen, die in diese Studienarbeit mit einfließen können oder Anregungen für die Umsetzung bieten.

4 Planung / Struktur

4.1 Projektmanagement

Die Projektentwicklung ist in zwei Phasen unterteilt. In der ersten Phase, die während des fünften Semesters stattfindet, werden zunächst eine Planung sowie die passende Architektur und ein Prototype entwickelt. In der zweiten Phase, die während des sechsten Semesters stattfindet, wird eine Implementierung nach der entwickelten Architektur erstellt und getestet.

Das Projekt wird mit einem vier Mann staken Team umgesetzt, aus diesem Grund wird ein guter und regelmäßiger Austausch gepflegt. Hierzu findet wöchentlich ein Meeting des gesamten Teams statt um den aktuellen Vorschritt zu besprechen und neue Aufgaben zu bestimmen. Zur Sicherstellung das in die richtige Richtung gearbeitet wird findet monatlich ein Treffen mit dem Auftraggeber beziehungsweise Betreuer statt. Hierbei wird über den aktuellen Stand berichtet und weitere Ziele diskutiert. Die Protokolle der Meetings finden sich im Anhang zu diesem Dokument.

Aufgrund dieser agilen Softwareentwicklung entstehen nur grobe Zeitpläne und Meilensteine.

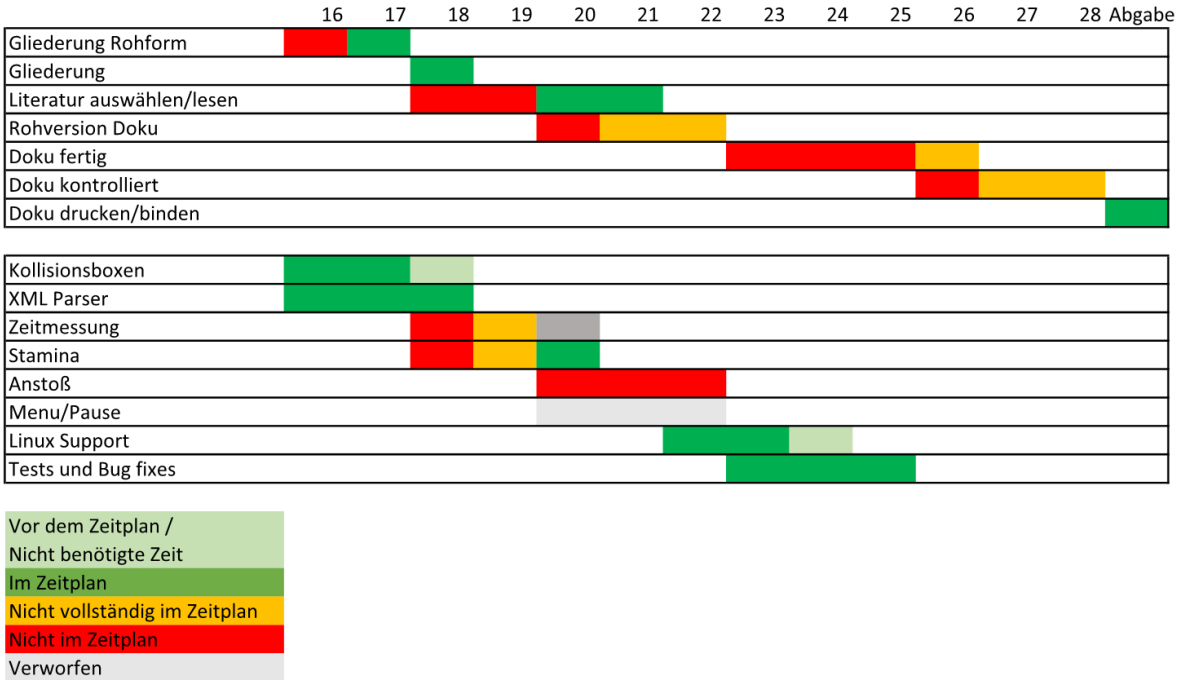

Abbildung 1 Meilensteinplan Theoriesemester 2

Der in Abbildung 1 Meilensteinplan abgebildete Plan zeigt den Verlauf des Projekts im zweiten Theoriesemester. Die Entwicklung des Projektes fand hierbei parallel zur Erstellung der Dokumentation statt. Besonders auffällig ist, dass die Entwicklung des eigentlichen Projektes möglichst früh abgeschlossen wurde, um die so verfügbaren Ressourcen für die Dokumentation zu nutzen. Trotz dieser Ausrichtung wurde der Zeitaufwand der Dokumentation unterschätzt und der Zeitplan konnte auf Dauer nicht eingehalten werden.
Der aufgeführte Punkt „Menu/Pause“ galt als optionales Ziel, welches nur realisiert werden würde, wenn der Projektstand weit vor dem Zeitplan liegen würde. Da das Projekt auch ohne diesen Punkt voll funktionsfähig ist und der Zeitplan keine weitere Verzögerung zuließ wurde dieser Punkt verworfen.

4.2 Analyse der Aufgabenstellung

Aus der vorgegebenen Aufgabenstellung werden zunächst die elementaren und zusätzlichen Eigenschaften herausgearbeitet und in Muss-, Kann- und Sollkriterien unterteilt. Die erstellten Kriterien werden nach ihrer Machbarkeit analysiert und eine geeignete Umsetzung ausgewählt.

4.2.1 Muss-Kriterien

- Simulation eines Fußballspiels
    - Physikalische Korrektheit
    - Einhaltung der Fußballregeln
    - Ausdauersystem für Spieler
    - Kompatibel mit Windows und Linux
- Erstellen einer KI für die Fußballspieler
- Nachträgliches erstellen neuer KIs ermöglichen

4.2.2 Soll-Kriterien

- Bestrafung der Spieler bei Nichteinhaltung der Regeln
- 3D-Darstellung

4.2.3 Kann-Kriterien

- Grafischer Editor zur Erstellung von einer KI

4.2.4 Analyse der Kriterien

Die Anforderung einer physikalisch korrekten 3D Simulation wird über eine Game Engine realisiert, da diese eine eingebaute Physik Engine besitzen. In diesem Projekt wird die Unreal Engine von „Epic Games“ verwendet. Die Unreal Engine erfüllt ebenso die Bedingung der Plattformunabhängigkeit, es wird Windows und Linux unterstützt.

Um die KIs der Spieler nachträglich ändern zu können kann eine Skriptsprache verwendet werden, in diesem Projekt wird auf die freie Skriptsprache LUA gesetzt. Um die Skripte den Spielern zuzuordnen oder andere Einstellungen des Teams vorzunehmen wird eine XML Schnittstelle implementiert.

4.3 Architektur

Die Architektur baut auf der Unreal Engine auf und wird aus diesem Grund vollständig in C++ entwickelt. Das Projekt greift auf existierende Technologien wie XML und LUA zurück:

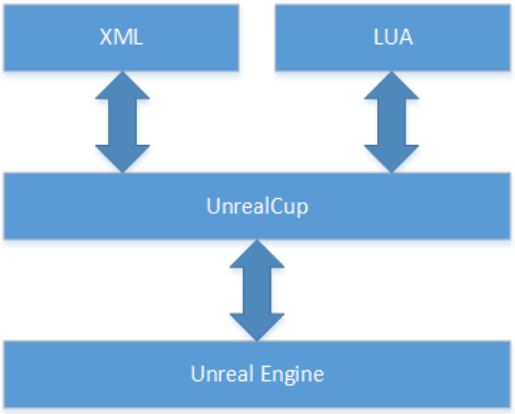

Abbildung 2: Architektur Übersicht

Der wichtigste Teil der Architektur ist die Schnittstelle zwischen den LUA-Skripten und den Fußballspielern. Hierbei müssen 22 Threads für die 22 Spieler mit dem Gamethread synchronisiert werden.

Um die Schnittstelle zwischen den Workern und dem Hauptthread zu ermöglichen, wird ein vereinfachtes Remote Procedure Call Verfahren verwendet (vgl. [13, p. 71ff.]) Hierzu wird in der Klasse RobotWorker z.B. getPosition() aufgerufen. Der Thread muss angehalten werden, da diese Funktion einen Rückgabeparameter hat. Der gewünschte Funktionsaufruf wird mit insertCall() in die RobotControl übergeben. Sobald der Hauptthread Rechenzeit zur Verfügung hat ruft dieser die Methode tick() auf. Diese Methode arbeitet dann die Queue, der Funktionsaufrufe ab. Wenn ein Aufruf auf eine Antwort wartet wird diese zurück an den Worker mit setReturnValue() gegeben. Die Busy-Waiting Schleife im Thread kann nach Erhalt des Ergebnisses dies an das Skript zurückgeben. Um spätere Erweiterungen leicht zu ermöglichen z.B. andere Skriptsprachen oder eine Implementierung direkt in C++ wird über eine Spezialisierung des RobotWorkers die LUA Schnittstelle hinzugefügt.

In Abbildung 3 ist der Ablauf eines Funktionsaufrufs vom Worker im Hauptthread dargestellt.

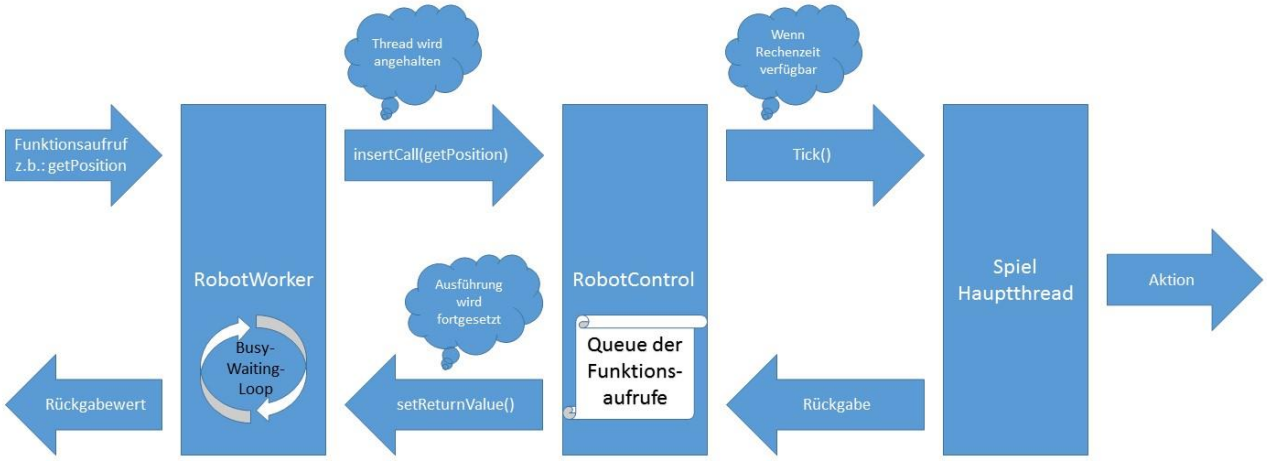

Abbildung 3: Remote Procedure Call Umsetzung

Die 22 Threads der Spieler müssen gescheduled und in ihrer Leistung beschränkt werden um eine faire Simulation zu ermöglichen. In der LUA Implementierung geschieht dies über die Funktion allowedToRun() diese wertet die bisherige Rechenzeit aus und veranlasst ein Anhalten des Threads um das zehnfache seiner Rechenzeit um seine Auslastung auf 1/10 eines Prozessorkerns zu beschränken. Dies hat zur Folge, dass für alle 22 Spieler 2,2 CPUs benötigt werden. Aufgrund der zusätzlichen Threads für die 3D Engine wird ein Vierkernprozessor zum idealen Ablauf benötigt.

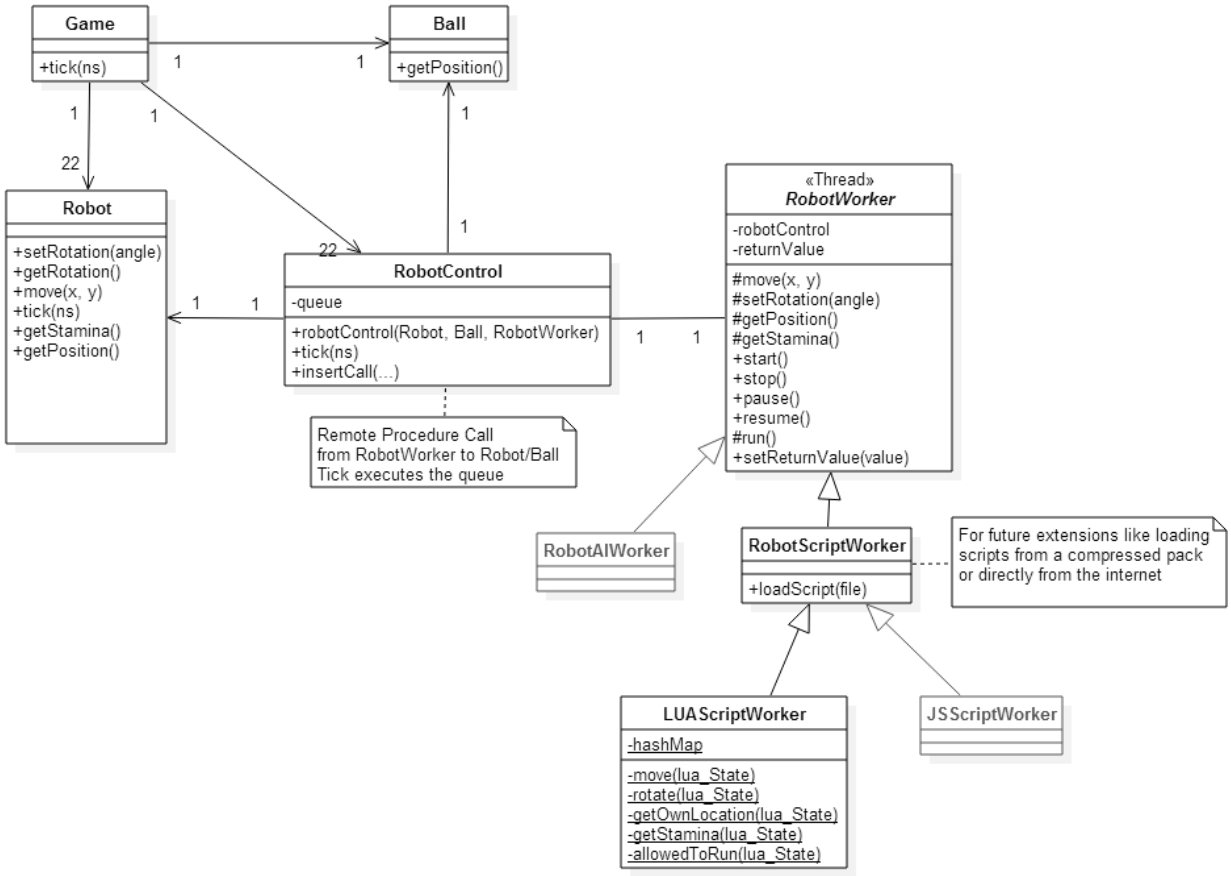

Abbildung 4: Architektur als Klassendiagramm

5 Umsetzung

5.1 Umsetzung der Simulationsumgebung

5.1.1 Grundsätzliches

Die Umsetzung der Simulationsumgebung basiert auf der Unreal Engine 4. Die Unreal Engine 4 ist eine Game Engine, also ein Framework um Computerspiele zu erstellen. Sie bietet bereits viele vorgefertigte Funktionen an, auf die bei der Eigenentwicklung aufgesetzt werden können.
Der große Vorteil eines solchen Frameworks besteht darin, dass die Entwicklung nicht von Null an begonnen werden muss, sondern auf ein breites Spektrum von Basisfunktionen zugegriffen werden kann. Gerade im Hinblick auf die Implementierung einer 3D-Grafik oder einer realistischen Physiksimulation bleibt hier viel Grundlagenarbeit erspart. Ohne den Einsatz dieses Frameworks wäre die Umsetzung dieser Studienarbeit unmöglich.
Nicht nur bei der Entwicklung bietet ein Framework große Vorteile. Durch den modularen, komponentenbasierten Aufbau von Frameworks, können eigene Teile einfacher und besser getestet werden. Das liegt daran, dass bereits eine grundlegende Architektur vorgegeben ist und Schnittstellen klar spezifiziert sind. Auch die spätere Weiterentwicklung durch andere Teams wird durch die Nutzung eines Frameworks vereinfacht.
Der Einarbeitungsaufwand in ein solches Framework darf nicht unterschätzt werden, da Game Engines sehr umfangreich sind und deutlich mehr Funktionalität bereitstellen, als für dieses Projekt notwendig ist. Verglichen mit einer kompletten Eigenentwicklung ist dieser Overhead für die Einarbeitung in das Framework allerdings zu vernachlässigen.

Die Gründe warum auf die Unreal Engine 4 gesetzt wird, wurden bereits im Teil Game Engine im Kapitel Stand der Technik angesprochen. Der wichtigste Vorteile ist die kostenfreie Verfügbarkeit für studentische Projekte wie dieses. Zudem ist die Unreal Engine 4 im Gegensatz zu anderen Engines quelltextoffen. Ein weiterer

Punkt, der sowohl als positiv, als auch negativ gesehen werden kann, ist die Aktualität der Engine. Das Framework wurde erstmals im März 2014 veröffentlicht und seither regelmäßig mit Updates versorgt. Damit ist diese Studienarbeit ein Projekt am Puls der Zeit. Auf der anderen Seite existieren durch die Aktualität noch kaum andere Projekte und es gibt noch verhältnismäßig viele Fehler in der Engine. Auch Anpassungsprobleme durch Updates sind ein Thema. Während der Bearbeitung wurden Teilweise die Schnittstellendefinitionen des Frameworks geändert.

Insgesamt teilt sich die Umsetzung der Simulationsumgebung in drei Teile:
Zum einen in die Erstellung des Spielrahmens mit dem Spielfeld und vor allem der Regelimplementierung. Dieser Teil ist die Steuerungskomponente der Simulationsumgebung.
Ein weiterer Teil ist die Implementierung der Spieler innerhalb der Simulationsumgebung. Hier gehört die Implementierung der Funktionen der Spieler, auf die über die Skriptschnittstelle zugegriffen werden kann.
Der letzte Teil ist die Skriptschnittstelle, die die Weitergabe der Spielerfunktionalitäten an die KI-Skripte vornimmt. Dazu gehört auch der im Kapitel Architektur beschriebenen Remote-Procedure-Call, der die internen Spielerfunktionen im Spiel-Thread aufruft.

5.1.2 Regel- und Spielimplementierung

Die Implementierung des Spiels ist untergliedert in die Erstellung eines Spielfeldes und der Programmierung der Spielregeln.
Die Erstellung des Spielfeldes ist über den graphischen Editor der Unreal Engine 4 realisiert. Dieser bietet die Möglichkeit 3D-Objekte zu einer Szene zu verbinden und anzuordnen. Das funktioniert per Drag and Drop mit zusätzlichen Menüs für Detaileinstellungen. Somit ist das Konzept hier sehr ähnlich zu anderen Tools zur Erstellung von 3D-Szenen und Objekten wie beispielsweise Blender oder andere CAD Systeme.

Zum großen Teil kommen von der Unreal Engine mitgelieferte 3D-Modelle zum Einsatz. Stellvertretend seien hier die Spieler genannt. Auf die 3D-Modelle sind zum Teil spezielle Texturen wie der Rasen mit den Linien oder die DHBW-Logos aufgebracht. In Abbildung 5 ist das so entstandene Spielfeld zu sehen.

Abbildung 5: Modelliertes Spielfeld

Das Spiel verwendet ein kartesisches Koordinatensystem, dessen Ursprung sich im Mittelpunkt(Anstoßpunkt) befindet.

Die Implementierung der Fußballregeln wird zum einen über die Gestaltung der 3D-Modellierung realisiert. Um den Aufwand für die Implementierung der Fußballregeln zu reduzieren, wird mit Banden gespielt. Diese wurden als Kollisionsboxen modelliert und umgeben das gesamte Spielfeld. Die Kollisionsboxen sind weder für die Spieler, noch für den Ball durchlässig. Alle Objekte prallen daran ab wie es bei einer Bande der Fall wäre. Diese Entscheidung wurde getroffen um der Implementierung von Seiten- und Toraus, sowie Einwurf und Eckball aus dem Weg zu gehen. Damit entspricht die Simulation hier den Hallenfußballregeln bei denen ebenfalls Banden

zum Einsatz kommen und so die Aus-Regelung nicht zum Tragen kommt. Abbildung 6 zeigt das Spielfeld mit den modellierten Kollisionsboxen im Editor. Diese sind als transparente Rechtecke im Hintergrund zu erkennen.

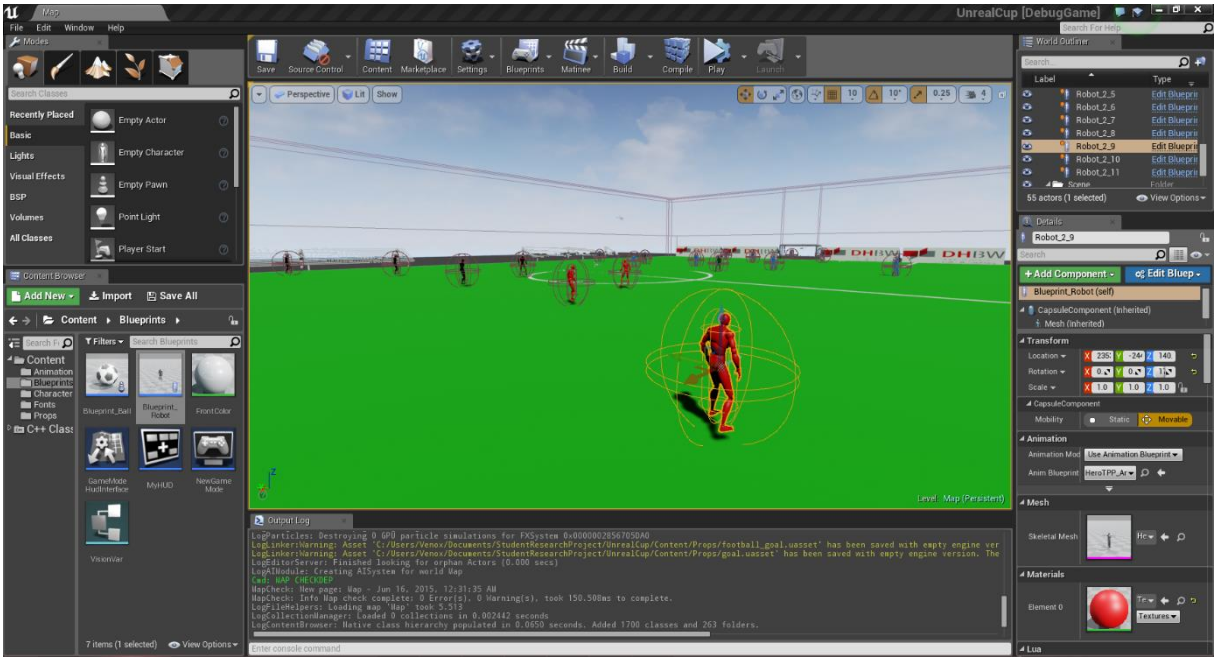

Abbildung 6: Spielfeld mit Kollisionsboxen

Weiterhin ist im Spiel eine Torerkennung und daraufhin das Ausführen eines Anstoß implementiert. Diese sind in einer für die Unreal Engine spezifischen Technologie, die sich Blueprint nennt, umgesetzt.

Blueprints sind ein Ansatz der grafischen Programmierung. Darin lassen sich relativ komfortabel Funktionen modellieren. Automatische Syntax- und Semantikprüfungen vereinfachen die Programmierung innerhalb des komplexen Frameworks zusätzlich.

Ein Blueprint kann eine oder mehrere logische Funktionalitäten enthalten. Funktionsaufrufe oder Events sind als Rechtecke dargestellt. Verbunden sind diese Rechtecke über verschiedenfarbige Linien. Diese Linien repräsentieren Kontroll- und Datenflüsse. Die Kontrollflüsse bestimmen die Reihenfolge der Methodenaufrufe. Zum anderen werden die Datenflüsse zum Austausch von Variablen, als Rückgabewerte einer Funktion oder als Übergabeparameter an eine andere Funktion

verwendet. Verschiedene Farben symbolisieren verschiedene Typen von Variablen. Dadurch ist auch ein komplexes Blueprint übersichtlich gehalten.

In Abbildung 7 ist das Blueprint für die Torerkennung zu sehen. Sobald sich ein Objekt innerhalb der Kollisionsbox des Tors befindet, wird geprüft, ob es sich um den Ball handelt. Dann wird geprüft welches Team das Tor erzielt hat und die Durchführung eines Anstoßes angestoßen.

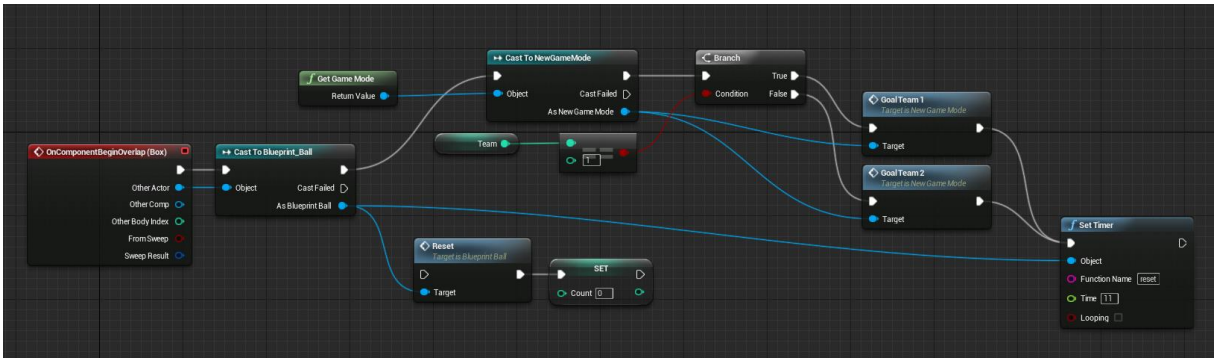

Abbildung 7: Blueprint für die Torerkennung

Genauso sind weitere Funktionalitäten, die das Zurücksetzen des Balls nach Ablauf eines Timers von 10 Sekunden oder die Messung der Spielzeit in Blueprints realisiert.

Die Konfiguration des Spiels wird aus der Datei „Teams.xml“ eingelesen. Darin ist spezifiziert aus welchen XML-Dateien die Konfiguration der beiden Teams gelesen werden soll. Außerdem wird hier die Spieldauer für ein Spiel festgelegt.

In den Konfigurationsdateien der einzelnen Teams werden für jeden Spieler ein KI-Skript, ein Name und die Startposition angegeben. Abbildung 8 zeigt eine solche Teamkonfiguration.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Players>
  <Player ScriptLocation="script_Player1_new.lua" Name="Peter" X="100" Y="700"></Player>
  <Player ScriptLocation="script_Player1_new.lua" Name="dieter" X="150" Y="100"></Player>
  <Player ScriptLocation="script_Player1_new.lua" Name="franz" X="200" Y="100"></Player>
  <Player ScriptLocation="script_Player1_new.lua" Name="hans" X="250" Y="100"></Player>
  <Player ScriptLocation="script_Player1_new.lua" Name="rudolf" X="300" Y="100"></Player>
  <Player ScriptLocation="script_Player1_new.lua" Name="heinz" X="350" Y="100"></Player>
  <Player ScriptLocation="script_Player1_new.lua" Name="umberto" X="400" Y="100"></Player>
  <Player ScriptLocation="script_Player1_new.lua" Name="konrad" X="450" Y="100"></Player>
  <Player ScriptLocation="script_Player1_new.lua" Name="angelo" X="500" Y="100"></Player>
  <Player ScriptLocation="script_Player1_new.lua" Name="merte" X="550" Y="100"></Player>
  <Player ScriptLocation="script_Player1_new.lua" Name="plopp" X="600" Y="100"></Player>
</Players>
```

Abbildung 8: Konfiguration eines Teams

Das Auslesen der XML-Dateien erfolgt über einen in C++ realisierten Parser beim Starten des Programms. Die Konfiguration über XML-Dateien ist sinnvoll, da XML auch von Menschen gut bearbeitet werden kann und weit verbreitet ist. Es wird ermöglicht, dass bei der Erstellung der künstlichen Intelligenzen für jeden Spieler eine spezielle Version erstellt werden kann. Außerdem hat der Anwender die Möglichkeit eine eigene Formation zu konfigurieren, ohne Änderungen am Quellcode vornehmen zu müssen.

Im Bereich der Regelimplementierung gibt es noch einige Möglichkeiten zur Erweiterung. Beispielsweise fehlt die Erkennung der Abseitsregel oder die Erkennung von Fouls. Wobei hier vor allem die Erkennung desjenigen Spielers, der ein Foul begeht schwierig zu lösen ist. Ebenfalls fehlen die oben angesprochenen Seiten- und Toraus Erkennung mit dazugehörigem Einwurf beziehungsweise Eckball.

5.1.3 Spielerimplementierung

In diesem Unterkapitel wird die Umsetzung des menschlichen Fußballspielers in der Simulation beschrieben. Daraus gehen einige Funktionen hervor, die der simulierte Spieler besitzt. Diese sind in geringerem Umfang als die Möglichkeiten, die ein echter Fußballspieler hat.

Die Implementierung der Funktionen des Spielers ist zum großen Teil im C++ Code vorgenommen. Kleine Teile sind wie die Implementierung der Spielregeln im Blueprint umgesetzt.
Die hier beschriebenen Spielerfunktionalitäten können über die Skriptschnittstelle durch die künstliche Intelligenz aufgerufen werden.

Die im Blueprint umgesetzten Funktionen sind das Annehmen des Balls, die Erkennung, ob der Spieler den Ball hat und die Bewegung zu einer bestimmten Position. Diese Funktionalitäten sind nicht im C++ Code umgesetzt, da sich die Umsetzung hier deutlich komplizierter gestaltet hätte als im Blueprint. Das liegt vor allem daran, dass auf Objekte zugegriffen werden muss, die innerhalb des verwendeten Basisspielers der Unreal Engine 4 liegen. Der Zugriff darauf ist im Blueprint deutlich einfacher gestaltet. Die Blueprint-Funktionen werden durch Events, die von Funktionen im C++ Code gesendet werden, aufgerufen. Für die Ausführung des Spiels macht es keinen Unterschied, ob die Funktionen nativ im C++ Code oder im Blueprint implementiert sind. Die Blueprints werden genau wie der C++ Code kompiliert. Durch diese Übersetzung in Machinencode bestehen keinerlei Unterschiede in der Ausführungsgeschwindigkeit wie es bei einer interpretierten Sprache möglich wäre.
In Abbildung 9 sind Teile der umgesetzten Funktionalitäten zu sehen. Es ist zu erkennen, dass diese Funktionen relativ klein sind. Eine Entsprechung im C++ Code wäre viel größer.

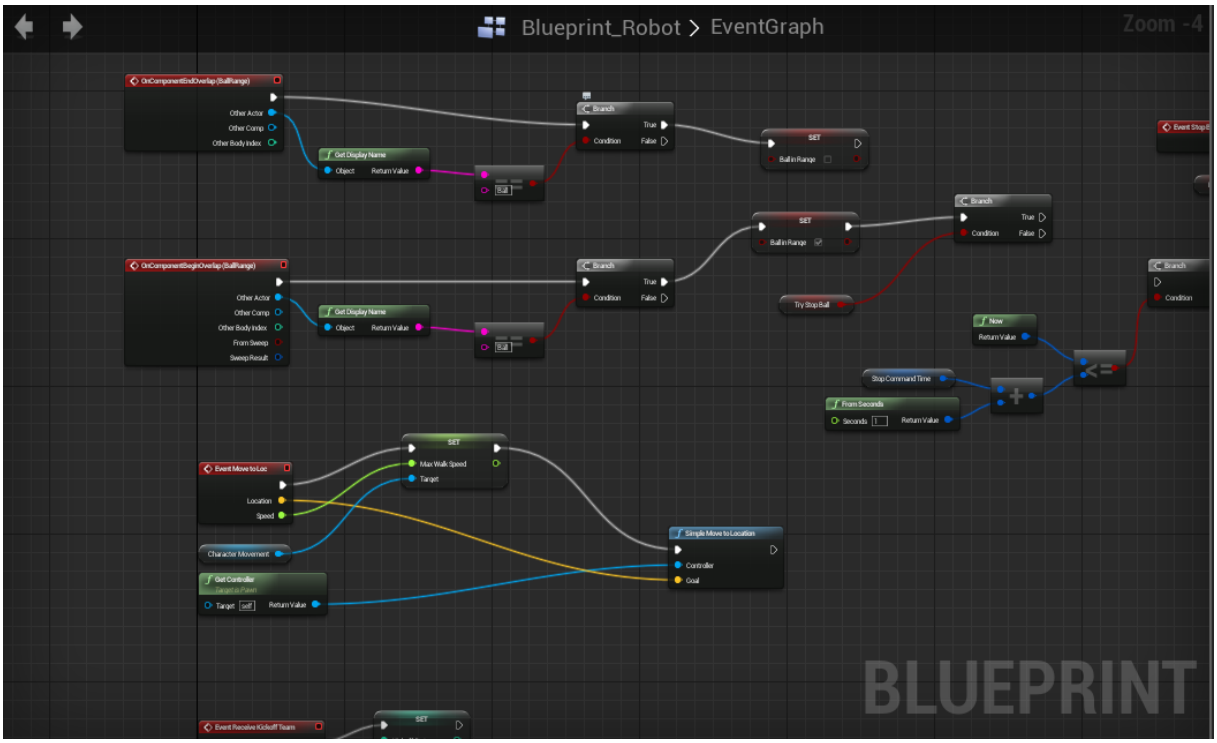

Abbildung 9: Blueprint-Funktionen des Spielers

Die weiteren Funktionen des Spielers sind direkt im C++ Code entwickelt. Diese finden sich alle in der Datei „robot.cpp“.

Es gibt Funktionen, die die Bewegung des Spielers realisieren. Dazu gehören eine Funktion für einfache Vorwärtsbewegung, eine Funktion zum Drehen des Spielers und eine Funktion, die den Spieler zu einer bestimmten Position bewegt. Für diese Funktion werden im C++ Code Berechnungen für die Geschwindigkeit und die Rotation in Bewegungsrichtung durchgeführt. Mit diesen Werten wird dann die Bewegungsfunktion im Blueprint aufgerufen.

Bei allen Bewegungsfunktionen wird der Ausdauerwert des Spielers abhängig von der Geschwindigkeit und der zurückzulegenden Distanz verringert. Hat der Spieler nicht genug Ausdauer um die Strecke mit der angegebenen Geschwindigkeit zurückzulegen, wird die Geschwindigkeit reduziert. Damit wird verhindert, dass Spieler sich unbegrenzt mit voller Geschwindigkeit über das Spielfeld bewegen. Diese Entwicklung eines Ausdauersystems ist zum einen in der Aufgabenstellung

gefordert und macht zum anderen auch Sinn, da sich die Simulation hier an der Realität eines Fußballspielers orientiert.

Der Ausdauerwert des Spielers wird mit einem bestimmten Intervall wieder erhöht. Damit ist eine Erholung des Spielers von einer Anstrengung möglich.

Im C++ Code ist die Kick-Funktion, die den Spieler den Ball kicken lässt implementiert. Diese Funktion ruft eine Funktion im Blueprint des Balls auf, über die eine Kraft auf den Ball eingewirkt wird. Dieser bewegt sich dann aufgrund der simulierten Physik. Innerhalb der Kick-Funktion wird abhängig von der Stärke des Schusses der Ausdauerwert reduziert. Hier wird analog zur Bewegung die Stärke des Schusses reduziert, wenn nicht genug Ausdauer zur Verfügung steht.

Eine weitere Gruppe von Funktionen liefert Positionen von Objekten zurück. Die Position des Spielers und der beiden Tore sind hierbei zu jeder Zeit bekannt und werden zurückgeliefert.
Die Position des Balls und anderer Spieler ist nur bekannt, wenn diese sich im Sichtfeld des Spielers befinden. Ist das nicht der Fall werden die Spielerpositionen gar nicht zurückgeliefert. Das ist möglich, da diese als Liste zurückgegeben werden. Beim Ball wird hier die Null-Position zurückgegeben. Dadurch erscheint der Ball für den Spieler im Ursprung des Koordinatensystems zwar unsichtbar, doch eine andere Lösung war an dieser Stelle nicht möglich, da eine Position zurückgegeben werden muss. Auch ein negativer Wert ist nicht möglich, da diese Werte durch die Wahl des Koordinatensystems während des Spiels auftreten. Die Problematik tritt allerdings selten auf, da es sehr unwahrscheinlich ist, dass sich der Ball während des Spiels genau im Ursprung befindet.
Die Wahrnehmung des Spielers ist so konfiguriert, dass er alle Objekte in einem Radius von 5 Metern wahrnehmen kann. Diese Wahrnehmung simuliert je nach Position des Objekts die Sinne Hören, Fühlen oder Sehen. Zusätzlich kann er beliebig weit entfernte Objekte in einem Horizontalen Sichtfeld von 120 Grad sehen. Das entspricht nicht unbedingt der menschlichen Realität. Es ist jedoch sehr

schwierig das menschliche Sehvermögen zu modellieren. Das liegt daran, dass der Mensch ein horizontales Sichtfeld von knapp 180 Grad besitzt. Ohne Augenbewegung kann allerding immer nur in einem Bereich von etwa 10 Grad scharf gesehen werden. Aus diesen Gründen wurde das Sichtsystem mit 120 Grad modelliert, in dem der Spieler alle Objekte scharf erkennen kann. Eine Erweiterungsmöglichkeit wäre hier, die Position der Objekte abhängig von der Entfernung und eventuell des Winkels mit einem Fehler zu versehen.

Im Spieler sind auch Methoden zur Kommunikation implementiert. Die Spieler besitzen eine Funktion um eine Nachricht zu sprechen. Diese Nachricht wird an alle anderen Spieler übertragen und in einer Liste gespeichert. Über eine weitere Funktion können die Nachrichten aus der Liste gehört werden. In dieser Implementierung kann jeder Spieler, das gesprochene eines anderen Spielers unabhängig von der Entfernung gehört werden. Dieses Modell könnte in einer Überarbeitung so erweitert werden, dass die Entfernung beim Hören zum Tragen kommt.

Dann besitzt der Spieler noch Funktionen, die die Team-Id, die Spieler-Id innerhalb des Teams und die bereits gespielte Zeit zurückliefern.

Die Funktionen des Spielers, sind Funktionen innerhalb des Spiels. Sie werden über die Skriptschnittstelle an die künstliche Intelligenz weitergereicht und von dort aus aufgerufen.

5.1.4 Skriptschnittstelle

Die Skriptschnittstelle ist die Implementierung des im Kapitel Architektur beschriebenen Remote-Procedure-Calls.
Die von der künstlichen Intelligenz kommenden Funktionsaufrufe werden in der „RobotControl“ gespeichert. Von dieser werden sie, wenn Rechenzeit für den Spieler zu Verfügung steht, im Spiel aufgerufen. Funktionen, die einen Rückgabewert

erwarten, pausieren die Abarbeitung der künstlichen Intelligenz, bis der Rückgabewert vom Spiel geliefert wird.
Die KI-Skripte werden so gescheduled, dass keine künstliche Intelligenz die gesamte Rechenzeit für sich beanspruchen kann. Über ein faires Verfahren wird jeder künstlichen Intelligenz 1/10 der Rechenzeit eines Prozessorkerns zugeordnet.

Die Verwendung des Entwurfsmusters Remote-Procedure-Call macht es notwendig die 22 KI-Threads mit dem Hauptthread des Spiels zu synchronisieren. Dazu muss ein geeignetes Schutzverfahren verwendet werden. Hierzu wird in der „RobotControl“ eine CriticalSection verwendet, die den Datenaustausch mit dem Hauptthread synchronisiert. Das ist notwendig, um einen sicheren Funktionsaufruf aus den Skripten zu ermöglichen. Ohne Schutzmechanismen könnten Daten unkontrolliert verändert werden, da von mehreren Threads gleichzeitig auf gemeinsamen Speicher zugegriffen wird.

Auf der anderen Seite der Skriptschnittstelle ist die Skriptsprache LUA in die Engine eingebunden. Die Gründe für die Verwendung für LUA als Skriptsprache finden sich im Kapitel Scripting-Ansatz. Hauptsächlich sind das der geringe Implementierungsaufwand, die Kompaktheit der Bibliothek und die MIT-Lizenz.
Die Einbindung der Skriptbibliothek findet im „LUAScriptWorker“, der vom allgemeinen „ScriptWorker“ abgeleitet ist, statt. Die Architektur ermöglicht es hier durch andere Ableitungen der Klasse „ScriptWorker“ andere Skriptsprachen zu integrieren. Denkbar wären hier mächtigere Sprachen wie JavaScript oder Python.

Über den „LUAScriptWorker“ werden die Funktionen des Spielers zur Verwendung in einem LUA-Skript gemappt. Der Austausch von Parametern wird über einen Stack realisiert. In Tabelle 1 sind alle Funktionen mit ihren Rück- und Übergabeparametern aufgelistet. In der Skriptsprache LUA ist es möglich, dass eine Funktion mehrere Parameter zurückgibt. Die Datentypen in der Tabelle sind nicht zwangsweise notwendig, da LUA keine explizite Angabe von Datentypen verwendet. Der Übersichtlichkeit halber sind sie hier mit angegeben.

Funktion	Übergabe parameter	Rückgabep arameter	Beschreibung
MoveForward	Double: Way	-	Vorwärtsbewegen des Spielers
MoveTo	Double: PosX, PosY, Speed	-	Spieler an eine bestimmte Position bewegen
Rotate	Double: Angle	-	Spieler um seine Achse drehen
Kick	Double: DirX, DirY, DirZ, Force	-	Den Ball in eine Richtung kicken
StopBall	-	-	Den Ball stoppen
Speak	String: Message	-	Mit anderen Spielern kommunizieren
Listen	-	String: Message	Hören, ob ein anderer Spieler kommuniziert
AllowedToRun	-	Boolean: Active	Gibt zurück, ob die künstliche Intelligenz ausgeführt werden darf
TimePlayed	-	Double: TimePlayed	Gibt die aktuelle Spielzeit in Prozent zurück
IsKickoff	-	Boolean: Kickoff	Gibt zurück, ob ein Anstoß stattfindet
GetTeamId	-	Int: Team-Id	Gibt die Team-Id des Spielers zurück
HasBall	-	Boolean: HasBall	Gibt zurück, ob der Spieler am Ball ist
GetStamina	-	Double: Stamina	Gibt die aktuelle Ausdauer des Spielers zurück

GetOwnLocation	-	Double: PosX, PosY, PosZ	Gibt die aktuelle Position des Spielers zurück
GetBallLocation	-	Double: PosX, PosY, PosZ	Gibt die aktuelle Ballposition zurück, wenn sichtbar
GetVisiblePlayers	-	Array[Int Team-Id; Int Player-Id; Double PosX; Double PosY]	Gibt Positionen der sichtbaren Spieler zurück
GetGoal1Position	-	Double: PosX, PosY, PosZ	Gibt die Position des 1. Tors zurück
GetGoal2Position	-	Double: PosX, PosY, PosZ	Gibt die Position des 2. Tors zurück

Tabelle 1: Funktionen im LUA-Skript

Alle Funktionen, die über das LUA-Skript aufgerufen werden, rufen Funktionen des implementierten Spielers auf. Im Weiteren folgt eine kurze Beschreibung der Funktionen mit Hinweisen zur Verwendung der Übergabe- und Rückgabeparameter aus dem LUA-Skript.

Über die Funktion „MoveForward“ wird der Spieler um eine angegebene Strecke vorwärts bewegt.
Die Funktion „MoveTo“ bewegt den Spieler mit einer angegebenen Geschwindigkeit an eine angegebene Position.

Mit „Rotate“ wird der Spieler in eine angegebene Richtung gedreht. Der Drehwinkel muss als absoluter Winkel angegeben werden. Eine Drehung relativ zur letzten Rotation ist nicht möglich. Der Rotationswinkel ist in Grad anzugeben und für 360 Grad für eine komplette Drehung startend bei 0 in positiver X-Richtung definiert.

Die Funktion „Kick“ lässt den Spieler den Ball in eine angegebene Richtung mit angegebener Kraft kicken. Die Kraft wird im Bereich von 0 bis 100 Prozent angegeben. Der Richtungsvektor ist in den definierten Achsenrichtungen des globalen Koordinatensystems anzugeben. Die Funktion kann nur aufgerufen werden, wenn der Spieler in Ballbesitz ist.

Über die Funktion „StopBall“ nimmt der Spieler den Ball an, wenn dieser in der Nähe ist.

Die Funktion „Speak“ lässt den Spieler eine Nachricht an alle anderen Spieler schicken. Über die Funktion „Listen“ wird der Erhalt einer solchen Nachricht abgefragt.

Die Funktion „AllowedToRun“ gibt wahr zurück, sobald das Spiel bereit ist und das LUA-Skript ausgeführt werden kann.

Über die Funktion „TimePlayed“ wird die aktuelle Spielzeit in Prozent abgefragt.

Mit der Funktion „IsKickoff“ muss geprüft werden, ob ein Anstoß stattfindet. Falls das der Fall ist, muss die künstliche Intelligenz darauf reagieren.

Die Funktion „GetTeamId“ gibt die Id des Teams zurück, zu dem der Spieler gehört.

Über „HasBall“ kann geprüft werden, ob der Spieler in Ballbesitz ist.

Die Funktion „GetStamina“ liefert den aktuellen Ausdauerwert des Spielers in Prozent zurück.

Mit der Funktion „GetOwnLocation“ wird die aktuelle Position des Spielers abgefragt. Die Rückgabeparameter sind die Spielerposition in X-, Y- und Z-Richtung des globalen Koordinatensystems.

Die Funktion „GetBallLocation“ gibt die Position des Balls in X-, Y- und Z-Richtung zurück, falls dieser vom Spieler gesehen wird. Anderenfalls sind alle Rückgabewerte 0.

Die Funktion „GetVisiblePlayers“ liefert ein Array der vom Spieler sichtbaren Mitspieler zurück. Für jeden gesehenen Spieler gibt es eine Zeile in diesem Array.

Diese besteht aus der Team-Id des Spielers, der Player-Id innerhalb des Teams und der Position in X- und Y-Richtung.
Die Funktionen „GetGoal1Position“ und „GetGoal2Position“ geben die Positionen der beiden Tore auf dem Spielfeld zurück.

Die aufgeführten Funktionen stehen im LUA-Skript zur Verfügung und können zur Entwicklung einer künstlichen Intelligenz verwendet werden. Diese wird im nachfolgenden Kapitel beschrieben.

5.2 Entwicklung der künstlichen Intelligenz

Künstliche Intelligenz beschäftigt sich mit der Nachahmung und Simulation menschlicher, kognitiver Fähigkeiten. Das Ziel ist die effiziente Nachbildung von intelligenter Leistung und menschlichem Verhalten in Computerprogrammen (vgl. [14, p. 17f.]).
Im speziellen Fall hier geht es darum, die Intelligenz und das Verhalten eines Fußballspielers bzw. einer ganzen Fußballmannschaft nachzubilden. Dabei sind Interaktionen mit dem Ball und anderen Spielern ein wichtiger Bestandteil. Andere Spieler teilen sich in gegnerische Spieler und eigene Teammitglieder auf. Es ist notwendig, dass jeder Spieler eine individuelle Strategie verfolgt. Diese Strategie muss abhängig von der Spielposition und dem aktuellen Zustand des Spiels sein. Ziel ist es dass sich die individuellen Strategien der einzelnen Spieler zu einer Strategie für die Mannschaft ergänzen. Da Fußball ein Mannschaftssport ist, ist diese Zusammenarbeit der einzelnen Spieler ein zentraler Punkt für die Entwicklung der künstlichen Intelligenz.

Eine Simulation der Fußballmannschaft kann daher als ein verteiltes, wissensverarbeitendes System angesehen werden. Ein verteiltes System ist dadurch charakterisiert, dass mehrere unabhängige Prozesse eine gemeinsame Aufgabe erfüllen. Dabei besitzen die Prozesse keinen gemeinsamen Speicher, sondern kommunizieren über Nachrichten. Zudem wird oft die Forderung gestellt, dass ein verteiltes System für den Benutzer nicht als solches erkennbar sein darf. Das

bedeutet, der Benutzer sieht nur ein einzelnes System und kann die individuellen Systeme nicht erkennen (vgl. [15, p. 19]). Die Umsetzung einer Fußballmannschaft in künstlicher Intelligenz erfüllt im Fall dieser Studienarbeit die allgemeine Definition. Jeder Spieler wird durch einen eigenständigen Thread repräsentiert. Die Kommunikation mit der Simulationsumgebung erfolgt über entfernte Funktionsaufrufe. Eine Kommunikation der Spieler untereinander ist ebenfalls über entfernte Funktionsaufrufe über die Simulationsumgebung möglich. Es besteht also kein gemeinsamer Speicher. Dass die einzelnen Prozesse ein gemeinsames Ziel verfolgen liegt bei einer Fußballmannschaft auf der Hand. Es geht darum als Team möglichst effektiv zu spielen um das Spiel zu gewinnen. Die Forderung nach der Erkennbarkeit der einzelnen Prozesse ist schwer zu beantworten. Auf der einen Seite ist jeder Spieler klar als Individuum mit individuellem Verhalten zu erkennen. Auf der anderen Seite ist bei korrekter Implementierung der künstlichen Intelligenzen das Verhalten als Mannschaft deutlich zu sehen.

Nach der Betrachtung als verteiltes, wissensverarbeitendes System, muss ein geeigneter Ansatz gefunden werden um dieses umzusetzen. Im Gebiet der computerbasierten Wissensverarbeitung stehen verschiedene Ansätze zur Entwicklung von intelligentem Verhalten zur Verfügung.

Es kann grundsätzlich zwischen expliziter und impliziter Wissensverarbeitung unterschieden werden. Bei der expliziten Wissensverarbeitung müssen konkrete Regeln definiert werden. Diese geben dem System ein Entscheidungsverhalten vor. Die Eingaben werden mit bereits bekannten Fakten und Regeln verarbeitet und so neue Erkenntnisse generiert.

Im Gegensatz dazu kommen implizite Ansätze ohne eine explizite Definition von Regeln aus. Sie sind in der Lage aus Beispielen selbst Regeln zu generieren. Sie benötigen allerdings Beispiele aus denen sie die Regeln erlernen können. Daher eignen sich solche Verfahren besonders gut zur Klassifikation (vgl. [16, p. 6f.]).

Der wichtigste Vertreter für einen solchen Ansatz ist das neuronale Netz. Neuronale Netze bilden die Neuronenstruktur des menschlichen Gehirns nach. Es werden die Neuronen mit ihren Verbindungen, den Synapsen, simuliert (vgl. [17, p. 3ff.]). Neuronale Netze und andere implizite Ansätze werden in diesem Dokument nicht

weiter betrachtet, da sie für diese Studienarbeit zu weit führen würden. Außerdem ist die Umsetzung der künstlichen Intelligenz eines Fußballspielers auf diese Weise nicht trivial. Es muss zunächst die Frage nach dem Erlernen der Regeln bzw. dem Spielverhalten geklärt werden.

Im Weiteren spielen daher nur noch explizite, wissensverarbeitende Systeme eine Rolle. Ansätze die auf reiner Logik basieren sind wegen der Implementierung problematisch. Durch die implementierte Skriptschnittstelle wird mit LUA-Skript eine prozedurale Sprache verwendet. Darin lassen sich logische Aussagen und Regeln nur schwer formulieren. Deklarative Sprachen wie Prolog oder Lisp sind zum Umsetzen von Aussagenlogik deutlich besser geeignet. Das gilt auch für die Prädikatenlogik.
In dieser Studienarbeit wurde mit einfachen Entscheidungsbäumen gearbeitet. Die Entscheidungen basieren auf Daten, die die Spieler von der Simulationsumgebung erhalten. Die Spieler sind zusätzlich in der Lage selbst generierte Daten zu speichern und für Entscheidungen zu nutzen.
Wie im Kapitel Skriptschnittstelle bereits beschrieben, stehen für die Programmierung der künstlichen Intelligenz wichtige Informationen über den aktuellen Spielstatus zur Verfügung. In folgender Tabelle ist eine Übersicht über die von der Skriptschnittstelle angebotenen Funktionen zu sehen. Eine ausführliche Übersicht mit Übergabeparametern und Verwendungshinweisen findet sich im Kapitel Skriptschnittstelle.

Funktion	Beschreibung
AllowedToRun	Gibt zurück, ob die künstliche Intelligenz ausgeführt werden darf
TimePlayed	Gibt die aktuelle Spielzeit in Prozent zurück
IsKickoff	Gibt zurück, ob ein Anstoß stattfindet
GetTeamId	Gibt die Team-Id des Spielers zurück
HasBall	Gibt zurück, ob der Spieler am Ball ist
GetStamina	Gibt die aktuelle Ausdauer des Spielers zurück
GetOwnLocation	Gibt die aktuelle Position des Spielers zurück
GetBallLocation	Gibt die aktuelle Ballposition zurück, wenn sichtbar
GetVisiblePlayers	Gibt Positionen der sichtbaren Spieler zurück
GetGoal1Position	Gibt die Position des 1. Tors zurück
GetGoal2Position	Gibt die Position des 2. Tors zurück

Tabelle 2: Passive Funktionen der Skriptschnittstelle

Es ist zu sehen, dass der künstlichen Intelligenz Informationen über das Spiel selbst in Form der aktuellen Spielzeit oder der Information über den Anstoß bereitgestellt werden. Zusätzlich werden auch noch statische Daten wie die Position der Tore angegeben.

Besonders wichtig um Entscheidungen treffen zu können sind die Daten über den Spieler selbst. Dazu gehören seine Teamzugehörigkeit, sein aktueller Ausdauerwert und seine aktuelle Position auf dem Spielfeld. Außerdem werden noch Informationen über vom Spieler gesehene Objekte bereitgestellt. Das sind der Ball und die anderen Spieler. Diese Informationen stehen nur zur Verfügung, wenn sich die Objekte im Sichtfeld des Spielers befinden.

Auf Basis dieser Informationen kann die künstliche Intelligenz weitere Daten wie die Aufstellung der Teammitglieder oder die Bewegungsrichtung des Balls ableiten. Damit ist es möglich auch komplexere Spielentscheidungen zu realisieren. Einige Beispiele für die Umsetzung eines solchen Entscheidungsbaums folgen später in diesem Kapitel.

Abgesehen von den Informationen bietet die Skriptschnittstelle natürlich auch die Möglichkeit Aktionen des Spielers über die künstliche Intelligenz zu steuern. In der nachfolgenden Tabelle sind die zu Verfügung stehenden Funktionen aufgelistet. Die ausführliche Übersicht ist wieder im Kapitel Skriptschnittstelle zu finden.

Funktion	Beschreibung
MoveForward	Vorwärtsbewegen des Spielers
MoveTo	Spieler an eine bestimmte Position bewegen
Rotate	Spieler um seine Achse drehen
Kick	Den Ball in eine Richtung kicken
StopBall	Den Ball stoppen
Speak	Mit anderen Spielern kommunizieren
Listen	Hören, ob ein anderer Spieler kommuniziert

Tabelle 3: Aktive Funktionen der Skriptschnittstelle

Über die integrierte Schnittstelle kann die künstliche Intelligenz Bewegungen des Spielers auslösen. So ist es möglich den Spieler geradeaus oder auch zu einer bestimmten Position zu bewegen. Ebenso ist es möglich den Spieler zu rotieren.
Zudem hat die künstliche Intelligenz die Möglichkeit den Spieler mit dem Ball interagieren zu lassen. Wenn der Ball in der Nähe ist, kann der Ball angenommen und in eine bestimmte Richtung gespielt werden.
Auch die Kommunikation zwischen den Spielern kann gesteuert werden. Der Spieler kann entweder Nachrichten an andere Spieler weitergeben oder aktiv auf Nachrichten anderer Spieler hören. Der Inhalt der Nachrichten kann dabei frei gewählt werden und zum Beispiel taktische Anweisungen enthalten. Nachrichten können von allen Spielern gehört werden. Über diese Funktionen ist die künstliche Intelligenz eines Spielers in der Lage mit der künstlichen Intelligenz eines anderen Spielers zusammenzuarbeiten.

Mit den oben vorgestellten Funktionen ist es möglich Entscheidungsbäume aufzubauen. Es kann sowohl auf Daten über den Spieler, als auch auf allgemeine Daten über den Spielzustand zugegriffen werden. Durch die Verknüpfung dieser Daten kann entschieden werden welche Aktion des Spielers ausgelöst werden sollte.
Der Vorteil eines Entscheidungsbaums liegt darin, dass er einfach verständlich und nachvollziehbar ist. Somit lässt er sich auch verhältnismäßig leicht formulieren. Dieser Vorteil relativiert sich allerdings durch die Tatsache, dass auch Entscheidungsbäume bei steigender Komplexität an Übersichtlichkeit verlieren. Vor allem dadurch, dass hier nicht nur binäre Entscheidungsbäume eingesetzt werden, lassen unüberschaubare, stark verzweigte Bäume zu. Binäre Entscheidungsbäume verzweigen sich an jedem Knoten in genau zwei Richtungen. Dadurch entstehen in der Regel schmale, tiefe Bäume.
Die hier für die Entscheidungen herangezogenen Informationen lassen sich oft nicht binär darstellen. Ein wichtiges Beispiel dafür ist die aktuelle Position des Spielers.

Im Folgenden wird beispielhaft ein Teil eines Entscheidungsbaums vorgestellt, der feststellt, wie der Spieler mit dem Ball interagieren soll. Abbildung 2 zeigt den dazugehörigen Codeausschnitt.
Zunächst wird geprüft, ob der Ball in einer bestimmten Entfernung zum Spieler ist. Der Spieler soll daraufhin den Ball annehmen. Danach wird für jedes Teammitglied in Sichtweite geprüft, ob es näher am gegnerischen Tor steht als der Spieler. Falls es einen Spieler gibt, der näher am generischen Tor befindet, wird die Position des am nächsten zum Tor gelegenen Spielers gespeichert. Diese Position wird für einen späteren Pass benötigt. Die Überlegung dabei ist, dass ein Spieler, der schon näher am gegnerischen Tor steht, nicht mehr so weit laufen muss mit dem Ball. Dabei sinkt das Risiko den Ball zu verlieren und somit steigt die Chance ein Tor zu erzielen.
Wenn ermittelt wird, dass der Spieler bereits derjenige ist, der am nächsten am Tor ist, wird geprüft, ob ein gegnerischer Spieler in unmittelbarer Nähe befindet. Falls das der Fall ist, wird der nächstgelegene Mitspieler für einen Pass gesucht. Dieser hat vermutlich größere Chancen mit dem Ball zum gegnerischen Tor zu kommen als der hier gesteuerte Spieler.

Wenn die obigen Prüfungen ergeben haben, dass der Spieler abspielen soll, wird noch geprüft wie der aktuelle Ausdauerwert des Spielers ist. Ist dieser hoch genug, passt der Spieler den Ball zu seinem Mannschaftskameraden.
Wenn der Spieler nicht abspielen soll, prüft er seine Distanz zum Tor und wagt daraufhin einen Torschuss oder spielt den Ball leicht Richtung Tor um dann nachzulaufen.

```
if(math.abs(ballX-x)<150 and math.abs(ballY-y)<150) then
    StopBall()
    ownAbs = math.pow(goal1_x-x, 2)+math.pow(goal1_y-y, 2) - 300
    pass = false
    pass_x = 0
    pass_y = 0
    for i = 1, #player, 1 do
        if (player[i][1] == 1) then
            playerAbs = math.pow(goal1_x-player[i][3], 2)+math.pow(goal1_y -player[i][4], 2)
            if (playerAbs < ownAbs) then
               ownAbs = playerAbs
               pass_x = player[i][3] - x
               pass_y = player[i][4] - y
               pass = true
            end
        end
    end
    --Gegner in der nähe?
    if (pass == false) then
        for i = 1, #player, 1 do
            if (player[i][1] == 2) then
                enemyAbs = math.sqrt(math.pow(x-player[i][3], 2)+math.pow(x -player[i][4], 2))
                if (enemyAbs < 1000) then
                   pass = true
                   for j = 1, #player, 1 do
                       if (player[j][1] == 1) then
                           pass_x = player[j][3] - x
                           pass_y = player[j][4] - y
                           break
                       end
                   end
                   break
                end
            end
        end
    end
    if (pass) then
        if (stamina > 50) then
            Kick(pass_x, pass_y,0,100)
        end
    else
        if (ownAbs < 5000000) then
            Kick(goal1_x-x, goal1_y-y,1200,100)
        else
            Kick(goal1_x-x, goal1_y-y,0,10)
        end
    end
end
```

Abbildung 10: Beispiel einer künstlichen Intelligenz

Der oben beschriebene Entscheidungsbaum ist verhältnismäßig klein und übersichtlich. Außerdem arbeitet er nur eine einzelne Situation ab. Um die gesamt Intelligenz des Spielers zu beschreiben sind deutlich mehr Prüfungen zu machen und

mehr Informationen zu berücksichtigt. Der oben gezeigt Ausschnitt ist deshalb nur als Beispiel für einen möglichen Entscheidungsbaum zu sehen und soll die grundlegenden Möglichkeiten demonstrieren. Dieses Beispiel berücksichtigt zwar schon Interaktionen zwischen den Spielern, es kann aber noch nicht von einer gemeinsamen Strategie oder gar von einer verteilten künstlichen Intelligenz gesprochen werden.

Die Programmierung ist nicht auf die Erstellung von Entscheidungsbäumen beschränkt. Wie zu Beginn des Kapitels beschrieben gibt es verschiedene andere Formen der künstlichen Intelligenz, die zwar im Rahmen dieser Studienarbeit als ungeeignet eingestuft wurden, aber bei Arbeiten mit dem Schwerpunkt auf die Entwicklung einer künstlichen Intelligenz durchaus interessant sind.

5.3 Grafischer Editor

Durch die Teilnahme am Microsoft Imagine Cup eröffnete sich die Möglichkeit, das Projekt auch von einer anderen Seite zu betrachten.

So bot es sich an, vermehrt auf Kundenbedürfnisse einzugehen, so zum Beispiel die einfache Verwendbarkeit des Produkts.

In diesem Zuge wurde ein Prototyp eines graphischen Editors entwickelt, der auch Programmieranfängern einen Einstieg in die Welt der KI bietet. Das Ziel dabei war, eine weitere Abstraktionsebene über die Skripte zu stellen.

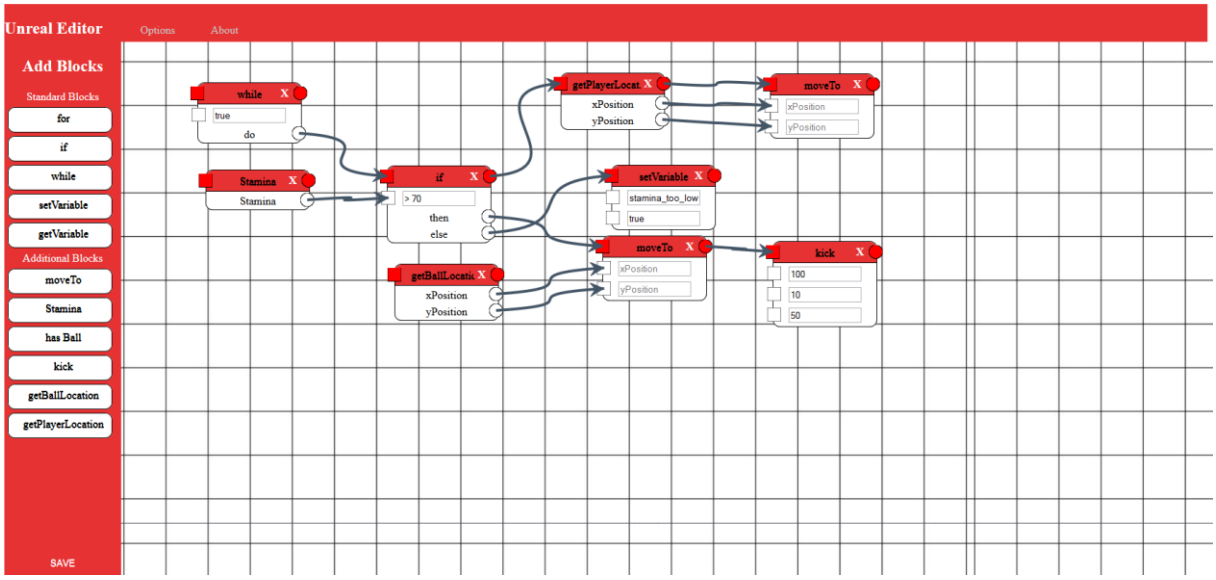

Abbildung 11: Grafischer Editor

In Abbildung 11 ist der Stand des Prototyps zu sehen. Der Editor bietet eine Reihe von vorgefertigten Funktionen, mit denen unerfahrene Anwender einen einfachen Einstieg bekommen können.

Diese Funktionen werden vom Anwender per Drag and Drop in die gewünschte Position gebracht und verbunden.
Im direkten Vergleich zur direkten Programmierung von Skripten ist der Editor zwar einfacher zu bedienen, bietet dafür allerdings weniger Möglichkeiten.

Der Prototyp bietet derzeit lediglich eine Oberfläche, das bedeutet, dass die Auswertung des erstellten Ablaufs noch nicht funktionsfähig ist.

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg
Campus Friedrichshafen

6 Ausblick

Im folgenden Kapitel wird aufgezeigt, welche Erweiterungsmöglichkeiten es für zukünftige Studienarbeiten gibt, welche sich mit dem Thema UnrealCup beschäftigen.

Der Stand zum Ende der Bearbeitungszeit der Studienarbeit umfasst alle Grundfunktionalitäten der Anwendung, welche für ein Spiel nötig sind. Diese Version besitzt eine sehr rudimentäre Implementierung der Fußballregeln, welche sich im Grunde auf einen simplen Anstoß beläuft. Dies bietet Spielraum für weitere Regeln: Die unsichtbaren Wände, welche das Spielfeld umgeben, können entfernt werden und eine “Aus-Regelung“ inklusive Einwurf könnte implementiert werden. Um den Realismus der Simulation zu erweitern können auch komplexere Regeln wie zum Beispiel Abseits implementiert werden. Mit der Einführung eines dynamischen Bestrafungssystems kann die Funktionalität der Spieler um Aktionen wie grätschen oder Tacklings erweitert werden.

Da bei der Entwicklung des UnrealCups die Funktionalität der Anwendung im Vordergrund stand ist die Optik noch eher zweckgemäß. Die verwendete Unreal Engine 4 ist eines der neusten und fortschrittlichsten Produkte im Bereich der Spiele Engines und bietet daher die Möglichkeit aufwendige Effekte, Animationen und detaillierte Umgebungen darzustellen. So könnte das Aussehen der Spieler individuell gestaltet werden, jeweils mit aufwendiger Simulation der Haare oder realistische Bewegungen seines Trikots. Sogenannte Shadereffekte können die Beleuchtung und die allgemeine Lichtstimmung drastisch beeinflussen und so ein realistischeres Spielgefühl vermitteln. Mit dem Ausbau der Umgebung auf ein vollwertiges Fußballstadium können in diesem Zuge die statische Lichtquelle durch dynamische, wie z.B. die Beleuchtung des Stadiums, ersetzt werden. Dies sorgt dafür, dass die Lichtverhältnisse realistisch nach dem tatsächlichen Streuradius der

einzelnen Lichtquellen berechnet werden und somit z.B. auch mehrerer Schatten geworfen werden.
Ein weiterer wichtiger Punkt bei einer möglichst glaubhaften Simulation sind die Bewegungen der einzelnen Akteure, in unserem Fall die Spieler. Diese besitzen aktuell hauptsächlich Laufanimationen. Weitere Animationen für die einzelnen Aktionen der Spieler sind möglich und würden z.B. die Effizient des Torwarts durch die Möglichkeit von komplexen Sprungaktionen deutlich erhöhen.

Ein Punkt, in dem der UnrealCup erweitert werden könnte ist die sogenannte Usability. Im Moment findet die komplette Konfiguration des Spiels außerhalb in XML-Dateien statt. Ein Teil dieser Konfiguration könnte in einem Menü abgebildet werden. Dieses Menü könnte z.B. eine Auswahl der Teams beinhalten und Möglichkeiten den Spielfluss zu kontrollieren.

Durch die Verwendete Architektur ist es möglich, neue Schnittstellen zu anderen Skriptsprachen zu implementieren. Beispiele wären hier Python oder sogar JavaScript. Dadurch kann eine größere Zielgruppe erreicht werden, da die Hürde des Lernens einer neuen Skriptsprache entfällt.

Die mit dem Spiel eingereichten KIs haben einen nur sehr eingeschränkten Umfang. So spielen z.B. alle Spieler einer Mannschaft mit der gleichen Logik. Es wäre möglich die einzelnen Spieler mit unterschiedlichen KIs für die jeweiligen Positionen auszustatten. Dadurch können Rollen wie Torwart, Mittelfeldspieler oder Stürmer ermöglicht werden

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg
Campus Friedrichshafen

7 Fazit

7.1 Bewertung der Ergebnisse

Nachfolgend wird betrachtet, inwieweit diese Studienarbeit ihre Ziele erfüllt. Es soll bewertet werden, ob die Arbeit als Erfolg gesehen werden kann oder ob noch zu viele Punkte der Aufgabenstellung nicht bearbeitet sind.

Alle in Kapitel 4.2.1 gegebenen Muss-Kriterien sind erfüllt:
Die Simulation des Fußballspiels umfasst alle gegeben Anforderungen. Die physikalische Korrektheit wird über die Verwendung der „Unreal Engine 4“ sichergestellt. Diese beinhaltet eine Physik Engine, die die Simulation von z.B. Kräften und Bewegungen übernimmt.
Auch die Fußballregeln werden innerhalb der Simulation eingehalten. An dieser Stelle muss allerdings angemerkt werden, dass der Regelumfang nicht dem der offiziellen FIFA Regeln entspricht. Die implementierten Regeln beschränken sich auf ein kleines Set. Hierzu gehören Torerkennung und anschließender Anstoß. Regeln wie Eckball oder Einwurf sind durch Abprallen des Balls an Banden umgangen. Dies entspricht den Regeln des Hallenfußballs.
Ein Ausdauersystem, dass die Leistungsfähigkeit der Spieler limitiert ist in der Simulationsumgebung integriert. Ebenfalls wird verhindert, dass einzelne KI Skripte die gesamte Rechenzeit für sich beanspruchen können.
Über die Verwendung der „Unreal Engine 4“ wird die Kompatibilität mit sowohl Windows als auch Linux realisiert. Zusätzlich wurde bei der Programmierung darauf geachtet, dass keine plattformspezifischen Besonderheiten genutzt werden. Die Lauffähigkeit der Simulationsumgebung wurde auf beiden Plattformen getestet um diese sicherzustellen. Diese Tests wurden unter Windows und Linux bestanden.
Durch die integrierte Skriptschnittstelle können auch nachträglich KIs erstellt werden. Die KI Skripte werden momentan in der Skriptsprache LUA geschrieben und zur Laufzeit interpretiert. Ebenso kann die Team Konfiguration nachträglich über XML Dateien konfiguriert werden.

Im Rahmen der Arbeit ist eine kleinere künstliche Intelligenz für die Spieler entstanden. Diese ist in der Lage auf niedrigerem Niveau Fußball zu spielen. Die KI führt keinerlei komplexere Spielzüge aus und nutz keine Kommunikation zwischen den Spielern. Sie dient hauptsächlich zu Testzwecken und beweist, dass das Konzept funktioniert.
Alles in Allem wurden die Muss-Kriterien erreicht. Einige Ziele können weiter ausgebaut werden. Für den Rahmen dieser Studienarbeit ist der Entwicklungsstand allerdings durchaus ausreichend.

Die im Kapitel 4.2.2 aufgeführten Soll-Kriterien wurden zum Teil erfüllt:
Die 3D-Darstellung ist durch den Einsatz der „Unreal Engine 4“ realisiert. Zusätzlich ist das Spielfeld dreidimensional modelliert. Ebenso die Spieler und die Tore. Die „Unreal Engine 4“ lässt durch ihre Physiksimulation dreidimensionale Bewegungen zu, ohne die eine 3D-Darstellung wenig sinnvoll wäre.
Die Bestrafung der Spieler bei Nichteinhalten der Regeln ist in der Arbeit nicht umgesetzt. Hier wurden lediglich Ansätze geschaffen, die noch erweitert werden können. Bisher könnte eine Bestrafung sowieso nur nach Nichteinhaltung des Anstoßes folgen. Die Möglichkeit zu foulen ist im Spiel nicht implementiert.
Das Nichterreichen dieses Punktes ist für die Studienarbeit nicht dramatisch, da dieser Punkt nur eine mittlere Priorität in der Planung erhalten hat. Damit ist er nice-to-have, nicht aber essentiell.

Das im Kapitel 4.2.3 angegebene Kann-Kriterium eines grafischen Editors zur Erstellung von KI Skripten ist, wie im Kapitel 5.3 beschrieben, angeschnitten worden. Der entstandene Prototyp des Editors beweist die allgemeine Machbarkeit des Vorhabens. Durch die niedrige Priorisierung des Editors wurde er während der Studienarbeit nur beiläufig untersucht.

Insgesamt betrachtet erfüllt die Studienarbeit die Erwartungen und die gegebene Aufgabenstellung. Es kann nicht behauptet werden, dass die Erwartungen im großen Stil übertroffen wurden. Jedoch sind die als wichtig priorisierten Anforderungen solide

erfüllt. Hier ist vor allem die Simulationsumgebung mit der Skriptschnittstelle zu nennen. Diese wurde plattformunabhängig implementiert.
Es bestehen an einigen Stellen noch Möglichkeiten zur Erweiterung, da nicht alle niedriger priorisierten Punkte während der Studienarbeit umgesetzt werden konnten. Hier ist die Entwicklung einer komplexeren KI für ein interessanteres Fußballspiel und die Erweiterung der Fußballregeln zu nennen.
Alles in Allem darf die Studienarbeit als Erfolg gewertet werden. Es sind keine kritischen Punkte offen. Ebenso sind zum aktuellen Zeitpunkt keine Bugs mehr bekannt.

7.2 Persönliches Fazit

Der UnrealCup ist trotz der thematischen und auch namentlichen Nähe zum RoboCup schließlich ein eigenständiges Projekt mit völlig unterschiedlichen Zielen geworden. Besonders positiv ist die Verwendung modernster Technologien in Form der Unreal Engine zu erwähnen. Hierbei entstand ein erfrischender Gegensatz zu den sonst vermittelten Technologien, da diese im Laufe der Projektzeit wiederholt Updates erfahren hat und so die Entwicklung mitverfolgt werden konnte.
Ein kleines Highlight war die Teilnahme des Microsofts Imagine Cup's 2015 in der Kategorie Gaming. Hierbei sind wir mit Prof. Dr. Judt als Mentor angetreten und haben den dritten Platz im Deutschland Finale erreicht. Diese Teilnahme führte zwar zu Mehraufwand in Form von Dokumenten und Videos, welche aber teilweise in diese Dokumentation mit einflossen.

Des Weiteren war es möglich Vorlesungsinhalte im Laufe des Projektes anzuwenden: Dazu gehören die Vorlesungen im Bereich „Theoretische Informatik" von Prof. Fahr. Diese beinhalten die logischen Grundlagen, welche bei der Einbindung und Erstellung der künstlichen Intelligenz von Hilfe waren. Auch die Anwendung von Design Pattern aus der Vorlesung „Software Engineering" von Prof. Dr. Judt wurde berücksichtigt. Ebenso kamen Kenntnisse aus dem Bereich „Echtzeitsysteme" und „Betriebssysteme" zum Tragen. Diese waren notwendig um

das Scheduling der KIs zu realisieren. Da große Teile der Implementierung in C++ erfolgt sind, war die Vorlesung zu diesem Thema ebenfalls eine nützliche Grundlage. Die Vorlesungen „Agile Prozessmodelle“ und „Projektmanagement“ haben die Grundsteine für die Planung und Organisation des Teams geliefert.

Im Rückblick sind wir mit unserem Fortschritt zufrieden: Das Spiel ist lauffähig, bietet alle Basisfunktionalitäten und bietet noch viele Ansatzpunkte für zukünftige Entwicklungen.

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg
Campus Friedrichshafen

Literaturverzeichnis

[1] Valve, „Valve Technology,“ [Online]. Available: https://developer.valvesoftware.com/wiki/Valve_Technology. [Zugriff am 2 Juli 2015].

[2] Valve, „Valve Software,“ [Online]. Available: http://www.valvesoftware.com/news/. [Zugriff am 2 Juli 2015].

[3] Unity Technologies, „Unity,“ [Online]. Available: http://unity3d.com/unity. [Zugriff am 3 Juni 2015].

[4] Unity Technologies, „Unity,“ [Online]. Available: https://store.unity3d.com/products. [Zugriff am 3 Juni 2015].

[5] Epic Games, „Unreal Engine 4,“ [Online]. Available: https://www.unrealengine.com/what-is-unreal-engine-4. [Zugriff am 10 Juni 2015].

[6] Epic Games, „Unreal Engine 4 Games,“ Epic Games, [Online]. Available: https://wiki.unrealengine.com/Category:Games. [Zugriff am 07 07 2015].

[7] Epic Games, „Unreal Engine 4,“ [Online]. Available: https://www.unrealengine.com/blog/ue4-powered-tekken-7-announced-at-evo-2014. [Zugriff am 10 Juni 2015].

[8] Epic Games Inc, „Academic Use for Students and Teachers | Unreal Engine 4,“ Epic Games Inc, 2014. [Online]. Available: https://www.unrealengine.com/education. [Zugriff am 02 07 2015].

[9] Lua.org, „Lua 5.3 Reference Manual,“ 17 06 2015. [Online]. Available: http://www.lua.org/manual/5.3/manual.html. [Zugriff am 02 07 2015].

[10] Lua.org, „Lua: About,“ Lua.org, 17 06 2015. [Online]. Available: http://www.lua.org/about.html. [Zugriff am 08 07 2015].

[11] RoboCup Federation, „RoboCup Soccer Server Users Manual,“ RoboCup Federation, 2003.

[12] Lämmel, U., Cleve, J., Künstliche Intelligenz, München: Hanser, 2012.

[13] Mühlhäuser, M., Schill, A., Software Engineering für verteilte Anwendungen, Berlin Heidelberg: Springer, 1992.

[14] Norvig, R., Russel, S., Künstliche Intelligenz - Ein moderner Ansatz - 2. Auflage, München: Pearson Studium, 2004.

[15] Tanenbaum, A. , Steen, M., Verteilte Systeme - Prinzipien und Paradigmen, München: Pearson Studium, 2008.

[16] Scherer, „Wissensbasierte Systeme Semester 6 Skript," Karlsruhe, 2009.

[17] Kriesel, D., „Ein kleiner Überblick über Neuronale Netze," Bonn, 2005.

[18] ANSI, „Port70," [Online]. Available: http://port70.net/~nsz/c/c89/c89-draft.html. [Zugriff am 8 Juli 2015].

[19] P. D. R. Gillenkirch, „GABLER WIRTSCHAFTSLEXIKON," [Online]. Available: http://wirtschaftslexikon.gabler.de/Definition/entscheidungsbaum.html. [Zugriff am 8 Juli 2015].

[20] Epic Games, „Unreal Engine 4," [Online]. Available: https://docs.unrealengine.com/latest/INT/Engine/Blueprints/index.html. [Zugriff am 8 Juli 2015].

[21] Microsoft, „Imagine Cup," [Online]. Available: https://www.imaginecup.com. [Zugriff am 8 Juli 2015].

[22] OpenGL, „OpenGL Wiki," [Online]. Available: https://www.opengl.org/wiki/Shader. [Zugriff am 8 Juli 2015].

[23] IT Wissen, „IT Wissen," [Online]. Available: http://www.itwissen.info/definition/lexikon/Scriptsprache-script-language.html. [Zugriff am 8 Juli 2015].

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg
Campus Friedrichshafen

Abbildungsverzeichnis

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg
Campus Friedrichshafen

Tabellenverzeichnis

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg
Campus Friedrichshafen

Abkürzungsverzeichnis

3D	Drei Dimensional
CPU	Central processing unit
ID	Identifier
KI / AI	Künstliche Intelligenz
UDP	User Datagram Protocol
UE4	Unreal Engine 4
WYSIWYG	What You See Is What You Get
XML	Extensible Markup Language

Glossar

ANSI C	Der erste Standard(C89) der Programmiersprache C wurde von ANSI(American National Standards Institute) herausgegeben. Anschließend hat die ISO(International Organization for Standardization) das definieren der neuen Versionen übernommen. Allerdings hält sich bis heute der Begriff „ANSI C“, nur wenige bezeichnen es als „ISO C“. Zum Zeitpunkt dieser Arbeit ist die Version C11 aktuell. (vgl. [18])
Entscheidungsbäume	„Form der Darstellung mehrstufiger Entscheidungen. Der Entscheidungsbaum wird aus einer Erweiterung des Zustandsbaums gewonnen, indem in den einzelnen Zeitpunkten neben den erwarteten Umweltzuständen zusätzlich die verfügbaren Handlungsalternativen (Aktionen) einbezogen werden.“ [19]
Blender	3D Modellierungssoftware http://www.blender.org
Blueprint	„Blueprints Visual Scripting“ ist ein Teil der Unreal Engine 4. Mit dem grafischen Editor ist es möglich Spielverhalten zu gestalten ohne dabei eine Zeile Code schreiben zu müssen. Das erstellte Skript wird vor dem Ausführen auch in Maschinencode kompiliert und bietet daher keine Nachteile gegenüber der Implementierung in C++. (vgl. [20])
Game Engine	Eine Game Engine ist ein großes Framework welches die Entwicklung von Spielen erleichtern soll. Hierzu wird eine Grafik, Physik, Netzwerk, Sound und meist auch eine KI Engine kombiniert.
Microsofts Imagine Cup	„Imagine Cup ist sowohl globales Technologieprogramm für Studierende als auch Wettbewerb und bietet Studierenden in allen Disziplinen Möglichkeiten zur Bildung von Teams und zum Einsatz ihrer Kreativität, Leidenschaft und IT-Kenntnisse

zum Entwickeln von Anwendungen und Spielen sowie zum Integrieren von Lösungen, die die Art und Weise ändern können, in der wir leben, arbeiten und spielen.“ [21]

Shadereffekte

Unter Shadern versteht man in der 3D Grafik die Anwendung von Effekten während der Berechnung der Scene. Hierbei unterscheidet man zwischen fünf verschiedenen Typen. Als Beispiel sein hier zwei Arten genannt: Pixel(Fragment)- und Vertexshadern. Vertexshader verändern die Geometrie der Objekte. Ein Pixelshader hingegen arbeitet auf Basis eines Bildes(Textur eines Objektes oder das Ergebnis des Renderings) und wird für die einzelnen Pixel ausgeführt. (vgl. [22])

Skriptsprache

„Scriptsprachen sind Programmiersprachen, deren Ziel im Grundsatz es ist, zumeist nur kleinere Anwendungen oder Anweisungsfolgen zu realisieren. Dabei weisen die Codesequenzen - die häufig auch als Scripte oder Scripts bezeichnet werden - bestimmte spezifische Merkmale auf. Eines dieser Merkmale ist es, dass Scripte vielfach nicht von einem Compiler in maschinenlesbaren Code übersetzt werden, sondern zur Laufzeit von einem Interpreter ausgeführt werden. Scriptsprachen sind oft auch spezifischen Aufgaben zugeordnet, so dass deren Syntax leicht zu überschauen ist.“ [23]