

Realistisches Verhalten bei computergesteuerten Agenten in Rennspielen

WERNER WAGNER

MASTERARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

INTERACTIVE MEDIA

in Hagenberg

im September 2013

© Copyright 2013 Werner Wagner

Diese Arbeit wird unter den Bedingungen der *Creative Commons Lizenz Namensnennung–NichtKommerziell–KeineBearbeitung Österreich* (CC BY-NC-ND) veröffentlicht – siehe <http://creativecommons.org/licenses/by-nc-nd/3.0/at/>.

Erklärung

Ich erkläre eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen entnommenen Stellen als solche gekennzeichnet habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Hagenberg, am 30. September 2013

Werner Wagner

Inhaltsverzeichnis

Erklärung	iii
Kurzfassung	vi
Abstract	vii
1 Einleitung	1
1.1 Motivation	1
1.2 Aufbau und Zielsetzung der Arbeit	1
2 Künstliche Intelligenz in Spielen	3
2.1 Einführung	3
2.2 Aufbau einer Künstlichen Intelligenz	6
2.3 Rennspiele und Genres	8
2.3.1 Genres	9
3 State of the Art	12
3.1 Movement	13
3.2 PID-Regler	15
3.2.1 P-Regler	16
3.2.2 PI-Regler	16
3.2.3 PID-Regler	18
3.3 Implementierungsmöglichkeiten einer künstlichen Intelligenz in Rennspielen	18
3.3.1 Steuerung der KI durch Kollisionserkennung	18
3.3.2 Steuerung der KI durch Folgen von Pfaden oder Weg- punkten	19
4 Eigener Ansatz	21
4.1 Prinzip	21
4.2 Unity	22
4.3 Car Tutorial	22
4.4 Aufbau und Umsetzung	25
4.4.1 Steuerung der KI	26

4.4.2	Datenbank und Wegpunkte	28
4.4.3	Automatisierter Test	34
5	Auswertung und Ergebnis des eigenen Ansatzes	41
6	Fazit und Ausblick	47
6.1	Zusammenfassung	47
6.2	Ergebnis	47
6.3	Ausblick	49
A	Inhalt der CD-ROM/DVD	52
A.1	Masterarbeit	52
A.2	Projektdateien	52
A.3	Online-Quellen	52
Quellenverzeichnis		53
Literatur		53
Online-Quellen		53

Kurzfassung

Rennspiele gehören zu einem der ältesten und beliebtesten Genres in der Games Branche. Gerade bei dieser Art von Spielen ist es besonders wichtig sich mit anderen messen, zu können. Daher ist die Entwicklung einer *Künstlichen Intelligenz* gerade in diesem Bereich von großer Bedeutung. Ein herausfordernder, aber trotzdem schlagbarer Gegner ist das Ziel bei einer Entwicklung einer KI – Künstlichen Intelligenz. Diese Arbeit beschäftigt sich dabei genau um die Entwicklung eines solchen Gegners. Dabei wird neben allgemeinen Informationen, die für eine Rennspiel KI von Bedeutung sind, genauer auf ein System zur Steuerung und Verbesserung einer KI eingegangen. Bei diesem System werden für Rennspiele gängige Rennlinien zur Steuerung von computergesteuerten Gegnern von Spielern während der Fahrt erzeugt. Diese werden dann verwendet um auf zwei verschiedene Arten neue Rennlinien generieren, zu können. So soll durch ein Mitteln zweier Rennlinien oder durch ein Kombinieren von Streckenabschnitten zweier Rennlinien eine neue Rennlinie entstehen, die zu einer besseren und schnelleren KI führt. Mithilfe dieses Systems soll die KI nun mit dem Anstieg der Erfahrung des Spielers auch weiterhin eine Herausforderung darstellen können. Dies wird unter Verwendung eines automatischen Testes geprüft und das Ergebnis auf mögliche Vor- und Nachteile analysiert.

Abstract

Racing games are one of the oldest and most popular genre in the game industry. Especially in racing games players like to compete their skills with other players. This is one of the reasons why creating a challenging but still beatable AI has a big part in the development of racing games.

This work is about developing such an AI. Therefore this work proposes a system for creating racing lines for computer-controlled drivers. During a player is driving a track, a racing line consisting of way points is created. These racing lines are used to operate an AI through a track.

The main part of this work is focusing on creating new and better racing lines out of given ones. Two approaches were used and tested to achieve this goal. One approach creates a racing line by taking the mean value of two racing lines. The other one tries to create better racing line by combining sections of two tracks.

A automatic test system using these two approaches provides finally a result, which is being analysed for pros and cons.

Kapitel 1

Einleitung

1.1 Motivation

Mit dem Beginn der Videospiele waren Rennspiele ein großer Teil davon. Gamer erfreuen sich bereits seit 1974 mit dem Atari Arcade-Spiel *Gran Trak 10*, eines der ersten Video-Rennspiele, an Rennspielen. Seither kamen mehr und mehr Videospiele an den Markt. Gerade in Rennspielen ist der Wettkampf bzw. der direkte Vergleich mit anderen Spielern oder computergesteuerten Agenten besonders wichtig. Vor allem in den letzten Jahren gewann die KI – Künstlichen Intelligenz in Videospiele immer mehr an Bedeutung. Dabei versucht man das Verhalten der Künstlichen Intelligenz so gut wie möglich an das Verhalten echter Spieler anzupassen, um dem Spieler die Illusion eines Wettkampfes gegen menschliche Gegner zu geben. Damit der Spielspaß des Spielers von Dauer ist, muss das Können der KI mit dem Skilllevel des Spielers mitwachsen, sodass sie auch im späteren Spielverlauf eine Herausforderung darstellen kann. Des Weiteren sollten für die KI die gleichen Regeln wie für den Spieler gelten, um einen fairen Wettkampf zu erreichen. Das heißt, dass auf Mittel wie dem sogenannten *Rubber banding*, beschrieben in Kapitel 2.3 verzichtet werden soll.

1.2 Aufbau und Zielsetzung der Arbeit

Die Masterarbeit beschäftigt sich mit der Künstlichen Intelligenz in Rennspielen. Zu Beginn wird es im Kapitel 2 eine Einführung zu KI in Spielen geben. Unter anderem wird kurz auf die Geschichte der KI eingegangen, erklärt, was eine KI ausmacht und worauf bei der Entwicklung von KI für Spiele Wert gelegt werden soll. Dabei wird es vor allem um die Künstliche Intelligenz in Spielen gehen, die dem Rennspiel Genre angehören. Zusätzlich es auch einen kleinen Einblick in die Welt der Rennspiele geben. Welche Arten von Rennspielen es gibt und wodurch zeichnen sich diese aus.

Als Nächstes wird im Kapitel 3 auf die Entwicklung einer Rennspiel

KI eingegangen. Zuerst wird auf die allgemeine Entwicklung, wie zum Beispiel die Lenkung und das Gas einer Rennspiel KI, eingegangen - also die Steuerung des Fahrzeuges an sich. Danach werden bestehende Systeme zur Steuerung der KI durch eine Rennstrecke untersucht. Dazu zählen beispielsweise das Wegpunkt und das Sensor System. Diese werden erklärt und auf ihre Vor- und Nachteile verglichen. Unter anderem wird dabei auf ihre Möglichkeit auf Verbesserung und dem Realismus der KI geachtet.

Nachdem der allgemeinere Teil abgeschlossen ist und man nun das Basiswissen über die Entwicklung einer KI in Rennspielen hat, folgt der praktische Teil dieser Arbeit. Als Erstes wird das Prinzip des eigens entwickelten Ansatzes erklärt und welches Ziel dieser verfolgt. Als Nächstes folgt der Aufbau und die Umsetzung des eigenen Ansatzes. Hier erfolgt eine schrittweise Erklärung der praktischen Umsetzung. Verwendete Komponenten, wie Unity oder eine MySQL Datenbank zur Speicherung der Wegpunkte, werden nacheinander vorgestellt und genauer beschrieben.

Des Weiteren werden Schwierigkeiten und Probleme genannt und Lösungsansätze besprochen. Interessante Aspekte in der Implementierung der Künstlichen Intelligenz werden präsentiert und erklärt. Für die Umsetzung dieses Systems werden Rennlinien in Form von Wegpunkten, welche im Abschnitt 4.4.2 erklärt werden, gespeichert. Durch das Fahren einer Strecke und dem Erreichen des Zieles wird solch eine Rennstrecke in eine Datenbank gespeichert. Zusätzlich wird die Rennstrecke in Streckenabschnitte unterteilt. Diese Unterteilung erfolgt meist auf geraden Rennstreckenabschnitten um das Kombinieren mehrerer Abschnitte verschiedener Rennlinien leichter bzw. möglich zu machen. Wenn man nun mehrere Rennlinien in der Datenbank gespeichert hat, erfolgt der wichtigste Part dieser Arbeit. Im Abschnitt 4.4.3 soll mithilfe eines eigens entwickelten Tests ermittelt werden, ob es möglich ist Rennlinien so zu kombinieren, dass sie eine bessere Rennlinie ergeben. Die Kombination dieser Rennlinien erfolgt dabei zum einen durch ein Mitteln zweier bzw. mehrerer Rennlinien (also der Wegpunkte) oder durch die Kombination von Abschnitten verschiedener Rennlinien. Sowohl bei der gesamten Rennlinie als auch bei Abschnitten spielt die Zeit, in der die Strecke zurückgelegt wird, die entscheidende Rolle. Dies wird dann auch letztendlich als Bewertung für die neu entstandenen Rennlinien hergenommen.

Dies alles führt dann zu einem Ergebnis des praktischen Teils, welcher abschließend im Kapitel 5 erläutert wird. Hier wird sich nun zeigen ob es möglich ist eine KI durch die vorhin genannten Methoden (Mitteln oder Kombinieren mehrerer von Spieler kommenden Rennlinien) zu verbessern. Abschließend erfolgt ein Vergleich und eine kritische Betrachtung des erarbeiteten Ansatzes. Hier werden Vor- und Nachteile ausgearbeitet. Gründe für oder gegen die Verwendung des getesteten Ansatzes werden unter die Lupe genommen. Dies erfolgt im Kapitel 6. Im letzten Abschnitt 6.3 wird noch auf mögliche Verbesserungen und Erweiterungsmöglichkeiten des getesteten Systems eingegangen.

Kapitel 2

Künstliche Intelligenz in Spielen

Dieses Kapitel behandelt die Künstliche Intelligenz von Spielen. Begonnen wird mit einer allgemeinen Einführung von Künstlicher Intelligenz in Spielen. Hier wird beschrieben, was eine Künstliche Intelligenz in Spielen ausmacht und wozu man sie benötigt. Danach wird auf den Aufbau einer Künstlichen Intelligenz eingegangen. Dies erfolgt dabei anhand des Schemas von *Ian Millington*. Nach dieser allgemeinen Einführung wird im letzten Abschnitt dieses Kapitels das Rennspielgenre behandelt. Hier wird auf die Entwicklung dieses Genres mit Beispielen eingegangen und mögliche Arten von Rennspielen aufgezählt und beschrieben.

2.1 Einführung

Wenn man von einer Künstlichen Intelligenz spricht, denken viele an die Künstliche Intelligenz in der Wissenschaft. Obwohl man bei der Entwicklung einer KI in Spielen auf Methoden und Techniken der KI in der Wissenschaft zurückgreift, unterscheidet sie sich doch stark von dieser. Letztendlich geht es doch bei computergesteuerten Gegnern darum, dass sie dem Spieler die Illusion einer gewissen Intelligenz geben. Sie sollen den Spieler herausfordern, aber zu guter Letzt trotzdem bezwingbar sein. Des Weiteren soll sie im Vergleich zur wissenschaftlichen KI nicht perfekt sein, sondern menschliches Verhalten widerspiegeln, was das Machen von Fehlern beinhaltet. Lars Lidén fasst dies wie folgt zusammen [5]:

What makes a game entertaining and fun does not necessarily correspond to making its computer-controlled opponents smarter. The player is, after all, supposed to win. As long as the player has the illusion that a computer-controlled character is doing something intelligent, it doesn't matter what AI (if any)

was actually implemented to achieve that illusion.

Um diese Illusion zu erreichen, darf man in der Entwicklung der KI auf verschiedenste Tricks zurückgreifen. Beispielsweise darf bzw. kann eine KI Dinge und Gegner sehen, die für den normalen Spieler nicht sichtbar wären. Wichtig dabei ist nur, dass der Spieler nicht das Gefühl haben darf, dass seine computergesteuerten Gegner *cheaten* – also mehr wissen als der Spieler selbst. Oft reichen dabei einfache Implementierungen, um ein komplexes Verhalten zu simulieren. Ian Millington drückt dies wie folgt sehr gut aus [4]:

Knowing when to be complex and when to stay simple is the most difficult element of the game AI programmer's art. The best AI programmers are those who can use a very simple technique to give the illusion of complexity.

Dies spielt bei der Entwicklung einer KI eine große Rolle, da in der Entwicklung von Spielen wenig Budget und Rechenzeit für solch eine KI zur Verfügung stehen. So muss man in gewissen Situationen auf vereinfachte Methoden zugreifen, um die Anforderungen des Spieles letzten Endes zu reduzieren. Dies kann so weit führen, dass einem computergesteuerten Gegner nicht in jedem Frame Rechenzeit zur Verfügung gestellt wird. Unter anderem kann eine einfache Implementierung im Gegensatz zu einer komplexeren ein besseres Verhalten der KI erzielen, da diese dafür aufgrund der geringeren benötigten Rechenzeit in jedem Frame ausführbar ist.

Damit nun ein Spieler seine Freude an einem Spiel mit computergesteuerten Gegnern hat, muss sie wie folgt sein:

- Sie muss schlagbar sein.
- Sie muss herausfordernd sein.
- Ihr Verhalten muss für den Spieler plausibel sein.
- Sie muss bis zu einem gewissen Grad vorhersehbar sein.
- Sie muss dem Spieler eine gewisse Reaktionszeit auf ihr Verhalten geben.
- Sie darf *cheaten*, solange dies vor dem Spieler verborgen bleibt.
- Sie darf nicht zu dumm bzw einfach sein.
- Sie sollte auf selbst gemachte Fehler reagieren.
- Sie sollte sich konsistent verhalten.

Ein paar der aufgezählten Eigenschaften wurden bereits vorher genannt, manche sind jedoch neu und sollen hier nochmals kurz beschrieben werden. Damit der Spieler seinen Spaß mit computergesteuerten Gegnern haben kann, muss sie bis zu einem gewissen Grad vorhersehbar sein. Wenn man gegen andere Spieler spielt, sind diese doch ebenfalls teilweise vorhersehbar.

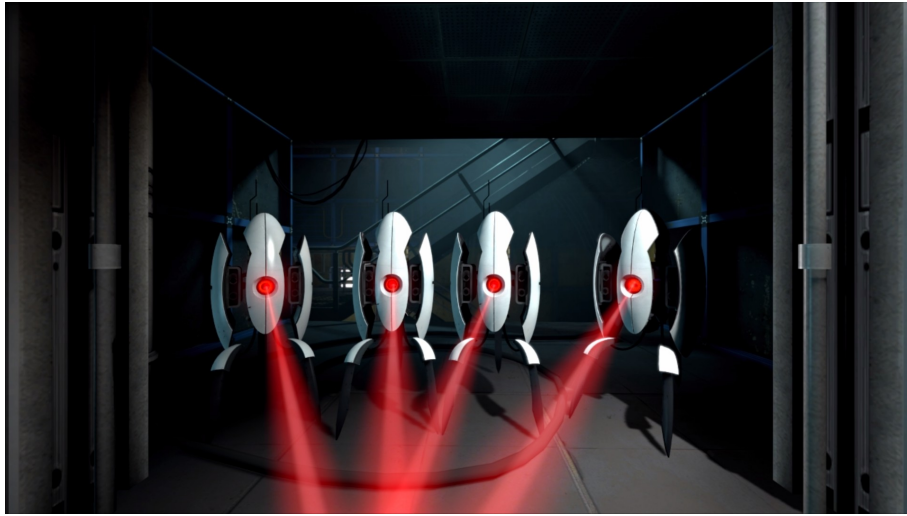


Abbildung 2.1: Das Anvisieren der *Turrets* mittels Laser gibt den Spieler die gewünschte Reaktionszeit [9].

Dies spielt stark mit einem konsistenten Verhalten der Gegner zusammen. Je öfters man gegen den gleichen Spieler spielt, desto besser kennt man ihn und kann gegen ihn Strategien entwickeln. Dasselbe gilt für computergesteuerte Gegner. Der Spieler sollte sich teilweise darauf verlassen können, dass sich beispielsweise Wachen in einem *Shooter* zumindest ähnlich verhalten. So werden sie meistens, wenn sie einen Gegner entdecken erstmals Deckung suchen, bevor sie weitere Aktionen durchführen. Des Weiteren muss die KI dem Spieler eine Reaktionszeit auf ihr Verhalten geben. Dies bedeutet, dass eine KI oftmals zu erkennen gibt, ob und beziehungsweise was sie als Nächstes tun wird. Dies ist natürlich je nach Spiel oder Spielgenre unterschiedlich stark ausgeprägt. Als Beispiel folgt ein Bild aus *Portal 2*. Die *Turrets* zeigen mittels ihres Lasers an, ob sie den Spieler entdeckt und im Visier haben und geben den Spieler zusätzlich noch etwas Zeit, bevor sie zu attackieren beginnen.

Dabei handelt es sich um eine windup animation, welche von Matt Gilgenbach im Buch *AI Game Programming Wisdom 2* [5] beschrieben wird. Abbildung 2.2 welche ebenfalls aus diesem Buch stammt zeigt dies.

Ein weiterer wichtiger Punkt, der oben aufgezählt wurde ist, dass die KI auf selbst gemachte Fehler reagieren soll. Dies ist ein wichtiger Punkt, um dem computergesteuerten Gegner intelligent erscheinen zu lassen. Sollte ein gegnerisches Rennauto aus irgendeinem Grund von der eigentlichen Rennstrecke abkommen, so sollte diese versuchen den Fehler zu beheben und wieder versuchen auf die vorgesehene Rennstrecke zurückzukommen. Ein anderes Beispiel wäre wenn eine von einem Soldaten geworfene Granate, die an einem Objekt abprallt und zurückfliegt. In so einem Fall sollte die KI

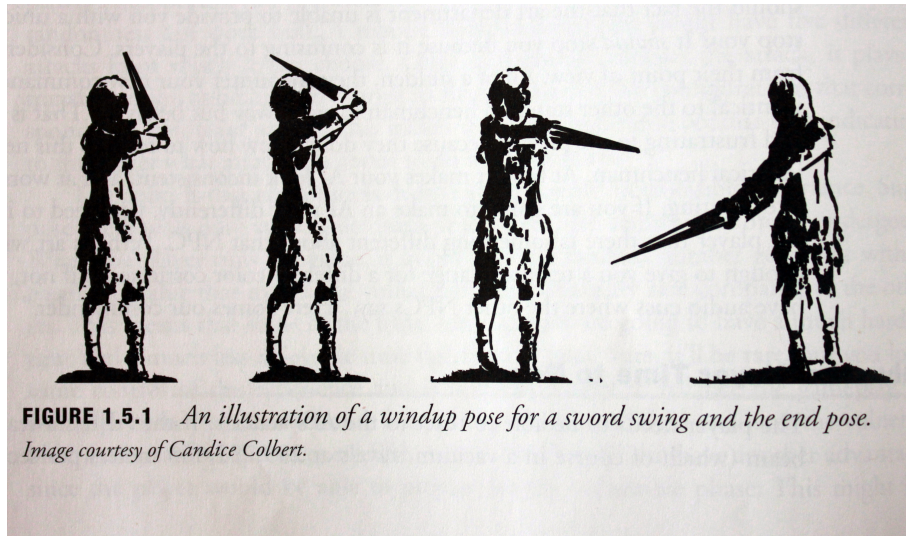


Abbildung 2.2: Windup animation [5].

nicht einfach neben der Granate stehen bleiben sondern auf diese reagieren und Deckung suchen.

2.2 Aufbau einer Künstlichen Intelligenz

In den Anfängen war die KI in Spielen sehr einfach und man musste sich deshalb noch wenig Gedanken über den Aufbau eines solchen Gegners machen. So mussten sich Gegner in Pacman nur fortbewegen können. Dabei hatten sie zwei unterschiedliche Stadien – Verfolgen und Flüchten. Bei jeder Abzweigung wird mit einer bestimmten Wahrscheinlichkeit entschieden, ob sich der Geist auf den Spieler zubewegen soll oder nicht. Diese Wahrscheinlichkeiten ändern sich in Abhängigkeit des Stadiums in dem sie sich befinden. Noch heute wird versucht die KI so einfach wie möglich zu halten, doch meist sind sie trotzdem um einiges komplizierter aufgebaut als die Geister in Pacman. Ein bekanntes und gängiges Model für den Aufbau einer KI befindet sich im Buch *Artificial Intelligence for Games*[4] von *Ian Millington*. Abbildung 2.3 zeigt wie Ian Millington eine KI prinzipiell aufbaut.

Millington teilt die Aufgaben einer KI in 3 große Layer ein:

1. *Strategy*,
2. *Decision Making* und
3. *Movement*.

Je nach Spielgenre und Computergegner sind diese Layer in einer KI unterschiedlich ausgeprägt. Der *Strategylayer* beschäftigt sich mit Entscheidungen, welche einer Gruppe von Computergegnern betrifft. In einem Stra-

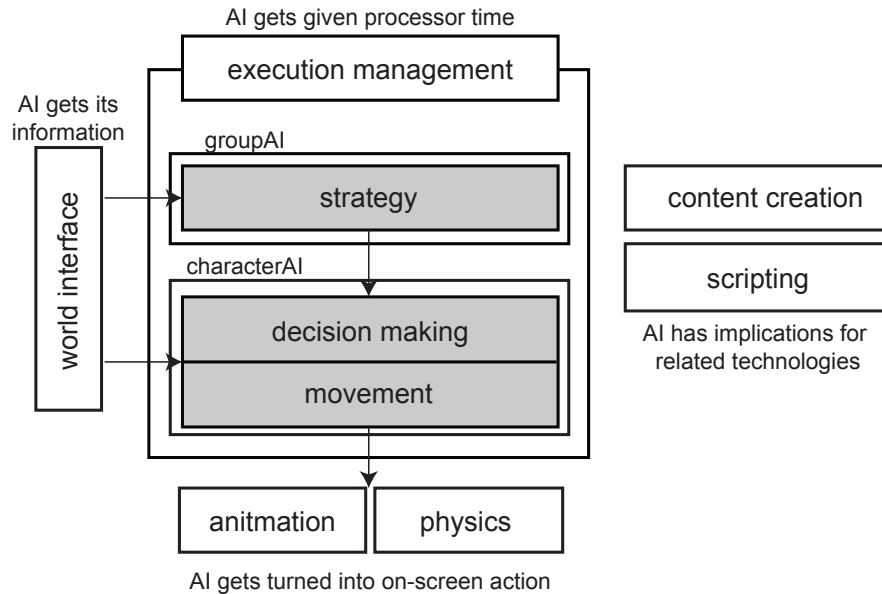


Abbildung 2.3: AI Model nach Ian Millington [4].

tegiespiel sollte beispielsweise nicht jede Einheit für sich entscheiden, ob sie den Spieler angreifen soll oder nicht. So etwas sollte für eine gesamte Gruppe entschieden werden. Dies sorgt auch für eine Einsparung der Rechenzeit, die meist ohnehin für die KI in Spielen knapp bemessen ist. Neben dem *Strategylayer* gibt es nun auch den *decision making Layer*, welcher nun nicht mehr eine gesamte Gruppe, sondern nur eine einzelne KI oder Einheit betrifft. Wenn man von Künstlicher Intelligenz in Spielen spricht, denkt man dabei meist an diesen Layer. Er ist nun für die Entscheidungen die eine KI für sich trifft zuständig. Wenn es sich gerade nicht um ein digitalisiertes Brettspiel handelt wird dieser meistens benötigt. So entscheidet hier beispielsweise eine KI in einem *shooter* ob sie bei Verlust von Leben weiterkämpfen oder vielleicht doch fliehen soll. Sogar in Rennspielen wird diese Schicht zum Treffen von Entscheidungen benötigt. Ein computergesteuertes Fahrzeug kann so Entscheiden ob und wann es zum Überholen ansetzen soll. Oder auch, ob es sich um einen guten Moment handelt um ein anderes Fahrzeug abzudrängen. Der letzte und zumeist wichtigste Layer dient zur Bewegung der KI und der Ausführung der Entscheidungen die im *decision making* beschlossen wurden. In Rennspielen handelt es sich hierbei um den wichtigsten und am meist ausgeprägtesten Layer. Möglichkeiten wie sich die KI in Rennspielen fortbewegen kann werden im Kapitel [refchap:State of the Art](#) genauer behandelt.

Wie man nun eine KI aufbaut, bleibt letzten Endes dem Entwickler über-

lassen. Je nach Komplexität der KI ist es ratsam sich im Vorhinein schon Gedanken über den Aufbau zu machen. Das Layermodel von Ian Millington kann hierbei ein guter Ausgangspunkt sein.

2.3 Rennspiele und Genres

Was alles unter den Begriff Rennspiel fällt beschreibt Wikipedia wie folgt [10]:

racing video game is a genre of video games, either in the first-person or third-person perspective, in which the player partakes in a racing competition with any type of land, air, or sea vehicles. They may be based on anything from real-world racing leagues to entirely fantastical settings. In general, they can be distributed along a spectrum anywhere between hardcore simulations, and simpler arcade racing games. Racing games may also fall under the category of sports games.

Rennspiele waren schon von Beginn an ein Teil der Videospielwelt. Bereits in den 70iger Jahren gab es für den Atari (Abbildung 2.4) ein Rennspiel namens *Space Race*. Folgendes Zitat zeigt auch warum [11]:

It makes perfect sense that racing games have been a part of the video game phenomenon from the very start. Racing, be it in a car or on a motorcycle, is pure visceral excitement, and that's the same rush to which gamers are addicted, and have been since the advent of the coin-op arcades and the dawning of the Atari Age.

Ein weiterer Schritt in der Entwicklung von Rennspielen war das Spiel *Moto-Cross* von Sega, welches eines der Ersten mit einer 3D 3rd Person Ansicht war. Im Allgemeinen tendierten in den 80er und 90er Jahren die Rennspiele mehr in Richtung *Arcade Games*. Hauptgrund war die eingeschränkte Möglichkeit Simulationen zu entwickeln aufgrund der zu schwachen Hardware dieser Zeit. Ein Spiel, welches jedoch zu dieser Zeit schon in Richtung einer Simulation ging, war *REVS* von *Geoff Crammond*. Ein weiterer wichtiger Aspekt in der Entwicklung von Rennspielen kam mit der Einführung des *Rubberbanding* in *Final Laps*. In dem Artikel *Learning to Love Handicaps in Competitive Games* von *Keith Burgun* wird *Rubberbanding* wie folgt beschrieben:

A system of rewarding poor play and punishing good play, creating an effect of bringing all players to the center.

In Rennspielen kann dies beispielsweise durch ein Erhöhen der maximalen Geschwindigkeit der zurückgefallenen Spieler erreicht werden. Damit soll



Abbildung 2.4: Version 3 des Atari [12].

wie im Abschnitt 2.1 genannt für gute Spieler weiterhin eine Herausforderung bestehen, auch wenn dies durch unfaire Mittel geschieht. Ein sehr gutes Beispiel für *Rubberbanding* ist *Mario Kart*. Der Spieler kann hier durch den Einsatz verschiedenster Items den Spielablauf beeinflussen – andere Spieler behindern oder sich selbst vorantreiben. Abbildung 2.5 zeigt ein Item das mit höherer Wahrscheinlichkeit der letzte Spieler in einem Rennen erhält. Dieser Gegenstand ist einfach zu verwenden und hält den ersten Spieler für einen kurzen Moment auf und lässt somit alle anderen Spieler wieder näher ran kommen. Somit fördert dieses Item schlechte Spieler und bestraft Gute, um alle Spieler wieder näher zusammen zuführen.

2.3.1 Genres

Hier ein kurzer Überblick welche Arten von Rennspielen es gibt mit Beispielen. Rennspiele kann man prinzipiell in zwei große Kategorien unterteilen:

- Arcade Rennspiele und
- Rennspiel Simulationen.

Arcade Spiele unterscheiden sich zu den Rennspiel Simulationen vor allem durch ihre tolerante Physik und den schnelleren Erfahrungsanstieg des Spielers. Das heißt, dass Spieler sich bei einem Arcade Games schneller eingewöhnen und folglich schneller Erfolge erzielen. *Powerslides* um die hohen Geschwindigkeiten in Kurven beibehalten zu können und übertriebene Kollisionen mit Hindernissen und anderen Fahrzeugen sind nur Beispiele bei dieser Art von Spielen um den Spaßfaktor zu erhöhen. *Mario Kart*, das vor-



Abbildung 2.5: *Rubberbanding* in Form eines *Stachi* Panzers im Spiel *Mario Kart*.

hin genannte Beispiel zu *Rubberbanding*, ist ein gutes Beispiel für das Arade Genre.

Im Gegensatz zu den Arcade Spielen versuchen Rennspiel Simulationen möglichst nah an die Realität ran zukommen. Hier spielen Aspekte wie die Abnützung der Reifen, Tankfüllung, Straßenverhältnisse und die technischen Eigenschaften des Fahrzeuges wie die Federung eine erhebliche Rolle. Diese Spiele fordern dementsprechend mehr Erfahrung vom Spieler und sind daher schwieriger zu bewältigen. Genau diese Herausforderung stellt für Spieler einen Ansporn dar. So gilt der im Abschnitt 2.1 genannte Punkt, das eine KI eine Herausforderung darstellen soll, nicht nur für computergesteuerte Gegner sondern auch weiterführend für das Spiel selbst.

Zur Vollständigkeit werden hier noch die Subgenres aufgezählt, kurz beschrieben und mit einem Beispielspiel angeführt [10]:

1. Off-road,

Simulation-style racing games: Spiele, die auf Realitätsnähe abzielen
– *Formula one series*, *Assetto corsa*.

Semi-simulation-style racing games: Ein Kompromiss aus Simulation und Arcade – *F1 20012*, *Gran Turismo Series*.

Arcade-style racing games: Typischen Arcade Spiele – *Trackmania Series*, *Burn Out Series*.

Street racing games: Straßenrennsportspiele zumeist in Großstädte – *Midnight Club Series*, *Rush Series*.

2. Sports games,

Event racing: Rennspiele an Sportevents angelehnt – *Amped Series*,

ATV Series.

Exaggerated sports racing: Rennspiele die sich an den Extremsport halten – *Trials Evolution*, *Trackmania*.

Monster truck racing games: *Monster Truck Madness Series*, *Monster Jam Series*.

Truck racing games: *Baja: Edge of Control*, *Euro Truck Simulator*.

Motorcycle racing games: *MotoGP 07*, *Tourist Trophy*.

Kart racing games: *Mario Kart*, *Crash Team Racing*.

3. Water racing games,

Jetski racing games: *Carve*, *Splashdown*.

4. Off-road racing games,

Baja: Edge of Control, *Fuel*.

5. Stunt racing,

Re-Volt, *Rush*.

6. Futuristic racing games,

Super Sonic Racer, *Wipeout 2048*.

7. Racing role playing games,

Need for Speed: World, *Forza Motorsport*.

8. Mission- based racing games und

Zombie Driver, *Cracy Taxi Series*.

9. Vehicular combat games

Zombie Driver, *Cracy Taxi Series*.

Wie diese Liste veranschaulicht gibt es eine Vielzahl an Rennspielarten mit dazugehörigen Beispielen. Ebenfalls der Liste zu entnehmen ist, dass Rennspiele nicht immer eindeutig zu zuordnen sind, sondern auch verschiedenste Genres kombinieren können. So gehört *Crash Team Racing* (oben angeführt unter *kart racing games*) auch dem *futuristic racing game* und *vehicular combat game* Genre an.

Kapitel 3

State of the Art

In diesem Kapitel wird die Künstliche Intelligenz in Rennspielen genauer behandelt. Es wird beschrieben welche Möglichkeiten es zum Einbau einer KI in Rennspiele gibt und welche Vor- und Nachteile diese mit sich bringen.

Bei der Entwicklung einer Rennspiel KI ist der *Movement Layer* von zentraler Bedeutung. Abbildung 3.1 zeigt wo genau sich dieser im Aufbau einer KI nach *Ian Millington* befindet.

Dieser *Movement Layer* wird im ersten Unterkapitel genauer beschrieben, bevor in den nächsten Unterkapitel auf Möglichkeiten zur Steuerung

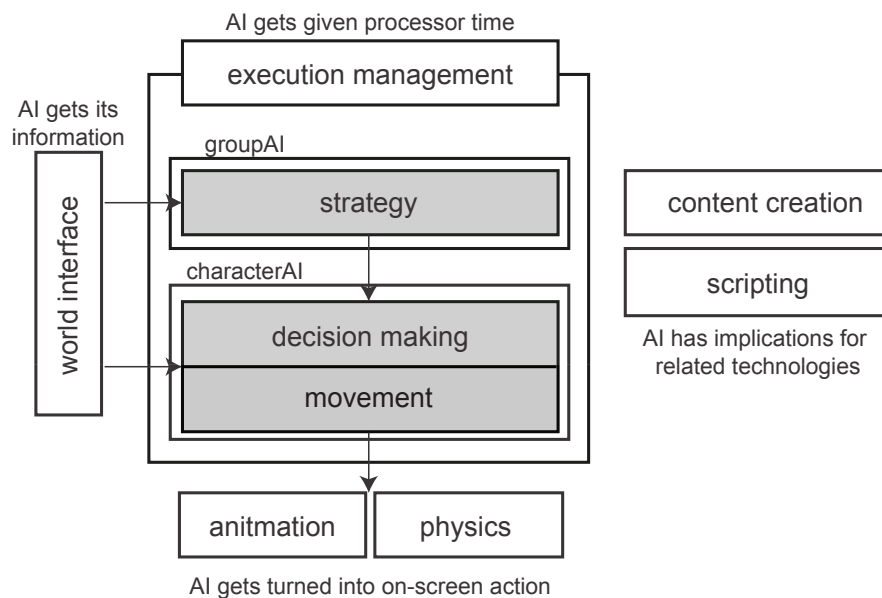


Abbildung 3.1: *Movement Layer* des AI Models [4].

einer Rennspiel KI eingegangen werden.

Auch wenn in diesem Kapitel Implementierungsmöglichkeiten und weit verbreitete Methoden für die Entwicklung einer KI in Rennspiele vorgestellt werden, sollte man dabei immer im Hinterkopf behalten, dass auch wenn sie nicht eine fehlerfreie Lösung bieten, letztendlich folgenden Zweck zu erfüllen haben [1]:

A hugely clever and almost unbeatable opponent is rarely the goal of gameplay fun. You will be wise to remind yourself of this from time to time because, believe me, it is very easy to miss the point and get bogged down with trying to make the most intelligent game agent known to man instead of one that makes a player laugh and shout with glee.

3.1 Movement

In diesem Unterkapitel wird auf die Bedeutung des *Movement Layers* für Rennspiele und auf die dafür wichtigsten Algorithmen eingegangen.

Prinzipiell dient der *Movement Layer* einer KI zur Umsetzung einer im *Decision Layer* beschlossenen Sache und ist somit das ausführende Element in diesem Konzept. Dabei muss es nicht nur um die Fortbewegung eines *NP-Cs* (*non player character*) gehen, sondern auch um Animationen die dazu abgespielt werden müssen. Als Beispiel kann hier ein Rennauto nach einem Crash dienen. Dieses Rennauto sollte nach einem Unfall nicht nur weiterfahren, sondern auch noch zusätzlich einen kaputten Reifen mithilfe einer Animation anzeigen. So ein beschädigter Reifen könnte dann durch ein Wackeln oder ein Aufsteigen von Rauch dargestellt werden. Abbildung 3.2 zeigt die Korrelation zwischen den Layern *movement*, *animation* und *physics*. So werden aufgrund von Informationen des *Movement Layer* unterschiedliche Aktionen in der Physik- und Animationsschicht getätigt.

Wie in der Abbildung zu sehen ist, spielt die Physik mit dem *Movement Layer* ebenfalls eng zusammen. Denn bei Physik basierenden Spielen muss die KI über das Wirken von Kräften in Bewegung versetzt werden. Durch die Verwendung von Physik-Engines hat sich die Verwendung einer kinematischen Bewegung in Rennspielen reduziert und sich dabei die Nutzung einer dynamischen Bewegung erhöht. Die dynamische Bewegung berücksichtigt die momentane Bewegung eines *NPC* und lässt anhand dieser eine Kraft so wirken, dass der computergesteuerte Gegner sich seinem Ziel nähert bzw. dieses Ziel erreicht.

Durch diese Art der Fortbewegung erhält man auch die natürlichen *ease-in* und *ease-out* Bewegungen. Die Schwierigkeit der dynamischen Bewegung gegenüber der kinematischen Bewegung wurde von Ian Millington folgendermaßen beschrieben [4]:

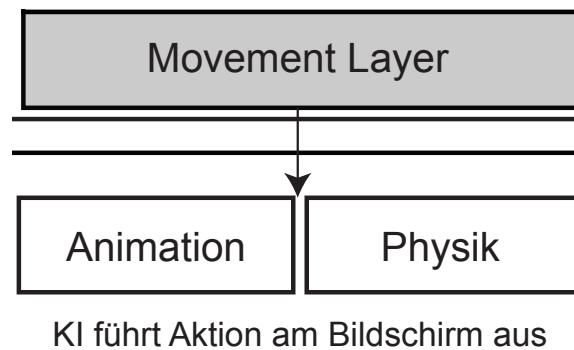


Abbildung 3.2: Korrelation zwischen Movement, Physik und Animation [4].

Dynamics adds an extra layer of complexity. Let's say your character needs to move from one place to another. A kinematic algorithm simply gives the direction to the target, you move in that direction until you arrive, whereupon the algorithm returns no direction: you've arrived. A dynamic movement algorithm needs to work harder. It first needs to accelerate in the right direction, and then as it gets near its target, it needs to accelerate in the opposite direction, so its speed decreases at precisely the correct rate to slow it to a stop at exactly the right place.

Damit sich eine KI von einem Punkt zu einem anderen bewegen kann, muss sie sowohl Informationen über sich selbst, (wie zum Beispiel Position und Geschwindigkeit) als auch Informationen über ihre Umgebung (Position des Zieles, Pfade zum Ziel, Hindernisse, andere Charaktere) beziehen. Zu diesen Informationen gehören auch der Einschlag des Autos und ob der Spieler bzw. die KI Gas gibt oder bremst. Diese Informationen kommen beim Fahrzeug des Spielers direkt über den Input der Tastatur. Kapitel 4 beschreibt anschließend woher diese Informationen für einen computergesteuerten Gegner kommen. Abbildung 3.3 zeigt eine schematische Darstellung des vorhin beschriebenen Zusammenhangs zwischen den zu bewegenden Charakter und den dafür nötigen Informationen.

Craig Reynolds ist mit der Entwicklung seiner steering behaviors[14] einer der wichtigsten *keyplayer* in diesem Bereich. Zu unterscheiden sind hierbei steering behaviors zur Steuerung einzelner Charaktere oder Paare und Algorithmen, die zur Steuerung einer ganzen Gruppe von Charakteren dienen. Je nach Rennspielgenre sind diese Algorithmen in unterschiedlichen Ausmaß einsetzbar. Soll beispielsweise ein Charakter sich auf einen Punkt zubewegen, so könnte man den Seek- Algorithmus verwenden. Um

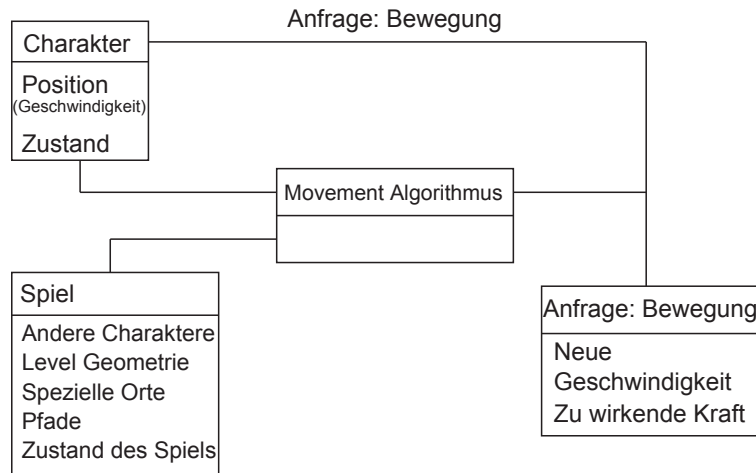


Abbildung 3.3: Struktur des Movement Algorithmus.

eine Gruppe von computergesteuerten Fahrzeugen einen Spieler folgen zu lassen, würde sich der Leader Following Algorithmus anbieten.

3.2 PID-Regler

Eine Möglichkeit zur Steuerung von Objekten oder Agenten in Computerspielen stellt der PID-Regler dar. Dieser Regler besteht dabei aus folgenden drei Teilen:

- Proportionalanteil,
- Integralanteil und
- Differentialanteil.

Der PID-Regler dient zur Regelung, also zur Erreichung und Erhaltung einer bestimmten Eigenschaft oder eines Wertes. Dies erreicht er mit Hilfe des momentan bestehenden Wertes (Ist- Wert) und dem zu erreichenden Wertes (Soll- Wert). Abbildung 3.4 veranschaulicht den grundlegenden Aufbau eines Reglers. Somit handelt es sich beim Ist- Wert und Soll- Wert um Eingangsgrößen und bei der Stellgröße um die Ausgabe des Reglers.

Euan Forrester fasst dies wie folgt zusammen[6]:

PID controllers are feedback-based algorithms that operate on a process by controlling an output variable to minimize the difference between a measured value and a desired value.

Diese oben genannte Differenz bezeichnet man als Regelfehler e und wird folgendermaßen errechnet 3.1:



Abbildung 3.4: Grundlegender Aufbau eines Reglers.

$$e(t) = \text{Ist- Wert}(t) - \text{Soll- Wert}(t). \quad (3.1)$$

Das heißt, dass sich der Fehler durch die Differenz aus dem zu erreichenden Wert (Soll- Wert) und dem momentan gemessenen Wert (Ist- Wert) errechnet.

3.2.1 P-Regler

Der Proportionalregler stellt dabei den einfachsten Ansatz dar. Bei diesem Regler handelt es sich bei der Eingangsgröße des Reglers um die Regelabweichung und bei der Ausgangsgröße um die Stellgröße. Folglich erzeugt der Regler eine zur Regelabweichung proportionale Stellgröße. Der P-Regler beschäftigt sich somit nur mit dem momentanen Unterschied zwischen dem Ist- Wert und dem Soll- Wert. Dies bedeutet, dass eine große Abweichung zwischen dem Ist- Wert und dem Soll- Wert (Regelfehler 3.1) eine große Stellgröße zur Folge hat. Als Einstellungsmöglichkeit bietet dieser Regler nur einen Verstärkungsfaktor c_P . Die Stellgröße y errechnet sich dabei folgendermaßen 3.2:

$$y(t) = c_P \cdot e(t). \quad (3.2)$$

Mit einem höheren Verstärkungsfaktor nähert sich der Regler den Soll- Wert schneller, wenn auch dieser nie genau erreicht wird. Eine gleichbleibende Genauigkeit und eine vollständige Ausregelung einer Störung kann mit dem Proportionalregler nicht erzielt werden [3]. Abbildung 3.5 zeigt den Verlauf des gemessenen Wertes bei einem zu niedrigen und zu hohen Verstärkungsfaktor.

3.2.2 PI-Regler

Um die Nachteile des P-Reglers ausgleichen zu können, kann man auf den PI-Regler zurückgreifen.

Bei diesem Regler handelt es sich um eine Kombination aus einem P-Regler und einem I-Regler. Der PI-Regler ist einer der am häufigsten verwendete Regler. Der I-Regler wirkt durch eine zeitliche Integration der Regelabweichung (also des Regelfehlers) auf die Stellgröße mit einer Gewichtung

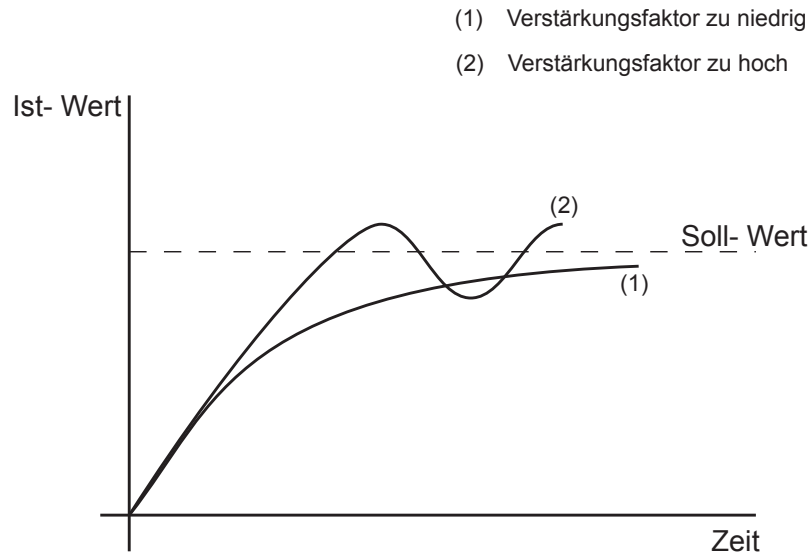


Abbildung 3.5: P-Regler mit zu hohen und zu niedrigen Verstärkungsfaktor.

durch die Nachstellzeit T_N ¹. Formel 3.3 zeigt den PI-Regler bestehend aus dem Teil des P-Reglers und dem Teil des I-Reglers.

$$y(t) = c_P \cdot e(t) + c_I \cdot \int_0^t e(t) \cdot dt. \quad (3.3)$$

Der Verstärkungsfaktor c_I ergibt sich dabei aus dem Kehrwert von T_N 3.4.

$$c_I = \frac{1}{T_N}. \quad (3.4)$$

Der I-Regler für sich ist ein langsamer Regler, verfügt aber über eine hohe Genauigkeit. Abbildung 3.6 zeigt eine Sprungantwort des I-Reglers. Der Eingangssprung $e(t)$ beträgt 1. Bei einem konstanten Soll- Wert führt dies zu einem linearen Anstieg des Ist- Wertes. Die Nachstellzeit T_N gibt dabei an in welcher Zeit der Ist- Wert bei gleichbleibenden Soll- Wert erreicht werden soll. Der PI-Regler verfügt somit über die positiven Eigenschaften des jeweils einzelnen Reglers. Die Genauigkeit des I-Reglers und die Schnelligkeit des P-Reglers.

¹<http://de.wikipedia.org/wiki/Regler>

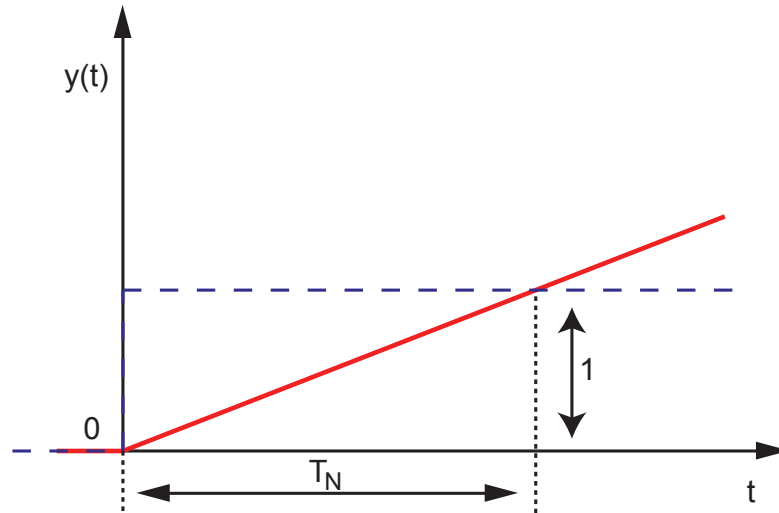


Abbildung 3.6: Sprungantwort eines I-Reglers.

3.2.3 PID-Regler

Der PI-Regler verwendet den momentanen Unterschied zwischen dem Ist-Wert und den Soll-Wert (P-Regler), als auch die vergangenen Regelfehler (I-Regler). Mit dem PID-Regler kommt nun auch noch eine dritte Komponente hinzu. Mithilfe des D-Reglers soll eine Regelabweichung bereits beim Entstehen bekämpft werden. Deshalb wird beim PID-Regler noch ein Differenzialanteil hinzugefügt. Formel 3.5 zeigt den PID-Regler, welcher sich im letzten Teil zur Formel des PI-Reglers 3.3 unterscheidet:

$$y(t) = c_P \cdot e(t) + c_I \cdot \int_0^t e(t) \cdot dt + c_D \cdot \frac{de(t)}{dt}. \quad (3.5)$$

3.3 Implementierungsmöglichkeiten einer künstlichen Intelligenz in Rennspielen

3.3.1 Steuerung der KI durch Kollisionserkennung

Jung-Ying Wang und Yong-Bin Lin beschreiben, wie eine KI mit Hilfe von Kollisionserkennung eine Rennstrecke durchfährt [8]. Hierfür werden wie Abbildung 3.7 zeigt zwei Kollisionserkennungspunkte vor dem Auto platziert. Der Algorithmus benötigt, um die KI durch die Rennstrecke zu steuern, dabei zwei Parameter. Einer enthält die Distanz zwischen Auto und Kollision der Kollisionserkennungspunkte und der andere beinhaltet den Wendekreis. Durch die Kollision von einem der beiden Kollisionspunkte wird erkannt ob

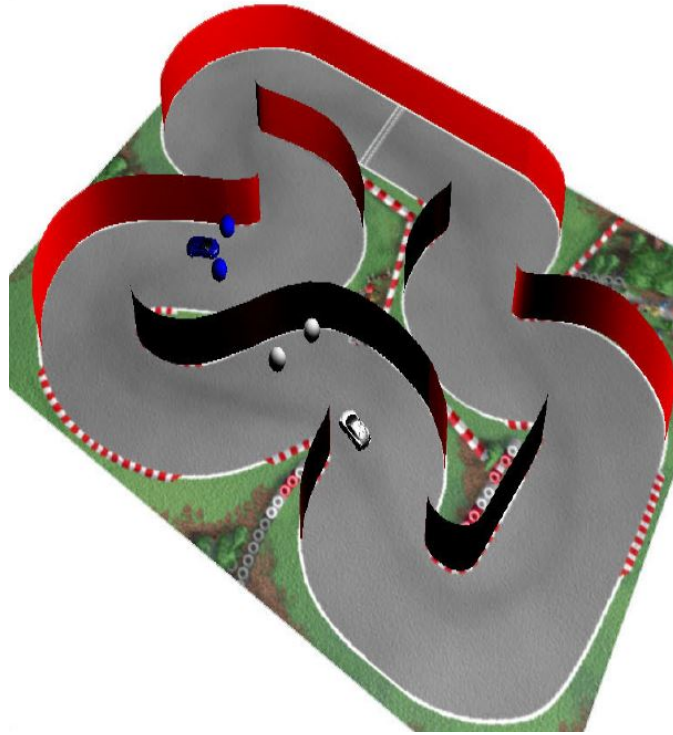


Abbildung 3.7: Kollisionserkennungspunkte des Fahrzeuges [8].

es sich um eine Links- oder Rechtskurve handelt und das Auto kann somit um die Kurve fahren. Für den Fall, dass beide Punkte zugleich eine Kollision auslösen werden die Punkte in einem anderen Winkel zum Auto versetzt, um somit wieder auf eine Kollision zu kommen. Je nach Wertigkeit der beiden Parameter fährt das Auto nun schneller oder langsamer die Rennstrecke ab. Jung-Ying Wang und Yong-Bin Lin testeten den Algorithmus in einem vereinfachten Rennspiel. Das heißt, dass in diesem Spiel keine Physik zum Einsatz kam. Letztlich bietet dieses System Vorteile wie eine einfache Implementierung und die Einstellbarkeit des Fahrverhaltens durch die oben genannten Eingangsparameter, doch ist es auf Strecken mit sehr unterschiedlichen Kurven weniger geeignet.

3.3.2 Steuerung der KI durch Folgen von Pfaden oder Wegpunkten

Eine häufig genutzte Variante um eine KI durch eine Rennstrecke zu bringen ist das Definieren von Pfaden bzw. Wegpunkten, an die sich die KI so gut wie möglich zu halten versucht. Dies ermöglicht, dass die KI von der Spur abgedrängt werden kann und sich daraufhin trotzdem weiterhin plausibel verhält. Diese Rennlinien müssen noch meist von Hand gesetzt und optimiert

werden. Loiacono fasst dies in seinem Paper wie folgt zusammen [2]:

In commercial games, the target racing lines are usually drawn by domain experts and then tested and tuned by game developers through actual game-play. While the problem of computing the optimal racing line is well studied in the real car racing domain, so far, only few works focused on the problem of optimizing the racing lines to speedup the development of non-player characters in racing games.

Das Ansteuern dieser Wegpunkte kann dann beispielsweise durch den seek Algorithmus oder durch einen PID Regler erfolgen. Dieser errechnet die Richtung zum Ziel (ausgehend vom Charakter/ Auto) und erhält zusätzlich noch eine Zielgeschwindigkeit, mit welcher er seinen nächsten Wegpunkt erreichen soll. Handelt es sich bei der vordefinierten Strecke nicht um eine bestimmte Anzahl Wegpunkte, sondern um eine Rennlinie die es zu folgen gilt, so kann der pathfollowing Algorithmus verwendet werden. Dieser Algorithmus errechnet sich einen anzusteuernenden Wegpunkt entlang der Linie.

Es gibt auch Versuche diese Pfade für eine Rennstrecke automatisch generieren zu lassen. C.I. Tan [7] versucht beispielsweise solche Pfade für ein Pferderennspiel automatisch generieren zu lassen:

User can define the race by providing a race course of 3D model and weights of the devised turn and heuristic functions in our system.

Dabei wird die Strecke in ein Raster unterteilt und die einzelnen Kacheln gewichtet. Anschließend werden mit Hilfe eines angepassten A*Algorithmus Pfade generiert, welche abschließend vom Benutzer angepasst werden können oder müssen. Ein großer Nachteil dabei ist, dass das System kreisförmige Kurven benötigt und somit für Strecken mit unterschiedlichen Kurvenformen unbrauchbar ist. Zusätzlich müssten die sogenannten Kampflinien, welche zum Überholen anderer Spieler dienen trotzdem von Hand gesetzt werden.

Kapitel 4

Eigener Ansatz

In vielen Fällen werden die Rennlinien einer Künstlichen Intelligenz durch den *Game Designer* von Hand gesetzt. Dieses Kapitel setzt sich nun mit der Möglichkeit der Generierung von Rennlinien aus gegebenen Benutzerdaten auseinander. Folgende Fragestellung soll dabei beantwortet werden:

Kann man die von Spielern gegebenen Rennlinien durch ein Mittel oder durch Kombination von Abschnitten verschiedener Rennlinien eine bessere Rennlinie für eine KI in Rennspielen erzeugen, sodass sie mit dem Anstieg der Erfahrung des Spielers auch weiterhin eine Herausforderung darstellt?

4.1 Prinzip

Mit dieser Arbeit soll versucht werden computergesteuerten Gegner in Rennspielen mehr Variation in ihrem Verhalten zu geben. Hauptaugenmerk liegt dabei auf die Variation im Fahrverhalten der computergesteuerten Gegner, also in der Anzahl an möglichen Spuren die der Agent fahren kann und das Darstellen einer Herausforderung auch nach Erfahrungsanstieg des Spielers. Das heißt, dass eine KI in Rennspielen beim mehrmaligen Befahren einer Rennstrecke eine Vielfalt an Möglichkeiten zur Verfügung hat und somit für den Spieler nicht immer gleich fährt. Des Weiteren sollte die KI durch die Sammlung von Spielerdaten, in Bezug auf ihre benötigte Zeit, besser werden um somit auf ein unfaires Verhalten verzichten zu können. Folglich sollte eine KI nicht auf Tricks zurückgreifen müssen, um mit dem Spieler mithalten zu können. Eine KI in Rennspielen soll einen menschlichen Spieler imitieren und deshalb auch unter den gleichen Bedingungen wie der Spieler fahren müssen. In Kapitel 2 wurde ein Betrügen der KI anhand von *Rubberbanding* bereits erklärt.

Durch Betrügen soll einer zu schlechten KI die Möglichkeit gegeben werden auch mit dem Anstieg der Erfahrung des Spielers weiterhin eine Her-

ausforderung darstellen zu können. Dies erfolgt beispielsweise durch eine Erhöhung der Maximalgeschwindigkeit der KI. Das kann dazu führen, dass ein Spieler durch einen kleinen Fehler im letzten Abschnitt der Rennstrecke gegen eine KI verliert, obwohl dieser den Rest der Strecke im Vergleich zur KI besser durchfuhr, da die KI bis zum letzten Abschnitt nur durch eine höhere Maximalgeschwindigkeit mithalten konnte.

Letztendlich sollte der Spieler im besten Fall vergessen können, dass es sich bei seinen Gegnern nur um computergesteuerte Gegner handelt. Das soll durch Kombinationen verschiedenster Rennlinien passieren. In dieser Arbeit erfolgt das Kombinieren durch ein Mitteln oder durch ein Unterteilen und Zusammensetzen zweier Rennlinien. Dies wurde gewählt um Fehler, welche von Spieler während der Fahrt gemacht werden, zu beseitigen. Kapitel 5 zeigt, dass ein Fehler, wie das Abkommen von der Straße, behoben werden kann. In Abbildung 4.1 werden hierbei die zwei ursprünglichen Rennlinien in Rot und Blau dargestellt. Im linken Ausschnitt wird ein Mittel aus den zwei Rennlinien durch eine violette Linie dargestellt und im rechten Bild ist eine mögliche Kombination aus den gegebenen Linien zu sehen.

4.2 Unity

Für die Umsetzung der Idee zur Kombination von Rennlinien wurde Unity3D hergenommen. Bei Unity handelt es sich um eine von *Unity Technologies* entwickelte GameEngine.

Ein nennenswertes Beispiel für ein in Unity entwickeltes Rennspiel ist *Drift Mania Championship 2*. Dieses Spiel wurde sowohl für PC (Windows und Mac OS X) als auch den mobilen Plattformen (Android, iPhone und iPad) entwickelt. Abbildung 4.2 zeigt ein geöffnetes Unity Projekt mit den für die Entwicklung eines Spieles wichtigsten Fenstern. Unity ist ein auf Komponenten basierendes System, was ein leichtes Anpassen von Objekten an geänderte Bedingungen zur Folge hat – dies erfolgt durch ein einfaches Hinzufügen einer weiteren Komponente auf das gewünschte Objekt. Abbildung 4.3 zeigt beispielsweise alle Komponenten, die sich auf dem computergesteuerten Fahrzeug befinden. Bei diesen Komponenten handelt es sich um Scripte, die meist eine ganz bestimmte Aufgabe zu erfüllen haben. Unity bietet hierbei die Möglichkeit diese Scripte in C#, JavaScript und Boo zu verfassen. Mitgrund für die Verwendung von Unity ist die gute Online Dokumentation¹ mit Anwendungsbeispielen in allen drei Sprachen.

4.3 Car Tutorial

Um mit der eigentlichen Umsetzung der KI zu beginnen, musste zuerst ein Grundgerüst – also ein einfaches Rennspiel erstellt werden. Dafür wurde auf

¹<http://unity3d.com/learn/documentation>

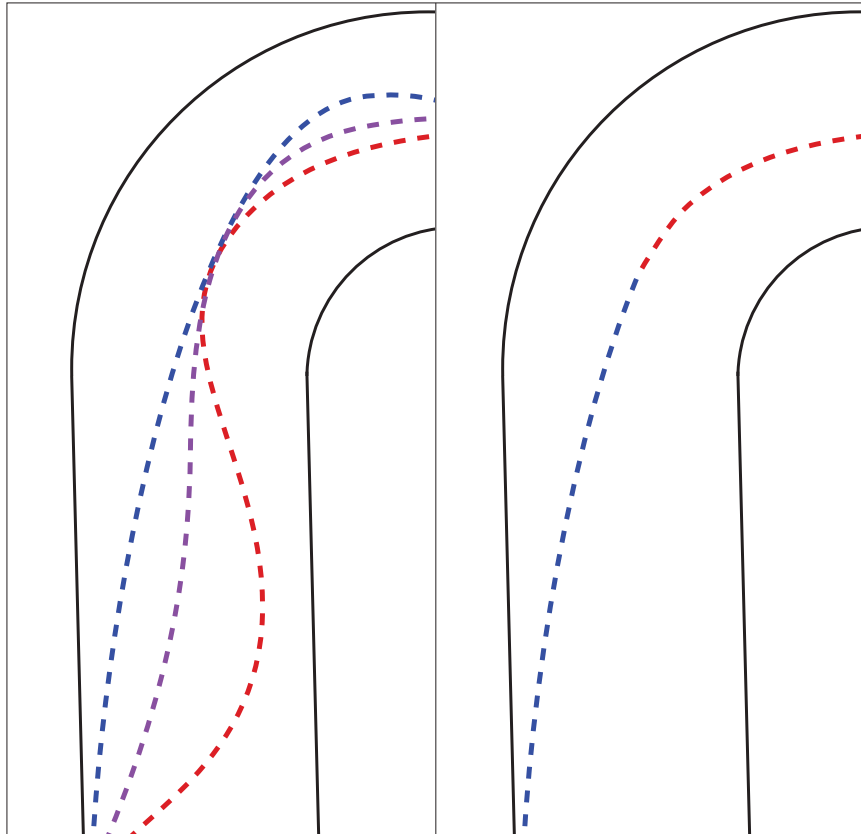


Abbildung 4.1: Mitteln und Kombinieren zweier Rennlinien.

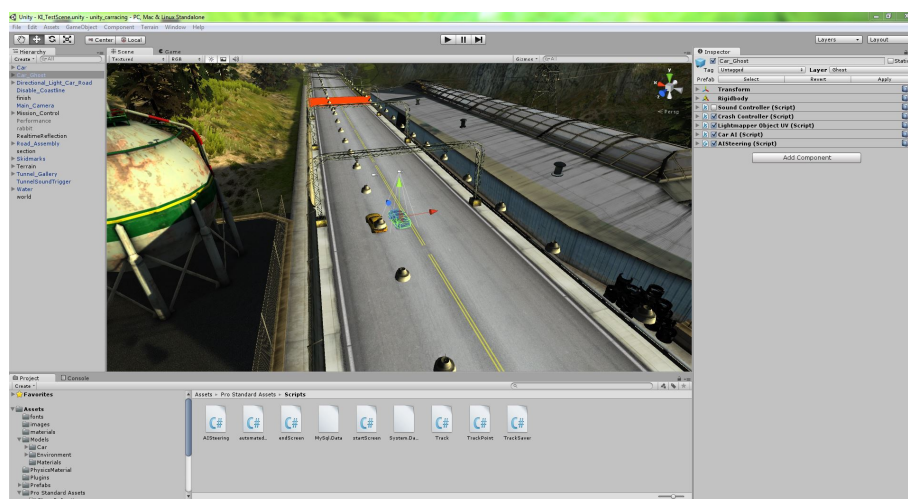


Abbildung 4.2: GameEngine Unity3D.

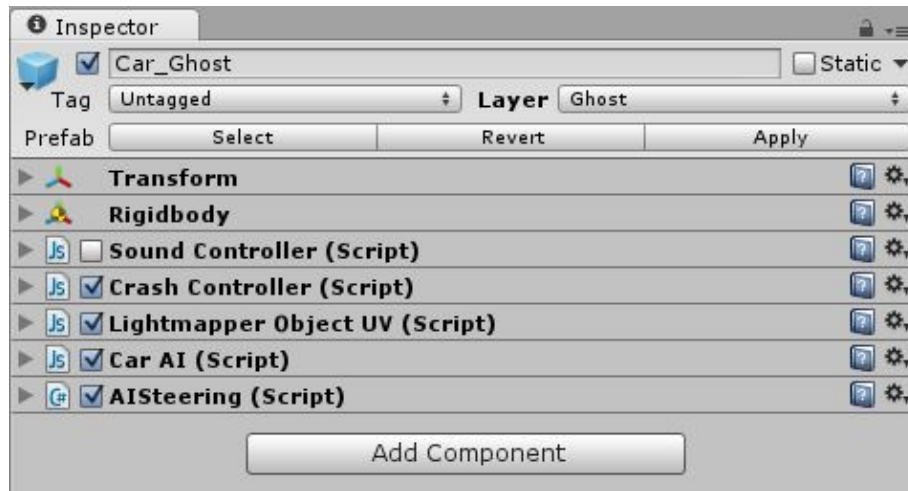


Abbildung 4.3: Unity Inspector – Komponenten eines Objektes.

den *Asset Store*² von Unity zurückgegriffen. Als Ausgangspunkt für die Umsetzung des eigenen Ansatzes und dem Einbau einer KI wurde Car Tutorial gewählt. Dieses Tutorial, herausgegeben von Unity Technologies, ist frei zum Download verfügbar und stellt nach vollständiger Durchführung des Tutorials ein fertiges Rennspiel ohne computergesteuerte Gegner zur Verfügung³. Das durch das Tutorial erstellte Spiel, besteht nun aus einer Vielzahl an Objekten und Scripte, deren Erläuterung aber nicht Teil dieser Arbeit sind. Eines dieser Scripte ist jedoch relevant für diese Arbeit – das Car.js Script. Dieses Script ist eine am Fahrzeug des Spielers hinzugefügte Komponente und hat folgende Aufgaben:

- Einstellung des Verhalten des Fahrzeuges und
- Entgegennahme und Weiterverarbeitung des Keyboard Inputs.

Das Verhalten des Fahrzeuges kann dabei durch Veränderung verschiedener Parameter beeinflusst werden. Abbildung 4.4 zeigt diese im Unity Inspector.

Durch die Veränderung dieser Werte ist es möglich ein simulationsähnliches Spiel oder ein arcadenähnliches Spiel zu schaffen. Folgende Eigenschaften nehmen in diesem Script dabei am meisten Einfluss:

- Radaufhängung (suspension range, damper, spring front, spring rear),
- Höchstgeschwindigkeit (topspeed),
- Luftwiderstand (drag multiplier),
- Schwerpunkt (center of mass) und

²<http://unity3d.com/asset-store>

³<https://www.assetstore.unity3d.com/#!/content/10>

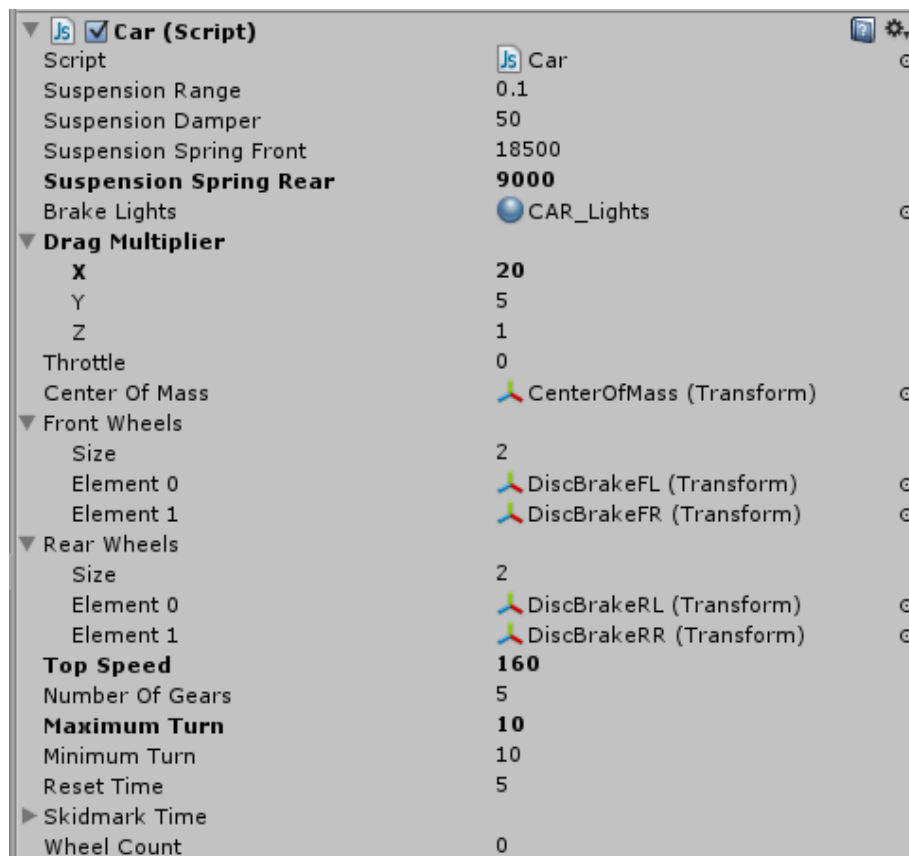


Abbildung 4.4: Unity Inspector – Car.js Komponente des spielergesteuerten Fahrzeuges.

- Maximale Einschlag, minimale Einschlag (maximum turn, minimum turn).

4.4 Aufbau und Umsetzung

Dieses Kapitel behandelt den Kern meiner Arbeit. Es wird dabei genauer auf die Umsetzung der KI und die Erstellung der dafür notwendigen Rennlinien eingegangen. Des Weiteren werden dann Möglichkeiten zur Erstellung von neuen Rennlinien aus gegebenen aufgezeigt und beschrieben. Diese werden durch einen automatisierten Test erstellt und auf ihre mögliche Verwendung zur Verbesserung einer KI getestet. Anschließend werden diese Ergebnisse im Kapitel 5 ausgewertet und analysiert.

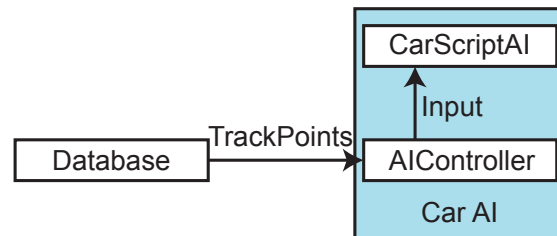


Abbildung 4.5: Steuerung der KI durch die AI Controller Komponente.

4.4.1 Steuerung der KI

In diesem Unterkapitel geht es um den Austausch der Steuerung des Spielers durch eine computergesteuerte Steuerung. Hierzu muss der Input über das Keyboard mit einem vom Computer gegebenen Input ersetzt werden. Zu beachten ist hierbei, dass der Input durch verschiedenste Einstellungen konfigurierbar ist und diese bei der KI nachempfunden werden müssen. Der Input wird für den Spieler über die Car.js Komponente entgegengenommen und weiterverarbeitet. In diesem Script gibt es dafür eine eigene Funktion, die sich nur um den vom Spieler kommenden Input kümmert. In dieser Funktion werden folgende Methoden der Input Klasse verwendet:

- `Input.GetAxis("Vertical")` und
- `Input.GetAxis("Horizontal")`.

```

1 function GetInput()
2 {
3   throttle = Input.GetAxis("Vertical");
4   steer = Input.GetAxis("Horizontal");
5
6   if(throttle < 0.0)
7     brakeLights.SetFloat("_Intensity", Mathf.Abs(throttle));
8   else
9     brakeLights.SetFloat("_Intensity", 0.0);
10
11   CheckHandbrake();
12 }

```

Wie Abbildung 4.5 zeigt, musste nun für den computergesteuerten Gegner dieser Input mit einem errechneten Input ausgetauscht werden. Hierfür besitzt die KI ein eigenes Script, welches dem Fahrzeug als weitere Komponente hinzugefügt wurde. Diese Komponente (*AI Controller*) simuliert nun den Keyboard Input des Spielers.

Damit der AI Controller nun einen Input errechnen beziehungsweise ausgeben kann muss es noch ein Ziel geben, das es anzusteuern gilt. Dafür gibt es für die KI Wegpunkte auf der Strecke, welche sie nacheinander ansteuert,

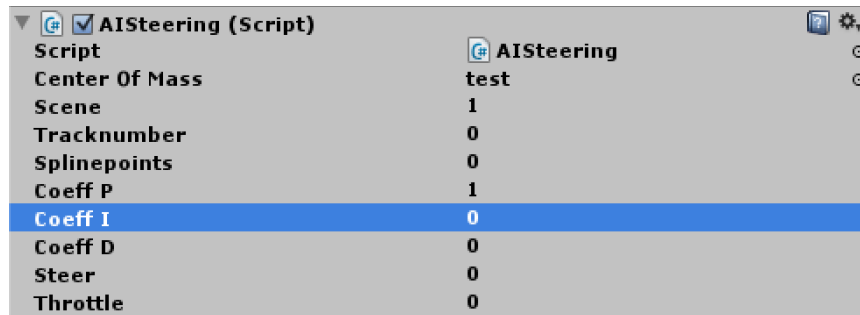


Abbildung 4.6: AISteering Komponente mit den PID Koeffizienten.

um letztlich das Ziel zu erreichen. Diese Wegpunkte werden im nächsten Unterkapitel genauer behandelt. Um diese Punkte ansteuern zu können, wird immer ein bestimmter Wegpunkt vor der KI hergenommen. Mithilfe einer Methode werden die Koordinaten des Wegpunktes relativ zu den Koordinaten des Fahrzeuges gesetzt. Somit weiß die KI nun, ob sich der Punkt beispielsweise rechts oder links zur Fahrtrichtung befindet und in welche Richtung sie einschlagen muss. Die Methode um die Position eines Wegpunktes relativ zur Position des Autos zu setzen – also um einen Punkt vom *worldspace* in den *localspace* zu transformieren sieht wie folgt aus:

```
function InverseTransformPoint(position:Vector3):Vector3
```

Somit weiß man in welchem Winkel sich der Wegpunkt zum Fahrzeug befindet. Da der Winkel zum nächsten Wegpunkt gegeben ist, kann der im Kapitel 3 beschriebene PID-Regler zum Einsatz kommen. Dieser bietet sich für die Steuerung des Fahrzeuges an, da es immer den Winkel zwischen Wegpunkt und Auto zu regeln gilt. Wie zuvor beschrieben benötigt der Regler einen Ist- Wert und einen Soll- Wert. Ziel des Regler ist es hierbei den Winkel zwischen Auto und Wegpunkt auf 0 zu bringen, das heißt, dass das Fahrzeug auf den Wegpunkt direkt zusteuern soll. Abbildung 4.6 zeigt die AISteering Komponente mit den 3 Koeffizienten.

In dieser Abbildung sind die Koeffizienten für den I- Anteil und den D- Anteil auf 0 gestellt, was bedeutet, dass es sich momentan um einen reinen P-Regler handelt. Um die Regelgröße für den PID-Regler zu errechnen, benötigt man zuerst den Regelfehler. Dieser entspricht dem Winkel des angesteuerten Wegpunktes zum Fahrzeug, da man einen Winkel von 0 Grad erreichen möchte. Somit hat man bereits den Proportionalanteil für den PID-Regler berechnet. Der Koeffizient c_P führte dabei bei einem Wert von 0,95 zur besten Ansteuerung der Wegpunkte. Dies Wert wurde durch ein schrittweises Erhöhen und Testen der KI gewählt. Der Integralteil ergibt sich aus der Summe aller Regelfehler multipliziert mit den dazugehörigen

Zeitschritten zwischen den Fehlern. Folgender Code zeigt die Berechnung des I Anteils:

```
1 float error = angle - desiredAngle;
2 float errorDeltaTime = error * Time.deltaTime;
3 float sumErrorDeltaTime += errorDeltaTime;
4 float pid_i = sumErrorDeltaTime * coeffI;
```

Der Wert des Koeffizienten c_I fiel dabei auf 0,85. Abschließend muss noch der Differentialanteil errechnet werden. Dieser betrachtet die Änderung des Regelfehlers von einem Frame auf den Anderen und versucht somit den Fehler beim Entstehen entgegen zu wirken. Diesen Wert erhält man durch die Differenz des momentanen Winkels zum Wegpunkt und dem Winkel im Frame davor und dividiert diesen Wert mit der dazwischenliegenden Zeit. Dies sieht wie folgt aus:

```
1 float deltaError = error - lastError;
2 float pid_d = deltaError/Time.deltaTime * coeffD;
```

Da kein Wert des c_D Koeffizienten zu einer merkbaren Verbesserung führte, wurde dieser Wert auf 0 gestellt. Somit handelt es sich beim Regler zur Steuerung der KI um einen PI-Regler, welcher im Abschnitt 3.2.2 erläutert wurde. Da es sich bei der Regelgröße um einen Winkel handelt, muss dieser noch im Anschluss in einen Einschlag des Fahrzeuges zwischen -1 und 1 gebracht werden. Der maximale Einschlag des Autos ist dabei in der Car Script Komponente des Fahrzeuges definiert. Beträgt dieser beispielsweise 10, so würde der Einschlag bei einem Winkel von 10 Grad zum nächsten Wegpunkt den Wert 1 ergeben. Für Wegpunkte die sich links des Fahrzeuges befinden, nimmt der Einschlag einfach einen negativen Wert an.

Mit dem Einschlag war nun auch noch die gewünschte Geschwindigkeit notwendig mit der das computergesteuerte Auto fahren sollte. Diese wird im jeweiligen angesteuerten Wegpunkt mitgespeichert. Durch ein einfaches Vergleichen der momentanen Geschwindigkeit mit der gewünschten Geschwindigkeit des nächsten Wegpunktes weiß die KI, ob sie Gas geben oder Bremsen soll. Hier kam kein PID-Regler zum Einsatz, da es beim Gas geben und bremsen nur diese zwei Zustände gibt.

4.4.2 Datenbank und Wegpunkte

Wie am Anfang dieses Kapitels 4 beschrieben, soll sich durch eine Kombination von Rennlinien eine neue und möglicherweise auch bessere Rennlinie ergeben, um den Spieler auch beim Erfahrungsanstieg weiterhin eine Herausforderung darstellen zu können. Um solche Rennlinien kombinieren zu können, müssen bereits welche bestehen. Dieser Abschnitt behandelt nun die Erstellung und Speicherung von Rennlinien durch Spieler, welche dann als Grundlage zur Erstellung neuer Rennlinien dienen. Abbildung 4.7 zeigt eine visuelle Darstellung einer möglichen Rennlinie beziehungsweise der Wegpunkte in Form von roten Kugeln.



Abbildung 4.7: Wegpunkte einer Rennlinie.

Um eine KI mithilfe solcher Wegpunkte durch eine Strecke fahren zu lassen, müssen diese folgende Informationen beinhalten:

ID: Die ID ist eine fortlaufende Nummer und unterstützt die KI bei der Auswahl des Wegpunktes, den es zu befahren gilt.

Position: Die Position dient zur Steuerung der KI. Mit Hilfe der Koordinaten wird die Relative Position des Wegpunktes zum Auto und folglich der gewünschte Einschlag errechnet.

Geschwindigkeit: Durch die gespeicherte Geschwindigkeit des Wegpunktes weiß die KI, ob sie ihre Geschwindigkeit erhöhen oder senken muss, um die Strecke problemlos durchfahren zu können.

Dies würde prinzipiell reichen um eine KI durch eine Strecke zu bringen. Folgende Informationen werden jedoch zusätzlich gespeichert:

Scene: Diese Information dient zur Speicherung der Rennstrecke, da es in den meisten Computerspielen mehr als eine Rennstrecke gibt.

Track: Da es für eine Rennstrecke auch mehrere Rennlinien gibt muss dies ebenfalls gespeichert werden.

Um solche Wegpunkte erstellen und speichern zu können wurden zwei

Scripte erzeugt, welche dem Fahrzeug des Spielers als Komponente hinzugefügt wurden.

TrackPoint.cs: Diese Klasse dient zur Speicherung der Wegpunkte und beinhaltet die oben genannten Informationen als Variablen.

TrackSaver.cs: Dieses Script ist für die Herstellung einer Datenbankverbindung und für Speicherung der Wegpunkte zuständig.

Wie oben genannt dient das TrackPoint.cs rein zur Erstellung von Wegpunkten. Diese Datei ist einfach gehalten und besteht folglich aus 7 Variablen und 2 Konstruktoren.

```
1 public class TrackPoint {  
2  
3     static int counter = 1;  
4     public int pointId;  
5     public Vector3 position;  
6     public double speed;  
7     public double time;  
8     public int sectionNumber;  
9     public int trackNumber;  
10  
11     public TrackPoint(Vector3 p, double s, double t, int sectionNumber)  
12         {...}  
13     public TrackPoint(Vector3 p, double s, double t, int sectionNumber,  
14         int trackNumber){...}  
15 }
```

Nennenswert hierbei ist noch, dass die time Variable die Zeit der Erstellung des Wegpunktes ab den Start eines Levels speichert. Dazu gibt es eine von Unity bereitgestellte Variable die wie folgt lautet:

```
float time = Time.timeSinceLevelLoad;
```

Um später aus Wegpunkten bestehende Rennlinien kombinieren zu können, wird eine Rennlinie in Streckenabschnitte unterteilt. Um welchen Streckenabschnitt einer Rennlinie es sich handelt, wird in der sectionNumber Variable gespeichert. Sollte eine KI einfach eine von einem Spieler kommende Strecken nach fahren, so könnte man auf diese Variable verzichten.

Da nun mit Hilfe der Klasse TrackPoint.cs Wegpunkte erstellt werden können, müssen diese noch in einer Rennlinie zusammengefasst und in eine Datenbank gespeichert werden. Dazu besitzt das Fahrzeug des Spielers die TrackSaver.cs Komponente. Dieses Script wird von der KI nicht benötigt, da eine KI nur Wegpunkte abfahren und nicht Wegpunkte erstellen und speichern soll. Abbildung 4.8 zeigt diese Komponente mit ihren Einstellmöglichkeiten im Unity Inspector.

Die *Set Point Interval* Einstellung hängt, wenn man das Script dann genauer betrachtet mit der *Time as Interval* Einstellung zusammen. Dabei

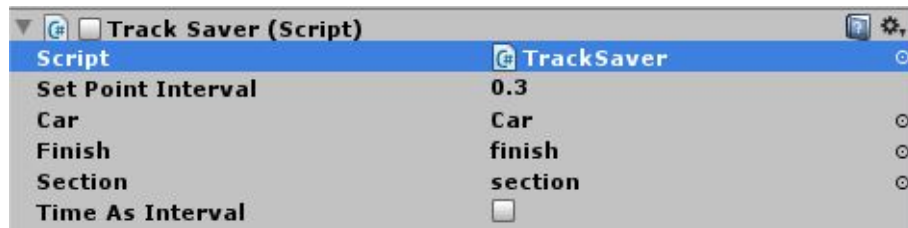


Abbildung 4.8: TrackSaver Komponente im Unity Inspector.

gibt die *Time as Interval Checkbox* an ob das Setzen der Wegpunkte in bestimmten Zeit oder Streckenabschnitten erfolgen soll. Die *Set Point Interval* Einstellung konfiguriert die Zeit oder die Strecke die zwischen zwei Wegpunkten sein soll. In diesen Abständen wird dann in der Update Methode ein Wegpunkt erzeugt und gesetzt:

```

1  if(timeAsInterval){
2      var timeNow = Time.realtimeSinceStartup;
3
4      if( timeNow > lastInterval + setPointInterval ){
5          lastInterval = timeNow;
6          TrackPoint tmpTrackPoint = new TrackPoint(car.transform.
              position, car.rigidbody.velocity.magnitude, Time.timeSinceLevelLoad,
              sectionNumber);
7          track.Add(tmpTrackPoint);
8      }
9  } else {
10     if (transform.InverseTransformPoint(lastPosition).magnitude >= 5){
11         TrackPoint tmpTrackPoint = new TrackPoint(car.transform.position
              , car.rigidbody.velocity.magnitude, Time.timeSinceLevelLoad,
              sectionNumber);
12         track.Add(tmpTrackPoint);
13         lastPosition = transform.position;
14     }
15 }

```

Des Weiteren gibt es noch den *finish* und den *section* Parameter. Diese Parameter benötigen jeweils ein *GameObject* aus der Szene. Abbildung 4.9 zeigt diese zwei Objekte in der Szene. Beim Befahren der Strecke sind diese nicht zu sehen, werden aber trotzdem benötigt.

Diese Objekte rufen, sobald ein Objekt mit der *TrackSaver* Komponente (Fahrzeug des Spielers) sie durchfährt, folgenden Codeabschnitt auf:

```

1 void OnTriggerEnter(Collider other) {
2     if (other.name == finish.name){
3         Debug.Log("Saving track in database!");
4         checkAndSaveTrack();
5     }
6 }
7
8 }

```

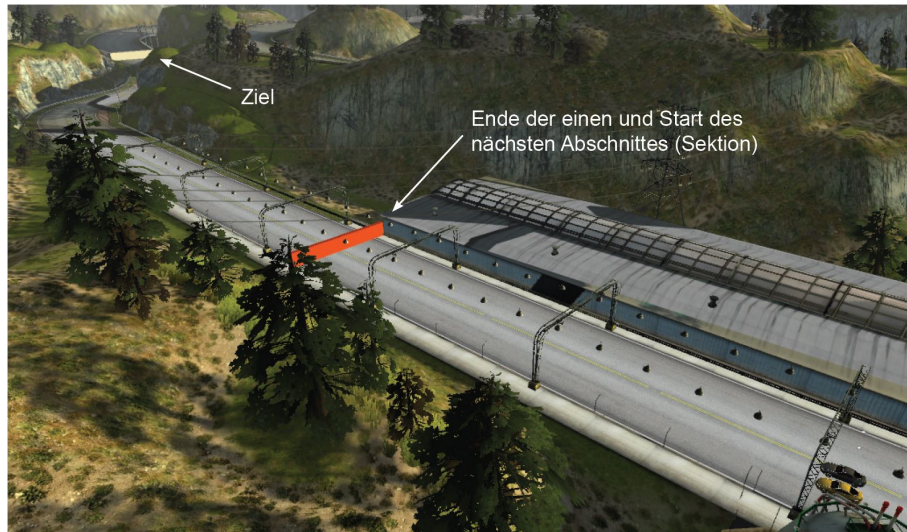


Abbildung 4.9: GameObjects - finish und section.

```

9
10 void OnTriggerExit(Collider other) {
11     if (other.name == section.name){
12         Debug.Log("New section reached!");
13         sectionNumber++;
14     }
15 }

```

Je nachdem welches Objekt der Spieler durchfährt wird einer dieser beiden Methoden aufgerufen. Die Überprüfung um welchen Objekt es sich handelt ist dabei folgende:

```

1 if (other.name == section.name) ...
2 if (other.name == finish.name) ...

```

Bei den beiden Variablen *section* und *finish* beinhalten die zwei Objekte die im Bild 4.8 angegeben wurden.

Die Wegpunkte werden nun in gewissen Zeit- oder Wegabständen mit den nötigen Informationen in einer Liste gespeichert. Erreicht das Auto nun das Ziel, wird wie im Code oben gezeigt die *checkAndSaveTrack* Methode aufgerufen. Diese Methode dient zur Herstellung einer Datenbankverbindung und der Speicherung der Wegpunkte in dieser. Um in Unity eine Datenbankverbindung herstellen zu können müssen noch zwei DLLs in das jeweilige Script eingebunden werden:

- MySql.Data und
- System.Data.

		id	position_x	position_y	position_z	speed	time	scene	track	section
☐	Edit Copy Delete	396	1096.71	103.083	648.039	53.2530975341797	9.19225597381592	1	4	1
☐	Edit Copy Delete	383	1045.47	101.83	700.823	46.3003883361816	7.69712352752686	1	4	1
☐	Edit Copy Delete	382	1041.93	101.758	705.083	45.7757720947266	7.57012557983398	1	4	1
☐	Edit Copy Delete	381	1038.36	101.69	709.216	45.1996803283691	7.45522212982178	1	4	1

Abbildung 4.10: Auszug aus der Datenbank.

Mit Hilfe dieser DLLs kann man nun folgendermaßen eine Datenbankverbindung herstellen.

```

1 string connectionString = "Server=localhost;" + "Database=DBname;" + "
  User ID=userID;" + "Password=pw;" + "Pooling=false";
2 dbConnection = new MySqlConnection(connectionString);
3 dbConnection.Open();

```

Hat man nun eine Verbindung mit einer Datenbank hergestellt, so kann man die Liste mit den gespeicherten Wegpunkten durchgehen und sie in einer Datenbanktabelle speichern. Der Code zur Speicherung der Wegpunkte in eine Datenbank sieht wie folgt aus:

```

1 void checkAndSaveTrack(){
2     foreach (TrackPoint p in track){
3         string sqlQuery = "INSERT into trackpoints (scene, position_x,
4           position_y, position_z, speed, time, track, section) VALUES (" +
5           Application.loadedLevel + "," + p.position.x + "," + p.position.y +
6           "," + p.position.z + "," + p.speed + "," + p.time + "," +
7           trackcounter + "," + p.sectionNumber + ")";
8
9         IDbCommand dbCommand = dbConnection.CreateCommand();
10        dbCommand.CommandText = sqlQuery;
11        IDataReader reader = dbCommand.ExecuteReader();
12
13        reader.Close();
14        reader = null;
15        dbCommand.Dispose();
16        dbCommand = null;
17    }
18    trackSaved = true;
19 }

```

Hat man die Wegpunkte in eine Datenbank gespeichert so sieht die Liste beispielsweise wie in Abbildung 4.10 aus.

Da man nun Rennlinien in der Datenbank gespeichert hat, muss nur noch eine Rennlinie für die KI ausgewählt werden, welche sie dann abfahren soll. Dies geschieht, wie die Steuerung, ebenfalls in der *AISteering* Komponente der KI. Dazu ist es möglich im Unity Inspector eine Strecke und eine Rennlinie auszuwählen, die vor dem Start für die KI geladen werden soll (siehe Abbildung 4.11).

Um Daten aus einer Datenbank zu erhalten, muss wie bereits erklärt eine Verbindung hergestellt und ein SQL Statement ausgeführt werden. Im SQL



Abbildung 4.11: AISteering Komponente der KI.

Statement werden dann die im Inspector angegebenen Parameter eingefügt um die richtige Rennlinie zu erhalten. Zusätzlich wird das Ergebnis nach der ID sortiert, um gleich die richtige Reihenfolge der Wegpunkte zu erhalten.

```
string getTrack = "SELECT * FROM trackpoints WHERE scene = " + scene +  
" AND track = " + tracknumber + " ORDER BY id ASC";
```

Anschließend erhält man ein `IDataReader` Objekt zurück welches auf folgender Art ausgelesen werden kann:

```
1 while (reader.Read()) {  
2     float x = (float) reader["position_x"];  
3     float y = (float) reader["position_y"];  
4     float z = (float) reader["position_z"];  
5  
6     double speed = (double) reader["speed"];  
7     double time = (double) reader["time"];  
8  
9     TrackPoint t = new TrackPoint(new Vector3(x,y,z), speed, time,  
    sectionNumber);  
10    this.track.Add(t);  
11 }
```

Hier muss nur noch beachtet werden, dass der Index mit dem Namen der Spalte in der Datenbank übereinstimmen muss. Nun hat die KI eine Liste namens `track` die alle für sie notwendigen Wegpunkte beinhaltet. Beim Start nimmt die KI nun den ersten Wegpunkt her und steuert diesen wie im Abschnitt 4.4.1 beschrieben an. Dieser wird so lange angesteuert, bis die KI einen gewissen Abstand zu diesen erreicht hat. Dieser Abstand hängt von der Geschwindigkeit des Fahrzeuges ab, mit dem es sich gerade fortbewegt.

```
if(transform.InverseTransformPoint(targetTrackPoint.position).magnitude  
    <= (10 + rigidbody.velocity.magnitude/3) || relativeWaypointPosition  
    .z <= 3)
```

Abbildung 4.12 zeigt anhand einer roten Linie welcher Punkt gerade von der KI angesteuert wird.

4.4.3 Automatisierter Test

Da die KI Rennlinien aus einer Datenbank holen und abfahren kann, geht es in diesem Abschnitt um die eigentliche Fragestellung. Hier wird beschrieben ob und wie Rennlinien kombiniert und getestet werden können. Letztendlich

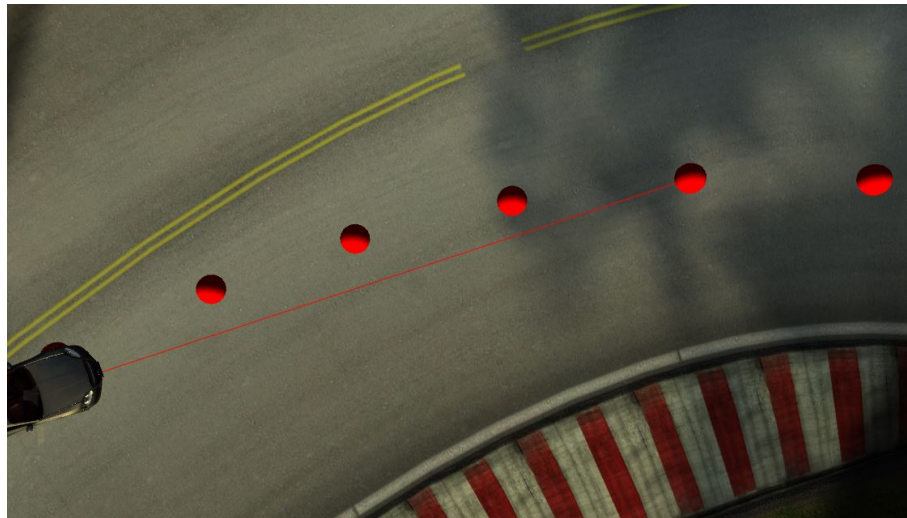


Abbildung 4.12: Angesteuerter Wegpunkt der KI.

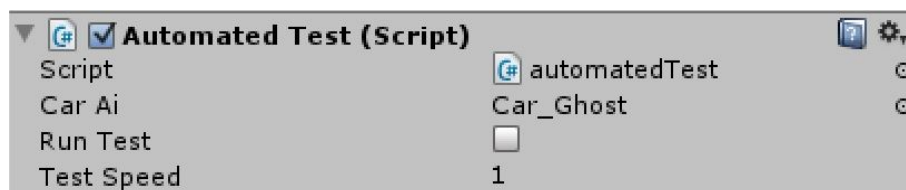


Abbildung 4.13: Inspector der automatedTest Komponente.

soll die Frage geklärt werden ob es möglich ist aus gegebenen Rennlinien eine bessere zu erzeugen, damit die KI mit dem Anstieg der Spielerfahrung des Spielers auch weiterhin eine Herausforderung darstellen kann. Abbildung 4.1 aus dem Abschnitt 4.1 zeigt dies nochmals. Der linke Abschnitt stellt dabei das Mitteln einer roten und blauen Rennlinie anhand einer violetten Rennlinie dar und der rechte Abschnitt zeigt eine Kombination der Streckenabschnitte dieser zwei Rennlinien. Letztendlich soll das Kombinieren und Mitteln der Rennlinien automatisch ablaufen und die Ergebnisse ausgegeben werden. Dazu kommt eine weitere Komponente hinzu – das `automatedTest.cs` Script. Dieses ist für die Ausführung des Testes zuständig. Dieses Script muss dazu als Komponente an ein *GameObject* angefügt werden. Dies kann beispielsweise ein sogenanntes *Empty Object* sein. Diese Komponente hat, wie Abbildung 4.13 zeigt, folgende Einstellungsmöglichkeiten:

Nun ist es möglich durch das Auswählen der *Checkbox Run Test* entweder den Test auszuführen oder wie im vorigen Kapitel beschrieben die KI eine bestimmte Strecke fahren zu lassen. Des Weiteren kann die Geschwindigkeit des Tests ausgewählt werden. Der Wert 1 bedeutet hierbei die normale Ge-

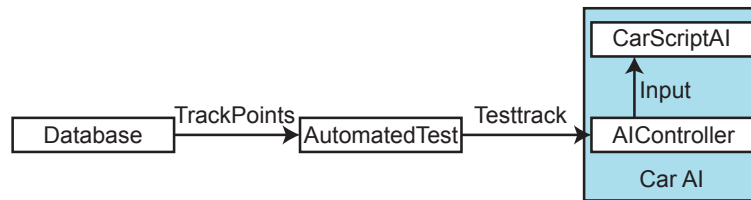


Abbildung 4.14: Aufbau des automatischen Tests.

schwindigkeit. Möchte man beispielsweise das die KI die Strecke doppelt so schnell durchfährt als gewöhnlich müsste man den Wert 2 einstellen. Bei dieser Erhöhung der Geschwindigkeit handelt es sich um die Abspielgeschwindigkeit des Tests und nicht um die Maximalgeschwindigkeit des Fahrzeugs. Diese Einstellung kann auch während des Testes durch Drücken der Tasten + und – beeinflusst werden. Dies erfolgt durch eine Input Überprüfung in der Update Methode des Scripts.

```

1 if(Input.GetKeyUp(KeyCode.KeypadPlus)){
2     testSpeed++;
3     Time.timeScale = testSpeed;
4 }

```

Ist die *Run Test Checkbox* aktiv, so wird der Test ausgeführt. Dazu wird in der *AISteering* Komponente der KI nun die *StartForTest* Methode ausgeführt. Diese wird durch folgende Methode des *AutomatedTest* Scripts aufgerufen:

```
carAi.BroadcastMessage("StartForTest");
```

Durch diese *BroadcastMessage* Methode werden alle Methoden die das Objekt *carAI* mit diesem Namen besitzt aufgerufen. Bevor die *StartforTest* Methode des *AISteering* Scripts aufgerufen wird, lädt das *AutomatedTest* Script alle notwendigen Rennlinien der zu testenden Rennstrecke aus der Datenbank. In der *StartForTest* Methode der *AISteering* Komponente wird nun nicht mehr direkt auf die Datenbank zugegriffen, sondern eine Rennlinie vom *AutomatedTest* Script angefordert. Dabei wird entweder eine Rennlinie durch ein Mitteln oder durch eine Kombination aus Streckenabschnitten angefordert. Dies sieht wie folgt aus:

```

1 track = automatedScript.createTrackToTest(currentSectionOne,
    currentSectionTwo);
2 track = automatedScript.createAverageTrackToTest(currentTrackOne,
    currentTrackTwo);

```

Abbildung 4.14 zeigt den grundlegenden Aufbau des Testes. Diese veranschaulicht, dass die KI die Rennlinie bzw. die Wegpunkte nicht mehr direkt aus der Datenbank, sondern vom *AutomatedTest* Script bekommt.

Bei diesem Test werden zuerst Rennlinien durch eine Kombination der Sektionen zweier Rennlinien getestet. Das *AutomatedTest* Script hat hierzu bereits die Rennlinien beim Laden aus der Datenbank in ihre Sektionen unterteilt und in eigene Listen gespeichert.

```

1 TrackPoint t = new TrackPoint(new Vector3(x,y,z),speed, time, section, i
    );
2
3     switch(section){
4     case 0:
5         tmpTrackPointsSectionOne.Add(t);
6         break;
7
8     case 1:
9         tmpTrackPointsSectionTwo.Add(t);
10        break;
11    }

```

Mit Hilfe dieser zwei Listen kann das Test Script in der *createTrackToTest* Methode die angeforderte Rennlinie erzeugen und zurückgeben:

```

1  public List<TrackPoint> createTrackToTest(int sectionOne, int
    sectionTwo){
2  ...
3
4  foreach( TrackPoint t in trackListSectionOne[sectionOne].trackpoints
    ){
5      trackToTest.Add(t);
6  }
7
8  foreach( TrackPoint t in trackListSectionTwo[sectionTwo].trackpoints
    ){
9      trackToTest.Add(t);
10 }
11
12 return trackToTest;
13
14 }

```

Hat der Test die Kombinationen verschiedenster Sektionen der Rennlinien getestet, so testet dieser anschließend ein Mitteln zweier Rennlinien. Dazu wird in der *createAverageTrackToTest* Methode eine neue Rennlinie aus zwei bestehenden Rennlinien durch ein Mitteln generiert.

```

1  public List<TrackPoint> createAverageTrackToTest(int trackOne, int
    trackTwo){
2      trackToTest.Clear();
3
4      for(int i = 0; i< Mathf.Min (trackListSectionOne[trackOne].
        trackpoints.Count, trackListSectionOne[trackTwo].trackpoints.Count);
        i++){
5          Vector3 position = (trackListSectionOne[trackOne].trackpoints[i].
            position + trackListSectionOne[trackTwo].trackpoints[i].position)/2;
6          double speed = Mathf.Max((float)trackListSectionOne[trackOne].

```

```

        trackpoints[i].speed, (float)trackListSectionOne[trackTwo].
        trackpoints[i].speed);
7     TrackPoint t = new TrackPoint(position, speed,0,0);
8     trackToTest.Add(t);
9 }
10
11 ...
12
13 }
```

Dieser Code zeigt das Mitteln zweier Rennlinien nur für eine Sektion, da dies für die weitere Sektion gleich ist. Beim genaueren Betrachten des Code-Stückes ist auch ersichtlich, dass Wegpunkte, die eine Sektion im Vergleich zur anderen zu viel hat, wegfallen. Des Weiteren ist im Code ersichtlich, dass nur der Verlauf der Rennlinie gemittelt wird, nicht aber die Geschwindigkeit. Die Wahl der Geschwindigkeit fällt dabei auf den Wegpunkt mit der höheren Geschwindigkeit. Wenn die KI es schafft die Rennlinie mit der höheren Geschwindigkeit entlang zu fahren, würde sie dies zu einer besseren Zeit verhelfen.

All diese erzeugten Rennlinien werden nun von der KI mit schnellerer Spielgeschwindigkeit abgefahren und auf ihre Funktionalität getestet. Da durch die Kombination zweier Rennlinien auch Rennlinien entstehen können, welche zu einem stecken bleiben der KI führen, muss dies beim Ausführen des Testes auch erkannt und behoben werden. Dieser Mechanismus prüft dabei die Strecke, um die sich das Fahrzeug in einem gewissen Zeitabstand fortbewegt hat. Unterschreitet dies einen bestimmten Schwellenwert, der gegen Null geht, so kann der Test annehmen, dass das Fahrzeug steckt. Folglich protokolliert der Test mit, um welche Rennlinie es sich handelt und bricht diesen ab und startet den Test mit der nächsten vorgesehenen Rennlinie. Der Code für diese Überprüfung sieht dabei wie folgt aus:

```

1  if(Time.timeSinceLevelLoad > 3 && this.rigidbody.velocity.magnitude <=
    0.3f){
2      Debug.Log ("car stuck" );
3
4      ...
5
6  Application.LoadLevel(Application.loadedLevel);
7  }
```

Während der Durchführung des Tests ist es wie bereits erwähnt möglich die Geschwindigkeit des Tests zu erhöhen oder verlangsamen. Zu beachten ist hierbei, dass man die Erhöhung der Geschwindigkeit des Testes in Abhängigkeit zur Rechenleistung des Computers steht. Zusätzlich besteht die Möglichkeit die Fahrt der KI aus einem anderen Blickwinkel zu betrachten um somit leichter erkennen zu können, wie genau sie sich dabei an die



Abbildung 4.15: Perspektiven während des Tests.



Abbildung 4.16: GUI während des Testes.

Rennlinien hält. Abbildung 4.15 zeigt dabei die möglichen Perspektiven, die während des Testes zur Auswahl stehen.

Um während des Tests wissen zu können, was gerade getestet wird, wird zusätzlich am oberen Bildschirmrand ein *GUI* (*graphical user interface*) angezeigt. Dies gibt an ob es sich um eine gemittelte oder aus Sektionen kombinierte Rennlinie handelt und aus welchen ursprünglichen Rennlinien sie besteht. Abbildung 4.16 zeigt dieses *GUI* Element während der Durchführung des Tests.

Ist der Test alle möglichen Kombinationen von Rennlinien durchgegangen und hat sie auf ihre Funktionalität überprüft, so gibt er im folgenden

Schritt die Ergebnisse für diese Rennlinien aus.

Kapitel 5

Auswertung und Ergebnis des eigenen Ansatzes

In diesem Kapitel wird das Ergebnis eines vorhin beschriebenen Tests genauer betrachtet und analysiert. Abbildung 5.1 zeigt hierzu ein Ergebnis eines Tests.

Aus diesem Ergebnis ist zu schließen, dass insgesamt vier Rennlinien auf ihre Kompatibilität in Bezug auf ein Mitteln und ein Kombinieren der Sektionen getestet worden sind. Dabei handelt es sich um die Rennlinien (tracks) 0 bis 3. Auffallend ist ebenfalls, dass das computergesteuerte Fahrzeug immer wieder stecken geblieben ist. Analysiert man nun in welchen Situationen dies geschah, so erkennt man, dass track 3 jedes mal eine der beiden Rennlinien ist. Folglich muss es auf der Strecke von Rennlinie 3 eine Passage geben, die dafür verantwortlich ist. Abbildung 5.2 zeigt hierzu die Zeit jeder Rennlinie, die benötigt wurde um die Strecke zu durchfahren.

Wenn man sich den ersten Teil des Ergebnisses ansieht, kann man den Schluss fassen, dass sich das Problem in der zweiten Sektion der Rennlinie 3 befindet. Der Ausschnitt 5.3 aus dem Ergebnis veranschaulicht dies. Alle Kombinationen der Sektion 1 von Rennlinie 3 fielen positiv aus, abgesehen von der Kombination der Sektion 1 mit Sektion 2 der Rennlinie 3 – also der ursprünglichen Rennlinie.

Abbildung 5.4 zeigt diese Stelle, bei der die KI stecken bleibt. Betrachtet man nun wieder das Ergebnis, so zeigt dieses, dass durch einen Austausch dieser Sektion mit einer Sektion einer anderen Linie wieder zu einem positiven Ergebnis für die KI führt. Dies muss nicht bedeuten, dass es auch zu einem besseren Ergebnis als eine der ursprünglichen Rennlinien führt. So führt beispielsweise eine Kombination aus track 3 und track 0 zu einem ähnlichen Ergebnis wie die originale Rennlinie track 0.

track 3: Das Fahrzeug bleibt stecken.

track 0: Das Fahrzeug durchfährt die Strecke in 14,5 Sekunden.

Sektion 1: track 3, Sektion 2: track0: Das Fahrzeug durchfährt die Stre-

Results of combining sections:

Time to Finish: 14.5 | Section 1 of Track 0 and Section 2 of Track 0
 Time to Finish: 15.47 | Section 1 of Track 0 and Section 2 of Track 1
 Time to Finish: 16.23 | Section 1 of Track 0 and Section 2 of Track 2
 Car stuck!!!! - Section 1 of Track 0 and Section 2 of Track 3
 Time to Finish: 14.57 | Section 1 of Track 1 and Section 2 of Track 0
 Time to Finish: 15.43 | Section 1 of Track 1 and Section 2 of Track 1
 Time to Finish: 16.26 | Section 1 of Track 1 and Section 2 of Track 2
 Car stuck!!!! - Section 1 of Track 1 and Section 2 of Track 3
 Time to Finish: 14.53 | Section 1 of Track 2 and Section 2 of Track 0
 Time to Finish: 15.42 | Section 1 of Track 2 and Section 2 of Track 1
 Time to Finish: 16.17 | Section 1 of Track 2 and Section 2 of Track 2
 Car stuck!!!! - Section 1 of Track 2 and Section 2 of Track 3
 Time to Finish: 14.53 | Section 1 of Track 3 and Section 2 of Track 0
 Time to Finish: 15.42 | Section 1 of Track 3 and Section 2 of Track 1
 Time to Finish: 16.13 | Section 1 of Track 3 and Section 2 of Track 2
 Car stuck!!!! - Section 1 of Track 3 and Section 2 of Track 3

Results of combining Tracks:

Time to Finish: 14.51 | Average of Track 0 and Track 0
 Time to Finish: 14.09 | Average of Track 0 and Track 1
 Time to Finish: 13.66 | Average of Track 0 and Track 2
 Time to Finish: 14.01 | Average of Track 0 and Track 3
 Time to Finish: 15.41 | Average of Track 1 and Track 1
 Time to Finish: 13.44 | Average of Track 1 and Track 2
 Time to Finish: 13.79 | Average of Track 1 and Track 3
 Time to Finish: 16.15 | Average of Track 2 and Track 2
 Time to Finish: 16.4 | Average of Track 2 and Track 3
 Car stuck!!!! - Average of Track 3 and Track 3

Abbildung 5.1: Ergebnis eines Tests mit 4 von User erzeugten Rennlinien.

cke ebenfalls in 14,5 Sekunden.

Mittel aus track 0 und track 3: Das Fahrzeug durchfährt die Strecke in 14 Sekunden.

Eine weitere Information, die uns diese Tabelle zeigt ist, dass durch ein Mitteln der beiden Rennlinien die KI noch schneller durch diese Strecke fährt. Durch solch ein Mitteln könnte die Problemstelle wie Abbildung 5.3 es zeigt eine (für die KI) noch bessere Rennlinie ergeben. Abbildung 5.6 zeigt eine aus track 0 und track 3 gemittelte Rennlinie. Diese Abbildung zeigt auch, dass die KI es nicht schafft, sich genau an die Rennlinie zu halten, sie aber trotzdem die Strecke gut durchfährt.

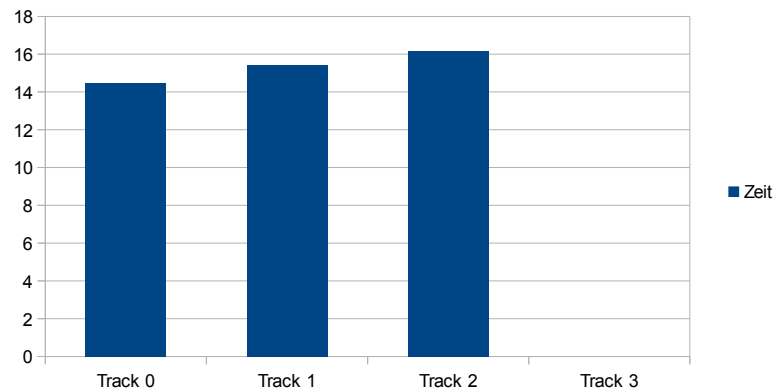


Abbildung 5.2: Benötigte Zeit der Rennlinien für eine Strecke.

Time to Finish: 14.53 | Section 1 of Track 3 and Section 2 of Track 0
 Time to Finish: 15.42 | Section 1 of Track 3 and Section 2 of Track 1
 Time to Finish: 16.13 | Section 1 of Track 3 and Section 2 of Track 2
 Car stuck!!!! - Section 1 of Track 3 and Section 2 of Track 3

Abbildung 5.3: Ergebnis des Tests durch Kombinationen mit Sektion 1 der Rennlinie 3.

Abschließend zeigt Abbildung 5.5 ein Mitteln aller Rennlinien mit der Rennlinie 0. Das Diagramm zeigt, dass jede mit track 0 gemittelte Rennlinie ein besseres Ergebnis liefert als die ursprüngliche Rennlinie 0.

Sucht man nun im Test nach dem besten Ergebnis, so sieht man, dass dies ein Schnitt zweier Rennlinien ist. Das Mittel der Rennlinien 1 und 2 ergibt eine Zeit von 13,4 Sekunden. Im Vergleich dazu haben die ursprünglichen Rennlinien nur eine Zeit von 15,43 Sekunden und 16,17 Sekunden erreicht. Ein Kombinieren der Sektionen ist in Bezug auf ihrer benötigten Zeit für diese Teststrecke nicht positiv ausgefallen. Die Kombination aus Sektion 1 der Rennlinie 2 und Sektion2 der Rennlinie 1 ergab eine sehr ähnliche Zeit wie die ursprüngliche Rennlinie 1. Ein Aspekt, der dabei nicht außer Acht gelassen werden darf, ist dass es durch solche Rennlinien zu mehr Variation im Fahrverhalten der KI kommt. Dies ist neben dem Verbessern von Rennlinien ein weiterer Punkt, der Ziel dieser Arbeit ist.

Abbildung 5.7 zeigt das Ergebnis von zwei weiteren Tests, in denen ebenfalls bessere Zeiten erzielt werden konnten.



Abbildung 5.4: Problemstelle der Rennlinie 3 in Sektion 2.

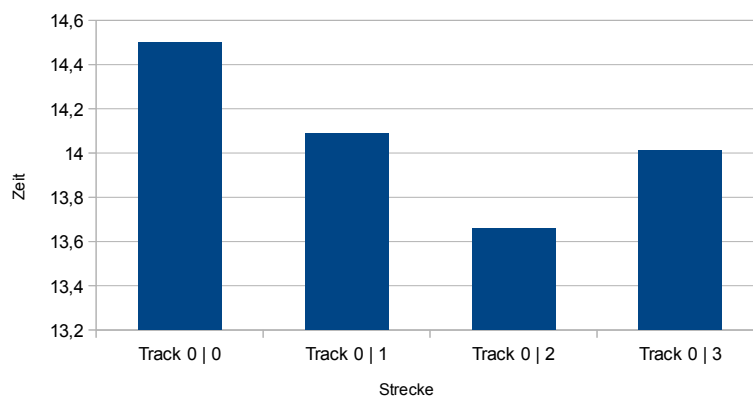


Abbildung 5.5: Benötigte Zeit der gemittelten Rennlinien.

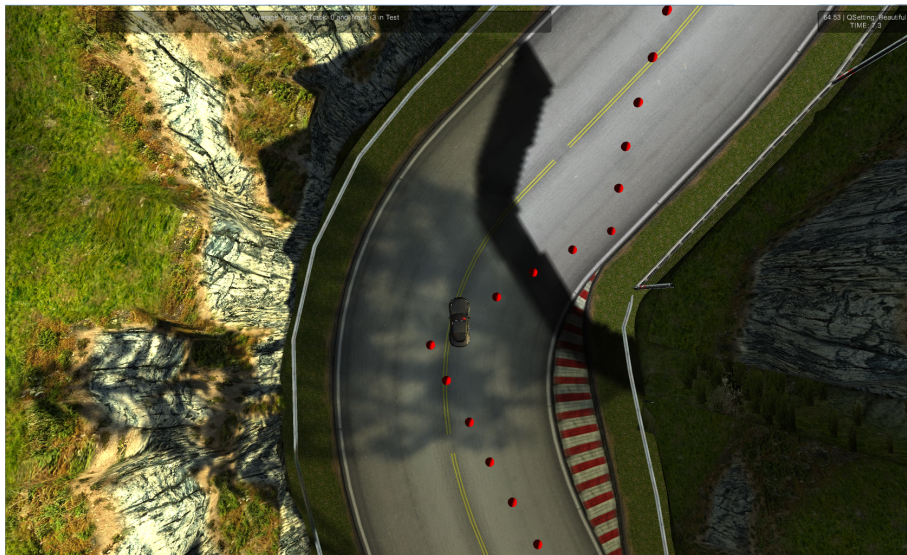


Abbildung 5.6: Rennlinie entstanden durch ein Mitteln von track 0 und track 3.

Time to Finish: 14.45		Section 1 of Track 0 and Section 2 of Track 0
Time to Finish: 16.46		Section 1 of Track 0 and Section 2 of Track 1
Time to Finish: 14.51		Section 1 of Track 0 and Section 2 of Track 2
Time to Finish: 14.81		Section 1 of Track 0 and Section 2 of Track 3
Time to Finish: 14.53		Section 1 of Track 1 and Section 2 of Track 0
Time to Finish: 16.71		Section 1 of Track 1 and Section 2 of Track 1
Time to Finish: 14.43		Section 1 of Track 1 and Section 2 of Track 2
Time to Finish: 14.68		Section 1 of Track 1 and Section 2 of Track 3
Time to Finish: 14.55		Section 1 of Track 2 and Section 2 of Track 0
Time to Finish: 16.31		Section 1 of Track 2 and Section 2 of Track 1
Time to Finish: 14.44		Section 1 of Track 2 and Section 2 of Track 2
Time to Finish: 14.61		Section 1 of Track 2 and Section 2 of Track 3
Time to Finish: 14.53		Section 1 of Track 3 and Section 2 of Track 0
Time to Finish: 16.34		Section 1 of Track 3 and Section 2 of Track 1
Time to Finish: 14.56		Section 1 of Track 3 and Section 2 of Track 2
Time to Finish: 14.69		Section 1 of Track 3 and Section 2 of Track 3
Time to Finish: 14.46		Average of Track 0 and Track 0
Time to Finish: 14.46		Average of Track 0 and Track 1
Time to Finish: 13.69		Average of Track 0 and Track 2
Time to Finish: 14.19		Average of Track 0 and Track 3
Time to Finish: 16.32		Average of Track 1 and Track 1
Time to Finish: 14.26		Average of Track 1 and Track 2
Time to Finish: 14.5		Average of Track 1 and Track 3
Time to Finish: 14.37		Average of Track 2 and Track 2
Time to Finish: 13.75		Average of Track 2 and Track 3
Time to Finish: 14.65		Average of Track 3 and Track 3

Time to Finish: 15.24		Section 1 of Track 0 and Section 2 of Track 0
Time to Finish: 13.9		Section 1 of Track 0 and Section 2 of Track 1
Time to Finish: 13.19		Section 1 of Track 0 and Section 2 of Track 2
Time to Finish: 20.12		Section 1 of Track 0 and Section 2 of Track 3
Time to Finish: 15.18		Section 1 of Track 1 and Section 2 of Track 0
Time to Finish: 13.87		Section 1 of Track 1 and Section 2 of Track 1
Time to Finish: 13.12		Section 1 of Track 1 and Section 2 of Track 2
Time to Finish: 20		Section 1 of Track 1 and Section 2 of Track 3
Time to Finish: 15.27		Section 1 of Track 2 and Section 2 of Track 0
Time to Finish: 13.97		Section 1 of Track 2 and Section 2 of Track 1
Time to Finish: 13.27		Section 1 of Track 2 and Section 2 of Track 2
Time to Finish: 20.03		Section 1 of Track 2 and Section 2 of Track 3
Time to Finish: 15.17		Section 1 of Track 3 and Section 2 of Track 0
Time to Finish: 13.87		Section 1 of Track 3 and Section 2 of Track 1
Time to Finish: 13.12		Section 1 of Track 3 and Section 2 of Track 2
Time to Finish: 20.01		Section 1 of Track 3 and Section 2 of Track 3
Time to Finish: 15.22		Average of Track 0 and Track 0
Time to Finish: 13.65		Average of Track 0 and Track 1
Time to Finish: 13.15		Average of Track 0 and Track 2
Time to Finish: 15.23		Average of Track 0 and Track 3
Time to Finish: 13.87		Average of Track 1 and Track 1
Time to Finish: 13.1		Average of Track 1 and Track 2
Time to Finish: 15.3		Average of Track 1 and Track 3
Time to Finish: 13.26		Average of Track 2 and Track 2
Time to Finish: 13.93		Average of Track 2 and Track 3
Time to Finish: 19.99		Average of Track 3 and Track 3

Abbildung 5.7: Weitere Testergebnisse.

Kapitel 6

Fazit und Ausblick

6.1 Zusammenfassung

Im Rahmen dieser Arbeit wurde die Möglichkeit zur Verbesserung der KI aus Spieler gegebenen Rennlinien gezeigt. Dazu wurde auf wichtige Aspekte für die Entwicklung einer computergesteuerten Intelligenz in Computerspielen eingegangen. Möglichkeiten zur Steuerung einer künstlichen Intelligenz wurden beschrieben und erklärt. Anschließend ging es im Hauptteil der Arbeit um die Speicherung der Rennlinien in Form von Wegpunkten und der Ansteuerung dieser Punkte. Dazu wurde während der Fahrt des Spielers in gewissen Abständen Punkte, mit den für die KI wichtigen Information, erstellt und gespeichert. Zusätzlich wurden solche Rennlinien in Sektionen unterteilt, um im weiteren Schritt neue Rennlinien generieren zu können. Dazu wurden zwei Varianten umgesetzt und präsentiert. Eine Variante erzeugte aus zwei Rennlinien eine gemittelte Rennlinie, dessen Geschwindigkeit nicht gemittelt, sondern die der schnelleren Rennlinie verwendet wurde. Die zweite Variante unterteilte die Rennlinie in zwei Sektionen und kombinierte die Sektion einer Rennlinie mit einer Sektion einer Anderen. Mit Hilfe eines Tests, dessen Aufbau Abbildung 6.1 nochmals zeigt, wurden dann diese neu generierten Rennlinien getestet und anschließend analysiert. Ziel war es aus gegebenen Rennlinien neue und bessere zu erzeugen, um somit eine herausfordernde und Spaß bringende KI auch mit dem besser werden des Spielers zur Verfügung stellen zu können.

6.2 Ergebnis

Das Ergebnis zeigte, dass durch ein Kombinieren und ein Mitteln der von Spieler kommenden Rennlinien möglich ist, eine neue Rennlinie zu erzeugen, die zu einem besseren Ergebnis führt. Durch die Verwendung dieser Varianten ist es auch möglich schwierige Streckenabschnitte die ein Spieler schlecht durchfahren ist (siehe Abbildung 5.3) auszugleichen oder einfach durch den

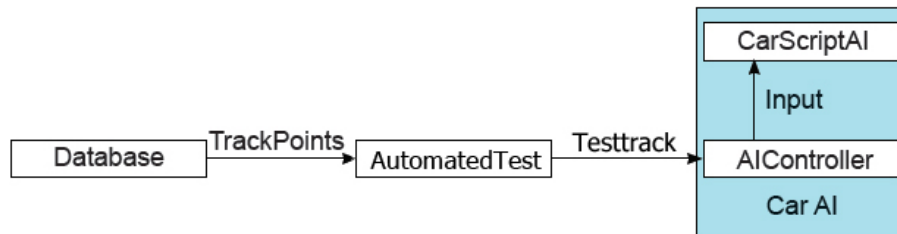


Abbildung 6.1: Aufbau des Testes zur Generierung neuer Rennlinien.

Tausch des Sektors mit einem Sektor einer anderen Rennlinie wegzubekommen. Dies führte zum Ziel, dass der computergesteuerte Gegner mit der vom Test erzeugten Rennlinie schneller die Strecke durchfuhr als mit den ursprünglichen Rennlinien. Dieser ganze Test passierte dabei unter der Verwendung von *Ghosts* als Gegner, das bedeutet, dass der computergesteuerte Gegner nicht mit dem Spieler kollidieren konnte. Es wurde die Verwendung von *Ghosts* gewählt, um ein deutlicheres Ergebnis und somit eine bessere Aussage über das Ergebnis tätigen zu können. Des Weiteren ist das Verhalten der KI bei Kollisionen mit Spielern zu umfangreich um dies in diese Arbeit miteinbeziehen und implementieren zu können. Die Verwendung von Daten, welche vom Spieler erzeugt werden, bringt weitere Vorteile mit sich. Wie im Abschnitt 3.3.2 genannt, werden heute noch großteils Rennlinien für eine KI von *Gamedesignern* von Hand gesetzt. Da die Rennlinien aber nun von Spieler kommen, könnte man beispielsweise unter Verwendung einer Beta Phase diese Daten zur Erzeugung von Rennlinien sammeln und auf das Setzen von Hand verzichten. Im Gegenzug ist bei dieser Methode mit einem höheren Implementierungsaufwand zu rechnen.

Des Weiteren kommen durch die Verwendung dieser Methode mehr Rennlinien zustande, an die sich eine KI halten kann und bietet somit weitaus mehr Variation im Fahrverhalten der computergesteuerten Gegner. So kann man aus einer Vielzahl an Rennlinien, welche eine ähnliche oder vielleicht auch die gleiche Zeit für eine Rennstrecke benötigen, wählen. Dies hat zur Folge, dass mehrere Gegner gleich lange für eine Strecke benötigen können, ohne die gleiche Rennlinie fahren zu müssen. Durch diese vielen Möglichkeiten an Rennlinien und deren unterschiedlichen Zeiten für das Durchfahren einer Strecke, ist es möglich auf Stufen im Schwierigkeitsgrad der KI zu verzichten und einen stufenlosen Übergang zwischen leichten und schweren Gegner zu erzeugen.

Auch auf *rubberbanding*, erklärt im Kapitel 2, kann durch die große Auswahl an Rennlinien teilweise verzichtet werden, da die KI auf die durch den Test neu generierten und besseren Rennlinien zurückgreifen kann.

Die Verwendung des in dieser Arbeit beschriebenen Ansatzes zur Ge-

nerierung von Rennlinien bringt aber auch seine Nachteile. Zur Erzeugung von neuen Rennlinien werden bereits Rennlinien benötigt. Sollte man also ein Rennspiel mit solch einer Methode auf den Markt bringen, so müssten für die KI bereits Rennlinien bereitstehen, auf die sie zugreifen kann. Diese Rennlinien müssten dann während einer Testphase durch Entwickler und Testpersonen bereits vor dem Verkauf des Spieles entstehen. Man könnte auch eine *Beta Version* zur Sammlung von Daten veröffentlichen, in der man alleine oder nur gegen andere Spieler fährt. Um die Useranzahl bei der Einzelspieler Variante hochzuhalten, könnte man zusätzlich Rekordzeiten, die dann in öffentlichen Netzwerken wie Facebook veröffentlicht werden können, einführen.

Das System hat den Nachteil, dass auch noch Zugang zu einer online Datenbank benötigt, um die für die KI benötigten Rennlinien downloaden zu können. Dieser Download erfolgt jeweils beim Starten eines Rennens wie im Abschnitt 4.4.2 beschrieben. Hierbei könnte man sich Systeme überlegen, die keine dauerhafte Internetverbindung benötigen.

Ein weiterer Nachteil den dieses System mit sich bringt, ist die Größe die solch eine Datenbank benötigen kann, um diese Unmengen an Daten speichern zu können. Denn je mehr Spieler dieses Rennspiel spielen, desto mehr Rennlinien werden erzeugt und gespeichert. Hier müsste man im nächsten Schritt ein System zur Auswahl an Rennlinien erstellen, welches intelligent über die Aufnahme der Rennlinien in die Datenbank entscheidet.

6.3 Ausblick

Dieser Abschnitt handelt von den Erweiterungsmöglichkeiten und mögliche Verbesserungen die für diesen Ansatz noch umgesetzt werden könnten. Im vorigen Abschnitt wurden bereits mögliche Probleme aufgezählt, welche hier nochmal genauer behandelt werden.

Ein vorhin genanntes mögliches Problem ist die Größe der Datenbank, die in Abhängigkeit der Useranzahl zu viele ähnliche Rennstrecken speichern würde. Ein Spieler allein könnte bei einer Strecke jedes mal an derselben Stelle von der Straße abkommen bevor er das Ziel erreicht. Diese würde in so einem Fall auch jedes mal in der Datenbank aufgenommen werden. Hierzu könnte man sich ein System überlegen, welches Rennlinien in Kategorien einteilt und speichert. Dabei könnten verschiedenste Aspekte miteinbezogen werden. Eines der wichtigsten hierbei wäre die Zeit, die ein computergesteuerter Gegner für solch eine Rennstrecke benötigt, da dies das Hauptkriterium für eine gute oder schlechte KI ist. Anhand dieser Zeit könnte man ab einen gewissen Pool an Rennlinien jede weiter einfach nicht mehr in die Datenbank aufnehmen. So kann man zum Beispiel ab 100 Rennlinien für eine Strecke mit einer benötigten Zeit von zwischen 1 Minute und 1 Minute und 5 Sekunden jede weitere ablehnen und nur noch schlechtere oder bessere der

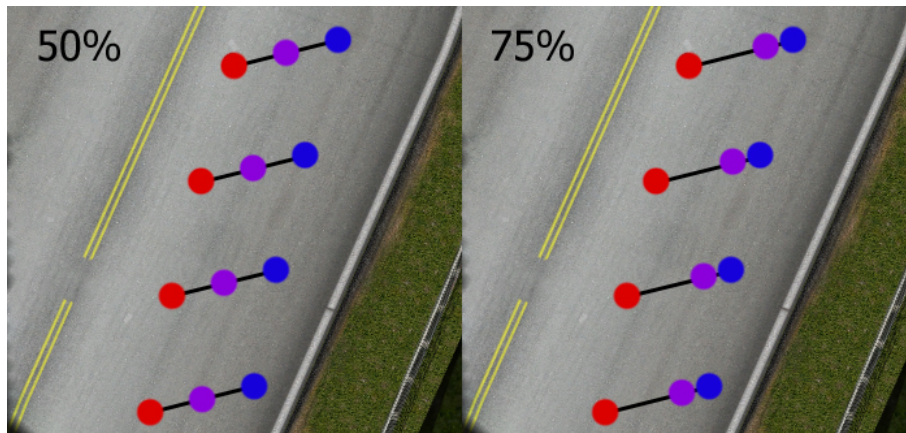


Abbildung 6.2: Mögliche Gewichtungen von Rennlinien mit deren Ergebnis

Datenbank hinzufügen. Des Weiteren könnte man auch noch die Rennlinien anhand ihrer Durchschnittsgeschwindigkeit der einzelnen Sektoren unterteilen um somit mehr Varianz im Fahrverhalten der KI einbringen zu können. So hätte man letztendlich Rennlinien in denen die KI anfangs oder im letzten Streckenabschnitt besser fährt. Über die Anzahl der Wegpunkte in einem Sektor könnte man schließen ob die KI eine längere oder kürzere Strecke zu fahren hat. Dieser Punkt führt dabei nicht zu einem Besserwerden der KI, sondern dient lediglich der Optimierung der Datenspeicherung.

Auch bei den zwei Methoden zur Erstellung neuer Rennlinien gibt es Möglichkeiten, die zu einer Verbesserung führen könnten. Das Mitteln der Rennlinien geschieht momentan durch ein Mitteln der Position zweier Wegpunkte, wobei die Geschwindigkeit nicht gemittelt, sondern der höhere Wert genommen wird. Hier könnte man versuchen die Rennlinie die schneller durch eine Strecke führt eine höhere Gewichtung geben. Abbildung 6.2 zeigt dies. Im linken Bild wurden beide Rennlinien gleich stark bewertet. Das rechte Bild zeigt eine Gewichtung von 75 Prozent für die blaue Rennlinie. Nicht nur bei der Wahl der Gewichtung der Position der Wegpunkte kann es zu Verbesserungen kommen. Man könnte auch bei der Wahl der Geschwindigkeit mögliche Tests versuchen. Als Beispiel könnte man, falls der Test mit der höheren Geschwindigkeit fehlschlägt und das Auto stecken bleibt, die Geschwindigkeit schrittweise um einen bestimmten Wert verringern und wiederholen. All diese Möglichkeiten zur Verbesserung der Rennlinie bzw. der KI können im Test mit aufgenommen werden. Dies würde jedoch zu einer Erhöhung der benötigten Zeit für Tests führen.

Beim Erstellen neuer Rennlinien durch Sektoren hängt das Ergebnis stark von der Einteilung der Rennstrecke ab. Hier könnte man ein System entwickeln, welches die Rennstrecke analysiert und anhand der Schwierigkeit gewisser Passagen diese in Sektionen unterteilt. Dabei sollte man auf die

Stellen achten, in denen man die Strecke in Sektoren unterteilt. Ein Wechsel der Sektoren in einer Kurve führt dabei häufiger zu Problemen als ein gerader Streckenabschnitt, da das Fahrzeug in einer Kurve schwieriger die Spur ändern kann. So kann beispielsweise die KI in einer Kurve, bei der sie bereits am äußeren Rand fährt, nicht aufgrund einer Änderung der Sektion plötzlich zum inneren Rand wechseln.

Abschließend könnte man noch analysieren wie weit voraus eine KI bei der Wahl des anzusteuernenden Wegpunktes sehen soll. In dieser Arbeit geschah dies mit einem Mindestwert zu dem ein Drittel der Geschwindigkeit des Fahrzeuges hinzuaddiert wurde. Abbildung 4.12 zeigt dies anhand einer roten Linie. Dies könnte zu Beginn durch ein Testsystem so gewählt werden, dass die KI eine Rennlinie an sich bestmöglich befährt.

Anhang A

Inhalt der CD-ROM/DVD

Format: CD-ROM, Single Layer, ISO9660-Format

A.1 Masterarbeit

Pfad: /

Wagner_Werner_2014 Masterarbeit (Gesamtdokument)

A.2 Projektdateien

Pfad: /project

/executables/KITest.exe Executable des Tests

/executables/racing_game.sql Datenbank

/executables/readme.txt Readme

/unity_carracing.zip . . Projekt mit Source Code

/important_scripts/* . Wichtigsten Source Code Files

A.3 Online-Quellen

Pfad: /web-pdfs

*.pdf Kopien der Online-Quellen

Quellenverzeichnis

Literatur

- [1] Mat Buckland. *Programming Game AI by Example*. Jones & Bartlett Publishers, 2004 (siehe S. 13).
- [2] Daniele Loiacono. „Learning, Evolution and Adaptation in Racing Games“. In: *Proceedings of the 9th Conference on Computing Frontiers*. CF '12. Cagliari, Italy: ACM, 2012, S. 277–284 (siehe S. 20).
- [3] K. Michels u. a. *Fuzzy-Regelung: Grundlagen, Entwurf, Analyse*. Springer-Verlag, 2002 (siehe S. 16).
- [4] I. Millington und J.D. Funge. *Artificial Intelligence for Games*. Morgan Kaufmann. Morgan Kaufmann/Elsevier, 2009 (siehe S. 4, 6, 7, 12–14).
- [5] S. Rabin. *AI Game Programming Wisdom 2*. AI Game Programming Wisdom. Charles River Media, Incorporated, 2004 (siehe S. 3, 5, 6).
- [6] S. Rabin. *AI Game Programming Wisdom 3*. AI Game Programming Wisdom. Charles River Media, 2006 (siehe S. 15).
- [7] C.I. Tan u. a. „An AI tool: Generating paths for racing game“. In: *2008 International Conference on Machine Learning and Cybernetics*. Bd. 6. Kunming, 2008, S. 3132–3137 (siehe S. 20).
- [8] Jung-Ying Wang und Yong-Bin Lin. „An effective method of pathfinding in a car racing game“. In: *The 2nd International Conference on Computer and Automation Engineering (ICCAE)*. Bd. 3. Singapore, 2010, S. 544–547 (siehe S. 18, 19).

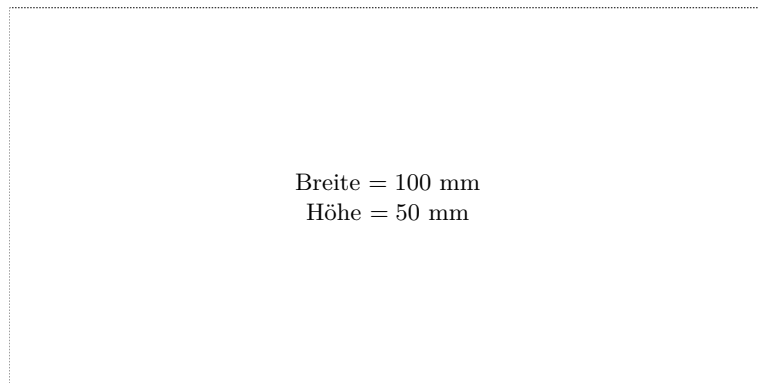
Online-Quellen

- [9] URL: <http://www.tikihumor.com/4847/portal-turret-hallway-2/portal-turret-hallway-3/> (besucht am 20.08.2013) (siehe S. 5).
- [10] URL: http://en.wikipedia.org/wiki/Racing_video_game (besucht am 22.08.2013) (siehe S. 8, 10).

- [11] URL: <http://web.archive.org/web/20091016060221/http://www.ugo.com/channels/sports/features/racing/games.asp> (besucht am 25.08.2013) (siehe S. 8).
- [12] URL: http://en.wikipedia.org/wiki/Atari,_Inc. (besucht am 22.08.2013) (siehe S. 9).
- [13] URL: http://www.gamasutra.com/view/feature/192211/learning_to_love_handicaps_in_.php?page=1 (besucht am 29.08.2013) (siehe S. 8).
- [14] URL: <http://www.red3d.com/cwr/steer/> (besucht am 03.09.2013) (siehe S. 14).

Messbox zur Druckkontrolle

— Druckgröße kontrollieren! —



— Diese Seite nach dem Druck entfernen! —