KI in Rennspielen

Seminar „KI in Spielen“

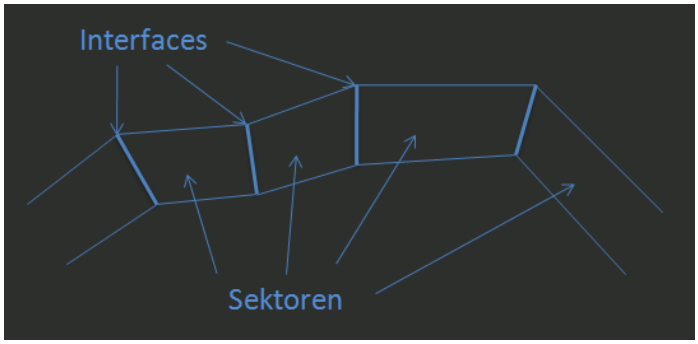
Yannick Block, Minf 2549

Inhalt

1 Die Strecke

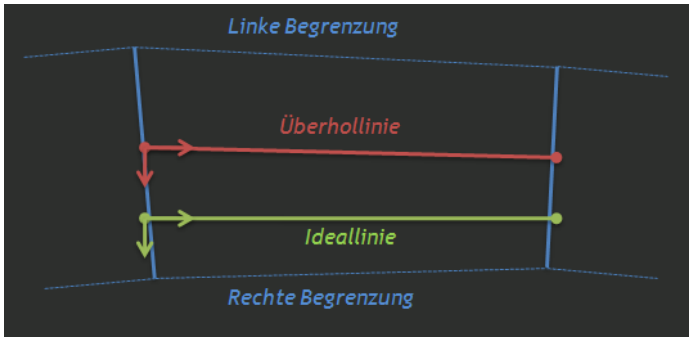
1. Die Strecke definieren

1.1.1 Sektoren und Interfaces



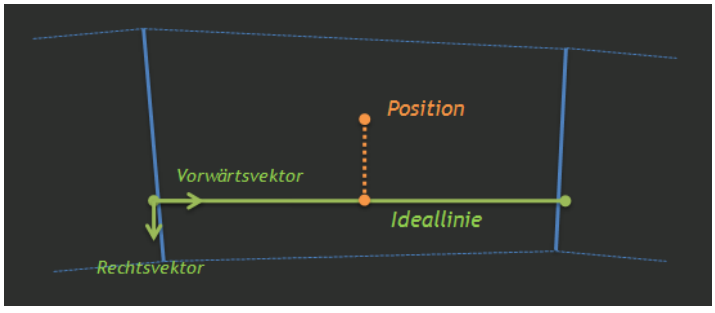
Eine Strecke wird eingeteilt in mehrere Sektoren. Zwischen zwei Sektoren gibt es je eine Schnittstelle (“Interface”). Falls die Streckenumgebung es zulässt, kann auch außerhalb der Sektorengrenzen gefahren werden.

1.1.2 Ideal- und Überhollinien



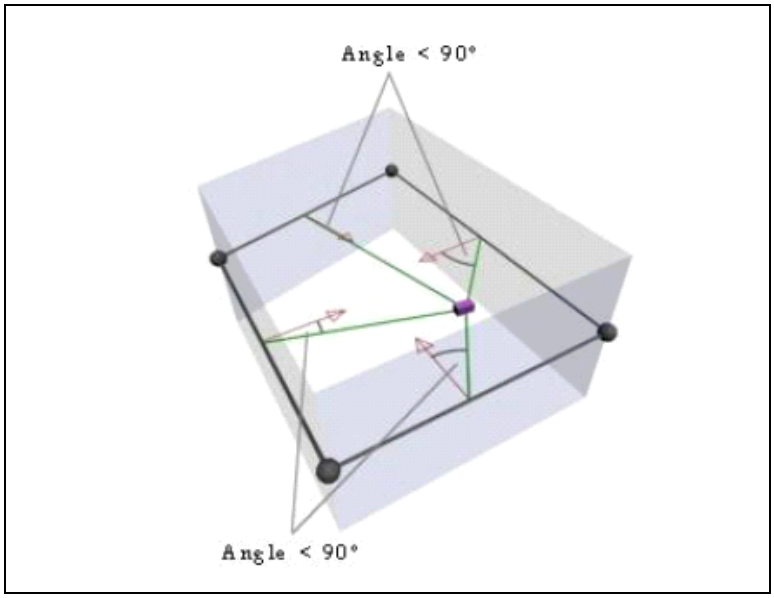
Auf jedem Interface wird ein Punkt für die Ideallinie und ein Punkt für eine potentielle Überhollinie (es wären aber auch mehrere denkbar) festgelegt. Verbindet man jeweils die Punkte zwischen den Interfaces, erhält man die Ideallinie bzw. Überhollinie.

1.1.3 Die Distanz berechnen



Jedem Sektor ist bekannt, wie weit er von der Start-/Ziel-Linie entfernt ist. So kann die Distanz leicht berechnet werden, in dem man die Ideal-Linien der Sektoren addiert.
Die Ideallinie setzt sich zusammen aus ihren Weltkoordinaten, ihrer Länge, dem Vorwärts- und dem Rechtsvektor. Der Vorwärtsvektor wird aus dem normalisierten Vektor zwischen Start- und Endpunkt der Linie (mit Y gleich null) erzeugt. Das Y Element wird vernachlässigt, da dies garantiert, dass Auto und Strecke auf einer Ebene liegen und auf diese Weise leichter gerechnet werden kann. Der Rechtsvektor steht senkrecht zum Vorwärtsvektor und wird benötigt, um den Abstand zur Ideallinie festzustellen.

1.1.4 Den aktuellen Sektor herausfinden



[SemKI]

Um die vier Sektorengrenzen zu markieren, werden Ebenen benutzt. Diese können relativ leicht erzeugt werden, in dem man jeweils drei Punkte benutzt: Den Startpunkt, den Endpunkt und ein Punkt, der vom Startpunkt aus etwas in Y-Richtung verschoben ist.
Um einfach testen zu können, ob ein Punkt innerhalb eines Sektors liegt, muss sichergestellt werden, dass ein Sektor konvex ist. Dies wird am einfachsten erreicht, in dem man bei jeder Ebene jeweils testet, ob die beiden anderen Punkte auf der positiven Seite der Ebene liegen.
So weiß die KI durch diese Methode auch, dass ein Auto sich in einem bestimmten Sektor befindet, wenn es sich auf der positiven Seite aller vier Grenzebenen befindet.
Um die zurückgelegte Distanz innerhalb eines Sektors bestimmen zu können, berechnet man die zurückgelegte Distanz parallel zur Ideallinie (S. 442), in dem man das Skalarprodukt aus dem Vorwärtsvektor und dem Differenzvektor zwischen Ideallinie und Position bildet.

1.2 Informationen über die Strecke

1.2.1 Pfadtyp

Es kann verschiedene Pfadtypen geben, wie z.B. normal, Abkürzung, Umweg, Waffen-Aufsammel-Weg, Serpentine, etc. Die KI benutzt diese Informationen immer wenn sie eine Entscheidung treffen muss, wie z.B. bei Weggabelungen.

1.2.2 Terraintyp

Es sind verschiedene Terraintypen denkbar. So könnte es z.B. Terrain geben, welches nur für Geländewagen in Frage kommt und von anderen Autos gemieden werden muss. So müssten die anderen Autos diese Strecke meiden, auch wenn sie kürzer ist.

1.2.3 Wände

Da manche Strecken auch von Wänden umgeben sein können (wie z.B. Tunnel), muss die KI darüber informiert werden, um den notwendigen Sicherheitsabstand zu den Wänden zu halten.

1.2.4 Haarnadel-Kurven

Die KI sollte ebenfalls über Haarnadel-Kurven in beide Richtungen informiert werden. Haarnadel-Kurven behindern das Auto jedoch nicht in seiner Fahrt.

1.2.5 Beschleunigung/Bremsen

Um der KI in besonders schwierigen Streckenteilen zu helfen, ist ein Brems- bzw. Beschleunigungs-Wert durchaus hilfreich. Die Werte gehen von -1.0 (Vollbremsung) bis +1.0 (Vollgas).

1.2.6 Fazit

Wenn man der Rennstrecke relativ viele Informationen mitgibt, kann man die Komplexität der KI verringern. Häufig lassen sich die Berechnungen auch von 3D auf 2D reduzieren, um weiter an Komplexität zu sparen.
Die Ideallinien ließen sich noch verbessern durch Verwendung von nichtlinearer Interpolation, wie z.B. Catmull-Rom splines, bei denen die Kurven durch die Kontrollpunkte laufen.
Obwohl die KI such ungefähr an der Ideallinie orientiert, folgt sie ihr nicht ganz genau. Um die KI etwas realistischer aussehen zu lassen, sollte sie agieren, als ob sie Input von einem menschlichen Spieler bekommen würde, der einen Joystick, ein Keyboard o.ä. benutzt.

2 Spiellogik

2.1 Das Framework

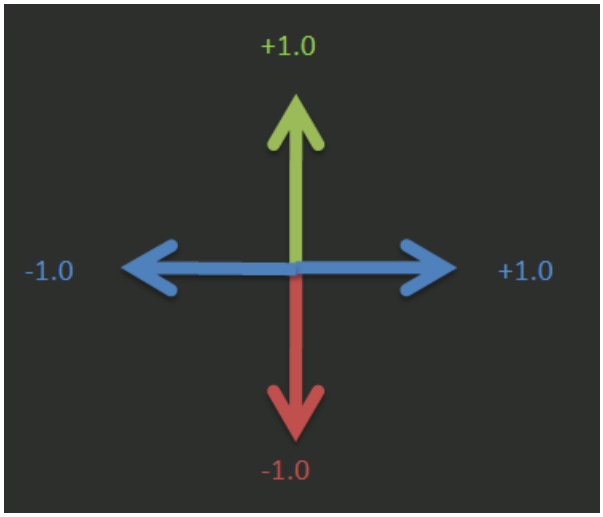
2.1.1 Endliche Automaten

Es werden endliche Automaten eingesetzt. Endliche Automaten wechseln immer zwischen Zuständen. Für ein Rennspiel benutzen wir folgende Zustände: STATE_STARTING_GRID, STATE_RACING, STATE_RECOVER_TO_TRACK, STATE_AIRBORNE und STATE_OFF_TRACK.

2.1.2 Feste Zeitschritte

Es müssen feste Zeitschritte eingesetzt werden, damit die KI unabhängig von der Plattform oder der Frame Rate überall gleich läuft. Um dies zu realisieren, bekommt die jeweilige Methode die vergangene Zeit seit dem letzten Update.

2.1.3 Steuerung des Autos



Die Steuerung des Autos setzt sich aus der Lenkung und der Beschleunigung zusammen. Sowohl die Lenkung als auch die Beschleunigung bewegt sich in Werten zwischen -1.0 (links bzw. Vollbremsung) und +1.0 (rechts bzw. Vollgas).

2.1.4 Reduzierung auf 2D

Auch wenn die Strecke uneben sein kann, bleibt der Pfad 2D. Nutzt man diese Feststellung aus, lassen sich viele Berechnungen von 3D auf 2D reduzieren, in dem man das Y Element auf null setzt und dann normalisiert.

2.2 Das Rennen

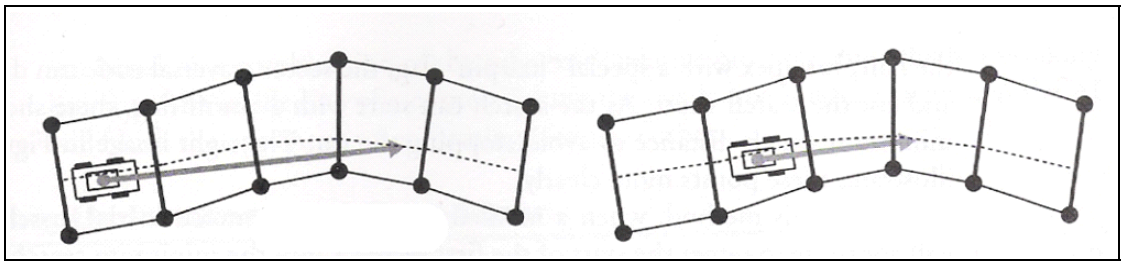
2.2.1 Der Start

Vor dem Start des Rennens werden die Werte der KI initialisiert. So wird auch gespeichert, in welchem Sektor sich das Auto gerade befindet. Für ein Auto wird immer der aktuelle Sektor und der letzte gültige Sektor gespeichert, falls das Auto von der Strecke fliegt o.ä. Während die Autos auf „Grün“ warten, ist der endliche Automat auf STATE_STARTING_GRID gestellt, sobald das Rennen beginnt dann auf STATE_RACING. Dieser ist gleichzeitig der komplizierteste Zustand.

2.2.2 Traversieren der Sektoren

Damit die KI nicht gegen jeden Sektor testen muss, ob das Auto sich in ihm befindet, wird der letzte Sektor gespeichert, in dem sich das Auto befand. Die Sektoren werden als doppelt verkettete Liste gespeichert. So wird vom zuletzt gespeicherten Sektor aus gesucht, bis das Auto in einem neuen Sektor gefunden wurde.
Da das Auto sich aber auch außerhalb aller Sektoren befinden kann, muss der endliche Automat auf STATE_RECOVER_TO_TRACK gesetzt werden und die KI muss auf die Strecke zurück finden.

2.2.3 Vorausschauendes Fahren



[AIPW]

Schnelles Fahren *Langsames Fahren*

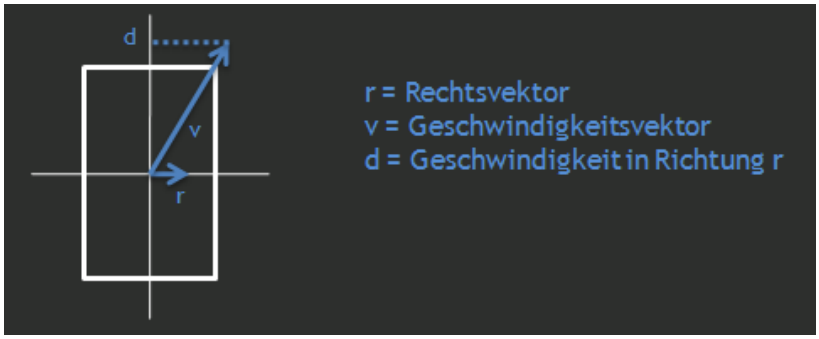
Um eine möglichst gelungene KI zu erzeugen, muss die KI etwas Voraussicht zeigen. Die KI muss also antizipieren, was gleich passiert und nicht nur auf Ereignisse reagieren, die bereits eingetreten sind. So muss z.B. bei einer Kurve rechtzeitig gebremst und die Geschwindigkeit an die Kurve angepasst werden.
Das vorausschauende Fahren kann durch Traversierung der kommenden Sektoren erreicht werden. Die Voraussicht wird an die Geschwindigkeit des Autos angepasst. Dies hat den Vorteil, dass bei geringer Geschwindigkeit nur die lokale Region betrachtet wird und eine höhere Präzision erhält. Gleichzeitig hat dies bei hoher Geschwindigkeit den Vorteil, dass Objekte früh gesehen und somit umfahren werden können.
Dies kann jedoch bei scharfen Kurven den unangenehmen Nebeneffekt haben, dass das Auto abkürzen möchte. Um dies zu verhindern, müssen scharfe Kurven als solche gekennzeichnet werden. Wird eine solche Markierung entdeckt, kann sich die KI die weitere Suche sparen. Des

weiteren bewirkt die Markierung, dass die KI auf eine angemessene Geschwindigkeit herab bremst und dann der Kurve folgt, wenn sie nah genug ist.

2.2.4 Überholen

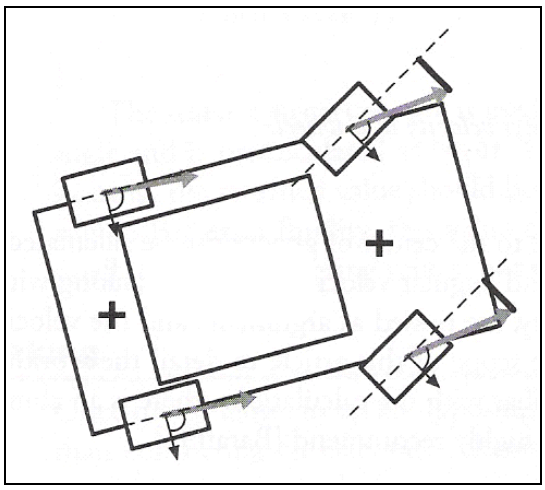
Sobald die KI auf gegnerische Autos trifft, muss sie einen Weg um diese herum finden. Hierfür benutzt sie die bereits angesprochenen Überhollinien. Denkbar wären auch mehrere Überhollinien, z.B. links und rechts. So könnte die KI sich für eine Überhollinie entscheiden, je nachdem wo das Auto vor ihr ist oder welche Überhollinie näher an der Innenkurve liegt.

2.2.5 Stabilität des Autos



Die Stabilität des Autos kann durch Vergleiche der Seitwärts-Geschwindigkeiten der jeweiligen Reifen festgestellt werden. Die Seitwärts-Geschwindigkeit lässt sich durch das Skalarprodukt aus dem Geschwindigkeitsvektor des Reifens und dem Rechtsvektor im Einheitsmaß berechnen. Um Unter- und Übersteuern zu umgehen, werden jeweils die Seitengeschwindigkeiten vorne und hinten gemittelt.

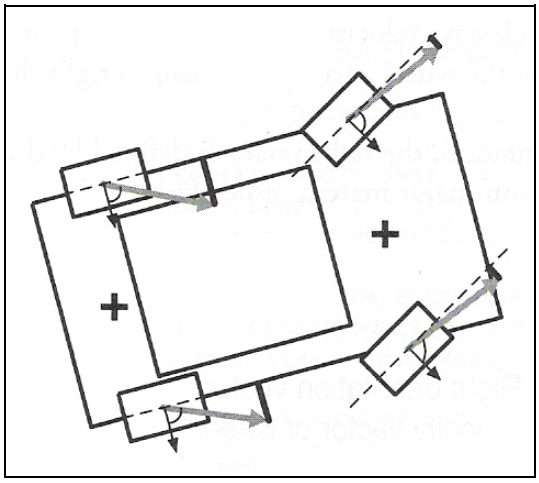
2.2.6 Untersteuern



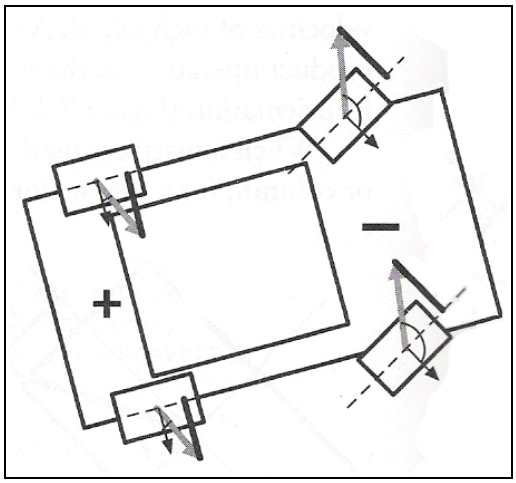
[AIPW]

Untersteuern tritt auf, wenn das Auto versucht einzuschlagen, aber trotzdem geradeaus fährt. Dies tritt auf, wenn die Seitwärsvektoren vorne und hinten die gleiche Richtung haben. Je größer die Seitwärtsvektoren vorne sind im Verhältnis zu den hinteren, desto mehr Untersteuern findet statt.

2.2.7 Übersteuern



[AIPW]



[AIPW]

Beim Übersteuern reagiert das Auto beim Einschlagen viel zu stark und droht, auszubrechen. Übersteuern kann in zwei Fällen auftreten: Zum einen, wenn – wie beim Untersteuern – die Seitwärtsvektoren in die gleiche Richtung gehen, aber hinten stärker als vorne. Der zweite Fall tritt ein, wenn das Auto sich bereits dreht und die Seitwärtsvektoren in verschiedene Richtungen gehen. Je größer die Differenz, desto größer das Übersteuern. Die Werte müssen zur Korrektur dementsprechend angepasst werden.

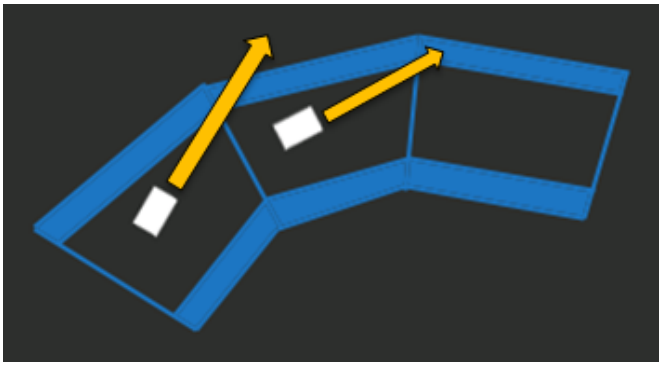
2.2.8 Das Auto stabilisieren

Tritt Über- oder Untersteuern auf (siehe 2.2.6 und 2.2.7), muss das Auto stabilisiert werden. Wie stark korrigiert wird, hängt von der Stärke der Instabilität ab. Es wird jedoch nicht nur die Lenkung angepasst, sondern die Beschleunigung wird genau so viel verringert, wie die Lenkung korrigiert wird. Wie genau korrigiert wird, hängt von der Art der Instabilität ab. Tritt Untersteuern auf, muss z.B. wie folgt korrigiert werden:

Korrektur = | (GeschwVorne + GeschwHinten) | / Untersteuerungswert

Der Untersteuerungswert sorgt dafür, dass der Korrekturwert zwischen 0 und 1 liegt, da sonst die Gefahr besteht, dass das Auto „überkorrigiert“ wird.

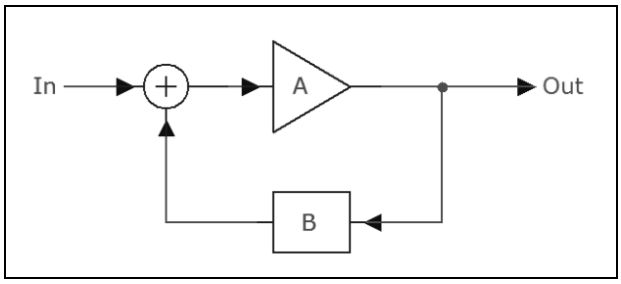
2.2.9 Wände



Die mögliche Kollision mit Wänden wird getestet, in dem berechnet wird, wo das Auto sich in Zukunft befände, würde es die aktuelle Richtung und Geschwindigkeit beibehalten. Es wird eine bestimmte Zeit draufgerechnet (z.B. 0,25 Sek.). Es wird der Sektor der „neuen“ Position ermittelt und geprüft, ob die Position sich nahe einer

Wand befindet und ob korrigiert werden muss. Wenn ein Null-Pointer zurückgegeben wird, wird das Auto voraussichtlich von der Strecke fliegen, die Bremen sollten also auf -1.0 gesetzt werden. Damit das Auto nicht ins Schleudern gerät, sollte die Steuerung auf „geradeaus“ gesetzt werden. Da das Auto Geschwindigkeit verliert, wird die Berechnung das Auto irgendwann in einem Sektor sehen und der Wand-Vermeidungs-Algorithmus wird es von der Wand weglenken.

2.2.10 Feedback Loops



[WIKI-FL]

Feedback Loops wurden aus der Steuerungs- und Regelungstechnik übernommen und dienen dem Zweck, einen Ist-Wert einem Soll-Wert anzunähern. Dies hat bei Rennspielen den Vorteil, dass es dem Spiel mehr „Echtheit“ verleiht. Wenn ein Fahrer z.B. beschließt zwecks eines Überholvorgangs auszuscheren, würde er das Auto zur Seite lenken. Die Feedback Loops bewirken nun, dass der neue Wert nicht einfach gesetzt wird, sondern sich langsam annähert.

2.3 Besondere Zustände des endlichen Automaten

2.3.1 STATE_AIRBORNE

Bei Stecken, die z.B. Sprünge beinhalten, wird ein Zustand „in der Luft“ gebraucht. Ist dieser Zustand eingetreten, sollte die Steuerung auf „geradeaus“ und eine hohe „Geschwindigkeit“ gestellt werden, da das Auto sonst bei der Landung möglicherweise ins Schleudern gerät. Sobald die Räder wieder Kontakt mit dem Boden haben, wird der Zustand zurück auf STATE_RACING gesetzt.

2.3.2 STATE_OFF_TRACK

Wenn das Auto von der Strecke abgekommen ist, muss es zur Strecke zurück finden. Dies geschieht, in dem sich das Auto in Richtung des zuletzt gespeicherten Sektors bewegt. Hierbei sollte eine größere Vorausschau-Distanz gewählt werden, damit das Auto in einem spitzeren Winkel auf die Strecke zurückkommt, was auch eleganter aussieht.

2.4 Anpassen des Spiels

Da Rennspiele Spaß machen sollen, sollten sie ggf. an die Fähigkeiten des Spielers angepasst werden.

2.4.1 Das Spiel einfacher machen

Wenn die KI „zu gut“ ist für den Spieler, gibt es mehrere Methoden, dies so zu korrigieren, dass es dem menschlichen Spieler nicht eklatant auffällt. Man könnte z.B. die Top Speed in Proportion zur Position im Feld begrenzen. Eine andere Methode ist, vor Kurven früher zu bremsen und langsamer herauszubeschleunigen. Eine dritte Methode wäre, die KI absichtlich längere Strecken oder ungünstige Terraintypen wählen zu lassen bei Weggabelungen. Als letzte Methode käme in Frage, bei Rennspielen mit Waffeneinsatz den Spieler mit Waffenzuteilungen zu bevorzugen und die Autos der KI einander abschießen zu lassen.

2.4.2 Das Spiel schwerer machen

Möchte man das Spiel schwerer machen, muss die Schritte genau entgegengesetzt zu 2.4.1 machen, also die Top Speed erhöhen, richtige Terraintypen wählen, die KI mit guten Waffen bestücken und den menschlichen Spieler abschießen lassen, die NPCs einander abschießen lassen, etc.

3 Die KI trainieren

3.1 Die Ideallinie finden

Die Ideallinie findet man am besten, in dem man die beste Runde eines menschlichen Spielers nimmt. Dafür müssen die Punkte auf den Interfaces gespeichert werden, also da, wo das Auto in den neuen Sektor eintritt. Dies wird zum Zeitpunkt des Sektorwechsels erledigt, indem man den Schnittpunkt zwischen der Geraden (neue Position – letzte Position) und der Ebene (das Interface) ermittelt.

3.2 Die Einstellungen am Auto

An jedem Auto der KI sind diverse Voreinstellungen vorzunehmen, wie z.B. Bremswege, Balance, etc. Da im Allgemeinen mit dem besten Setup die besten Rundenzeiten gefahren werden, gilt dies als Indikator.
Um sich dem besten Setup anzunähern, bestimmt man für jeden Parameter einen Minimum- und einen Maximum-Wert. Dann lässt man die KI einige Runden fahren. Nach der letzten Runde, ändert man einen Parameter und lässt die KI erneut fahren. Danach ändert man den Wert erneut, je nachdem wie sich die vorige Änderung ausgewirkt hat. Diesen Ablauf macht man immer wieder. Um bei dem ganzen Ablauf Zeit zu sparen, lässt man das Spiel so schnell wie möglich durchlaufen (jedoch immer noch mit festen Zeitschritten). So könnte man z.B. die Spielschleife mehrfach aufrufen, ohne darauf zu warten, dass die Szene fertig gerendert wurde.

4 Ausblick

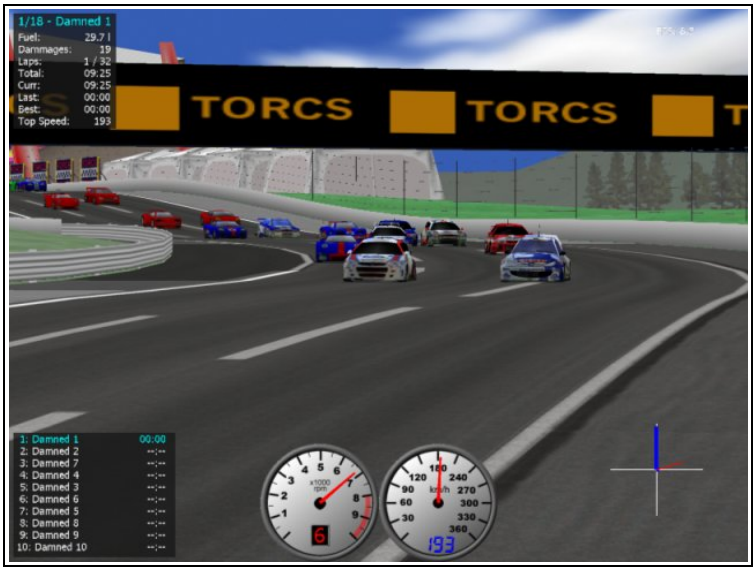
4.1 Optimierungen

Insgesamt lassen sich die Algorithmen in Rahmen der Rennspiel-KI sicherlich noch weiterentwickeln. So bedingen bestimmte Rennspiele (z.B. Rally-Spiele wie „Colin McRae“) auch andere Ansätze, da man dort beispielsweise nicht mit normalen Ideallinien auskommt. In besagtem Spiel wurden Neuronale Netze eingesetzt, wie Jeff Hennan in seinen Interviews beschreibt ([Hannan1] und [Hannan2]). Aber auch der Einsatz anderer „eigenständiger“ Algorithmen könnte interessant sein, wie z.B. Fuzzy-Logic oder Genetische Algorithmen.

4.2 Roboter Rennen

Es gibt inzwischen mehrere Roboter-Rennen. Bei diesen Rennen treten nur KI-Spieler gegeneinander an, welche von Programmierern entwickelt werden. Jeder teilnehmende Programmierer entwickelt also seine eigene KI, seinen eigenen „Robot“. Diese kommunizieren über festgelegte Schnittstellen mit „der Rennstrecke“. Die bekanntesten Roboter-Rennen werden im folgenden vorgestellt.

4.2.1 TORCS (The Open Racing Car Simulator)



Bei TORCS wird in C/C++ entwickelt. Jedes Jahr gibt es eine Meisterschaft.
http://torcs.sourceforge.net

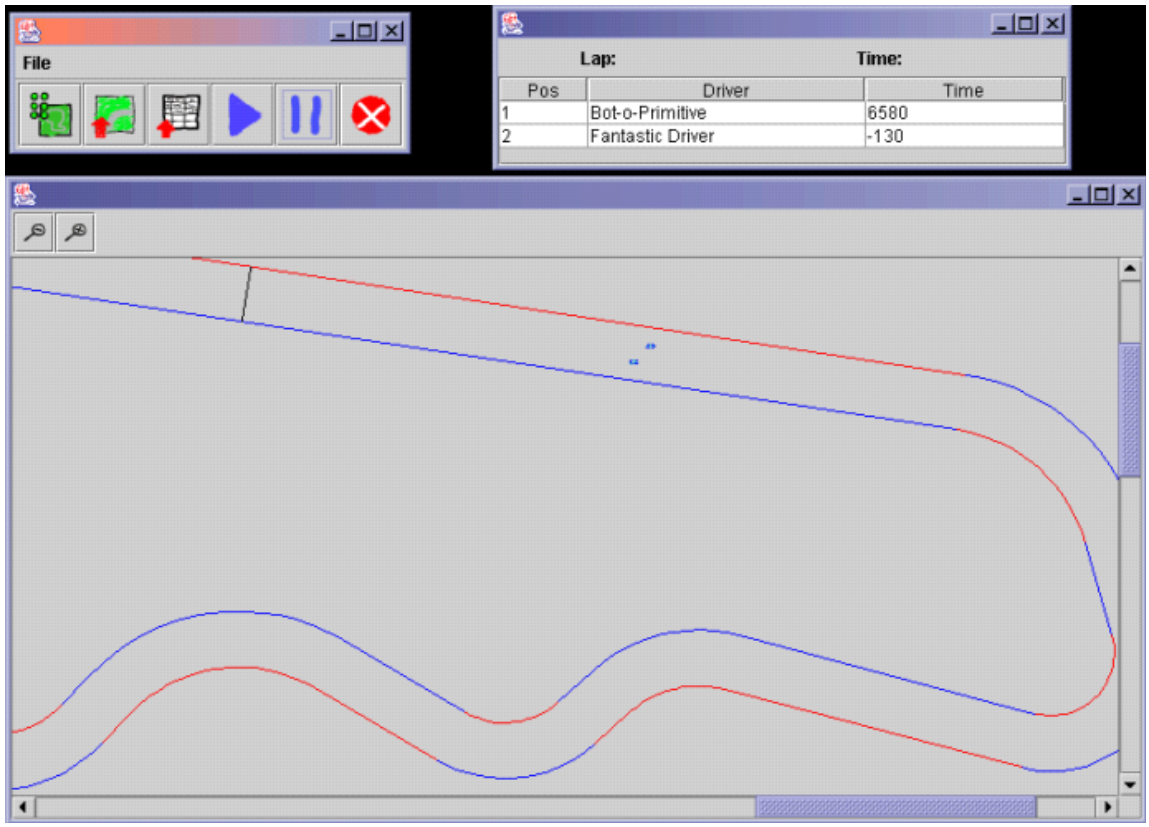
4.2.2 RARS (Robot Auto Racing Simulation)

RARS ist ebenfalls ein Wettbewerb für Programmierer, die beste Renn-KI zu entwickeln. Zusätzlich zum Programmier-Aufwand kann sich jeder per CAD/3D-Software ein Rennauto modellieren.

http://rars.sourceforge.net

4.2.3 JBotRace

JbotRace wird in Java entwickelt. Die grafische Oberfläche ist bei diesem Spiel allerdings nicht so ansprechend, wie bei den vorigen Beispielen.



http://jbotrace.sourceforge.net

Literaturverweise

[WIKI-FL]
Englischer Wikipedia-Eintrag zum Thema „Feedback Loops“
http://en.wikipedia.org/wiki/Feedback_loops

[SemKI]
Seminar „KI in Spielen“ an der Uni Kassel, Ausarbeitung zum Thema „KI für Rennspiele“
http://www.plm.eecs.uni-kassel.de/ki-in-spielen-dateien/Ki_in_Spielen_Racing_Ai_Ausarbeitung.pdf

[AIjunkie]
Artikel zum Thema „State-Driven Game Agent Design“ auf ai-junkie.com
http://www.ai-junkie.com/architecture/state_driven/tut_state1.html

[Berniw]
Website über TORCS Rennen, inklusive Anleitung zum Erstellen eines „Robots“
http://www.berniw.org

[Hannan1]
Interview mit Jeff Hannan über den Einsatz von Neuronalen Netzen bei „Colin McRae“
http://www.generation5.org/content/2001/hannan.asp

[Hannan2]
Interview mit Jeff Hannan über den Einsatz von Neuronalen Netzen bei „Colin McRae“
http://www.ai-junkie.com/misc/hannan/hannan.html

[Red3D]
Steering behaviors for autonomous characters
http://www.red3d.com/cwr/steer/

[TORCS]
The Open Racing Car Simulator
http://torcs.sourceforge.net

[RARS]
Robot Auto Racing Simulation
http://rars.sourceforge.net

[JBot]
JBotRace
http://jbotrace.sourceforge.net

[AIPW]
Steve Rabin – AI Game Programming Wisdom
2002, Charles River Media Inc.