Künstliche Intelligenz - Stand der Bibliotheken und Programmierung einer künstlichen Intelligenz für Super Mario

Bachelor-Thesis

zur Erlangung des akademischen Grades B.Sc.

Henry Tieu

[redacted]

Hochschule für Angewandte Wissenschaften Hamburg
Fakultät Design, Medien und Information
Department Medientechnik

Erstprüfer: Prof. Andreas Plaß

Zweitprüfer: Prof. Nils Martini

Hamburg, 01.08.2019

Inhaltsverzeichnis

Inhaltsverzeichnis **2**
1 Abstract **3**
2 Einleitung in Künstliche Intelligenz **4**
2.1 Was ist eine Künstliche Intelligenz? 4
2.2 Mustererkennung und Prognose 5
2.3 Starke und schwache künstliche Intelligenz 7
2.4 Machine Learning 7
2.5 Deep Learning 9
2.6 Stand der KI-Bibliotheken 11
3 Konzept einer KI für Super Mario Bros **12**
3.1 Super Mario Bros 12
3.2 OpenAI Gym Retro 13
3.4 Anforderung an die KI 14
3.4 Problematik 14
4 KI und Videospiele **15**
4.1 Entwicklung der KI in Videospielen 15
4.2 KI als Spieler 16
4.2.1 KI gegen Singleplayer Spiele 16
4.2.2 KI gegen reale Spieler 17
4.2.3 OpenAI Five 19
5 Lernalgorithmen **21**
5.1 Genetic Algorithms 22
5.1.1 NEAT(NeuroEvolution of Augmenting Topologies) 23
5.2 Reinforcement Learning 27
5.2.1 PPO(Proximal Policy Optimization) 28
5.3 Auswahl des Lernalgorithmus 31
6 OpenAI Retro SuperMarioBros(Neat) **32**
6.1 Umsetzung 32
6.1.1 Software, Umgebung und Bibliotheken 32
6.1.2 Aufbau und Durchführung 33
6.1.3 Beobachtung 37
6.2 Schlussfolgerung und Probleme 39
7 Fazit **42**
8 Abbildungsverzeichnis **43**
9 Softwareverzeichnis **45**
10 Literaturverzeichnis **46**

1 Abstract

Künstliche Intelligenzen nutzen wir heutzutage in den verschiedensten Anwendungsgebieten, zum Beispiel bei der Gesichtserkennung in unseren Smartphones. Meine Arbeit befasst sich mit künstlichen Intelligenzen in Videospielen und dem Stand der Open Source Bibliotheken in diesem Bereich. Dabei betrachte ich, wie einfach es ist, eine künstliche Intelligenz für Nutzer zu erzeugen, die keine Erfahrung in der Arbeit mit künstlichen Intelligenzen besitzen.

Um diese Forschungsfrage zu beantworten, programmiere ich in Python mit Hilfe der OpenAI - Gym Retro Bibliothek eine einfache künstliche Intelligenz für das Videospiel Super Mario Bros. Den Algorithmus, den ich hierfür verwende, ist der genetische “Neuroevolution of Augmenting Topologies” Algorithmus (NEAT-Algorithmus).

Der derzeitige Stand von Open Source Bibliotheken bietet eine Vielzahl an Möglichkeiten für Anfänger und erfahrene Nutzer künstliche Intelligenzen zu entwickeln. Über die Bibliotheken und Frameworks lassen sich bereits trainierte Modelle oder eingebaute Emulatoren für alte Videospielkonsolen nutzen. Außerdem lassen sich neuronale Netze grafisch darstellen und es ist möglich, ohne eine Zeile Quelltext zu schreiben, eine einfache künstliche Intelligenz zu erzeugen.

2 Einleitung in Künstliche Intelligenz

2.1 Was ist eine Künstliche Intelligenz?

Künstliche Intelligenz (KI) oder im englischen Artificial Intelligence (AI) ist ein Teilgebiet der Informatik. KI befasst sich mit der Erforschung der Automatisierung des intelligenten menschlichen Verhaltens und des maschinellen Lernens. Maschinen oder Programme werden konstruiert, die ein “intelligentes” menschliches Verhalten simulieren sollen. Sie erledigen Aufgaben selbst und liefern keine fest programmierten Ergebnisse, sondern geben uns Ergebnisse, für die eine menschliche “Intelligenz” benötigt wird. Der Begriff Intelligenz ist oftmals nicht konkret eingrenzbar und klar definiert, wird aber trotzdem für die Beschreibung des Programmverhaltens genutzt. Intelligent wird ein Programm bezeichnet, wenn es abstrakte Probleme lösen und aus ihnen lernen kann. Der Lernprozess geschieht über die Lernalgorithmen, die versuchen, das neuronale Netz des menschlichen Gehirns nachzubilden.

KIs werden im Alltag häufig genutzt. Sie befinden sich im Großteil der neuen Haushaltsgegenstände, in Autos, Videospielen und Suchmaschinen. Gegenstände erhalten oftmals den Namen “Smart”, wenn sie von KIs gesteuert werden (z.B. das Smart Home oder Smart Factory). Das Smartphone ist ein gutes Beispiel für unsere alltägliche Nutzung von KIs . Sprach-,Schrift- und Gesichtserkennung nutzen alle eine KI. KIs sind auch in der Lage, komplexere Aufgaben zu überwinden. Durch ihre Lernfähigkeit können sie Musik komponieren, Krankheiten diagnostizieren oder Bilder zeichnen bzw. rekonstruieren. Mit jedem Jahr verbessern sich die Fähigkeiten der KIs durch die Entwicklung und Forschung in bessere Hardware und Algorithmen.

2.2 Mustererkennung und Prognose

KIs besitzen immer zwei elementare Fähigkeiten.
Die Mustererkennung und Prognose aus den vorliegenden Informationen. Die Mustererkennung ist in fünf Schritte unterteilt. Der erste Schritt ist die Aufnahme von Information durch z.B. Sensoren oder einer Kamera. Nach der Aufnahme kommt die Vorverarbeitung. In diesem Schritt werden die Daten skaliert, normiert und aufgeteilt, außerdem wird Rauschen beseitigt. Die vereinheitlichten Daten werden dann im nächsten Schritt nach Merkmalen gefiltert, welche dann extrahiert werden.
Die Daten werden dann in kompakte Formen transformiert. Im Schritt der Merkmalsextraktion erreicht man den Punkt, an dem die KI lernt und seine Fähigkeit Merkmale zu filtern verbessert. In der Merkmalsreduktion werden nicht relevante Merkmale gefiltert und entfernt. Die übrigen Merkmale werden auf ihre Trennbarkeit geprüft und auf wesentliche Merkmale reduziert. Im letzten Schritt der sogenannten Klassifikation werden die Merkmale der Objekte anhand des Klassifikators vergleicht. Dies wird oft mithilfe von künstlichen neuronalen Netzen umgesetzt, welche sich vor allem für komplexe Muster eignen. Im Anschluss gibt es die Option eine Musteranalyse durchzuführen, die z.B. bei einer Bilderkennung, Objekte innerhalb des Bildes erkennt.

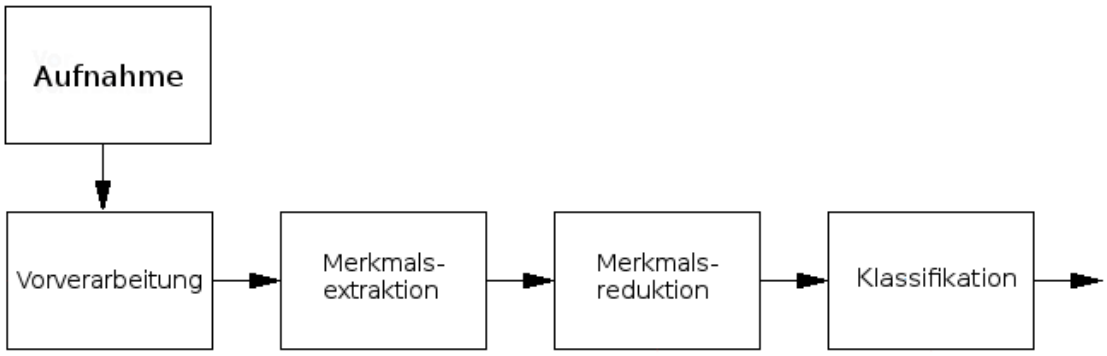

Abbildung 1: Mustererkennung Schema

Für die Prognose benötigt die KI das trainierte Modell der Mustererkennung. Die KI kann aus den Daten einschätzen was das Objekt ist oder vorhersagen welches Objekt als nächstes in der Reihe sein wird. Dies geschieht mit Hilfe des trainierten Modells. Aus den Daten kann mittels des Modells ein Muster oder Objekt erkannt werden. Soll vorhergesagt werden welche Klassifizierung das Objekt besitzt, wird das Objekt auf Merkmale überprüft. Die Merkmale werden dann mit vorhandenen Merkmalen im Modell verglichen und dementsprechend liefert die KI eine Klassifizierung mit der größten Übereinstimmung von Merkmalen. Wenn aus den Daten das nächste Objekt vorhergesagt wird, werden die Daten auf ein Muster analysiert. Wenn ein Muster gefunden wird, liefert die KI ein Objekt was die höchste Wahrscheinlichkeit besitzt als nächstes in der Reihe aufzutreten.

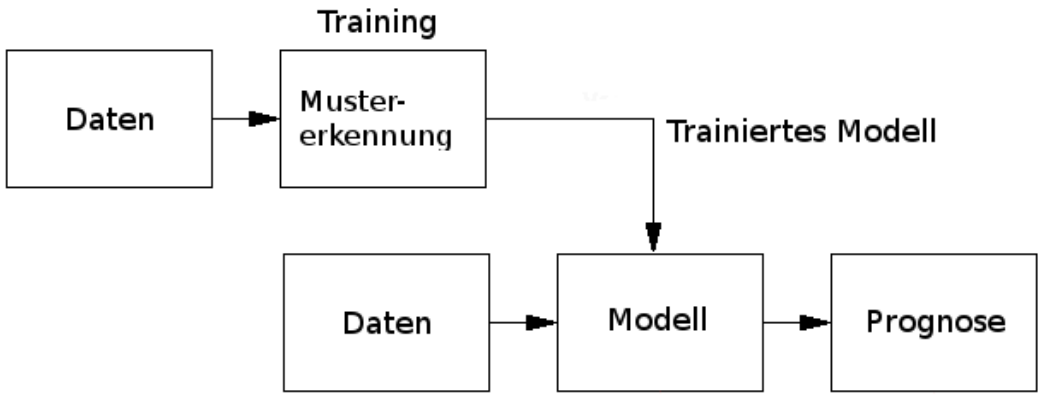

Abbildung 2: Prognoseschema

2.3 Starke und schwache künstliche Intelligenz

KIs lassen sich in zwei Typen kategorisieren. Die schwache KI und die starke KI. Die schwache KI fokussiert sich auf die Lösung von konkret definierten Anwendungsproblemen. Die Lösung erfolgt auf Basis von Methoden der Mathematik und Informatik. Dabei werden die Methoden speziell entwickelt oder je nach Anwendung angepasst. Das resultierende Programm hat die Fähigkeit zu lernen und sich selbst zu optimieren. Das Programm ist beschränkt auf den speziellen Anwendungsbereich und kann keine sinnvollen Ergebnisse in anderen Bereichen hervorbringen. Alle Systeme, die derzeit existieren, sind als schwache KIs kategorisiert.

Starke KIs, oder auch Superintelligenz, haben das Ziel, die intellektuellen Fähigkeiten des Menschen zu erlangen bzw. sie zu übertreffen. Sie handelt von selbst und kann selbst Entscheidung treffen. Sie besitzt eine universelle Intelligenz und kann in verschiedenen Bereichen Leistungen erbringen und sogar mehrere Fähigkeiten selbständig kombinieren, um ein Ziel zu erreichen. Derzeit ist man noch nicht in der Lage eine starke KI zu entwickeln. Ob die Entwicklung einer solchen KI überhaupt möglich ist wird weiterhin diskutiert. Laut der Studie “Future Progress in Artificial Intelligence: A Survey of Expert Opinion” von Nick Bostrom, ist das durchschnittlich vorhergesagte Jahr, in dem eine 90 % Wahrscheinlichkeit besteht, eine Superintelligenz zu entwickeln, 2075. Eine große Frage ist noch ob diese KI ein Bewusstsein entwickeln kann und Empathie empfindet. Auch auf diese Frage sind sich die Forscher nicht einig, doch gehen die Forscher davon aus, dass sie keine wirkliche Empathie empfinden wird, sondern diese nur simulieren könne.

2.4 Machine Learning

Machine Learning ist ein Teilgebiet der KI. Machine Learning Systeme werden nicht explizit programmiert, sondern trainiert. Dieser wird mit Daten versorgt, welche relevant für die zu lösenden Aufgaben sind. Mit Hilfe dieser Daten werden Erkenntnisse gewonnen, die wiederum verallgemeinert werden können, um für neue Probleme Regeln zu erzeugen, die für die Automatisierung der Aufgabe genutzt werden. Bevor das System eigenständig lernen kann, benötigt es relevante Dateien und Algorithmen. Das Erkennen von Mustern und die Generierung von Lösungen geschieht mittels Algorithmen, die sich in drei große Gruppen aufteilen lassen.

Algorithmen, die sich in der Gruppe des überwachten Lernens befinden, benötigen für den Lernprozess Datenpaare, die aus Inputs und Outputs bestehen. Die Outputs sind hierbei Inputs, die bereits mithilfe der geforderten Logik gelöst sind. Ziel ist mithilfe verschiedener Datenpaaren dem Netz die Fähigkeit anzutrainieren Assoziationen zu erstellen. Daraus folgend soll das Netz die Fähigkeit haben Daten, die eine ähnliche Struktur zu den gelernten Datensätzen besitzen, mithilfe der erzeugten Assoziation eigenständig zu klassifizieren. Im Unterschied zum überwachten Lernen benötigt die zweite Gruppe der Algorithmen, das unüberwachte Lernen, keine Datensätze die aus Paaren von Inputs und Outputs bestehen. Das Netz erhält Daten welche nur aus Inputs bestehen.

Was für eine Form oder Logik die Outputs haben sollen, bleiben dem Netz unbekannt. Die Inputs werden über eine Clusteranalyse anhand ihres Musters in sogenannte Cluster gruppiert.
Die Clusteranalyse ist ein Verfahren um Ähnlichkeitsstrukturen in Datenbeständen zu entdecken. Da nicht spezifiziert wird, wie das Netz die Daten klassifizieren soll oder wie die Aufgabe ist, liefert er uns Outputs, die er aus seiner eigenen Klassifizierung bestimmt. Die dritte große Gruppe der Algorithmen ist das bestärkende Lernen, öfter auch Reinforcement Learning genannt. Sie ist an das natürliche Lernverhalten des Menschen orientiert. Das Netz kann in diesem Fall mit spezifischen Modellen zur Klassifizierung, oder wie im unüberwachten Lernen, ohne trainiert werden. Im Unterschied zu anderen Lernalgorithmen werden die Ergebnisse des Netzes im Anschluss ausgewertet. Abhängig vom Programmierer festgelegten Parameter wird das Netz belohnt oder bestraft. Das Netz versucht dann ähnlich wie ein Mensch den erwarteten Belohnungswert über die Zeit zu maximieren.

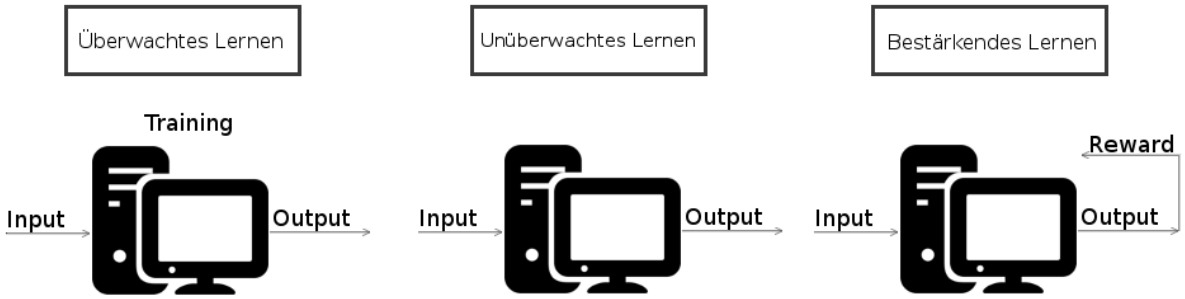

Abbildung 3: Machine Learning Typen

2.5 Deep Learning

Ein weiterer Begriff, der oft in Zusammenhang mit KI genannt wird, ist das Deep Learning. Deep Learning ist ein Teilgebiet des Machine Learning und befasst sich mit der Methode der Informationsverarbeitung. Deep Learning ist inspiriert durch das Lernen des menschlichen Gehirns und nutzt wie im Gehirn neuronale Netze. Das neuronale Netz kann vorhandene Information vom Erlernten mit neuerer Information verbinden und kontinuierlich weiterlernen. Deep Learning eignet sich vor allem bei besonders großen Mengen an klassifizierten Daten, aber benötigen dementsprechend deutlich mehr Rechenleistung. Das neuronale Netz besteht aus drei verschiedenen Typen von künstlichen Neuronen. Eingangsneuronen dienen zur Aufnahme von Daten, während Ausgangsneuronen das Ergebnis liefern. Zwischen den Eingangsneuronen und Ausgangsneuronen befinden sich Schichten sogenannte Hidden Layers, die aus Zwischenneuronen bestehen. Durch das Lernen verbinden sich Eingangsneuronen über unterschiedliche Zwischenneuronen mit Ausgangsneuronen. Je mehr Schichten existieren, desto komplexer können Sachverhalte dargestellt werden. Außerdem steigt mit der “Tiefe” (Anzahl an Schichten) des Netzes seine Produktivität. Deep Learning Netze verbessern sich häufig indem mehr Daten bereitgestellt werden. Im Vergleich zu vielen anderen Ansätzen des Machine Learnings, die sich in der Regel auf deutlich geringere Mengen an Schichten konzentrieren, arbeiten Deep Learning Netze mit teilweise hunderten von komplexen Schichten.

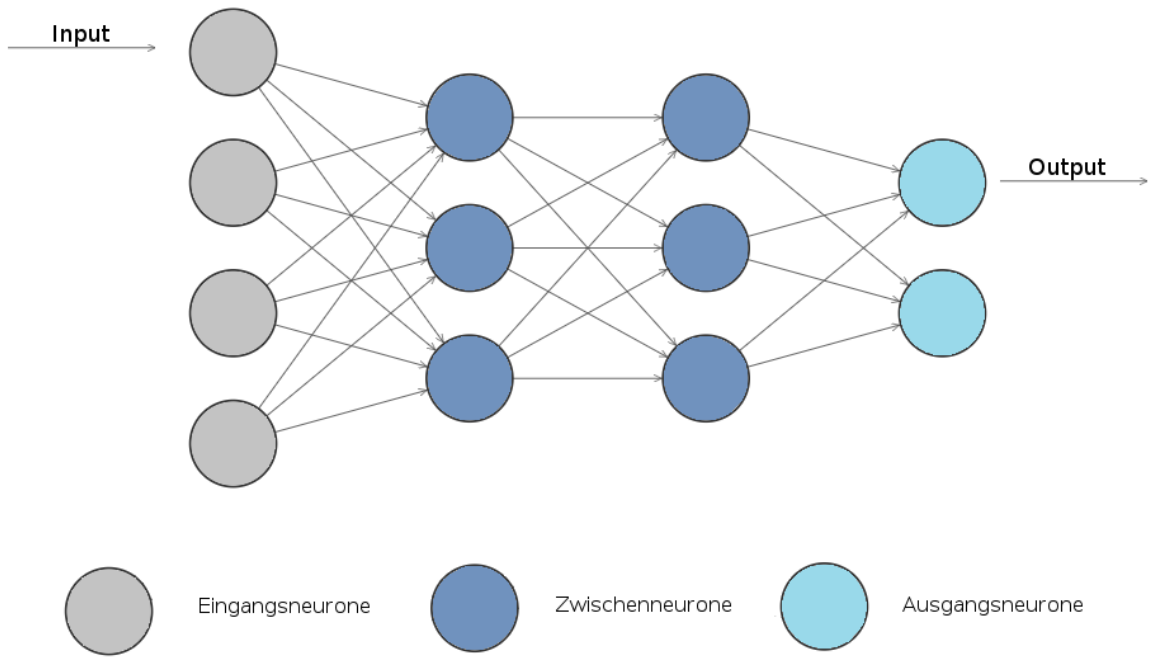

Abbildung 4: Neuronales Netz

Die einzelnen Schichten liefern unterschiedliche Informationen aus und arbeiten an verschiedenen Aufgaben. Bei einer Gesichtserkennung würde in einer Schicht die Nase des Menschen erkannt werden, in der nächsten Schicht werden dann die Augen erkannt. Da nicht definiert wird was die Klassen der einzelnen Merkmale sind, lernt das neuronale Netz hierbei selbst, in welche Klassen es die Daten aufteilen kann. Welche Daten an welche Neuronen weitergegeben werden entscheidet das neuronale Netz

selbst durch die Gewichtungen der Verbindungen zwischen den Neuronen. Im Laufe des Trainings werden diese Gewichtungen angepasst. Die einzelnen Neuronen besitzen zwei Funktionen. Eine Funktion dient der Zusammenfassung der gelieferten Informationen aus den anderen Neuronen. Die Zusammenfassung der Information wird daraufhin an die zweite Funktion geliefert, die sobald ein bestimmter Wert erreicht wird, ausgeführt wird. Dies führt dazu, dass das Neuron nur aktiviert wird, wenn die gelieferten Daten wichtig sind. Sollte das Neuron aktiviert werden, wird mittels der zweiten Funktion die Daten transformiert, inwiefern sich die Daten verändern hängt von der gewählten Transformationsfunktion ab. Die transformierten Daten werden dann an die verbundenen Neurone weitergeleitet.

2.6 Stand der KI-Bibliotheken

Mit der Entwicklung und steigenden Popularität der KI in den letzten Jahren, erscheinen vermehrt neue Open Source Frameworks und Bibliotheken, die kostenlos genutzt werden können. Es wird eine Vielzahl von Programmiersprachen unterstützt. Die meist genutzten Programmiersprachen sind Python, R, Lisp, Prolog und Java. Viele dieser Frameworks und Bibliotheken bieten trainierte Modelle oder eine Visualisierung des neuronalen Netzes an. Außerdem sind einige auf verschiedene Plattformen angepasst z.B. Android, Linux, Mac OS. Bibliotheken wie OpenAIs Gym Retro besitzen einen integrierten Emulator für alte Konsolenspiele. Die populärsten Frameworks und Bibliotheken sind unter anderem Googles TensorFlow, Microsofts CNTK, Theano, Caffe, Torch und Keras. Für die meisten dieser Frameworks und Bibliotheken sind Dokumentationen und Anfänger Tutorials vorhanden. Bibliotheken wie Keras sind in der Lage mit Frameworks wie TensorFlow zusammenzuarbeiten. Keras gibt den Nutzer eine Schnittstelle um auf einfache Weise neuronale Netze zu bearbeiten und zu konfigurieren unabhängig vom Framework. Frameworks wie Lobe, die noch in der Entwicklung sind, sollen den Nutzer die Möglichkeit geben, ohne Quelltext eine KI zu programmieren. Das entwickeln der KI soll hierbei mithilfe eines Baukastensystems entstehen.

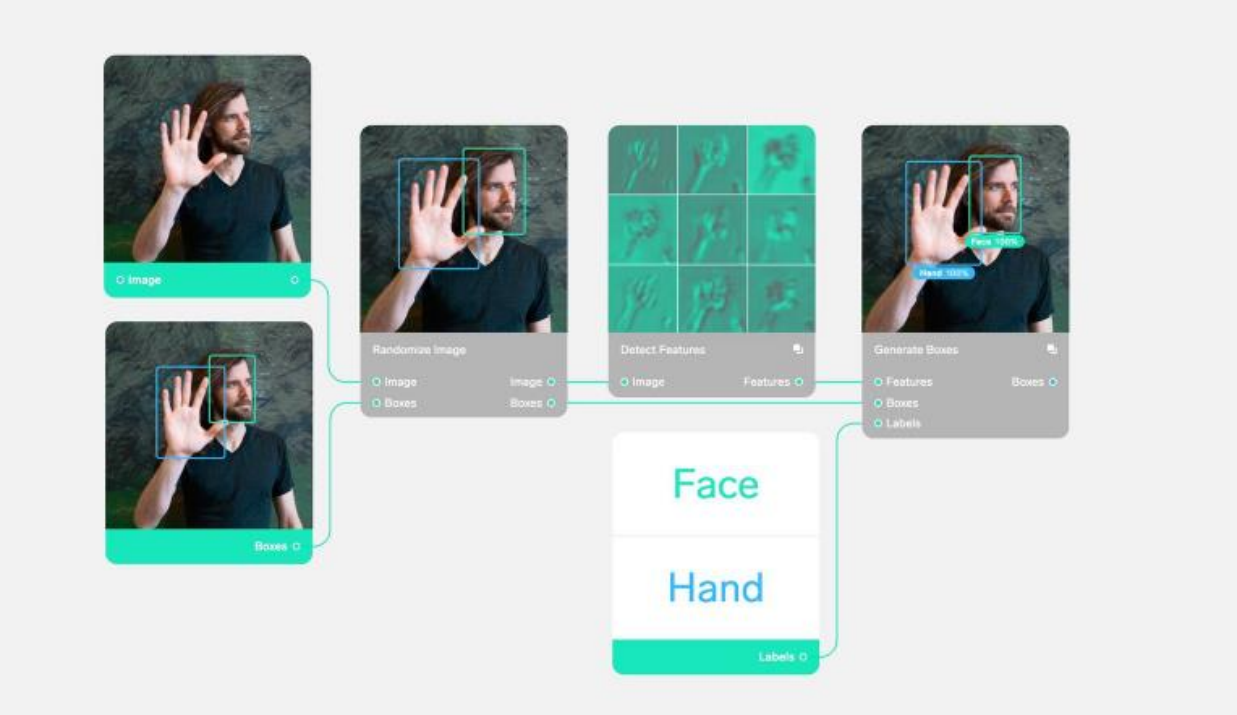

Abbildung 5: Lobe User Interface

3 Konzept einer KI für Super Mario Bros

3.1 Super Mario Bros

Um den aktuellen Stand der Bibliotheken zu betrachten wird eine KI programmiert, die das Videospiel Super Mario Bros spielen soll. Super Mario Bros ist ein Jump-'n'-Run Spiel, welches für das Nintendo Entertainment System (NES) am 13. September 1985 erschien.
In Super Mario Bros übernimmt der Spieler die Rolle des Hauptcharakters Mario und soll die Prinzessin retten. Dafür muss er eine Vielzahl von Leveln bezwingen, die mit Gegnern und Hindernissen gefüllt sind. Zudem wird ihm das Spiel durch die begrenzte Anzahl an Leben und Zeit erschwert, die er pro Level besitzt. Mithilfe von Gegenständen innerhalb der Level ist der Spieler jedoch in der Lage seine Anzahl an Leben zu erhöhen oder die Fähigkeit zu erlangen Feuerbälle zu werfen. Die Welten in die Super Mario Bros aufgeteilt sind, unterscheiden sich in der Auswahl an Gegnern und Hindernissen, aber auch in der Art wie der Spieler Mario steuert.
In den normalen Leveln bewegt sich Mario mit Hilfe des Steuerkreuzes, springt mit der A-Taste und rennt mit der B-Taste. Sobald er sich im Wasserlevel befindet, kann er nicht mehr laufen und schwimmt durch mehrfaches betätigen der A-Taste.

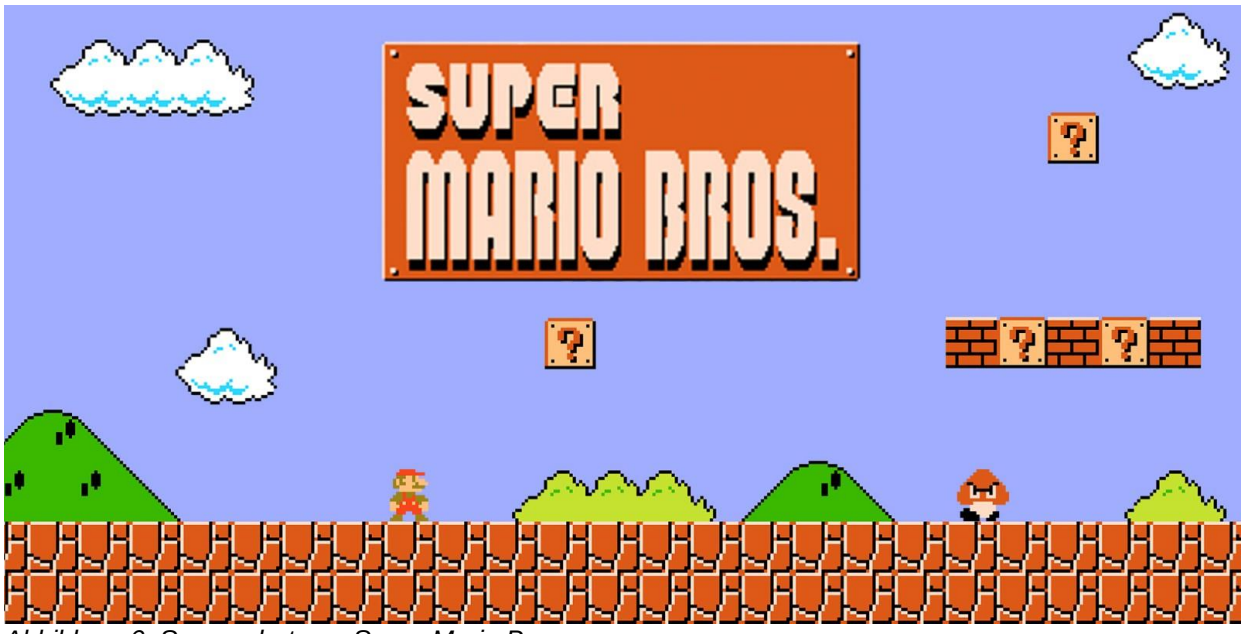

Abbildung 6: Screenshot von Super Mario Bros

3.2 OpenAI Gym Retro

Die Entwicklung der KI geschieht mittels der OpenAI Gym Retro Bibliothek. Gym Retro ist eine Erweiterung der OpenAI Gym Bibliothek, die als Plattform dient um Reinforcement Learning (RL) Algorithmen zu entwickeln, sie zu trainieren und zu vergleichen. Gym Retro selbst spezialisiert sich auf die Anwendung der RL Algorithmen auf Videospiele. Dementsprechend enthält Gym Retro eine Vielzahl an Emulatoren für klassische Videospielkonsolen wie die NES oder die Genesis. Gym Retro erlaubt es dem Nutzer sich die Spiele zu rendern, um das Training der KI zu betrachten. Gym Retro trennt außerdem die Spiele auf ihre einzelnen Level und erlaubt es Simulationen in bestimmten Leveln zu starten. Neben den getrennten Leveln sind für jedes Spiel gewisse Werte definiert, die aus dem Speicher des Spiels gelesen werden und für den Punktestand oder der Anzahl an Leben stehen.

Das optionale Integration Tool für Gym Retro bietet eine Benutzeroberfläche, die den Nutzer eine Vielzahl von Werkzeugen für die KI bietet. Mit Hilfe des Tools lassen sich eigene Szenarien für das Training der KI im Spiel erstellen. Außerdem kann der Nutzer in der Benutzeroberfläche spezifische Werte im Spiel suchen und sich die jeweilige Aufrufstelle aus dem Speicher auslesen. Somit ist es möglich Zustandsänderung wie das Beenden eines Levels zu finden und die KI für das Erreichen dieser zu belohnen. Das Tool visualisiert dem Nutzer die Eingaben der KI und kann, wenn nötig benutzerdefinierte Eingaben als Trainingsdaten der KI übergeben. Mit der Speicherfunktion lassen sich kompakte Filmdateien speichern, die unter anderem die Eingabesequenz des Durchlaufs enthält.

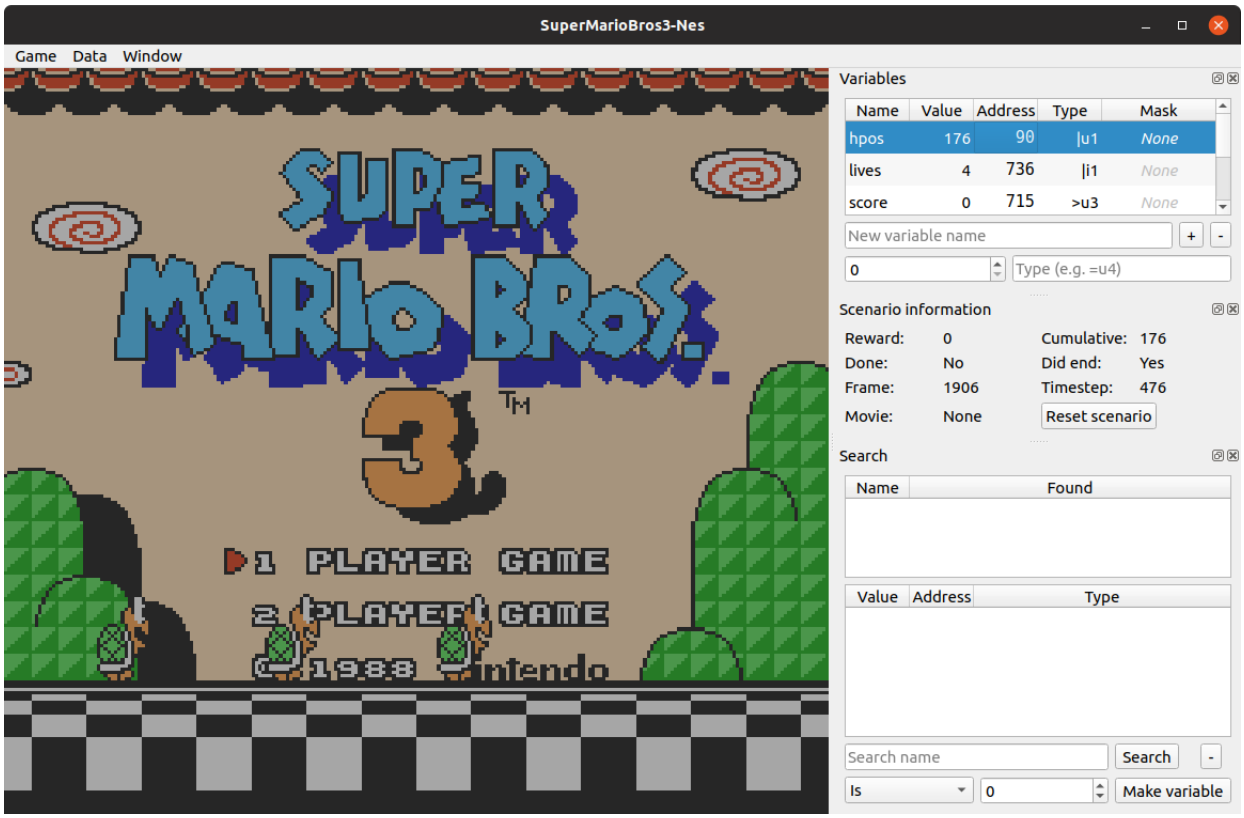

Abbildung 7: Integration Tool von OpenAI

3.4 Anforderung an die KI

Die KI soll das Ende des ersten Levels erreichen und dafür nur ein Leben benötigen.
Der Algorithmus, der für den Lernprozess genutzt wird, heißt "Neuroevolution of Augmenting Topologies"(NEAT). Die KI endet einen Durchlauf, wenn die Zeit ausläuft oder sie ein Leben verliert. Die KI beendet die komplette Simulation, sobald sie das Ende des Levels erfolgreich erreicht hat. Sofern dies geschieht, soll das trainierte Modell abgespeichert werden. Die KI wird für ihre zurückgelegte Distanz belohnt. Jedoch bekommt sie keine Belohnung für das Aufsammeln von Gegenständen wie Münzen und Upgrades, da diese nicht relevant sind für das Abschließen des ersten Levels. Werte wie der Punktestand werden ebenfalls ignoriert, weil die KI nur die Aufgabe hat das Level so schnell wie möglich mit Hilfe von nur einem Leben zu beenden.

3.4 Problematik

Es existieren einige Probleme, die für die KI entstehen können. Der NEAT Algorithmus, der für das Training der KI genutzt wird, ist ein alter Lernalgorithmus aus dem Jahr 2002. Es ist bekannt, dass in Spielen wie Sonic the Hedgehog für die Genesis Levelabschnitte vorhanden sind, die für die KI mittels NEAT nicht immer lösbar waren. Super Mario Bros besitz wie die meisten Spiele Fehler. Einige dieser Fehler im Spiel führen dazu, dass der Spieler das Spiel nicht beenden kann. Es besteht zum Beispiel die Möglichkeit das Ziel zu überspringen oder der Spieler könnte ohne jeglichen Kontakt mit dem Gegner sterben.
Während zufällige Tode keine große Relevanz für das Training der KI hat, ist der Spielfehler das Ziel zu überspringen ein Problem, das dazu führen könnte, dass die KI das Spiel nicht beenden wird. Sobald der Charakter das Ziel überspringt, kann er immer weiter nach rechts laufen. Dadurch sammelt die KI immer mehr Belohnungen, bis die Zeit abläuft oder er die vorgegebene maximale Menge an Belohnungen erreicht hat und somit das Level als abgeschlossen betrachtet. Der Spielfehler selbst tritt sehr selten auf und lässt sich umgehen, indem die Scrolldistanz statt der X-Position des Charakters genutzt wird. Denn während des Fehlers bewegt sich der Charakter außerhalb der maximalen Scrolldistanz.

4 KI und Videospiele

4.1 Entwicklung der KI in Videospielen

In der Vergangenheit waren Begleiter und Gegner in Videospielen fest programmiert. Sie besaßen vorgegebene Reaktionen und Aktionen. Dadurch waren sie vorhersehbar und in der Regel keine Herausforderung für den Spieler. Heutzutage nutzen Entwickler KIs, um dem Spieler eine glaubhafte und immersive Welt zu bieten. Nicht spielbare Charaktere (NPC) sollen mit Hilfe von KIs ein intelligentes Verhalten simulieren und handeln wie es ein menschlicher Spieler tun würde. Das Verhalten der Begleiter soll flexibler sein und sich dem menschlichen Spieler oder der Umgebung anpassen. Gegner hingegen entwickeln Strategien gegen den Spieler und heben somit den Schwierigkeitsgrad an, womit die Welt realistischer wirkt. KIs werden allerdings nicht nur für das Verhalten der NPCs genutzt, sondern auch um die Spielwelt und den grafischen Aspekt zu verbessern. Lebensnahe Animationen und Gefühlsausdrücke werden mit KIs generiert (z.B das Training einer KI durch Motion Capture Daten). Unter Anderem werden KIs genutzt, um alte Spiele komplett grafisch zu verbessern. Oftmals werden Videospiele heutzutage aufgrund ihrer komplexen Umgebungen und Mechaniken als Testplattformen für diverse KI-Systeme genutzt. Ein Beispiel ist der fiktive Bundesstaat San Andreas aus der Spielreihe Grand Theft Auto wird als Testumgebung genutzt, um ein neuronales Netz auf autonomes Fahren zu trainieren.

Abbildung 8: Sicht der KI in Grand Theft Auto

4.2 KI als Spieler

Wie bereits beschrieben werden KIs oft in Videospielen trainiert, hierbei übernehmen sie die Rolle des Spielers. Dabei ist die Funktion der KI abhängig von der Art des Spiels. Videospiele werden schon seit Jahren wegen ihrer Mechanismen und Technologien als Grundlage für die Entwicklung von komplexen Simulationen eingesetzt. Diese Simulationen werden für Trainings- und Experimentierzwecken in der Industrie verwendet. Spiele, die reale Aufgaben darstellen (z.B Flugsimulatoren), werden oft als Trainingsumgebung für KI-Systeme genutzt, welche auf die echte Welt übertragen werden sollen. In den meisten Fällen werden KIs jedoch genutzt, um Strategien für die effektivste Handlung oder den schnellsten Weg zu beobachten und um vorhandene Strategien zu optimieren.
2012 schafften es zwei KI gesteuerte Bots den Games Turing Test für das Spiel Unreal Tournament zu bestehen. Der Games Turing Test ist eine Variante des Turing Test in dem Experten herausfinden sollen, ob die Spieler menschlich oder KI gesteuert sind. Ist es den Experten nicht möglich diese zu unterscheiden, gilt der Test als bestanden. Es gibt zwei unterschiedliche Arten von Spielen, in der die KI die Rolle des Spielers übernehmen kann. Die KI spielt allein gegen programmierte Bots oder sie wird gegen andere reale oder KI gesteuerte Spieler gestellt.

4.2.1 KI gegen Singleplayer Spiele

Spiele wie Super Mario Bros sind sogenannte Singleplayer Spiele. In Singleplayer Spielen existiert nur ein gesteuerter Spieler. Alle anderen Gegner und Begleiter sind über fest programmierte Bots gesteuert. Aufgrund dessen besitzen Singleplayer Spiele relativ konstante Umgebungen und bieten je nach Spiel simple bis komplexe Mechaniken. Neben KI-Systemen, die für die Anwendung in der echten Welt oder für die Forschung und Entwicklung von Algorithmen trainiert werden, werden KI-Systeme auch häufig in Singleplayer Spielen für die Optimierung sogenannter Speedruns genutzt. Als Speedruns werden Spieldurchläufe beschrieben, in dem der Spieler innerhalb der schnellstmöglichen Zeit das Spiel beendet. Es existieren Speedruns in dem der Spieler mithilfe von Fehlern im Spiel Mechaniken und Abschnitte umgeht und somit schnelle Zeiten erreicht, aber auch Durchläufe wo die Nutzung dieser Fehler untersagt ist. Optimierungen und Strategien, die von der KI entwickelt werden, sind jedoch zu kompliziert für menschliche Spieler, um sie effektiv einzusetzen. Viele dieser Optimierungen und Strategien fordern nämlich Pixel genaue Positionierungen und Eingaben in extrem kleinen Zeitfenstern. Dementsprechend sind menschliche Spieler derzeit nicht in der Lage KIs in Speedruns ohne die Nutzung von Programmierfehlern zu besiegen.

4.2.2 KI gegen reale Spieler

Im Vergleich zu Singleplayer Spielen besitzen Multiplayer Spiele mehr als nur einen Spieler gesteuerten Charakter. Hier kann die KI die Rolle eines Mitspielers, Gegenspielers oder sogar komplett die menschlichen Spieler ersetzen. Eine der bekanntesten KIs ist IBMs Schachcomputer DeepBlue, der vor 20 Jahren den damaligen Schachweltmeister Gary Kasparov geschlagen hat. Spiele heutzutage werden jedoch immer komplexer und benötigen Strategien, die auf ein Zusammenspiel der Teamspieler basieren. Für KIs bilden diese Art von Spielen sehr herausfordernde Trainingsumgebungen. Grund hierfür sind komplexere Mechaniken, mehr Möglichkeiten für Aktionen und eine Umgebung mit weniger Konstanten. Ein weiterer Faktor ist die Menge an steuerbaren Charakteren, welche je nach Spiel von zwei bis teilweise 128 Spielern variieren kann. Eins der großen Projekte für KIs in solchen Spielen ist im Jahr 2018 für das Echtzeit Strategiespiel Starcraft 2 entstanden. In Starcraft werden die Spieler als eine von sieben spielbaren Fraktionen in eine begrenzte Welt gesetzt. Der Spieler hat die Aufgabe andere Spieler auf der Welt zu finden und ihre Basis einzunehmen. Um dieses Ziel zu erreichen, gibt er seinen Arbeitern die Aufgabe Ressourcen zu sammeln oder Gebäude zu verbessern, um neue Einheiten zu erzeugen. Dabei ist es relevant, welche Fraktion gewählt wird, denn jede Fraktion besitzt andere Einheiten und eignet sich für unterschiedliche Strategien.

Googles Tochterfirma DeepMind hat in Zusammenarbeit mit Spiele Entwickler Blizzard Entertainment eine KI entwickelt, die gegen professionelle Spieler antreten soll. Die KI wurde hierfür mit Daten aus gespielten Matches trainiert. Die KI ist nicht imstande gewesen professionelle Spieler in vereinfachten Spielen zu bezwingen. Ende 2018 hat DeepMind ihre verbesserte KI namens AlphaStar vorgestellt. AlphaStar spielt Starcraft 2 nicht mehr wie zuvor in vereinfachter Form (z.B. eine vordefinierte Welt oder limitierte Auswahl an Fraktionen für Spieler und KI), sondern bestreitet Spiele unter Regeln und Einstellungen der professionellen Turniere.
AlphaStar benötigt für die Spiele gegen professionelle Spieler nur 290 Aktionen pro Minute (APM). Dies entspricht etwas weniger als der durchschnittlichen APM von professionellen Spielern von etwa 300 - 700 APM. Was die KI aber von Spielern unterscheidet, ist die Sicht, welche sie besitzt. Die KI kann die gesamte bisher erkundete Welt auf einmal betrachten. Der Spieler ist andererseits auf ein kleines Fenster begrenzt, was er bewegen muss, um die Welt zu sehen. AlphaStar blieb ungeschlagen gegen professionelle Spieler, bis dieselbe Restriktion auf die KI übertragen wurde, womit es den professionellen Spielern ermöglicht wurde AlphaStar zu besiegen.

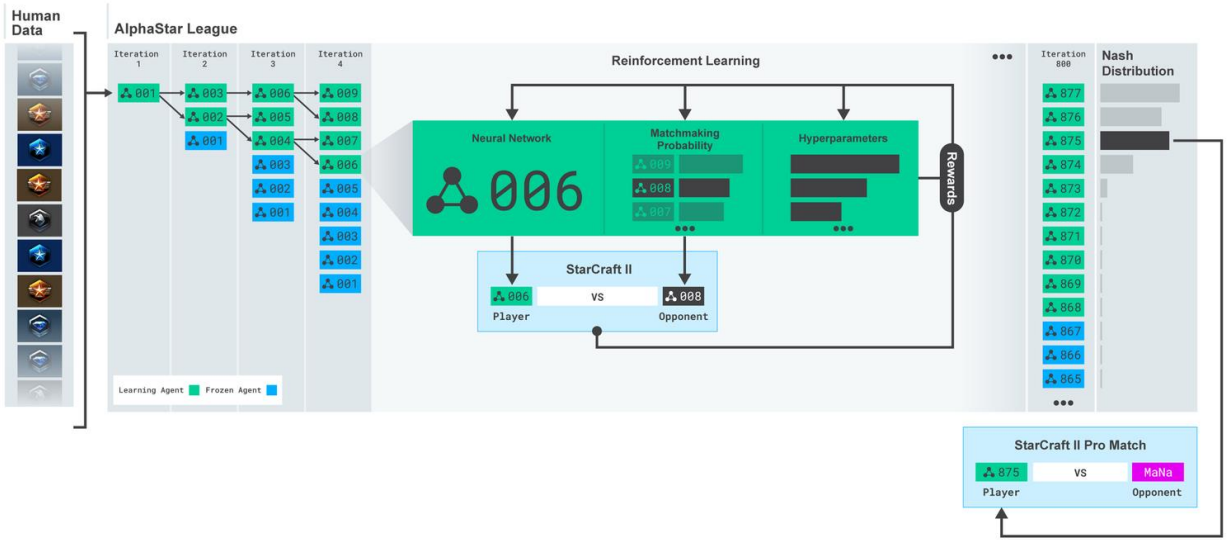

Abbildung 9: Trainingsschema von AlphaStar

Das Training von AlphaStar teilt sich in zwei Phasen auf. In der ersten Phase wird das neuronale Netz über Supervised Learning (überwachtes Lernen) mithilfe von Spieldaten von Blizzard trainiert. Die resultierenden Modelle (Agents) werden im zweiten Schritt mittels Reinforcement Learning weiter trainiert. Hierfür spielen die Agents gegeneinander und aus den resultierenden Ergebnissen werden aus den Agents neue Agents erzeugt, die dann erneut einander gegenübergestellt werden. So entwickeln sich in den 14 Tagen Training über eine Spielzeit von insgesamt 200 Jahren eine Vielzahl von Agents mit verschiedenen Strategien. Die Strategien entwickeln sich von simplen Angriffen auf die gegnerische Basis bis zur nachhaltigen Störung der gegnerischen Wirtschaft durch gezielte Angriffe auf Ressourcenpunkte und Arbeiter.

4.2.3 OpenAI Five

Im OpenAI Forschungsprojekt OpenAI Five von Elon Musk wird ähnlich wie bei AlphaStar eine KI entwickelt, die ein Videospiel spielen soll. Der Unterschied zu AlphaStar ist jedoch, dass das OpenAI Five nicht nur ein neuronales Netz ist. OpenAI Five ist ein Team aus fünf künstlichen neuronalen Netzen für das Multiplayer Online Battle Arena (MOBA) Spiel Dota 2. Ähnlich wie bei Starcraft 2 besteht das Ziel darin die gegnerische Basis zu zerstören. Im Unterschied zu Starcraft 2 werden in Dota 2 die Partien in 5 gegen 5 Teams ausgetragen und somit muss die KI die Fähigkeit aufweisen in einem Team zusammenarbeiten zu können. Jeder Spieler hat dabei eine Auswahl von über 100 verschiedenen Helden mit unterschiedlichen Spielstylen.

Abbildung 10: Screenshot von Dota 2 ohne UI

Die Komplexität von Dota 2 spiegelt sich unter anderem in der Menge an möglichen Aktionen ab. In Schach ist die durchschnittliche Menge an möglichen Aktionen 35, in Go sind es sogar 250. Die KI besitzt in Dota 2 pro Helden etwa 170.000 verschiedene Aktionen, wovon im Durchschnitt 1000 pro Bild ausführbar sind. Die KI selbst beschränkt sich auf die Sicht, die auch ein Spieler sehen würde und erhält hierfür etwa 20000 Zahlen. Im Vergleich zu Schach erhält die KI daher nicht die gesamte Information des Spielfeldes.
Die Ausgabe findet in Form von 8 Zahlen statt, die verschiedene Aktionen der KI ins Spiel übertragen. Dabei richten sich die Aktionen nach der Strategie, welche die größte Belohnung erbringt. Die Belohnungen sind in zwei Kategorien unterteilt, die individuelle Belohnung der KI und die Belohnung als Team. Da keine festen Kommunikationskanäle zwischen den KIs existieren, wird das Teamwork über einen Parameter gesteuert. Der Parameter “team spirit” hat die Werte von 0 bis 1 und soll jeder einzelnen KI übermitteln, welcher der beiden Belohnungswerte zu dem Zeitpunkt Priorität besitzt.

Die Strategien die OpenAI Five nutzt, um ihre Belohnungen zu maximieren, sind komplett ohne Hilfe von menschlichen Trainingsdaten entstanden und ähneln den Strategien, die von professionellen Spielern genutzt werden. Das Training der KI geschieht hierbei mit der Nutzung des RL Proximal Policy Optimization (PPO) Algorithmus. Aufgrund der komplexen Mechaniken von Dota 2 trainieren die KIs anfangs ohne bestimmte Spielmechaniken und mit einer begrenzten Auswahl von Helden in 1 gegen 1 Partien mit sich selbst. Die KI wird dann in 5 gegen 5 Partien mit anderen KIs trainiert. Mit zunehmender Entwicklung und Training ist OpenAI Five mittlerweile fähig professionelle Spieler in 1v1 und 5v5 Spielen erfolgreich zu schlagen. Außerdem werden immer weniger Mechaniken gesperrt und die Auswahl an Helden größer. Das Forschungsteam war zunächst skeptisch, ob die KI mit ihren Algorithmen es schafft Entscheidungen zu treffen, die positive langzeitige Auswirkung auf den Spielverlauf haben. Sie waren der Meinung das für solche Entscheidungen ein hierarchischer RL Algorithmus benötigt wird. In einem hierarchischen RL Algorithmus wird die Aufgabe in immer kleinere Subaufgaben zerlegt, welche von Arbeitern erledigt werden. Diese werden von sogenannten Managern befehligt, welche wiederum von einer höheren Ebene von Managern angeführt werden. Der derzeitige Stand von OpenAI Five hat 45.000 Jahre an Spielzeit angesammelt und wird im Projekt OpenAI Arena darauf trainiert mit menschlichen Spielern als Team zusammenzuarbeiten.

5 Lernalgorithmen

Ein großer Schwerpunkt der KI sind die unterschiedlichen Arten von Lernalgorithmen.
Die verschiedenen Lernalgorithmen unterscheiden sich in der Angehensweise an die dargestellten Probleme. Genetic und RL Algorithmen gehören zu den zwei großen Angehensweisen für komplexe Probleme. RL Algorithmen basieren auf der Lernweise von Menschen und Tieren. Über “Trial and Error” werden Auswirkungen auf das Umfeld von Aktionen und Entscheidungen beobachtet und ausgewertet. Im echten Leben erhalten Menschen unter anderem ein Feedback in Form von zum Beispiel Lob oder soziale Akzeptanz. Die KI hingegen unterscheidet das Feedback deutlich simpler in Form von Belohnungen und Strafen. Genetic Algorithms oder auch evolutionäre Algorithmen basieren nicht auf Lernweisen einer Spezies, sondern basieren auf ihren evolutionären Aspekt. Die Algorithmen arbeiten nach dem Prinzip des “Survival of the Fittest”, dabei überleben nur neuronale Netzwerke deren Mutationen positive Auswirkungen besitzen. Dementsprechend arbeiten beide mit einer unterschiedlichen Anzahl an Netzen. RL besitzt in der Regel ein Netz, was zufällige Aktionen durchführt, um zu lernen. Genetic Algorithms nutzen eine Vielzahl von Netzen, die alle verschiedene Aktionen durchführen. Netze die keine optimalen Aktionen durchführen werden dann verworfen.

5.1 Genetic Algorithms

Genetische Algorithmen arbeiten nach einer bestimmten Folge von Schritten, welche die Evolution eine Spezies spiegeln soll. Der Anfang des Lernprozesses besteht in der Initialisierung von der sogenannten ersten Generation mit zufälligen Parametern. Im Folgenden werden die einzelnen Netze auf ihre Fitness geprüft. Der Fitnesswert beschreibt wieweit das Netz an die Lösung des Problems kommt. Aus der evaluierten Population werden dann die Netze mit der höchsten Fitness für die Erzeugung der nächsten Generation genutzt. Die Netze der nächsten Generation werden dann zufällig einzeln mutiert und wieder auf ihre Fitness geprüft. Die Kombination und Mutation von neuen Netzen wird solange wiederholt, bis der vorgegebene Fitnesswert erreicht ist.

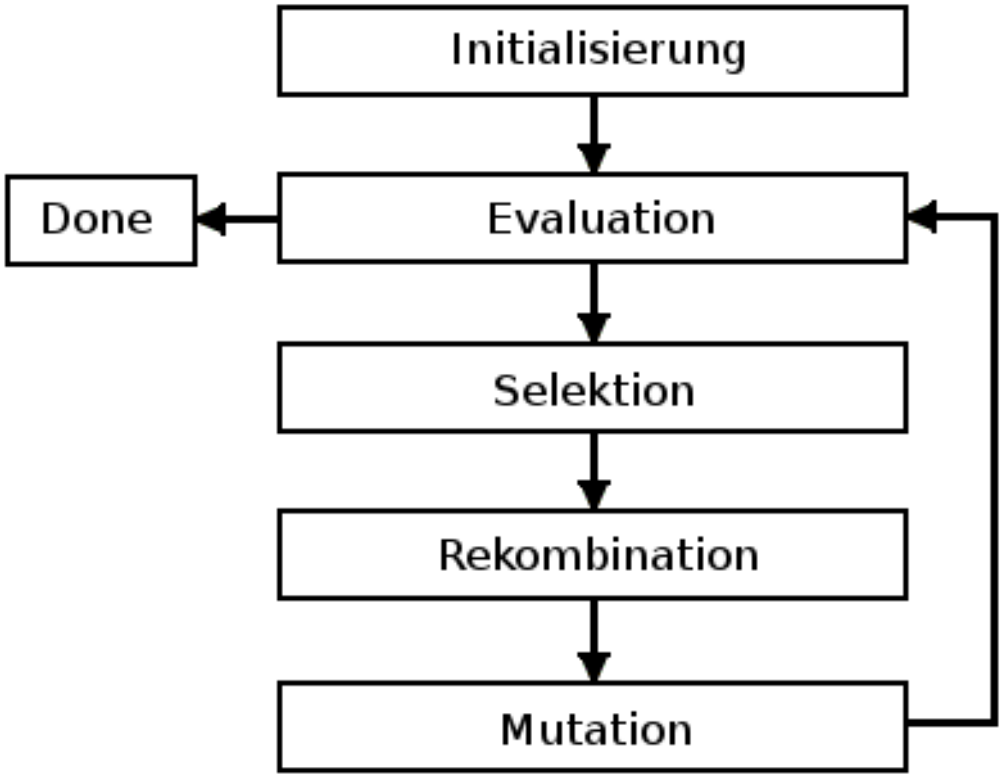

Abbildung 11: Diagramm des Ablaufes von GA Algorithmen

5.1.1 NEAT(NeuroEvolution of Augmenting Topologies)

NEAT ist ein in 2002 von Ken Stanley entwickelter genetischer Algorithmus für neuronale Netze. NEAT optimiert neben den Parametern und Gewichtungen in einem Netz auch die Struktur des Netzes. Im Vergleich zu anderen Algorithmen erzeugt NEAT nicht nur neue Verbindungen zwischen Neuronen, denn NEAT kann neue Neuronen in der Hidden Layer erzeugen. Dementsprechend benötigt man mit NEAT keine vorgegebene Hidden Layer. Die Netze, die aus NEAT resultieren, sind durch die eigenständigen Erweiterungen der Hidden Layer in der Regel nicht komplexer als nötig.
Der generelle Ablauf des Algorithmus gleicht den meisten genetischen Algorithmen. Es beginnt mit der zufälligen Initialisierung der ersten Generation und ihrer Evaluierung. Daraufhin werden wieder aus den fittesten Netzen sogenannte Genome neue Genome kombiniert und mutiert bis der vorgegebene Fitnesswert erreicht ist.

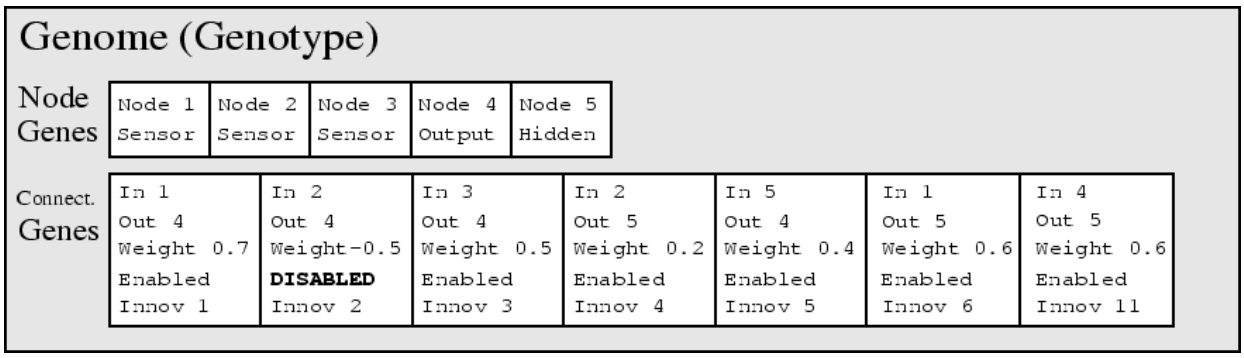

Abbildung 12: Aufbau der einzelnen Gene

Wie in der Abbildung zusehen, ist jedes Genom zusammengesetzt aus einer Anzahl von sogenannten Genen. Die Gene unterscheiden sich in Node Genes und Connect Genes. Node Genes liefern den festen Index und die Funktion der Neurone. Die Connect Genes dagegen setzen sich aus verschiedenen Informationen zusammen. “In” und “Out” bezeichnen die zwei Neuronen, zwischen den die Verbindung ist. “Weight” liefert die Gewichtung der Verbindung. “Enabled” oder “Disabled” zeigt, ob das aktuelle Gen aktiv ist. Die letzte Information, die das Gen liefert, ist die Innovationsnummer, welche für die Art des Gens einzigartig ist. Mit Hilfe der von den Genen gelieferten Informationen lässt sich dann ein neuronales Netz darstellen.

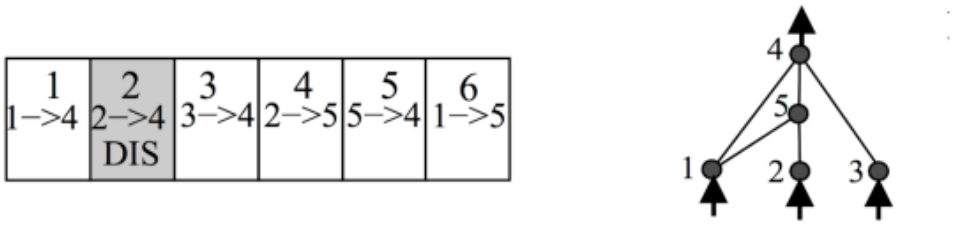

Abbildung 13: Genom und das resultierende neuronale Netz

Wenn zwei unterschiedliche Genome für den Kombinations Schritt gepaart werden, wird zunächst der Aufbau der Connect Genes verglichen. Gene mit derselben Innovationsnummer, die in beiden Connect Genes vorhanden sind, werden Matching Genes genannt. Trotz identischer Innovationsnummer können Gewichtung und der Aktivierungszustand der Gene unterschiedlich sein. Gene, die nur in einem der beiden Genomen vorkommen, besitzen zwei Bezeichnungen Disjoint und Excess Genes. Disjoint Genes bezeichnen Gene, die nur in einem der Gene vorhanden sind. Als Excess Genes werden alle Genpaare bezeichnet, die über der höchsten Innovationsnummer vom Partnergenom liegen.

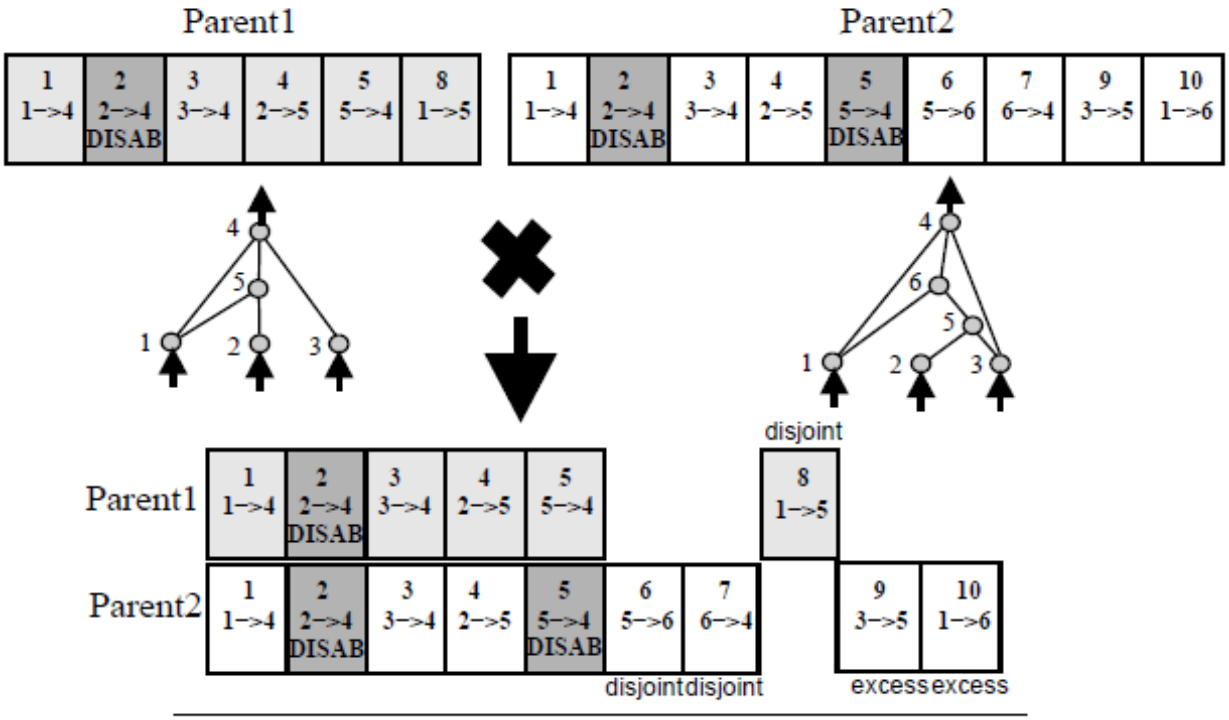

Abbildung 14: Paarung zweier Genome

Abhängig von den Fitnesswerten der Genome werden unterschiedliche Gene vererbt. Wenn beide Genome denselben Fitnesswert besitzen, werden Disjoint und Excess Genes zufällig vererbt. Sollte eins der Genome einen größeren Fitnesswert als sein Partner besitzen, dann werden nur seine Disjoint und Excess Genes weitervererbt. Matching Genes werden in beiden Fällen zufällig von den Genomen vererbt. Inaktive Gene haben bei der Vererbung eine 25 % Wahrscheinlichkeit aktiviert zu werden. Somit besteht die Möglichkeit alte Gene erneut zu optimieren und zu nutzen.

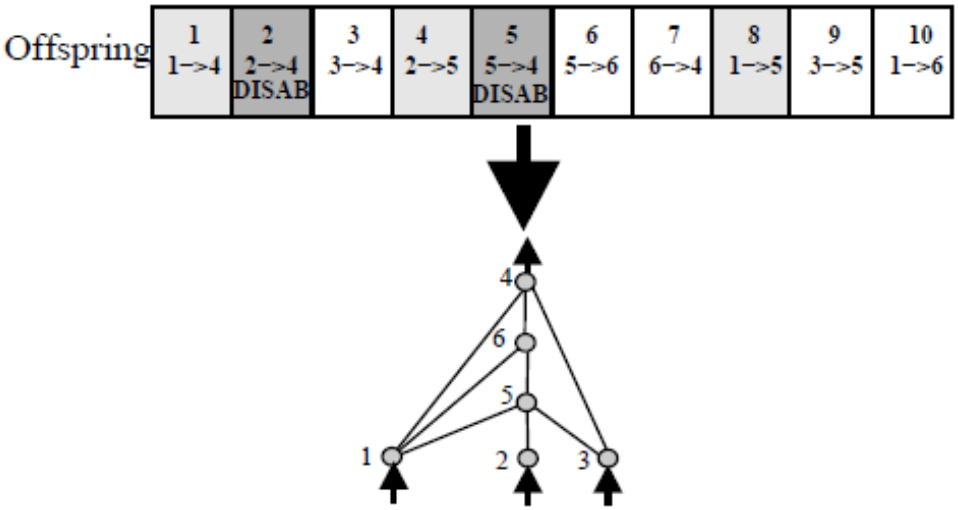

Abbildung 15: Genom aus zwei Genomen mit selber Fitness

Sobald ein neues Genom erzeugt wurde, kommt es zu einer zufälligen Mutation. Die Mutation verändern verschiedene Aspekte des Netzes. Sie können neue Verbindungen zwischen Neuronen erstellen oder komplett neue Neuronen hinzufügen. Andere Mutation verändern Parameter innerhalb der vorhandenen Gene. Gene können dadurch aktiviert oder deaktiviert werden. Die Gewichtung der Verbindungen zwischen zwei Neuronen kann verändert werden, entweder innerhalb eines gewissen Radius oder komplett zufällig.

Mutation	Beschreibung
mutate_link	Erzeugt oder entfernt Verbindung zwischen Neuronen.
mutate_node	Erzeugt oder entfernt Neuronen innerhalb der Hidden Layer.
mutate_weight_shift	Verändert die Gewichtung einer Verbindung anhand eines vorgegebenen Wertes.
mutate_weight_random	Verändert die Gewichtung einer Verbindung auf einen zufälligen Wert.
mutate_enable_disable	Aktiviert oder deaktiviert ein Gen.

Abbildung 16: Mögliche Mutationen in NEAT

Nicht jede Mutation bringt direkt eine Verbesserung und senkt teilweise die Fitness des Genoms, kann aber im Laufe der Zeit zu einer Erhöhung der Fitnesswerte führen. Wenn die Mutation jedoch im nächsten Kombinations Schritt verworfen wird, besteht keine Möglichkeit die möglichen Resultate zu sehen. Vergleichen lässt sich diese Situation mit der Entwicklung des ersten Fahrrads. Das erste Fahrrad war nicht das effektivste Fortbewegungsmittel, doch durch Optimierung von Raddurchmesser oder Sitzhöhe wurde das Fahrrad immer effektiver.
Um zu verhindern, das solche Gene direkt verworfen werden, unterteilt der NEAT Algorithmus mithilfe einer Gleichung alle Genome in eine Spezies. Dadurch müssen Genome nur innerhalb ihrer Spezies konkurrieren und erhalten genug Zeit sich zu optimieren bevor sie gegen eine andere Spezies antreten müssen. Um zu bestimmen, welcher Spezies ein Genom angehört, wird es mit anderen Genomen verglichen. Relevant sind hierbei die Menge der Excess und Disjoint Genes in Abhängigkeit zur größten Menge an Genen eines Genoms. Zusätzlich wird die durchschnittliche Differenz der Gewichtungen in den Matching Genes bestimmt. Jeder der drei Werte wird mithilfe eines Faktors unterschiedlich gewichtet. Der resultierende Wert beschreibt die Distanz zwischen den verglichenen Genomen. Wenn die Distanz klein genug ist, wird das verglichene Genom derselben Spezies beigefügt. Wenn die Distanz zu groß ist, wird das Genom mit weiteren Genomen verglichen. Sofern er keine Spezies findet, die er beigefügt werden kann, wird eine neue Spezies erzeugt. Jedes Genom kann genau zu einer Spezies gehören.

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \cdot \overline{W}.$$

$\delta =$ Distanz zwischen 2 Genomen
$c_1\ c_2\ c_3 =$ Gewichtungen der 3 Faktoren
$E =$ Anzahl der Excess Genes
$D =$ Anzahl der Disjoint Genes
$N =$ Anzahl der Gene vom größeren Genom
$\overline{W} =$ Durchschnittliche Differenz der Gewichtungen in Matching Genes

Abbildung 17: Gleichung zur Bestimmung der Spezies

5.2 Reinforcement Learning

RL ist ein Teil des überwachten Lernens und ähnelt ihm dementsprechend im Ablauf des Trainings. Was jedoch RL vom überwachten Lernen unterscheidet, ist die Unabhängigkeit von menschlichen Trainingsdaten. RL erzeugt eigene Daten, an denen es sein Netz optimiert, dadurch ist aber auch die resultierende Ausgabe undefiniert. Im Videospiel Pong existieren drei mögliche Ausgaben für das Netz, entweder wird der Spieler runter und hoch bewegt mit “Down” und “Up” oder behält seine aktuelle Position. Ein Netz was mit Trainingsdaten trainiert wurde, kann anhand der vorhandenen Daten bestimmen, ob seine Ausgabe korrekt oder falsch ist. Das RL Netz jedoch hat keinen Zugriff auf das Wissen, ob die aktuelle Ausgabe korrekt ist, sondern kann nur anhand eines Feedbacks erkennen, ob die Aktion positive oder negative Auswirkungen hat. Durch das Feedback in Form von Belohnungen und Strafen wird das Netz mit einer von zwei Methoden optimiert. Value based Methoden wie QLearning und Deep QLearning erzeugen eine Belohnungsfunktion, die für jede Aktion zu einem bestimmten Zustand, einen konkreten Wert zuweist. Dadurch wird für jeden Zustand die Aktion gewählt, die den größten möglichen Belohnungswert erzeugt. Die zweite Art der Optimierung sind Policy based Methoden wie PPO. Policy based Methoden passen die Hidden Layer des Netzes an. Hierfür werden Paare gebildet aus Reihen von Aktionen und die resultierenden Zustände. Das Resultierende Netz erzeugt dann Ausgaben, die für den Zustand “korrekt” sind.

5.2.1 PPO(Proximal Policy Optimization)

Die für das Forschungsprojekt OpenAI Five genutzte Methode PPO ist eine sogenannte Policy gradient Methode. PPO wurde entwickelt, um eine Methode zu erzeugen, die einfacher zu implementieren und anzupassen ist, aber auch auf demselben Level oder sogar besser als aktuelle Policy gradient Methoden performt.
Policy gradient Methoden starten mit einem zufälligen Netz, welchem eine Menge an Eingaben geliefert wird. Wenn wir Pong als Beispiel betrachten, erhält das Netz jeden einzelnen Pixel als Eingabe und die möglichen Ausgaben sind “Up”, “Stay“ und “Down”.

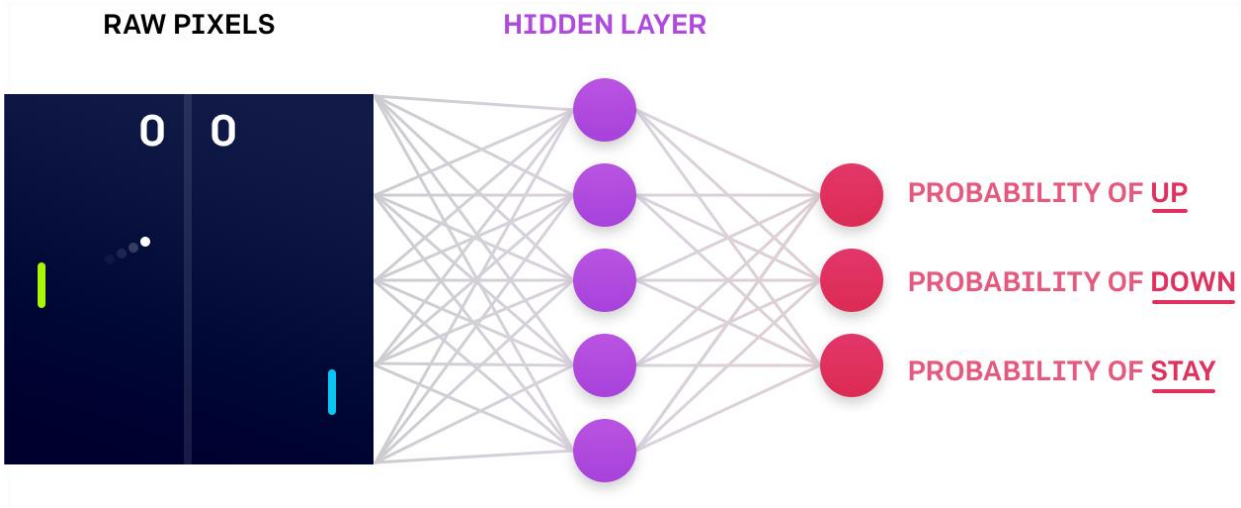

Abbildung 18: Neuronales Netz anhand von Pong

Nach jedem gelieferten Bild der Spielengine verarbeitet das Netz diese in die Ausgaben, welche als Wahrscheinlichkeiten für die möglichen Aktionen dargestellt sind. Da die Ausgaben als Wahrscheinlichkeiten ausgegeben werden, wird nicht immer nur eine bestimmte Aktion durchgeführt und das Netz hat die Fähigkeit sein Umfeld zu erkunden. Die Ausgaben werden der Spielengine übermittelt und führen dann zu einer Veränderung, die wiederum ein neues Bild erzeugt, welches wieder ans Netz übertragen wird.

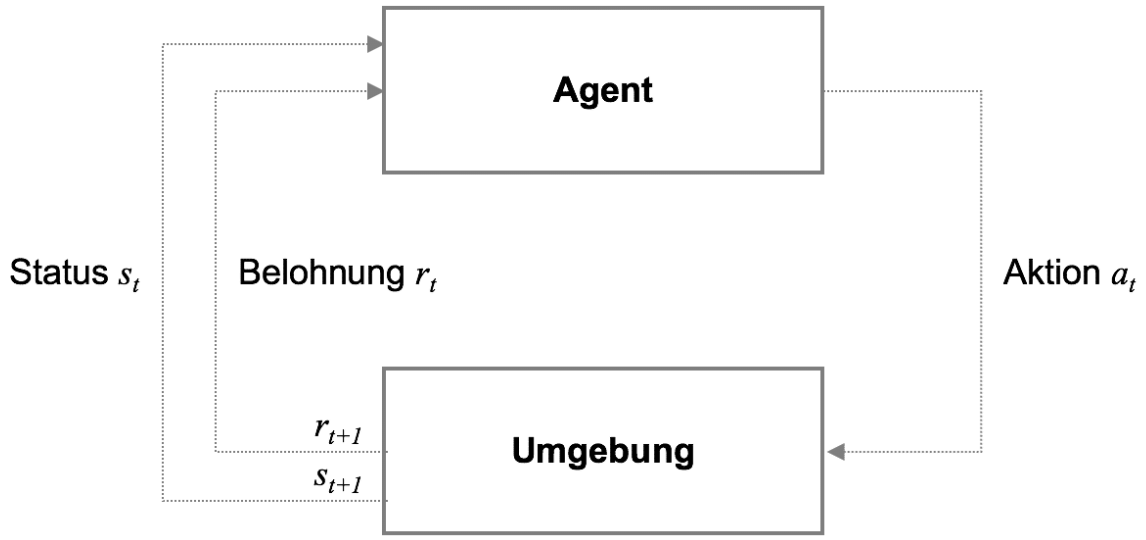

Abbildung 19: Ablauf des Trainings in RL

Anhand des Punktestands in Pong kann dem Netz eine Belohnung oder Strafe je nach Veränderung des Punktestandes zugewiesen werden. Abhängig davon, ob das Netz eine Strafe oder Belohnung erhalten hat, optimiert er seine Hidden Layer auch Policy Network genannt. Bei Pong geschieht die Optimierung nach jeder Episode. Eine Episode bezeichnet die Reihe an Aktionen, die zu einer positiven oder negativen Belohnung führen. Bei einer positiven Belohnung wird die Wahrscheinlichkeit die Reihe an Aktionen durchzuführen mit einem vorgegebenen Gradienten erhöht, sollte die Belohnung negativ sein wird derselbe Gradient mit -1 multipliziert und genutzt, um die Wahrscheinlichkeit für die Reihe an Aktionen zu verringern. Im Laufe des Trainings werden so negative Aktionen ausgefiltert.

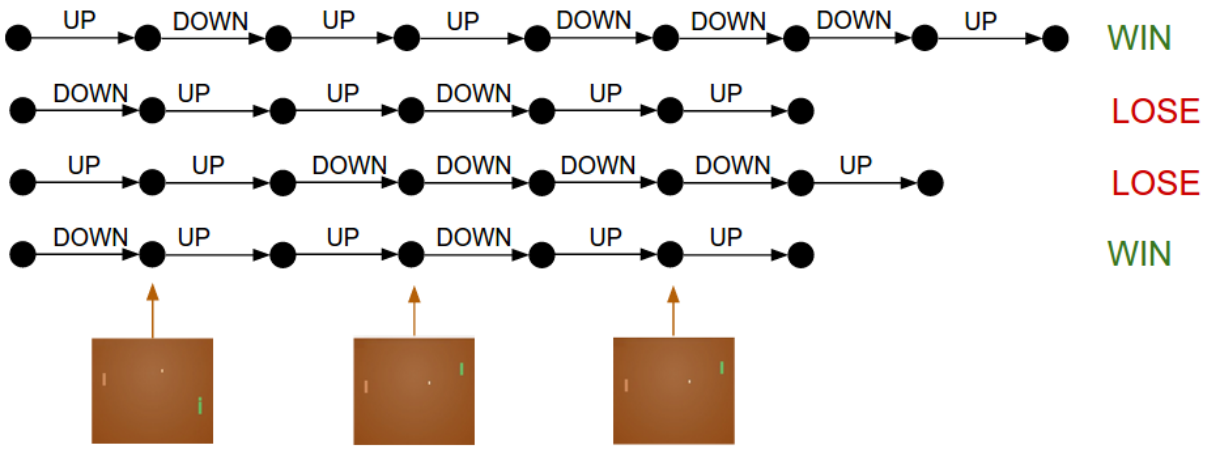

Abbildung 20: Beispiel Reihen für Pong

Das Problem jedoch was entsteht, ist dass wenn die letzte Aktion einer Reihe der einzige Grund für eine negative Belohnung ist, die gesamte Reihe bestraft und die Wahrscheinlichkeit der Aktionen gesenkt wird. Dieses Problem wird auch als Credit Assignment Problem bezeichnet, was aus der Weise entsteht, wie das Netz belohnt wird. Das Netz wird nämlich nur nach jeder Episode belohnt und nicht nach jeder

Aktion, daher muss es selbst lernen, welche Reihe an Aktionen innerhalb der Episode Relevanz für die Belohnung hat. In Pong sind es die Eingaben kurz bevor der Ball den Spieler trifft, die relevant sind, die Aktionen während der Ball wegfliegt sind irrelevant. In Pong müssen aufgrund der simplen Mechaniken keine weiteren Belohnungen außerhalb des Punktestandes definiert werden, um ein Netz zu trainieren.

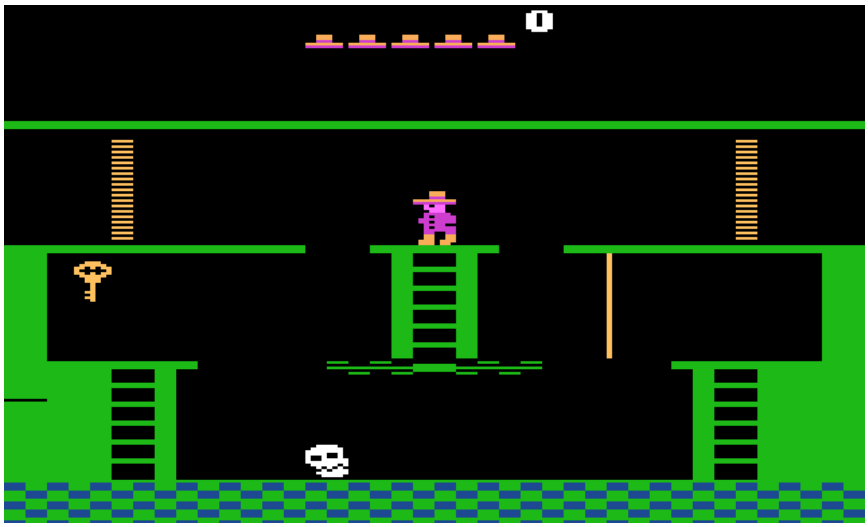

Abbildung 21: Screenshot von Montezuma.

Bei Spielen wie Montezuma's Revenge sind die Mengen an möglichen Aktionen zu groß. Dadurch ist es nahezu unmöglich, dass das Netz durch zufällige Aktionen die bestimmte Reihe an Aktionen durchführt und erhält dementsprechend nie eine positive Belohnung. Das Netz wird nicht in der Lage sein das Spiel zu beenden. Um solche Probleme zu lösen, werden neue Belohnungsfunktionen programmiert. Neben dem Punktestand wird das Netz nun zusätzlich für das Einsammeln des Schlüssels und für das Umgehen der Feinde belohnt. Nun kann das Netz Belohnungen sammeln und seine Policy optimieren. PPO trainiert im Vergleich zu anderen Methoden wie QLearning nur einmal mit den Daten. Sobald der Durchlauf vorbei ist, werden die alten Daten verworfen und neue generiert. Außerdem wird in PPO die Policy in kleinen Schritten optimiert. Dies geschieht ähnlich wie in TRPO (Trust Region Policy Optimization) durch die Berechnung einer KL-Divergenz. Die KL-Divergenz oder auch Kullback-Leibler-Divergenz beschreibt das Maß der Unterschiedlichkeit von zwei Wahrscheinlichkeitsverteilungen. In TRPO wird die KL-Divergenz genutzt, um eine Region zu bestimmen, in der sich die Policy verändern kann. Somit wird verhindert, dass die Policy sich zu stark in eine Richtung verändert. PPO nutzt die KL-Divergenz aber nicht als feste Begrenzung, sondern als Region, in der die Policy sich verändern soll. Sollte die Policy sich außerhalb der Region entwickeln, so wird der Gradient verringert. Sobald der Gradient 0 erreicht, verwirft das Netz die alten Trainingsdaten und trainiert mit neuen Daten.

5.3 Auswahl des Lernalgorithmus

Beide Algorithmen sind in der Lage das Spiel Super Mario Bros zu lösen. Sowohl Genetic als auch RL Algorithmen sind mit Gym Retro nutzbar. RL Algorithmen besitzen den Vorteil, dass OpenAI eine Bibliothek namens Baselines zur Verfügung stellt, die Algorithmen wie PPO und TRPO mit vortrainierten Modellen beinhaltet. Mit Baselines lässt sich Super Mario Bros in Gym Retro mit einer Zeile an Befehlen komplett trainieren. NEAT lässt sich nicht so einfach in Gym Retro integrieren, hat aber den Vorteil einfacher verständlich zu sein, denn PPO was auf TRPO basiert ist mathematisch komplexer als NEAT. NEAT benötigt weniger Hyperparameter und kommt ohne komplexe Belohnungsfunktion aus. Hyperparameter beschreiben Parameter, die vor dem Training des Modells definiert werden. Die Parameter können dabei die Menge an Hidden Layers bestimmen, die das Netz habe besitzt, oder wie viele Genome eine Generation besitzen soll. Da die Implementierung von RL Algorithmen in Gym Retro schon vorhanden sind, wird für Super Mario Bros versucht NEAT zu implementieren und zu nutzen.

6 OpenAI Retro SuperMarioBros(Neat)

6.1 Umsetzung

6.1.1 Software, Umgebung und Bibliotheken

Für die Entwicklung der KI für Super Mario Bros wird eine Vielzahl von Bibliotheken genutzt.
Die grundsätzlichen Bibliotheken, die benötigt werden, sind Neat und OpenAI Gym Retro. Diese beinhalten den Lernalgorithmus und die Trainingsumgebung für die KI. Die genutzte Programmiersprache ist Python. Python gehört zu den beliebtesten Programmiersprachen für KIs und besitzt dementsprechend eine Menge von nutzbaren Bibliotheken, Werkzeugen und Lernkurse. Zudem ist Python durch seine Einfachheit schneller und leichter zu verstehen als andere Programmiersprachen wie C++. Weitere genutzte Bibliotheken sind OpenCV und Pickle. OpenCV ist eine Open Source Bibliothek, die verschiedene Möglichkeiten bietet Bildmaterial zu verarbeiten und analysieren. Für die KI wird OpenCV genutzt, um die erzeugten Bilder des Emulators zu bearbeiten. Pickle erlaubt es in Python jedes Python-Objekt als Datei zu speichern oder falls nötig sie zu öffnen. Mit Pickle kann das trainierte Netz am Ende gespeichert werden. Neben den Bibliotheken werden zusätzlich zwei Programme genutzt. Dazu gehören Komodo Edit ein Texteditor, der für Programmiersprachen wie Python ist und das OpenAI Integration Tool, das für die Suche bestimmter Parameter im Speicher benutzt wird.

6.1.2 Aufbau und Durchführung

Zunächst muss eine Umgebung in Gym Retro erzeugt werden. Innerhalb dieser Umgebung soll das Netz trainiert werden. In Gym Retro setzt sich die Erzeugung dieser Umgebung aus zwei Parametern zusammen. Der erste Parameter übergibt Gym Retro welches Spiel auf welcher Konsole emuliert werden soll. Der zweite Parameter beschreibt den sogenannten State, in dem die Umgebung erzeugt wird. Die vorgegebenen States in Super Mario Bros sind die ersten Levels jeder einzelnen Welt. Der State wird auf die erste Welt gesetzt. Eigene States können mit der Nutzung des Integration Tools erzeugt werden. Gym Retro emuliert nun das erste Level von Super Mario Bros mit der Auflösung von 240 x 224 Pixel. Ähnlich wie im PPO Beispiel von Pong werden dem Netz die einzelnen Pixel als Eingabe geliefert. Die Menge an Eingangsneuronen, die dann benötigt werden, sind 53.760. Um die Menge der Eingangsneuronen zu senken wird mit OpenCV das Bild zunächst in X und Y Richtung durch 8 geteilt. Die daraus resultierende Anzahl an Pixeln beträgt dann 840. Aus dem skalierten Bild werden im nächsten Schritt wieder mit OpenCV die farbigen Pixel in verschieden Grautöne gesetzt. Aus den einzelnen Pixeln wird dann ein eindimensionales Array erzeugt, was dem Netz geliefert wird.

Auflösung: 240x224	Auflösung: 30x28
Pixel: 53.760	Pixel: 840

Abbildung 22: Skalierung des Bildes von Super Mario Bros.

Ein normaler NES Controller besitzt 8 verschiedene Tasten. Die vier Richtungstasten, die Select und Start Tasten, sowie die A und B Tasten. Der emulierte Controller innerhalb von Gym Retro gibt jedoch 9 verschiedene Werte aus. Der letzte Wert beschreibt, um welche Art von Controller es sich handelt. Aus diesem Grund erhält das Netz 9 mögliche Ausgaben. Da in Neat die Hidden Layer nicht definiert werden muss, startet das Netz mit 849 Neuronen.

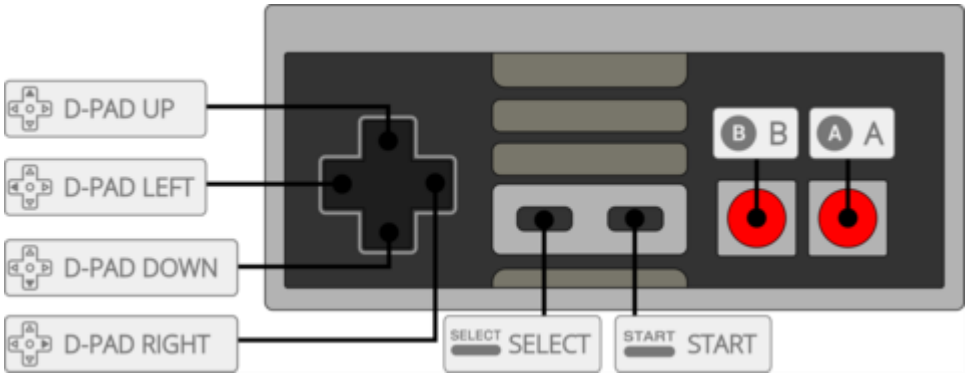

Abbildung 23: NES Controller Tastenlayout

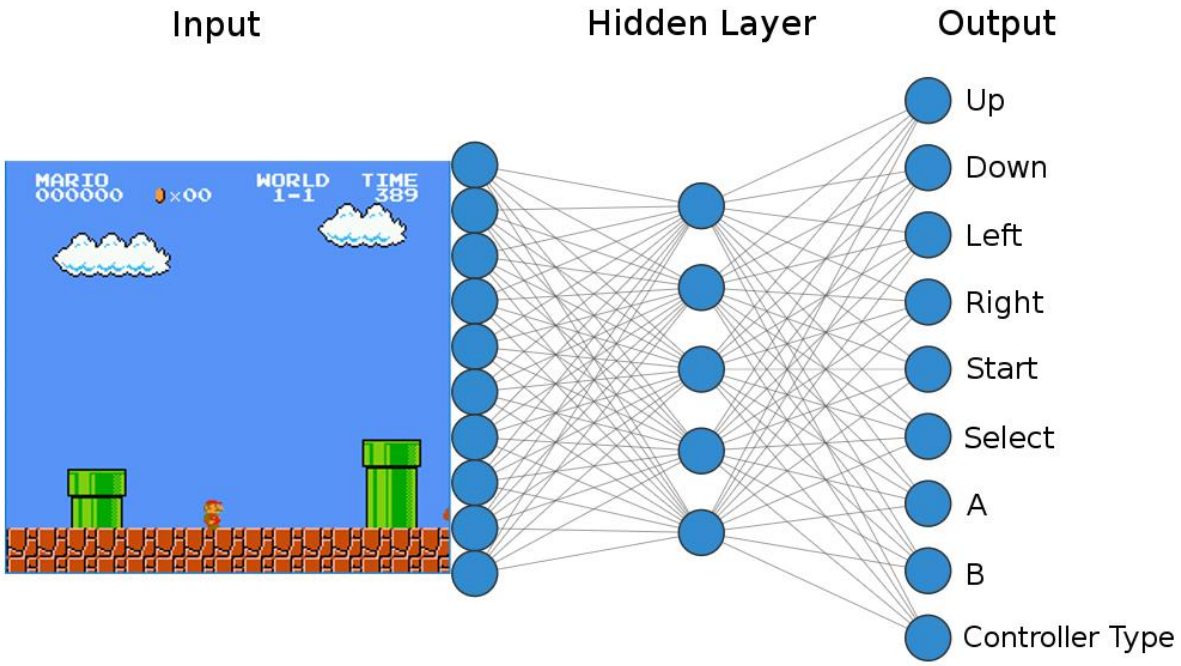

Abbildung 24: Beispielnetz für Super Mario Bros.

Mit diesen Werten lässt sich innerhalb der NEAT Konfigurationsdatei die Parameter anpassen. In der Konfigurationsdatei wird die Größe der Population auf 25 gesetzt und der gesuchte Fitnesswert auf 50.000. Der Fitnesswert wird so gewählt, sodass die KI den Wert nicht erreichen kann, ohne eine vorgeschriebene Bedingung erfüllt zu haben. Außerdem wird der Wert für die maximale Stagnation auf 50 erhöht. Die maximale Stagnation regelt, wie viele Generationen eine Spezies von Genomen existieren darf, ohne einen Fortschritt im Fitnesswert zu erbringen. Die Konfigurationsdatei enthält außerdem noch alle anderen relevanten Parameter für die Entwicklung des Netzes, wie die Mutationswahrscheinlichkeiten und die Mutationsstärken. Die anderen Parameter bleiben zunächst bei den Grundwerten und werden angepasst, sobald die Evolution des Netzes betrachtet wird. Aus der Konfigurationsdatei wird in NEAT die Population erzeugt.

Jedes der Genome wird zufällig generiert und führt zufällige Aktionen aus. Um zu beurteilen, welches Genom sich am besten für das Spielen von Super Mario Bros eignet, wird jedes Genom auf seine Fitness überprüft. Der Fitnesswert ist in Abhängigkeit zu der zurückgelegten Distanz des Charakters. Gym Retro besitzt standardmäßig für Super Mario Bros nur die Adresse für die Scrolldistanz des Bildschirms. Diese lässt sich zwar auch nutzen, ist jedoch ungenau. Um die Position von Mario innerhalb des Levels zu bestimmen, muss mithilfe des Integration Tools oder des Super Mario Bros Disassembly, die X Position von Mario und die X Position des Bildschirmrandes gesucht werden. Um diese in Gym Retro einzubinden, müssen die Speicheradressen, die in Hexadezimal angegeben werden, in Dezimal umgewandelt werden. Neben der Speicheradresse wird ein Typ benötigt, dieser Typ setzt sich aus drei Werten zusammen. Der erste Wert ist der Endian, der die Ordnung der Bytes im Speicher definiert. Bei Little-Endian wird der kleinste Wert zuerst gesetzt, bei Big-Endian der größte. Der zweite Wert beschreibt, wie der Wert im Speicher abgelegt wird. Der dritte Wert liefert die Anzahl an Bytes, die der Wert im Speicher belegt.

Mithilfe der der Position von Mario im Level wird das Genom für die Distanz die Mario zurücklegt belohnt. Hierfür wird für jedes Bild seine aktuelle Position mit der letzten gespeicherten Position verglichen, wenn die neue Position größer als die vorherige ist, erhält das Netz eine Belohnung von +1. Das Genom spielt, solange bis eine der Abbruchkriterien erfüllt ist und ordnet dann den aktuellen Belohnungswert seiner Fitness zu. Das Genom bricht den Durchlauf ab, wenn er ein Leben verloren hat oder 300 Bilder lang keine Belohnung gesammelt hat. Unter diesen Bedingungen wird im Folgenden das nächste Genom in der Spielumgebung getestet. Sobald alle Genome einer Generation das Spiel gespielt haben, werden in der Konsole das aktuell beste Genom und weitere Informationen der Generation ausgedruckt. Die Informationen zeigen die Anzahl an Spezien und ihre Population, sowie den höchsten Fitnesswert der Spezies.

```
Population's average fitness: 441.92000 stdev: 308.82253
Best fitness: 1556.00000 - size: (12, 1907) - species 122 - id 39123
Average adjusted fitness: 0.260
Mean genetic distance 1.906, standard deviation 0.564
Population of 25 members in 2 species:
   ID   age  size  fitness  adj fit  stag
  ====  ===  ====  =======  =======  ====
   122   36    15   1556.0    0.279    34
   123   18    10    596.0    0.240    15
Total extinctions: 14
Generation time: 62.216 sec (56.411 average)

****** Running generation 4434 ******
```

Abbildung 25: Ausgabe der Information einer Generation nach Abschluss einer Generation

Sollte das Genom jedoch einer der Zielkriterien erfüllen, wird sein Belohnungswert um 50.000 erhöht und es erreicht somit das Fitnessziel. Sobald das Fitnessziel erreicht ist, speichert das Programm mit Pickle das Genom, welches das Ziel erreicht hat, ab und das Training wird beendet. Das erste Zielkriterium ist die Veränderung des Levelwertes. Der Levelwert wird aus dem Speicher von Super Mario Bros ausgelesen und beschreibt, in welchem Level sich der Charakter derzeit aufhält. Wenn der Wert größer als 0 ist, bedeutet es, dass der Charakter das nächste Level betreten hat. Das zweite Kriterium betrachtet den Wert der Position von Mario im Level, denn sobald seine Position das Ende des Levels bei 3266 erreicht, soll das Training beendet werden. Da das Levelende bei 3266 liegt, wird das Genom nur in der Lage sein das Fitnessziel zu erreichen, wenn er einer der Zielkriterien erfüllt. Alle 300 Sekunden wird über die NEAT Klasse Checkpointer die aktuelle Generation als Datei gespeichert, diese lässt sich später mit einer Funktion aufrufen. Die Checkpoints erlauben es die Fitnessfunktion zu verändern, ohne das vorhandene Netz von vorne trainieren zu müssen.

Für die Durchführung werden drei verschiedene Populationen trainiert. Alle drei Populationen werden mit derselben Fitnessfunktion belohnt und unterscheiden sich nur in der Konfiguration von den Parametern in NEAT. Die erste Population arbeitet mit den Parametern aus dem oben genannten Aufbau. Die zweite Population behält immer die zwei Spezien mit der höchsten Fitness egal wie lange sie stagniert, dieses Verhalten wird auch als Spezien Elitismus bezeichnet. Die Dritte Population besitzt die doppelte Menge an Genomen und eine erhöhte Wahrscheinlichkeit neue Verbindung und Neuronen zu mutieren. Zudem werden aus jeder Spezies nur noch die zwei besten Genome, statt drei in die nächsten Generationen übernommen.

6.1.3 Beobachtung

Alle drei Populationen hatten einen unterschiedlichen Start und sind daher schwer zu vergleichen. Sie ähneln sich aber in der Geschwindigkeit mit den sich die ersten Generationen entwickeln. Population 1 benötigt für eine Generation im Durchschnitt 60 Sekunden und ist somit 50% schneller als die anderen Populationen. Population 3 ist langsamer aufgrund seiner größeren Population und benötigt daher mehr Zeit, da die Anzahl an Spieldurchläufen höher ist. Population 2 besitzt anfangs dieselbe Geschwindigkeit wie Population 1 und wird im Laufe der Generationen langsamer. Dies passiert durch die Einstellung des Elitismus bei Erreichung der maximalen Stagnierung in NEAT. Population 1 und 3 löschen jede Spezies aus, wenn diese eine Stagnierung von 50 Generation erreicht hat. Daher kann in Population 1 und 3 beobachtet werden wie nach längerer Stagnierung an einem Fitnesswert, die Population neu generiert wird und die Genome wieder bei 0 Fitness starten. Population 2 behält immer die zwei besten Spezies am Leben und besitzt dementsprechend in der Regel einen höheren Durchschnittswert der Fitness. Je größer der Fitnesswert, desto länger spielt das Genom das Spiel, aufgrund dessen benötigt Population 2 die längste Zeit zum Abschließen der Generation. Auch die Geschwindigkeit mit den die 3 Population an Fitness gewinnen unterscheidet sich stark voneinander. Alle 3 Populationen starten zwar mit zufällig erzeugten Genomen, erreichen aber relativ konstant ⅙ des Levels in den ersten 20-30 Generationen. Ab diesem Punkt steigt die Fitness von Population 2 deutlich stärker als Population 1 und 3. Innerhalb der ersten 80 Generationen erreicht Population 2 die Hälfte des ersten Levels. Population 1 und 3 sind vergleichsweise deutlich langsamer. Population 1 erreicht die Hälfte des Levels bei Generation 416 und Population 3 aufgrund seiner stärkeren Mutationswahrscheinlichkeit und größerer Population schon bei Generation 200. Sowohl Population 1 und 3 scheinen nach mehreren hunderten bis tausenden von Generationen nicht über die 80 Prozent des ersten Levels zu kommen. Population 2 hat im Vergleich 90 Prozent des Levels nach 500 Generationen bewältigt, stagniert aber seit hunderten Generationen an dem Punkt. Im Laufe der Durchführung wurde bei allen drei Populationen die Fitnessfunktion angepasst. Die ursprüngliche Fitnessfunktion hat selbe Distanzen mit unterschiedlichen Fitnesswerten bewertet und führte zu einer langsameren Entwicklung der Population.

Name	Population 1	Population 2	Population 3
Pop_size	25	25	50
Species_elitism	0	2	0
Node_delete_prob	0.2	0.2	0.5
Connect_delete_Prob	0.5	0.5	0.1
Connect_add_prob	0.5	.05	0.7
Zeit pro Generation	~60 sec	~90-100 sec	~100-130 sec
Höchste Fitness	2306	2825	2340

Abbildung 26: Unterschiede zwischen den NEAT Parametern der Populationen.

6.2 Schlussfolgerung und Probleme

Wie in der Beobachtung beschrieben war die ursprüngliche Fitnessfunktion fehlerhaft. Sie hat für jedes Bild betrachtet, ob sich die X Position des Charakters positiv verändert hat und dann den Belohnungswert um 1 erhöht. Der Fehler, der dadurch entsteht, hängt mit der Geschwindigkeit des Charakters zusammen.

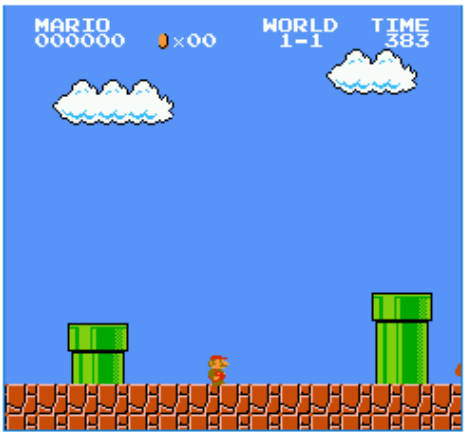	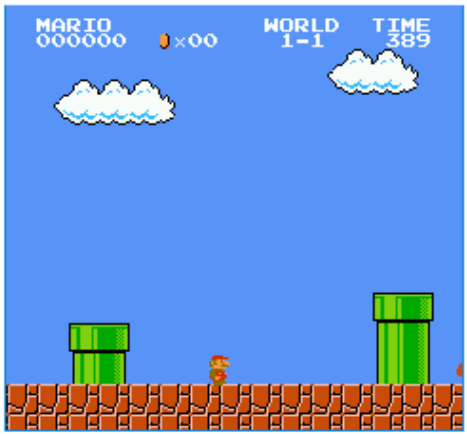
Distanz: 810	Distanz: 810
Fitness: ~670	Fitness: ~590
Übrige Zeit: 383	Übrige Zeit: 389

Abbildung 27: Vergleich der Fitness von 2 Genomen mit derselben Distanz

Wie in Abbildung 26 zu sehen, erreicht der rechte Mario dieselbe Distanz wie der linke Mario, erhält aber weniger Belohnungen und besitzt daher einen geringeren Fitnesswert. Der rechte Mario hat im Unterschied zu dem linken Mario in einer kürzeren Zeitspanne die Distanz erreicht. Dadurch ist die Menge an vergangenen Bildern geringer und da jedes Bild die Belohnung nur um 1 steigern kann, sammelt der schnellere rechte Mario weniger Belohnungen. Um diesen Fehler zu beheben, wird der Belohnungswert jedes Bild um die Differenz der vorherigen und der aktuellen X Position von Mario erhöht. Trotz der verbesserten Fitnessfunktion ist es merkbar, dass NEAT ein relativ alter Algorithmus ist und sehr viel Zeit benötigt für das Training. In Population 2 ist dieses Problem gut erkennbar. Der Standard NEAT Algorithmus hat nach Absolvieren der ersten 90% des Levels bei Generation 500 fast 700 weitere Generationen benötigt, bevor er einen positiven Zuwachs im Fitnesswert aufbringen konnte. Um den Lernprozess zu beschleunigen, gibt es einige Optionen. Das derzeitige Programm läuft jedes Genom nacheinander ab, stattdessen könnte das Programm umgeschrieben werden, sodass mehrere Genome gleichzeitig gespielt werden. Die Anzahl an parallellaufenden Genomen ist aber abhängig von der Anzahl der Prozessorkerne des Systems.

Eine andere Option ist die Limitierung der möglichen Ausgaben des Netzes. Tasten wie Start/Select oder der Controllerwert haben keinen Einfluss auf das Spiel und können daher komplett entfernt werden, dadurch kann die Anzahl an Ausgangsneuronen auf 6 reduziert werden. Um die Anzahl an Ausgangsneuronen noch weiter zu verringern, kann im Extremfall die Bewegung der KI auf das Laufen nach rechts und das Springen reduziert werden. Neben den möglichen Ausgaben können die Qualität und Menge der Daten verändert werden. Die für alle 3 Population genutzten Version des Bildes könnten durch die Verkleinerung der Auflösung und des Grautons einige Daten verloren haben. Mehr Daten führen in der Regel zu besseren Ergebnissen aber auch zu längerer Verarbeitungszeit. Im Fall von Super Mario Bros ist es effektiver die Qualität der Daten zu verbessern. Dafür werden alle Grafiken des Spiels durch verschiedene Zeichen ersetzt. Die Grafiken oder auch Sprites in Mario bestehen aus 16x16 Pixeln, dadurch verringern wir die Menge an Eingaben auf 225. Im Vergleich zu der genutzten Skalierung für die Durchführung werden keine Informationen verloren. Außerdem können die Zeichen, die für den Himmel oder die Hintergrundhügel stehen, entfernt werden ohne das relevante Informationen verloren gehen. Aus den Populationen lässt sich auch erkennen, dass die maximale Stagnation in der NEAT Konfiguration angehoben werden muss, da Population 1 und 3 selten über einen gewissen Fitnesswert gekommen sind. Ein höheres Maximum der Stagnierung gibt den Spezies mehr Zeit sich zu entwickeln.

Abbildung 28: Umwandlung des Bildes in Zeichen

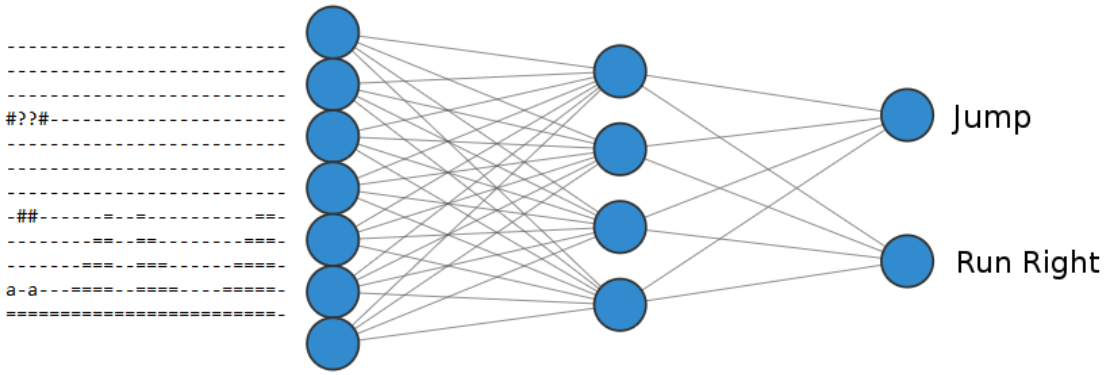

Abbildung 29: Reduziertes neuronale Netz

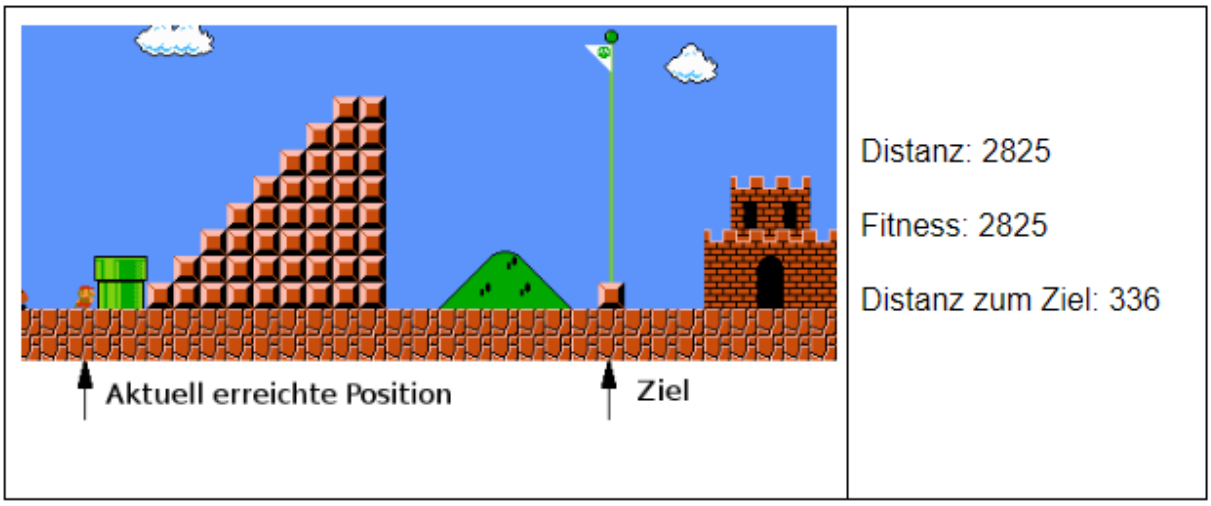

Abbildung 30: Letztes Hindernis und Stand von Population 2

Während Population 1 und 3 in absehbarer Zeit das Ziel nicht erreichen werden, wird Population 2 mit deutlich weniger Zeit das Spiel beenden können. Die Population 2 stagniert zurzeit an der Kombination aus Hindernis und Treppenstufen. Schlussfolgernd lässt sich sagen, dass die Entwicklung der Super Mario Bros KI erfolgreich ist. Die KI ist zwar noch nicht in der Lage das gesamte erste Level abzuschließen, hat aber Lösungen für die Überwältigung beider Objekte als einzelne bereits gefunden. Um sie für den letzten Abschnitt zu trainieren, kann mithilfe des Integration Tools ein neuer Zustand erzeugt werden. Im neuen Zustand ist die Startposition des Charakters an das letzte Hindernis gesetzt. Die Anbindung der NEAT Bibliothek an Gym Retro und ihre Nutzung verlief ohne Probleme oder Fehler. NEAT ist zugegebenermaßen nicht der aktuellste Algorithmus, bietet aber trotzdem eine einfache Angehensweise an die Entwicklung eines neuronalen Netzes und ist für Super Mario Bros mehr als ausreichend.

7 Fazit

Um zu bestimmen, wie der aktuelle Stand der KI Bibliotheken ist, wurde im Laufe dieser Arbeit eine KI für das Spiel Super Mario Bros entwickelt. Die Evaluierung des aktuellen Standes bezieht sich auf die Benutzerfreundlichkeit der Bibliotheken für Nutzer ohne Erfahrungen in der Nutzung von KIs, aber auch auf den technologischen Stand der Bibliotheken. Die Benutzerfreundlichkeit ist abhängig von den Bibliotheken. Nicht alle bieten wie Gym Retro eine eigene Benutzeroberfläche an, besitzen dafür aber eine Vielzahl von Beispielen und Tutorials. Zudem sind viele Bibliotheken ausführlich dokumentiert. Die meisten Bibliotheken wie Gym Retro erlauben es dem Nutzer auf verschiedenen Betriebssystemen zu arbeiten. Um die Hardware muss sich der Nutzer auch nicht mehr kümmern. Viele Bibliotheken bieten die Option vortrainierte Modelle zu nutzen. Sollte ein eigenes Modell trainiert werden, können über Onlinedienste wie AWS von Amazon die nötige Rechenleistungen gemietet werden. Der in der Arbeit genutzte Algorithmus NEAT mag zwar nicht auf den aktuellen Stand der Forschung sein, bietet trotzdem eine einfache und verständliche Angehensweise an Machine Learning. Die Forschung und Entwicklung von Algorithmen führen neben ihrer verbesserten Effektivität auch zu einer Vereinfachung der Nutzung. Zum Beispiel ist das Forschungsziel von PPO gewesen die Implementierung und Bearbeitung des TRPO Algorithmus zu vereinfachen.

Dennoch bestehen viele Probleme in den KI-Bibliotheken. Obwohl sie für die meisten Betriebssysteme angepasst sind, ist die Umsetzung nicht immer fehlerfrei. Das Integration Tool von OpenAI ist auf Windows sehr instabil und scheitert oftmals bei der Simulierung des Spiels und bei der Extraktion von Speicheradressen. Zudem wird die Einbindung einer KI abhängig von dem Nutzungsgebiet komplizierter und benötigt die Einarbeitung in komplexe Algorithmen wie PPO. Komplexe Algorithmen wie TRPO auf dem PPO basiert benötigen ein gewisses Verständnis im Bereich der Mathematik.

Abschließend lässt sich sagen, dass die Entwicklung von KIs mit der Nutzung von KI-Bibliotheken immer einfacher wird. Größere Bibliotheken lassen sich aufgrund ihrer detaillierten Dokumentation und der Vielfalt an unterstützten Programmiersprachen schnell erlernen. Darüber hinaus entstehen durch das steigende Interesse für KIs mehr Bibliotheken für spezifische Anwendungsgebiete, was die Einsatzmöglichkeiten von KIs erweitert.

8 Abbildungsverzeichnis

Abbildung 1: Mustererkennung Schema....5
Quelle: Eigene Darstellung
Abbildung 2: Prognoseschema....6
Quelle: Eigene Darstellung
Abbildung 3: Machine Learning Typen....8
Quelle: Eigene Darstellung
Abbildung 4: Neuronales Netz....9
Quelle: Eigene Darstellung
Abbildung 5: Lobe User Interface....11
Quelle: https://lobe.ai/examples
Aufgerufen am 25.06.2019
Abbildung 6: Screenshot von Super Mario Bros....12
Quelle:
https://www.google.com/search?q=super+mario+bros&safe=active&source=lnms&tbm=isch&sa=X&ved=0ahUKEwjohoigo5bjAhVS_aQKHccoAPUQ_AUIESgC&biw=1920&bih=920#imgrc=-KtmapC258XosM:
Aufgerufen am 12.07.2019
Abbildung 7: Integration Tool von OpenAI....13
Quelle: https://www.videogames.ai/assets/openai_romtool/hpos_variable.png
Aufgerufen am 12.07.2019
Abbildung 8: Sicht der KI in Grand Theft Auto....15
Quelle: https://cdn0.tnwcdn.com/wp-content/blogs.dir/1/files/2016/09/GTA-self-driving-cars.png
Aufgerufen am 12.07.2019
Abbildung 9: Trainingsschema von AlphaStar....18
Quelle: https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/
Aufgerufen am 20.07.2019
Abbildung 10: Screenshot von Dota 2 ohne UI....19
Quelle: https://www.altchar.com/games-news/573686/quit-practicing-dota-2-openai-will-beat-you-anyway
Abbildung 11: Diagramm des Ablaufes von GA Algorithmen....22
Quelle: Eigene Darstellung
Abbildung 12: Aufbau der einzelnen Gene....23
Quelle: https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume21/stanley04a-html/node3.html
Aufgerufen am 18.07.2019
Abbildung 13: Genom und das resultierende neuronale Netz....24
Quelle: https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume21/stanley04a-html/node3.html
Aufgerufen am 18.07.2019
Abbildung 14: Paarung zweier Genome....24
Quelle: https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume21/stanley04a-html/node3.html
Aufgerufen am 18.07.2019

Abbildung 15: Genom aus zwei Genomen mit selber Fitness 25
Quelle: https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume21/stanley04a-html/node3.html
Aufgerufen am 18.07.2019
Abbildung 16: Mögliche Mutationen in NEAT 25
Quelle: Eigene Darstellung
Abbildung 17: Gleichung zur Bestimmung der Spezies.....26
Quelle: Eigene Darstellung
Abbildung 18: Neuronales Netz anhand von Pong 28
Quelle: https://openai.com/blog/evolution-strategies/
Aufgerufen am 22.07.2019
Abbildung 19: Ablauf des Trainings in RL.....29
Quelle: https://www.statworx.com/de/blog/einfuehrung-in-reinforcement-learning-wenn-maschinen-wie-menschen-lernen/
Aufgerufen am 20.07.2019
Abbildung 20: Beispiel Reihen für Pong 29
Quelle: http://karpathy.github.io/2016/05/31/rl/
Aufgerufen am 19.07.2018
Abbildung 21: Screenshot von Montezuma. 30
Quelle: http://karpathy.github.io/2016/05/31/rl/
Aufgerufen am 19.07.2018
Abbildung 22: Skalierung des Bildes von Super Mario Bros.....33
Quelle: Eigene Darstellung
Abbildung 23: NES Controller Tastenlayout 34
Quelle: https://cdn.shopify.com/s/files/1/1165/0504/files/4f0c5b8e-15e6-11e5-9255-b920543518d6_large.png?3168465211638717362
Aufgerufen am 21.07.2019
Abbildung 24: Beispielnetz für Super Mario Bros. 34
Quelle: Eigene Darstellung
Abbildung 25: Ausgabe der Information einer Generation nach Abschluss einer Generation.....36
Quelle: Eigene Darstellung
Abbildung 26: Unterschiede zwischen den NEAT Parametern der Populationen.....38
Quelle: Eigene Darstellung
Abbildung 27: Vergleich der Fitness von 2 Genomen mit der selben Distanz 39
Quelle: Eigene Darstellung
Abbildung 28: Umwandlung des Bildes in Zeichen.....40
Quelle: Eigene Darstellung
Abbildung 29: Reduziertes neuronale Netz 41
Quelle: Eigene Darstellung
Abbildung 30: Letztes Hindernis und Stand von Population 2 41
Quelle: Eigene Darstellung

9 Softwareverzeichnis

Komodo Edit 11, Texteditor
Quelle: https://www.activestate.com/products/komodo-ide/downloads/edit/
Aufgerufen am 20.06.2019

OpenAI Gym Retro, Bibliothek
Quelle: https://github.com/openai/retro
Aufgerufen am 21.07.2019

NEAT, Bibliothek
Quelle: https://github.com/CodeReclaimers/neat-python
Aufgerufen am 25.07.2019

OpenCV, Bibliothek
Quelle: https://opencv.org/
Aufgerufen am 29.06.2019

10 Literaturverzeichnis

OpenAI (28.08.2017). Proximal Policy Optimization Algorithms. Abgerufen am 25.07.2019 von https://arxiv.org/pdf/1707.06347.pdf

Shangtong Zhang, Osmar R.Zaiane (07.03.2018). Comparing Deep Reinforcement Learning and Evolutionary Methods in Continuous Control. Abgerufen am 10.07.2019 von https://arxiv.org/pdf/1712.00006.pdf

Walter Simon (21.02.2019). Künstliche Intelligenz. Abgerufen am 04.07.2019 von https://books.google.de/books?id=bXOMDwAAQBAJ&printsec=frontcover&dq=k%C3%BCnstliche+intelligenz&hl=de&sa=X&ved=0ahUKEwjA5ZHHh_jhAhUBZFAKHcprDFwQ6AEIWDAJ#v=onepage&q=starke&f=false

Stuart Armstrong, Kaj Sotala, Sean S. OhEigeartaigh (20.05.2014). The errors, insights and lessons of famous AI predictions – and what they mean for the future. Abgerufen am 05.06.2019 von http://www.fhi.ox.ac.uk/wp-content/uploads/FAIC.pdf

Nick Bostrom, Vincent C. Müller (2014). Future Progress in Artificial Intelligence: A Survey of Expert Opinion. Abgerufen am 31.07.2019 von https://nickbostrom.com/papers/survey.pdf?source=post_page----------

Ich versichere, die vorliegende Arbeit selbstständig ohne fremde Hilfe verfasst und keine anderen Quellen und Hilfsmittel als die angegebenen benutzt zu haben. Die aus anderen Werken wörtlich entnommenen Stellen oder dem Sinn nach entlehnten Passagen sind durch Quellenangaben eindeutig kenntlich gemacht.

Ort, Datum Henry Tieu