Heinrich Heine
Universität
Düsseldorf

INSTITUT FÜR INFORMATIK
Softwaretechnik und Programmiersprachen
Universitätsstr. 1 D–40225 Düsseldorf

Validierung von Reinforcement-Learning-Agenten mittels Trace-Analyse

Bachelorarbeit

im Studiengang Informatik
zur Erlangung des akademischen Grades

Bachelor of Science (B.Sc.)

vorgelegt von
Davin Holten

Beginn der Arbeit: 08. Februar 2023
Abgabe der Arbeit: 08. Mai 2023

Erstgutachter: Prof. Dr. Michael Leuschel
Zweitgutachter: Prof. Dr. Stefan Conrad

Selbstständigkeitserklärung

Hiermit versichere ich die vorliegende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Düsseldorf, den 08. Mai 2023

Davin Holten

Zusammenfassung

Formale Modelle werden immer wichtiger für die Modellierung und Validierung von Reinforcement-Learning Systemen, jedoch kann die Validierung zeitaufwendig und ressourcenintensiv sein. In dieser Arbeit wird ein neuer Ansatz untersucht, bei der ein abstraktes formales Modell mithilfe von Künstlicher Intelligenz(KI) simuliert wird. Die resultierenden Traces werden von SimB validiert, ohne dass eine eigene separate Simulation geschrieben werden muss. Diese Methode ist neuartig, da weder ProB2-UI noch SimB die Logik der einzelnen Operationen kennen muss, da diese direkt von der KI berechnet werden. Für die Forschung an der Methode wurde im ersten Schritt ein Reinforcement-Learning Agent auf einer Highway-Umgebung trainiert. Im Anschluss wurden die generierten Traces mithilfe von SimB validiert. Die Ergebnisse der Forschung zeigen, dass die Validierung von Reinforcement-Learning Agenten durch diesen Ansatz es ermöglicht, einen umfangreichen Überblick über das Verhalten des Agenten zu gewinnen.

Danksagung

An erster Stelle möchte ich bei Prof. Dr. Michael Leuschel bedanken, der mir die Möglichkeit gegeben hat, diese Arbeit zu schreiben. Ein weiteres großen Dank, geht an meine Betreuer Jannik und Fabian, die mir während wöchentlichen Meetings viel Feedback zu meiner Arbeit gegeben haben. Außerdem möchte ich mich bei meinen Freunden und meiner Familie bedanken, die mich während der Arbeit unterstützt haben.

Inhaltsverzeichnis

1 Einleitung

Künstliche Intelligenz [Rus10] hat in den letzten Jahren eine rasante Entwicklung erlebt, die weitreichende Auswirkungen auf zahlreiche Branchen und Anwendungsbereiche hat. Die Fortschritte in der KI-Forschung haben die Möglichkeiten der Automatisierung und Optimierung in den verschiedensten Bereichen revolutioniert, wie zum Beispiel im Bereich der Spracherkennung [HDY+12], Bildverarbeitung [KSH17], Medizin [EKN+17] oder dem autonomen Fahren [BDTD+16]. Ein entscheidender Faktor, der zu dieser Entwicklung beigetragen hat, ist das maschinelle Lernen (ML) [Tom97], das als eine der Haupttechniken innerhalb der KI angewandt wird. Insbesondere das Reinforcement Learning (RL) [SB18], ein Teilgebiet des ML, hat großes Potenzial gezeigt, indem es künstlichen Agenten ermöglicht, aus Interaktionen mit ihrer Umgebung zu lernen und ihr Verhalten auf Grundlage von Feedback effizient und zielgerichtet anzupassen. Nach Sutton und Barto [SB18] ist das Verfahren, aus der Umwelt zu lernen, intuitiv, da auch wir Menschen bereits von klein auf kontinuierlich aus den Reaktionen unserer Umgebung auf unsere Interaktionen mit ihr lernen. In Reinforcement-Learning Systemen ist es das Ziel des lernenden Agenten, dessen Reward zu maximieren. Entscheidend hierbei ist es, dass der Agent selbstständig herausfinden muss, welche Aktionen er ausführt, um sowohl kurzfristige, als auch langfristige Ziele zu erreichen. Denn jede Aktion wird durch das Environment mit einem positiven oder negativen Reward bewertet.
Von immer größer werdender Bedeutung ist dabei die Gewährleistung der Sicherheit [AOS+16] des Agenten selbst, aber vor allem auch die seines Umfeldes, denn mit der zunehmenden Relevanz innerhalb sicherheitskritischer Anwendungsbereiche, sind die potenziellen Risiken und Folgen fehlerhafter Entscheidungen immens.

1.1 Motivation

Um die Risiken und Folgen fehlerhafter Entscheidungen zu minimieren, werden immer wieder neue Methoden entwickelt, die es ermöglichen sollen, die Sicherheitsaspekte von RL-Agenten zu realisieren und zu validieren. Nathan Fulton und André Platzer von der Carnegie Mellon University erläutern im Paper „Safe Reinforcement Learning via Formal Methods: Toward Safe Control Through Proof and Learning“ eine Methode, um die Sicherheit während des Reinforcement Learning zu garantieren. Justified Speculative Control (JSC) kombiniert „formal verification“ und Reinforcement Learning. Diese Methode beruht darauf, dass Reinforcement Learning die „formal verification“ um ihre Robustheit gegen unvollständige Modelle ergänzt. Denn trotz der hohen Sicherheit, welche durch „formal verification“ gegeben ist, besteht ihr Nachteil darin, dass die zu überprüfenden Modelle mit der Realität übereinstimmen müssen [FP18]. Um die Entwicklung von Modellen zu erleichtern, die es erlauben, das Verhalten von RL-Agenten zu analysieren und zu validieren, soll mit SimB ein neuer Ansatz getestet werden. Bei herkömmlichen Validierungsansätzen würde man ein formales Modell erstellen, anschließend eine Simulation für dieses Modell entwickeln und dann validieren. In dieser Arbeit hingegen wird ein abstraktes formales

Modell verwendet und die Durchführung der Simulation einer KI überlassen. Die daraus resultierenden Traces werden dann in SimB mit verschiedenen Methoden validiert, ohne dass wir eine eigene Simulation entwickeln müssen. Dieser Ansatz soll es erlauben, die Validierung effizienter und flexibler zu gestalten, da leicht neue Tests für Verhaltensüberprüfung geschrieben werden können. Vorteil von SimB zur Validierung von RL-Agenten ist, dass SimB es ermöglicht, die Validierung von RL-Agenten mit Trace Replay anschaulich und effizient durchzuführen. Grund dafür ist, dass wenn wir ein abstraktes Modell verwenden und die Simulation der KI überlassen, das Modell weniger komplex wird, da die Logik hinter einzelnen Operationen aufseiten von SimB vernachlässigt werden kann. Dadurch soll dieser Ansatz eine Alternative zu traditionellen Validierungsmethoden bieten.

1.2 Ziel der Arbeit

Im Rahmen dieser Bachelorarbeit soll ein weiterer Ansatz innerhalb des Reinforcement Learning Bereiches untersucht und evaluiert werden. Dabei liegt der Fokus auf der Verwendung von Trainingsdaten, die in die Form von Traces umgewandelt werden, um mithilfe der Simulationssoftware SimB [VLM21a] das Verhalten eines RL-Agenten in einer Highway-Umgebung [Leu18] zu analysieren. Im Vergleich zu bestehenden Methoden sollen sowohl die Vor- als auch die Nachteile dieses Ansatzes herausgearbeitet und diskutiert werden. Die Arbeit ist in zwei Hauptteile untergliedert, die sich jeweils mit unterschiedlichen Aspekten der Forschung befassen.

Der erste Teilbereich konzentriert sich auf die Entwicklung eines RL-Agenten auf einer Highway-Umgebung. Hierbei werden verschiedene RL-Algorithmen angewandt, um Entscheidungen hinsichtlich der Navigation und der Interaktion mit anderen Verkehrsteilnehmern zu treffen. Nach dem erfolgreichen Training des Agenten werden die dabei entstandenen Trainingsdaten in Form von Traces gespeichert. Diese Traces sollen dazu beitragen, das erlernte Verhalten des Agenten im Detail nachvollziehen zu können.

Im zweiten Teilbereich der Arbeit werden die zuvor generierten Traces mithilfe von ProB2-UI [BGJ+21] und SimB untersucht. Hierbei werden verschiedene Aspekte des Agentenverhaltens, wie die Einhaltung der Geschwindigkeitsgrenzen, betrachtet. Die Ergebnisse dieser Analyse sollen es ermöglichen, einen Überblick über die Leistung des Agenten innerhalb der Highway-Umgebung zu erhalten.

Die Arbeit soll abschließend einen Überblick über die Ergebnisse der Untersuchung geben und eine fundierte Bewertung der Verwendbarkeit von Trace Validierung für die Analyse von RL-Agenten liefern. Der Fokus liegt dabei auf dem Ansatz, durch ein abstraktes formales Modell und der Durchführung der Simulation von der KI selbst, die daraus resultierenden Traces von SimB validieren zu lassen, ohne dafür eine separate Simulation schreiben zu müssen.

In der vorliegenden Arbeit wird angestrebt, eine umfassende Übersicht über die Ergebnisse der Untersuchung zu präsentieren und eine fundierte Beurteilung der Anwendbarkeit

von Trace Validierungsmethoden für die Analyse von Reinforcement-Learning-Agenten (RL-Agenten) zu liefern. Der Schwerpunkt liegt hierbei auf der Verwendung eines abstrakten formalen Modells, für welches die Simulation von der Künstlichen Intelligenz selbst durchgeführt werden kann.

2 Grundlagen

2.1 Machine Learning

Maschinelles Lernen (ML) [Tom97] ist ein bedeutendes Teilgebiet innerhalb der Künstlichen Intelligenz, das sich primär auf die Entwicklung und Verbesserung von Algorithmen konzentriert. Diese Algorithmen ermöglichen es Computern, aus Daten und Erfahrungen zu lernen, um somit ihre Leistung bei bestimmten Aufgaben zu optimieren. Wie in der Definition von Mitchell et al. (1997) beschrieben: „Ein Computerprogramm wird als lernfähig bezeichnet, wenn es aus Erfahrungen E in Bezug auf eine Klasse von Aufgaben A und einer Leistungsmessung L, seine Leistung bei Aufgaben aus A, gemessen an L durch Erfahrungen E verbessert [Tom97]“. Im Gegensatz zur traditionellen Programmierung, bei der Algorithmen durch den Programmierer explizit implementiert werden, liegt der Fokus beim maschinellen Lernen auf der Automatisierung des Lernprozesses. Dadurch ist es möglich, mit wenigen direkten menschlichem Eingriffe komplexe Zusammenhänge innerhalb der Daten zu erkennen und beispielsweise Verfahren wie die Gesichtserkennung anhand von Mustern und Daten zu meistern. Dabei gibt es verschiedene Vorgehensweisen des maschinellen Lernens, die je nach Art der zu verarbeitenden Informationen und dem vorliegenden Anwendungsfall angewandt werden können. Die drei Hauptkategorien des maschinellen Lernens sind das überwachte Lernen, das unüberwachte Lernen und das Reinforcement-Learning.

2.1.1 Supervised Learning

Beim überwachten Lernen, häufiger bekannt als „Supervised Learning“ [GBC16], wird ein Algorithmus auf der Grundlage von gelabelten Trainingsdaten trainiert. Diese gelabelten Trainingsdaten enthalten sowohl Eingabewerte als auch korrekte Ausgabewerte. Der Zweck dieses Ansatzes besteht darin, den Algorithmus dazu zu bringen, Muster und Zusammenhänge [BN06] in den Daten zu erkennen und auf der Grundlage dieser Erkenntnisse Vorhersagen für bisher unbekannte Eingaben zu treffen. Im Regelfall werden die Daten vorher in zwei Teilmengen aufgeteilt. Die erste Teilmenge wird zum Trainieren des Algorithmus verwendet, während die zweite Teilmenge zur Validierung der Leistung des Algorithmus eingesetzt wird. In Abbildung 1 wird der Ablauf eines überwachten Lernprozesses dargestellt. Zunächst wird der Algorithmus mit den Trainingsdaten und den zugehörigen korrekten Ausgaben versorgt. Dadurch lernt der Algorithmus, die Eingabewerte mit den entsprechenden Ausgabewerten zu verknüpfen. Im Laufe des Trainings passt der Algorithmus seine internen Parameter an, um die Abweichung zwischen seinen Vorhersagen und den tatsächlichen Ausgabewerten zu minimieren. Als einer der bekanntesten Einsatzgebiete von überwachtem Lernen zählt die Bildererkennung [LBH15].

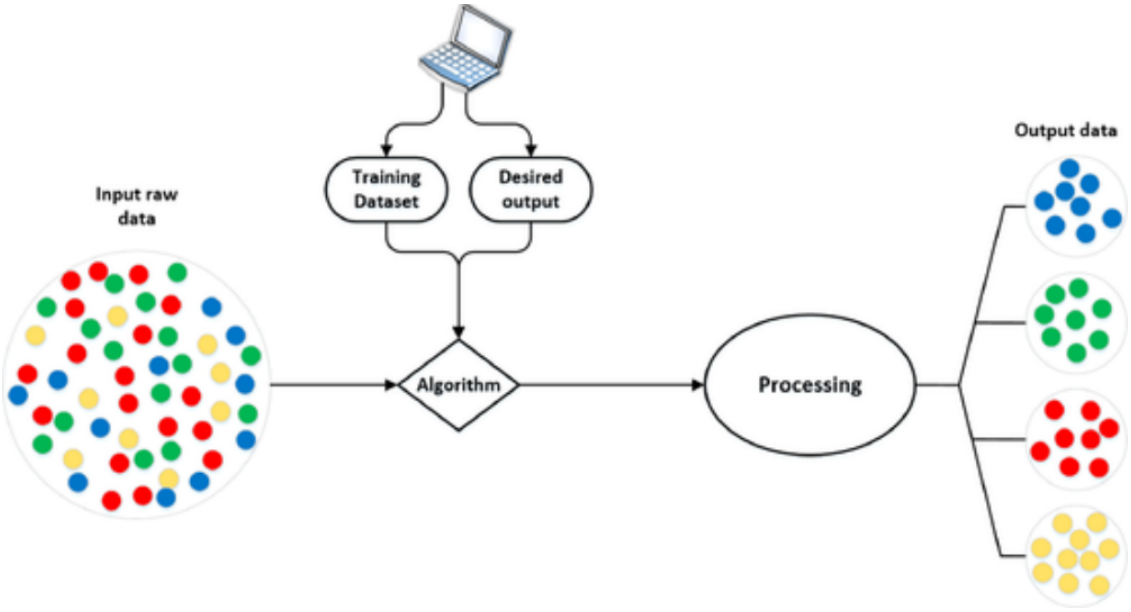

Abbildung 1: Illustration von Supervised Learning[SGAS20]

2.1.2 Unsupervised Learning

Der signifikante Unterschied vom überwachten zum unüberwachten Lernen, auch „Unsupervised Learning“ genannt, charakterisiert Zoubin Ghahramani [Gha04] darin, dass der Algorithmus ohne vorherige Kenntnisse über die korrekte Ausgabe, auf einem ungelabelten Traingssatz trainiert wird. Weiterhin schildert er, dass „Unsupervised Learning“ sich als Suche nach Mustern und Strukturen in Daten beschreiben lässt, in denen erstmal keine Zusammenhänge erkennbar sind. In Abbildung 2 wird der Prozess des „Unsupervised Learning“ verbildlicht. Der Algorithmus erhält eine Menge von unsortierten Daten. Aufgrund dieser Daten versucht der Algorithmus, Muster und Zusammenhänge zu erkennen, mit denen dann Vorhersagen auf diese unbekannten Daten getroffen werden. Eindeutiger Vorteil gegenüber „Supervised Learning“ ist, dass keine korrekt gelabelten Trainingsdaten von Menschen erstellt werden müssen. Dies ist insbesondere bei großen Datenmengen von Vorteil, da die Erstellung der „labels“ sich oft als kostspielig und zeitintensiv herausstellen kann [GBC16].

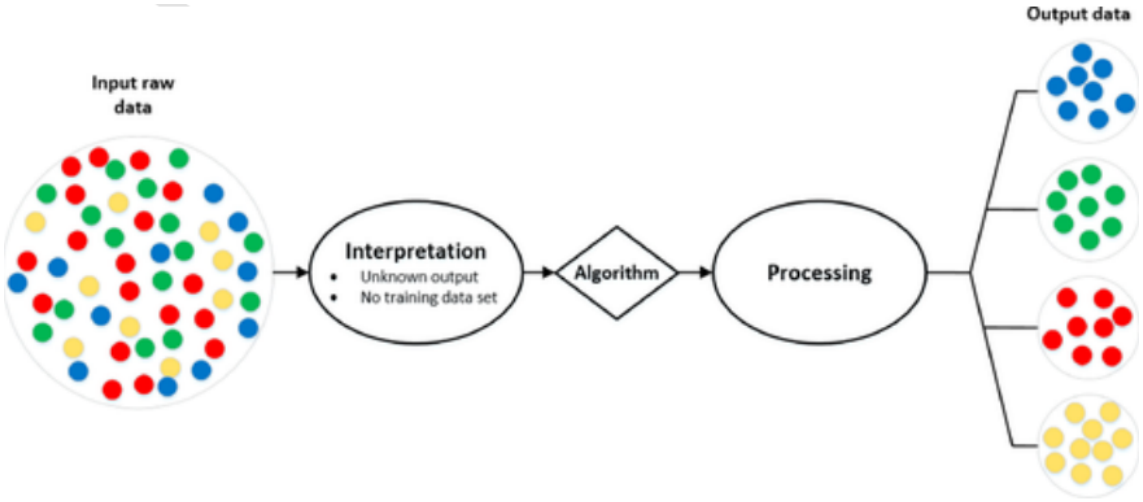

Abbildung 2: Illustration von Unsupervised Learning[SGAS20]

2.1.3 Reinforcement Learning

In dieser Arbeit wird mit einer weiteren Technik des maschinellen Lernens gearbeitet, dem bestärkenden Lernen „Reinforcement Learning“. Sutton und Barto [SB18] beschreiben „Reinforcement Learning“ wie folgt: „Reinforcement Learning ist Lernen, was zu tun ist und wie in einer bestimmten Situation eine Belohnung Reward zu maximieren ist.“ Hierbei wird dem lernenden Agenten nicht mitgeteilt, welche Aktion er zu wählen hat. Stattdessen muss dieser selbst herausfinden, welche Aktionen den höchsten Reward bringen. Dabei ist es wichtig, dass ausgeführte Aktionen Einfluss auf den kurzfristigen, als auch langfristigen Reward haben können. Sie betonen dabei, dass „Trial-and-Error“ und der verzögerte Einfluss auf den Reward eine tragende Rolle beim „Reinforcement Learning“ spielen.

Grundlegende Bausteine des Reinforcement Learning sind:

- Agent: Der Agent ist der lernende Teil des Systems. Er interagiert mit der Umgebung und lernt aus den gemachten Erfahrungen. Ziel des Agenten ist es, seinen Reward zu maximieren, in dem eine optimale Strategie gefunden wird.

- Environment: Das „Environment“ ist die Umgebung, in der der Agent agiert. Sie gibt den Agenten Informationen über ihren aktuellen Zustand und reagiert auf die gewählten Aktionen des Agenten. Das „Environment“ kann dabei versuchen die reale Welt widerzuspiegeln oder abstrakte Problemstellungen darzustellen.

- Reward: Die Belohnung „Reward“, ist der Rückgabewert der Umgebung, der dem Agenten Informationen über die Qualität seiner gewählten Aktion gibt. Belohnungen können positiv, negativ oder neutral sein. Das Ziel des Agenten ist es, seinen Reward zu maximieren.

- Policy: Eine „Policy“ ist eine Strategie oder Regel, die angibt, welche Aktion der Agent in einem bestimmten Zustand ausführen soll, um den Reward zu maximieren. Die „Policy“ kann dabei deterministisch oder stochastisch sein.

Veranschaulicht an einem Pseudocode sieht der Ablauf des Reinforcement Learning wie folgt aus:

Algorithmus 1 Q-Learning Algorithmus

```
Initialisiere die Umgebung und den Agenten
agent = Agent()
umgebung = Umgebung()
for Jeden Trainingsdurchlauf do
    Initialisiere zufälligen Startzustand zustand = umgebung.reset()
    while ist nicht beendet do
        Wähle Aktion auf der Grundlage des aktuellen Zustands
        aktion = agent.aktionAuswählen(zustand)
        Führe Aktion aus, beobachte Belohnung und nächsten Zustand
        nächsterZustand, belohnung, beendet = umgebung.aktionAusführen(aktion)
    end while
end for
```

2.2 Validierung

2.2.1 Traces

Traces sind chronologische Aufzeichnungen von Ereignissen oder Aktionen, die während der Ausführung eines Programmes stattfinden. Der Zweck von Traces liegt darin, dem Programmierer Einsicht in die Funktionsweise eines Programmes zu liefern. Dadurch können gezielt Fehler in der Anwendung identifiziert und behoben werden. Im Zusammenhang mit Modellen in der B-Methode können nach Stock et al. [SMLE22] Traces folgende Informationen liefern: „Erstens ein bestimmter Endzustand ist erreichbar, nachdem Schritte in einer bestimmten Reihenfolge ausgeführt wurden. Zweitens eine bestimmte Reihenfolge von Übergängen ist durchführbar. [SMLE22]“

2.2.2 B Methode

Die B-Methode ist ein formeller Softwareentwicklungsprozess, Jean-Raymond Abrial [ALN+91] definiert diesen folgendermaßen: „Die B-Methode ist ein formaler Softwareentwicklungsprozess zur Erstellung von hochzuverlässiger, portabler und wartbarer Software, die nachweislich korrekt in Bezug auf ihre funktionale Spezifikation ist. [ALN+91]“

Die Grundlagen der B-Methode lassen sich in mehrere Schlüsselkonzepte unterteilen:

Abstrakte Maschine: Die B-Methode beginnt mit der Definition einer abstrakten Maschine, die das zu entwickelnde System in einer abstrakten, formalen Notation beschreibt. Die abstrakte Maschine besteht aus Variablen, Invarianten, Initialisierungen und Operationen. Nach Abriel [Abr96] ist das Konzept der abstrakten Maschine nahe verwandt mit Klassen oder abstrakten Datentypen in der objektorientierten Programmierung.

Variablen: Variablen repräsentieren den Zustand des Systems. Sie werden mit Elementen aus Mengen oder Relationen ausgedrückt.

Invarianten: Invarianten sind Bedingungen, die für alle möglichen Zustände des Systems wahr sein müssen.

Initialisierungen: Initialisierungen definieren den Anfangszustand des Systems. Sie legen fest, welche Werte die Variablen zum Start des Systems haben.

Operationen: Operationen werden verwendet, um zwischen Zuständen zu wechseln und dadurch das System zu verändern [SMLE22].

2.2.3 Prob2-UI und SimB

ProB2-UI [BGJ+21] ist eine JavaFX basierte Benutzeroberfläche für den ProB Animator, Constraint Solver und Model Checker. ProB2-UI unterstützt Formalismen wie B und Event-B. Die Benutzeroberfläche bietet folgende Funktionen. Erstens eine Projektansicht, die es dem Nutzer erlaubt Modelle, Einstellungen und Konfigurationen für Validierungsaufgaben zu verwalten. Die Daten eines Projektes sind innerhalb einer Projektdatei gespeichert und enthalten alle wichtigen Informationen, damit der Benutzer bei einem späteren Check des Projektes nicht alle Daten neu eingeben muss.

Nach dem Laden eines Modells haben Benutzer die Möglichkeit, dieses zu animieren. Sie können dabei interaktiv die Ausführungsschritte auswählen und kontrollieren. Sobald eine Operation ausgewählt wird, ändert sich der aktuelle Zustand des Modells entsprechend. In der Zustandsübersicht werden dem Benutzer sowohl die aktuellen als auch die vergangenen Zustände angezeigt, wobei die Unterschiede zwischen ihnen hervorgehoben werden.

Während der Benutzer manuell die verschiedenen Zustände des Modells untersucht, werden die ausgeführten Operationen in einer Trace-Datei erfasst. Diese Trace-Datei kann gespeichert werden, um die Simulation später mit denselben Parametern und durchgeführten Operationen erneut abzuspielen. Darüber hinaus ermöglicht die ProB2-UI mithilfe der VisB-Komponente [WL20] eine Visualisierung der Zustände sowie der ausgeführten Operationen, wodurch ein besserer Einblick in das Verhalten des Modells ermöglicht wird.

Des Weiteren beinhaltet ProB2-UI einen Simulator namens SimB [VLM21a], der es Benutzern ermöglicht, Echtzeitsimulationen, Monte-Carlo-Simulationen und statistische Analysen des Modells mit Hypothesentests durchzuführen. SimB basiert auf Dateien, in denen festgelegt wird, wie sich Operationen gegenseitig anhand der Zeit oder Wahrscheinlichkeiten

aktivieren [VLM21b]. Alle Simulationen können als „timed trace“ gespeichert werden und in Echtzeit nachgespielt werden.

3 Methodik und Implementierung

3.1 Environment

3.1.1 Highway-Environment

Die Grundlage, auf der das Training der Agenten für diese Bachelorarbeit aufbaut, ist das Highway-Environment [Leu18]. Die zentrale Problemstellung für den Agenten in dieser simulierten Umgebung besteht darin, auf einer mehrspurigen Autobahn zu navigieren. Das Hauptziel des Agenten ist es, eine möglichst hohe Geschwindigkeit aufrechtzuerhalten, ohne dabei Kollisionen mit anderen Verkehrsteilnehmern zu verursachen. Währenddessen soll die Maximalgeschwindigkeit von 40 m/s nicht überschritten werden. Zusätzlich wird der erzielte Reward für den Agenten erhöht, wenn dieser sich auf der rechten Spur bewegt. Um den Reward zu maximieren, stehen dem Agenten insgesamt fünf verschiedene Aktionen zur Verfügung, weshalb die Highway-Umgebung als diskret zu klassifizieren ist:

- Idle: Der Agent fährt auf der aktuellen Spur weiter und behält dabei seine Geschwindigkeit bei.
- Lane_Left: Der Agent fährt nach links.
- Lane_Right: Der Agent fährt nach rechts.
- Slower: Der Agent reduziert seine Geschwindigkeit.
- Faster: Der Agent beschleunigt.

Aufgrund der diskreten Umgebung ist es für den Agenten möglich, den Zusammenhang zwischen seinen Aktionen und den daraus resultierenden Belohnungen besser zu verstehen und darauf aufbauend optimale Strategien zu entwickeln. Einhergehend damit ist die Tatsache, dass das Highway-Environment eine vereinfachte Darstellung der realen Autobahnen darstellt und ein Agent, welcher auf dem Highway-Environment trainiert wurde, nicht ohne weiteres auf realen Autobahnen eingesetzt werden kann. Die Highway-Umgebung dient als Grundlage für das Training der Agenten, um die Generierung von Traces und deren Validierung mit SimB zu durchzuführen.

Das Highway-Environment besteht aus zwei wesentlichen Elementen: Zum einen dem Highway, der sich durch eine variable Anzahl paralleler Fahrspuren auszeichnet, wobei jede Spur eine Breite von vier Metern aufweist. Zum anderen umfasst die Umgebung die Fahrzeuge, die durch folgende charakteristische Merkmale gekennzeichnet sind:

- Fahrzeuglänge: 5 Meter.
- Fahrzeugbreite: 2 Meter.
- Maximalgeschwindigkeit: 40m/s.
- Minimalgeschwindigkeit: -40m/s.

Die Umgebung gibt nach jeder Aktion des Agenten den aktuellen Zustand der Fahrzeuge auf dem Highway zurück. Dazu zählen die Position, die Geschwindigkeit und die Beschleunigung jedes Fahrzeuges.

3.1.2 Anpassungen an das Environment

Um das Generieren der Traces und die Validierung des Agenten zu ermöglichen, wurden einige Anpassungen am Highway-Environment vorgenommen. Erstens wird eine schnellere Version des Highway-Environment „highway-fast-v0“ verwendet. Diese Änderung bringt folgende Unterschiede mit sich: Eine geringere Simulationsgeschwindigkeit, weniger Fahrzeuge sowie weniger Spuren und kürzere Episoden. Außerdem wird der Kollisionscheck zwischen Fahrzeugen ausschließlich für den Agenten ausgeführt. Diese Änderungen ermöglichen eine 15-fache Beschleunigung der Simulation. Zweitens wurde das Environment so konfiguriert, dass die Kinematic-Observation verwendet wird. Hierbei können die folgenden Eigenschaften für eine variable Anzahl an Fahrzeugen in einem Array gespeichert werden:

- Presence: Gibt an, ob das Fahrzeug existiert.
- Vx: Geschwindigkeit des Fahrzeuges in X-Richtung.
- Vy: Geschwindigkeit des Fahrzeuges in Y-Richtung.
- x: X-Koordinate.
- y: Y-Koordinate.

Die Anzahl der zu observierenden Fahrzeuge kann durch die Einstellung von „vehicles_count“ festgelegt werden, um eine feste Größe des Observation Arrays zu gewährleisten. In dieser Arbeit wird der Standardwert von fünf zu beobachtenden Fahrzeugen verwendet. Sollten sich weniger als fünf Fahrzeuge in der Szene befinden, werden die restlichen Reihen mit Nullen aufgefüllt und sind als Platzhalter anhand des Presence Wert zu erkennen. Da die Werte im Environment normalisiert werden, um eine bessere Leistung des Agenten zu erzielen, müssen diese danach wieder in leichter nachzuvollziehende Werte umgerechnet werden. Hierfür werden die ausgegebenen Werte des Environments mit folgenden Werten multipliziert. Der „Presence“ Wert wird mit 1 multipliziert, während „Vx“ und „Vy“ jeweils mit 80 vervielfacht werden. Des Weiteren wird „x“ mit 200 und „y“ mit 24 multipliziert.

3.2 Agent

3.2.1 Auswahl des Algorithmus

Um das Highway-Environment Problem zu lösen, wurden Agenten mittels Stable-Baselines3 [RHG+21] trainiert. Beim Training der Agenten wurde jeweils eine MlpPolicy verwendet. In jedem Lernprozess spielt die Policy des Agenten eine entscheidende Rolle. Die Policy ist eine Funktion, die dem Agenten angibt, welche Aktionen in welchem Zustand der Umgebung ausgeführt werden sollen, um das angestrebte Ziel zu erreichen. Policy-Gradient-Methoden sind eine Unterklasse von RL-Algorithmen, die den Gradienten der erwarteten kumulativen Belohnung in Bezug auf die Parameter der Policy direkt schätzen und optimieren. In anderen Worten, sie aktualisieren die Policy, indem sie den Gradienten der erwarteten Belohnung verwenden, um die Policy-Parameter zu verbessern. MlpPolicy steht für „Multilayer Perceptron Policy“ und ist eine Implementierung einer Policy, die auf einem Multilayer Perceptron (MLP) basiert. Ein MLP ist ein künstliches neuronales Netzwerk, das aus mehreren Schichten besteht, einer Eingabeschicht, einer oder mehrerer versteckter Schichten und einer Ausgabeschicht. Jede Schicht besteht aus einer Anzahl von Neuronen, die mit den Neuronen der benachbarten Schichten verbunden sind. Die Verbindungen zwischen den Neuronen haben Gewichte, die im Laufe des Trainings angepasst werden, um das gewünschte Verhalten des Netzwerks zu erreichen. Während des Trainings wird der Agent Aktionen entsprechend der aktuellen Policy ausführen und Belohnungen von der Umgebung erhalten. Die Policy-Gradient-Methode verwendet die gesammelten Erfahrungen, um den Gradienten der erwarteten kumulativen Belohnung in Bezug auf die Policy-Parameter (Gewichte des MLP) zu berechnen. Dann werden die Parameter entsprechend dem Gradienten aktualisiert, um die Policy zu verbessern und die erwartete kumulative Belohnung zu maximieren. Auf Grundlage der MlpPolicy wurden zwei verschiedene RL-Algorithmen verwendet, um die Agenten zu trainieren. Zum einen Deep Q-Network (DQN) und zum anderen der Proximal Policy Optimization (PPO) Algorithmus.

DQN. DQN (Deep Q-Network) [MKS+13] ist ein Algorithmus, der das Q-Learning-Verfahren mit tiefen neuronalen Netzen kombiniert. Entwickelt von DeepMind im Jahr 2013, stellt DQN einen bedeutenden Fortschritt in der Anwendung von RL auf hochdimensionale oder komplexe Aufgaben dar. Q-Learning ist ein wertebasiertes RL-Verfahren, bei dem ein Agent versucht, eine optimale Handlungsstrategie (auch als Q-Funktion bezeichnet) zu erlernen, indem er den erwarteten kumulativen Wert der Belohnungen abschätzt. In DQN wird ein tiefes neuronales Netz verwendet, um diese Q-Funktion zu approximieren. DQN implementiert zudem Techniken wie Erfahrungswiedergabe (experience replay) und Zielnetzwerke (target networks), um die Stabilität und Effizienz des Lernprozesses zu verbessern.

PPO. PPO [SWD+17] ist ein policy-basiertes Verfahren, das darauf abzielt, den Lernprozess zu optimieren, indem es den Policy-Gradienten-Ansatz verwendet. Anstatt direkt die

Q-Funktion zu schätzen, konzentriert sich PPO auf die Optimierung einer stochastischen Entscheidungsstrategie. PPO zeichnet sich durch seine einfache Implementierung und hohe Stabilität aus, da es ein besonderes Kriterium namens „Trust Region“ einführt, um große Schritte im Parameter-Update zu verhindern. Dieses Kriterium stellt sicher, dass die Aktualisierung der Entscheidungsstrategie nicht zu weit von der aktuellen Strategie entfernt ist, wodurch die Gefahr einer Destabilisierung des Lernprozesses reduziert wird.

DQN und PPO sollten sich also hervorragend zur Lösung des Highway-Environment-Problems eignen, da beide Algorithmen spezifische Stärken und Schwächen aufweisen, die sie in unterschiedlichen Situationen für die Lösung von RL-Aufgaben nützlich machen. DQN hat eine bewährte Leistung in einer Vielzahl von Anwendungen, insbesondere bei Spielen und diskreten Entscheidungsproblemen. Durch den Einsatz von Erfahrungswiedergabe und Zielnetzwerken kann DQN stabile und zuverlässige Lernergebnisse erzielen. Darüber hinaus ermöglicht die Verwendung von Deep Learning-Techniken zur Approximation der Q-Funktion die Anwendung von DQN auf größere und komplexere Probleme. Allerdings ist DQN für diskrete Aktionen konzipiert und eignet sich daher nicht für Umgebungen mit kontinuierlichen Aktionen. Zudem kann das Lernen in DQN aufgrund der Verwendung von Erfahrungswiedergabe langsamer sein als bei anderen RL-Algorithmen.

PPO hingegen zeichnet sich durch seine einfache Implementierung und geringere Notwendigkeit von Hyperparameter-Tuning im Vergleich zu anderen policy-basierten Methoden aus. Die Verwendung der Trust Region-Methode gewährleistet Stabilität im Lernprozess, ohne große Schwankungen in der Performance, wie in Abbildung 7 zu sehen ist. PPO eignet sich sowohl für diskrete als auch für kontinuierliche Aktionen, was es für eine breitere Palette von Umgebungen nützlich macht. Trotzdem kann PPO bei sehr großen und komplexen Umgebungen weniger effizient sein als DQN oder andere Algorithmen, die Deep Learning-Techniken verwenden. In einigen Fällen kann PPO auch suboptimale Lösungen erzeugen, insbesondere wenn das Problem hochgradig komplex ist.

Obwohl DQN und PPO gut geeignet sein sollten, um das Highway-Environment-Problem zu lösen, gibt es auch andere RL-Algorithmen, die man in Betracht ziehen könnte, abhängig von den spezifischen Anforderungen an das Verhalten des Agenten für die Lösung des Problems oder der gewünschten Lerngeschwindigkeit unter Berücksichtigung der verfügbaren Rechenressourcen. Beispiele dafür wäre A3C [MBM+16]

3.2.2 Training der Agenten

Nachdem die Auswahl der RL-Algorithmen getroffen wurde, wurden die Parameter der einzelnen Algorithmen, wie in Quellcode 1 und Quellcode 2 festgelegt:

Quellcode 1: DQN Training

```
1:  import gym
2:  import highway_env
3:  from stable_baselines3 import DQN
4:  from stable_baselines3.common.callbacks import CheckpointCallback
5:
6:  env = gym.make("highway-fast-v0")
7:  model = DQN('MlpPolicy',
8:    env,
9:    policy_kwargs=dict(net_arch=[256, 256]),
10:   learning_rate=5e-4,
11:   buffer_size=15000,
12:   learning_starts=200,
13:   batch_size=32,
14:   gamma=0.8,
15:   train_freq=1,
16:   gradient_steps=1,
17:   target_update_interval=50,
18:   verbose=1,
19:   tensorboard_log="highway_dqn/highway_dqn/DQNFast/")
```

- policy_kwargs: Ein Wörterbuch, das Schlüssel-Wert-Paare für zusätzliche Argumente enthält, die an die Policy weitergegeben werden. In diesem Fall ist net_arch=[256, 256], was bedeutet, dass das neuronale Netzwerk zwei versteckte Schichten mit jeweils 256 Neuronen hat.
- learning_rate: Die Lernrate ist ein essenzieller Hyperparameter, der die Geschwindigkeit des Lernprozesses eines Modells während des Trainings beeinflusst. Eine geringere Lernrate resultiert in einer langsameren Anpassung der Modellparameter und damit in einem verlangsamten Lernprozess. Im Gegensatz dazu führt eine höhere Lernrate zu einer schnelleren Anpassung der Modellparameter und einem beschleunigten Lernprozess. Allerdings besteht bei einer zu hohen Lernrate die Gefahr, dass das Modell nicht konvergiert und somit keine stabile Lösung für das zugrundeliegende Problem erreicht wird.
- buffer_size: Die Größe des Puffer- bzw. Erfahrungswiedergabespeichers. Dieser Puffer speichert Erfahrungen (Zustände, Aktionen, Belohnungen, nächste Zustände usw.) und wird zum Sammeln von Trainingsdaten verwendet.
- learning_starts: Die Anzahl der Schritte, die der Agent absolvieren muss, bevor der eigentliche Lernprozess einsetzt. Diese Vorgehensweise ermöglicht es dem Agenten, eine ausreichende Menge an Erfahrungen zu sammeln, bevor das Training initiiert wird.
- batch_size: Die Anzahl der Erfahrungen, die aus dem Puffer ausgewählt und gleichzeitig für das Training verwendet werden. Eine größere Batch-Size führt in der Regel

zu stabileren Gradienten, erhöht jedoch auch die Rechenanforderungen.

- gamma: Der Diskontfaktor bestimmt, wie stark zukünftige Belohnungen im Vergleich zu aktuellen Belohnungen gewichtet werden. Ein Wert nahe 0 bedeutet, dass der Agent kurzfristige Belohnungen bevorzugt, während ein Wert nahe 1 bedeutet, dass der Agent langfristige Belohnungen bevorzugt.
- train_freq: Die Trainingsfrequenz gibt an, wie oft das Modell trainiert wird. In diesem Fall wird das Modell nach jedem Schritt trainiert.
- gradient_steps: Die Anzahl der Gradienten-Updates, die bei jedem Trainingsschritt durchgeführt werden. Ein Gradienten-Update ist ein Schritt, bei dem die Parameter des Modells aktualisiert werden.
- target_update_interval: Das Intervall, in dem das Zielnetzwerk aktualisiert wird. Das Zielnetzwerk wird verwendet, um stabile Schätzungen der Q-Werte während des Trainings zu erhalten.
- verbose: Der Informatiosgehalt, welcher zurückgegeben wird. Ein Wert von 1 bedeutet, dass grundlegende Informationen ausgegeben werden. Ein Wert von 2 würde bedeuten, dass Debugnachrichten ausgegeben werden.

Quellcode 2: PPO Training

```
1:  import gym
2:  import highway_env
3:  from stable_baselines3 import PPO
4:  from stable_baselines3.common.callbacks import CheckpointCallback
5:
6:  env = gym.make("highway-fast-v0")
7:  model = PPO("MlpPolicy",
8:    env,
9:    policy_kwargs=dict(net_arch=[dict(pi=[256, 256], vf=[256, 256])]),
10:   n_steps= 2,
11:   batch_size=32,
12:   n_epochs=10,
13:   learning_rate=5e-4,
14:   gamma=0.8,
15:   verbose=1,
16:   tensorboard_log="highway_dqn/highway_dqn/PPOFast/")
```

- policy_kwargs: Ein Wörterbuch, das Schlüssel-Wert-Paare für zusätzliche Argumente enthält, die an die Policy weitergegeben werden. In diesem Fall ist net_arch=[dict(pi=[256, 256], vf=[256, 256])] angegeben, was bedeutet, dass sowohl das Aktor-Netzwerk (pi) als auch das Kritiker-Netzwerk (vf) zwei versteckte Schichten mit jeweils 256 Neuronen haben.

- n_steps: Die Anzahl der Schritte, die der Agent für das Rollout durchführt, bevor er die Gradienten berechnet und das Modell aktualisiert. Längere Rollouts (größere n_steps) führen im Allgemeinen zu stabileren Gradienten, erhöhen jedoch auch die Varianz.

- batch_size: Die Anzahl der Erfahrungen, die aus dem Puffer ausgewählt und gleichzeitig für das Training verwendet werden. Eine größere Batch-Size führt in der Regel zu stabileren Gradienten, erhöht jedoch auch die Rechenanforderungen.

- n_epochs: Die Anzahl der Epochen, in denen das Modell trainiert wird. Eine Epoche ist ein vollständiger Durchlauf durch den gesamten Trainingsdatensatz. Mehr Epochen führen in der Regel zu einem besser trainierten Modell, können jedoch zu Überanpassung führen.

- learning_rate: Die Lernrate ist ein essenzieller Hyperparameter, der die Geschwindigkeit des Lernprozesses eines Modells während des Trainings beeinflusst. Eine geringere Lernrate resultiert in einer langsameren Anpassung der Modellparameter und damit in einem verlangsamten Lernprozess. Im Gegensatz dazu führt eine höhere Lernrate zu einer schnelleren Anpassung der Modellparameter und einem beschleunigten Lernprozess. Allerdings besteht bei einer zu hohen Lernrate die Gefahr, dass das Modell

nicht konvergiert und somit keine stabile Lösung für das zugrundeliegende Problem erreicht wird.

- gamma: Der Diskontfaktor bestimmt, wie stark zukünftige Belohnungen im Vergleich zu aktuellen Belohnungen gewichtet werden. Ein Wert nahe 0 bedeutet, dass der Agent kurzfristige Belohnungen bevorzugt, während ein Wert nahe 1 bedeutet, dass der Agent langfristige Belohnungen bevorzugt.

- verbose: Der Informationsgehalt, welcher zurückgegeben wird. Ein Wert von 1 bedeutet, dass grundlegende Informationen ausgegeben werden. Ein Wert von 2 würde bedeuten, dass Debugnachrichten ausgegeben werden.

3.2.3 Performance Evaluation und Reproduzierbarkeit

Damit das Training von eigenen Agenten auf der Highway-Umgebung mit vergleichbaren Ergebnissen reproduziert werden kann, wurde das Training des DQN-Agenten unter Verwendung der Hyperparameter aus Quellcode 1 implementiert. Das Training des DQN-Agenten erstreckte sich über 500.000 Schritte, dauerte etwa 6 Stunden und führte zu den in Abbildung 3, Abbildung 4 und Abbildung 5 dargestellten Ergebnissen.

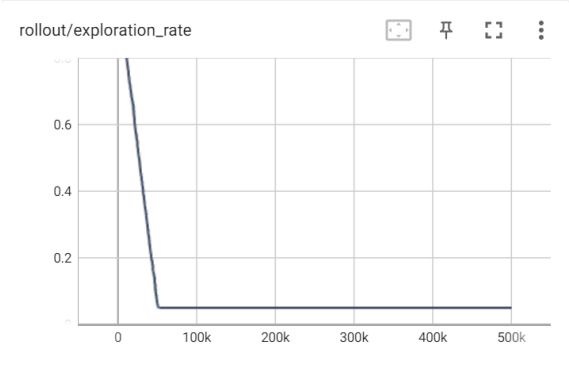

Abbildung 3: Exploration Rate Dqn Agent

Während der ersten 80.000 Schritte zeigt der Agent aufgrund der hohen Explorationsrate ein beschleunigtes Lernverhalten. Nachdem die Explorationsrate abflacht, ist zu erkennen, dass das sowohl die Episodenlänge, als auch der Reward nur noch relativ kleine Schwankungen aufweisen. Am Ende des Trainings erreicht der Agent eine durchschnittliche Episodenlänge von 25 Sekunden und einen durchschnittlichen Reward von 23.

Der PPO-Agent wurde unter Anwendung der Hyperparameter aus Quellcode 2 trainiert und erzielte nach einer Trainingsdauer von ebenfalls 500.000 Schritten folgende Ergebnisse. Nachdem der Agent schnell eine durchschnittliche Episodenlänge von 30 Sekunden aufwies und einen durchschnittlichen Reward von ungefähr 21 erzielte, gab es ähnlich wie beim

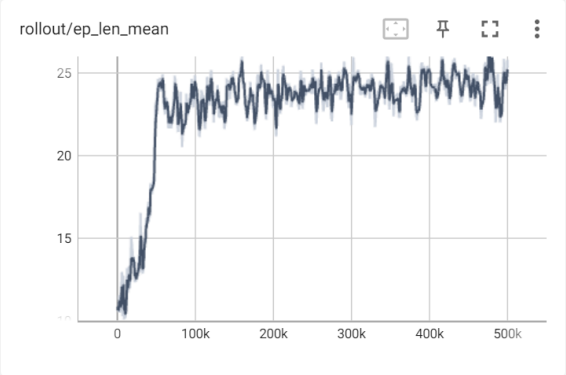

Abbildung 4: Durchschnittliche Episoden Länge DQN-Agent

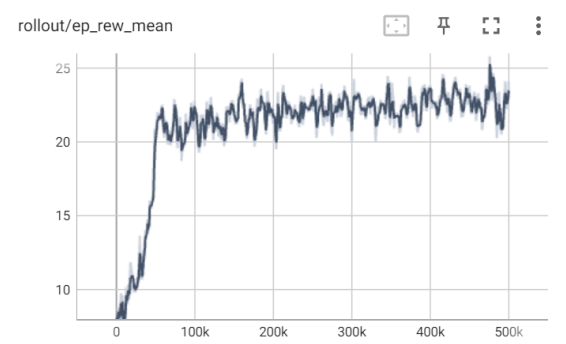

Abbildung 5: Durchschnittlicher Episoden Reward DQN-Agent

DQN-Agenten wenig Änderungen in den Ergebnissen. Zum Vergleich siehe Abbildung 6 und Abbildung 7.

Die in den Abbildungen Abbildung 3, Abbildung 4, Abbildung 5, Abbildung 6 und Abbildung 7 dargestellten Ergebnisse lassen darauf schließen, dass sowohl der DQN- als auch der PPO-Agent in der Lage waren, die Highway-Umgebung erfolgreich zu meistern. Während der ersten 80.000 Schritte ermöglichte die hohe Explorationsrate den Agenten, das Modell schnell anzupassen. Die erzielte durchschnittliche Episodenlänge von 25 Sekunden und der durchschnittliche Reward von 23 für den DQN-Agenten legen nahe, dass der Agent eine gewisse Stabilität und Effizienz in seiner Leistung erreicht hat. Ähnliche Ergebnisse wurden für den PPO-Agenten erzielt, was darauf hindeutet, dass beide Algorithmen in dieser speziellen Umgebung effektiv genutzt werden können.

Allerdings war der erste Trainingsdurchlauf nicht erfolgreich genug, um die gewünschten Ergebnisse für den PPO-Agenten zu erzielen.

Durch die Analyse der generierten Traces konnte bei suboptimalen Trainingsdurchläufen festgestellt werden, dass die trainierten Agenten gelegentlich dazu neigten, ausschließlich

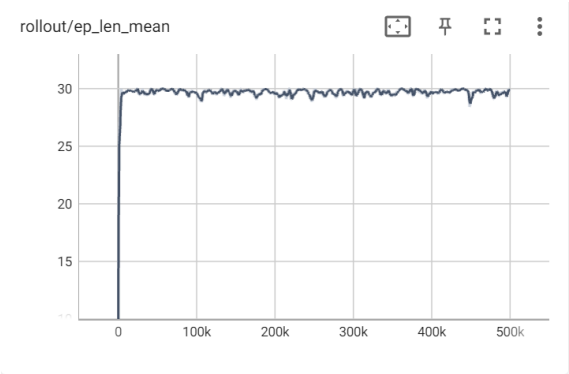

Abbildung 6: Durchschnittliche Episoden Länge PPO-Agent

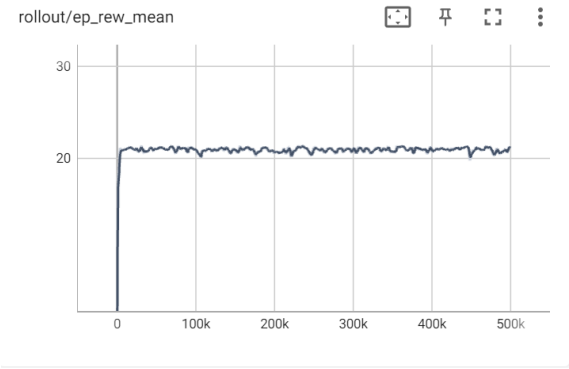

Abbildung 7: Durchschnittlicher Episoden Reward PPO-Agent

„Idle“ oder „Slower“ Aktionen auszuführen. Dieses Verhalten führte zwar zu einer Maximierung der Episodenlänge, jedoch nicht zu einem optimalen Reward. So stellte sich auch nach der Analyse des ersten PPO-Agenten heraus, dass dieser in einigen Episoden ausschließlich „Slower“ Aktionen ausführte und versuchte auf diese Weise seinen Reward zu maximieren. Daraufhin wurde ein neuer PPO-Agent über eine Schrittzahl von 1 Million trainiert, um bessere Ergebnisse zu erzielen. Die Ergebnisse dieses Trainings sind in Abbildung 8 und Abbildung 9 dargestellt. Es ist zu erkennen, dass der Agent in diesem Fall mit einer Episodenlänge von knapp unter 25 und einem Reward von 20 ähnlichere Ergebnisse zum DQN-Agenten erzielt als der erste PPO-Agent.

Um die Leistung der Agenten weiter zu optimieren, besteht die Möglichkeit, weitere Kombinationen von Hyperparametern zu testen, um beispielsweise eine konstantere Lernkurve zu erreichen. Darüber hinaus kann die Highway-Umgebung modifiziert werden, um die Robustheit der Agenten in unterschiedlichen Szenarien zu testen. Durch die Durchführung des Trainings mit unterschiedlichen Variationen der Parameter kann ein tieferes Verständnis für die Leistung und die Fähigkeiten der trainierten Agenten gewonnen werden.

Zum Reproduzieren der Ergebnisse der in diesem Projekt entwickelten DQN- und PPO-

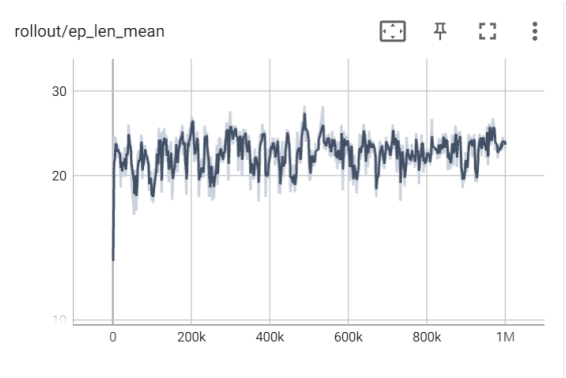

Abbildung 8: Durchschnittliche Episoden Länge PPO-Agent mit 1 Million Schritte

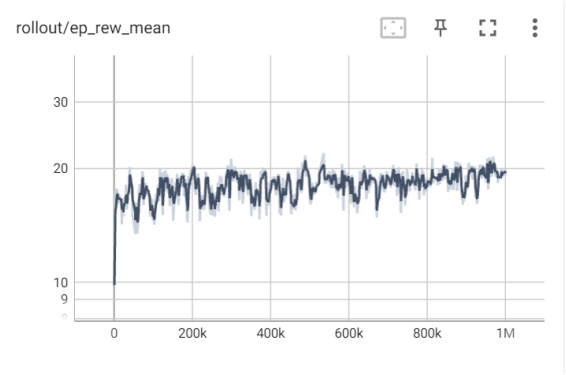

Abbildung 9: Durchschnittlicher Episoden Reward PPO-Agent mit 1 Million Schritte

Agenten, können die TrainAgent- und CreateTraces-Dateien verwendet werden. Die TrainAgent-Dateien trainieren die jeweiligen Agenten, während die CreateTraces-Dateien die Agenten in einer vorgegebenen Anzahl von Episoden durch das Highway-Environment laufen lassen und Traces erstellen. Für das Training der Agenten stehen zwei separate Dateien zur Verfügung: TrainAgent_DQN.py und TrainAgent_PPO.py. Durch das Ausführen einer dieser Dateien wird der entsprechende Agent für eine festgelegte Anzahl von Schritten trainiert. Während des Trainingsprozesses können die Statistiken des Agenten mithilfe von Tensorboard beobachtet und analysiert werden. Nach Abschluss des Trainings können die erstellten Agenten mithilfe der CreateTraces-Dateien, CreateTraces_DQN.py und CreateTraces_PPO.py, eingesetzt werden, um Traces für eine beliebige Anzahl von Episoden zu erzeugen. Diese Traces können anschließend sowohl für SimB als auch für ProB2-UI verwendet werden. Damit die korrekte Funktionalität der Programme sichergestellt ist, ist es notwendig, alle erforderlichen Libraries und ihre entsprechenden Versionen zu installieren. Eine requirements.txt-Datei wird bereitgestellt, um diesen Prozess zu erleichtern. Die Installation der benötigten Libraries erfolgt durch den Befehl pip install -r requirements.txt. Die Programme wurden unter Windows mit Python 3.9.5 entwickelt und getestet. Die wichtigsten Libraries, welche für das Training der Agenten verwendet wurden, sind Highway_env 1.5, Stable-Baselines3

1.7.0 und OpenAI Gym 0.21.0. Mithilfe dieser Anleitung können die Ergebnisse der Agenten reproduziert werden, wodurch eine eigenständige Analyse und Weiterentwicklung der Agenten ermöglicht wird. Zugriff zu den Dateien ist unter folgendem Link möglich: https://gitlab.cs.uni-duesseldorf.de/stups/abschlussarbeiten/davin-holten-bachelor

3.3 Traces

3.3.1 Zu validierende Eigenschaften

Im ersten Teil des Experiments wurde ein künstlicher Agent in einer simulierten vereinfachten Umgebung einer Autobahn trainiert, um autonomes Fahren zu erlernen. Daraufhin wird nach Abschluss des Trainingsprozesses die Leistung des Agenten im Hinblick auf bestimmte Kriterien während einer Episode analysiert. Diese Kriterien waren nicht explizit Teil des Trainingsprozesses, sondern dienen als zusätzliche Evaluationsmaßnahmen, um einen umfangreicheren Einblick in das Verhalten des Agenten zu erhalten. Um dies zu ermöglichen, werden während einer Episode bei jedem Schritt des Agenten in der Umgebung die relevanten Daten sowohl für das Ego-Fahrzeug (den Agenten) als auch für die vier nächstgelegenen Fahrzeuge in sogenannten Traces erfasst und gespeichert.

Durch die Verwendung dieser gesammelten Daten kann das Verhalten des Agenten unter Berücksichtigung der folgenden Aspekte validiert werden:

- Minimale Geschwindigkeit in x-Richtung: Der Agent sollte stets eine Mindestgeschwindigkeit von 10 m/s beibehalten, um den Verkehrsfluss nicht unnötig zu beeinträchtigen.
- Maximale Geschwindigkeit in x-Richtung: Der Agent sollte eine Höchstgeschwindigkeit von 30 m/s nicht überschreiten, um die Sicherheit und Einhaltung von Geschwindigkeitsbegrenzungen zu gewährleisten.
- Gleichmäßiges Abbremsen: Der Agent sollte mit maximal 3 m/s abbremsen, um abrupte Bremsmanöver und das Risiko von Auffahrunfällen zu minimieren.
- Gleichmäßiges Beschleunigen: Der Agent sollte eine Beschleunigung von 5 m/s nicht überschreiten, um ein ruckartiges Fahrverhalten zu verhindern.
- Mindestabstand zu anderen Fahrzeugen in x-Richtung: Der Agent sollte einen Sicherheitsabstand von mindestens 10 m (entsprechend zwei Fahrzeuglängen) zu anderen Fahrzeugen einhalten, um ausreichend Reaktionszeit und Platz für Brems- und Ausweichmanöver zu ermöglichen.
- Mindestabstand zu anderen Fahrzeugen in y-Richtung: Der Agent sollte einen seitlichen Sicherheitsabstand von mindestens 2 m (entsprechend einer Fahrzeugbreite) zu anderen Fahrzeugen einhalten, um Kollisionen beim Spurwechsel oder Ausweichen zu verhindern.

- Vermeidung von Unfällen: Der Agent sollte jegliche Kollisionen mit anderen Fahrzeugen vermeiden, um die Sicherheit aller Verkehrsteilnehmer zu gewährleisten.

Die Analyse dieser Aspekte ermöglicht eine differenziertere Beurteilung der Leistungsfähigkeit und Sicherheit des Agenten in der Highway-Umgebung, verglichen mit der Betrachtung von Abbildung 4, Abbildung 5 und Abbildung 6 sowie Abbildung 7, welche lediglich die durchschnittliche Episodenlänge und den durchschnittlichen Reward abbilden. Dadurch tragen diese Eigenschaften dazu bei, mögliche Schwachstellen oder Verbesserungspotenziale aufzuzeigen.

3.3.2 Extraktion und Aufbau

Um die Traces aus den Kinematic-Observations des Highway-Environments zu generieren, wurden zuerst die Observations des Environments in eine Liste gepackt. Für das Nachspielen und Analysieren der Traces in SimB, muss ein bestimmtes Format eingehalten werden. Dafür wurden die Werte aus den Observations jeweils für Prob2-UI und SimB in json Dateien geschrieben. Der Inhalt der einzelnen Traces sieht wie folgt aus: In SimB [VLM21a] enthält ein Trace ein „activations field“, welches für jeden Schritt, den der Agent im Environment macht folgende Elemente für die Simulation enthält:

- execute: Ist der Name des ausgeführten Events.
- after: Gibt die Zeit an, nach der ein Event ausgeführt wird.
- priority: Gibt die Priorität des Events an, desto niedriger die Zahl, desto höher die Priorität.
- additionalGuards: Speichert optionale Bedingungen, die beim Ausführen des Ereignisses, das in „execute“ gespeichert ist, geprüft werden.
- activationKind: Speichert die Art der Aktivierung. Der Default Wert ist „multi“, das heißt die Aktivierung wird in die Warteschleife eingereiht.
- fixedVariables: Speichert eine Map mit Variablen, welche einen Wert zugewiesen bekommen.
- activating: Gibt an, welches Event als nächstes aktiviert werden soll.
- id: Definiert eine eindeutige ID für das Event.

In Abbildung 10 ist ein Ausschnitt eines Trace für SimB dargestellt. Zu erkennen ist ein „execute“ Schritt, der mit den Rückgabewerten der Observation gefüllt ist. Allerdings muss die Maschine, bevor die eigentlichen Events (Aktionen) des Agenten im SimB-Trace nachgespielt werden können, mit „initiliase_maschine“ und „setup_constants“ aufgesetzt werden, siehe Abbildung 11.

```
{
  "execute": "FASTER",
  "after": "300",
  "priority": 0,
  "additionalGuards": null,
  "activationKind": null,
  "fixedVariables": {
    "Crash": "FALSE",
    "Critical_DistanceX": "TRUE",
    "Critical_DistanceY": "TRUE",
    "VehiclesVx": "{EgoVehicle|->29.341552257537842, Vehicles2|->-11.672801971435547, Vehicles3|->-9.152221083641052, Vehicles4|->-17.10389018058777, Vehicles5|->-16.16421103477478}",
    "VehiclesY": "{EgoVehicle|->7.986992597579956, Vehicles2|->-7.986992597579956, Vehicles3|->5.627161860466003, Vehicles4|->-7.986992597579956, Vehicles5|->-7.986992597579956}",
    "VehiclesVy": "{EgoVehicle|->0.027121149469166994, Vehicles2|->-0.027121149469166994, Vehicles3|->3.3195245265960693, Vehicles4|->-0.027121149469166994, Vehicles5|->-0.027121149469166994}",
    "VehiclesX": "{EgoVehicle|->200.0, Vehicles2|->9.07636284828186, Vehicles3|->36.255013942718506, Vehicles4|->54.70516085624695, Vehicles5|->75.74909329414368}"
  },
  "probabilisticVariables": null,
  "activating": [
    "LANE_RIGHT3"
  ],
  "id": "FASTER2"
},
```


Abbildung 10: Beispiel eines Trace Ausschnittes

```
{
  "execute": "$setup_constants",
  "after": "0",
  "priority": 0,
  "additionalGuards": null,
  "activationKind": null,
  "fixedVariables": {},
  "probabilisticVariables": null,
  "activating": [
    "$initialise_machine"
  ],
  "id": "$setup_constants"
},
{
  "execute": "$initialise_machine",
  "after": "0",
  "priority": 0,
  "additionalGuards": null,
  "activationKind": null,
  "fixedVariables": {
    "Crash": "FALSE",
    "Critical_DistanceX": "FALSE",
    "Critical_DistanceY": "FALSE",
    "VehiclesVx": "{EgoVehicle|->25.0, Vehicles2|->-3.642393946647644, Vehicles3|->-3.5900506377220154, Vehicles4|->-2.1305933594703674, Vehicles5|->-3.2839560508728027}",
    "VehiclesY": "{EgoVehicle|->0.0, Vehicles2|->0.0, Vehicles3|->8.000000238418579, Vehicles4|->0.0, Vehicles5|->0.0}",
    "VehiclesVy": "{EgoVehicle|->0.0, Vehicles2|->0.0, Vehicles3|->0.0, Vehicles4|->0.0, Vehicles5|->0.0}",
    "VehiclesX": "{EgoVehicle|->152.1647810935974, Vehicles2|->21.60925418138504, Vehicles3|->46.55172526836395, Vehicles4|->71.64013385772705, Vehicles5|->94.23356056213379}"
  },
  "probabilisticVariables": null,
  "activating": [
    "LANE_RIGHT1"
  ],
  "id": "$initialise_machine"
},
```


Abbildung 11: Anfang eines Trace

3.3.3 Validierung mit SimB und Ergebnisse

Nach der Durchführung des Trainings konnten die Traces innerhalb der ProB2-UI mithilfe von SimB validiert werden. Für den Erhalt aussagekräftiger Ergebnisse bezüglich des Verhaltens der Agenten wurden pro Agent 1000 Traces generiert. Für jede zu validierende Eigenschaft wurde ein Test innerhalb von SimB erstellt. Es ist wichtig zu beachten, dass die validierten Eigenschaften nicht Bestandteil des Trainings der Agenten selbst waren. Die ausgewählten Eigenschaften sind vielmehr dazu da, die Übersicht über das Verhalten der Agenten zu erweitern, um Potenzial für Verbesserungen aufzuzeigen.

Bei der Erstellung der Tests in SimB können verschiedene Eigenschaften oder Bedingungen festgelegt werden. In der Auswahl der Bedingungen geht es darum, wann eine Eigenschaft des Agenten innerhalb eines Trace überprüft werden soll. Es können Startbedingungen, eine maximale Anzahl an Schritten vor Beginn der Überprüfung und Endbedingungen für die Überprüfung der Eigenschaften festgelegt werden. Start- und Endbedingungen können

dabei als Zeitpunkt, Schrittanzahl oder Prädikat angegeben werden.

Bei der Validierung kann festgelegt werden, welche Eigenschaften der Agenten überprüft werden sollen. In dieser Arbeit wurden zwei Arten der Überprüfung der Eigenschaften verwendet: erstens, die Prüfung von Prädikaten innerhalb der Traces und zweitens, die Verwendung von Timing-Tests, um zu untersuchen, wie lange der Agent bestimmte Eigenschaften einhält oder verletzt.

Die folgenden Eigenschaften wurden als Prädikate überprüft:

- Minimale Geschwindigkeit in x-Richtung: Der Agent sollte stets eine Mindestgeschwindigkeit von 10 m/s beibehalten, um den Verkehrsfluss nicht unnötig zu beeinträchtigen.
- Maximale Geschwindigkeit in x-Richtung: Der Agent sollte eine Höchstgeschwindigkeit von 30 m/s nicht überschreiten, um die Sicherheit und Einhaltung von Geschwindigkeitsbegrenzungen zu gewährleisten.
- Gleichmäßiges Abbremsen: Der Agent sollte mit maximal 3 m/s abbremsen, um abrupte Bremsmanöver und das Risiko von Auffahrunfällen zu minimieren.
- Gleichmäßiges Beschleunigen: Der Agent sollte eine Beschleunigung von 5 m/s nicht überschreiten, um ein ruckartiges Fahrverhalten zu verhindern.
- Mindestabstand zu anderen Fahrzeugen in x-Richtung: Der Agent sollte einen Sicherheitsabstand von mindestens 10 m (entsprechend zwei Fahrzeuglängen) zu anderen Fahrzeugen einhalten, um ausreichend Reaktionszeit und Platz für Brems- und Ausweichmanöver zu ermöglichen.
- Mindestabstand zu anderen Fahrzeugen in y-Richtung: Der Agent sollte einen seitlichen Sicherheitsabstand von mindestens 2 m (entsprechend einer Fahrzeugbreite) zu anderen Fahrzeugen einhalten, um Kollisionen beim Spurwechsel oder Ausweichen zu verhindern.
- Vermeidung von Unfällen: Der Agent sollte jegliche Kollisionen mit anderen Fahrzeugen vermeiden, um die Sicherheit aller Verkehrsteilnehmer zu gewährleisten.

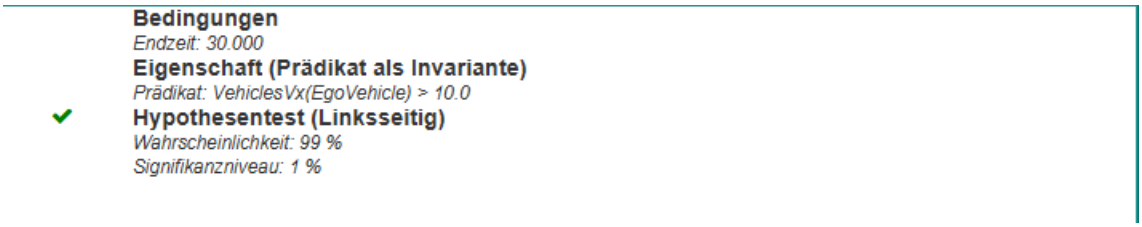

Abbildung 12: Beispiel eines Tests für Prädikate

In Abbildung 12 ist ein Beispiel für einen Test mit Prädikaten dargestellt. Die Endzeit des Tests wurde auf 30.000 festgelegt, was 30 Sekunden und somit der Dauer einer Episode

entspricht. Die getestete Eigenschaft betrifft die Einhaltung der minimalen Geschwindigkeit in x-Richtung in mindestens 99 Prozent der Traces.

Zweitens wurde mit Timing Tests die Möglichkeit genutzt, zu testen, wie lange der Agent eine Eigenschaft einhält oder verletzt. Dazu wurde die Zeit gemessen, die der Agent benötigt, um sich von einer Verletzung einer Eigenschaft zu erholen, oder wie lange er eine Eigenschaft einhält, bevor er sie verletzt. Die folgenden Eigenschaften wurden mit Timing Tests überprüft:

- Wie lange hält der Agent eine optimale Geschwindigkeit von 20 bis 30 m/s ein?
- Wie lange bremst oder beschleunigt der Agent mit einer kritischen Geschwindigkeitsabnahme oder -zunahme?
- Wie lange wird der minimale Abstand zu anderen Fahrzeugen in x- und y-Richtung verletzt?

Abbildung 13: Beispiel eines Tests für Timing Eigenschaften

Abbildung 13 zeigt einen Test für Timing Eigenschaften. Die Endzeit des Tests wurde auf aufgrund der Dauer einer Episode ebenfalls auf 30.000 gesetzt. Es wurde getestet, ob der Agent die Eigenschaft einer optimalen Geschwindigkeit von 20 bis 30 m/s für 20 Sekunden in einem Trace einhält.

Die Ergebnisse der Validierung der Traces des DQN-Agenten zeigen, dass der Agent die Eigenschaften der minimalen und maximalen Geschwindigkeit in x- und y-Richtung in 99 Prozent der Traces einhält. Ebenfalls werden eine gleichmäßige Beschleunigung und gleichmäßiges Abbremsen zu 95 Prozent eingehalten. Allerdings werden die Eigenschaften des minimalen Abstands zu anderen Fahrzeugen in x- und y-Richtung häufig verletzt. Das führte zu einer Unfallquote von knapp 16 Prozent. Anhand der Abbildung 14, Abbildung 15 lässt sich der Unfallvorgang des Agenten mit einem anderen Fahrzeug nachvollziehen. Wie zu sehen ist, kollidiert der Agent mit dem sich von ihm aus rechts befindendem Fahrzeug. Die Kollisionen lassen sich auch innerhalb der Traces erkennen, da sich die Geschwindigkeit des Agenten abrupt ändert. Zusätzlich ist zu erkennen, dass sich der Agent und das kollidierende Fahrzeug in einem y-Bereich von 0 bis 2 m sowie einem x-Bereich von 0 bis 5 m befinden.

Des Weiteren zeigt sich, dass die Fahrzeuge bei einem Unfall voneinander abprallen und sich nicht ineinander verkeilen, wie es in der Realität der Fall sein könnte. Dies führt dazu, dass bei Auffahrunfällen die x-Koordinaten der Fahrzeuge mindestens 5 m betragen. Das liegt daran, dass die gemessene Koordinate der Fahrzeuge den Mittelpunkt dieser darstellt und somit für jedes Fahrzeug eine Länge von 2,5 m addiert werden muss. Bei der Auswertung der Timing Eigenschaften, spiegeln sich die Ergebnisse der ersten Tests wider. Der Agent hält die Geschwindigkeitseigenschaften sehr gut ein, verletzt aber die minimalen Abstandsregeln über einen längeren Zeitraum.

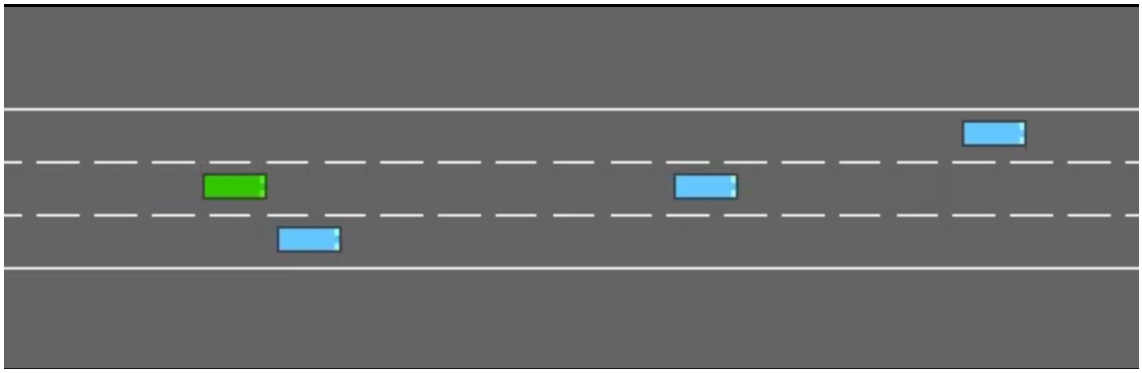

Abbildung 14: Vor dem Crash

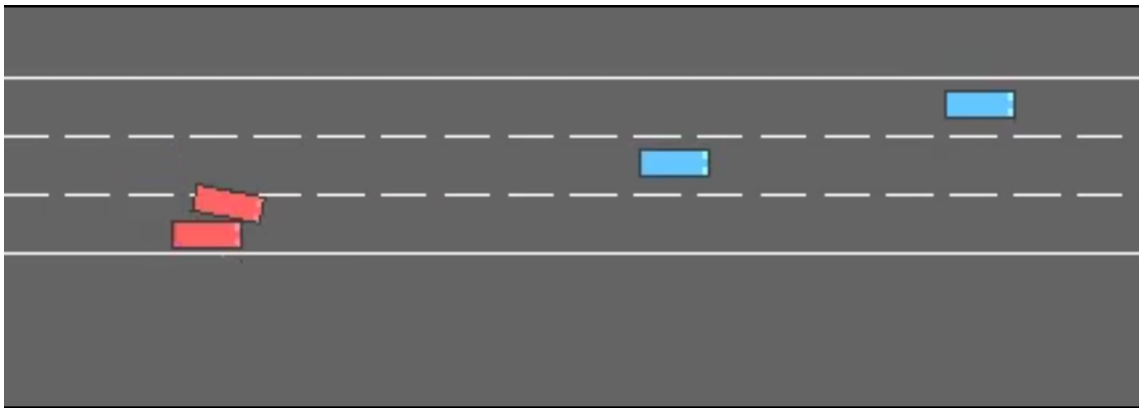

Abbildung 15: Beispiel eines Unfalles

Bei der Evaluation der PPO-Agenten konnten folgende Ergebnisse über das Verhalten der Agenten festgestellt werden. Der erste Agent konnte aufgrund von vielen „Slower“Aktionen eine Unfallqoute von knapp 2.5 Prozent erreichen. Obwohl dies ein sehr gutes Ergebnis ist, werden die Erwartungen an den Agenten in der Highway-Umgebung nicht erfüllt. Der zweite PPO-Agent Schnitt mit Unfällen in 46 Prozent der Traces eher unterdurchschnittlich ab und konnte auch in den anderen Tests aufgrund der hohen Unfallquote nicht überzeugen.

4 Ergebnisse der Forschung

Die Anwendung von SimB auf die generierten Traces deutet darauf hin, dass vor allem der DQN-Agent seine Stärken primär im Bereich der Geschwindigkeitskontrolle aufweist. Allerdings besteht Verbesserungspotenzial hinsichtlich des Verhaltens des Agenten bei der Einhaltung der Sicherheitsabstände zu anderen Fahrzeugen. Da die Validierung des Agenten auf Eigenschaften abzielte, die nicht explizit Teil des Trainings waren, ist dieses Ergebnis nicht überraschend. Im Vergleich zum DQN-Agenten haben sich die PPO-Agenten in der Highway-Umgebung schwer getan. Ursache dafür könnte sein, dass PPO teilweise in komplexeren Umgebungen nur schlecht eine optimale Lösung findet. Deshalb kann, um die gewünschten Eigenschaften im Trainingsprozess zu fördern, mit verschiedenen Techniken gearbeitet werden:

Modifizierung der Reward Funktion: Die Reward Funktion kann so modifiziert werden, dass sie die gewünschten Eigenschaften berücksichtigt. Zum Beispiel kann der Agent für gleichmäßiges Abbremsen und Beschleunigen sowie das Einhalten der Sicherheitsabstände belohnt werden. Einhergehend damit kann der Agent einen negativen Reward erzielen, wenn er die Geschwindigkeitsbegrenzungen überschreitet oder zu abrupt bremst oder beschleunigt.

Curriculum Learning [BLCW09]: Dabei wird der Agent zunächst mit einfacheren Szenarien konfrontiert, bevor er sich schrittweise zu komplexeren Situationen vorarbeitet. Auf diese Weise kann der Agent die gewünschten Verhaltensweisen schrittweise erlernen und festigen.

Verwendung von Imitation Learning [HE16]: Bei dieser Methode lernt der Agent, indem er das Verhalten eines Experten (z.B. eines menschlichen Fahrers oder eines Regel-basierten Systems) imitiert. Dies kann dazu beitragen, dass der Agent die gewünschten Eigenschaften schneller erlernt und sein Verhalten an die vorgegebenen Regeln anpasst.

Die Kombination dieser Techniken kann dazu beitragen, dass der Agent die gewünschten Eigenschaften effektiv im Training erlernt und umsetzt, um ein sicheres und effizientes Fahrverhalten zu gewährleisten.

Ziel der Arbeit war es, ein abstraktes formales Modell zu schreiben und die Simulation von der KI durchführen zu lassen. Die daraus resultierenden Traces wurden dann von SimB auf verschiedene Eigenschaften überprüft, ohne dafür eine eigene Simulation schreiben zu müssen. Die Validierung abstrakter formaler Modelle ist ein wichtiger Aspekt in der Entwicklung von Reinforcement-Learning Agenten. Deshalb werden die Vor- und Nachteile dieser neuen Methode abgewogen, um ihre Eignung als Alternative zu herkömmlichen Validierungsansätzen zu bewerten.

Der Hauptteil der Validierung mittels SimB und Trace Replay besteht darin, dass die Simulation des abstrakten Modells an die Künstliche Intelligenz ausgelagert wird. Dies erhöht die Flexibilität und ermöglicht die Validierung auf einer höheren Abstraktionsebene, da die Logik hinter den Operationen lediglich der KI bekannt ist. Dies gestattet den Agenten, schnell neue Eigenschaften zu überprüfen. Oft reicht es aus, die Prädikate oder

Timing-Eigenschaften zu ändern, um einen neuen Test zu entwickeln. Dies stellt einen bedeutenden Vorteil gegenüber herkömmlichen Validierungsansätzen dar, bei denen für jede neue Eigenschaft eine separate Simulation erstellt werden muss.

Die Auslagerung der Simulation an die KI trägt zudem zur Effizienzsteigerung bei und ermöglicht die Validierung großer und komplexer Modelle vergleichsweise in kurzer Zeit. Darüber hinaus kann der manuelle Aufwand bei der Validierung reduziert und somit menschliche Fehler minimiert werden. In Bezug auf den zeitlichen Aspekt lassen sich Traces und die zugehörigen Tests innerhalb von SimB schnell erstellen. Bei großen Modellen, die mit vielen Traces validiert werden sollen, um eine möglichst hohe Abdeckung zu erreichen, kann jedoch die Simulationszeit zunehmen, was die Effizienz des Ansatzes beeinträchtigen kann. Dadurch kam es vor, dass bei er Überprüfung einiger Eigenschaften die Zeit der Überprüfung bis zu einer Stunde dauern konnte. Bezüglich der Fehlertoleranz kann der Einsatz von SimB und Trace Replay die Wahrscheinlichkeit menschlicher Fehler reduzieren, wodurch die Qualität der Validierungsergebnisse verbessert wird.

Insgesamt bieten SimB und Trace Replay in der vorgestellten Methode eine vielversprechende Alternative zu herkömmlichen Validierungsansätzen, indem sie mit dem Fokus auf abstrakte Modelle eine effiziente und flexible Validierung ermöglichen. Es ist dennoch wichtig, die potenziellen Herausforderungen und Limitationen dieser Methode zu berücksichtigen. So ist es derzeit in SimB noch nicht möglich zu überprüfen, wie lange eine Eigenschaft innerhalb eines gesamten Trace verletzt oder eingehalten wurde. Zudem ist zu beachten, dass die Simulationszeit bei einer großen Anzahl von Traces sich stark verlängern kann, was die Effizienz des Ansatzes beeinträchtigen könnte. In Anbetracht dieser Aspekte ist es essenziell, den Kontext und die Anforderungen der jeweiligen Modellierungs- und Validierungsprojekte sorgfältig abzuwägen, um die passendste Methode auszuwählen.

5 Related Work

Im Bereich der Validierung von KI-Agenten und der Modellierung von Fahrzeugen in Verkehrssimulationen sowie im Bereich von formellen Methoden zur Validierung von Reinforcement Learning Agenten, hat es in den letzten Jahren eine Vielzahl von Forschungsarbeiten gegeben. Im Folgenden sind einige wichtige Arbeiten aufgeführt, die innerhalb dieser Gebiete publiziert wurden:

Sutton, R.S und Barto, A.G. [SB18] beschreiben in ihrem Buch „Reinforcement Learning: An Introduction“ die Grundlagen des Reinforcement Learnings. Dabei wird auf die verschiedenen Algorithmen und Methoden eingegangen, die für das Lernen von Agenten verwendet werden. Außerdem wird die Funktionsweise von Reinforcement Learning anhand von Beispielen erläutert. Es dient als wichtige Grundlage für die Entwicklung von KI-Agenten in verschiedenen Anwendungsbereichen.

Einen Überblick über die Prinzipien von „Model Checking“ geben Baier et al. in ihrem

Paper „Principles of Model Checking“ [BK08]. Dabei wird auf die verschiedenen Arten von Model Checking eingegangen und die Funktionsweise von Model Checking anhand von Beispielen erläutert.

Im Paper „Shield Synthesis for Reinforcement Learning“ stellen Bettina Könighofer et al. Schildsynthese vor. Bei dem Ansatz der Schildsynthese gibt es zwei verschiedene Arten von Schildern. Einerseits gibt es die Variante „pre-shielding“, diese schränkt die Auswahl der zu treffenden Entscheidungen des Agenten im Vorfeld ein. Im Gegensatz dazu existiert das „post-shielding“, hierbei wird die Entscheidung des Agenten beobachtet und nur dann abgeändert, falls diese zu einer Gefährdung führt. Dabei ist es wichtig, dass folgende Voraussetzungen erfüllt werden. Erstens garantierte Sicherheit, solange das Schild genutzt wird. Zweitens möglichst wenig und möglichst kleine Eingriffe in die Entscheidung des Agenten, um dessen Lernprozess nicht zu stark zu beeinflussen [KLJB20].

Während Könighofer et al. sich mit der Entscheidungsfindung von Agenten beschäftigen, wird im Paper „An STL-Based Formulation of Resilience in Cyber-Physical Systems“ von Hongkai Chen et al. die Resilenz betrachtet [CLSP22]. Resilenz wird beschrieben als die Fähigkeit, sich schnell von einer Verletzung zu erholen und zukünftige Verletzungen so lange wie möglich zu vermeiden. Das kann besonders interessant sein, wenn es darum geht das Verhalten der Agenten hinsichtlich der Einhaltung von Sicherheitsabständen zu verbessern.

6 Fazit

Zum Abschluss dieser Bachelorarbeit werden das Vorgehen und die wichtigsten Ergebnisse noch einmal zusammengefasst.

Im Rahmen dieser Arbeit, wurden mit Stable-Baselines, Agenten auf einem Highway-Environment trainiert. Daraufhin wurden die Agenten innerhalb SimB durch die Anwendung verschiedener Tests auf Eigenschaften validiert, die einen umfangreicheren Überblick über das Verhalten des Agenten lieferten. Die Ergebnisse der Validierung zeigten, dass die trainierten Agenten Schwächen bei der Einhaltung von Sicherheitsabständen aufwiesen, während dessen sie jedoch eine hohe Geschwindigkeit hielten. Dies resultierte in einer Unfallquote der Agenten von ungefähr 17 Prozent. Entscheidend bei der Validierung der Agenten durch Traces war, dass in dieser Arbeit ein abstraktes Modell geschrieben wurde und die Simulation des Modells durch die Künstliche Intelligenz selbst durchgeführt wurde.

Aus den Resultaten der Arbeit lässt sich schlussfolgern, dass die Validierung von RL-Agenten in Form der aufgezeigten Methode eine gute Alternative zu bereits bestehenden Validierungstechniken bildet. Grundsätzlich ermöglicht diese Methode eine schnelle und umfangreiche Überprüfung des Verhaltens der Agenten, die aufgrund der Nutzung eines abstrakten Modells schnell angepasst und skaliert werden kann.

Anhang

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmenverzeichnis

Quellcodeverzeichnis

Literatur

[Abr96] Abrial, Jean-Raymond: *The B-Book: Assigning Programs to Meanings*. New York, NY, USA : Cambridge University Press, 1996

[ALN+91] Abrial, J. R. ; Lee, M. K. O. ; Neilson, D. S. ; Scharbach, P. N. ; Sørensen, I. H.: The B-method. In: Prehn, Hans Sørenand T. Sørenand Toetenel (Hrsg.): *VDM '91 Formal Software Development Methods*. Berlin, Heidelberg : Springer Berlin Heidelberg, 1991. – ISBN 978–3–540–46456–3, S. 398–405

[AOS+16] Amodei, Dario ; Olah, Chris ; Steinhardt, Jacob ; Christiano, Paul ; Schulman, John ; Mané, Dan: Concrete problems in AI safety. In: *arXiv preprint arXiv:1606.06565* (2016)

[BDTD+16] Bojarski, Mariusz ; Del Testa, Davide ; Dworakowski, Daniel ; Firner, Bernhard ; Flepp, Beat ; Goyal, Prasoon ; Jackel, Lawrence D. ; Monfort, Mathew ; Muller, Urs ; Zhang, Jiakai u. a.: End to end learning for self-driving cars. In: *arXiv preprint arXiv:1604.07316* (2016)

[BGJ+21] Bendisposto, Jens ; Gelessus, David ; Jansing, Yumiko ; Leuschel, Michael ; Pütz, Antonia ; Vu, Fabian ; Werth, Michelle: ProB 2-UI: a java-based user interface for ProB. In: *Formal Methods for Industrial Critical Systems: 26th International Conference, FMICS 2021, Paris, France, August 24–26, 2021, Proceedings 26* Springer, 2021, S. 193–201

[BK08] Baier, Christel ; Katoen, Joost-Pieter: *Principles of model checking*. MIT press, 2008

[BLCW09] Bengio, Yoshua ; Louradour, Jérôme ; Collobert, Ronan ; Weston, Jason: Curriculum learning. In: *Proceedings of the 26th annual international conference on machine learning*, 2009, S. 41–48

[BN06] Bishop, Christopher M. ; Nasrabadi, Nasser M.: *Pattern recognition and machine learning*. Springer, 2006

[CLSP22] Chen, Hongkai ; Lin, Shan ; Smolka, Scott A. ; Paoletti, Nicola: An STL-based formulation of resilience in cyber-physical systems. In: *Formal Modeling and Analysis of Timed Systems: 20th International Conference, FORMATS 2022, Warsaw, Poland, September 13–15, 2022, Proceedings* Springer, 2022, S. 117–135

[EKN+17] Esteva, Andre ; Kuprel, Brett ; Novoa, Roberto A. ; Ko, Justin ; Swetter, Susan M. ; Blau, Helen M. ; Thrun, Sebastian: Dermatologist-level classification of skin cancer with deep neural networks. In: *nature* 542 (2017), Nr. 7639, S. 115–118

[FP18] Fulton, Nathan ; Platzer, André: Safe Reinforcement Learning via Formal Methods: Toward Safe Control Through Proof and Learning. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 32 (2018), Nr. 1. http://dx.doi.org/10.1609/aaai.v32i1.12107. – DOI 10.1609/aaai.v32i1.12107

[GBC16] Goodfellow, Ian ; Bengio, Yoshua ; Courville, Aaron: *Deep learning*. MIT press, 2016

[Gha04] Ghahramani, Zoubin: *Unsupervised Learning*. Springer Berlin Heidelberg, 2004. http://dx.doi.org/10.1007/978-3-540-28650-9_5. http://dx.doi.org/10.1007/978-3-540-28650-9_5. – ISBN 978–3–540–28650–9

[HDY+12] Hinton, Geoffrey ; Deng, Li ; Yu, Dong ; Dahl, George E. ; Mohamed, Abdel-rahman ; Jaitly, Navdeep ; Senior, Andrew ; Vanhoucke, Vincent ; Nguyen, Patrick ; Sainath, Tara N. ; Kingsbury, Brian: Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. In: *IEEE Signal Processing Magazine* 29 (2012), Nr. 6, S. 82–97. http://dx.doi.org/10.1109/MSP.2012.2205597. – DOI 10.1109/MSP.2012.2205597

[HE16] Ho, Jonathan ; Ermon, Stefano: Generative adversarial imitation learning. In: *Advances in neural information processing systems* 29 (2016)

[KLJB20] Könighofer, Bettina ; Lorber, Florian ; Jansen, Nils ; Bloem, Roderick: Shield Synthesis for Reinforcement Learning. In: Margaria, Tiziana (Hrsg.) ; Steffen, Bernhard (Hrsg.): *Leveraging Applications of Formal Methods, Verification and Validation: Verification Principles*. Cham : Springer International Publishing, 2020. – ISBN 978–3–030–61362–4, S. 290–306

[KSH17] Krizhevsky, Alex ; Sutskever, Ilya ; Hinton, Geoffrey E.: Imagenet classification with deep convolutional neural networks. In: *Communications of the ACM* 60 (2017), Nr. 6, S. 84–90

[LBH15] LeCun, Yann ; Bengio, Yoshua ; Hinton, Geoffrey: Deep learning. In: *nature* 521 (2015), Nr. 7553, S. 436–444

[Leu18] Leurent, Edouard: *An Environment for Autonomous Driving Decision-Making*. https://github.com/eleurent/highway-env, 2018

[MBM+16] Mnih, Volodymyr ; Badia, Adria P. ; Mirza, Mehdi ; Graves, Alex ; Lillicrap, Timothy ; Harley, Tim ; Silver, David ; Kavukcuoglu, Koray: Asynchronous methods for deep reinforcement learning. In: *International conference on machine learning* PMLR, 2016, S. 1928–1937

[MKS+13] Mnih, Volodymyr ; Kavukcuoglu, Koray ; Silver, David ; Graves, Alex ; Antonoglou, Ioannis ; Wierstra, Daan ; Riedmiller, Martin: Playing atari with deep reinforcement learning. In: *arXiv preprint arXiv:1312.5602* (2013)

[RHG+21] Raffin, Antonin ; Hill, Ashley ; Gleave, Adam ; Kanervisto, Anssi ; Ernestus, Maximilian ; Dormann, Noah: Stable-Baselines3: Reliable Reinforcement Learning Implementations. In: *Journal of Machine Learning Research* 22 (2021), Nr. 268, 1-8. http://jmlr.org/papers/v22/20-1364.html

[Rus10] Russell, Stuart J.: *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010

[SB18] Sutton, Richard S. ; Barto, Andrew G.: *Reinforcement learning: An introduction*. The MIT Press, 2018

[SGAS20] Shakarami, Ali ; Ghobaei-Arani, Mostafa ; Shahidinejad, Ali: A survey on the computation offloading approaches in mobile edge computing: A machine learning-based perspective. In: *Computer Networks* 182 (2020), 107496. http://dx.doi.org/https://doi.org/10.1016/j.comnet.2020.107496. – DOI https://doi.org/10.1016/j.comnet.2020.107496. – ISSN 1389–1286

[SMLE22] Stock, Sebastian ; Mashkoor, Atif ; Leuschel, Michael ; Egyed, Alexander: Trace Refinement in B and Event-B. In: *Formal Methods and Software Engineering: 23rd International Conference on Formal Engineering Methods, ICFEM 2022, Madrid, Spain, October 24–27, 2022, Proceedings* Springer, 2022, S. 316–333

[SWD+17] Schulman, John ; Wolski, Filip ; Dhariwal, Prafulla ; Radford, Alec ; Klimov, Oleg: Proximal policy optimization algorithms. In: *arXiv preprint arXiv:1707.06347* (2017)

[Tom97] Tom, Mitchell: Hill, McGraw. In: *Machine Learning* (1997)

[VLM21a] Vu, Fabian ; Leuschel, Michael ; Mashkoor, Atif: Validation of Formal Models by Timed Probabilistic Simulation. In: *Proceedings ABZ* Bd. 12709, 2021 (LNCS), S. 81–96

[VLM21b] Vu, Fabian ; Leuschel, Michael ; Mashkoor, Atif: Validation of formal models by timed probabilistic simulation. In: *Rigorous State-Based Methods: 8th International Conference, ABZ 2021, Ulm, Germany, June 9–11, 2021, Proceedings* Springer, 2021, S. 81–96

[WL20] Werth, Michelle ; Leuschel, Michael: VisB: a lightweight tool to visualize formal models with SVG graphics. In: *Rigorous State-Based Methods: 7th International Conference, ABZ 2020, Ulm, Germany, May 27–29, 2020, Proceedings 7* Springer, 2020, S. 260–265