Informatik
Hauptcampus

HOCHSCHULE TRIER

Reinforcement Learning

Simulationstechnik und Reinforcement Learning

Frederic Nicolas Schneider

Simulationstechnik und Reinforcement Learning

Betreuer: Prof. Dr. Christoph Lürig

Trier, 11.07.2023

Inhaltsverzeichnis

1

Erweiterung der Simulation

1.1 Rückblick auf die aktuelle Simulation

Die Simulation wurde in drei Elementen aufgeteilt: der Fahrstuhlsteuerung, Personensteuerung und der verbindenden Simulation. Dabei wird in jedem Takt überprüft, ob Personen einen Fahrstuhl zu einer Etage rufen. Ist der Fahrstuhl leer, prüft er in jedem Takt, ob er gerufen wurde. Hat er jedoch Passagiere, so fährt er zur nächsten Zieletage und nimmt, wenn möglich, auf dem Weg weitere Passagiere mit.

1.2 Die Entscheider-Schnittstelle

Als erste Erweiterung wurde eine Entscheider-Schnittstelle erstellt, die von der Fahrstuhlsteuerung aufgerufen wird. Diese gibt in der reinen Simulation die Entscheidung der Scanning-Strategie zurück. Ist der Entscheider des Reinforcement Learnings aktiviert, gibt diese Schnittstelle eine neutrale Entscheidung zurück. Die Entscheidung der Reinforcement Learners wurde bei der ersten Implementierung nach dem Fahrstuhlhandlings angewendet (Abbildung 1.1).

In einer zweiten Implementierung wurde die Möglichkeit hinzugefügt, die Simulation nur Schrittweise auszuführen und beim Training die Entscheidung außerhalb der Simulation zu treffen und diese für den nächsten Takt zu übergeben. Ist nur der Entscheider dieser Reinforcement-Implementation genutzt, so kann die Simulation diese in jedem Takt selbstständig vom Netz anfragen.

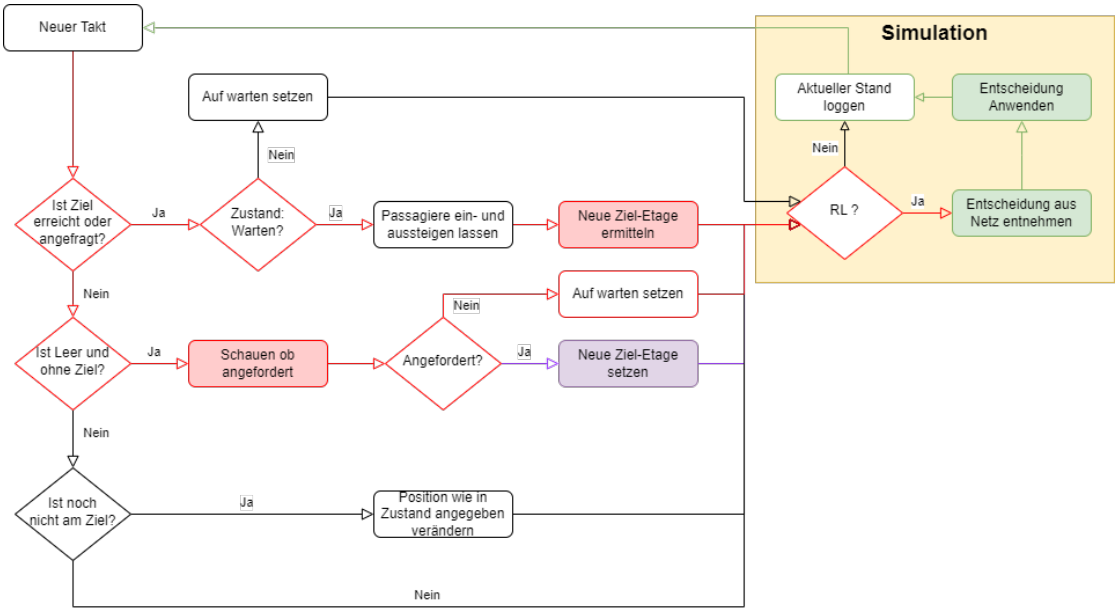

Abbildung 1.1: Der neue abgeänderte Ablauf der Fahrstuhlsteuerung. Entscheider (grün), Neue Schritte (rot), Nicht mehr genutzt (lila)

2

Reinforcement Learning Konzepte

Im Laufe des Projektes wurden zwei verschiedene Konzepte und Implementierungen verwendet. Übergreifend wurde sich für die Proximal Policy Optimization (PPO) entschieden.

2.1 Proximal Policy Optimization (PPO)

2017 führte OpenAI den Algorithmus ein und aufgrund seiner Benutzerfreundlichkeit und guten Leistung zum Standardalgorithmus für verstärktes Lernen bei OpenAI geworden. PPO ist ein on-policy Algorithmus, der, wie die meisten klassischen Reinforcement-Learning-Algorithmen, am besten durch ein dichtes Belohnungssystem lernt. OpenAI definierte für die Zielfunktion ihres Algorithmus wie folgt:

$$L^{\text{CLIP}}(\theta) = \hat{E}_t[min(r_t(\theta), 1-\varepsilon, q+\varepsilon)\hat{A}_t)]$$

- θ ist der Policy Parameter
- $\hat{E}_t$ bezeichnet die empirische Erwartung über Zeitschritte
- r_t ist das Verhältnis der Wahrscheinlichkeit unter der neuen und der alten Policy
- $\hat{A}_t$ ist der geschätzte Advantage zur Zeit t
- ε ist ein Hyperparameter, normalerweise 0.1 oder 0.2

Diese Formel wird vor allem in der Implementierung in Kapitel 4 genutzt.

2.2 Grundkonzept vor der Implementierung

Zunächst wurde sich ein Lernziel definiert, welches die Maximierung der Reward-Funktion vorsieht. So sollen alle Personen am Ende des simulierten Tages zu Hause sein und im Gesamten die Wartezeit minimiert werden.

Um dies zu erreichen, muss der Reinforcement Learner Beobachtungen in der simulierten Welt machen und diese in der Auswertung der trainierten Iteration einfließen lassen.

2.2.1 Weltbeobachtungen des Reinforcement Learners

Im ersten Konzept wurden daher folgende Parametern als Beobachtung dem Algorithmus weitergegeben: Die Zeit in Takten, Anzahl der Personen im Aufzug, der aktuelle Steuerungszustand (Hoch, Runter, Warten) des Fahrstuhls, auf welche Etagen ein Fahrstuhl in welche Richtung angefordert wurde und zuletzt die Zieletagen der Passagiere eines jeden Fahrstuhls.

Später wurden noch die durchschnittliche Wartezeit, die verbleibenden Personen im Gebäude und für jeden Aufzug die Fahrtrichtung und Position in der Beobachtung ergänzt.

2.2.2 Erste Reward-Funktion

Als Reward sollte zunächst die durchschnittliche Wartezeit prozentual zur Tageslänge errechnet werden. Dies würde mit der maximalen Anzahl der Personen, die im Bürokomplex arbeiten, multipliziert werden, wenn keiner mehr im Büro ist. Ansonsten wäre der Reward 0. Da sich jedoch über die Zeitschritte in der Simulation hinweg nur mit sehr viel Glück ein Reward ergeben würde, und somit der Reinforcement Learner etwas zum Optimieren hätte, musste die Reward-Funktion angepasst werden.

In der ersten Implementierung wurde die aktuelle durchschnittliche Wartezeit als negativer und Personen, die nach Hause gingen, als positiver Reward übergeben. Dazu später in Kapitel **??**.

3

Implementierung nach Zombie Dice

Die erste Implementierung eines PPO-Algorithmus basiert auf die Implementierung des Professors Dr. C. Lürig für das Spiel Zombie Dice. Dessen zweiköpfiges Netz besteht aus verschiedenen Linear Layers und verwendet die Leaky ReLu-Aktivierungsfunktion. Als Optimierer wird der Adam Optimizer genutzt. Die zwei Köpfe geben Value und Policy und haben eine gemeinsame Basis. Der Entscheidungsoutput liefert dabei die je für ein Stockwerk für jeden Aufzug zurück. Der Input Tensor mit den Beobachtungen enthält 57 Elemente, wie zuvor in Kapitel 2.2.1 beschrieben.

3.1 Trainingsprozess nach Dice

Der Trainingsprozess des Zombie Dice Reinforcement Learners basiert auf der Kreuzentropie Methode und der Policy Gradient Theorems. Dabei wird das Monte-Carlo Verfahren angewendet, welche ganze Episoden, in diesem Fall einen Batch mit simulierten Tagen in Anwendung der aktuellen Policy, benötigt.

Um Daten für den Trainingsprozess zu sammeln, wurde in jeder Epoche 64 Tage mit der aktuellen Policy simuliert. Anschließend wird der Loss berechnet und die Policy angepasst. Die Daten eines Tages werden innerhalb der Simulation gespeichert und am Ende zurückgegeben. Da in dieser Implementation ein Batch aus abgeschlossen simulierten Tagen besteht, ist eine schrittweise Ausführung, wie die Bibliothek SimPy unterstützt, nicht Notwendig.

Nach dem Trainingsschritt wird das Netz 25 Mal getestet, um einen stabilen durchschnittlichen Reward zur Bewertung des Netzes zu erhalten. Ist das aktuelle Netz besser als seine Vorgänger, so wird der aktuelle Stand zwischengespeichert. Der Trainingsprozess ist mit dem Erreichen des Ziels, dass keine Personen sich im Bürokomplex mehr befinden dürfen fürs Erste beendet.

4

Implementierung nach Neuralnet

2021 veröffentlichte der YouTube-Kanal MachineLearningwithPhil ein Video zur Implementierung eines einfachen Reinforcement Learners mit Proximal Policy Optimization. Als Simulation wurde auf die Gym Bibliothek zurückgegriffen, welche kleine Beispielsimulationen bereithält, auf denen der eigene Reinforcement Learner trainieren kann.

4.1 Trainingsprozess nach Neuralnet

Der Reinforcement Learner von Neuralnet basiert auf die in Kapitel 2.1 gezeigte Formel von OpenAI.

Im Gegensatz zur Implementierung nach Zombie-Dice i, voran gegangenen Kapitel 3.1, wird nicht erst mehrere Episoden mit der aktuellen Policy simuliert und anschließend ausgewertet, sondern setzt den Lernprozess noch während der Simulation ein. Im Original-Code wird alle 20 Takte der Trainingsprozess gestartet und eine neue Policy bestimmt.

Diese Trainingsrate ist jedoch im Falle der Fahrstuhlsimulation zu hoch gewählt, da in den ersten 700 Takten keine Personen den Bürokomplex betreten und somit jegliche Entscheidungen des Reinforcement Deciders keine Auswirkung auf die Simulation zeigt. Daher wurde sich für eine Trainingsrate von 8 Iterationen pro simulierten Tag, etwa alle 3 Stunden, entschieden. Zwar zeigen so die ersten zwei Iterationen keine Änderungen durch die Entscheidungen, jedoch die restlichen 6 Iterationen sind die wichtigsten Zeiten, in denen die Befehle der Fahrstühle eine hohe Auswirkung zeigen. In einer weiteren Variante könnte man die ersten zwei Iterationen überspringen, und somit diesen Effekt verringern. Dies wurde erst in der Auswertung des Trainings erkannt und nicht mehr rechtzeitig für weitere Trainingsversuche notiert.

Das Netz dieser Implementierung ähnelt dem aus der Implementierung nach Zombie-Dice. Jedoch werden hier die zwei Köpfe als getrennte Netze behandelt.

5

Trainingsergebnisse nach Zombie Dice

Nach dem ersten Trainingsprozess erreichte das Netz nach Zombie-Dice eine durchschnittliche Wartezeit von 955 Takten und alle Personen verbleiben im Gebäude. Als Reward wurde die durchschnittliche Wartezeit als negativer Wert übergeben. Wie in Abbildung 5.1 zu sehen, viel die Leistung des Netzes jedoch schnell.

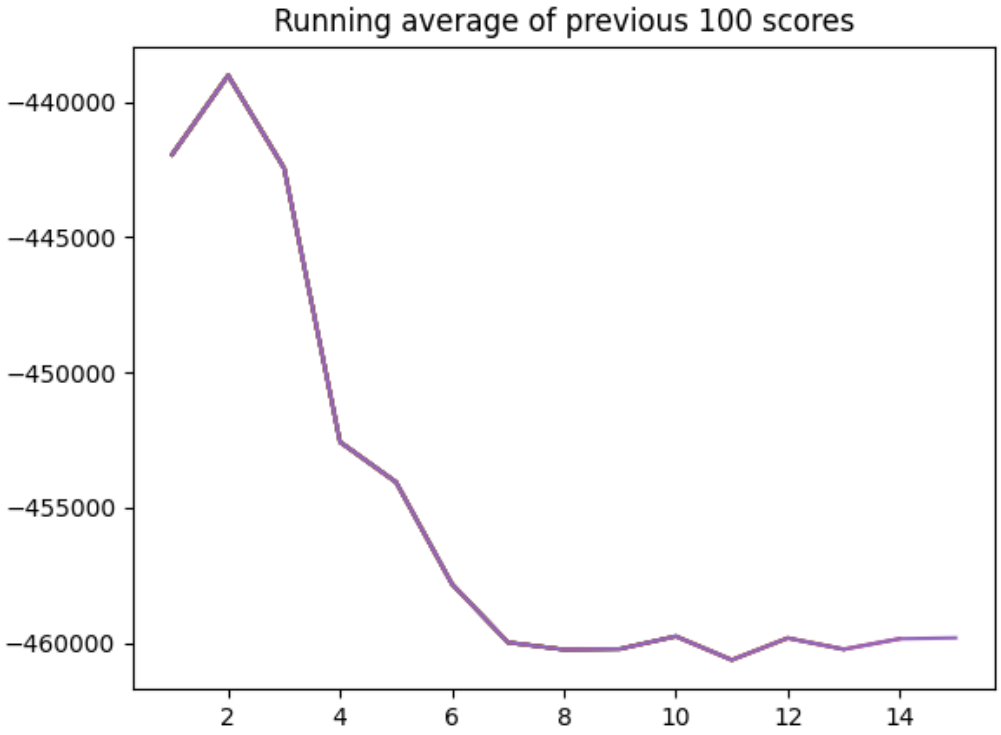

Abbildung 5.1: Trainingsergebnis nach 35 Iterationen

Daher wurde der Reward zu einem positiven Wert für Einstiege und doppelter Wert für Ausstiege von Passagieren vergeben. Dadurch wurde eine durchschnittliche Wartezeit von 897 Takten und alle Personen verbleibend im Gebäude erreicht.

Die Abbildung 5.2 zeigt, wie nach 35 Iterationen erneut die Leistung kurz stieg, aber schnell wieder abfiel. Daher wurde die Implementierung nach Neuralnet weiterverwendet.

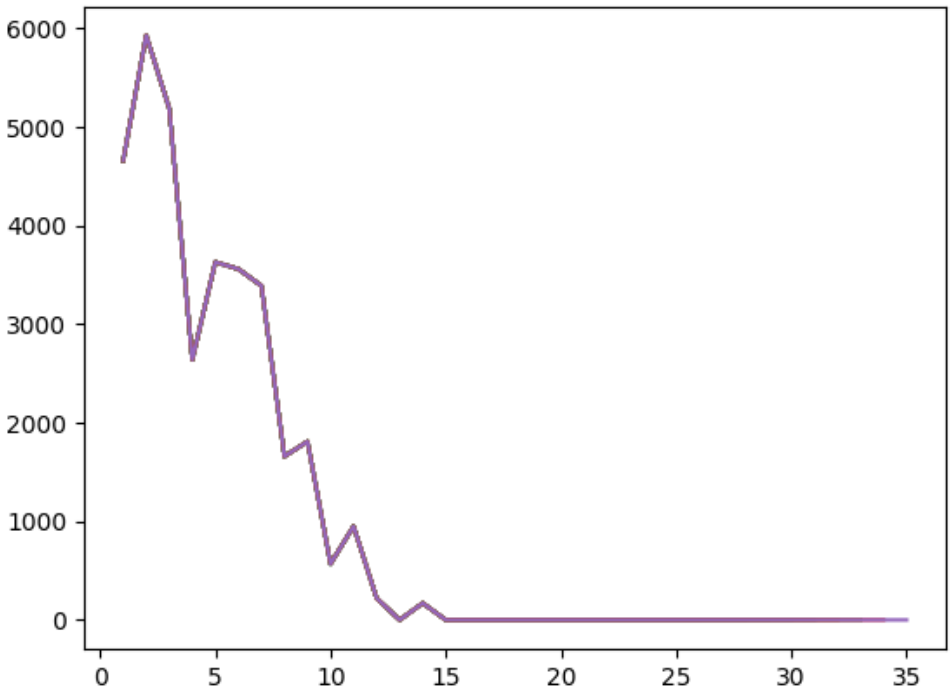

Abbildung 5.2: Trainingsergebnis nach 35 Iterationen

6

Trainingsergebnisse nach Neuralnet

Nach dem Trainingsprozess nach Neuralnet erreichte das Netz eine durchschnittliche Wartezeit von 876 Takten und 97/100 Personen verbleibend im Gebäude. Die Abbildung 6.1 zeigt, dass er trotz große Schwankungen ins gesamt wellenförmig lernt. Das Netz wird besser und fällt dann auch wieder und pendelt sich bei den oben angegebenen Werten ein. Durch das Hinzufügen von mehreren Schichten konnte sich Ergebnis verbessern, jedoch wird sich wohl eine Anpassung der Netze und der Reward-Funktion eine Besserung ergeben, welche aus Zeitgründen nicht weiter optimiert werden konnte.

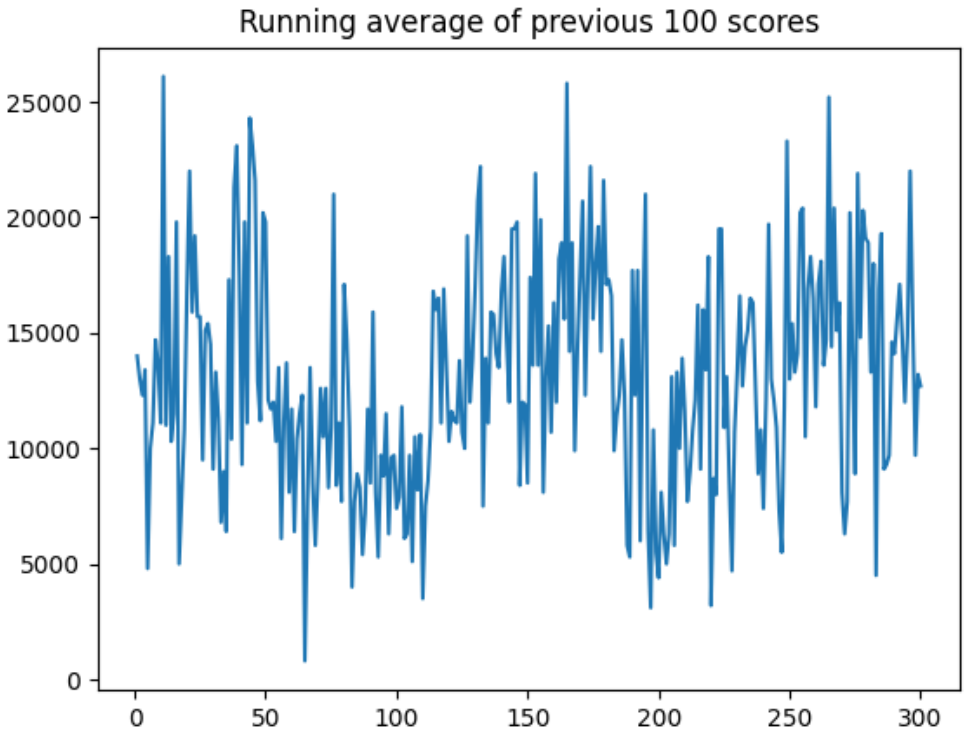

Abbildung 6.1: Trainingsergebnis nach 900 Iterationen

7

Erkenntnis aus Simulation und Reinforcement Learner

Neben dem Aufbau einer Simulation mit ihren vielen Parametern gehört das Konstruieren eines passendes Entscheider-Netz und dazugehörigen Reward-Funktion zu den schwersten Aspekten des Reinforcement Learnings. Neben den Herausforderungen, die ein künstliches neuronales Netz mit sich bringt, ist der erweiterte Aspekt des Rewards eine der größten Fehlerquellen. So kann der Entscheider falsche schlüsse aus seinen Entscheidungen und der daraus resultierenden Beobachtungen schließen und die gesamte Performance verschlechtern.