Honourusarbeit

Zur Programmierung eines Autorennspiels unter Unity: Die Planung und Implementation einer künstlichen Intelligenz des Fahrens und der Wegfindung

Module Number:	SAE6302
Block Name:	Major Project
Date Submitted:	03.09.2018
Award Name:	Bachelor of Science (Hons.) Games Programming
Course:	GPR 0916
Name:	Daniel Stein
City:	Hamburg
Country:	Germany
Staffing:	Ina Arendt und Jan-Friedrich Conrad
Word Count:	6306
Weighting:	50% Theoretical / 50% Practical

Hiermit bestätige ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken (dazu zählen auch Internetquellen) entnommen sind, wurden unter Angabe der Quelle kenntlich gemacht.

Hamburg, den 27.03.2018 — Ort, Datum

Daniel Stein — Unterschrift Student

Information:

Das Inhaltsverzeichnis und einige Bilder der digitalen Version dieser Arbeit beinhalten Hyperlinks. Ein Klick auf das Inhaltsverzeichnis sorgt für einen Sprung direkt zum Kapitel. Die Bilder, die als Screenshots aus Videos aufgenommen wurden, sind mit dem eigentlichen Video verlinkt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Anhang - Inhaltsverzeichnis

1. Einleitung

1.1 Idee des Projektes

Im Verlauf dieser Arbeit soll eine künstliche Intelligenz (KI) für Rennspiele konzipiert und erstellt werden. Sie soll in der Lage sein, mehrere Fahrzeuge möglichst authentisch über eine vorgegebene Strecke zu steuern. Im Prozess der Umsetzung werden Pfadfindung, ein realistisches Fahrverhalten und intelligentes Handeln mit einbezogen. Außerdem wird die Arbeit in Kooperation mit einem anderen Studenten entstehen, der die Fahrphysik für das praktische Endprodukt erstellt.

1.2 Persönliche Motivation

Im Verlauf meines Studiums habe ich mich Schritt für Schritt auf das Gebiet der Programmierung von künstlichen Intelligenzen spezialisiert. Im Produktionsprozess möchte ich meine Fähigkeiten und mein Wissen in diesem Bereich weiter ausbauen. Deshalb werde ich tiefer gehende Aufgaben übernehmen. So beinhaltet der Aufbau einer KI für Rennspiele Elemente, die in meinen bisherigen Projekten von den Spieleengines gestellt wurden. Dazu gehören die angepasste Pfadfindung und die Verwendung von speziellen Bewegungsfunktionen.

1.3 Zielsetzung

Das Ziel des Projektes ist die Fertigstellung einer autonomen KI, welche ein Autorennen komplett eigenständig bestreiten kann. Im Endprodukt sollen verschiedene Situationen erzeugt werden können, sodass das Verhalten der KI ausgiebig getestet und betrachtet werden kann.

Ein zusätzliches Kann-Ziel ist die Einbindung einer Steuerung für menschliche Spieler. Dieses Feature ist bei der Programmierung eines Spieles zwar notwendig, allerdings fällt die Betrachtung einer KI von einer externen Position einfacher. Aus diesem Grund wird die Erstellung einer externen Betrachtungsweise priorisiert.

Die Umsetzung der KI kann in mehrere Einzelziele unterteilt werden:

1.3.1 Pfadfindung

Zunächst muss die Straße markiert, gespeichert und an die KI übergeben werden. Nur so ist sie in der Lage, einen vorgegebenen Weg abzufahren. Die Funktionsweise der Pfadfindung wird auf Rennspiele angepasst sein.

1.3.2 Fahren / Fahrverhalten

Das Fahrverhalten beinhaltet zunächst die grundsätzlichen Funktionen des Fahrens, wie zum Beispiel Beschleunigen, Bremsen und Lenken. Diese basieren auf der Fahrphysik und stellen damit den Schnittpunkt der Kooperation dar.

Weiterführend verwendet die KI die Funktionen, um die Strecke sicher und schnell entlangzufahren.

1.3.3 Interaktion mit anderen Fahrzeugen

Im finalen Spiel sollen immer mehrere Fahrzeuge auf der Strecke sein. Demzufolge muss die KI auf andere Fahrzeuge, die von anderen künstlichen Intelligenzen oder dem Spieler gesteuert werden, reagieren können.

1.4 Verortung in der Industrie

Das Konzept der Steuerung gegnerischer Fahrzeuge durch eine KI ist in beinahe jedem Rennspiel anzutreffen. Beinhaltet ein Rennspiel einen Singleplayer Modus oder das Auffüllen von Teilnehmern mit Computergegnern, ist immer eine künstliche Intelligenz vorhanden. Beispiele für unterschiedliche Vertreter des Genres sind die Spiele *„Mario Kart“* oder *„DriveClub“*.

Dementsprechend ist diese Arbeit für die Sparte der Rennspiele relevant und kann auch in einem industriereifen Produkt seine Anwendung finden.

2. Kontext

2.1 Referenzen

Rennspiele sind bereits seit Jahrzehnten ein Teil der Spieleindustrie. In der Vergangenheit und auch heutzutage basieren sie auf schnellen Spielweisen. Durch hohe Geschwindigkeiten und einer ständigen Konkurrenz versuchen sie den Spieler an das Geschehen zu fesseln. Dabei haben die Entwickler verschiedene Wege eingeschlagen. Das Genre „Rennspiel“ umfasst eine große Spanne an variierenden Unterkategorien. Zuerst sind da die Spiele, die ihren Fokus auf die pure Unterhaltung setzen. Es werden Gegenstände ins Spielgeschehen eingefügt, mit denen man Gegner ausschalten kann. Die Strecken, auf denen gefahren wird, sind möglichst ausgefallen. Realistische Physik steht eher im Hintergrund. Ein bekannter Vertreter dieser Kategorie ist *„Mario Kart“* (siehe Abbildung 1).

Abbildung 1: Mario Kart 8: Gameplay Screenshot

Im Gegensatz dazu haben es sich andere Spiele zur Aufgabe gemacht, die Realität perfekt widerzuspiegeln. Das Hauptaugenmerk liegt auf realistischer Grafik und Physik. Auch die Umwelt und die Rennstrecken werden der Realität nachempfunden. Das Spiel *„DriveClub“* spiegelt diese Kategorie genau wieder (siehe Abbildung 2).

Abbildung 2: DriveClub: Gameplay Screenshot

Des Weiteren gibt es die Spiele, bei denen Action ganz oben steht. Sie nehmen die Grafik und Physik der realistischen Spiele und machen das Spielgeschehen aufregender und spektakulärer. Explosionen, Stunts und Zeitlupenaufnahmen sind beliebte Stilmittel, die von Spielen wie *„Need for Speed“* verwendet werden (siehe Abbildung 3).

Abbildung 3: Need for Speed Payback: Gameplay Screenshot

Das Spiel, welches in dieser Arbeit entsteht, wird sich an dem Spiel *„DriveClub“* orientieren. Damit fällt es in die Sparte der realistischen Rennspiele.

2.2 Bezug auf aktuelle Diskussion

Der aktuelle Trend der Rennspiele wird von der immer weiter voranschreitenden Technik gesteuert. Leistungsfähige Konsolen und PCs ermöglichen immer realistischer und aufwendiger werdende Physik und Grafik. Dem Trend folgend, zielt das Projekt darauf ab, ein realistisches Rennspiel zu erzeugen. Das Hauptaugenmerk wird auf einem authentischen Verhalten der Gegner liegen. Die KI soll die Fahrzeuge wie bei einem echten Rennen steuern. Von dem Kooperationspartner wird angestrebt, die physikalischen Eigenschaften der verwendeten Fahrzeuge exakt nachzubilden.

2.3 Arbeitsumfeld

In der Spieleindustrie ist es bei der Entwicklung eines Rennspieles üblich, eine hauseigene Engine zu programmieren, die speziell auf das Genre zugeschnitten ist. Die Verwendung einer kostenlos zugänglichen Spieleengine ist eher unüblich. Dies ist daran zu erkennen, dass nur wenige professionell veröffentlichte Rennspiele mit kostenlosen Engines programmiert wurden. Eines der wenigen Spiele ist beispielsweise *„Moto Racer 4“* (vgl. astragon 2016).

Dieses Projekt wird jedoch eine freie Engine als Basis verwenden, da die Programmierung einer gesamten Engine einen zu großen Arbeits- und Zeitaufwand bedeuten würde. Zur Verfügung stehen *„Unity 3D“* und *„Unreal Engine 4“*. Für das Spiel dieser Arbeit wird *„Unity 3D“* verwendet, da in *„Unreal Engine 4“* die Physik nicht beliebig ersetzt werden kann.

2.4 Pfadfindung

Der erste Schritt für die Entwicklung der KI eines Rennspieles ist die Markierung und Übermittlung der Rennstrecke. Normalerweise wird in Spielen ein System verwendet, das den Weg zu einem Zielpunkt dynamisch in Echtzeit berechnet. Das hierfür meist genutzte System ist der *A* Algorithmus*. Der Algorithmus ist darauf ausgelegt, in offenen Welten einen Weg zwischen zwei beliebigen Punkten zu finden. Dabei bezieht er alle Arten von Hindernissen in seine Wegfindung ein.

Für die Umsetzung der Pfadfindung werden im Folgenden zwei Möglichkeiten, die auf Rennspiele zugeschnitten sind, vorgestellt.

2.4.1 Statisches Wegpunktsystem

Der erste Ansatz besteht in der Erstellung eines statischen Systems. Die Strecke und die zu fahrenden Wege werden also einmal bestimmt und danach nicht mehr geändert. Zunächst werden die Straßengrenzen und die zu fahrenden Wege mit Wegpunkten markiert. Dabei können mehrere Fahrlinien markiert werden. So werden eine Hauptfahrlinie und beliebig viele Überhollinien erzeugt. Die KI setzt sich nun zum Ziel den Linien so genau wie möglich zu folgen und fährt somit den vorgegebenen Weg ab (vgl. Gari 2002: 439-442; Tomlinson/Melder 2013: 482,487). Ein Video von Alex Brooks stellt die Umsetzung in einem zweidimensionalen System dar (siehe Abbildung 4) (vgl. Brooks 2011).

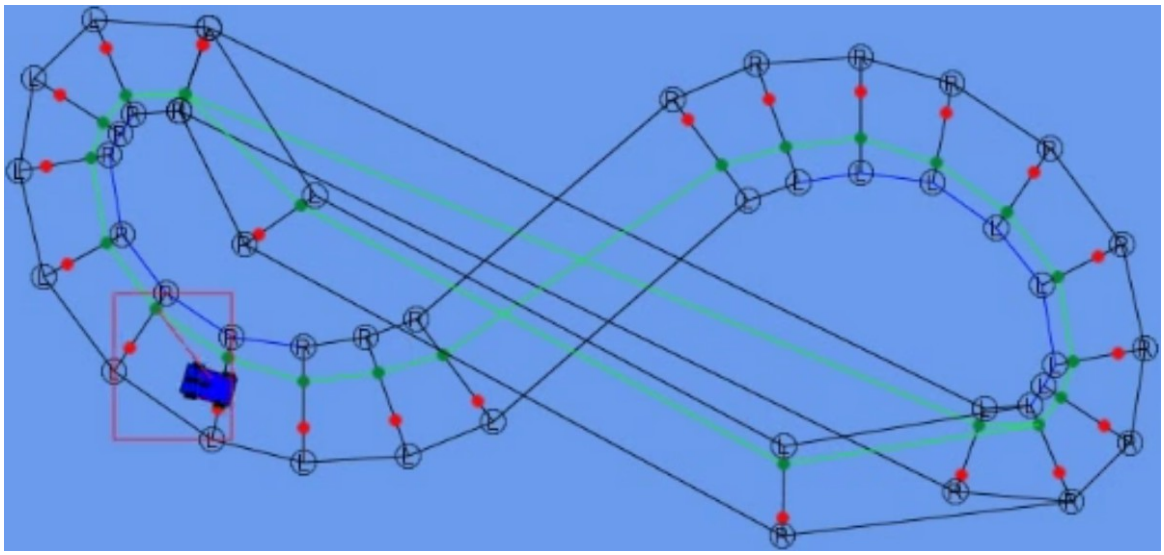

Abbildung 4: Pfadfindung durch ein statisches Wegpunktsystem

2.4.2 Anpassung des A* Algorithmus

Im Zentrum des zweiten Ansatzes steht die Anpassung des *A* Algorithmus* auf die Anforderungen eines Rennspieles. Zudem wird angestrebt, dem Level Designer so viel Arbeit wie möglich abzunehmen. Die erste Verwendungsmöglichkeit ist die Platzierung der Wegpunkte von dem Algorithmus übernehmen zu lassen. Hierbei wird die gesamte Fahrbahn in kleine Felder unterteilt. Bestimmt man nun ein Startfeld, testet der Algorithmus jeden möglichen Weg in Fahrtrichtung aus (siehe Abbildung 5 links). Nachdem er den effizientesten Weg gefunden hat, erzeugt er die benötigten Wegpunkte (siehe Abbildung 5 rechts).

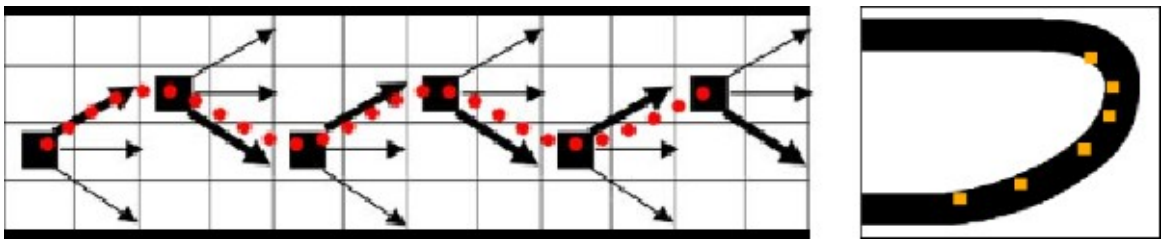

Abbildung 5: Pfadfindung durch den angepassten A Algorithmus*

Ein weiterer Schritt ist die Entwicklung einer Echtzeitpfadfindung. Verwendet man den angepassten A* Algorithmus in sehr kurzen Zeitabständen (ca. 0,1 Sekunden), kann der Weg auf aktuelle Gegebenheiten angepasst werden. Es ist also auch möglich, bewegliche Hindernisse einzubeziehen und einen Weg um sie herum zu planen. Im Falle eines Rennspieles können die beweglichen Hindernisse auch andere Fahrzeuge sein und die Anpassung des Fahrweges würde einen Überholvorgang zur Folge haben (vgl. Wang/Lin 2012: 13-18).

2.5 Künstliche Intelligenz

2.5.1 Endlicher Automat

Die künstliche Intelligenz ersetzt in einem Rennspiel andere Spieler. Um die Rolle authentisch erfüllen zu können, benötigt sie dieselben Fähigkeiten wie ein menschlicher Spieler. Dazu zählen zum einen Methoden, die die verschiedenen Aktionen eines Autorennens beinhalten. Zum anderen muss sie entscheiden können, welche der Funktionen sie in einer beliebigen Situation ausführt. Der Vorgang der Entscheidungsfindung findet bei dieser KI in einem *endlichen Automaten* statt. Der endliche Automat speichert jede einzelne Aktion und definiert die Bedingungen, die für einen Wechsel zwischen verschiedenen Aktionen erfüllt werden müssen (vgl. Tomlinson/Melder 2013: 473f).

Die verschiedenen Aktionen können auf unterschiedliche Weise und mit unterschiedlichem Umfang umgesetzt werden. Außerdem ist es möglich, zusätzliche Features einzubauen.

2.5.2 Wahrnehmung und Kollisionsvermeidung einer KI

Das Überholen anderer Fahrzeuge und das Ausweichen vor Hindernissen sind grundlegende Fähigkeiten, die eine KI beherrschen muss. Um diese Fähigkeiten jedoch zu ermöglichen, muss die KI andere Objekte zunächst wahrnehmen können. Für die Umsetzung ihrer Wahrnehmung existieren verschiedene Ansätze.

Raycasting

Die meist verwendete Methode ist die Nutzung eines *Raycastsystems*. Die KI schießt also in verschiedene Richtung *Raycasts* ab (siehe Abbildung 6). Wenn sie auf Objekte treffen, können Eigenschaften und Daten der Objekte abgerufen und gespeichert werden. So erhält die KI zum Beispiel die Position von Hindernissen oder die Geschwindigkeit anderer Fahrzeuge und kann ihre Aktionen entsprechend anpassen.

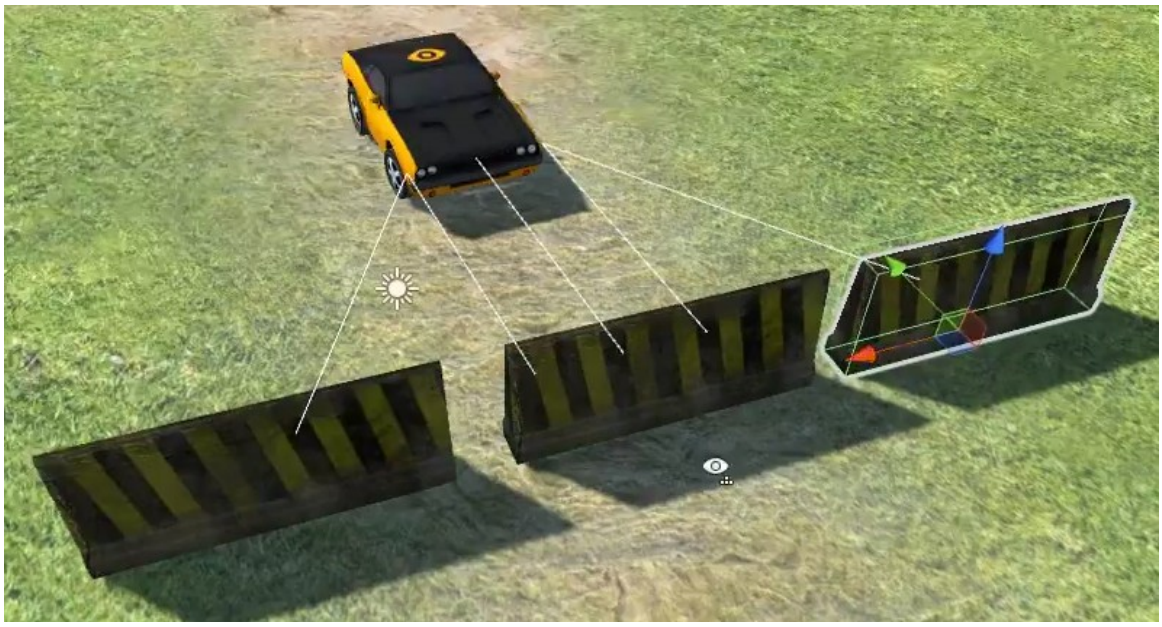

Abbildung 6: Raycasts treffen Hindernisse

Wärmebildsystem

Bei der Verwendung des *Wärmebildsystems* registriert die KI sämtliche Fahrzeuge, die sich vor ihr befinden. In Form einer *Hitzelinie* werden die Positionen und Entfernungen der Gegner gespeichert (siehe Abbildung 7 unten). Die *Hitzelinie* wird nach folgenden Regeln aufgebaut.

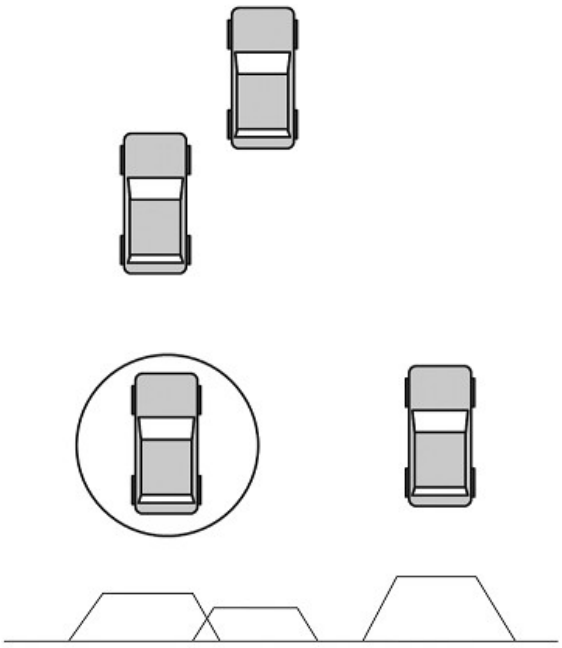

Abbildung 7: Hitzelinie für das eingekreiste Fahrzeug

Die *Hitzelinie* ist wie ein Graph anzusehen, dessen linkes und rechtes Ende der x-Achse auf der linken und rechten Begrenzung der Fahrbahn liegt. Die Werte zwischen den Enden des Graphen befinden sich auf den entsprechenden Positionen der Fahrbahn. Befindet sich ein Fahrzeug vor der KI, wird der Graph der *Hitzelinie* über die gesamte Breite des anderen Fahrzeuges erhöht. Je geringer der Vorsprung des anderen Fahrzeuges ist, desto höher steigt der Graph. Die so entstandene *Hitzelinie* zeigt an ihrem Tiefpunkt die Stelle an, an der die KI mit der höchsten Wahrscheinlichkeit Kollisionen vermeiden kann. Die KI hält sich folglich immer an der kältesten Stelle des Graphen auf (vgl. Tomlinson/Melder 2013: 501-505).

Künstliche Potenzialfelder

Die dritte mögliche Methode ist die Verwendung von künstlichen *Potenzialfeldern*. Bei dieser Vorgehensweise werden allen relevanten Objekten abstoßende und anziehende *Potenziale* zugeordnet. Fahrzeuge und Hindernisse erhalten abstoßende *Potenziale*, Wegpunkte erhalten anziehende *Potenziale*. Anschließend werden alle *Potenziale* auf einer Karte zusammengefügt (siehe Abbildung 8 links). Anhand der erstellten Karte berechnet die KI den Weg, der an allen Hindernissen vorbei führt (siehe Abbildung 8 rechts) (vgl. Salman 2013:14-18).

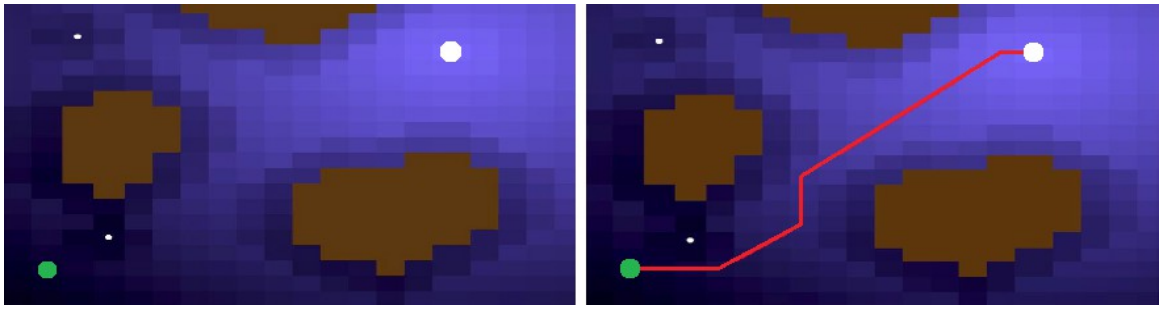

Abbildung 8: Die KI findet einen Weg durch die künstlichen Potenzialfelder (dunkel: abstoßende Potenziale, hell anziehende Potenziale)

2.5.3 Rubberbanding

In einem Rennspiel ist es notwendig, die Fähigkeiten des Spielers mit einzubeziehen. Ist der Spieler zu langsam oder zu schnell für die Gegner, verfällt schnell der Spielspaß und die Herausforderung. Um dem entgegenzuwirken, wird *Rubberbanding* verwendet. Die gegnerischen Fahrzeuge werden wie von einem Gummiband nah am Spieler gehalten, sodass er sich zu jeder Zeit im Spielgeschehen befindet. Um diesen Effekt zu erzeugen, werden die Fahrzeuge vor dem Spieler verlangsamt und hinter dem Spieler beschleunigt (vgl. Tomlinson/Melder 2013: 507).

3. Methodik

3.1 Arbeitsweise

3.1.1 Zielsetzung

Zu Beginn des Projektes fand mit dem Studenten, der die Physik programmiert, eine Bedingungsanalyse mit folgenden Fragen statt. Welche Art von Rennspiel sollte erstellt werden? Welche Engine sollte für die Umsetzung verwendet werden?

3.1.2 Recherche

In der Recherchephase werden Textanalysen verwendet, um Expertenmeinungen zur Umsetzung einer Rennspiel KI zusammenzutragen. Diese Expertenmeinungen liegen in Form von wissenschaftlichen Arbeiten und Literaturveröffentlichungen vor.

Die zusammengetragenen Umsetzungsmöglichkeiten werden auf ihr Verhältnis zwischen Aufwand und Ergebnis untersucht und anschließend untereinander verglichen. Auf diese Weise werden die Methoden ausgewählt, die im Projekt angewendet werden.

Die auf der Recherche basierende Umsetzung der ausgewählten Methoden wird in Kapitel 4 unter den Überschriften „Aufbau“ aufgeführt.

3.1.3 Finale Planung

Aufgrund der ausgewählten Methoden werden nun die finalen Meilensteine für das Projekt festgelegt. Außerdem werden die Schnittstellen zwischen KI und Physik definiert.

Zu diesem Zeitpunkt wird auch zum ersten Mal die iterative Arbeitsweise angewendet. Die finale Planung wird einem externen Sachkundigen ausführlich geschildert. Anschließend wird seine professionelle Meinung einbezogen und in die Planung integriert.

3.1.4 Umsetzung

In der Umsetzung haben die verwendeten Methoden zum Ziel, das Erfüllen der Meilensteine und das Erreichen des Projektabschlusses zu sichern.

Grundlegend werden sämtliche Meilensteine inkrementell abgehandelt. Die einzelnen Meilensteine werden jedoch iterativ bearbeitet.

Treten während der Arbeit Probleme auf, werden drei Behebungsverfahren angewendet.

Visuelle Analyse

Während des laufenden Spieles können Fehler auftreten, die sich in Form von unerwarteten Verhaltensweisen zeigen. Der Ablauf des Spieles erzeugt aber keine Programmfehler, mit denen die Spieleengine nicht umgehen kann. Fehler solcher Art stammen also nicht von fehlerhaftem Code, entstehen aber meist aus einem Denkfehler des Programmierers.

Zur Behebung dieser Fehler werden alle sichtbaren Abläufe analysiert und mit ihrem Gegenpart im Code verglichen. Auf diese Weise folgt man dem Spielgeschehen, bis die ungewollten Verhaltensweisen auftreten und ermittelt so die betroffene Stelle im Code.

Feedback-Methode

Reicht die visuelle Analyse für falsche Verhaltensweisen nicht aus, werden zwei Arten der Feedback-Methode verwendet, die ebenfalls bei Programmierfehlern angewandt werden.

Bei der ersten Art fügt man an den relevanten Codestellen Textausgaben hinzu. Die Textausgaben beinhalten zum Beispiel berechnete Daten oder einfache Ausführungsbestätigungen.

Reichen die erhaltenen Daten nicht aus, um den Fehler zu beheben, werden *Breakpoints* verwendet. Die Ausführung des Codes wird an vorgegebenen Stellen pausiert. Hierdurch werden sämtliche Variablen abrufbar und kontrollierbar. Des Weiteren wird der Code nun Zeile für Zeile ausgeführt, sodass jede Änderung der Variablen ersichtlich wird.

Die Feedback-Methode ist die genauste und detailreichste Methode der Fehlerbehebung und erzeugt einen entsprechenden Aufwand, weshalb die visuelle Analyse immer zuerst Anwendung findet.

Iterative Arbeitsweise

Nach der Vollendung eines Meilensteins oder beim Auftreten eines unlösbaren Fehlers wird die iterative Arbeitsweise angewendet. Es wird also ein Sachkundiger einbezogen, der Vorschläge für Problemlösungen oder Verbesserungen einbringt.

3.1.5 Projektabschluss

Nach der Vollendung des Projektes findet eine letzte Überprüfung statt. Sie soll sicherstellen, dass die Zielsetzung aus der Einleitung erfüllt wurde. Die Zielsetzung und das Spiel werden mehreren Fachkundigen vorgestellt. Diese beurteilen die einzelnen Ziele auf Vollständigkeit und sichern damit den gesamten Umfang des Spieles.

3.2 Verwendete Spielelemente

3.2.1 Pfadfindung

Welche Umsetzungsmöglichkeit aus Kapitel 2 verwendet wird, ist von dem Arbeitsaufwand der jeweiligen Umsetzung abhängig.

Die Verwendung des *A* Algorithmus* verringert den Arbeitsaufwand beim Erstellen des Levels, da der optimale Fahrweg automatisiert gefunden wird. Die Programmierung des *A* Algorithmus* ist dagegen komplexer. Zusätzlich muss ein automatisiertes System zur Unterteilung der Fahrbahn in kleine Felder entwickelt werden.

Das statische Wegpunktsystem erfordert beim Erstellen des Levels mehr manuelle Arbeit, ist im Vergleich jedoch einfacher zu programmieren.

Da dieses Projekt lediglich ein Level und nur eine Rennstrecke enthalten wird, ist die einmalige Platzierung von Wegpunkten effizienter. Deshalb wird das statische Wegpunktsystem angewendet.

3.2.2 Wahrnehmung der KI

Auch bei der Wahrnehmung der KI bestimmt das Verhältnis zwischen Aufwand und Ergebnis, welche der drei Möglichkeiten gewählt wird.

Das *Wärmebildsystem* und die *künstlichen Potenzialfelder* sind jeweils sehr umfangreich. Für das *Wärmebildsystem* wird eine Graphenfunktion benötigt, um die *Hitzelinie* zu speichern. Die *künstlichen Potenzialfelder* setzten eine Kartenerstellung voraus. Diese beiden zusätzlichen Elemente müssen erst programmiert werden, bevor sie in dem Projekt verwendet werden können.

Das *Raycastsystem* erfordert zunächst *Raycasts*, die abgeschossen und andere Objekte registrieren können. Die *Raycastfunktion* muss jedoch nicht mehr neu programmiert werden, da sie bereits von der Spieleengine gestellt wird.

Aufgrund der bereits bestehenden *Raycasts* und dem damit verringerten Arbeitsaufwand wird das *Raycastsystem* in diesem Projekt verwendet.

3.2.3 Ruberbanding

Das *Rubberbanding* ist für ein spannendes Spielgeschehen unabdingbar. Sollte es im Verlauf des Projektes zur Programmierung der Steuerung für einen Spieler kommen, wird es implementiert.

4. Durchführung

Während der praktischen Durchführung muss die Programmierung der KI und der Physik zeitlich aufeinander angepasst werden. Die Physik stellt die Basis für das restliche Spiel dar. Die KI benutzt beispielsweise Funktionen, die in der Physik gestellt werden, um die Fahrzeuge zu bewegen. Aus diesem Grund wird die Physik zuerst programmiert und beinahe vollständig beendet, bevor die Programmierung der KI begonnen wird.

4.1 Finale Planung: Iterative Arbeit

Vor Beginn der Programmierung gab ein Sachkundiger seine Meinung zur Planung der praktischen Umsetzung ab. In der Konversation wurden zunächst alle erarbeiteten Methoden bestätigt. Die Pfadfindung, der endliche Automat und die Wahrnehmung sollten wie geplant umgesetzt werden.

Zur Umsetzung des Rubberbandings wurde vorgeschlagen, das System nur auf den ersten Gegner vor dem Spieler und allen Gegnern hinter dem Spieler anzuwenden. Zusätzlich sollte das Fahrzeug hinter dem Spieler besonderen Druck auf diesen ausüben, indem es häufig Überholversuche startet und etwas aggressiver fährt. Durch das Auslassen der restlichen Gegner vor dem Spieler sollte die Entstehung einer Gegneransammlung verhindert werden.

4.2 Pfadfindung

Wie bereits beschrieben, wird zuerst die Pfadfindung mittels eines statischen Wegpunktsystems umgesetzt.

4.2.1 Markierung der Rennstrecke

Aufbau

Die Rennstrecke wird in *Sektoren* unterteilt, indem man paarweise *Begrenzungspunkte* auf dem rechten und linken Fahrbahnrand platziert. Werden jeweils alle linken und rechten *Grenzpunkte* verbunden, entstehen die Fahrbahngrenzen, die von der KI nicht überfahren werden dürfen. Die Verbindungslinien zwischen den Punkten eines Paares werden als *Schnittflächen* der *Sektoren* bezeichnet (siehe Abbildung 9).

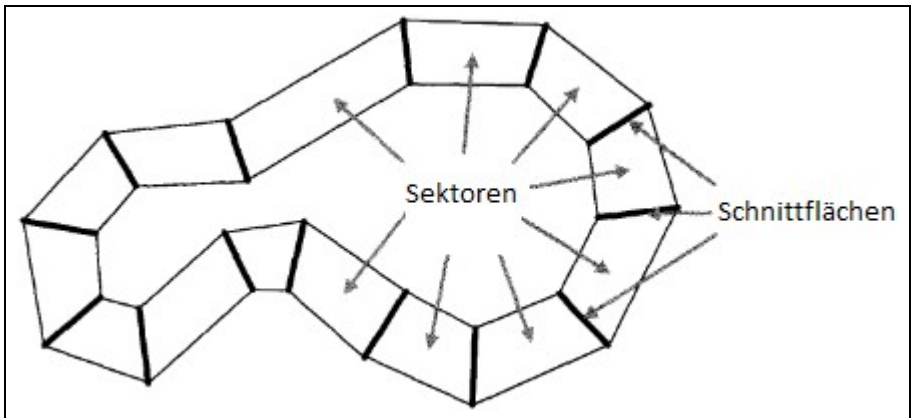

Abbildung 9: Unterteilung einer Rennstrecke in Sektoren

In den *Schnittflächen* sind die *Knotenpunkte der Fahrlinien* gespeichert. Verbindet man die *Knotenpunkte der Fahrlinien*, erhält man die Ruten, an denen sich die KI im Spiel orientieren wird. Die Anzahl der *Fahrlinien* ist grundsätzlich unbegrenzt, wird aber durch die Breite der Fahrbahn limitiert. In der Regel werden mindestens zwei *Fahrlinien* angegeben. Die erste Linie ist die *Rennlinie*. Ihre *Knotenpunkte* werden nach der schnellstmöglichen Rute auf den *Schnittflächen* platziert. Die zweite *Fahrlinie* wird als *Überhollinie* verwendet und befindet sich dementsprechend weiter außen auf der Strecke (siehe Abbildung 10) (vgl. Gari 2002: 439-441).

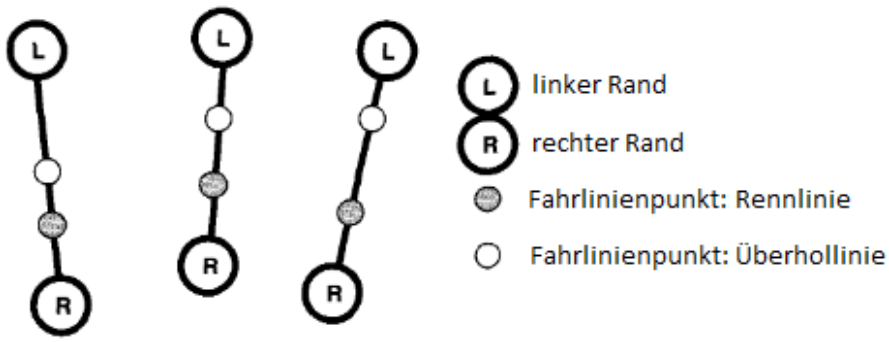

Abbildung 10: Schnittflächen der Sektoren

Eine Verbesserung dieses Systems ist es, alle Positionsdaten in zweidimensionalen Vektoren zu speichern, um die Höhenunterschiede der Strecke auszublenden. Da ein Fahrzeug ohnehin nicht in der Lage ist, nach oben und unten zu steuern, wird auf diese Daten verzichtet (vgl. Gari 2002: 443, 445).

Umsetzung

Die wesentliche Aufgabe während der Fahrbahnmarkierung ist das Platzieren der Wegpunkte. Je nachdem wie lang und kurvig die Strecke jedoch ist, sammelt sich eine große Anzahl an Wegpunkten an. Die Hauptaufgabe besteht also darin, die Platzierung so weit wie möglich zu automatisieren. Folglich müssen Hilfswerkzeuge programmiert werden.

Das erste Hilfswerkzeug umfasst die Funktionen, die die gesamte Rennstrecke betreffen. Es bietet Hilfestellungen zum Erzeugen, Verwalten, Löschen und Ordnen aller Wegpunkte (siehe Abbildung 11).

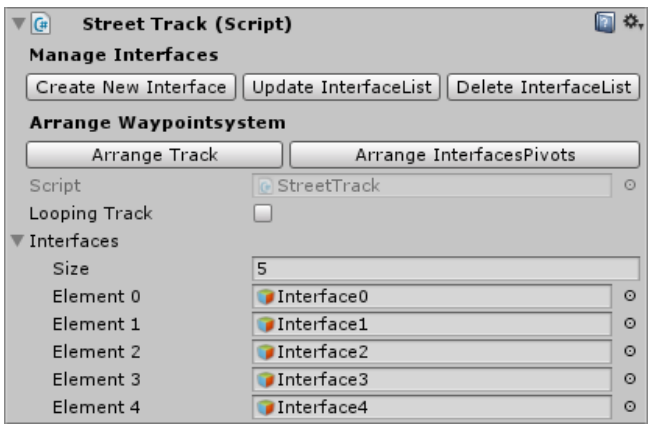

Abbildung 11: Hilfswerkzeug: Rennstrecke

Die erste Funktion platziert auf Knopfdruck eine vollständige Schnittfläche (Knopf: „Create New Interface“). Es werden also vier Wegpunkte gleichzeitig erzeugt. Zusätzlich wird jede Schnittfläche automatisch in Fahrtrichtung hinter der Vorherigen platziert.

Die zweite Hilfestellung speichert alle Schnittflächen und zeigt diese in einer Liste an. Sollten die Schnittflächen per Hand vertauscht oder gelöscht werden, wird zusätzlich eine Funktion geboten, die die gespeicherte Liste von Schnittflächen aktualisiert (Knopf: „Update InterfaceList“).

Die dritte Funktion löscht sämtliche Schnittflächen und aktualisiert gleichzeitig die gespeicherte Liste (Knopf: „Delete InterfaceList“).

Die vierte Methode behandelt Höhenunterschiede der Fahrbahn. Die Methode übernimmt die Höhenpositionierung der Wegpunkte und platziert sie 20 cm über dem Untergrund (Knopf: „Arrange Track“).

Die letzte Funktion löst ein Problem, welches bei der Bewegung einzelner Punkte einer Schnittfläche entsteht. Da alle Wegpunkte einer Schnittfläche als ein zusammenhängendes Objekt gespeichert werden, teilen sie sich einen gemeinsamen Verankerungspunkt. Dieser behält seine Position bei, wenn die einzelnen Wegpunkte verschoben werden. Aus diesem Grund sind die Wegpunkte oft weit von ihrem Verankerungspunkt entfernt. Die letzte Funktion setzt den Verankerungspunkt auf die Position des Fahrlinienpunktes, sodass sofort ersichtlich ist zu welcher Schnittfläche er gehört (Knopf: „Arrange Interface Pivots“).

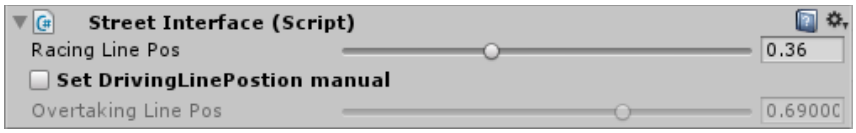

Abbildung 12: Hilfswerkzeug: Schnittfläche

Das zweite Hilfswerkzeug bietet Funktionen, die die Platzierung von Wegpunkten einzelner Schnittflächen vereinfachen (siehe Abbildung 12).

Eine Schnittfläche besteht aus den Begrenzungspunkten und den Fahrlinienpunkten. Die Fahrlinienpunkte müssen sich unmittelbar auf der Linie zwischen den Begrenzungspunkten befinden. Die erste Funktion sorgt bei der Verschiebung eines Begrenzungspunktes dafür, dass sich die Fahrlinienpunkte immer auf der geforderten Linie befinden.

Die zweite Hilfestellung bietet zwei Schieberegler, die die Verschiebung der Fahrlinienpunkte steuern. Der Wertebereich [0 - 1] stellt die Position zwischen der linken und rechten Straßenbegrenzung dar. Die Werte der Regler steuern die relativen Positionen der Rennlinie und der Überhollinie.

Die Positionierung der Überhollinie kann zusätzlich automatisch auf die Position der Rennlinie angepasst werden. Wird diese Funktion aktiviert, befindet sich die Überhollinie immer in einem festen Abstand zur Rennlinie. Sie wird stets auf der Seite der Rennlinie positioniert, die mehr Abstand zum Straßenrand bietet.

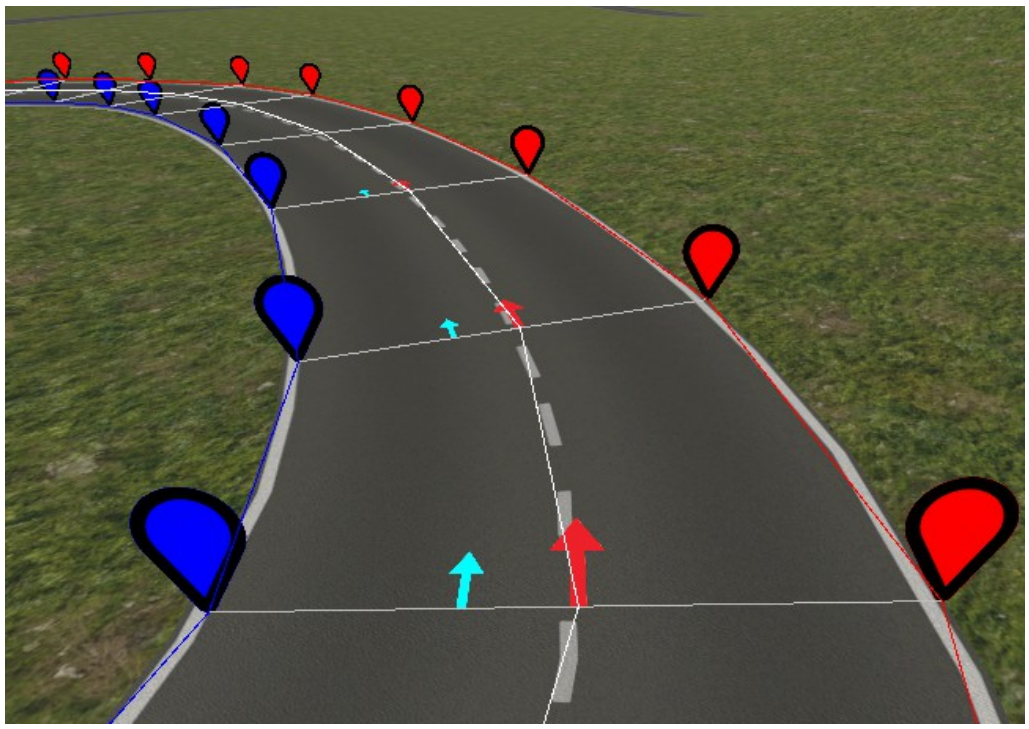

Abbildung 13: Visualisierung des statischen Wegpunktsystems

Das letzte Hilfswerkzeug ist die visuelle Darstellung der Wegpunkte und Verbindungslinien (siehe Abbildung 13). Die rechten und linken Streckenbegrenzungen werden durch rote und blaue Marker und Linien visualisiert. Die Rennlinie erhält eine weiße Linie und rote Pfeile, die in Fahrtrichtung deuten. Die Überhollinie wird durch kleinere blaue Pfeile dargestellt.

In einem Rennspiel eines vollständigen Entwicklerteams würde dieses Hilfswerkzeug zusätzlich das 3D-Modell der Straße erzeugen. Solch eine Funktion würde jedoch einen großen Entwicklungsaufwand bedeuten und die Zusammenarbeit mit einem Spiele-Artist voraussetzten. Aus diesen Gründen wird im Projekt das kostenlose Programm „EasyRoads3D Free v3“ zum Erzeugen der Fahrbahn benutzt (vgl. ANDASOFT 2011).

Die Markierung der Fahrbahn läuft mittels der aufgeführten Hilfestellungen einfach und mit geringem Arbeitsaufwand ab. Zuerst wird ein Objekt erzeugt, welches das erste Hilfswerkzeug beinhaltet. Die erste Schnittfläche wird durch einen Knopfdruck erzeugt und die Grenzpunkte können durch die spitz zulaufenden Marker präzise auf ihre vorbestimmte Position gezogen werden. Die Lage der Fahrlinienpunkte wird anschließend durch den Schieberegler des zweiten Hilfewerkzeuges bestimmt.

Die Erzeugung und Positionierung der Schnittflächen wird wiederholt, bis die gesamte Rennstrecke markiert ist.

Im Anhang befindet sich ein Video, in dem die Platzierung des Wegpunktsystems gezeigt wird (siehe Anhang: „Platzierung des Wegpunktsystems“).

4.2.2 Gespeicherte Daten

Aufbau

Der bisher beschriebene Aufbau wird von der KI verwendet, um die Strecke entlangzufahren. Die Positionen des Wegpunktsystems werden außerdem für weitere Berechnungen verwendet, die Echtzeitabfragen verringern und Leistung sparen. Die Berechnungen betreffen Streckeneigenschaften, die zur Vorausplanung des Fahrverhaltens genutzt werden.

Solch eine Streckeneigenschaft ist beispielsweise der Radius einer Kurve. Es wir also gespeichert, wie eng eine Kurve ist. Mit diesen Daten und der Fahrphysik berechnet die KI voraus, wie hoch ihre Maximalgeschwindigkeit in der Kurve sein darf. Die Formel für die Rechnung lautet:

$$\frac{Masse * Geschwindigkeit^2}{Radius} \leq Bodenhaftung_{Max}$$

Auf diese Art und Weise ist es auch möglich, den Zeitpunkt und Stärke einer Bremsung vorauszuberechnen (vgl. Tomlinson/Melder 2013: 487). Im Anhang zeigt ein Video die Auswirkung der Berechnung der Richtgeschwindigkeit (siehe Anhang: „Richtgeschwindigkeit“).

Zuletzt ist man durch die Knotenpunkte der Rennlinie in der Lage, eine aktuelle Rangliste des Rennens zu berechnen. Um das zu erreichen, speichert man in jedem Knotenpunkt der Rennlinie die Strecke vom Startpunkt bis zu dem jeweiligen Knotenpunkt. Wenn ein Fahrzeug nun von der Rennlinie abkommt, bestimmt man den Punkt auf der Rennlinie, der dem Fahrzeug am nächsten ist. Anschließend addiert man die Gesamtstrecke des letzten Knotenpunktes mit der Distanz zum aktuellen Rennlinienpunkt. Die so berechnete gefahrene Strecke spiegelt die aktuelle Position wieder (vgl. Gari 2002: 442).

Umsetzung

Um alle geforderten Daten einfach berechnen und aktualisieren zu können, wird das erste Hilfswerkzeug um eine Funktion erweitert. Auf Knopfdruck werden folgende drei Berechnungen für jede Schnittfläche durchgeführt.

Zuerst werden die zweidimensionalen Vektoren aus dem Unterkapitel 4.2.1 berechnet. Sie speichern die Position aller Wegpunkte ohne Höhenwerte, sodass die nachfolgende Berechnung ermöglicht wird. Im zweiten Schritt werden die Radien der einzelnen Kurven bestimmt. In dieser Berechnung wird jeder Schnittfläche ihr eigener Kurvenradius zugeordnet, indem ihre Begrenzungspunkte und die der nachfolgenden Schnittfläche verwendet werden. Aus den Begrenzungspunkten beider Schnittflächen wird jeweils eine lineare Funktionsgleichung gebildet: $y = m * x + b$. Diese stellen den Verlauf der der Schnittflächen dar (siehe Abbildung 14: G1 und G2). Durch die Gleichsetzung beider Geraden wird ihr Schnittpunkt berechnet (siehe Abbildung 14: S). Bestimmt man nun die Entfernung zwischen Rennlinienpunkt und Schnittpunkt, erhält man den Radius des Kurvenabschnittes. Im Anhang befindet sich der genaue Code zur Berechnung des Radius der ersten Schnittfläche (siehe Anhang: „Radiusberechnung“).

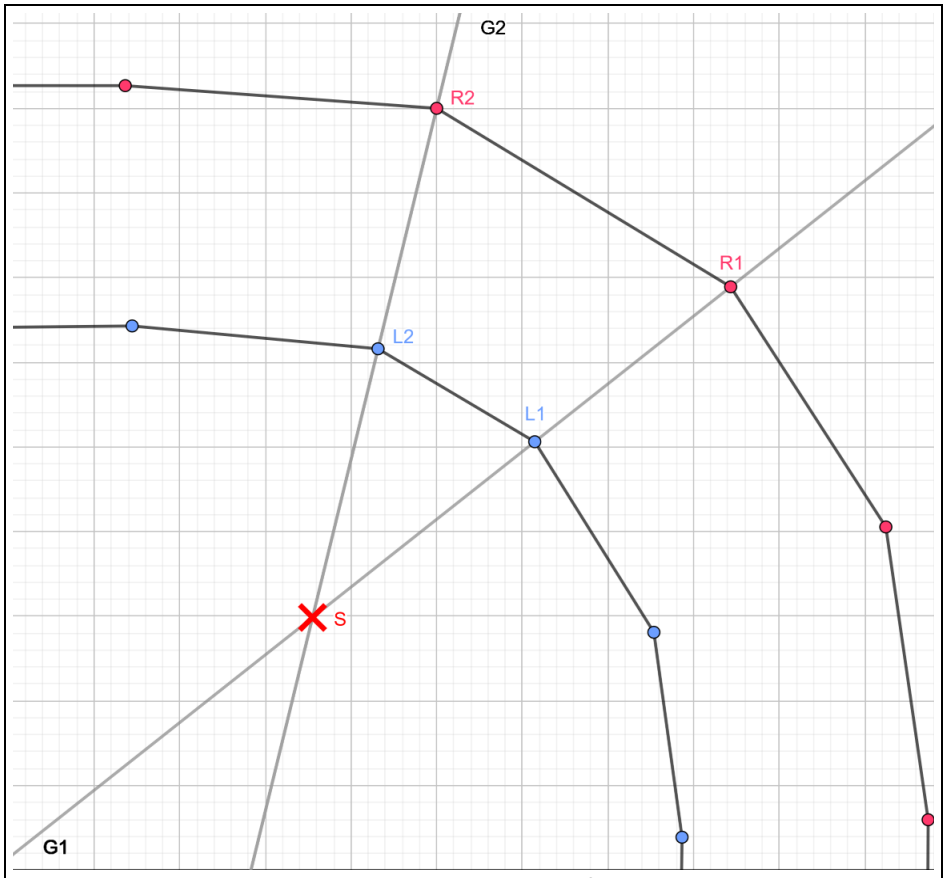

Abbildung 14: Der Schnittpunktes zweier Schnittflächen in 2D

Die letzte Berechnung bestimmt die Streckenlängen von dem Startpunkt bis zu jedem Rennlinienpunkt. Zuerst wird die Streckenlänge des Rennlinienpunktes, der sich auf der Startlinie befindet, auf null gesetzt. Nun wird der Abstand des nächsten Rennlinienpunktes zu seinem Vorgänger bestimmt. Der Abstand wird zu der Gesamtstrecke des Vorgängers addiert und als eigene Gesamtstrecke gespeichert. Indem diese Berechnung für alle Punkte der Rennlinie nacheinander durchgeführt wird, wird nach und nach die Gesamtstrecke für jeden Punkt ausgerechnet.

4.2.3 Iterative Arbeit

Nach dem Abschluss der Pfadfindung wurden wieder zwei Fachkundige hinzugezogen, die die Pfadfindung bewertet und Verbesserungsvorschläge eingebracht haben. Der wesentliche Teil der Umsetzung wurde befürwortet. Es ist jedoch aufgefallen, dass die sichtbare Straße nicht aus dem statischen Wegpunktsystem erzeugt wurde. Um mehr Arbeit zu sparen wurde vorgeschlagen, die Platzierung des Wegpunktsystems automatisiert und auf das 3D-Modell der Straße ausgerichtet ablaufen zu lassen.

Dieser Vorschlag setzt aber Programmierung im 3D-Mesh Bereich voraus. Die Einarbeitung in diesen Bereich erfordert einigen Zeitaufwand. Aus diesem Grund wird dieser Ansatz am Ende des Projektes nur umgesetzt, wenn genug Zeit zur Verfügung bleibt.

4.3 Künstliche Intelligenz

4.3.1 Steuerung

Die KI hat im Spiel die gleichen Möglichkeiten das Fahrzeug zu steuern, wie der Spieler. Sie benutzt also Lenkung, Gas und Bremse. Diese Funktionen steuert sie durch Dezimalzahlen.

Die Dezimalzahlen befinden sich für Lenkung und Gas im Wertebereich von -1 bis 1. Die Bremse wird mit dem Wertebereich 0 bis 1 gesteuert. Die Grenzen repräsentieren jeweils die Extremwerte ihrer Funktion (siehe Tabelle 1) (vgl. Gari 2002: 445).

Tabelle 1: Steuerung durch Dezimalzahlen

Wert	Lenkung	Gas	Bremse
-1	Links	Bremsen / Rückwärts	-
0	Geradeaus	Tempo halten	Nicht bremsen
1	Rechts	Beschleunigen	Vollbremsung

4.3.2 Vorausschauend fahren

Aufbau

Um die Fahrweise der KI realistisch erscheinen zu lassen, verzichtet man darauf, das Fahrzeug direkt von Knotenpunkt zu Kontenpunkt fahren zu lassen. Bei dieser Vorgehensweise würde die Lenkung einen unnatürlichen Ruck machen, sobald ein Kontenpunkt erreicht und ein neuer Knotenpunkt als Ziel gesetzt wird. Stattdessen verwendet man ein unsichtbares Objekt, welches Läufer genannt wird. Seinen Namen erhält der Läufer von seiner Funktionsweise. Er läuft in einem Abstand, der von der aktuellen Geschwindigkeit abhängig ist, vor dem Fahrzeug. Dabei befindet er sich stets auf der aktuell gewählten Fahrlinie und ersetzt die Knotenpunkte als ansteuerbares Ziel für die KI (siehe Abbildung 15).

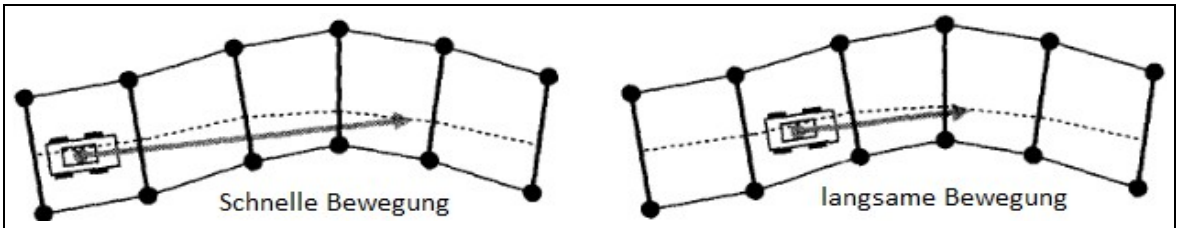

Abbildung 15: Läufer bei hoher und niedriger Geschwindigkeit

Die Bestimmung der Distanz des Läufers zum Fahrzeug folgt zwei Regeln. Zuerst gilt, je höher die aktuelle Geschwindigkeit des Fahrzeuges ist, desto größer wird der Abstand (siehe Abbildung 15). Die zweite Regel tritt ein, wenn das Fahrzeug in eine enge Kurve fährt. Hier sollte die Distanz nicht durch die aktuelle Geschwindigkeit bestimmt werden, da die Zielrichtung die Kurve scheiden könnte (siehe Abbildung 16 links). Stattdessen wird der Läufer auf den ersten Knotenpunkt der Kurve gesetzt. Erreicht das Fahrzeug einen Minimalabstand, springt der Läufer auf den nächsten Kontenpunkt. So steuert die KI immer erst auf den nächsten Abschnitt einer Kurve zu ohne den Rand der Kurve zu schneiden (siehe Abbildung 16 rechts) (vgl. Gari 2002: 446-448).

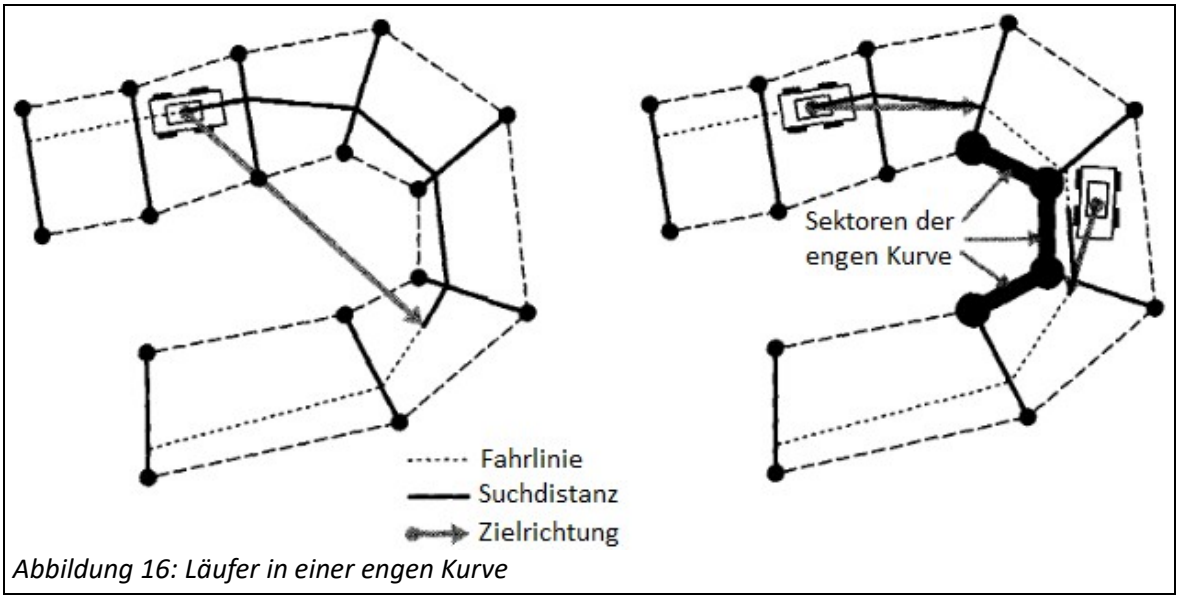

Abbildung 16: Läufer in einer engen Kurve

Umsetzung

Die Verwendung des Läufers verhindert das ruckartige Lenken zwischen einzelnen Sektoren. Durch das Ansteuern eines festen Punktes entsteht aber ein Problem, welches sich durch ein unnatürliches Verhalten zeigt. Kommt die KI von der Rennlinie ab, steuert sie wieder auf diese zurück. Durch die Trägheit des Fahrzeuges, die von der Physik simuliert wird, wird es jedoch auf die andere Seite der Fahrlinie hinaus gedrückt. Die KI steuert als Reaktion wieder in die entgegengesetzte Richtung, kehrt zur Fahrlinie zurück und wird durch die Trägheit abermals über die Fahrlinie gedrückt. Auf diese Weise entsteht ein Pendeln um die Fahrlinie herum, das immer größer wird.

Die Ursache des Problems ist die Funktion, die die Reifen immer genau auf den Läufer ausrichtet und damit eine scharfe Lenkung erzeugt. Im Anhang befindet sich die Berechnung, welche die Ausrichtung der Reifen bestimmt (siehe Anhang: „Reifenstellung“)(vgl. Gari 2002: 448).

Um das Pendeln zu unterdrücken, muss die Lenkung abgeschwächt werden, sobald sich die Ausrichtung des Fahrzeuges an die Zielausrichtung angleicht. Um diesen Effekt zu erzielen, wird für das Projekt ein neues System entwickelt. Es wird ein zweites unsichtbares Objekt genutzt, welches sich auf dem Vorwärtsvektor des Fahrzeuges befindet. Der Abstand zum Fahrzeug wird dem Abstand zwischen Läufer und Fahrzeug gleichgesetzt (siehe Abbildung 17).

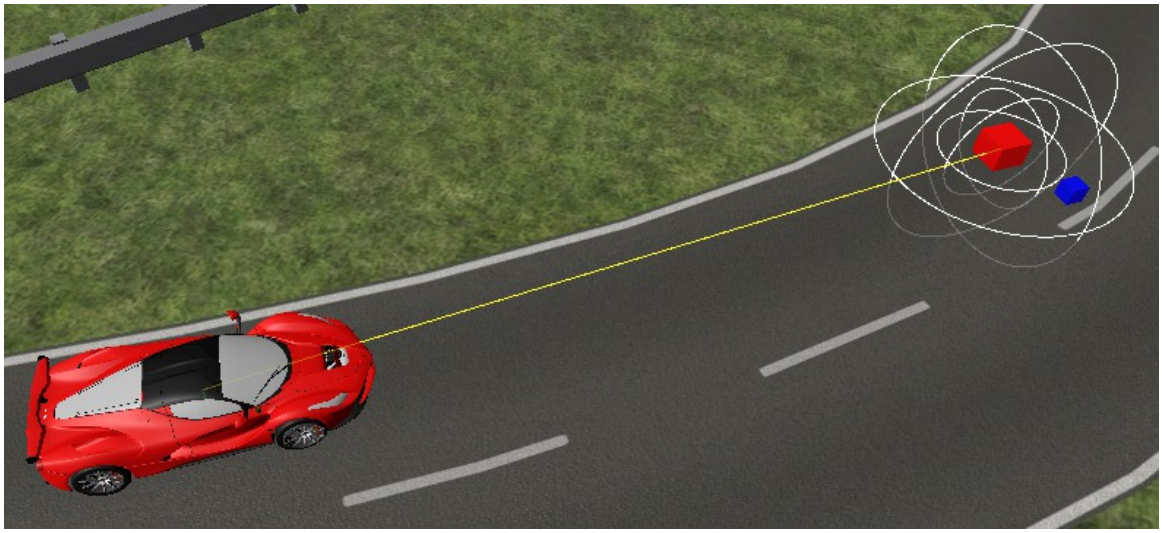

Abbildung 17: Läufer und Pufferzonen
Rot: Läufer, Blau: Vorwärtsobjekt, Weiß: Pufferzonen, Gelb: Abstand

Nun werden um den Läufer herum Pufferbereiche erstellt. Befindet sich das vorwärts ausgerichtete Objekt innerhalb der Bereiche, wird die Lenkung abgeschwächt. In Abbildung 17 werden zwei Bereiche verwendet. Innerhalb des äußeren Bereiches wird der berechnete *Lenkwert* (siehe Unterkapitel 4.3.1) halbiert und innerhalb des inneren Bereiches geviertelt. Auf diese Weise nähert sich das Fahrzeug langsamer der Zielausrichtung an und das Schlenkern wird minimiert. Im Anhang befindet sich ein Video, in dem die Verwendung und Auslassung des Pufferbereiches gezeigt wird (siehe Anhang: „Läufer und Pufferbereich“).

4.3.3 Wahrnehmung und Kollisionsvermeidung

Aufbau

Die Wahrnehmung der KI findet über ein Raycast-System statt, welches das Umfeld des Fahrzeuges abtastet. Dieses System besteht aus einzelnen Raycasts, die abhängig von ihrer Ausrichtung verschiedene Funktionen mit Daten beliefern können (siehe Abbildung 18).

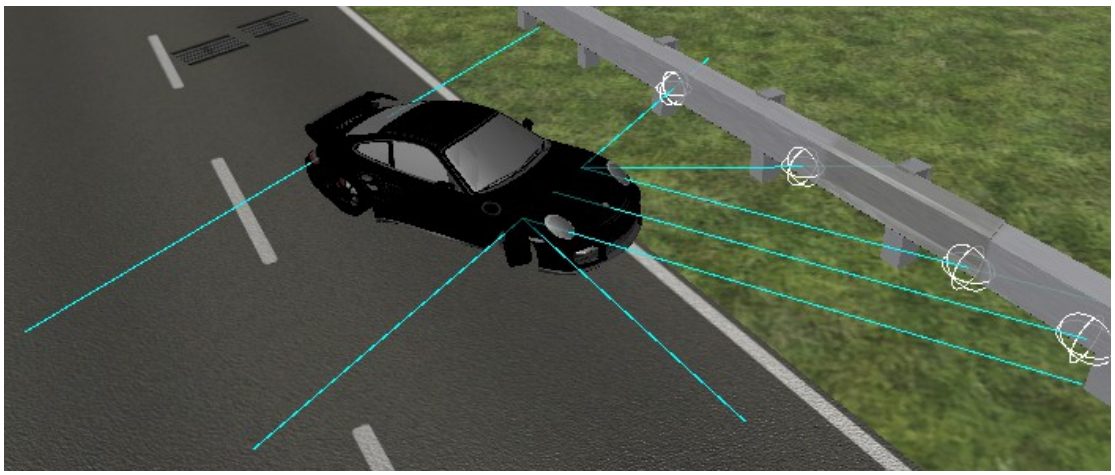

Abbildung 18: Raycasts Umsetzung

Zunächst werden drei Raycasts gerade nach vorne ausgerichtet, um Objekte wahrzunehmen, die sich vor dem eigenen Fahrzeug befinden. Werden andere Fahrzeuge registriert, kann die KI Abstand von ihnen halten oder einen Überholvorgang starten.

Des Weiteren werden sechs Raycasts verwendet, um Abstand von seitlichen Hindernissen und Fahrzeugen zu halten. Vier seitlich positionierte Raycasts erfassen Objekte, die sich bereits direkt neben dem Fahrzeug befinden. Die zwei restlichen Raycasts tasten die Bereiche seitlich vor dem Fahrzeug ab. Damit ermöglichen sie ein rechtzeitiges Ausweichen.

Umsetzung

Wahrnehmung und Kollisionsvermeidung lassen sich auf zwei Zielbereiche aufteilen: vorwärts und seitwärts.

Die Aufgaben der vorwärts ausgerichteten Wahrnehmung bestehen aus dem Abstandhalten zu anderen Fahrzeugen oder statischen Objekten und dem Starten von Überholvorgängen.

Um diese Aufgabe situationsabhängig zu erfüllen, werden aus den Daten der *Raycasts* drei entscheidungsrelevante Faktoren berechnet. Zuerst wird die Differenz zwischen der eigenen Zielgeschwindigkeit und der tatsächlichen Geschwindigkeit des anderen Fahrzeuges bestimmt. Ist die Differenz hoch genug, startet die KI einen Überholvorgang. Der eigentliche Überholvorgang wird in Kapitel 4.3.4 näher betrachtet.

Anschließend werden beide tatsächlichen Geschwindigkeiten verglichen. Ist die Eigene höher, setzt die KI ihre Zielgeschwindigkeit mit der aktuellen Geschwindigkeit des vorausfahrenden Fahrzeuges gleich. Auf diese Weise bremst sie ab und eine Kollision wird verhindert. Um einen Abstand zwischen beiden Fahrzeugen zu erzeugen, wird die Zielgeschwindigkeit ein weiteres Mal leicht verringert.

Zuletzt wird der Abstand zwischen den beiden Fahrzeugen bestimmt. Befindet sich die KI zu nahe an dem vorausfahrenden Fahrzeug, wird zusätzlich zu den vorherigen Maßnahmen stark abgebremst.

Die seitliche Wahrnehmung dient lediglich zum Abstand halten. Wird zwischen zwei Fahrzeugen ein Minimalabstand unterschritten, wird die aktuelle Situation auf die folgende Frage analysiert: Welches der involvierten Fahrzeuge überholt das andere? Grundsätzlich gilt die Regel, dass das überholende Fahrzeug für den Abstand sorgen muss. Sollte es dazu jedoch nicht vollständig in der Lage sein, da es zum Beispiel zu wenig Ausweichfläche zur Verfügung hat, weicht es nur bis zum Straßenrand aus. In diesem Fall muss das überholte Fahrzeug die fehlende Entfernung kompensieren. In Tabelle 2 wird dargestellt, in welchen Situationen die Funktionen „Ausweichen“ und „Kompensieren“ ausgeführt werden.

Tabelle 2: Überholsituationen zwischen Fahrzeug A und Fahrzeug B

Situation	Aktion für Fahrzeug A
A überholt B	Ausweichen
A wird von B überholt	Kompensieren
Weder A noch B überholen	Ausweichen (um die Hälfte des Minimalabstand) + Kompensieren

Das Ausweichen der Fahrzeuge wird durch eine Verschiebung des *Läufers* erreicht. Befindet sich beispielsweise ein Hindernis auf der rechten Seite der KI, wird der *Läufer* um den benötigten Abstand nach links verschoben.

Im Anhang ist ein Video, welches die das System der Wahrnehmung und Kollisionsvermeidung aufzeigt (siehe Anhang: „Wahrnehmung und Kollisionsvermeidung“).

4.3.4 Entscheidungsfindung über einen endlichen Automaten

Aufbau

Die Entscheidungsfindung der KI findet über einen endlichen Automaten (engl.: finite state machine) statt. Der Automat besteht aus verschiedenen Zuständen, die die verschiedenen Aktionen der KI beinhalten (siehe Abbildung 19). Der Wechsel zwischen einzelnen Zuständen ist von der aktuellen Situation abhängig. Dementsprechend ist die Wahrnehmung der KI stark mit ihrer Entscheidungsfindung verknüpft.

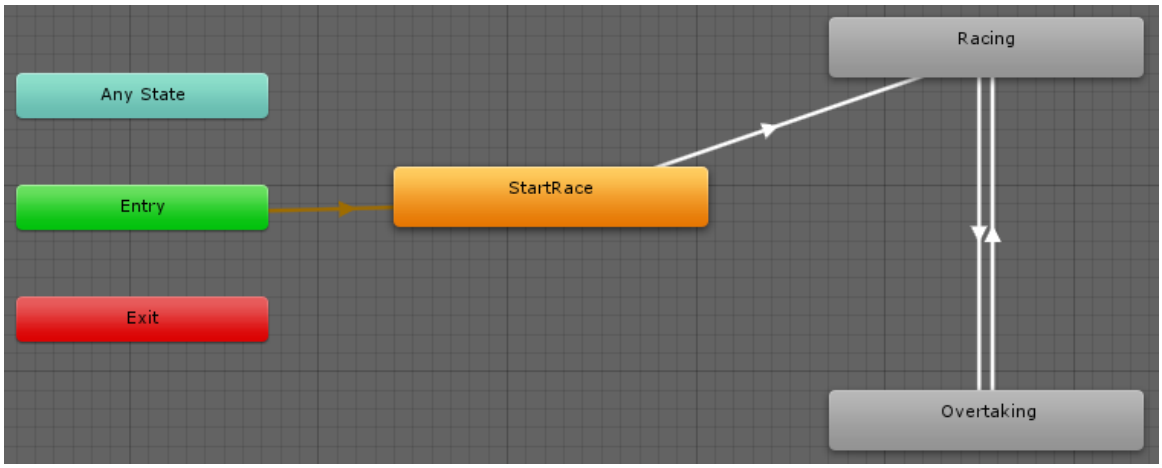

Abbildung 19: Endlicher Automat

Zustand_Start

Zu Beginn des Rennens befindet sich die KI in diesem Zustand. Sie wartet auf das Startsignal und wechselt dann in den *Zustand_Fahren* (vgl. Gari 2002: 445f).

Zustand_Fahren

In diesem Zustand wird stets die höchste Geschwindigkeit angestrebt, die das Fahrzeug auf dem aktuellen Streckenabschnitt erreichen kann. Die Berechnung der Höchstgeschwindigkeit eines Abschnittes wurde in Kapitel 4.2.2 aufgezeigt. In diesem Zustand wird immer der *Rennlinie* gefolgt (vgl. Tomlinson/Melder 2013: 474).

Zustand_Überholen

Registriert eine KI vor sich ein anderes Fahrzeug, wird die Möglichkeit zur Überholung nicht immer wahrgenommen. Entscheidend für den Start eines Überholvorganges ist die Geschwindigkeitsdifferenz beider Fahrzeuge. Ist die Geschwindigkeitsdifferenz groß genug, wird der Überholvorgang gestartet und die KI wechselt auf die *Überhollinie*. Im *Zustand_Überholen* wird ihre angestrebte Geschwindigkeit erhöht. Hat sie das gegnerische Fahrzeug überholt, fährt sie so lange im Überholzustand, bis sie einen vorgegebenen Vorsprung gewonnen hat. Anschließend wechselt sie wieder in den *Zustand_Fahren* (vgl. Tomlinson/Melder 2013: 474f; Gari 2002: 448f).

Umsetzung

Zu den geplanten Zuständen gehört normalerweise auch *Zustand_zurück_zur_Strecke*, der bei Kontrollverlust oder beim Abkommen von der Fahrbahn verwendet wird. Er würde dafür sorgen, dass sich das Fahrzeug stabilisiert und zurück zur Strecke findet. Die Verwendung eines *Läufers* und der *Vorausberechnung einer Zielgeschwindigkeit* lässt diesen Zustand jedoch überflüssig werden. Im Spielgeschehen sorgen die Funktionen immer für einen sicheren Fahrstil und ständiges Befolgen des vorgegebenen Weges.

Anstelle eines getrennten Zustandes wird eine Funktion eingebaut, welche ständig als Teil der anderen Zustände aktiv ist. Sollte ein Fahrzeug von der Strecke abkommen, wirkt diese Funktion entgegen, indem sie den Läufer in die benötigte Richtung verschiebt.

4.3.5 Iterative Arbeit

Zum Abschluss der künstlichen Intelligenz wurde wieder eine Expertenmeinung eingezogen. Diesmal wurden sämtliche Umsetzungen befürwortet. Zur praktischen Umsetzung wurden keine notwendigen Verbesserungen erkannt.

4.4 Weitere Funktionen des Endproduktes

4.4.1 Simulierte Situationen

In der Finalisierung des Projektes ist es das Ziel, die Funktionsweisen der KI betrachten und testen zu können. Zu diesem Zweck werden verschiedene Situationen vorbereitet und zur freien Auswahl bereitgestellt.

Einzelnes Fahrzeug

In der ersten Situation fährt ein einzelnes Fahrzeug die Rennstrecke entlang. So wird aufgezeigt, wie die KI ohne störende Einflüsse fahren würde. Besonders Teile der Programmierung, die nur auf das eigene Fahrzeug wirken, können betrachtet werden. Dazu gehören zum Beispiel die zwei Funktionen:

- Entlangfahren der Rennlinie
- Berechnung der Zielgeschwindigkeit

Zusätzlich kann jedes Fahrzeug, welches von dem Kooperationspartner mit der Physik erstellt wurde, einzeln aufgerufen werden.

Normales Rennen

Diese Situation simuliert ein normales Rennen zwischen acht Fahrzeugen und kommt einem normalen Rennen am nächsten. Jedes Fahrzeug hat das Ziel die Rennstrecke so schnell wie möglich entlang zu fahren. In dem Modus fahren nur Fahrzeuge desselben Fahrzeugtypen zur selben Zeit, doch auch in dieser Situation ist es möglich, jeden Fahrzeugtyp auszuwählen. Besonders die Startphase eines Rennens und die Kollisionsvermeidung vor Kurven wird anschaulich dargestellt.

Zufällige Aufstellung

Hier wird wieder ein Rennen zwischen acht Fahrzeugen bereitgestellt. Im Gegensatz zur vorherigen Situation werden die verwendeten Fahrzeugtypen zufällig bestimmt. Daher dient diese Situation dem Vergleich zwischen den verschiedenen Fahrzeugtypen und ihrer Fahreigenschaften.

Überholen

Während eines normalen Rennes fahren sämtliche Fahrzeuge so gut, wie es ihnen möglich ist. Deswegen kommt es nach der Startphase zu keinen Überholvorgängen mehr. Um die Anzahl von Überholvorgängen zu erhöhen, werden in diesem Fall ein langsames Fahrzeug und viele schnelle Fahrzeuge auf der Strecke verteilt. So kommt es auf jedem Teil der Strecke zu Überholvorgängen und das Fahrverhalten auf der Überhollinie wird demonstriert.

Falsche Richtung

Wenn die KI nicht in Fahrtrichtung ausgerichtet ist, muss sie sich eigenständig drehen können. Deshalb werden mehrere Fahrzeuge mit verschiedenen Rotationen erzeugt und das Verhalten der KI kann für jede Rotation betrachtet werden. In der Situation ist es ebenfalls möglich, aus allen Fahrzeugtypen zu wählen.

Zusammenstoß

Der Zusammenstoß ist die letzte vorgestellte Situation. Es werden zwei Fahrzeuge erzeugt, die als Hindernisse dienen. Des Weiteren werden einige Fahrzeuge platziert, die mit hoher Geschwindigkeit gegen die Hindernisse fahren werden. In dieser Situation können besonders die Physik und die Auswirkung einer Vollbremsung präsentiert werden.

4.4.2 Steuerung

Um das Nutzererlebnis zu verbessern, werden zusätzliche Funktionen eingebaut.

Zuerst kann die Kamera entweder frei bewegt oder auf ein Fahrzeug fokussiert werden. Durch die zwei alternativen Funktionen kann der Nutzer die aktuelle Situation ohne Einschränkung betrachten. Ist die Kamera an ein Fahrzeug geheftet, ist die Distanz zwischen Fahrzeug und Kamera frei anpassbar.

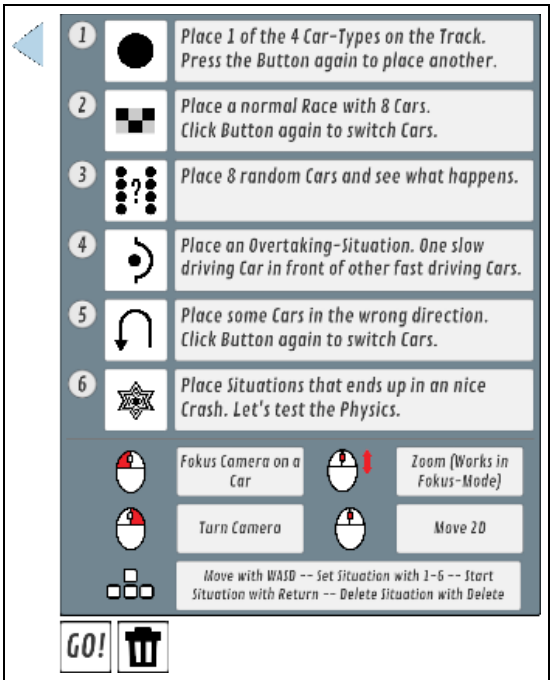

Abbildung 20: Situationsmanager

Die nächste Hilfestellung stellt sicher, dass die verschiedenen Situationen einfach platziert, gestartet und gelöscht werden können. Es ist möglich, das Management der Szenen über Tastaturbefehle und Bildschirmknöpfe zu regeln (siehe Abbildung 20).

Zuletzt wird die Funktion geboten, die Kamera in folgenden Attributen anzupassen: Bewegungsgeschwindigkeit, Drehgeschwindigkeit und Zoomgeschwindigkeit.

5. Ergebnisse und Zusammenfassung

5.1 Ergebnisse

5.1.1 Endprodukte:

Das Endergebnis beinhaltet ein ausführbares Programm, dessen Inhalte in Kapitel 4.4 vorgestellt wurden. Zusätzlich werden einige Videos über einzelne Funktionen der KI und des Endproduktes bereitgestellt (siehe Abbildung 21) (siehe Anhang „Videos“).

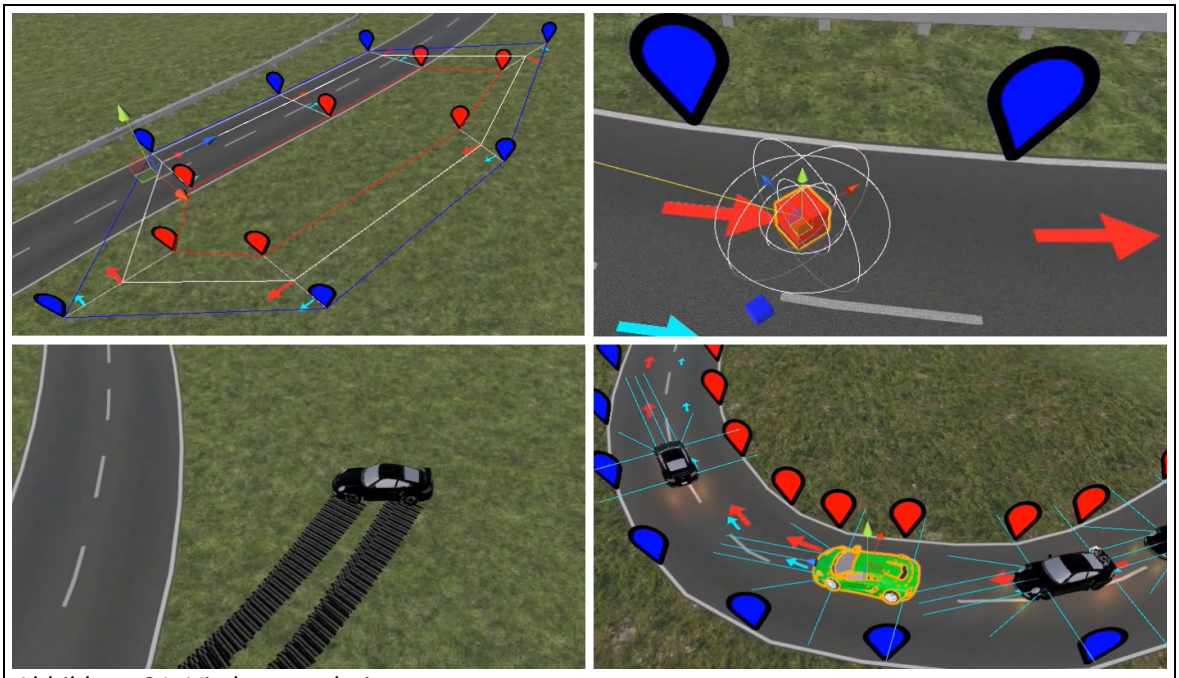

Abbildung 21: Viedeoausschnitte

5.1.2 Vergleich zur Zielsetzung

Das Ziel, eine Steuerung für einen Spieler zu programmieren, wurde in der Durchführung nicht umgesetzt. Aufgrund der limitierten Projektzeit wurde entschieden, die Umsetzung auf das Simulationsprogramm zu reduzieren. Dafür sollte dieses umfangreich und mit hoher Qualität umgesetzt werden.

Alle anderen Ziele wurden in vollem Umfang erfüllt. Das Simulationsprogramm fasste genügend Situationen in verschiedenen Variationen, um die KI und Teile der Physik darzustellen.

Die einzelnen Features der KI erforderten jeweils mehrere Unterfunktionen, um wie geplant zu funktionieren. Die Entwicklung der Unterfunktionen stellten jeweils kleine Herausforderungen dar, die alle durch Recherche und eigenen Ideen erfolgreich umgesetzt wurden.

In Tabelle 3 wird der Umsetzungsgrad jedes geplanten Features aufgelistet.

Tabelle 3: Umsetzungsgrad geplanter Features

Feature	Umsetzung
KI • autonomes Rennen fahren • Pfadfindung (auf Rennspiele spezialisiert) • Fahren / Fahrverhalten (sicher und schnell) • Reaktion auf andere Fahrzeugen	vollständig umgesetzt
Endprodukt • verschiedene Situationen • benutzerfreundliche Bedienung	vollständig umgesetzt
Kann-Ziele • Steuerung für einen Spieler	nicht umgesetzt

5.2 Zusammenfassung

Diese Arbeit behandelte jeden Schritt der Produktion einer KI für Rennspiele. Zuerst wurden verschiedene Umsetzungsmöglichkeiten zu den Bestandteilen Pfadfindung, Fahrverhalten und Interaktion zusammengetragen. Aus einem Vergleich zwischen Aufwand und Nutzen resultierten anschließend die verwendeten Umsetzungen. In dem Durchführungskapitel wurden anschließend die ausgewählten Methoden erstellt und die Besonderheiten der praktischen Umsetzung vorgeführt. Schlussendlich wurde ein Simulationsprogramm fertiggestellt, in dem die Eigenschaften und Verhaltensweisen der KI vorgestellt wurden.

5.2.1 Verbesserungen an der Vorgehensweise

Im Verlauf der Projektarbeit sind zwei Verbesserungsmöglichkeiten aufgetreten, die in zukünftigen Projekten von Anfang an betrachtet werden müssen.

Zuerst wurde zu Beginn der Programmierung nicht auf den Einbezug des endlichen Automaten geachtet. Da sich beinahe alle wichtigen Funktionen im *Zustand_Fahren* befanden, wurde zuerst dieser Zustand fertiggestellt. Dabei wurde nur nebenbei beachtet, dass der geschriebene Code auch für die anderen Zustände anpassbar sein musste. Nach dem Abschluss des Zustandes mussten deshalb einige Funktionen ausgelagert und erweitert werden, sodass sie variabler eingesetzt werden konnten.

Die zweite Verbesserung betraf die verschiedenen Fahrzeugtypen und ihre Einbindung in die Vorausberechnung der Maximalgeschwindigkeit auf einem Streckenabschnitt. Sämtliche Fahrzeuge hatten verschiedene Eigenschaften. Es bestand jedoch das Problem, dass die Physik keine Werte zur maximalen Bodenhaftung boten. Aus diesem Grund musste eine allgemeine Formel zur Berechnung der maximalen Kurvengeschwindigkeit genutzt werden. Demzufolge kam es bei einigen Fahrzeugtypen manchmal zu zu hohen oder zu niedrigen Geschwindigkeiten. In zukünftigen Projekten muss die Physik eine Funktion bieten, die die maximale Geschwindigkeit in Abhängigkeit vom Radius einer Kurve berechnet.

5.2.2 Fazit

Schlussendlich hat das Projekt mir den Programmierbereich „Künstliche Intelligenz“ weiter eröffnet. Es hat mir viele Möglichkeiten aufgezeigt, wie man unterschiedliche Probleme auf ein Projekt angepasst und mit unterschiedlichen Ansätzen lösen kann. Auch die Vorausplanung großer Code-Strukturen wird mir zukünftig noch leichter fallen.

6. Literaturverzeichnis

ANDASOFT (2011)
EasyRoads3D Free v3
Einsehbar unter:
https://assetstore.unity.com/packages/3d/characters/easyroads3d-free-v3-987
(Eingesehen: 14.07.2018)

astragon (2016)
Moto Racer 4
Einsehbar unter:
https://www.astragon.de/spiele/moto-racer-4-pcmac/
(Eingesehen: 04 Juli 2018)

Biasillo, Gari (2002)
Section 9.1-9.3 – Racing and Sports AI
In: Rabin, Steve (Hg.)
AI Game Programming Wisdom
Erscheinungsort: Hingham, Massachusetts, USA
Verlag: Cengage Learning
Seiten: 437 - 559
Einsehbar unter:
http://planiart.usherbrooke.ca/files/Rabin%202002%20-%20AI%20Game%20Programming%20Wisdom.pdf
(Eingesehen: 03 April 2018)

Brooks, Alex (2011)
Racing AI and Steering Behaviours (Video)
Einsehbar unter:
https://www.youtube.com/watch?v=_XKphuYviE0
(Eingesehen: 24 Mai 2018)

EYEmaginary (2017)
Car AI Series (Video)
Einsehbar unter:
https://www.youtube.com/playlist?list=PLB9LefPJI-5wH5VdLFPkWfnPjeI6OSys1
(Eingesehen: 19 März 2018)

Nintendo UK (2017)
Mario Kart 8 Deluxe - Overview trailer (Nintendo Switch) (Gameplay Trailer)
Einsehbar unter:
https://www.youtube.com/watch?v=jsTRUxzEl9U
(Eingesehen: 04 Juli 2018)

PlayStation (2014)
DRIVECLUB - PS4 Gameplay - Canada | Pagani Huayra (Gameplay Trailer)
Einsehbar unter:
https://www.youtube.com/watch?v=wBuAd8ffqgs
(Eingesehen: 04 Juli 2018)

PlayStation (2017)
Need for Speed Payback - PS4 Gameplay Trailer | E3 2017 (Gameplay Trailer)
Einsehbar unter:
https://www.youtube.com/watch?v=9U2MRfv8HQg
(Eingesehen: 04 Juli 2018)

Salman, Muhammad (2013)
Collision Detection and Overtaking Using Artificial Potential Fields in Car Racing game TORCS using Multi-Agent based Architecture (Master Thesis)
Erscheinungsort: Karlskrona, Sweden
Einsehbar unter: http://www.diva-portal.org/smash/get/diva2:830507/FULLTEXT01.pdf
(Eingesehen 16 Mai 2018)

Tomlinson, Simon und Melder, Nic (2013)
Section 6: Racing
In: Rabin, Steve
Game AI Pro: Collected Wisdom of Game Ai Professionals
Erscheinungsort: Boca Raton, Florida, USA
Verlag: Taylor & Francis Inc
Seiten: 471- 512
Einsehbar unter: http://www.gameaipro.com/
(Eingesehen: 07 Mai 2018)

Wang, Jung-Ying und Lin, Yong-Bin (2012)
Game AI: Simulating Car Racing Game by Applying Pathfinding Algorithms
In:
International Journal of Machine Learning and Computing, Vol. 2, No. 1, February 2012
Seiten: 13-18
Einsehbar unter: http://www.ijmlc.org/papers/82-A1090.pdf
(Eingesehen: 15 Mai 2018)

7. Abbildungsverzeichnis