AI-Racing

Nik und Kevin

06.05.2022-

05.07.2022

Inhaltsverzeichnis

[Ziele: 2](#_Toc107906027)

[Bis zum Projektende zu erreichende Ziele: 2](#_Toc107906028)

[Bis zum Projektende zu erreichende optionale Ziele: 2](#_Toc107906029)

[Einleitung: 2](#_Toc107906030)

[Zeitplanung: 2](#_Toc107906031)

[Aufbau / Bedienung: 2](#_Toc107906032)

[KI: 3](#_Toc107906033)

[Streckenaufbau: 3](#_Toc107906034)

[Schnittpunktberechnung: 5](#_Toc107906035)

[Effizienz: 5](#_Toc107906036)

[Sicht: 6](#_Toc107906037)

[Bewegung: 6](#_Toc107906038)

[GUI: 6](#_Toc107906039)

[Hit Box: 7](#_Toc107906040)

[Zusammenfassung: 7](#_Toc107906041)

# Ziele:

## Bis zum Projektende zu erreichende Ziele:

* Die Autos können eine Runde vollenden
* Mehrere Autos können gleichzeitig fahren
* Die besten Generationen anzeigen lassen

## Bis zum Projektende zu erreichende optionale Ziele:

* Die Autos können aufeinander Reagieren
* Man selber kann gegen die KI antreten
* KI kann leichter oder schwieriger gemacht werden

# Einleitung:

Das Thema unseres Software- Projektes ist es, ein von einer KI gesteuertes Auto über eine individuelle Strecke fahren zu lassen. Im Idealfall kann man dieses Auto am Ende über jede mögliche Strecke fahren lassen, ohne dass es gegen die Bande fährt. Kevin war der Hauptverantwortliche für die Erstellung der Strecke und Nik hat den Großteil des Neuronalen Netzwerkes entwickelt, welches unter „KI“ Erklärt wird.

# Zeitplanung:

Die Entwicklung des Neuronalen Netzwerkes, das Trainieren ausgenommen, und der unterliegenden Anwendung hat pro Person etwa 30 Stunden in Anspruch genommen. Diese Zeit war relativ ausgeglichen sowohl in der Schule, als auch zuhause verteilt.

# Aufbau / Bedienung:

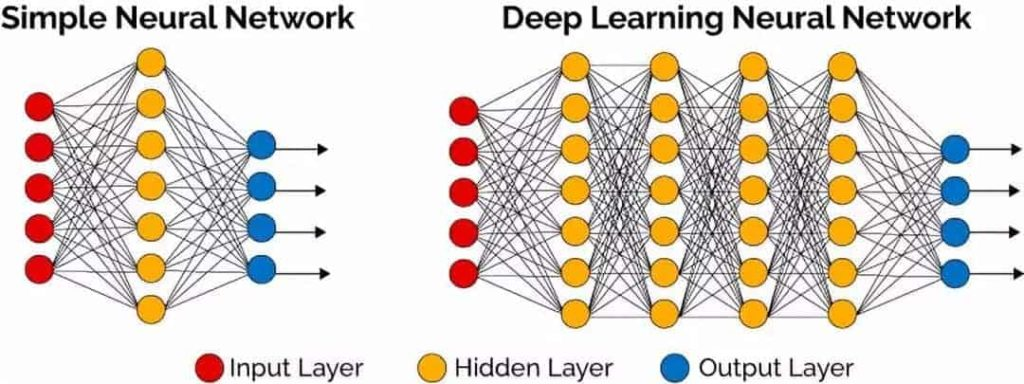
Für die Bedienung gibt es zwei Skripte. Im ersten dieser Skripte, main.py, kann man auf der GUI mit Klicks der Maus einzelne Streckenpunkte zeichnen, aus welchen die Strecke besteht. Diese Strecke kann man Exportieren(E), sowie Importieren(I). Das Exportierte Format der Strecke ist eine CSV-Datei mit den Streckenpunkten. Nachdem man die innere sowie die äußere Bande gezeichnet hat, kann man mit WASD das Auto manuell steuern. Auch kann man den Resetpunkt des Autos mit R verändern.

Das andere Skript beinhaltet die KI. Diese kann entweder von null auf, oder von einem vorherigen Checkpoint aus Trainiert werden. Ebenso kann man sich einen Checkpoint ansehen, ohne diesen Trainieren zu lassen. Man kann die Strecke entweder manuell festlegen, oder zufällig auswählen lassen. Hierbei ist zu beachten, dass die Strecke zuerst aus dem ersten Skript exportiert werden muss, da es kein UI gibt. Dies bedeutet auch, dass man nicht selber ein Auto steuern kann. Um die GUI zu sehen, muss man parallel zu dem Skript Neat\_with\_calculate.py das Skript neat\_GUI.py ausführen. Genaueres hierzu finden Sie unter „GUI“.

# KI:

Die KI arbeitet mit 6 Inputs und 4 Outputs. Eine der Inputs ist die Geschwindigkeit, die fünf anderen Inputs sind die Entfernungen zu den Fahrbahnbegrenzungen, welche in „Sicht“ erklärt werden. Die Outputs für die KI sind booleans zum Fahren, was bei „Bewegung“ gezeigt wird.

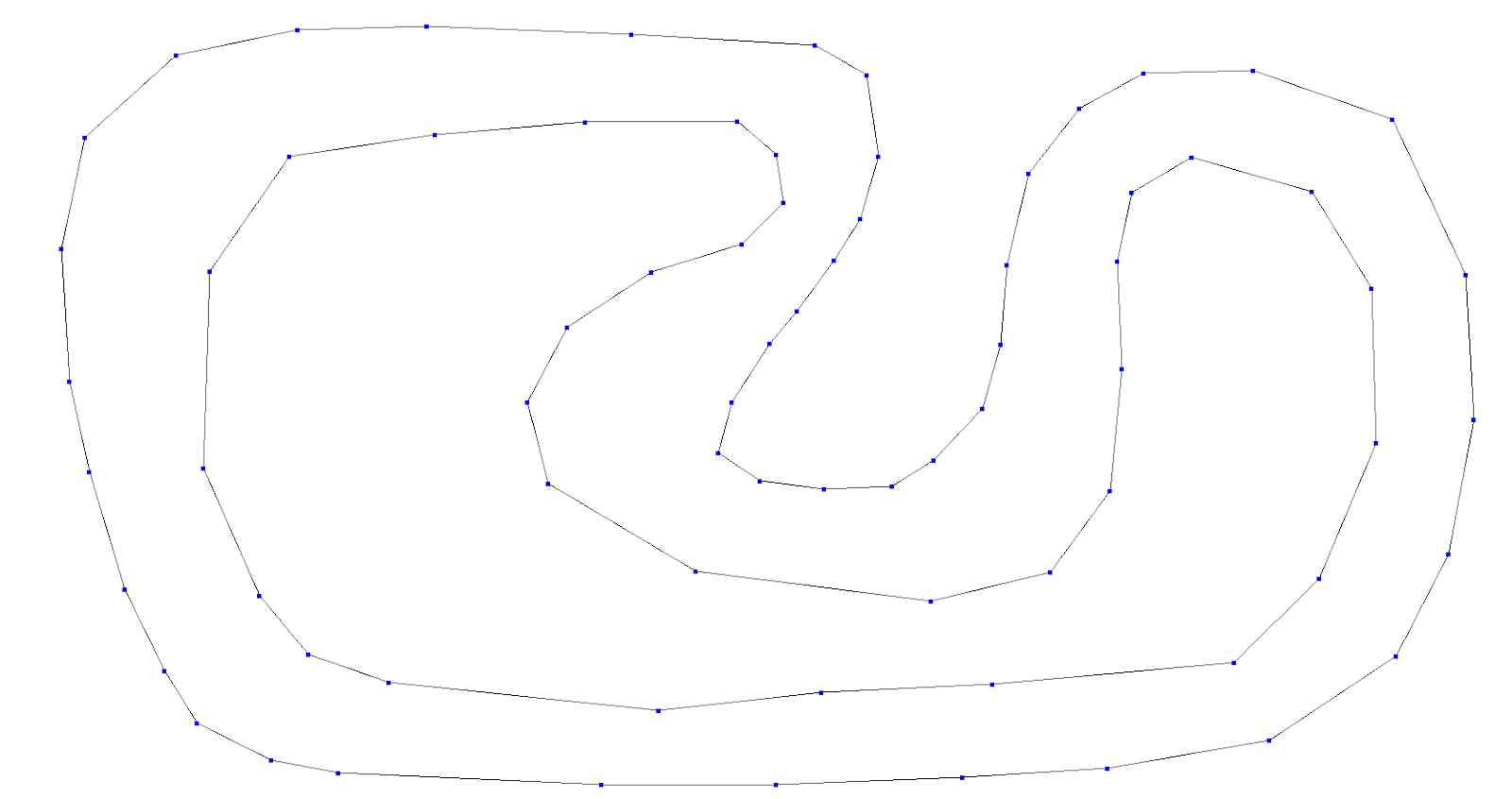
Der Algorithmus, welchen wir für die Steuerung der Autos verwenden, heißt NEAT. Dieser ist ein Neuronales Netzwerk. Neuronale Netzwerke bestehen aus Neuronen, welche sich in verschiedenen Schichten befinden. Bei den meisten dieser Algorithmen muss man von Anfang an die Menge an versteckten Schichten, sowie die Menge an Neuronen pro Schicht vor dem Trainieren Festlegen. Bei NEAT jedoch ist dies nicht der Fall, da NEAT diese versteckten Schichten von selbst zufällig erzeugt. Für genauere Infos über den NEAT Algorithmus gibt es eine genaue Erklärung vom MIT [4]. Für den NEAT Algorithmus benutzen wir die Library NEAT-Python [2].



[3]

# Streckenaufbau:

Wie bereits unter dem Punkt “Aufbau / Bedienung” erklärt, kann man die Strecken per Maus Zeichen, sowie Exportieren und Importieren. In dieser Exportierten CSV Datei werden die X und Y Koordinaten jedes Punktes gespeichert. Hierbei sind die X und Y Koordinaten in den Zeilen gespeichert. Zeile 1 ist X1, 2 ist Y1, 3 ist X2 und 4 ist Y2. Dies geht, da die Strecke aus Punkten aufgebaut ist, zwischen welchen die GUI Linien zeichnet. Die hier gezeichnete Strecke ist noch eine ältere Version, da es nur bei Kurven Punkte gibt. Dies veranschaulicht den Aufbau der Strecke besser, ist jedoch ineffizient. Der Grund hierfür steht in „Effizienz“.



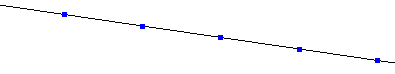
# https://lh5.googleusercontent.com/XusGiUwhObaEY3DJb3ZFflVjG5Meap4uHpv6C3n77h2IqbHgpN13hGgQTvmTE3YG6jjh_tpPkqc4mwIYSv3JgRLXCvUMMNm7fqEbFXIS_HcM9uW4su3fphHXKLveKTqppqSjNtZcUN6RQvaWGwSchnittpunktberechnung:

Die Linien, welche in “Streckenaufbau” erklärt wurden, werden nicht nur dargestellt, sondern auch zum Berechnen verwendet. Hierbei wird bei jedem Tick errechnet, ob die Hit Box des Autos, welche in „Hit Box“ erklärt wird, mit den Linien der Strecke einen Schnittpunkt hat. Für die Sichtstrahlen des Autos wird dies ebenso errechnet, sowie der Abstand vom Mittelpunkt des Autos zur Wand.

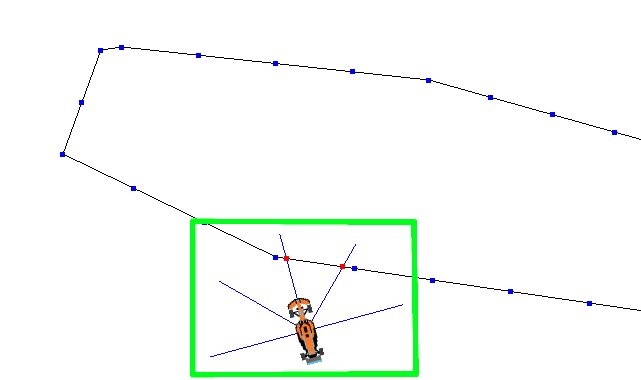
Die Formel zur Berechnung der Schnittpunkte kommt aus einem Stackoverflow-Post [1]. Für die Längenberechnung der Strahlen benutzen wir den Satz des Pythagoras.

# Effizienz:

Da der Code in Python geschrieben ist, welches eine relativ langsame Programmiersprache ist, konnten wir maximal 150 Autos in einer Generation Simulieren, da es ansonsten nicht flüssig lief. Dies lag zum Teil auch daran, dass bei jedem Tick bei jeder Linie des Autos und der Strecke der Schnittpunkt berechnet wurde, siehe „Schnittpunktberechnung“. Da das Auto eine maximale Sichtweite von 100 Pixel hat, und die Strecke über 1000 Pixel breit ist, macht dies keinen Sinn, weshalb wir bei der Strecke Erzeugung lange Linien in 80 Pixel große Abschnitte einteilen.



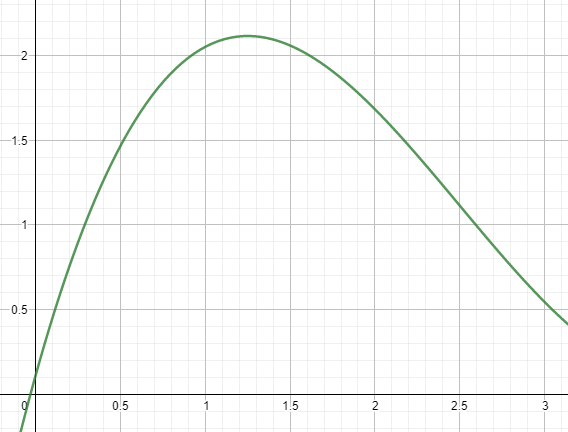
Somit können wir die Schnittpunktberechnung auf Linien im Umkreis von 160 Pixel begrenzen, was Rechenzeit spart. 80 Pixel würden nicht reichen, auch wenn die Abschnittlänge 80 Beträgt, da eine Linie immer aus zwei Punkten besteht, und beide dieser Punkte in der Sichtweite sein müssen. Diese Sichtweite wird symbolisch in diesem Bild dargestellt:

Durch diese Sichtweite können alle Punkte, welche sich nicht in dem grünen Quadrat befinden, übersprungen werden.

# https://lh3.googleusercontent.com/YV_6dXpmScPMceb_PoR2m4zS5QdCnm1yKCH2vlVdMMZt8nLTLbiCtinuT8qX_vGNQxdrEMiT0yjxIPSiYqpXB8QDVweZXZemFEywFtH0dhS2UjwW79Sx8EK5flnkxVoxYrx5QEEKhOfpe6cIRQSicht:

Die Sicht des Autos ist sehr Begrenzt. Hierbei gibt es 5 Linien, welche gleichmäßig auf einem Blickfeld von 180 Grad verteilt werden. Wir haben mit verschiedenen Kombinationen von Menge und Blickfeld gearbeitet, jedoch funktionierte dies am Besten. Die Länge der Linien beträgt 100 Pixel. Die KI bekommt den Abstand vom Mittelpunkt des Autos zum nächsten Hindernis als eine Fleißkommazahl zwischen theoretisch 0 und 100. Die Genaueren Informationen zur Berechnung dieser Zahl stehen in “Schnittpunktberechnung”.

# Bewegung:

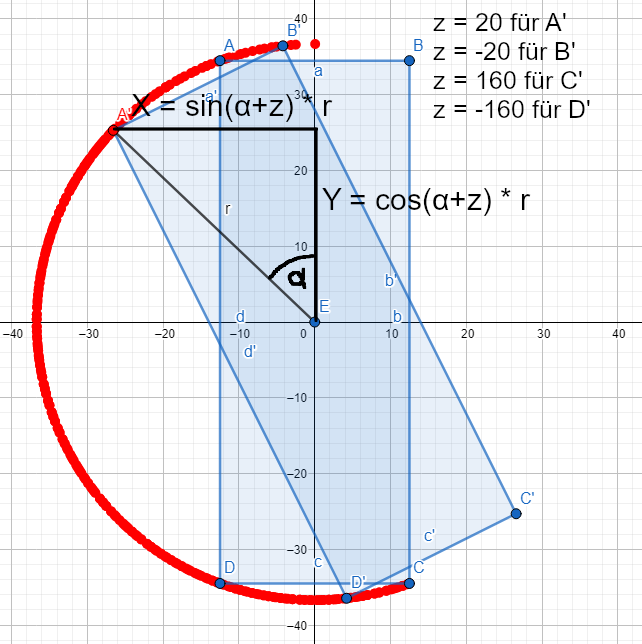
****Die Autos können wie echte Autos Beschleunigen, Bremsen und Lenken. Die Geschwindigkeit wird bei jedem Tick berechnet, wobei die Beschleunigung Linear ist. Die maximale Geschwindigkeit beträgt 3. Für das Bremsen gibt es zwei Werte. Der Erste ist die normal Motorbremse und Reibung, welche in jedem Tick auf das Auto wirkt. Die andere Bremskraft ist die aktive Bremse des Autos, wobei hierbei das Auto deutlich schneller bremst. Die Lenkung ist nicht immer die gleiche. Hierbei gibt es eine Formel, siehe Bild, welche die Lenkgeschwindigkeit im Verhältnis zur Geschwindigkeit berechnet. Die maximale Lenkgeschwindigkeit ist bei einer Geschwindigkeit von etwa 1,25 erreicht.

# GUI:

Für die GUI benutzen wir die Library Arcade [7]. Diese wurde uns von Christian und Alex empfohlen. Ein Problem das wir hatten war, dass viele NEAT Tutorials für die Library Pygame gemacht wurden, welche ein grundlegend anderes System ist. Deshalb mussten wir uns selbst mit der NEAT Implementation helfen. Da NEAT und Arcade nicht zusammen in ein Python-Skript passen, mussten wir die GUI als eigene Python Datei anlegen, um Multithreading anwenden zu können.

Hierfür verwenden wir nun zwei getrennte Python Instanzen. Dies bringt jedoch das Problem, dass diese Scripts nicht direkt miteinander kommunizieren können. Um dieses Problem zu umgehen, schreiben wir die Playerinformationen (Position und Rotation) in eine CSV Datei, und lassen diese von der GUI auslesen. Dies hat zur folge, dass die Anzeige gelegentlich flackert.

# Hit Box:

https://lh4.googleusercontent.com/Ipkhb8VsPKJjIZaTQ-GtX6f-kiU3I8gdEU6jpQPGBgKN0R_8fWsPi4htM-bttzW8w-abh62cXp6IlAndkUKyt0HUiUkQB_Vsk7fMkLfRNyWQjwj-AEAfw6UiXLaaSCXVvrxz0Ng2-ycjQM2O5wDie Ecken des Autos werden mit Hilfe von Sinus und Cosinus, wie im linken Bild zu sehen, berechnet. Die Hit Box Linien werden dann durch diese Punkte berechnet, wie im rechten Bild mit dem Auto zu erkennen. Wenn durch die “Schnittpunktberechnung” ein Schnittpunkt berechnet wird, wird das Auto zurückgesetzt. Da die Trigonometrischen Funktionen in Python im Bogenmaß rechnen, wurde unsere Hit Box am Anfang falsch berechnet. Dies lag daran, dass wir im Gradmaß rechneten.

# Zusammenfassung:

Unser erster Versuch war mit der Programmiersprache C#. Bei dieser müsste man jedoch die KI von Grund auf selber schreiben, wofür uns die Zeit nicht reichen würde, weshalb wir gezwungen waren, nach Python zu wechseln. Hierbei mussten wir Python nahezu von Grund auf lernen. Der erste Schritt war das Spiel aufzubauen. Dies diente dazu, die Grundlagen für die KI zu legen, sowie die Grundlagen von Python zu lernen. Nachdem dies funktionierte, entwickelten wir die KI. Dies brachte ein Problem, da unsere GUI anders als NEAT arbeitet, und dies auf eine Weise, dass diese beiden nicht zusammenpassen, weshalb wir uns etwas ausdenken mussten. Jedoch haben wir nach etwas überlegen eine Lösung gefunden, welche gut genug funktioniert. Anfangs hatten wir Probleme damit, die KI zum Lernen zu bringen, haben aber herausgefunden, dass wir einen Fehler bei der Inputberechnung hatten, was zufolge hatte, dass der Input dauerhaft den gleichen Wert hatte. Nachdem auch dieses Problem behoben wurde, hat die KI gelernt. Nach einigem herumprobieren sind wir auf gute Werte für die Menge und Anordnung der Inputs gekommen.

**Quellen:**

[1] <https://stackoverflow.com/questions/3838329/how-can-i-check-if-two-segments-intersect>[2] <https://neat-python.readthedocs.io/en/latest/>[3] <https://datasolut.com/was-ist-deep-learning/>[4] <https://nn.cs.utexas.edu/downloads/papers/stanley.ec02.pdf>  
[5] bit.ly/3bO3UdO  
[6] <https://www.youtube.com/watch?v=2o-jMhXmmxA&t=143s&ab_channel=CheesyAI>  
[7] https://api.arcade.academy/en/latest/