DZT - Projekt: KI lernt, wie man CrossyRoad spielt

1. Einleitung

Dieses Projekt hat zum Ziel, eine KI anzulernen, die eine vereinfachte Version des Spiels CrossyRoad spielen soll. Bei diesem Spiel bewegt man einen Spieler mit Pfeil-Tasten und versucht möglichst weit im Level voranzuschreiten, ohne mit Hindernissen wie etwa Autos oder dem Level-Rand zu kollidieren. Das Projekt wurde in der Programmiersprache Python umgesetzt. Für das Spiel wurde die Bibliothek Pygame genutzt. Für die Implementierung der KI wurde der NEAT-Algorithmus aus der Neat-Bibliothek genutzt

2. Hauptteil

2.1 CrossyRoad – Spielmechanik

Bei dieser reduzierten Version von CrossyRoad, wird der Spieler als rotes Quadrat abgebildet. Mithilfe der Pfeiltasten (oben, unten, rechts und links) kann der Spieler sich in die entsprechenden Richtungen bewegen. Das Spielfeld ist dabei in ein zweidimensionales Feld-Gitter eingeteilt. Der Spieler kann sich nur von Feld zu Feld bewegen. Sobald er eine Pfeiltaste drückt, beginnt die Bewegung des Spielers in die entsprechende Richtung. Während der Spieler in Bewegung ist, kann er keine weiteren Tasten mehr drücken. Erst wenn der Spieler das nächste Feld erreicht hat, wird seine Bewegung gestoppt. Nun kann der Spieler erneut eine Pfeiltaste drücken. Berührt der Spieler den Rand des Levels / des Bildschirms, hat er verloren und kann nicht mehr weiterspielen. Der Spieler wird zu Anfang des Spiels auf die obere Seite des Spielfeldes platziert. Um im Spiel voranzuschreiten, muss er weiter nach unten gelangen. Dabei bewegt sich das Level stetig nach oben. Der Spieler muss sich also beeilen, um nicht vom oberen Spielfeldrand überholt zu werden. Wenn ein Spieler deutlich schneller als das Level ist, wird das Level unter bestimmten Umständen beschleunigt. Sobald der Spieler eine festgelegte Distanz zum unteren Bildschirmrand unterschreitet, wird die Levelgeschwindigkeit auf die Geschwindigkeit des Spielers erhöht. Somit wird es besonders guten Spielern ermöglicht, deutlich schneller durch das Level voranzuschreiten, ohne durch die niedrigere Geschwindigkeit des Levels gebremst zu werden. Die Punktzahl bestimmt sich aus der Anzahl der Felder, die der Spieler in vertikaler Richtung überquert hat. Die Überquerung des Levels wird durch fahrende Autos erschwert, mit denen der Spieler nicht in Berührung kommen darf. Die Autos werden hier durch weiße Rechtecke abgebildet. Autos können dabei nur auf Straßen von links nach rechts (oder umgekehrt) fahren. Straßen sind schwarz markierte Levelbereiche, die sich vom linken bis zum rechten Bildschirmrand erstrecken. Levelbereiche, die keine Straßen

sind, gelten als Gras-Felder und sind grün markiert. Die Anzahl der aufeinanderfolgenden Gras- und Straßen-Bereiche wird zufällig generiert. D.h. ein Level kann bspw. mit 1-3 Grass-Bereichen beginnen, worauf 1-3 Straßen-Bereiche folgen, worauf wiederum 1-3 Grass-Bereiche folgen usw.. Bei der Generierung des Levels wird für jede Straße ebenfalls ein Schwierigkeitsgrad zufällig festgelegt. Abhängig von der Schwierigkeit einer Straße, ändert sich die Geschwindigkeit der Autos, sowie der Abstand zwischen den Autos. Der Code für das ursprüngliche Spiel ist unter folgender Datei zu finden: CrossyRoad_22_03_(unkommentiert).py

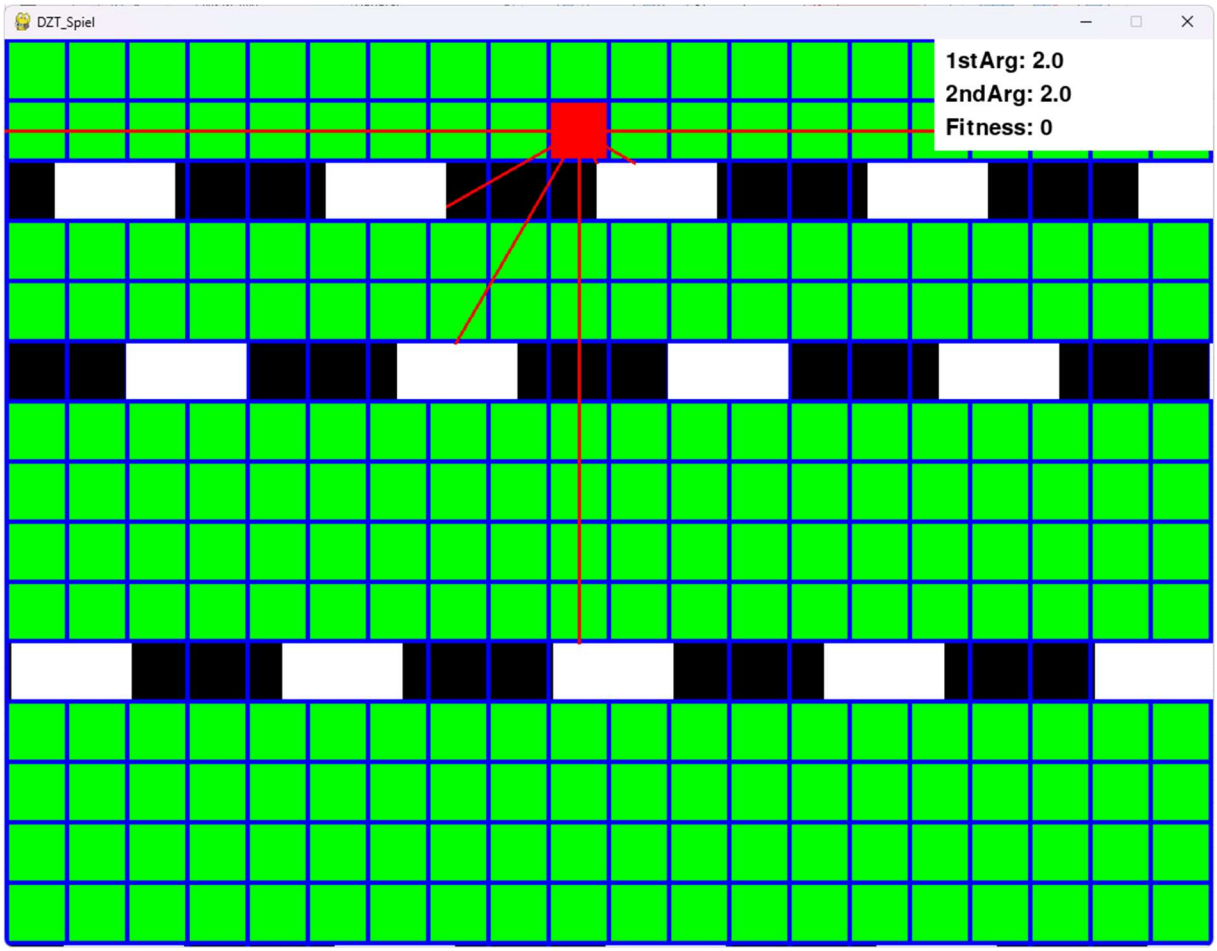

Abbildung 1 - Screenshot CrossyRoad

2.2Implementierung der KI

2.2.1 Neat-Algorithmus

Für die Implementierung der KI wurde der sog. Neat-Algorithmus (NeuroEvolution of Augmenting Topologies) genutzt. Dabei handelt es sich um einen genetischen Algorithmus, bei dem künstliche neuronale Netze generiert werden. Anders als bei anderen Algorithmen, werden bei diesem nicht nur die Gewichte der neuronalen Netze angepasst, sondern auch die Netz-Topologien. Das bedeutet, dass Neuronen und Verbindungen hinzugefügt oder entfernt werden können. Zu Anfang wird dabei eine Population von zufälligen neuronalen Netzwerken erzeugt. Der sog. Fitness-Parameter

wird dabei genutzt, um zu bewerten, wie erfolgreich eine Population ist. Im Kontext dieses Projektes kann es sich dabei bspw. um die erreichte Punktzahl im Spiel handeln. Für die nächste Generation werden die erfolgreichsten Netzwerke nach einem gewissen Schema kombiniert und / oder mutiert. Ziel ist es, eine Generation von Spielern „heranzuzüchten“, die die Aufgabe bestmöglich löst.

2.2.2 Inputs, Outputs und Fitness

Damit eine KI das Spiel spielen und erlernen kann, müssen die Eingabeparameter (Inputs) für das neuronale Netzwerk passend gewählt werden. Auf Grundlage dieser Inputs, muss die KI letztlich eine Entscheidung (Output) finden. Bei den Inputs kann es sich bspw. um die Position des Spielers, den Abstand zum nächsten Auto, oder die Entfernung zum Spielrand handeln. Die Outputs sind Zahlenwerte, die dann wiederum als Entscheidung interpretiert werden. In diesem Projekt hat der Spieler fünf Outputs. Diese Outputs stehen für: Bewegung nach links, Bewegung nach rechts, Bewegung nach oben, Bewegung nach unten und Stehenbleiben. Es wird die Entscheidungsmöglichkeit ausgewählt, bei der der Output am höchsten ist. Eine alternative Methode bei der nur vier Outputs genutzt wurden („Stehenbleiben wurde entfernt“), wurde getestet und verworfen. Dabei wurde eine Entscheidung nur getroffen, wenn der Output einen Wert von 0,5 überschreitet. Der Fitness-Parameter wird zur Bewertung der neuronalen Netzwerke genutzt. In der einfachsten Ausführung „belohnt“ man einen Spieler jedes Mal wenn er ein neues Feld erreicht, bzw. wenn er eine höhere Punktzahl erzielt, indem man die Fitness um einen gewissen Wert erhöht. Weitere Techniken können genutzt werden, um erwünschtes Verhalten zu belohnen, und unerwünschtes Verhalten zu bestrafen. So könnte man bspw. für Spieler, die eine gewisse Punktzahl besonders schnell erreichen, eine höhere Fitness geben als für langsamere. Da viele Spieler häufig mit dem linken und rechten Spielrand in Kontakt gekommen sind, wurden in diesem Projekt bspw. Spieler belohnt, die näher in der Mitte des Spielfeldes (bezogen auf die x-Achse) blieben.

2.2.3 Verlauf des Projektes und Ergebnisse

Zu Anfang des Projektes wurde die Fitness eines Spielers durch die Punktzahl, die er erreicht hat, definiert. Die Input-Parameter waren dabei: die x- und y-Position des Spielers, die x- und y-Position der nächsten zehn Autos. Dies hat jedoch nicht ausgereicht um eine KI anzulernen, die das Spiel einigermaßen erfolgreich spielt. Die Überlegung, dass nicht die Position, sondern der Abstand zu einem Auto relevant(er) ist, resultierte in der Entscheidung, die x- und y-Position der nächsten zehn Autos wegzulassen, und stattdessen die x- und y-Differenzen von der Spieler-Position zu den nächsten zehn Autos als Input zu wählen. Die Vermutung liegt nahe, dass es sonst sehr viele Generationen gebraucht hätte, bis das Netzwerk den Zusammenhang zwischen der Differenz der Spieler- und Auto-Positionen als relevanten Faktor „versteht“. Auch dies führte nicht zu akzeptablen Ergebnissen.

Zu Testzwecken wurde das Spiel daher stark vereinfacht. Die maximale Anzahl von aufeinanderfolgenden Straßen wurde auf eins reduziert. Mehrspurige Straßen waren

somit also nicht mehr möglich. Auch die Geschwindigkeit der Autos, sowie der Abstand der Autos zueinander, wurde auf einen festen Wert gesetzt.

Als weiterer Input wurden auch die Bewegungsrichtungen der nächsten zehn Autos mitaufgenommen (Boolean-wert: True = rechts, False = links). Die Größe der Anfangspopulation wurde zwischen 100 und 1000 variiert. Auch wurde die Anzahl der Inputs variiert. D. h. statt der nächsten zehn Autos, wurden nur fünf Autos oder auch nur ein Auto in Betracht gezogen. Auch wurde der Zeitfaktor für den Belohnungs-Mechanismus genutzt. Mit jedem Tick wurde die Fitness um die aktuelle Punktzahl erhöht. D. h. Spieler, die für eine lange Zeit eine höhere Punktzahl haben, werden gegenüber Spielern bevorzugt, die nur kurz eine hohe Punktzahl erreicht haben und dann direkt verloren haben. Damit werden Spieler die sich direkt „in den Tod stürzen“ benachteiligt. Dies hatte zur Folge, dass viele Spieler ein „Zitter“-Verhalten aufwiesen. Anstatt eine Straße in einem Durchlauf zu überqueren, sind die Spieler auf die Straße gegangen, nur um dann sofort wieder zurückzugehen. Dieser Vorgang hat sich fortlaufend wiederholt, bis die Spieler mit einem Auto kollidiert sind. Augenscheinlich war es die Taktik dieser Spieler, möglichst kurz auf der Straße zu bleiben, um das Risiko einer Kollision mit einem Auto zu minimieren. Um jedoch mehr Fitness-Punkte zu erhalten, haben diese dennoch teilweise den Schritt auf die Straße und den Schritt zurück gewagt. Zielführend ist dieses Verhalten offensichtlich nicht.

Ein mögliches Problem bei diesen Versuchen könnte sein, dass die Level für jede Generation zufällig neu generiert wurden. Wenn also in der ersten Generation ein leichtes Level generiert wurde, haben viele Spieler eine hohe Fitness erreicht. Wenn dann in der nächsten Generation ein schweres Level generiert wurde, haben viele Spieler entsprechend eine geringe Fitness erreicht. Dieser Rückgang in der Fitness könnte vom NEAT-Algorithmus als Stagnation bewertet werden. In diesem Fall wird teilweise auf Netzwerke früherer Generationen zurückgegriffen, und aktuelle Netzwerke werden fälschlicherweise verworfen. Um dies zu verhindern, und eine bessere Vergleichbarkeit zwischen der Fitness verschiedener Generationen gewährleisten zu können, wurde die Levelgenerierung so geändert, dass jede Generation dasselbe Level spielt.

Weiterhin gab es das Problem, dass es die augenscheinliche Taktik vieler Spieler war, sich einfach kontinuierlich ohne Rücksicht auf Hindernisse nach unten zu bewegen. Dieses Verhalten lässt sich womöglich damit erklären, dass es bei einigen Leveln zufällig möglich war, ein paar Punkte zu erzielen in dem man sofort mehrere Schritte nach unten macht. Eventuell hat sich die Kompetenz, Autos ausweichen zu können dadurch nicht durchgesetzt. Um dem entgegenzuwirken, werden von nun an eine Reihe von verschiedenen Levels generiert. Darunter sind mit höherer Wahrscheinlichkeit auch Levels enthalten, bei denen das besagte Verhalten nicht vorteilhaft ist. Die Spieler müssen also auch zwangsläufig Ausweich-Taktiken entwickeln, um zu den besseren Spielern zu gehören.

Die Ergebnisse, die somit erzielt wurden waren besser als die anfänglichen Ergebnisse. Es konnte jedoch kein Netzwerk generiert werden was ansatzweise einen menschlichen Spieler besiegen könnte. Selbst nach teilweise tausenden Generationen konnte keiner der bisherigen Netzwerke eine Punktzahl von 100 erreichen. Einige Netzwerke haben es manchmal geschafft den Autos korrekt auszuweichen, und haben somit Punktzahlen im Raum von 40 bis 50 erreicht. Danach haben dieselben Spieler jedoch teilweise grundlegend fehlerhafte Entscheidungen getroffen, bspw. indem sie direkt in den linken oder rechten Bildschirmrand gestürzt sind.

Um es den Spielern weiter zu vereinfachen, wurde eine neue Art von Input hinzugefügt. Dabei wurden vor jeder Entscheidung eines Spielers jeweils vier weitere künstliche Spieler erzeugt. Jeder dieser künstlichen Spieler wurde entweder ein Feld vor, hinter, rechts oder links vom eigentlichen Spieler platziert. Dann wurde überprüft, ob diese künstlichen Spieler mit Autos oder dem Bildschirmrand kollidieren. Diese Information wurde als weiterer Input für das neuronale Netzwerk hinzugefügt. Diese vier Boolean-Inputs sollten also den Spieler darauf hinweisen, ob eine Entscheidung offensichtlich „tödlich“ ist oder nicht. Tatsächlich hatte dies kaum einen positiven Effekt. Weiterhin haben sich die besten Spieler teilweise in den linken und rechten Bildschirmrand gestürzt.

In einem weiteren Versuch wurden nach demselben Schema künstliche Spieler erzeugt. Wenn hierbei bspw. der künstliche Spieler rechts vom eigentlichen Spieler mit einem Hindernis kollidierte, wurde dem Spieler die Möglichkeit genommen nach rechts zu gehen. Ab diesem Punkt ist jedoch nicht mehr die Rede von einer künstlichen Intelligenz, welche das Spiel erlernt, sondern vielmehr von einem Bot, welcher bereits alle Entscheidungen im Voraus berechnet, und die Handlungsmöglichkeiten ausschließt, welche offensichtlich falsch sind. Tatsächlich konnten so deutlich höhere Punktzahlen als bisher erzielt werden (über 100). Dennoch kam es dazu, dass Spieler verloren haben. Dies konnte beispielsweise geschehen, weil Spieler auf der Straße stehengeblieben sind, bis ein Auto sie berührt hat. Auch aufgrund der Bewegung der Autos, die bei den künstlichen Spielern nicht mitberechnet wurde, konnte es weiterhin zu Kollisionen kommen.

Da selbst bei dieser scheinbar enormen Hilfestellung keine zufriedenstellenden Ergebnisse erzielt werden konnten, wurde eine andere Möglichkeit zur Lösung der Problemstellung ohne die Nutzung von KI in Erwägung gezogen. In Kapitel 2.5 ist hierfür die Entwicklung eines Bots welcher CrossyRoad spielt aufgeführt.

2.3 Vergleich mit anderen Anwendungsfällen

Die Inspiration für dieses Projekt lag unter anderem in dem von techwithtim programmierten FlappyBird-Spiel, bei dem ebenfalls der NEAT-Algorithmus genutzt wurde, um das Spiel zu erlernen (Quelle: https://github.com/techwithtim/NEAT-Flappy-Bird). Es stellt sich die Frage, weshalb die Nutzung des NEAT-Algorithmus bei diesem Spiel scheinbar besser funktioniert als bei CrossyRoad. Dies könnte folgende Gründe

haben. Zum einen gibt es bei FlappyBird weniger Entscheidungsmöglichkeiten als bei CrossyRoad. Bei letzterem hat der Spieler fünf Möglichkeiten: Bewegung nach links, Bewegung nach rechts, Bewegung nach oben, Bewegung nach unten und Stehenbleiben. Bei FlappyBird sind es nur zwei: Springen und nicht Springen. Zudem ist die Anzahl der Hindernisse, und damit auch die Anzahl der Inputs bei FlappyBird geringer. Hier reichen tatsächlich drei Inputs aus: Die y-Position des Spielers, die y-Entfernung zum oberen und zum unteren Rohr. Bei CrossyRoad hingegen, gibt es auf einer Straße gleich mehrere Autos, die sich zudem in verschiedene Richtungen bewegen können. Dabei muss nicht nur die y-Achse (wie bei FlappyBird), sondern auch die x-Achse zwangsläufig miteinbezogen werden. Die Anzahl der Inputs und Outputs ist also deutlich höher, weshalb das Problem komplexer ist.

Eine Recherche nach Projekten, wo das Spiel CrossyRoad durch eine KI gespielt wird, hatte folgendes Ergebnis. Es gibt Projekte, die den Anspruch erheben dieses Projekt gelöst zu haben, jedoch wurden dafür andere Algorithmen als der NEAT-Algorithmus genutzt. Bei einem dieser Projekte wurde vermeintlich eine durchschnittliche Punktzahl von 45.09 und eine maximale Punktzahl von 370 erzielt. Inwiefern hier die Spielmechaniken, mit denen bei diesem Projekt übereinstimmen, wurde nicht untersucht. Bei einer Suche nach Projekten, die den NEAT-Algorithmen nutzen, wurden nur Spiele gefunden, die eine grundlegend andere Bewegungsmechanik aufweisen. Eine alternative Möglichkeit findet sich in einem französischsprachigen Video (Quelle: https://www.youtube.com/watch?v=71UbDN4csas&ab_channel=CodeBH), bei dem das Spiel CrossyRoad exakt nachprogrammiert wurde, wobei ein Bot einen Weg-Finde-Algorithmus sucht, um die bestmögliche Entscheidung zu finden. Dieser konnte mehr als 1000 Punkte erzielen.

Bei der Recherche nach Problemen mit ähnlicher Komplexität, wurde ein Spiel, bei dem Autos eine Rennstrecke fahren müssen vorgestellt (Quelle: https://www.youtube.com/watch?v=B0ptl-NChJQ&ab_channel=ActivationFunction).
Hierbei wurden Sensoren die relativ zum Auto einen festen Winkel haben und den Abstand zum nächsten Hindernis messen als Inputs genutzt (siehe Abbildung 2). Dies wurde als Inspiration genommen einen letzten Versuch zur Umsetzung einer KI für CrossyRoad zu starten (siehe Kapitel 2.4)

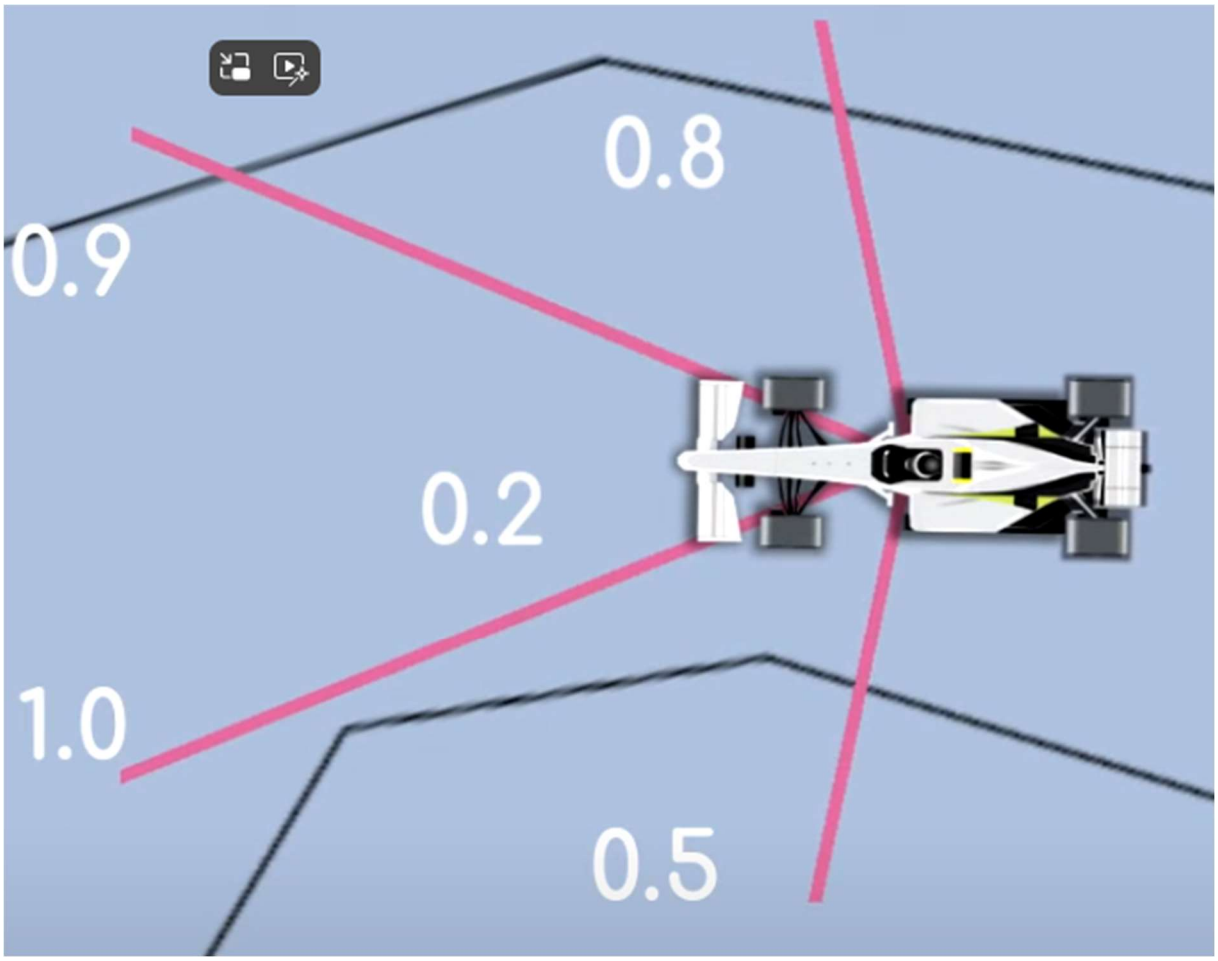

Abbildung 2 - Sensoren als Inputs für Rennwagen-Spiel

2.4 Sensoren als Inputs

In diesem Kapitel wird der letzte Versuch zur Umsetzung einer KI welche CrossyRoad spielt vorgestellt. Hierbei werden Sensoren als Inputs für das neuronale Netzwerk genommen. Dabei wird ausgehend von der Spieler-Mitte eine Linie unter einem festen Winkel gezeichnet. Die Linie endet dabei bei dem ersten Hindernis, auf das sie trifft. Die Länge dieser Linie gilt dann als Input. Dies wird für verschiedene Winkel gemacht. Die gewählten Winkel sind dabei von 0° bis 180°, in Abständen von jeweils 30° (siehe Abbildung 3). Für jede Generation wurden dabei zehn verschiedene Level generiert, die jeweils 100 Felder lang sind. Zudem wurde der Belohnungsmechanismus so geändert, dass Spieler nur belohnt werden, wenn sie gewisse Level-Thresholds überqueren. Bei Versuch 1 hatten diese Level-Thresholds Abstände von jeweils fünf Feldern. Bei Versuch 2 betrug der Abstand drei Felder. D.h. Für das Überqueren der Felder 3, 6, 9 …3*n wird der Spieler belohnt. Mit jedem Level-Threshold der überquert wurde, fällt diese Belohnung höher aus. Beide Versuche wurden für etwa 600 Generationen simuliert. Der Python-Code dafür ist unter (CrossyRoad_RaySensors_29_04.py) zu finden.

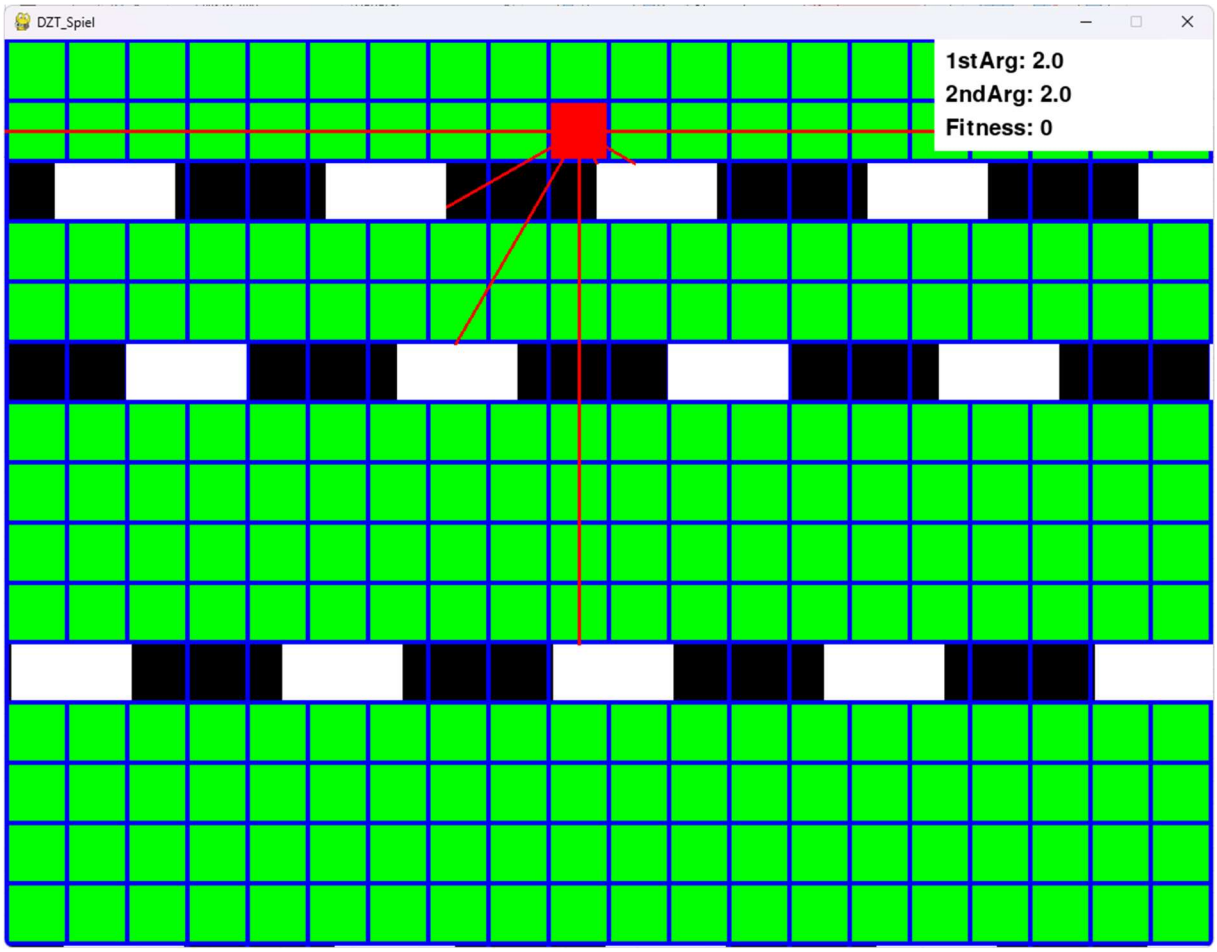

Abbildung 3 - Sensoren als Input

Auch bei diesem Versuch konnten keine zufriedenstellenden Ergebnisse erzielt werden. Zur graphischen Veranschaulichung wurden für jede Runde einer Generation, die maximal erreichte Punktzahl notiert, unabhängig davon, welche Spieler diese Maxima erzielten. Für jede Generation wurden dann die Maximalpunktzahlen gemittelt und in ein Diagramm eingezeichnet (siehe Abbildung 4 und Abbildung 5). In Versuch 1 steigt die Punktzahl ungefähr bis zur Generation 320. Danach sinkt sie etwas und steigt nicht mehr. Die maximale Punktzahl liegt zwischen 35 und 37. Ausgehend vom Diagrammverlauf ist eine wesentliche Verbesserung des Spielerverhaltens nicht zu erwarten. Gleiches gilt für Versuch 2. Hier stagniert die Punktzahl bereits nach 100 Generationen. Dabei wird eine Punktzahl von 30 nicht überschritten. Damit hat Versuch 1 eine besser KI hervorgebracht als Versuch 2.

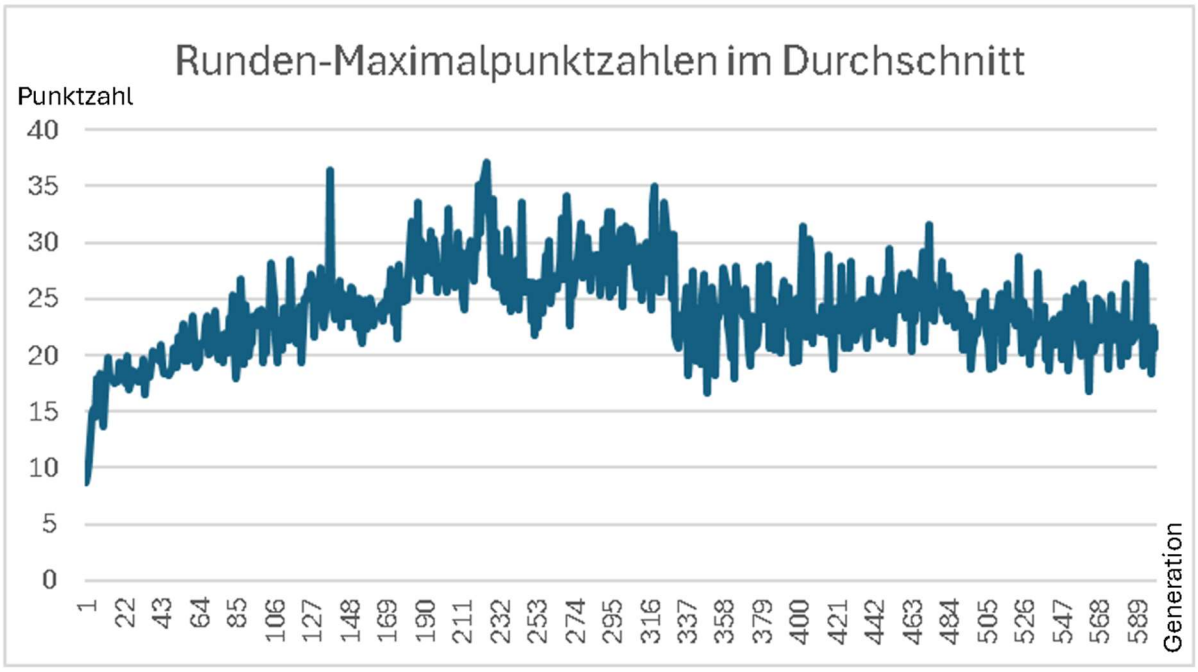

Abbildung 4 - Runden-Maximalpunktzahlen im Durchschnitt – Versuch 1

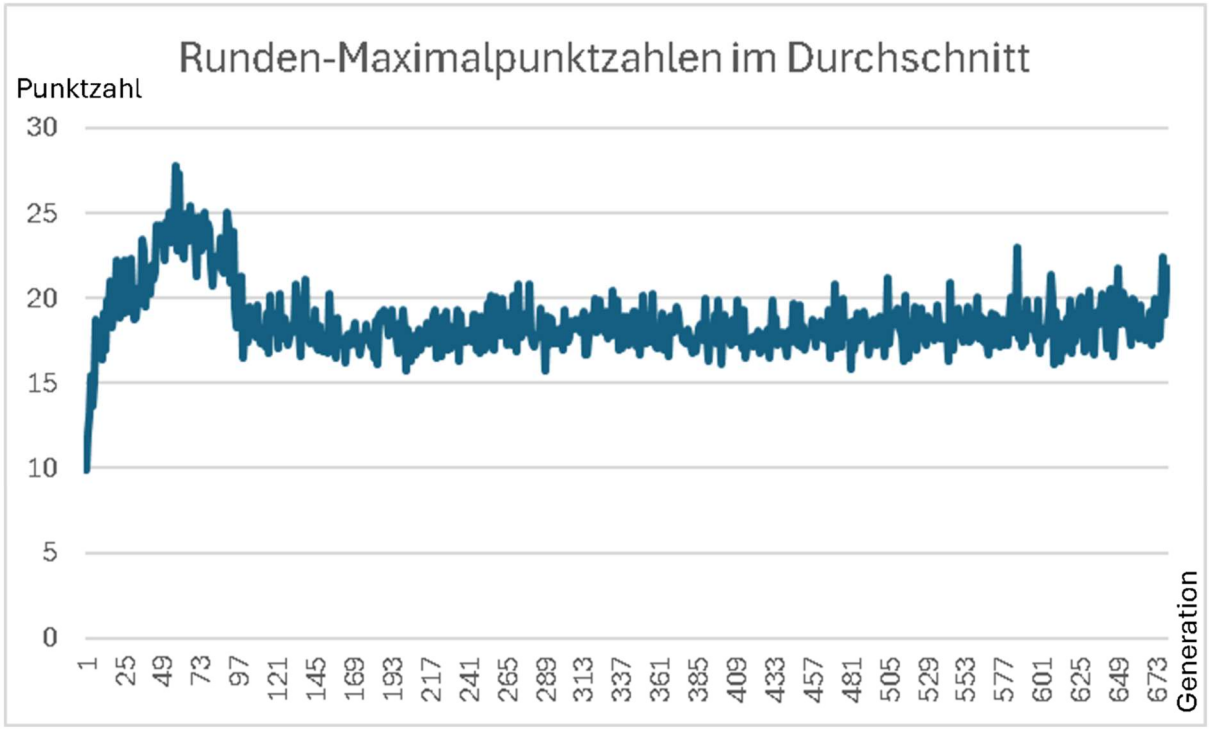

Abbildung 5 - Runden-Maximalpunktzahlen im Durchschnitt – Versuch 2

2.5 Entwicklung eines Bots

Da die KI nicht in der Lage war, das Spiel CrossyRoad zu spielen, soll nun ein Bot entwickelt werden, welcher theoretisch in der Lage ist für immer zu spielen. Dafür werden für jede Entscheidungsfindung drei Kopien des Spielers erstellt. Für jede Kopie werden nun die folgenden Entscheidungen simuliert: Bewegung nach unten, Bewegung nach rechts, Bewegung nach links. Die Bewegung nach oben wird in diesem Kontext ausgelassen, da es derzeit keine mehrspurigen Straßen gibt, und eine Bewegung nach

oben deshalb unter keinen Umständen sinnvoll ist. Anders als bei der zuvor beschriebenen einfachen Simulierung, wird hier tatsächlich jeder Frame simuliert. Es werden also dabei auch die Bildschirmbewegung und die Bewegung der Autos simuliert, um mit 100%-tiger Garantie bestimmen zu können, ob eine Entscheidung „tödlich“ endet oder nicht.

Standartmäßig wird die Bewegung nach unten bevorzugt, da der Spieler mehr Punkte bekommt, wenn er weiter unten ist. Diese Entscheidung wird nur dann vermieden, wenn sie „tödlich“ ist. In diesem Fall bewegt der Spieler sich entweder nach links oder rechts. Sollten beide Möglichkeiten (sowohl links als auch rechts) nicht „tödlich“ sein, wird folgendermaßen entschieden. Befindet sich der Spieler in der linken Bildschirmhälfte, bewegt der Spieler sich nach rechts. Umgekehrt bewegt er sich nach links, wenn er in der rechten Bildschirmhälfte ist. Damit soll garantiert werden, dass der Spieler sich möglichst mittig (bezogen auf die x-Achse) positioniert, da die Bildschirmränder oft weniger Bewegungsspielraum bieten.

Häufig hat der Spieler dann jedoch ein oszillierendes Verhalten gezeigt. Wenn ein Auto im Weg war, ist der Spieler kontinuierlich von links nach rechts, und wieder von rechts nach links gegangen, bis das Auto vorbeigefahren ist. In einigen Fällen hat dies zum Tod des Spielers gefühlt, da ihn der obere Bildschirmrand eingeholt hat. Um dies zu vermeiden, wurde eine weitere Regel implementiert: Wenn die letzte Spieler-Entscheidung „Bewegung nach links“ war, kann die nächste Spieler-Entscheidung nicht „Bewegung nach rechts“ sein, und umgekehrt.

Um einen Bot zu entwickeln, welcher das Spiel optimal spielt, müssten komplexere Weg-Finde-Algorithmen, welche weitere Schritte im Voraus berechnen, implementiert werden. Ziel dieses Bots war es jedoch nur, dass der Spieler das Spiel theoretisch unendlich lange Spielen könnte. Bei einem Test des Bots hat sich eine Punktzahl von über 30.000 ergeben. Das Ziel ist damit also erfüllt. Der Python-Code für den Bot ist unter folgender Bezeichnung zu finden: Bot_plays_Crossy_Road_12_04.py

3. Zusammenfassung

Ziel dieses Projektes war es, eine KI zu trainieren, welche lernt wie man CrossyRoad spielt. Es ist jedoch nicht gelungen, die Belohnungsmechanismen so zu gestalten, dass Spieler verlässlich die generierten Level durchqueren. Da selbst die Ausgänge einer simplen Simulierung der Entscheidungen, als Input für das neuronale Netzwerk nicht ausgereicht haben, liegt die Vermutung nahe, dass eine KI ohne komplexe Weg-Finde-Algorithmen nicht umsetzbar ist. Daher wurde darauf verzichtet die KI weiterzuentwickeln, und stattdessen einen Bot zu programmieren. Mit diesem Bot ist es gelungen Punktzahlen in Höhe von 30.000 zu erzielen, was jede der bisherigen KI’s bei weitem übertrifft. Falsche Parameter, zu viele Inputs- und Entscheidungsmöglichkeiten sind mögliche Gründe für das Scheitern der KI. Weiterhin könnte auch der NEAT-

Algorithmus die Fehlerquelle sein. Unter Umständen würden andere Lern-Algorithmen bessere Ergebnisse erzielen.

4. Ressourcen

Bot_plays_Crossy_Road_12_04.py

CrossyRoad_RaySensors_29_04.py

CrossyRoad_22_03_(unkommentiert).py