Gymnasium Donauwörth Abiturjahrgang 2023

SEMINARARBEIT

Rahmenthema des Wissenschaftspropädeutischen Seminars:

Künstliche Intelligenz

Leitfach: Informatik

Thema der Arbeit:

Entwicklung und Implementierung eines KI-basierten Spielers für das Spiel „Space Invaders"

Verfasser:
David Seitz

Kursleiterin:
May

Abgabetermin:
(2. Unterrichtstag im November)

08. November 2022

Bewertung	Note	Notenstufe in Worten	Punkte		Punkte
schriftliche Arbeit				x 3	
Abschlusspräsentation				x 1	
				Summe:	
				Gesamtleistung nach § 29 (7) GSO = Summe **: 2** (gerundet)	

Datum und Unterschrift der Kursleiterin bzw. des Kursleiters

Inhaltsverzeichnis

1. Grundvoraussetzungen

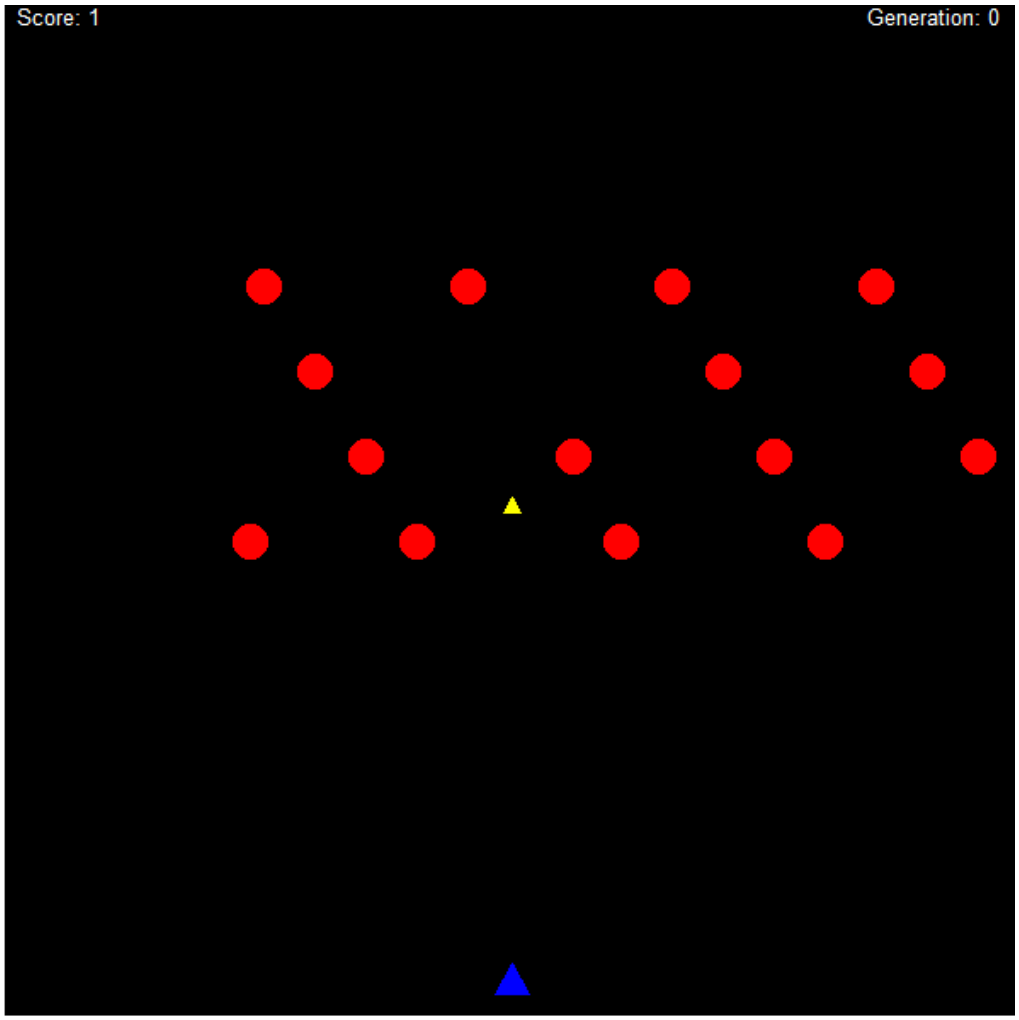
Ein Auto fährt autonom auf der Autobahn, im Internet führen zwei Bots eine Unterhaltung oder der weltbeste Go-Spieler wird von einem PC geschlagen . Was haben diese scheinbar unabhängigen Ereignisse miteinander zu tun? Ihnen allen liegt eine Künstliche Intelligenz (KI) zugrunde. Um zu verstehen wie diese immer alltäglicher werdenden Phänomene überhaupt funktionieren und auf welchen wissenschaftlichen Prinzipien sie basieren, werde ich in dieser Seminararbeit, am Beispiel des Spielers des Spiels Space Invaders, eine künstliche Intelligenz entwickeln und implementieren. Diese KI soll in der Lage sein die Aktionen des Spielers auf einem niedrigen Niveau auszuführen.

1.1. Künstliche Intelligenz

Es ist nützlich, sich bevor man anfängt zu Programmieren, zu überlegen was KI überhaupt bedeutet.
Generell kann man sagen, dass es keine eindeutige Definition für künstliche Intelligenz gibt. Ein Problem dieser Mehrdeutigkeit besteht in der Tatsache, dass selbst menschliche Intelligenz schwer definierbar ist. So wird sie „als Fähigkeit beschrieben, bestimmte Ziele zu erreichen "[Rös21, S.11]. Dies bedeutet nun für den Agenten in meinem Fall, dass dieser in der Lage sein muss Space Invaders spielen zu lernen, um als intelligent zu gelten. Da diese Fähigkeit jedoch unglaublich spezifisch ist, wird oftmals in „schwache" und „starke" künstliche Intelligenz unterschieden. Die„schwache" künstliche Intelligenz beschränkt sich dabei auf einen Bereich, während im Gegensatz dazu, die starke KI bereichsübergreifend agieren kann [vgl. Rös21, S.11]. Somit zählt die in 2 entwickelte KI des Spielers zur schwachen KI.

1.2. Space Invaders

Bevor man nun aber eine künstliche Intelligenz implementiert, lohnt es sich zuerst einen Blick auf die Umgebung zu werfen in dem der Spieler agiert. Da das originale Space Invaders von 1978 ein Arcadegame ist [vgl. Tai78], basiert meine Version auf dem Code Christian Thompson [vgl. Tho15], die in der Programmiersprache Python geschrieben ist.



Ein Screenshot von meiner Space Invaders Version

Wie man im obigen Bild erkennen kann, ist meine Version eher minimal gehalten, sodass nur die wichtigsten Informationen dargestellt werden. Dies ist zum Einen der Fall,

um Rechenleistung zu sparen, zum Anderen macht das visuelle Erscheinungsbild für die künstliche Intelligenz (KI) später keinen Unterschied. Dies liegt an der Art der Daten, welche die KI übermittelt bekommt. Um Rechenleistung und Zeit zu sparen, ist es deshalb sinnvoll, die grafische Darstellung des Trainings zu deaktivieren[1][2].

Zurück zum Aufbau, der Spieler ist das blaue Dreieck, die roten Kugeln die Gegner und das gelbe, kleine Dreieck die Kugel.

Der Spieler hat mehrere Handlungsoptionen:

Er kann sich nach links oder rechts bewegen, nichts tun oder schießen. Falls er sich dafür entscheidet zu schießen, bewegt sich das gelbe Dreieck, die Kugel, senkrecht nach oben. Es ist aber immer nur eine Kugel im Spiel, dass heißt wenn der Spieler schießt und es ist schon eine Kugel im Spiel, passiert nichts. Die Kugel wird zurückgesetzt, wenn sie entweder einen Gegner trifft oder das Spielfeld verlässt.

Von den Gegnern hingegen existieren standardmäßig 16 Instanzen[3]. Sie bewegen sich von einem Spielfeldrand zum Anderen. Wenn ein Gegner den Rand erreicht, werden alle Gegner eine Reihe nach unten gesetzt und ihre Bewegungsrichtung ändert sich. Sie gewinnen, wenn ein Gegner unterhalb des Spielers ist. Im Gegensatz dazu, gewinnt der Spieler, wenn kein Gegner mehr existiert. Damit ein Gegner geschlagen wird, muss ihn die Kugel treffen[4].

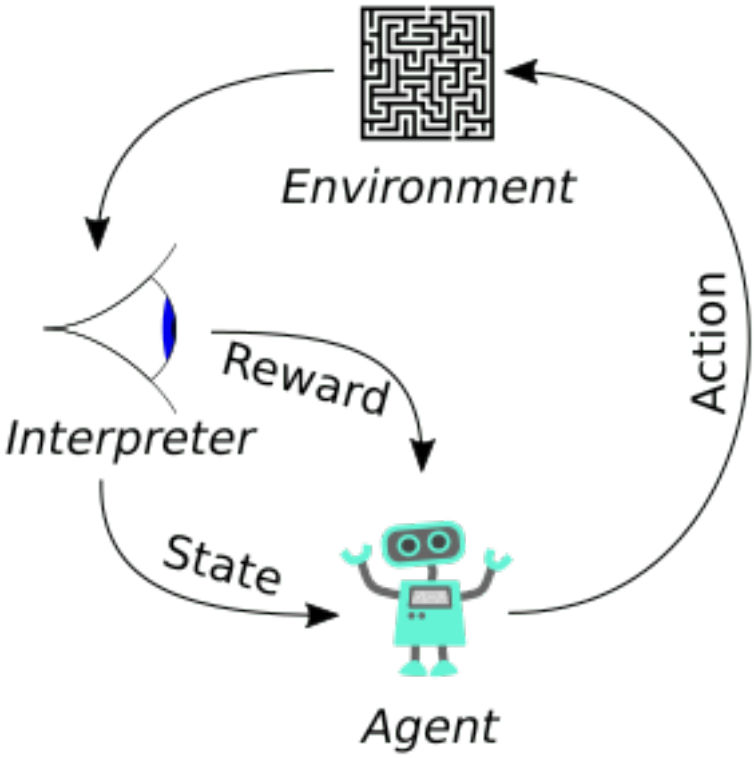
Im weiteren Verlauf dieser Seminararbeit werde ich nun versuchen den Spieler des Spiels Space Invaders durch eine KI zu steuern. Dafür werde ich verschiedene Möglichkeiten, wie Bestärkendes Lernen (Reinforcement Learning/ RL) oder Entscheidungsbäume, betrachten, um zu diese Aufgabenstellung zu lösen. Bei Beispielen werde ich mit Fußnoten darauf zu verweisen, wo das aktuell Diskutierte im Code zu finden ist.

[1] Auf das Training wird in 2.3 noch genauer eingegangen
[2] In main.py mit grafische Darstellung änderbar
[3] Mit self.Monsterzahl veränderbar
[4] Wird mit isCollision() überprüft

2. Bestärkendes Lernen

Das bestärkende Lernen, auch Reinforcement Learning genannt, ist neben dem überwachten Lernen und dem unüberwachten Lernen eine der Möglichkeiten des maschinellen Lernens . Es versucht aus den gegebenen Daten, ohne anderes vorhergehendes Wissen, Informationen zu gewinnen[vgl. Li17, S.7].

2.1. Funktionsweise



The typical framing of a Reinforcement Learning scenario[Meg17]

Auf der Darstellung kann man die typische Funktionsweise des Bestärkenden Lernens

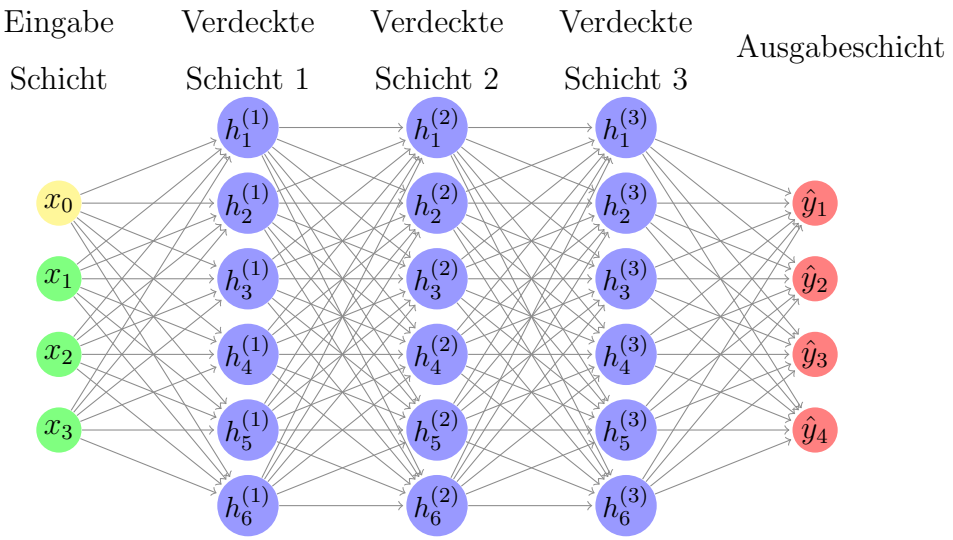
erkennen. Der Agent, führt eine Aktion aus, welche seine Umwelt (enviroment) beeinflusst. Daraufhin bekommt dieser wiederum die Information über den neuen Zustand (state) der Umwelt und eine Belohnung (reward), die auch negativ sein kann. Diese Schleife kann nun unendlich oft wiederholt werden. Für den Spieler im Projekt bedeutet das, dass er eine Aktion (Schuss, Linksbewegen oder Rechtsbewegen) ausführen kann und daraufhin die neuen Koordinaten der Gegner sowie den neuen Punktestand, für dessen Steigen er gleichzeitig eine Belohnung erhält, mitgeteilt bekommt.

2.2. Neuronales Netz

Ein künstliches neuronales Netzwerk, ist ein Modell des zentralen Nervensystems, welches lernt auf Informationen zu reagieren. Wie das Gehirn aus verschiedenen Nervenzellen aufgebaut ist, besteht das Neuronale Netz aus künstlichen Neuronen, welche in verschiedenen Schichten verknüpft sind, wie im nächsten Unterkapitel ausgeführt wird. Ein nicht zu vernachlässigender Unterschied von künstlichen neuronalen Netzen im Vergleich zu dem menschlichen Gehirn besteht in der Anzahl der Verknüpfungen. Denn während unser Gehirn ca. 10^{14} Verknüpfungen beinhaltet, haben selbst die größten künstlichen neuronalen Netzwerke nur einen Bruchteil der Verbindungen (ca. 100.000) [vgl. Tra+03, S.1055]. Eine Konsequenz dieser limitierten Größe besteht zum Beispiel darin, dass es noch keine KI gibt, die alle Aufgaben lösen kann.

2.2.1. Topologie

Topologie bezeichnet in Zusammenhang mit Neuronalen Netzen die Struktur des Netzes, also die Anordnung der Neuronen und deren Verbindungen untereinander.

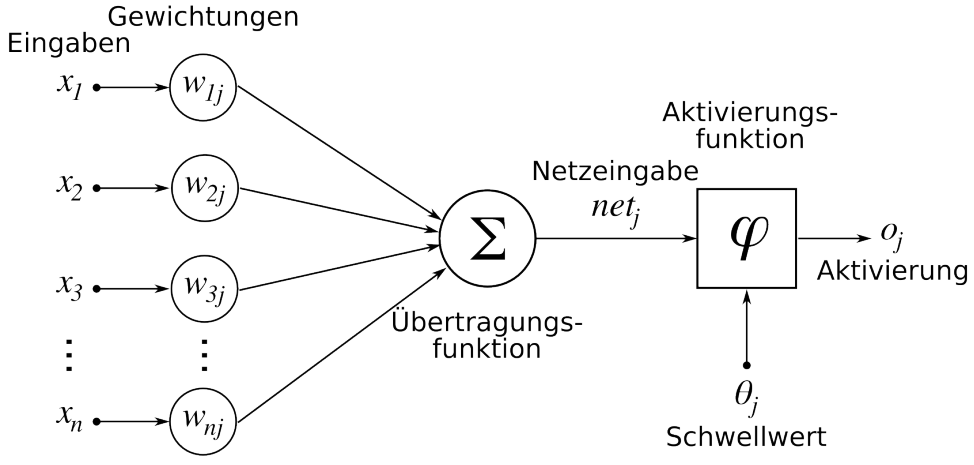


Nicht die Größe des im Projekt verwendeten Neuronalen Netzes

Auf der Abbildung oben erkennt man das Modell eines Neuronales Netzes mit einer Eingabeschicht mit 4 Eingabewerten, 3 verdeckten Schichten mit jeweils 6 Neuronen und einer Ausgabeschicht mit 4 Neuronen. Ebenfalls erkennt man, dass es sich bei der Art des Neuronalen Netzes um ein Feedforward Netz handelt, da die Daten von den Neuronen an die Neuronen der darauffolgenden Schicht weitergegeben werden.[5]

2.2.2. Künstliche Neuronen

Wie oben schon genannt bestehen Neuronale Netze aus künstlichen Neuronen, welche natürlichen nachempfunden sind.



[Chr05]

[5] In der Programmierung wird die Größe des Neuronalen Netzes mit der Methode HiddenLayers(AnzahlHiddenLayers) festgelegt

Wie man auf der Abbildung erkennen kann wird in einem künstlichen Neuron die Summe der Produkte aus Gewichten und Eingaben durch die Aktivierungsfunktion zur nächsten Eingabe.

Aktivierungsfunktion

Aktivierungsfunktionen werden dazu benutzt um die Information einer Schicht des neuronalen Netzwerks zu verarbeiten um sie an die nächste Schicht weiterzugeben.

Das Benutzen einer Aktivierungsfunktion ist von Vorteil, da so dem Neuronalen Netzwerk ermöglicht wird, komplexere Probleme zu lösen. Ein Neuronales Netz ohne Aktivierungsfunktion zu trainieren ist trotzdem möglich, es könnte aber nur lineare Probleme lösen.

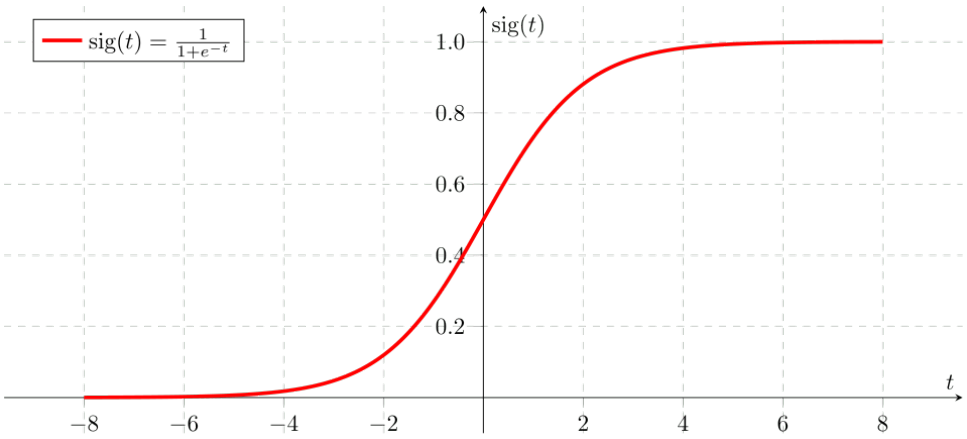
Deshalb werden im Maschinellen Lernen, wenn es mehrere verdeckte Schichten gibt oftmals Aktivierungsfunktionen benutzt.

Sigmoidfunktion

Die Sigmoidfunktion ist die am häufigsten benutzte Aktivierungsfunktion, da sie eine nicht lineare Funktion ist. Sie kann folgendermaßen beschrieben werden:

$$\mathrm{f(x)} = \frac{1}{1+e^{-x}}$$

Im Projekt wurde diese Sigmoidfunktion ebenfalls verwendet, da sie nicht linear ist, die Werte in einen Bereich zwischen 0 und 1 transformiert und ableitbar ist [vgl. SSA17, S.312 Abschnitt c].



Plot der Sigmoidfunktion im Intervall [-8;8] [Mar14]

Für zu große x-Werte kann es in Python zu einem „Overflow Error"kommen. In diesem Fall wird f(x) zu Null wenn x kleiner 0 ist und zu Eins wenn x größer 0 ist.

2.3. Optimierungsfunktion

Die Optimierungsfunktion ist im Projekt der Algorithmus, der die Gewichte des Neuronalen Netzes anpasst[6] nach der Form:

*Reward Increment = Nichtnegativer Faktor * Offset Reinforcement * Characteristic Eligibility* [vgl. Wil92, S.234]

oder

$$\Delta w_{ij} = \alpha_{ij}(r - b_{ij})e_{ij}$$

Wobei Δw_{ij} die Gewichtsänderung darstellt, der nicht negative Faktor α_{ij} auch eine Konstante sein kann, die auch als Lernfaktor bekannt ist[7], der Minuend r die aktuelle Punktzahl, der Subtrahend b die maximale Punktzahl aller Generationen und e_{ij} ein Produkt $e_{ij} = ln(f_{ij}) \cdot \frac{1}{w_{ij}}$ mit f_{ij}, dem durch die Aktivierungsfunktion berechneten Ausgabe eines Neurons und w_{ij} dem derzeitigen Gewicht des Neurons.

[6]Im Projekt die Methode: evaluieren(score)
[7]Im Projekt: self.Lernfaktor = 3

Diese Funktion hat jeden Durchgang eine Chance[8] aufgerufen zu werden, um das Neuronale Netz zu aktualisieren.

Probleme der Optimierungsfunktion

Wenn b gleich r wird die ganze Gleichung 0, somit kann kein weiteres Lernen mehr stattfinden.

2.4. Probleme des Bestärkenden Lernens

Beobachtet man den maximalen Score des Spielers in der derzeitigen Konfiguration über mehrere Generationen und Anläufe hinweg, so stellt man fest, dass dieser selten, wenn nicht sogar nie, über 5 Punkte hinausgeht. Daher muss ich nach den Problemen suchen, die verhindern, dass der Agent mehr lernt.

2.4.1. Underfitting

Underfitting (deutsch Unteranpassung) ist ein Problem, das im bestärkenden Lernen auftreten kann, wenn das Modell weniger komplex ist, als die Daten mit denen es zu arbeiten versucht[vgl. Koe18, S.6-7]. Dies ist zu Beispiel der Fall, wenn man versucht ein lineares Modell auf ein nicht lineares Problem anzuwenden. Dies ist bei dem hier verwendeten Neuronalen Netzwerk nicht der Fall, da die in 2.2.2 beschriebene Aktivierungsfunktion nicht linear ist und somit das Modell komplex genug sein sollte, um Space Invaders zu schlagen.

2.4.2. Overfitting

Was eher der Fall sein könnte, ist eine Art des sogenannten Overfittings, das im Deutschen als Überanpassung bekannt ist. Hierbei passt der Agent seine Strategie zu genau

[8] In der Methode maingameloop() mit der lokalen Variable *ch* änderbar

an das Problem an und probiert keine neuen Strategien mehr aus. Dieser Fall tritt hier ein, wenn der Spieler lernt nur zu schießen, weil er damit eine Punktzahl von 5 erreicht und diese höher ist als die vorherige Punktzahl. Eine Möglichkeit, dieser Überanpassung vorzubeugen, besteht darin, das Problem leicht zu variieren. Zum Beispiel kann man die Gegner jede Runde zufällig platzieren.[9]. Dies kann aber auch zur Folge haben, dass der Agent das Problem zufällig löst. Auch könnte dieses Problem gelöst werden, indem man die Belohnung an die vergangene Zeit koppelt, ähnlich wie ich noch in 3.1.1 erklären werde.

[9]Dies passiert wenn man in der reset-Funktion die lokale Variable random auf False setzt

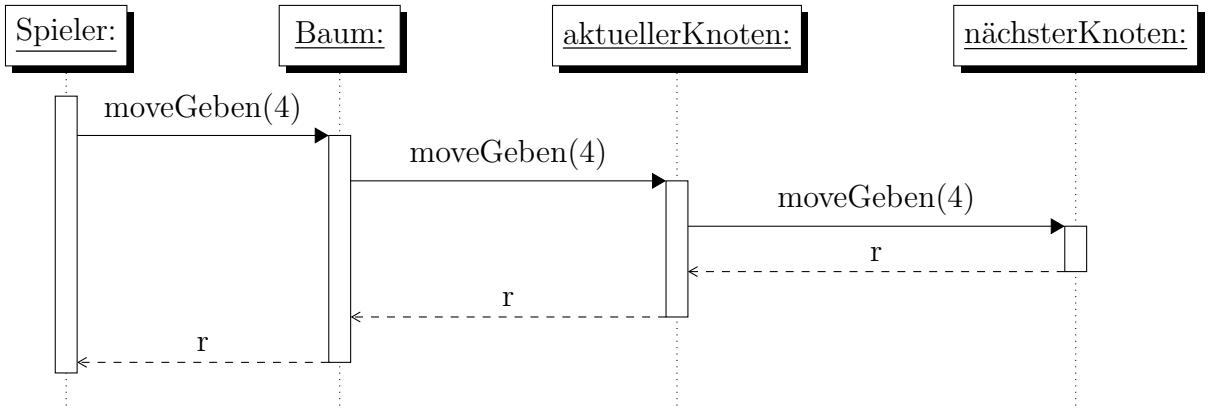
3. Alternativen zum Bestärkenden Lernen

Die in 1.1 aufgestellte Definition von Künstlicher Intelligenz wirft aber die Frage auf, wo künstliche Intelligenz überhaupt beginnt. Um diese Frage zu beantworten, werden in 3 nun zwei Alternativen zum Bestärken Lernen vorgestellt.
Eine weitere Idee, eine Künstliche Intelligenz zu trainieren, welche sich nicht auf ein neuronales Netz stützt, besteht darin, Entscheidungsbäume zu verwenden. Da das System, in welchem sich der Spieler bewegt, nicht überaus kompliziert ist, ist diese Möglichkeit durchaus machbar.

3.1. Entscheidungsbäume

Entscheidungsbäume zählen zu den wissensbasierten Systemen, die Wissen, das aus Beobachtungen oder Regeln besteht, speichern [vgl. Qui86, S.81-82]. Sie sind auch oft einfacher zu verstehen, als andere Systeme, wie zum Beispiel Neuronale Netzwerke. Entscheidungsbäume sind aber manchmal so komplex, dass die wahre Herausforderung darin besteht, die Pfade zu finden, welche zum gewünschten Ergebnis führen.

3.1.1. Implementierung im Projekt



Sequenzdiagramm einer möglichen Aktion des Spielers

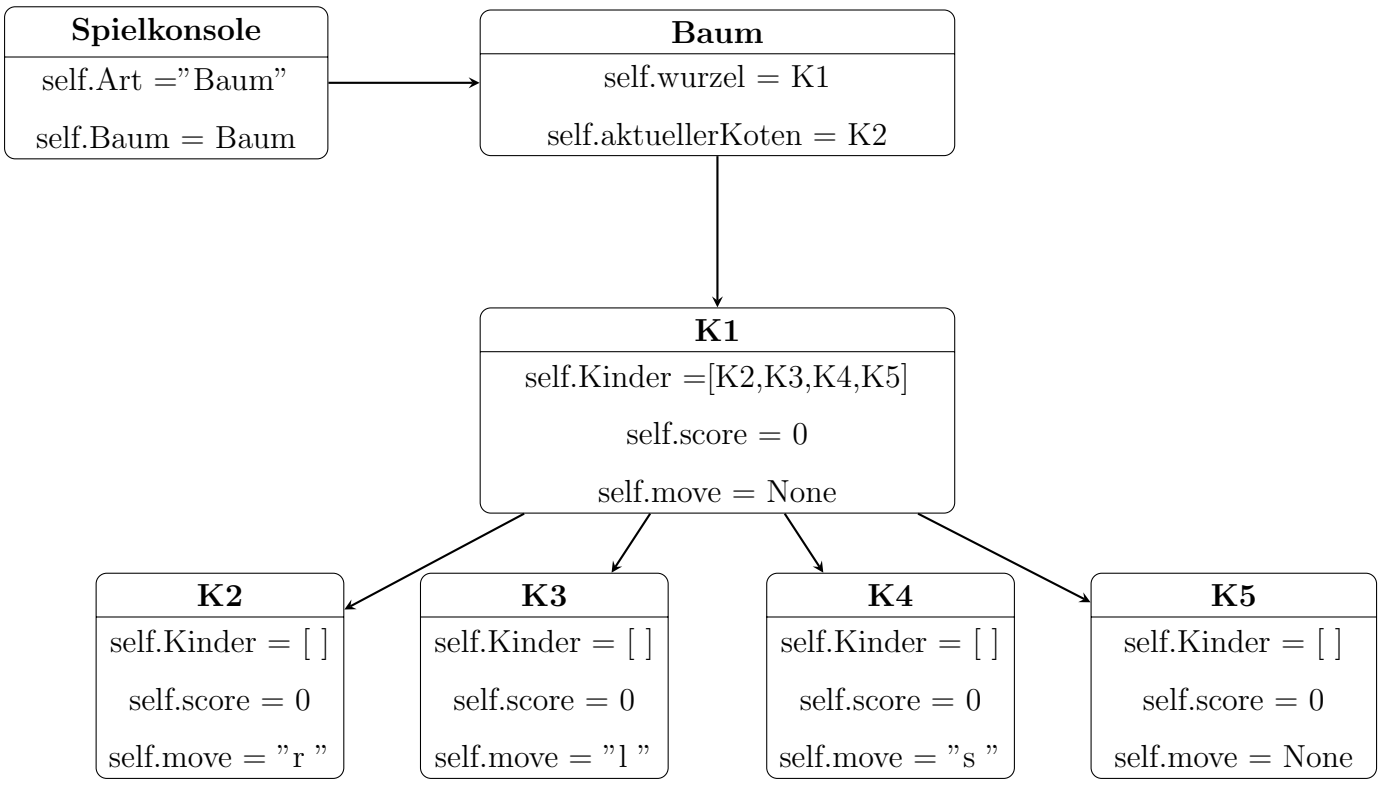
In diesem Diagramm kann man erkennen, wie der Spieler den Baum nach einer Aktion fragt. Bei der Methode moveGeben(4) drückt die 4 einen Übergabewert aus, der einen simulierten Score übergibt. Den tatsächlichen Score als Übergabewert zu benutzen ist auch möglich, aber weniger effektiv, da je früher ein Gegner zerstört wird, der nächste Gegner zerstört werden kann. Die Funktion F, die den Übergabewert errechnet, besitzt eine ähnliche Aufgabe zur Aktivierungsfunktion im Neuronalen Netz.

$$F(score, t_1, t_2) = score \cdot \frac{t_2 - t_1}{10}$$

Der simulierte Score F wird mit der obigen Gleichung berechnet, wobei t_1 die Zeit in Sekunden zu Beginn des Durchlaufs, t_2 die Zeit in Sekunden zum Zeitpunkt der Abfrage und *score* die tatsächliche Punktzahl ist.

Man kann diese Funktion noch verbessern, indem man den außerdem die Position des Gegners miteinbezieht, da, falls der äußerste Gegner zerstört wird, die Schritte bis zum Näherkommen der Gegner zunehmen. Somit ist es wichtiger, äußere Gegner zu zerstören, als Innere.
Der aktuelle Knoten vergleicht die Punktzahl seiner Kinder untereinander und gibt mit einer 19:1 Wahrscheinlichkeit den Move des Knotens mit der höchsten Punktzahl zurück.

Objektdiagramm



Objektdiagramm, dass einen Ausschnitt eines Baums zeigt

Wie im Sequenzdiagramm in 3.1.1 schon erläutert, werden die Aktionen des Spielers durch eine rekursive Methode ermittelt. Diese vergleicht die Punktzahlen der Kinder des aktuellen Knotens und die Aktion des Kindes mit der höchsten Punktzahl, mit einer Wahrscheinlichkeit von 95% zurückgibt.
Was passiert aber, wenn keine Kinder vorhanden sind oder der 20ste Fall eintritt?
In beiden Fällen wird zufällig eine Aktion ausgewählt und zurückgegeben. Im ersten Fall müssen zusätzlich noch Kinder erzeugt werden.

3.1.2. Probleme und Einschränkungen

Eine Beobachtung, welche ich im Laufe der Testläufe des Programms gemacht habe, besteht darin, dass die zufällige Art des Entscheidens, die ich verwendet habe, nur dann funktioniert, wenn die Gegneranzahl relativ klein ist (ca. 12).

Ein weiteres Problem, das mir bei meiner Implementierung im Weg stand, besteht im Speicherplatzverbrauchs des Baums. Denn während ein Neuronales Netz eine fixe Größe hat, nimmt die des Baums stetig zu. So hat zum Beispiel ein Baum nach vier Aktionen des Spielers vier Knoten und nach 16 Aktionen 16 Knoten. Im Gegensatz dazu bleibt die Größe eines künstlichen neuronalen Netzes konstant. Hinzu kommt ebenfalls, dass nicht alle Pfade im Baum sinnvoll sind, weil sie zum Beispiel das Gleiche aussagen.
Folgendes Beispiel:

	1	2	3	4
Aktion des Spielers	-	r	-	r
Alternativ	r	r	-	-

r $\hat{=}$ Schritt nach rechts; - $\hat{=}$ keine Bewegung

Auf dieser Tabelle sieht man, dass es mehrere Aktionsabfolgen des Spielers gibt, welche zum exakt gleichen Ergebnis führen. In diesem Fall bewegt sich der Spieler zwei Schritte nach rechts. Die Abfolge der Aktionen ist in diesem Fall egal, da nur die Position des Spielers zum Zeitpunkt eines Schusses einen Unterschied auf das Ergebnis macht.

3.1.3. Lösungsmöglichkeiten

Einige Maßnahmen die zur Verbesserung des Baumes getroffen werden können, werden im Folgenden aufgezählt und kurz erläutert:

- Man kann Pfade, welche nicht zum Ergebnis führen, trimmen (prunen) [vgl. Qui96, S.72], indem man beispielsweise den Pfad durch ein einzelnes Blatt ersetzt. Dies führt dazu, dass Speicherplatz frei wird, der dann wiederum für neue Pfade genutzt werden kann.

- Auch können mehrere Bäume zu Einem zusammengefügt werden, wodurch sich die Komplexität des Baumes vergrößert, indem nun in diesem Baum mehr Pfade enthalten sind[vgl. Qui96, S.72]. Dies trägt in meinem spezifischen Fall aber nicht zur Lösung des Problems des begrenzten Speicherplatzes bei. Im Gegenteil, es verstärkt

dieses Problem nur weiter, weil der neue Baum komplexer ist und somit mehr Speicherplatz verbraucht.

- Indem man in einem bestehenden Baum neue Pfade einfügt, anstatt immer wieder den kompletten Baum nach einem Durchlauf des Programms neu zu Generieren, verringert die Zeit, welche zum Generieren neuer Pfade benötigt wird [vgl. Qui96, S.72]

3.2. Ein einfacher Algorithmus

Alternativ könnte man auch einfach selbst einen Algorithmus schreiben mit dem die Aktionen des Spielers bestimmt werden können, wie folgendes Minimalbeispiel in Pseudocode:

```
def moveGeben(x):
    if x < -180:
        shoot()
    else:
        moveleft()
```

Natürlich wird dieses Minimalbeispiel nicht funktionieren, wenn es eine mittlere bis große Anzahl an Gegnern gibt. Da aber die Position der Gegner, die Gegnergeschwindigkeit, die Kugelgeschwindigkeit, sowie die Spielerposition bekannt sind, könnte man ein Gleichungssystem aufstellen. Dieses berechnet dann, wann der Spieler wo schießen muss, um einen Gegner zu treffen. Falls die Gegneranzahl dann so groß ist,das diese verbesserte Version aufhört zu funktionieren, kann man weitere Parameter, wie die Reihenfolge mit der die Gegner abgeschossen werden sollen, einführen.
Somit kann dieser Ansatz unter Miteinbeziehung aller möglichen Parameter auch ohne Lernfähigkeit effektiver sein, als jede mögliche Künstliche Intelligenz.

3.3. Fazit

Während der in 3.1 vorgestellte Entscheidungsbaum und der dazugehörige Algorithmus meine Kriterien für künstliche Intelligenz erfüllen, nämlich Lern- und Problemlösungsfähkeit (wenn auch nur für geringe Gegnerzahlen), ist der Algorithmus, selbst bei der losesten Definition von Künstlicher Intelligenz nicht intelligent, da die Lernfähigkeit fehlt. Ich habe ihn trotzdem aufgenommen um aufzuzeigen, dass es keiner Neuronalen Netze oder Entscheidungsbäume bedarf um die einfache Aufgabe, Space Invaders zu schlagen, zu meistern. Ein weiterer Aspekt, in dem sich Algorithmus und Künstliche Intelligenz unterscheiden, wird deutlich, wenn man den letzten Satz aus 3.2 betrachtet: „Somit kann dieser Ansatz unter Miteinbeziehung aller möglichen Parameter auch ohne Lernfähigkeit effektiver sein, als jede mögliche Künstliche Intelligenz“. Besonders auf den Teil der „ *Miteinbeziehung aller möglichen Parameter*“ möchte ich aufmerksam machen, da es in der wirklichen Welt zu viele Parameter gibt, als das diese alle in einem Modell berücksichtigt werden können.

4. Schlussgedanken

Wenn man die Zeit, welche ich dafür verwendet habe ein funktionierendes Neuronales Netzwerk zu entwickeln und programmieren, mit der Zeit gegenüberstellt, die für ich den Algorithmus in 3.2 gebraucht habe, so stellt man fest, das die eine Aufgabe über 20 Stunden und die andere nur um die 5 Minuten gedauert hat. Dabei ist das Ergebnis der einfacheren Aufgabe besser , als das der Komplizierteren. Folglich kann man daraus schließen, dass man für eine einfache Aufgabe, wie das Computerspiel Space Invaders zu schlagen, keine komplizierten Methoden anwenden sollte. Stattdessen sollte man die Methoden verwenden, die am schnellsten zum Ziel führen. Man muss auch bedenken, dass diese einfacheren Methoden einfacher zu verbessern, warten und auch verständlicher sind. Im Nachhinein hätte das Ziel für meine Seminararbeit nicht lauten sollen: „Versuch etwas Neues, dass noch niemand getan hat“, sondern „Benutze das, von dem du weißt, es funktioniert“.
Zudem bin ich einige Tage vor der Abgabe auf OpenAI Gym gestoßen [vgl. Bro+16]. Ein Projekt auf den man seine KI unkompliziert implementieren und verschiedene Lernalgorithmen miteinander vergleichen kann. Leider funktioniert die Bibliothek, welche dafür verwendet wird, nur auf LINUX und MacOs, aber nicht Windows, das ich benutze, ansonsten hätte ich eine einfache Möglichkeit mein Projekt mit Anderen zu vergleichen.

Falls man etwas von dieser Seminararbeit mitnehmen sollte ist es Folgendes: Ein Auto kann autonom fahren, zwei Bots können eine Unterhaltung führen oder der weltbeste Go-Spieler kann von einem PC geschlagen werden. Dabei darf man aber nicht vergessen, dass alle diese Fortschritte erst durch menschliche Intelligenz ermöglicht wurden. Somit sollte man diese Errungenschaften nicht der KI zusprechen, sondern deren Erschaffern.

A. Anhang

Auf dem beigelegten USB-Stick befinden sich:

- Die Seminararbeit in digitaler Form
- Die entsprechende Literatur (der Dateipfad ist in der Quellen.bib Datei hinterlegt)
- Das Programmierprojekt im Ordner Space Invaders

Alle obengenannten Dateien findet man auch auf: *gitlab.infolernen.de/seitzda1/seminararbeit*

Literatur

[Bro+16] Greg Brockman u. a. *OpenAI Gym*. 2016. eprint: `arXiv:1606.01540`.

[Chr05] Chrislb. *Schema eines künstlichen Neurons: Künstliches Neuron mit Index j*. Wikipedia. Juli 2005. URL: `https://commons.wikimedia.org/w/index.php?curid=224561` (besucht am 22. 08. 2022).

[Koe18] Will Koehrsen. »Overfitting vs. underfitting: A complete example«. In: *Towards Data Science* (2018).

[Li17] Yuxi Li. »Deep reinforcement learning: An overview«. In: *arXiv preprint arXiv:1701.07274* (2017).

[Mar14] MartinThoma. *Sigmoid-function*. Wikipedia. Mai 2014. URL: `https://commons.wikimedia.org/wiki/File:Sigmoid-function-2.svg` (besucht am 05. 11. 2022).

[Meg17] Megajuice. *Reinforcement learning diagram*. Wikipedia. Apr. 2017. URL: `https://commons.wikimedia.org/wiki/File:Reinforcement_learning_diagram.svg` (besucht am 04. 11. 2022).

[Qui86] J. Ross Quinlan. »Induction of decision trees«. In: *Machine learning* 1.1 (1986), S. 81–106.

[Qui96] J. Ross Quinlan. »Learning decision tree classifiers«. In: *ACM Computing Surveys (CSUR)* 28.1 (1996), S. 71–72.

[Rös21] Alexander Maximilian Röser. *Charakterisierung von schwacher und starker Künstlicher Intelligenz*. 79. Arbeitspapiere der FOM, 2021.

[SSA17] Sagar Sharma, Simone Sharma und Anidhya Athaiya. »Activation functions in neural networks«. In: *towards data science* 6.12 (2017), S. 310–316.

[Tai78] Kabushiki Kaisha Taito. *Space invaders*. US Copyright Office. US-Pat. 1978. URL: https://cocatalog.loc.gov/cgi-bin/Pwebrecon.cgi?v1=1&ti=1,1&SEQ=20221105063508&Search_Arg=Space%20Invaders&Search_Code=TALL&CNT=25&REC=0&RD=0&RC=0&PID=TeH7MUs-D-LOzuUl1AWCckFSz&SID=1 (besucht am 05. 11. 2022).

[Tho15] Christian Thompson. *Space Invaders*. Youtube. Nov. 2015. URL: https://github.com/wynand1004/Projects/tree/master/Space%20Invaders (besucht am 05. 11. 2022).

[Tra+03] M Traeger u. a. »Künstliche neuronale Netze«. In: *Der Anaesthesist* 52.11 (2003), S. 1055–1061.

[Wil92] Ronald J Williams. »Simple statistical gradient-following algorithms for connectionist reinforcement learning«. In: *Machine learning* 8.3 (1992), S. 229–256.

Eidesstattliche Erklärung

Ich habe diese Seminararbeit ohne fremde Hilfe angefertigt und nur die im Literaturverzeichnis angeführten Quellen und Hilfsmittel benutzt.

Ort Datum Unterschrift