Reinforcement Learning in Simulationsspielen: Repräsentation von großen Aktions- und Zustandsräumen am Beispiel von Ökolopoly

Bachelorarbeit zur Erlangung des Bachelor-Grades
Bachelor of Science im Studiengang Allgemeine Informatik
an der Fakultät für Informatik und Ingenieurwissenschaften
der Technischen Hochschule Köln

vorgelegt von:	Ralitsa Ivanova Raycheva
Matrikel-Nr.:	111 33 802
Adresse:	Reininghauserstr. 43
	51643 Gummersbach
	`ralitsa_ivanova.raycheva@smail.th-koeln.de`

eingereicht bei:	Prof. Dr. Wolfgang Konen
Zweitgutachterin:	Prof. Dr. Dietlind Zühlke

Gummersbach, 11.01.2022

Kurzfassung

Die vorliegende Abschlussarbeit erforscht mittels dem Framework RL Baselines3 Zoo die Anwendung von Reinforcement Learning (RL) auf das Spiel Ökolopoly. Dieses stellt ein komplexes, rückgekoppeltes System dar, das sich auf kybernetischen Prinzipien stützt. Aus diesem Grund verfügt Ökolopoly über komplexe Aktions- und Zustandsräume, die sich als eine große Herausforderung für das Erlernen des Spiels herausstellen. Daher werden passende Wrapper zur Modifizierung der Räume implementiert, die das Training der Agenten ermöglichen. Es werden unterschiedliche Reward-Varianten untersucht, die die Agenten beim Erstellen einer optimalen Policy leiten.

Mehrere Experimente werden mit den RL-Algorithmen PPO, TD3 und SAC durchgeführt, bei denen die Agenten jeweils für 800000 Timesteps trainiert werden. Dabei wird die Zusammenwirkung zwischen den Komponenten Action Wrapper, Observation Wrapper und Reward untersucht. Daraus lässt sich entnehmen, dass der domänenspezifische **Hilfsreward** sein Ziel erfüllt, die Überlebensfähigkeit der Agenten im Spiel zu stärken. Bei den verschiedenen Algorithmen wirken die Wrapper unterschiedlich. PPO arbeitet gut mit fast allen Kombinationen von Action und Observation Wrapper und Reward zusammen. TD3 hingegen erreicht positive Ergebnisse mit dem **Box Action Wrapper** und den Belohnunhgsvarianten **Hilfsreward** oder der Summe **Hilfsreward + Bilanzzahl**. SAC hingegen führt mit dem **Box Action Wrapper** zu guten Agenten, deren Leistung unabhängig von der eingesetzten Belohnungsart ist.

Bei dem Vergleich zwischen der erlernten Taktik zweier der besten Agenten und eines Menschen wird festgestellt, dass der mit **Box Action Wrapper** trainierte Agent eine vorausschauendere Strategie als der Mensch entwickelt hat, was ein tieferes Verständnis für die inneren Abläufe im Spiel zeigt. Beim Test der Robustheit weist der Agent, der mit dem **Simple Action** und dem **Simple Observation Wrapper** trainiert wurde, ein robusteres Verhalten als der andere auf.

Schlüsselwörter: Ökolopoly, Large Action Spaces, Large Observation Spaces, Reinforcement Learning, OpenAI Gym, RL-Baselines3-Zoo

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Listings

1 Einleitung

Das Spiel Ökolopoly wurde vom bekannten Biochemiker und Kybernetiker Frederic Vester entworfen und 1984 bei Ravensburger veröffentlicht. Sein Ziel war durch das Konzipieren eines greifbaren rückgekoppelten Systems das vernetzte Denken auf spielerische Art und Weise beizubringen. Der Spieler agiert als Herrscher eines fiktiven Landes und wirkt auf die realisierten Regelkreise ein [Vester, 1988].

Ökolopoly stellt als Studienobjekt ein besonderes Interesse für diese Arbeit dar, da es komplexe Zusammenhänge enthält. Das Erlernen dieser stellt schon eine Herausforderung für den Menschen dar, von dem letztendlich erwartet ist, eine hochwertige Spielstrategie entwickeln zu können. Diese Aufgabe erweist sich für einen Agenten im Bereich des Reinforcement Learnings sogar als schwieriger, da er im Unterschied zum Menschen die Regeln des Spiels zu Beginn überhaupt nicht kennt. Durch ständige Interaktionen mit der Umgebung und die erhaltenen Rewardsignale muss er ein Verständnis sowohl für den Spielregelsatz, als auch für die inneren Abläufen aufbauen. Es wird von ihm noch erwartet eine erfolgreiche Strategie zu erstellen und mehrere Runden im Spiel zu überleben.

Ein weiteres Problem bei so einer Aufgabe stellt die Größe des Aktions- und Zustandsraums von Ökolopoly dar, die das richtige Erforschen der Umgebung erheblich erschweren. In dieser Situation kann kaum geschätzt werden, wie viel Zeit für das Training eines Agenten erforderlich ist und ob er am Ende überhaupt etwas lernt. Diese Problematik erfordert dann spezielle Lösungen, die den Trainingsprozess eventuell erleichtern können. Das vernetzte Denken einem RL-Agenten beizubringen weist damit schon große Komplexität auf und es ist nicht direkt ersichtlich, ob Reinforcement Learning für Ökolopoly gelingen wird.

1.1 Motivation

Riesige Aktions- und Zustandsräume stellen eine Herausforderung in verschiedenen Anwendungsfällen dar. Recommender Systems wie solche in Youtube oder Amazon müssen dazu fähig sein, die Vielzahl an Informationen so zu verarbeiten, damit die Empfehlungen den Bedürfnissen des Nutzers entsprechen [Dulac-Arnold et al., 2015, S. 1]. In Simulationsspielen wird hauptsächlich Reinforcement Learning für das Verhalten von vorprogrammierten Entities eingesetzt, die mit dem Spieler interagieren [Merrick und Maher, 2009]. Bei hochdimensionalen Echtzeitspielen wie Dota2 [Berner et al., 2019] muss der Agent mit der Vielfalt an Aktionen und Zuständen zu jedem Zeitpunkt gut zurechtkommen. Dulac-Arnold äußert sich, dass riesige Räume eine Herausforderung im Bereich Reinforcement Learning darstellen:

“Recommender systems, industrial plants and language models are only some of the many real-world tasks involving large numbers of discrete actions for which current methods are difficult or even often impossible to apply.” [Dulac-Arnold et al., 2015, S. 1]

In Unterschied zu einfacheren RL Problemen wie zum Beispiel Mountain Car[1], wo es 2 Aktionen sind, kann dagegen Ökolopoly über Millionen von solchen verfügen (s. Kapitel

[1] https://gym.openai.com/envs/MountainCar-v0/

2.2). In diesem Sinne eignet es sich, die Räume zunächst umzugestalten, um deren Erlernbarkeit für den Agenten zu vereinfachen.

Die vorliegende Arbeit erforscht mögliche Repräsentationen des Aktions- und Zustandsraums von Ökolopoly. Dafür werden passende Wrapper implementiert und mittels des Framewoks RL-Baselines3-Zoo entsprechende Agenten mit verschiedenen Konfigurationen trainiert. Um deren erstellte Strategie während des Trainings zu bewerten, wird ein Analyseprogramm realisiert, das die Züge des Agenten Schritt für Schritt zeigt.

Folgende Forschungsfragen stehen dabei im Vordergrund:

F 1.1 Kann Reinforcement Learning auf Ökolopoly angewandt werden, gegeben den komplexen Aktions- und Zustandsraum?

F 1.2 Welchen Einfluss haben die verschiedenen Repräsentationen der Räume auf den Erfolg (Lernprozess und Performance) des Agenten?

F 1.3 Inwieweit beeinflusst der Reward den Erfolg des Agenten?

F 2.1 Wie robust ist ein trainierter Agent, wenn die Startposition geringfügig geändert wird?

F 2.2 Wie viele Trainingsrunden braucht der Agent, um zu lernen?

F 2.3 Kann der Agent bessere Ergebnisse als ein Mensch erzielen?

F 2.4 Kann der Mensch etwas von den Agenten lernen?

1.2 Verwandte Arbeiten

Die Recherchen zum Auffinden ähnlicher Arbeiten wie die vorliegende wurden in Google Scholar, IEEE Xplore, Karlsruher Virtueller Katalog, und TIB mit folgenden Schlagworten durchgeführt: *ökolopoly, ökolopoly künstliche intelligenz, ökolopoly reinforcement learning, ecopolicy machine learning, ecopolicy reinforcement learning*

Nach bestem Wissen wurde das KI-basierte Lernen in Bezug auf Ökolopoly in der Literatur bisher nur einmal behandelt. Der Artikel [Holzbaur, 1991] betrachtet das Spiel und neuronale Netze, ist doch schon älter und im Volltext nicht vorhanden.

Dieser Abschnitt konzentriert sich daher auf zwei Vorgehensweisen zur Behandlung riesiger diskreter Räume.

1.2.1 Wolpertinger Architektur

Eine Möglichkeit für effizienten Umgang mit sehr großen Aktionsräumen bietet der von Google DeepMind entwickelte Ansatz genannt Wolpertinger Architektur [Dulac-Arnold et al., 2015]. Der Name bezeichnet eine neu vorgestellte Struktur für die Policy. Dabei wird zunächst für den konkreten Zustand eine sogenannte Proto-Aktion erzeugt. Diese stammt aus einem kontinuierlichen Aktionsraum und muss im Endeffekt einem diskreten zugeordnet werden. Diese Anpassung verläuft grundsätzlich durch die Ermittlung der k nächsten Nachbarn

dieser kontinuierlichen Aktion. Die Auswahl wird verfeinert, indem jener mit dem höchsten Q-Wert in die Umgebung als valide Aktion eingesetzt wird. Auf diese Weise wird eine Art von Policy entwickelt, die eine Teilmenge des gesamten Aktionsraums erforscht und darauf basierend genügend generalisieren kann. Der Paper beweist, dass dies in vielen RL-Aufgaben zu effizientem Lernen und erheblichen Geschwindigkeitssteigerungen führt.

1.2.2 Action Elimination

Eine weitere Idee, den Agenten zu helfen, mit riesigen Räumen zurechtzukommen, ist der Ausschluss irrelevanter Aktionen, wie von [Zahavy et al., 2018] vorgestellt. Während des Lernprozesses bekommt der Agent eine Art von Hilfsreward, der auch als Elimination Signal bezeichnet wird. Er ist domänenspezifisch und gibt direkten Feedback über nicht optimale Aktionen für einen gegebenen Zustand. Um das zu realisieren werden zwei tiefe Netzwerke zur Function Approximation implementiert. Das eine stellt einen Satz an zulässigen Aktionen bereit. Das andere Netzwerk entscheidet dann, welche Aktion ausgeführt werden muss. Im Grunde genommen werden sowohl die Q-Funktion, als auch die Eliminierung von Aktionen gelernt. Auf diese Weise werden überwiegend die Q-Werte valider Action-State Paare in Betracht gezogen, was zu schneller Konvergenz führen sollte. Dieser Ansatz ist jedoch für den Bereich natürlicher Sprache gedacht und wird letztendlich auf einem text-basierten Spiel getestet. Die durchgeführten Experimente zeigen konsistente Ergebnisse – der Agent lernt schneller und erzielt höhere Belohnungen als der vergleichbare state-of-the-art Algorithmus, der nur die Q-Funktion approximiert.

1.3 Algorithmen

1.3.1 Reinforcement Learning

Im vorliegenden Abschnitt werden einige der Hauptelemente des Reinforcement Learnings sowie die grundsätzliche Funktionsweise der für diese Arbeit eingesetzten Algorithmen eingeführt. In dieser Hinsicht wird zusammenfassend darauf eingegangen, wie die Agenten mit den verschiedenen RL-Ansätzen lernen und die Exploration durchführen.

Im Fokus von Reinforcement Learning sind der Agent und seine Umgebung (s. Abbildung 1), wo er durch Interaktion mit dieser für begrenzte Anzahl von Zeitschritten lernt. Sein Verhalten bzw. seine Wahl der Aktion $\boldsymbol{a_t}$ in einem gegebenen Zustand $\boldsymbol{s_t}$ wird durch die Policy $\boldsymbol{\pi}$ entschieden, die so angepasst werden muss, dass am Ende ein möglichst maximaler Reward erzielt wird.

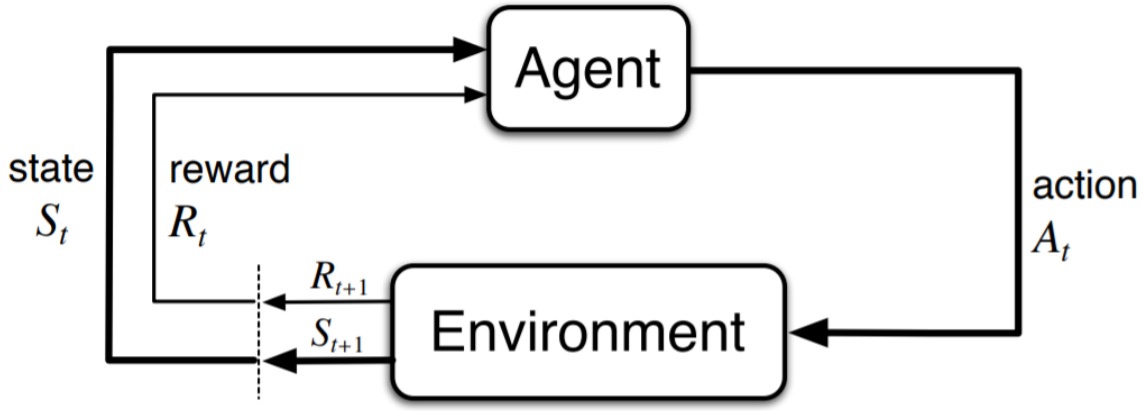


Abbildung 1 Interaktion zwischen den Agenten und der Umgebung [Sutton und Barto, 2018, S. 54]

Die vom Agenten ausgeführten Aktionen wirken auf das Environment und lösen dort den Zustandsübergang und den Reward **r** aus. Während **r** unmittelbar Feedback über die getroffene Aktion gibt, stellt die sogenannte Value-Funktion allgemein dar, welche zu guten Ergebnissen führen könnte. Der Wert (Value) eines Zustandes ist der Erwartungswert der Gesamtsumme aller Rewards, die der Agent ausgehend von diesem Zustand in der Zukunft bekommt. Sei der Return die kumulierte Summe aller zukünftigen Rewards und $\boldsymbol{\pi}$ die Policy, sind folgende Value-Funktionen zu unterscheiden [Sutton und Barto, 2018, S. 7]:

$\boldsymbol{v_\pi(s)}$ (state-value function): berechnet für den Zustand **s** den erwarteten Return, wenn **s** der Startzustand ist und danach $\boldsymbol{\pi}$ befolgt wird.

$\boldsymbol{q_\pi(s, a)}$ (action-value function): ermittelt den erwarteten Return für ein Aktion-Zustand Paar **(s, a)**, wenn **s** der Startzustand ist, Aktion **a** ausgeführt wird und danach $\boldsymbol{\pi}$ befolgt wird [Sutton und Barto, 2018, S. 70, 81].

Das Berechnen dieser Funktionen kann extrem kompliziert sein, wenn es sich um riesige Räume handelt. Dafür muss ein großer Teil ihrer Outputs mittels Generalisierungsmethoden angenähert werden:

"The kind of generalization we require is often called function approximation because it takes examples from a desired function (e.g., a value function) and attempts to generalize from them to construct an approximation of the entire function." [Sutton und Barto, 2018, S. 225]

Die approximierte Value-Funktion kann in funktionaler Form dargestellt werden, die in der Regel als Parameter den Vektor $\boldsymbol{w} \in \boldsymbol{R_n}$ erhält. Seine Dimensionen werden weiter in Abhängigkeit der gesammelten Erfahrung im Environment angepasst [Sutton und Barto, 2018, S. 226], damit im Endeffekt die Policy aufgrund der angenäherten Zustandswerte in Abhängigkeit von **w** optimiert werden kann. Genau diese Aufgabe liegt im Schwerpunkt von Generalisierungsansätzen wie Gradient Descent Methoden. Diese Herangehensweise für

das Erstellen der Policy nennt sich wertbasiert. Bei Actor-Critic Methoden wird hingegen die Policy als Akteur unabhängig von der Value-Funktion, dem Kritiker, dargestellt. Letzterer evaluiert die vom Akteur gewählte Aktion [Sutton und Barto, 2018, S. 257]. Diese Struktur steht im Kern von On- und Off-Policy Algorithmen. Der Unterschied zwischen den beiden lässt sich folgendermaßen erklären:

"Policy improvement is done by changing the estimation policy to the greedy policy (in off-policy methods) or to a soft approximation of the greedy policy such as the greedy policy (in on-policy methods). Actions are selected according to this same policy in on-policy methods, or by an arbitrary policy in off-policy methods." [Sutton und Barto, 2018, S. 242]

1.3.2 PPO

Bei dem On-Policy Algorithmus PPO handelt es sich um einen Policy Optimisation Ansatz, was bedeutet, dass die Policy durch eine separate Nährungsfunktion mit eigenen Parametern approximiert wird [Sutton et al., 2000]. Im Allgemeinen sammeln zunächst mehrere Akteure simultan Datenproben aus der Umgebung für Timesteps T. Basierend darauf wird die Loss Funktion L für den Gradienten der Policy für K Epochen mittels Stochastic Gradient Ascend optimiert (s. Abbildung 2). Diese Clipping-Variante hat zum

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

- θ is the policy parameter
- $\hat{E}_t$ denotes the empirical expectation over timesteps
- r_t is the ratio of the probability under the new and old policies, respectively
- $\hat{A}_t$is the estimated advantage at time t
- ε is a hyperparameter, usually 0.1 or 0.2

Abbildung 2 Formel der PPO L-Funktion, Ausschnitt aus OpenAI Baselines

Ziel eine pessimistische Schätzung über die Performance der Policy zu formen, um potentiell überschätzte Werte der Aktionen und somit große Policy Updates zu vermeiden. Auf diese Weise entwickelt sich die Policy langsamer und stabiler [Schulman et al., 2017].

Die Exploration verläuft bei PPO durch das durchgehende Sampling von Aktionen aus der alten Policy, die in den meisten Fällen stochastisch ist [Hollenstein et al., 2020, S. 5]. Bei Aufgaben mit kontinuierlichen Aktionsräumen wird Gaussian Noise zu der gewählten Aktion addiert [Zhang et al., 2020, S. 10].

1.3.3 TD3

Der Off-Policy Algorithmus TD3 baut auf der Actor-Critic Methode von DDPG [Lillicrap et al., 2015] auf. Er verfügt über einen Akteur und zwei Kritiker-Netzwerke mit ihren jeweils korrespondierenden Target-Netzwerken. Die Target-Netzwerken dienen hauptsächlich dazu, die Fehler pro Policy-Update zu reduzieren, um weiterhin Überschätzungen zu vermeiden. Als

großes Problem des Q-Learning erweist sich das Maximieren von stark verrauschten Werten [Fujimoto et al., 2018, S. 1]. Um dies zu vermeiden, bietet TD3 folgende Ergänzungen:

"First, we show that target networks, a common approach in deep Q-learning methods, are critical for variance reduction by reducing the accumulation of errors. Second, to ad-dress the coupling of value and policy, we propose delaying policy updates until the val-ue estimate has converged. Finally, we introduce a novel regularization strategy, where a SARSA-style update bootstraps similar action estimates to further reduce variance." [Fujimoto et al., 2018, S. 1]

Der Algorithmus unterscheidet zwischen Zielpolicy und eine Policy zur Exploration. Letztere wird zu Beginn für eine gegebene Anzahl von Zeitschritten verwendet. Wenn diese ablaufen, erfolgt die Erforschung neuer Aktionen, indem Gaussian Noise zu ihnen addiert wird. In jedem Zeitschritt werden danach die beiden Kritiker aktualisiert und der niedrigere zwischen den beiden approximierten Werten wird der Aktion zugewiesen [Fujimoto et al., 2018, S. 6]. Dieser Ansatz ist als Clipped Double Q-Learning bekannt. Auf diese Weise werden Zustände mit niedriger Varianz hervorgehoben, was sichere Policy-Updates garantieren sollte.

1.3.4 SAC

Ein weiterer Off-Policy Algorithmus ist SAC, bei dem der Akteur neben dem Reward auch die Entropy der Policy maximieren muss. Das bedeutet, der Agent muss Erfolg erzielen, während er willkürliche Aktionen auswählt. Dies sollte folglich ein verbessertes Explorationsverhalten garantieren. Ähnlich wie bei TD3, verfügt SAC über zwei Kritiker, von denen der niedrigere Wert für den Gradienten der Policy verwendet wird, um Überschätzung vorzubeugen. Die beiden werden im Unterschied zu TD3 unabhängig voneinander berechnet. Zu Beginn des Algorithmus wird eine Aktion unter der aktuellen Policy selektiert. Danach werden die daraus resultierenden Informationen wie Zustand und erhaltener Reward beobachtet und im Replay-Buffer gespeichert. Am Schluss werden die beiden Kritiker und die Policy aktualisiert [Haarnoja et al., 2018].

2 Ökolopoly

Dieses Kapitel gibt einen Überblick über die Umgebung der zukünftigen Agenten – das Spiel Ökolopoly [Vester und Korte, 1993]. Zunächst wird seine Essenz mit Betonung auf die grundsätzlichen Regeln und den Spielablauf vorgestellt. Danach wird die Implementation in OpenAI Gym kurz behandelt. Die aufgeführten Erklärungen fassen weitgehend Informationen aus dem vor dieser Arbeit stattgefundenen Praxisprojekt zusammen. Auch die Umsetzung und der Ablauf des Spiels werden in der Projektarbeit detaillierter darlegt [Raycheva, 2021].

Das Wesen von Ökolopoly stützt sich darauf, eine schematische Modellierung der Prozessabläufe in der realen Welt darzustellen. Acht Lebensbereiche machen den Stand eines fiktiven Landes im Spiel aus, wobei sie ein regelkreisbasiertes System bilden. Das Gesamtwirkungsgefüge ist dem Spieler verborgen, was ihn dazu animiert, ein Gefühl für die in Ökolopoly realisierten Rückkopplungsmechanismen zu entwickeln. Aufgrund dessen kann er ihre Auswirkung auf die verschiedenen Lebensbereiche rechtzeitig erkennen und die passenden Maßnahmen dagegen ergreifen [Huber, 2011, S. 2, 3]. Das Hauptziel liegt schließlich dabei, den Menschen das vernetzte Denken auf spielerische Art und Weise beizubringen. Daher lässt sich Ökolopoly auch als ein Simulationsspiel bezeichnen. Sein Autor, der bekannte Biochemiker und Forscher Frederic Vester, konzipiert Ökolopoly auf Basis kybernetischer Prinzipien, welche die Komplexität im internen Wirkungsgefüge voraussetzen. Ihr Wissenschaftsbereich lässt sich folgendermaßen erfassen:

„Der Begriff ›Kybernetik‹ bezeichnet einen interdisziplinären Forschungsansatz, der die Existenz universal gültiger Prinzipien wie ›Rückkopplung‹ (s. Kap. III.42), ›Information‹ (s. Kap. III.46) oder ›Selbstorganisation‹ für unterschiedliche Gegenstandsbereiche in Technik, Natur und Kultur postuliert und durch die Erforschung dieser Prinzipien zu einer Wiederannäherung spezialisierter Fachdisziplinen beitragen möchte.“ [Müggenburg, 2019]

Die Prinzipien der Rückkopplung und Selbstorganisation sind deutlich in den in Ökolopoly abgebildeten Prozessen zu beobachten, da jeder einzelne Lebensbereich sich selbst reguliert und gleichzeitig Einfluss auf andere hat.

2.1 Regeln

2.1.1 Spielanfang

Wie bereits erwähnt, sind die acht Lebensbereiche in Ökolopoly mit ihren zugehörigen Startpositionen vertreten. Abbildung 3 veranschaulicht diese inklusive ihrer minimalen und maximalen Werte:

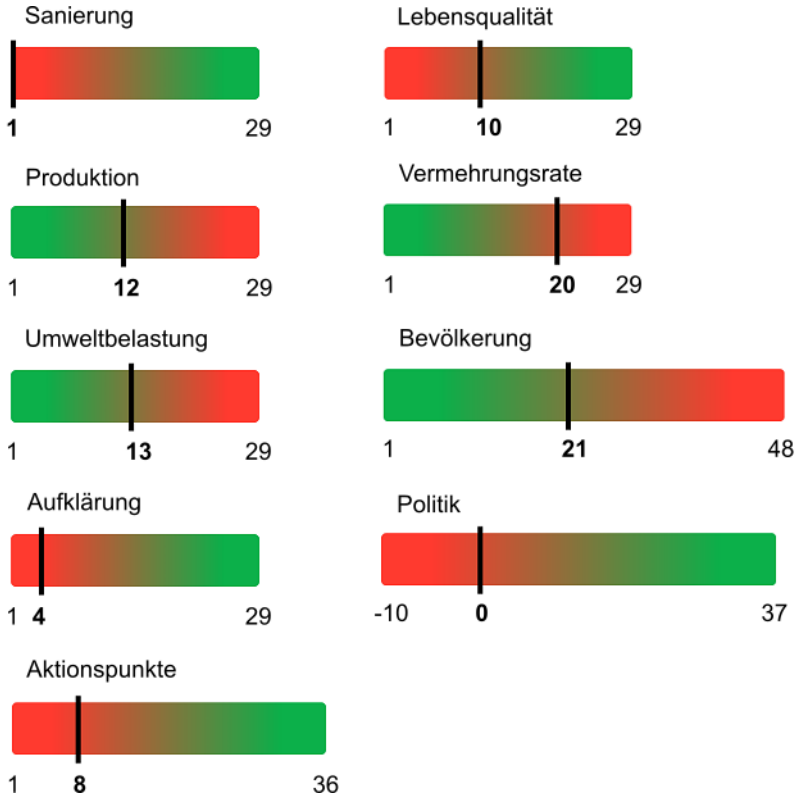


Abbildung 3 Startposition im Spiel

Die rote Farbe in den einzelnen Balken gibt an, dass sich der jeweilige Lebensbereich einer ungünstigen Lage befindet. Ähnlich bezeichnet die grüne Farbe eine günstige Lage. Dabei ist allerdings zu beachten, dass der Erfolg im Spiel nicht garantiert ist, wenn sich die Werte der Lebensbereiche zu stark dem Grün annähern. Die Grundidee besteht darin, einen Ausgleich zu finden, so dass alle Bereiche symbiotisch miteinander funktionieren.

Ausgehend von den vorliegenden Startpositionen kann festgestellt werden, dass sich die Sanierung, Aufklärung und Politik in einer schlechten Lage befinden. Dem Spieler ist demzufolge überlassen, eine wirksame Taktik zu entwerfen, sodass diese Bereiche in eine höhere Position überführt werden und sich die Lage des fiktiven Landes verbessert. Dafür muss er eine Punktvergabe durchführen, womit er die Rückkopplungsmechanismen im Spiel erkundet und die Entwicklung der Bereiche in die von ihm gewünschte Richtung steuert. Zu Beginn sind insgesamt 8 Aktionspunkte verfügbar, die wahlweise unter Sanierung, Produktion, Aufklärung, Lebensqualität und Vermehrungsrate zu verteilen sind.

2.1.2 Spielablauf

Nach der Verteilung der Aktionspunkte aktualisieren sich die Werte der einzelnen Lebensbereiche gemäß dem eingebauten Wirkungsgefüge im Spiel. Auf diese Weise ergibt sich der neue Zustand im Spiel. Die Aktionspunkte für die nächste Runde werden dabei auch berechnet. In der Regel hängt ihre Anzahl vom Zustand der Bereiche Politik, Bevölkerung, Lebensqualität und Produktion ab. Je höher die Werte in diesen Bereichen sind, desto mehr Aktionspunkte stehen für die nächste Verteilung zur Verfügung. Dabei ist zu beachten, dass das übermäßige Investieren von Punkten in diese Bereiche zu ihrem plötzlichen Übersteuern oder zur Beeinträchtigung anderer Lebensbereiche führen kann. Ist das nicht der Fall, kann die Anzahl der Aktionspunkte aufgrund des Erreichens der Extremwerte in den oben genannten Lebensbereichen die zulässigen Grenzen von 1 bis 36 auch verlassen. Also kann sich eine Strategie, die sich hauptsächlich darauf fokussiert, so viel Punkte zu erlangen, wie möglich, im Lauf des Spiels als schädlich erweisen und das vorzeitige Ende des Spiels herbeiführen.

Die nächste Runde fängt wieder mit der Vergabe von Punkten an. Diese ist allerdings optional und der Spieler trifft die Entscheidung, ob er diese investiert oder für später spart. Außerdem sind zu Beginn jeder Runde nur bei der Produktion Punktabzüge gestattet, was auch Aktionspunkte kostet. Bei den Lebensbereichen Aufklärung und Produktion muss ferner beachtet werden, wenn die Aufklärung Werte von mindestens 21 erreicht, kann die Vermehrungsrate einen Übertrag von minimal -5 bis maximal 5 bekommen (Spezialfall). Der konkrete Bereich der zu verteilenden Sonderpunkte hängt vom Wert der Aufklärung ab. Liegt die Aufklärung beispielsweise zwischen den Werten 21 und 24, darf der Spieler einen Übertrag aus $\pm 3 = -3, -2, -1, 0, +1, +2, +3$ auf die Vermehrungsrate anwenden.

2.1.3 Spielende und Bewertung eigener Leistung

Das Spielende wird erreicht, wenn ein Bereich inklusive der Anzahl der Aktionspunkte außerhalb des zulässigen Ranges gerät oder 30 Runden erreicht sind.

Letztendlich kann der Erfolg der angewandten Strategie mittels folgender Formel berechnet werden:

$$B = \left\{ \begin{array}{ll} \frac{10(P+3D_L)}{r+3}, & wenn\ 10 \leq r \leq 30 \\ \\ 0, & wenn\ r < 10 \end{array} \right\} \tag{1}$$

P: Politik
D_L: Fenster D im Bereich Lebensqualität
r: Rundenanzahl

Tabelle 1 fasst die Bedeutung der erhaltenen Bilanzzahl zusammen. Ihr Inhalt schreibt die entsprechenden Interpretationen aus der Spielanleitung kurz um, damit die Kernaussagen in einer kompakten und deutlichen Form ausgedrückt vorliegen. Der Skala ist demnach zu entnehmen, dass sich eine gute bis ausgezeichnete Leistung durch eine Bilanzzahl über 15 ergibt. Eine Bilanzzahl unter 5 lässt sich dementsprechend als schwach bis total erfolglos bezeichnen.

Bilanz	Interpretation
Über 20	Die Spielleistung legt einen unwiderlegbaren Beweis für das erfolgreiche Meistern der Herausforderungen im Spiel vor. Die angewandte Strategie zeichnet sich durch Weitblick und Scharfsinn aus.
15-20	Die Anwendung einer weit blickenden und vernünftigen Taktik überführt das fiktive Land in eine verbesserte Lage.
10-15	Die Spielleistung weist auf, dass anhand des relativ guten Umgangs mit den Mechanismen im Spiel ein Ausgleich zwischen den Lebensbereichen im fiktiven Land erreicht ist.
5-10	Die Spielstrategie kann verbessert werden, indem sie in eine weitschauendere Richtigung entwickelt wird.
0-5	Die angewandte Spielstrategie scheitert durch die überwiegend kurzsichtigen und einfältigen Entscheidungen.
Unter 0	Eine total unwirksame Spielleistung ist aufgrund fehlenden Verständnisses für die Regeln/Rückwirkungen des Spiels erbracht.

Tabelle 1 Skala der Bilanzzahl aus der Anleitung des Spiels

2.2 Größe der Aktions- und Zustandsräume

2.2.1 Aktionsraum

Die Aktionspunkte entsprechen den Werten $A \in \{1, \ldots, 36\}$. Bei ihrer Vergabe gibt es A^5 mögliche 5-Tupel $(a_1, .., a_5)$, die die Anzahl der den jeweiligen Lebensbereichen zugeteilten Aktionspunkte repräsentieren. Aus diesem ist rund die Hälfte valide, da die Aufteilungsbedingung $\sum_{i=1}^{5} a_i \leq A$ beachtet werden muss. Wenn die verfügbaren Punkte das Maximum 36 betragen, berechnen sich die möglichen Aktionen wie folgt:

$$\frac{36^5}{2} = \text{ungefähr } 3.02 \cdot 10^7 \text{ mögliche Aktionen bei einer Punktzahl von 36}$$

2.2.2 Zustandsraum

Anhand Abbildung 3 errechnet sich die Anzahl der Zustände in Ökolopoly folgendermaßen:

$$29^6 \cdot 48^2 \cdot 36 = 4.93 \cdot 10^{13} \text{ Zustände}$$

29 entspricht den maximalen Werten von 6 der Bereiche. Ähnlich 48 ist die obere Grenze von Bevölkerung und Politik. Letztlich steht 36 für das Maximum der Aktionspunkte.

2.3 Implementation

Bisher existiert keine Implementation von Ökolopoly mit verfügbarem Quellcode, deshalb muss das Spiel von Grund auf realisiert werden. Dafür eignet sich das OpenAI Gym Fra-

mework. Es ermöglicht die Anwendung von RL-Algorithmen und somit die zukünftige Erforschung des Spiels durch Agenten. Als Musterbeispiele dienen die schon implementierten Umgebungen in Gym, die bekannte RL-basierte Aufgaben darstellen wie das Balancieren einer Stange auf einem Wagen im Cartpole-Environment[2].

Bei OpenAI Gym[3] handelt es sich um ein von OpenAI unterstütztes Python Framework. Es legt ein einheitliches Schema – die gemeinsame Schnittstelle *Env* – zur Implementation beliebiger Umgebungen fest. Dieses Interface enthält die wichtigsten Attribute und Funktionen, durch die die Kerneigenschaften und -abläufe in einem Environment realisiert werden können. Weiterhin stellt das Framework eine Reihe von bereits realisierten Environments zur Verfügung. Diese dienen nicht nur als Muster bei dem Erstellen einer eigenen Umgebung, sondern auch für die Einrichtung von Experimenten mit unterschiedlichen RL-Algorithmen, sodass vergleichende Benchmarks erzeugt werden können. Dies trägt wesentlich dazu bei, eindeutige Aussagen über die Performance verschiedener RL-Methoden leichter treffen zu können.

Die Struktur jeder Gym-Umgebung enthält mindestens das Folgende [Lapan, 2020, S. 33]:

- Die Felder *action_space* und *observation_space* vom gemeinsamen Typ *Space*, die das Format der gültigen Aktionen und Zustände definieren. Diese können entweder diskret (*Discrete*/*MultiDiscrete*) oder kontinuierlich (*Box*) sein. Die Möglichkeit der Kombination beider Typen (in Form eines Tupels) wird nicht weiter berücksichtigt, da das für diese Ausarbeitung eingesetzte Framework RL Baselines3 Zoo dies nicht unterstützt. [Lapan, 2020, S. 32]
- Die *step()*-Funktion, die die Dynamik des Environments definiert. Aufgrund der übergebenen Aktion gibt sie den neuen Umgebungszustand *obs*, den erhaltenen Reward *reward*, die boolesche Variable *done*, ob ein Episodenende auftritt und den String *info* für sonstige Informationen als Rückgabewerte an.
- Die *reset()*-Funktion, die den initialen Zustand der Umgebung wiederherstellt.

Aufgrunddessen stellt OpenAI Gym das praktische Grundgerüst zur Entwicklung des Environments für Ökolopoly bereit.

2.3.1 Environment

Der komplette Quellcode ist im für diese Arbeit angelegten Repository enthalten (Link dazu im Anhang). In der Ökolopoly-Umgebung sind die Aktions- und Zustandsräume als *MultiDiscrete* definiert. Dieser Typ repräsentiert einen n-dimensionalen Vektor im Form eines *np.array*, wobei jedes seiner Elemente einen diskreten Raum darstellt. Seine Grenzwerte erfassen die Null bis $n - 1$, wobei n die eingegebene Obergrenze ist.

[2] `https://gym.openai.com/envs/CartPole-v0/`
[3] `https://gym.openai.com/`

```
85          self.action_space = spaces.MultiDiscrete([
86              29, # 0 Sanierung
87              57, # 1 Produktion
88              29, # 2 Aufklärung
89              29, # 3 Lebensqualität
90              29, # 4 Vermehrungsrate
91              11, # 5 Spezialfall Aufklärung > Vermehrung
92          ])
```

Listing 1 Quellcode Snippet: Ökolopoly Action Space in oeko_env.py

Die Action Space in Ökolopoly verfügt über 6 Dimensionen. Dabei ist zu beachten, dass das Maximum für die Produktion höher ist, damit die Punktabzüge abgebildet werden können. Weiterhin berücksichtigt das letzte Element des Arrays den besonderen Fall, in dem die Vermehrungsrate geschenkte Punkte erhält oder verliert (s. Ende von Absatz 2.1.2).

```
94          self.observation_space = spaces.MultiDiscrete([
95              29, # 0 Sanierung
96              29, # 1 Produktion
97              29, # 2 Aufklärung
98              29, # 3 Lebensqualität
99              29, # 4 Vermehrungsrate
100             29, # 5 Umweltbelastung
101             48, # 6 Bevölkerung
102             48, # 7 Politik
103             31, # 8 Runde
104             37, # 9 Aktionspunkte für nächste Runde
105         ])
```

Listing 2 Quellcode Snippet: Ökolopoly Observation Space in oeko_env.py

Auf ähnliche Weise ist der Vektor des Zustandsraumes 10-dimensional. Da der Lebensbereich Politik negative Zahlen in seinem Range einschließt, wird dieselbe Taktik wie bei der Produktion hier angewandt und eine größere Obergrenze eingegeben. Da *MultiDiscrete* keine negativen Werte unterstützt, werden die erzeugten Aktionen und Zustände ständig intern umgerechnet, damit sie formatkonform bleiben. Die *V*- und *obs*-Variable stellen jeweils die menschenfreundlichen und die für das Environment validen Repräsentationen des Umgebungszustands dar. Das Wechseln zwischen den beiden Darstellungsformen erfolgt durch die Hilfsvariablen *Vmin* und *Vmax* für die Observations und *Amin* und *Amax* für die Aktionen, die untere und obere Grenzwerte der Lebensbereiche enthalten. Zum Beispiel:

Aktion aus Aktionsraum		Amin		Aktion im Spiel
(0, 26, 5, 6, 0)	+	(0, -28, 0, 0, 0)	=	(0, -2, 5, 6, 0)

Abschließend erfolgt der Übergang von Zustand s_t zu s_{t+1} im Environment deterministisch. Der unten dargestellte Pseudocode erläutert kompakt die Dynamik des Ökolopoly-Environments.

```
Constructor __init__

1. Set init_v with initial game state which is used when the game is being
    reset
2. Set upper and lower bound limit for each value of the observation
   vector in arrays Vmin and Vmax
3. Set upper and lower bound limit for each value of the action vector in
   arrays Amin and Amax
4. Define type and upper bound limit of action and observation space
```

Listing 3 Pseudocode: Anlegen grundsätzlicher Variablen

```
Function Step
Input action: MultiDiscrete
Output: obs, reward, done, info

1. Transform passed action from action_space to valid game action values
2. Check if action is valid
       - check if more points are used than available
       - check if resulting regions are out of range
3. Add action points to values of regions
4. Make a game turn and update V (game state) according to game logic
5. If V is out of range, clip its values based on Vmin and Vmax
6. Build the sum of points for next round
7. Calculate balance if done=True
8. Calculate observation according to V
9. Update reward
```

Listing 4 Pseudocode: Implementation der Spiellogik

```
Function Reset
Output obs: MultiDiscrete

1. Set initial values in game using init_v
2. Calculate observation according to V
```

Listing 5 Pseudocode: Zurücksetzen des Spiels

Die Variable *init_v* stellt die Startbedingungen des Environments dar. Die Funktion *step()* implementiert die in Ökolopoly verborgenen Rückwirkungsmechanismen. Diese sind manuell aus dem wirklichen Spiel extrahiert und in einer zusätzlichen Datei *get_boxes.py* definiert.

2.3.2 Ökolopoly-GUI

In Rahmen des vor dieser Arbeit abgeschlossenen Praxisprojekts [Raycheva, 2021] wird für die Gym-Umgebung von Ökolopoly eine geeignete Benutzeroberfläche in PyQt erstellt, die die Abläufe im Spiel wiedergibt. PyQt[4] ist die Python-Version von Qt[5], ein Framework

[4] https://riverbankcomputing.com/software/pyqt/intro
[5] https://www.qt.io/

zum Realisieren unterschiedlichster Arten von Anwendungen mit hochwertigen und flexiblen grafischen Oberflächen. PyQt verfügt über eine Reihe von Klassenbibliotheken, die es ermöglichen, die jeweilige grafische Oberfläche nicht nur benutzerfreundlich, sondern auch ästhetisch durch Einsatz von Templates zu gestalten, was in den implementierten Funktionalitäten der GUI zu sehen ist. Daher stellt sich PyQt als passender Ansatz heraus, das Spiel für eigene Testzwecke verfügbar zu machen.

Ökolopoly wird bereits für Rechner bereitgestellt im Form von Ecopolicy[6] als offizieller PC-Version. Es gibt jedoch noch eine ältere Variante des Spiels[7] , die kleine Unterschiede zu der wirklichen Spielbrett-Version enthält.

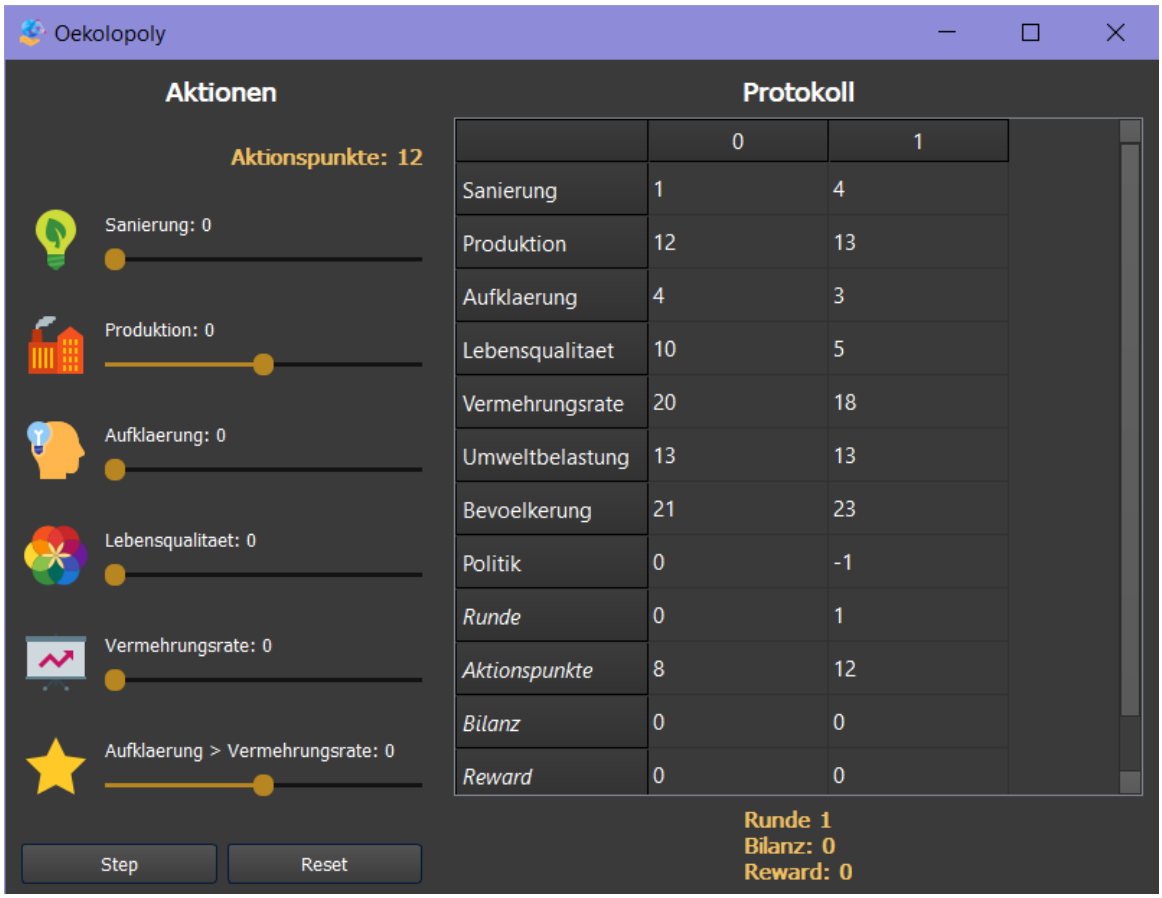


Abbildung 4 GUI vom Spiel Ökolopoly, Screenshot aus Ökolopoly

Abbildung 4 veranschaulicht die realisierte Benutzeroberfläche. Die Slider verfügen über eingebaute Constraints, die den Nutzer begrenzen, mehr als die vorhandenen Punkte für die jeweilige Runde zu verteilen. Die Anzahl der verfügbaren Aktionspunkte ändert sich dynamisch bei Interaktion mit den Slidern. Nachdem der *Step*-Button angeklickt wird, erscheint ein neuer Eintrag in der Tabelle links, der den neuen Spielzustand darstellt. Das Leeren der Tabelle erfolgt durch *Reset*, was das Spiel auch zu seinen initialen Werten zurücksetzt. Die Benutzeroberfläche von Ökolopoly gibt unter der Tabelle in jeder Runde

[6] https://www.frederic-vester.de/deu/ecopolicy/
[7] https://www.goodolddays.net/box/id%2C416/

zusätzlich Informationen über den aktuellen Status im Spiel an. Am Ende des Spiels wird noch die Ursache für den Abbruch mitgeteilt.

2.3.3 Visualisieren der Lebensbereiche

OpenAI Gym gibt die Möglichkeit zum Visualisieren des erreichten Fortschritts des trainierten Agenten mittels der *render*-Funktion. Für die Ökolopoly-Umgebung eignet es sich, das Ändern der Werte der einzelnen Lebensbereiche und der dafür verwendeten Aktionspunkte zu veranschaulichen. Dies wird in Abbildung 5 gezeigt.

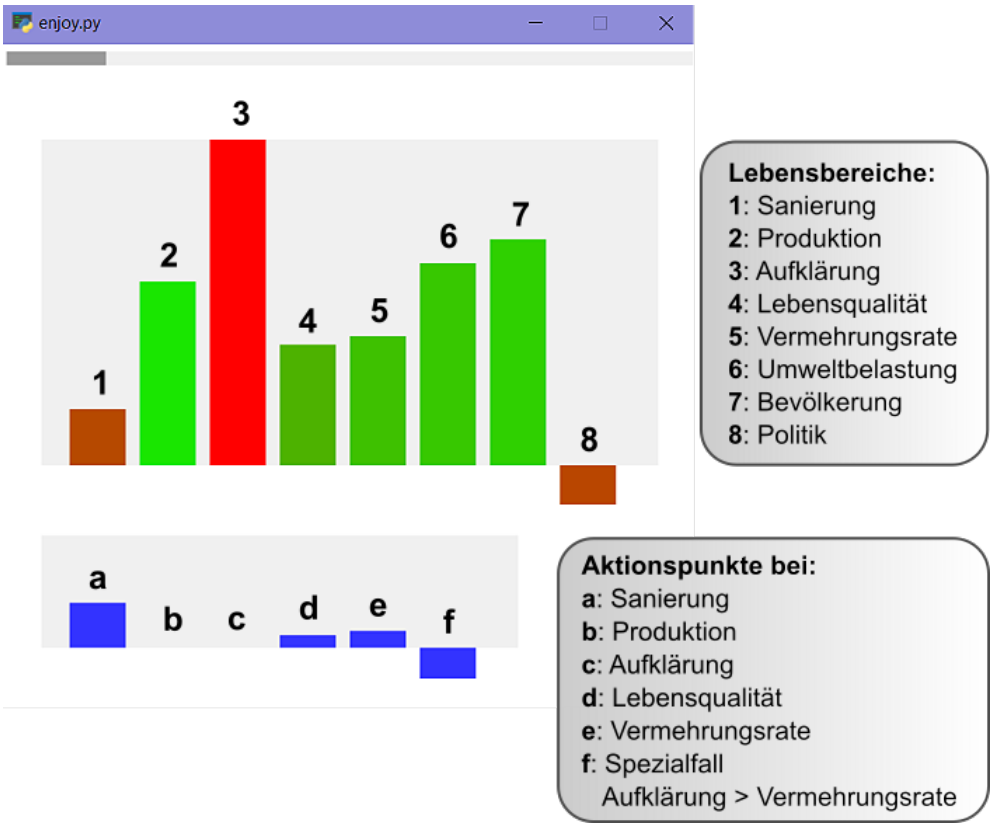


Abbildung 5 Screenshot enjoy.py

Die oberste Leiste in Grau visualisiert die Anzahl der erreichten Runden. Der erste Satz an Balken stellt die Lebensbereiche dar. Die unteren Balken repräsentieren die investierten Aktionspunkte. Die Animation veranschaulicht das Ergebnis aus der ausgeführten Action.

Render_rounds-Funktion: legt das Fenster inklusive dessen Größe beim Ausführen von *enjoy.py* (s. Kapitel 4.4), mittels *self.viewer* fest. Die Variable *t_end* gibt an, dass die Ergebnisse von jeder Runde für 3 Sekunden angezeigt werden. *t_steps* bestimmt mit welcher Geschwindigkeit sich die Animation ändert. Danach werden drei Hilfsfunktionen aufgerufen, die die Position der Elemente auf Bildschirm zeichnen. Diese sind:

- *Render_rounds*: definiert die Größe und Farbe der obersten Leiste. Dafür wird die Funktion *draw_polygon()* benutzt. Sie zeichnet zwei Rechtecke – ein graues und ein dunkelgraues, damit das Ansteigen der Rundenzahl visualisiert werden kann. Das dunklere Rechteck ändert seine Größe dynamisch nach jedem Ausführen der *step-*

Funktion in Abhängigkeit von den vergangenen Runden und überlappt teilweise das darunterliegende.

- *Render_action* und *render_vector*: zeichnen jeweils die Aktions- und Zustandsbalken in Abhängigkeit der Daten aus der vorherigen, der aktuellen Runde und *t*. Der Input Parameter *t* ist eine Hilfsvariable, die der flüssigeren Darstellung der Übergänge zwischen den Zuständen dienen soll. Sie wird von der Hauptrender-Funktion des Environments allmählich erhöht. Bei *t=0*, werden die Daten aus der vorherigen Runde visualisiert, wohingegen bei *t=1* die Informationen aus der aktuellen Runde gezeigt werden. Für Werte von *t* zwischen 0 und 1, wird ein entsprechend gewichteter Durchschnitt der beiden Endzustände dargestellt. Die Größe, Farbe und der Abstand zwischen den Balken werden auch festgelegt. Die Funktionen erstellen auch den grauen Hintergrund.

3 Repräsentation Action und Observation Spaces

Dieses Kapitel beschäftigt sich mit den konkreten Ansätzen und Strategien, mit denen die Aktions- und Zustandsräume von Ökolopoly umgeformt werden, um es dem Reinforcement Learning zugänglich zu machen. Dafür wird die von OpenAI Gym bereitgestellte Klasse *Wrapper* eingesetzt, um zwei Action Wrapper, einen Observation Wrapper und einen Reward Wrapper zu implementieren. Im Folgenden werden sie in dieser Reihenfolge vorgestellt. Es werden die Motivation für ihr Erstellen, ihre grundlegende Codelogik und eventuelle Problemstellen behandelt.

OpenAI Gym bietet neben dem gemeinsamen Interface *Env*, auch eine einheitliche Struktur zur Modifizierung der Aktions- und Zustandsräume mittels so genannter Wrapper. Jeder Wrapper erbt von *Env* und überschreibt deren *step*-Funktion, sodass der neue Aktions- bzw. Zustandsraum eingesetzt wird. Dazu implementieren die Action- und Observation Wrapper jeweils die Funktionen *action(self, action)* und *observation(self, observation)* (s. Abbildung 6). In diesen Funktionen werden die Räume neu modelliert. Für den Reward Wrapper muss die *reward*-Funktion implementiert werden. Im Fall des Action Wrappers

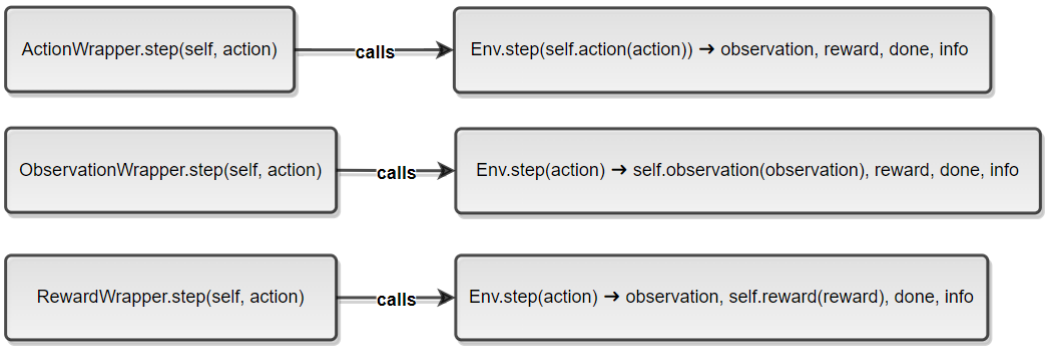


Abbildung 6 Diagramm Funktionsweise Wrapper

ruft er zunächst seine eigene *step*-Funktion auf. Danach wird die *step*-Funktion des originalen Environments ausgeführt, die die transformierte Aktion als Input bekommt. Der Observation Wrapper funktioniert ähnlich und gibt den modifizierten Zustand endlich im Return zurück. Die Umsetzung der gewünschten Belohnung erfolgt durch den Reward Wrapper. Seine *reward*-Funktion gibt den neuen Reward zurück. Die neue Belohnung des Reward Wrappers wird am Ende als Rückgabewert gegeben.

Das Implementieren von Wrapper erweist sich unter folgendem Aspekt besonders vorteilhaft: verschiedene Kombinationen von Aktions- und Zustandsräumen können getestet werden, ohne dass der Code der originalen Umgebung zusätzlich angepasst wird. Auf diese Weise dient das ursprüngliche Environment als Grundlage für weitere Untersuchungen.

3.1 Action Wrapper

Ein wichtiger Grund für die Entwicklung eines passenden Action Wrappers ergibt sich dadurch, dass der Agent aktuell beliebige Aktionen aus dem gesamten Aktionsraum verwenden kann. Seine Wahl kann als valide Action der *step*-Funktion übergeben werden,

sogar wenn diese im Kontext von Ökolopoly illegal ist.

Sei **n** die Anzahl der verfügbaren Aktionspunkte im Spiel. Für jede legale Aktion müssen folgende Anforderungen gelten:

(1) Die verwendeten Aktionspunkte dürfen **n** nie überschreiten.

(2) Falls zu viele Aktionspunkte einer Dimension (Lebensbereich) zugewiesen werden und diese über ihren Range hinausgeht, muss die Anzahl der zugewiesenen Aktionspunkte geclippt werden.

Weiterhin enthält der Aktionsraum bereits Millionen von Aktionen, aus denen der Agent selektieren kann. Wegen der Größe des Action Space und der fehlenden Rücksichtnahme auf die Zulässigkeit der ausgewählten Actions erschwert sich der Lernprozess für den Agenten. Er verliert viel Rechenzeit dadurch, eine gültige Auswahl an Aktionen zu treffen, um im Spiel zu überleben und etwas darüber zu lernen. Auf diese Weise ist es für ihn unmöglich, die Umgebung richtig zu erforschen und eine gute Policy zu erstellen oder den Wert seiner selektierten Aktionen überhaupt zu berechnen.

3.1.1 Box Action Wrapper

Der erste Lösungsansatz, womit diese Problematik umgegangen werden kann, ist im Action Wrapper *OekoBoxActionWrapper* implementiert. Er transformiert den vorliegenden Aktionsraum in einen kontinuierlichen, also vom Typ *Box*. Dieser Typ stellt einen n-dimensionalen Raum dar. Jeder Punkt dieses Raums kennzeichnet eine Aktion. Die Grenzwerte der Aktionen werden durch die Attribute *low* und *high* gesetzt. Die Definition des kontinuierlichen Aktionsraums sieht folgendermaßen aus:

```
34        self.action_min = np.float32 (np.array ([0, -1, 0, 0, 0, -1]))
35        self.action_max = np.float32 (np.array ([1, 1, 1, 1, 1, 1]))
36        self.action_space = gym.spaces.Box (low=self.action_min,
37                                            high=self.action_max)
```

Listing 6 Quellcode Snippet: Action Space in Box Action Wrapper in wrappers.py

Der Aktionsvektor enthält nun 6 Dimensionen, die den Bereichen im Vektor des originalen Umgebung entsprechen (s. Listing 1). Die zweite und letzte Dimension nehmen reelle Zahlen von -1 bis 1 an, alle anderen – von 0 bis 1.

Der folgende Pseudocode fasst die Logik für das Erstellen einer Aktion zusammen:

Transformation Aktionen mit Box Action Wrapper

```
# make sure that not too many points are used
IF sum(action) > 1 THEN
   action[i] /= action[sum]
# transforms action values to discrete ones
action[i] *= points
# reduce action with most points
WHILE sum(action) > points DO max(action)--
IF action[i] + region[i] > region_max[i] THEN
   action[i] = region_max[i] - region[i]
```

Listing 7 Pseudocode: Konstruktion der Aktionen in Box Action Wrapper

Als Erstes wird sichergestellt, dass nicht mehr als die für die gegebene Runde verfügbaren Aktionspunkte verwendet werden. Ist dies der Fall, wird jede Aktion mit der Formel neu berechnet:

$$a_i = \frac{a_i{}'}{\sum_i a_i{}'} \cdot n$$

Die korrigierte Aktion a_i ergibt sich, indem der Wert jeder illegalen Aktion $a_i{}'$ durch die Summe aller Dimensionen im Vektor dividiert wird. Das ganze wird mit der wirklichen Anzahl der verfügbaren Aktionspunkte **n** multipliziert. Somit ist es möglich, die Punkte unter den entsprechenden Lebensbereichen zu verteilen, da sie sich jetzt wieder in einer diskreten Form befinden. Genauer erklärt, die kontinuierlichen Werte der Aktionen stellen den Anteil an Punkte dar, die an den bestimmten Lebensbereich vergeben werden sollen. Die oben aufgeführte Formel stellt sicher, dass die Summe der Anteile nie 1 (100% der verfügbaren Aktionspunkte) überschreitet.

Falls die Gesamtzahl der zu vergebenden Aktionspunkte die maximal verfügbaren übersteigt, wird von dem Bereich auf den die meisten Aktionspunkte verteilt werden sollen, so lange 1 subtrahiert, bis die Obergrenze eingehalten wird. Auf diese Weise führt der Agent nur gültige Spielzüge aus. Die verteilten Aktionspunkte werden geclipped, sollte ihre Vergabe dazu führen, dass ein Bereich über seine Grenzen hinausgeht.

Mit dem Box Action Wrapper wird die Vielzahl von Aktionen im originalen Action Space von Ökolopoly durch theoretisch unendlich viele kontinuierliche Aktionen ersetzt. Damit sollte der Agent beim Einsatz dieses Wrapper während seiner Trainigssession besser generalisieren, da benachbarte Punkte im Box-Raum auf ähnliche oder identische Aktionen zeigen können. Das hilft beim Lernprozess, da er auf diese Weise mehr Aktionen kennenlernt, ohne diese tatsächlich ausprobiert zu haben.

3.1.2 Simple Action Wrapper

Die zweite Möglichkeit für einen potentiell guten Umgang mit den vielen möglichen Aktionen von Ökolopoly könnte sich daraus ergeben, ihre Anzahl signifikant zu reduzieren.

Zu diesem Zweck können die vorhandenen Aktionspunkte für eine Runde in drei gleiche

Teile aufgeteilt und unter den gewünschten Lebensbereichen vergeben werden. Alle legalen Aktionen sind als Zeichenketten in einer Liste erfasst. Jeder String besteht aus insgesamt 6 Zeichen. Die ersten fünf Stellen repräsentieren jeweils die Anzahl der jedem der 5 Bereiche zugewiesenen Anteile an Aktionspunkten. Das sechste Zeichen im String gibt an, ob der Produktion (zweite Stelle im String) die Punkte hinzugefügt (bei 0) oder abgezogen (bei 1) werden. Beispielsweise bekommt beim String ‘210001‘ der erste Bereich (Sanierung) zwei Drittel der vorhandenen Aktionspunkte und dem zweiten (Produktion) wird ein Drittel abgezogen.

Um alle legalen Aktionen richtig in der Liste zu erfassen, werden die ersten 5 Stellen jedes Strings nach dem Vierersystem generiert, also können nur 0, 1, 2, 3 als Zeichen vorkommen. Dabei muss beachtet werden, dass ihre Gesamtsumme 3 nie überschreiten darf. Die möglichen Kombinationen sind dann insgesamt 56. Danach wird das sechste Zeichen nur in den Fällen berücksichtigt, in denen die zweite Stelle (Produktion) einen anderen Wert als 0 erhält. Die daraus resultierenden Kombinationen sind 21. Damit beträgt die Anzahl der Strings in der Liste mit legalen Aktionen insgesamt 77. Der Spezialfall wird genauso wie beim originalen Aktionsraum behandelt, da er diesen nicht weiter erschwert.

Der Aktionsraum wird anschließend vom Typ *MultiDiscrete* angelegt:

```
127         self.action_space = gym.spaces.MultiDiscrete ([77,11])
```

Listing 8 Quellcode Snippet: Action Space von Simple Action Wrapper in wrappers.py

Die erste Dimension im Aktionsvektor stellt den Index eines Elements aus der vordefinierten Liste von Aktionen dar. Die zweite erfasst den Spezialfall Aufklärung > Vermehrungsrate. Auf diese Weise verbleiben dann 847 ($77 \cdot 11$) Aktionen. Die gesamte Logik, wie eine Aktion zur Übergabe an die originale Umgebung erstellt wird, fasst der folgende Pseudocode zusammen:

Transformation von Aktionen mit Simple Action Wrapper

```
#transforms parts to points for the original environment
action[i] = action_string[i] * (points / 3)
IF action[i] + region[i] > region_max[i] THEN
   action[i] = region_max[i] - region[i]
```

Listing 9 Pseudocode: Konstruktion der Aktionen in Simple Action Wrapper

Die Werte aus dem String werden in eine diskreten Form überführt, damit sie unter den Lebensbereichen als valide Aktionspunkte vergeben werden können.

Durch die Verringerung der Anzahl der Aktionen verbessert sich die Erforschung, da wenige Aktionen ausprobiert werden müssen. So wird das Sammeln von Stichproben während des Trainings effizienter. Das Reduzieren des Aktionsraums erfordert jedoch Fachwissen über die Umgebung. Weiterhin kann dies die Fähigkeiten der Agenten einschränken [Kanervisto et al., 2020, S. 2].

Die beiden implementierten Wrapper ermöglichen die sichere Behandlung nur legaler Ak-

tionen im Kontext von Ökolopoly. Zudem kann der Agent entscheiden, ob er Aktionspunkte sparen möchte.

3.2 Observation Wrapper

Der Zustandsraum von Ökolopoly besteht aus Milliarden von Observations. Es soll erforscht werden wie ein Agent mit einem so umfangreichen Zustandsraum umgeht und ob das Lernen durch eine Vereinfachung des Raums erleichtert werden kann.

In dieser Hinsicht können die Werte der einzelnen Lebensbereiche in drei Kategorien (niedrig-mittel-hoch) aufgesplittet werden. Im Observation Wrapper wird der Zustandsraum so definiert:

```
181         self.observation_space = gym.spaces.MultiDiscrete ([3] * self.
                obs_count)
```

Listing 10 Quellcode Snippet: Observation Space von Simple Observation Wrapper in wrappers.py

Obs_count steht für die Anzahl der Lebensbereiche, die in diesem Fall 8 beträgt. Auf diese Weise ergibt sich ein 8-dimensionaler Vektor. Jede Dimension stellt einen Lebensbereich dar und kann die Werte 0, 1 oder 2 annehmen. Insgesamt besitzt der Zustandsraum 6561 Zustände. Nach der Formel

$$obs'_i = \frac{obs_i}{obs'_{max}} \times 3$$

werden die Werte des modifizierten Zustands obs'_i berechnet. obs_i steht für den unveränderten Wert des Lebensbereiches aus dem originalen Zustandsraum, obs'_{max} ist seine Obergrenze. Der vereinfachte Zustand wird am Ende der Runde generiert.
Mit dem Observation Wrapper ist eine höhere Geschwindigkeit beim Lernprozess zu erwarten, da der Agent weniger Zustände zu erforschen hat.

Ein komplettes UML-Klassendiagram der Action- und Observation Wrapper mit ihren zugehörigen Attributen und Funktionen ist in Abbildung 7 zu sehen.

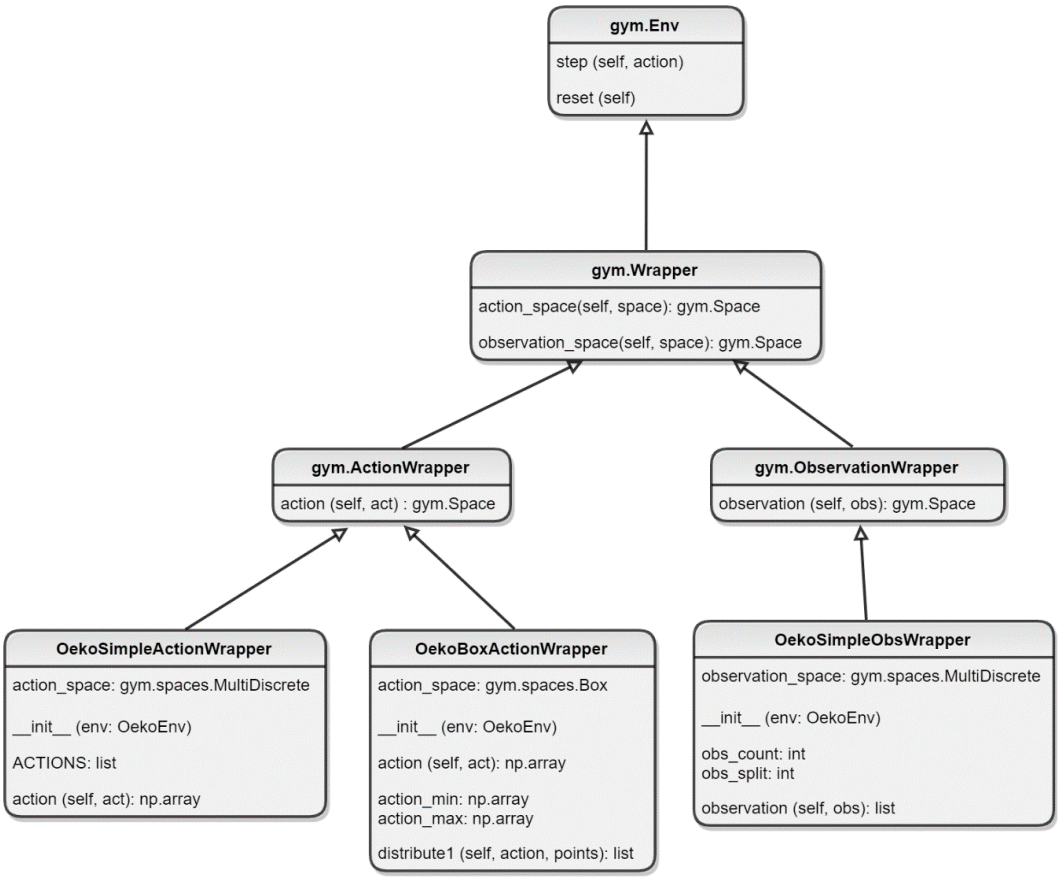


Abbildung 7 Klassendiagramm aller implementierten Wrapper

3.3 Reward Wrapper

Der Reward spielt im RL eine der entscheidendsten Rollen für den Erfolg des Agenten. Ein passender Reward erleichtert den Lernprozess, beschleunigt das Training und führt zu einem frühen Erreichen guter Ergebnisse. Das Erstellen einer geeigneten Belohnung kann aber eine Herausforderung sein, da es vom Entwickler gewisses Domänenwissen verlangt. Wesentlich bei dieser Arbeit ist, die Auswirkungen des Rewards auf das Environment in Kombination mit den verschiedenen Wrappern genauer zu erforschen. Dafür müssen passende Belohnungen entwickelt werden. Diese müssen sich sowohl gleichzeitig klar voneinander unterscheiden, als auch etwas zum Lernprozess des Agenten beitragen können.

Im Kontext von Ökolopoly kann die am Ende erhaltene Bilanzzahl (s. Gl. 1) als erste Reward-Variante übernommen werden. Einerseits gibt diese Rewardstruktur dem Agenten totale Freiheit beim Erforschen der Umgebung, andererseits kann sie ihm schaden, da er die Qualität seiner Aktionen nicht direkt nach jeder Runde erfährt.

Aus diesem Grund ist es sinnvoll, dass ein Reward bei jedem Schritt den Agenten (Ausführen der *step*-Funktion) übermittelt wird. So kann er entscheiden, ob er diesen bei der nächsten Auswahl von Aktionen beachtet. Damit der Agent überhaupt in eine gewinnversprechende Richtung gesteuert wird, muss die Belohnung auf einer bereits guten Spielstra-

tegie basieren. Darauf aufbauend ist der Hilfreward konzipiert.

Eine möglich erfolgreiche Spielpolicy zeigt, dass die Produktion und Bevölkerung große Auswirkung auf die anderen Lebensbereiche haben. Wenn diese beide Bereiche mittlere Werte annehmen, überlebt der Spieler mehr Runden und erzielt am Ende eine gute Bilanzzahl. Aus diesem Prinzip stammt folgende Formel zur Berechnung des neuen Hilfsrewards:

$$R_{prod} = 14 - |15 - V_{prod}|$$
$$R_{bev} = 23 - |24 - V_{bev}|$$
$$R = 14 - |15 - V_{prod}|$$

Solche Arten von Belohnungen steuern den Agenten implizit in eine gewünschte Richtung, was nicht unbedingt zum Erfolg führt. Er wird gewissermaßen manipuliert, eine geeignete Policy zu erstellen, die die bestimmten intermediären Rewards ständig maximiert. Um diese Beschränkung zu lockern, kann der intermediäre Reward erweitert werden und als die Summe des Hilfsrewards und Bilanzzahl festgelegt werden. Vermutlich sollte der Agent somit mehr Freiheit beim Erstellen seiner Policy bekommen.

Nicht alle Kontexte verlangen aber einen komplizierten oder domänenverbundenen Reward. Ein simpler Reward von 1 für jede Runde, die der Agent überlebt, könnte reichen. An dieser Stelle bietet sich die Möglichkeit an, die Auswirkung einer solchen Belohnung in der Ökolopoly-Umgebung zu erkunden. Anbei ist eine Übersicht aller untersuchten Rewardvarianten:

Name des Rewards	**Reward jeder Runde**	**Reward am Ende**	**Klassenname**
Bilanzzahl	0	B	OekoEnv
Reward 1	1	B	OekoEnvRew2
Hilfsreward	$R_{prod} + R_{bev}$	B	OekoEnvRew3
Hilfsreward + Bilanzzahl	$R_{prod} + R_{bev} + B$	B	OekoEnvRew4

Tabelle 2 Übersicht Rewards

Die Bilanz B (s. Gl. 1) bleibt bei 0 bis die Rundenanzahl mindestens 10 wird. Das gilt auch bei der Vergabe der Summe von dem Hilfsreward und der Bilanz. Diese Bedingung muss eingehalten werden, ansonsten kann die bisher angewandte Strategie im Spiel zu hoch oder zu niedrig geschätzt werden, wenn wenige Runden gespielt sind. Diese Bewertung entspricht allerdings der echten Qualität der Strategie nicht. Daher hält sich das Berechnen der Bilanz an der Vesters Definition fest.

Hilfsreward Wrapper

Der Hilfsreward Wrapper modifiziert die Belohnung nach dem Hilfsreward (s. Tabelle 2, 3. Zeile). Bei dessen Einsatz wurde beobachtet, dass nur die Belohnung aus der originalen Umgebung geloggt wird und nicht die Belohnung aus dem Wrapper. Das Problem ist ausführlich in Kapitel 4.6 behandelt.

Ein sicherer Umweg ist dabei das Anlegen zusätzlicher Klassen, die von der Hauptumge-

bung erben und somit die Funktionalität des Reward Wrappers ersetzen. Sie überschreiben die in der originalen Umgebung definierte Funktion *get_reward()* zur Vergabe des Rewards und setzen Custom-Belohnungen ein. Dieser alternative Lösungsansatz verlangt lediglich das Registrieren jeder zusätzlichen Reward-Klasse als selbstständige Umgebung in OpenAI. Der Vorgang ist in OpenAI Gym glücklicherweise einfach zu realisieren, das geringfügige Abändern zweier Initialisierungsdateien erforderlich ist (im Repository, Ordner *oekolopoly: oekolopoly/__init__.py* und *oekolopoly/envs/__init__.py*).

Eine angenehme Nebenwirkung dieser ausführlichen Vorgehensweise zum Lösen des Problems ergibt sich beim Erstellen der unterschiedlichen Agenten. Ihre Namen machen nun deutlich erkennbar, welcher Reward bei den Trainingssessions in Einsatz war.

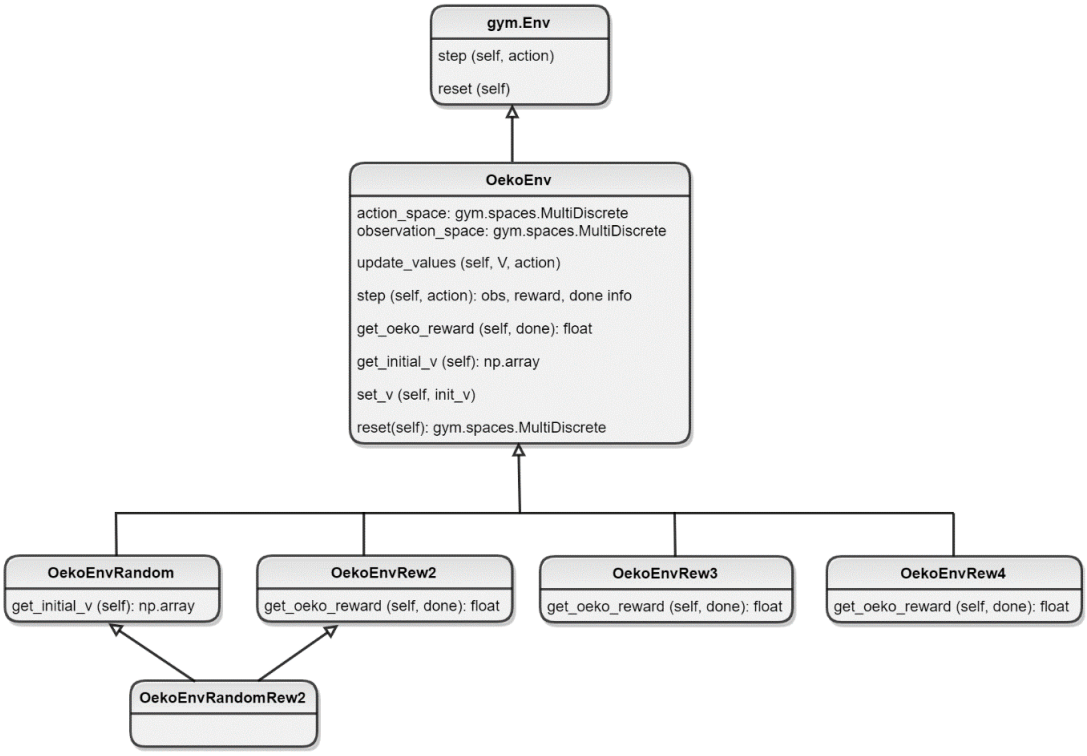


Abbildung 8 Klassendiagramm der verschiedenen Rewards von der Ökolopoly-Umgebung

Schlussendlich sollen die implementierten Wrapper helfen, die oben genannten Probleme zu lösen. Es wird somit erwartet, dass der Agent mit den riesigen Aktions- und Zustandsräumen zurechtkommen kann und die Umgebung lernt. Die zwei bereitgestellten Action Wrapper gehen mit dem Problem der illegalen Aktionen um, indem sie immer die übergebene Aktion korrigieren. Der Observation Wrapper verkleinert die Anzahl der zu beobachtenden Zustände drastisch. Abschließend ermöglichen die verschiedenen Rewards die weitere Untersuchung des Agentenverhaltens.

4 RL Baselines3 Zoo

In folgendem Kapitel werden die Kernfunktionalitäten des Frameworks RL Baselines3 Zoo umrissen. Es wird auf Gründe für seinen Einsatz in der vorliegenden Arbeit und die dabei entstandenen Probleme eingegangen. Schließlich ist eine Installationsanleitung in dem speziell für diese Ausarbeitung angelegten Repository auffindbar (Link dazu im Anhang).

Das RL Baselines3 Zoo[8] Framework bietet eine geeignete Grundlage, auf der Agenten nach einheitlichen Richtlinien in beliebigen Environments von OpenAI Gym trainiert werden können. Zu diesem Zweck stehen spezielle Python-Skripte zur Verfügung, die sowohl das Abspeichern von Daten über den Trainingsfortschritt, als auch dessen grafische Darstellung ermöglichen. Weiterhin verschafft die realisierte Schnittstelle zu Tensorboard eine weitere Erleichterung hinsichtlich des Veranschaulichens von zusätzlichen Informationen über das gesamte Training und den Erfolg jedes Agenten.

4.1 Anbinden eigener Umgebung

Die Ausrichtung des Frameworks erlaubt das Anbinden von eigenen Custom Environments, sobald diese den Implementierungsvorschriften von OpenAI Gym folgen. Dieser Prozess ist leicht umgesetzt und verlangt einfach das Hinzufügen der eigenen Umgebung in *utils/import_envs.py* und das Einstellen der Hyperparameters für den jeweiligen Algorithmus.

4.2 Hyperparameter

Ein großer Vorteil von RL Baselines3 Zoo sind die verfügbaren Packages mit bereits erfolgreich vortrainierten Agenten. Ihr Handeln in der jeweiligen Umgebung ist im Form eines Videos mittels des Skripts *enjoy.py* abrufbar. Diese Musterleistungen sind ein weiterer Orientierungspunkt bei der initialen Wahl von passenden Hyperparameter für die eigenen Agenten. Das Vorhandensein des vollen Hyperparamatersatzes von jedem vortrainierten Agenten zeichnet RL Baselines3 Zoo stark gegenüber anderen Frameworks mit ähnlichem Konzept aus. Ferner unterliegen die Hyperparameter relativ oft Tuning zwecks Optimierung. Dieses wird von den Entwicklern des Frameworks selbst durchgeführt. Die bereits getunten Hyperparameter sparen Zeit, Rechenleistung und erleichtern das Trainieren eines erfolgreichen Agenten. Die strukturierte Form der Hyperparameter ermöglicht den Vergleich verschiedener Experimente und somit das Filtern von irrelevanten Parameter.

4.3 Algorithmen

Für das Trainieren der Agenten setzt das Framework die Implementationen der Algorithmen von Stable Baselines3 (SB3) [Hill et al., 2018] ein. Letztere ist als die zuverlässiger entwickelte Verbesserung des originalen Stable Baselines entstanden, ein Fork von OpenAI Baselines. Es wurde als eine simplere und zugängliche Variante von OpenAI Baselines

[8] https://github.com/DLR-RM/rl-baselines3-zoo

entworfen und gestaltet die entnommenen Algorithmen in Richtung Effizienz um.

SB3 verfügt über einen guten Satz an Algorithmen, unter denen die bereits besprochenen PPO, TD3 und SAC für diese Arbeit relevant sind. Tabelle 3 stellt sie mit den von ihnen unterstützten Typen von Aktionsräumen dar.

Name Algorithmus	**Box**	**Discrete**	**MultiDiscrete**
PPO	Ja	Ja	Ja
TD3	Ja	Nein	Nein
SAC	Ja	Nein	Nein

Tabelle 3 Algorithmen und die von ihnen unterstützen Aktionsräumen, angepasst von Stable Baselines3

4.4 Übersicht Hauptskripte

RL Baselines3 Zoo verfügt über eine gut aufgebaute Dokumentation und Codestruktur. Die Python-Skripte sind meistens verständlich und nachvollziehbar entwickelt. Der Großteil der wichtigen Stellen im Code – nämlich Parameter und Funktionen sind mit geeigneten Kommentaren versehen, demnach kann der Code nach eigenem Bedarf angepasst werden. Die unten aufgeführten Skripte sind von RL Baselines3 Zoo bereitgestellt.

Train.py erstellt einen gesonderten Ordner, der Dateien über den neuen Agenten enthält. Diese umfassen *csv*- und *npz*-Dateien für Abspeichern von Logdaten aus dem Trainingssession und simple *yml*-Dateien mit dem Hyperparametersatz des bestimmten Agenten.

Enjoy.py veranschaulicht grafisch die Performance des Agenten nach Abschluss seines Trainings.

Plot-train.py generiert Grafiken, die für einen spezifischen Agenten die durchschnittliche Episodenlänge und den durchschnittlichen kumulativen Reward der letzten 100 Episoden darstellen.

Für PPO bestimmt das der sogenannte *rollout_buffersize*, der sich aus der Multiplikation der Hyperparameters *n_steps* und *n_envs* ergibt. Ist der *rollout_buffersize* beispielsweise gleich 256, werden die durchschnittliche Episodenlänge und die Summe der erhaltenen Rewards nach jedem 256. Zeitschritt aktualisiert.

Bei den Off-Policy Algorithmen wie TD3 und SAC entscheidet dies der Parameter *train_freq* in Abhängigkeit von *learning_starts*. Ist Letzterer gleich 10000, spezifiziert er, dass ab dieser Anzahl von Zeitschritten der Algorithmus seine Daten aktualisiert bzw. der Agent anfängt, zu lernen. Nach Ablauf von *log_interval = 4* Episoden werden die gesammelten Informationen über die letzten *train_freq*-Zeitschritte gezeigt.

All-plots.py erzeugt Graphen je Environment, die die durchschnittlichen Werte (Episodenlänge oder Reward) und die Standardabweichung einer Gruppe von Agenten visualisieren. Dabei kann zusammenfassend ihre gesamte Performance veranschaulicht werden.

Die Agenten können also nach verwendetem Algorithmus gruppiert und so verglichen werden.

4.5 Tensorboard

RL Baselines3 Zoo unterstützt das Toolkit Tensorboard[9] für das Visualisieren der Performance der trainierten Agenten im Stapel. Die Grafiken sind unter der Localhost-Adresse http://localhost:6006/ aufrufbar und können, wie auch die dazugehörigen Daten, heruntergeladen werden (s. Abbildung 9).

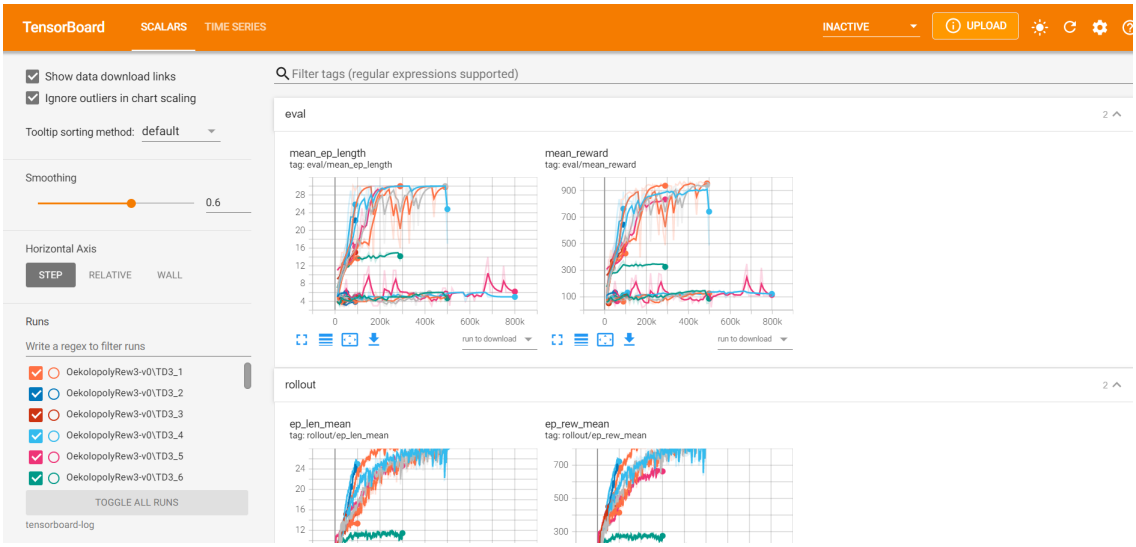


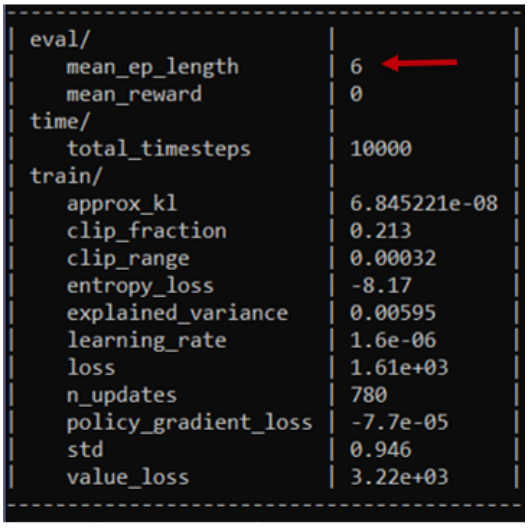
Abbildung 9 Tensorboard Screenshot

4.6 Fallstricke

Im Prozess der Arbeit mit dem Framework wurden einige Stolpersteine und Herausforderungen erkannt. Als erster problematischer Punkt erweist sich die Funktionsweise des Reward Wrappers. Der Wrapper sollte zwar eine gute und übersichtliche Weise anbieten, unterschiedliche Belohnungen dem originalen Environment zu übergeben, der entsprechende Reward wird dadurch allerdings nicht mehr ordnungsgemäß geloggt. Das ruft den unangenehmen Effekt hervor, dass die Werte der ermittelten Rewards während des Rundenablaufs dem Entwickler verborgen bleiben und die Grafiken eventuell nicht gemäß der richtigen Daten erzeugt werden. Beim Einsatz des ersten implementierten Reward Wrappers werden genau die Abweichungen bzgl. des Loggings der Rewards beobachtet. Die Ausgabe in der Konsole und später die Grafiken in Tensorboard, inklusive der mit den Skripten von RL Baselines3 Zoo *train_plot.py* und *all_plots.py* erstellten, zeigen nur die Belohnung aus der originalen Umgebung, nicht den Wert aus dem Reward Wrapper. Obwohl der Wert des Reward Wrappers nicht korrekt geloggt wird, beeinflusst er dennoch den Trainingsprozess des Agenten. Das kann durch die Angabe eines passenden Rewards im Reward Wrapper gezeigt werden. In einem solchen Fall sollte sich die Anzahl der gespielten Runden erhöhen, weil der Reward des Wrappers den Agenten beim Lernen helfen sollte.

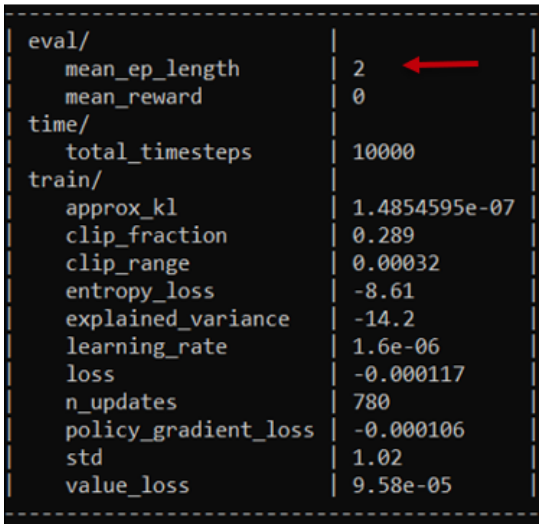
[9] `https://www.tensorflow.org/tensorboard`

Wenn der Reward im Wrapper 0 beträgt, müsste es den Agenten schwerfallen, mehrere Runden zu überleben.



eval/	
mean_ep_length	6
mean_reward	0
time/	
total_timesteps	10000
train/	
approx_kl	6.845221e-08
clip_fraction	0.213
clip_range	0.00032
entropy_loss	-8.17
explained_variance	0.00595
learning_rate	1.6e-06
loss	1.61e+03
n_updates	780
policy_gradient_loss	-7.7e-05
std	0.946
value_loss	3.22e+03

Abbildung 10 Ergebnisse mit Reward Wrapper



eval/	
mean_ep_length	2
mean_reward	0
time/	
total_timesteps	10000
train/	
approx_kl	1.4854595e-07
clip_fraction	0.289
clip_range	0.00032
entropy_loss	-8.61
explained_variance	-14.2
learning_rate	1.6e-06
loss	-0.000117
n_updates	780
policy_gradient_loss	-0.000106
std	1.02
value_loss	9.58e-05

Abbildung 11 Ergebnisse ohne Reward Wrapper

Der Reward im originalen Environment ist auf 0 gesetzt, damit er keine unnötige Auswirkung auf die kleinen Experimente hat. Beide Agenten werden mit den gleichen Hyperparametern, Algorithmen und der gleichen Anzahl von Zeitschritten trainiert. In Abbildung 10 werden die Ergebnisse unter Einsatz des Hilfsrewards (s. Tabelle 2) im Reward Wrapper aufgeführt. In Abbildung 11 wird die Belohnung im Reward Wrapper auf 0 eingestellt. Es fällt auf, dass der Agent dreimal mehr Episoden spielt, wenn er den Hilfsreward bekommt. Dieser Reward wird aber nicht geloggt und *mean_reward* bleibt 0. Somit bestätigt sich die getroffene Aussage über das Problem. Es ist wichtig dieses zu lösen, damit die kumulierte Summe der Rewards für das spätere Erzeugen der Grafiken richtig veranschaulicht wird. Wegen der Zeitbeschränkung und aufgrund der Einfachheit und Sicherheit wurde eine Alternative zu der auf der Issue-Seite[10] vorgeschlagenen Lösung erarbeitet (s. Abschnitt Hilfsreward Wrapper).

Eine weitere Schwachstelle ergibt sich bei dem Erzeugen von Visualisierungen über den zusammenfassenden Fortschritt einer Gruppe von Agenten. Obwohl Tensorboard sich als ein mächtiges Tool für Grafiken auszeichnet, zeigt es die Ergebnisse jedes einzelnen trainierten Agenten. In diesem Fall stellt das Skript *all-plots.py* die Möglichkeit bereit, ein gesamtes Bild über das Verhalten einer Gruppe von Agenten zu erhalten. Das bezieht sich hauptsächlich darauf, die durchschnittliche Episodenlänge oder die kumulierte Reward-Summe insgesamt mit der Standardabweichung für eine bestimmte Stichprobe von Agenten zu visualisieren. Dafür sind minimale Modifizierungen im Code erforderlich. Eine ist die Erweiterung der Parameterliste um ein weiteres Argument *-k* (*ep_length*), damit die durchschnittliche Episodenlänge veranschaulicht werden kann. Eine andere Anpassung ermöglicht es, Agenten, die in unterschiedlichen Umgebungen trainiert worden sind, in einem gemeinsamen Graph darzustellen. Für diese Arbeit werden allerdings eigene Skripte für das Generieren von Grafiken verwendet, die die Daten aus Tensorboard bekommen.

[10] `https://github.com/DLR-RM/stable-baselines3/issues/146`

4.7 Fazit

Abschließend stellt sich RL Baselines3 Zoo als ein hervorragendes Framework dar, womit erste Schritte im Reinforcement Learning vorgenommen werden können. Sein Grundkonzept, Einheitlichkeit beim Trainieren von Agenten und Generieren eines Satzes von vergleichbaren Benchmarks zu erreichen, ist effizient realisiert. Mit seiner simpel aufgebauten Anleitung und klaren Codestruktur verläuft das Einrichten des Frameworks schnell und reibungslos. Weiterhin können unterschiedliche Umgebungen, einschließlich eigener solche, mit state-of-the-art RL-Algorithmen durchgetestet werden. Diese Gelegenheit bietet einen erweiterten Spielraum für zukünftige Untersuchungen an. Die einsteigerfreundliche Eigenschaft ergibt sich weiterhin dadurch, dass alle zugrundeliegenden Komponenten des Reinforcement Learnings (Agent, Umgebung, Algorithmus) bereits implementiert sind und durch Ausführung der bereitgestellten Skripte in Aktion gesetzt werden können. Ebenfalls ist die Leistung der trainierten Agenten in Form unterschiedlicher Grafiken darstellbar – entweder durch die in dem Repository dafür bereitgestellten Skripte (*plot-train.py, all-plots.py*) oder mittels Tensorboard. Die Unterstützung von Tensorboard erweist sich deswegen als eine weitere Stärke des Framework, da es nützliche Daten über die einzelnen Agenten visualisiert. Aus diesen Gründen hat sich RL Baselines3 Zoo als eine geeignete Lösung erwiesen, Agenten auf der Ökolopoly-Umgebung zu trainieren.

5 Evaluation

Dieses Kapitel dokumentiert die Ergebnisse der trainierten Agenten unter dem Einfluss der implementierten Wrapper aus Kapitel 3. Die Kriterien und die Vorgehensweise beim Evaluieren ihres Erfolgs werden erläutert. Weiterhin wird die Strategie der besten Agenten tiefer analysiert und in Vergleich zu einer menschlichen Spielweise gesetzt. Ihre Robustheit wird dabei auch geprüft.

5.1 Kriterien

Das entscheidende Kriterium, mit dem die gelernte Strategie eines Agenten bewertet werden kann, ist die aus der Spielanleitung stammende Skala der **Bilanz** (s. Tabelle 1). Da es im Spiel ohne passende Taktik schwierig ist zu überleben, ist die Anzahl der erreichten Runden (Episodenlänge) auch aussagekräftig für den erreichten Erfolg. Diese muss zwischen 10 und 30 liegen.

Eine bedeutende Kenngröße kann die **Summe des kumulierten Rewards** über die Trainingssession sein. Beim Reinforcement Learning steht das Maximieren des Rewards im Mittelpunkt. Die erhaltenen Belohnungen kontrollieren die Richtung der Entwicklung eines Agenten in seinem Lernprozess. Für ihn sind sie ein Orientierungspunkt, damit er die Auswirkungen seiner gewählten Aktionen in der Umgebung einschätzen kann. Je größer die Summe der Rewards wird, desto besser sollte sich das Verhalten der Agenten entwickelt haben.

Als wesentlich beim Evaluieren der Performance eines Agenten kann sich seine **Laufzeit** erweisen. Das Training sollte sich nicht über Tage oder Wochen strecken, damit positive Ergebnisse erreicht werden können, gegeben die implementierten Wrapper.

5.2 Analyseprogramm

Für eine vertiefte Analyse der Performance der einzelnen Agenten wird ein Analyseprogramm (s. Abbildung 12) in PyQt implementiert. Die erstellte GUI ist in Abbildung 12 zu sehen.

Der Programm ermöglicht es, die Startpositionen im Spiel (rechts) beliebig zu ändern und somit das gelernte Verhalten des Agenten in verschiedenen Szenarien zu testen. Seine Züge werden dann links in der oberen Hälfte der Tabelle aufgeführt, wohingegen in der unteren der daraus resultierende Spielzustand erscheint.

In diesem Abschnitt wird ein Überblick über die dieser Oberfläche zugrunde liegenden Funktionen vermittelt. Der Quellcode befindet sich in dem Repository, das speziell für diese Arbeit angelegt ist (s. Anhang). Die grundlegenden Funktionen im Quellcode werden im Folgenden erklärt.

Die Funktion *clear()* löscht die Einträge aus der Tabelle. Der bereits gewählte Agent bleibt immer noch für Testen zur Verfügung, bis ein neuer geöffnet wird.
Mit *predict()* kann der trainierte Agent Ökolopoly bis zum Ende spielen (entspricht dem Play-Button in der GUI) oder Schritt für Schritt Züge vornehmen (Predict-Button).

Abbildung 12 GUI vom Analyseprogramm, Screenshot aus Analyseprogramm

Zunächst werden die in der GUI eingegebenen Initialwerte aufgenommen und für das Zurücksetzen des Gym Environments verwendet. Die Werte der Zustände werden dann gespeichert und an die *predict*-Funktion von Stable Baselines3 als Eingabeparameter übergeben. In Folge wird eine deterministisch vorhergesagte Aktion erhalten, die in die *step*-Funktion der Ökolopoly-Umgebung für den Zustandsübergang eingesetzt wird. Die Tabelle wird um einen Eintrag erweitert und der unten stehende Text entsprechend aktualisiert. Der Text beinhaltet Informationen darüber, warum das Spiel beendet wurde und solche über die erhaltene Bilanz und Belohnung aus der letzten Runde.
In *open_model()* muss als Erstes die *.zip*-Datei eines gewünschten Agenten ausgewählt werden. In Abhängigkeit davon wird die passende Umgebung geladen. Falls ein Wrapper benötigt wird, wird dieser importiert und er umhüllt das Environment. Danach wird der dazugehörige Algorithmus eingefügt, der das Model (den Agenten) erstellt. Beim erfolgreichen Laden der Agenten werden die Knöpfe in der GUI Play, Predict, Clear aktiviert, diese sind damit einsatzbereit. In main() werden die erforderlichen Layouts und Widgets gesetzt.

5.3 Experimente

Am Anfang wird der Versuch gemacht, einen Agent in der originalen Ökolopoly-Umgebung mit dem Algorithmus PPO zu trainieren. Dies hat zum Systemabsturz beim 9984. Zeitschritt und somit zu komplettem Misserfolg geführt. Das könnte an den in Kapitel 3 erwähnten Herausforderungen in Bezug auf die riesigen Räume von Ökolopoly liegen. Wegen Zeitbeschränkung konnte jedoch der konkrete Grund dafür nicht gefunden werden.

Der Einsatz der implementierten Wrapper ist somit nötig. Ihr Einfluss auf den Erfolg der Agenten wird in den nächsten Abschnitten dargelegt. Die eingestellten Hyperparameter für jeden Algorithmus befinden sich im Anhang.

Die Ergebnisse von allen trainierten Agenten werden in Tabellen 4, 5, 6 erfasst. Jede Tabellenzelle beinhaltet die durchschnittliche Rundenanzahl und Bilanz mit der zugehörigen Standardabweichung für eine Gruppe von Agenten, die mit den selben Konfigurationen (Wrapper und Reward) trainiert sind. Diese Daten werden manuell gesammelt, indem jeder Agent im Analyseprogramm geladen wird. Für ihn wird dann geprüft, wie viele Runden und welche Bilanz er erreicht, wenn er das Spiel ausgehend von dem gegebenen Startzustand bis zum Ende spielt.

Die Grafiken werden mittels der geloggten Daten in Tensorboard erzeugt. Für deren Erstellen wird die Evaluationsmethode von Stable Baselines3 verwendet, die nach jedem 10000. Timestep das Training aufhält und das Spiel basierend auf dem bisher gelernten Verhalten fünfmal deterministisch ausführt.

5.3.1 PPO Ergebnisse

PPO unterstützt sowohl diskrete als auch kontinuierliche Räume. Somit können alle möglichen Kombinationen der implementierten Wrapper mittels PPO getestet werden. Dabei werden 10 Agenten je Wrapper bzw. Kombination von Wrapper für 800000 Zeitschritte trainiert.

Abbildung 13 PPO: durchschnittliche Episodenlänge je Wrapper

Die Grafiken in Abbildung 13 zeigen die durchschnittliche Episodenlänge während des Trainings der Agenten unter Einfluss unterschiedlicher Rewards.

Es lässt sich daraus entnehmen, dass sich die Belohnungen Hilfsreward und Hilfsreward + Bilanzzahl in einem kontinuierlichen Aktionsraum trotz vorhandenen Schwankungen sehr gut auf die Überlebensfähigkeit der Agenten auswirken. In diesem Fall ist auch eine geringe Streuung der Ergebnisse zu sehen. Wenn die Aktionen hingegen diskret sind, trägt überwiegend der Hilfsreward zu der Überlebensfähigkeit der Agenten bei.

Im Gegensatz dazu beeinflusst die Bilanzzahl bei beiden Arten von Aktionsräumen das Überleben der Agenten nicht. Das könnte sich durch die Tatsache erklären, dass der Agent erst am Ende der Episode die Auswirkung der von ihm gewählten Aktionen erfährt. Dabei ist zu beachten, dass laut der Spielanleitung bis zu der 10. Runde die Bilanzzahl 0 beträgt. Dies erschwert weiter den Lernprozess. Im Unterschied dazu erhält der Agent den Hilfsreward in jeder Runde.

Als nächstes werden die Wrapper im Einsatz mit verschiedenen Rewards gegenübergestellt (s. Abbildung 14).

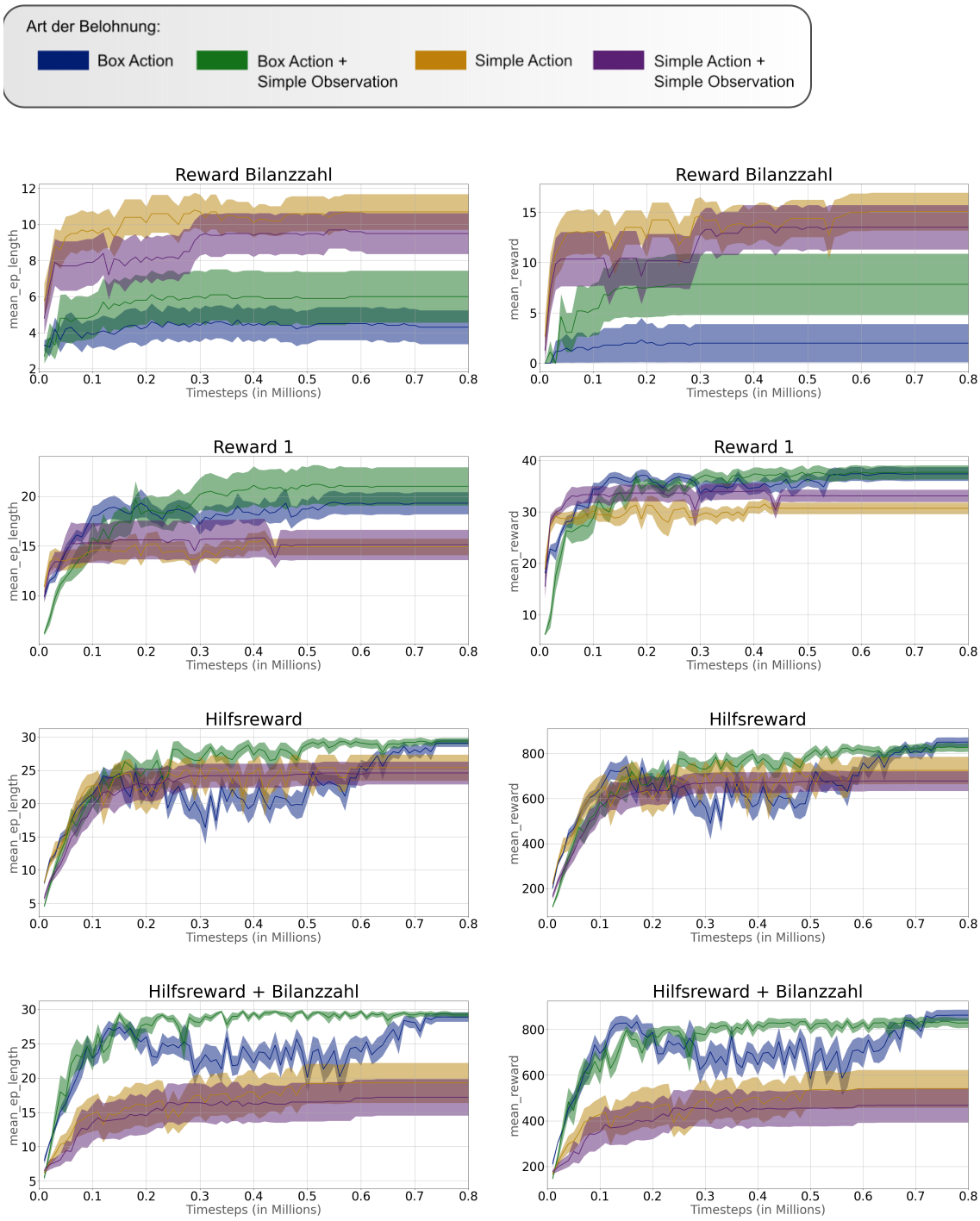


Abbildung 14 PPO: durchschnittliche Episodenlänge und Summe kumulierter Rewards je Belohnung

Es ist zu erkennen, dass sich in allen Fällen der Box Action Wrapper und die Kombination aus Box Action und Simple Observation Wrapper trotz der beobachteten Schwankungen positiv auf das Überleben und Maximieren der Summe des kumulierten Rewards auswirken. Dies geschieht unabhängig von der Belohnungsart. Bemerkenswert ist, dass unter dem Einfluss des Hilfsrewards alle Wrapper ähnliche Ergebnisse erzielen. Wird hingegen die Bilanzzahl als Reward vergeben, weisen diejenigen Agenten bessere Performance auf, die über den vereinfachten diskreten Aktionsraum (Simple Action Wrapper) verfügt haben. Ebenfalls erwähnenswert ist die Tatsache, dass diese Art von Action Space gegen

Ende des Trainings eine weitestgehend glatte Kurve aufweist. Das liegt vermutlich daran, dass der Agent nach einer gewissen Zeit keine neuen Aktionen mehr exploriert, sondern bei der erstellten Policy bleibt.

Die Ergebnisse aller trainierten Agenten sind in Tabelle 4 erfasst.

Wrapper	**Bilanzzahl**	**Reward 1**	**Hilfsreward**	**Hilfsreward + Bilanzzahl**
Box Action	4.3 ± 3.1 Runden	19.3 ± 3.7	**29.1 ± 1.9**	28.9 ± 2.3
	3 ± 6.8 Bilanz	**19.08 ± 3.1**	13.2 ± 1.1	12.1 ± 1.9
Box Action + Simple Observation	6 ± 4.8 Runden	21 ± 6.4	**29.4 ± 1.6**	29.2 ± 1.3
	7.8 ± 10.2 Bilanz	**17.7 ± 5.4**	10.5 ± 2.3	10.3 ± 4.1
Simple Action	10.7 ± 3.3 Runden	14.9 ± 2.8	**25.4 ± 6.6**	19.3 ± 9.7
	15.06 ± 6.3 Bilanz	**16.8 ± 3.8**	16.7 ± 3.5	11.7 ± 8.3
Simple Action + Simple Observation	9.5 ± 3.8 Runden	15.1 ± 5.1	**24.6 ± 5.7**	17.2 ± 8.9
	13.5 ± 7.4 Bilanz	**19 ± 2.5**	12.4 ± 2.8	8.8 ± 6.7

Tabelle 4 PPO: durchschnittlich erreichte Rundenanzahl und Bilanz

Die Agenten, die die höchste Bilanz mit minimaler Standardabweichung erreichen, sind jene, die während der Trainingssessions den Box Action Wrapper und die Kombination aus Simple Action und Observation Wrapper verwenden und Reward 1 bekommen. Ebenfalls ist der Tabelle zu entnehmen, dass der Hilfsreward die Überlebensfähigkeit der Agenten stark positiv beeinflusst. Dafür werden allerdings niedrigere Bilanzzahlen erreicht.

Ein wesentlicher Punkt zur Diskussion stellt hier der Einsatz vom Hilfsreward dar. Wie bereits erwähnt, fördert dieser den Agenten die Werte von Produktion und Bevölkerung in einem mittleren Range zu halten, damit eine höhere Rundenzahl r im Spiel erreicht wird. In dieser Hinsicht stellt die von Vester definierte Bilanzzahl eine gewisse Ungerechtigkeit dar. Da im Nenner durch $r + 3$ dividiert wird, vermindert sich die erzielte Bilanz, immer wenn mehr Runden überlebt werden. So ist es schwieriger am Ende des Spiels gleichzeitig eine hochbewertete Strategie und höhere Rundenanzahl erreicht zu haben.

Weiterhin ist noch relevant, die Entwicklung der Bilanz während des Trainings zu veranschaulichen. Dazu dienen die nächsten Grafiken (s. Abbildung 15), die die Ergebnisse der verschiedenen Rewards in Vergleich setzen. Das Logging der Bilanz ist zu einem späteren Zeitpunkt während des Arbeitsprozesses realisiert. Aus diesem Grund stellt jede Kurve nur einen Agenten dar, der mit der entsprechenden Konfiguration (Reward und Wrapper) trainiert wurde. Deswegen kann es sein, dass in einigen Fällen Ergebnisse über dem Durchschnitt zu sehen sind.

Abbildung 15 PPO: Entwicklung Bilanz mit verschiedenen Wrapper

Die Beschriftung der y-Achse *Balance* entspricht der am Ende erhaltenen Bilanzzahl im Spiel.

Bemerkenswert ist, dass die Reward-Variante der Bilanzzahl überall glatte Kurven erzeugt. Hat der Agent einmal eine höhere Bilanz erzielt, behält er diese und bemüht sich, sie zu erhöhen. In Vergleich dazu scheint es, dass die beiden Arten von Hilfsrewards den Agenten zum Ausprobieren verschiedener Aktionen zwingen, damit ihre Bedingungen (Produktion und Bevölkerung in der Mitte zu halten) erfüllt werden.

Weiterhin wird der Einfluss des Hilfsrewards auf die Produktion und Bevölkerung erforscht (s. Abbildung 16). Die Obergrenzen dieser Lebensbereiche betragen 29 bzw. 48. Das Speichern ihrer Werte während des Trainings ist zu einem späteren Zeitpunkt implementiert worden. Deswegen zeigt jedes Paar von Balken die Ergebnisse eines trainierten Agenten je nach Kombination von Wrapper und Reward mit der entsprechenden Abweichung.

Abbildung 16 PPO: Vergleich Werte der Produktion und Bevölkerung mit verschiedenen Rewards

Den Grafiken lässt sich entnehmen, dass sich die Produktion und die Bevölkerung unter Wirkung des Hilfsrewards auf mittlere Werte zubewegen. Im Fall von Simple Action + Simple Observation Wrapper ist der hohe Wert der Bevölkerung die Ursache für den Spielabbruch. Da sich die Daten auf einen Agenten beziehen und es sich daher um einen Einzelfall handelt, ist diese Beobachtung normal.

5.3.2 TD3 Ergebnisse

Im Unterschied zu PPO ist TD3 ein Off-Policy Algorithmus. TD3 unterstützt nur kontinuierliche Aktionsräume, deswegen kann hier nur ein der beiden Action Wrapper erforscht werden. Da die Trainingszeiten wesentlich länger als bei PPO sind, werden hier je Konfiguration lediglich 3 Agenten für jeweils 800000 Timesteps trainiert.

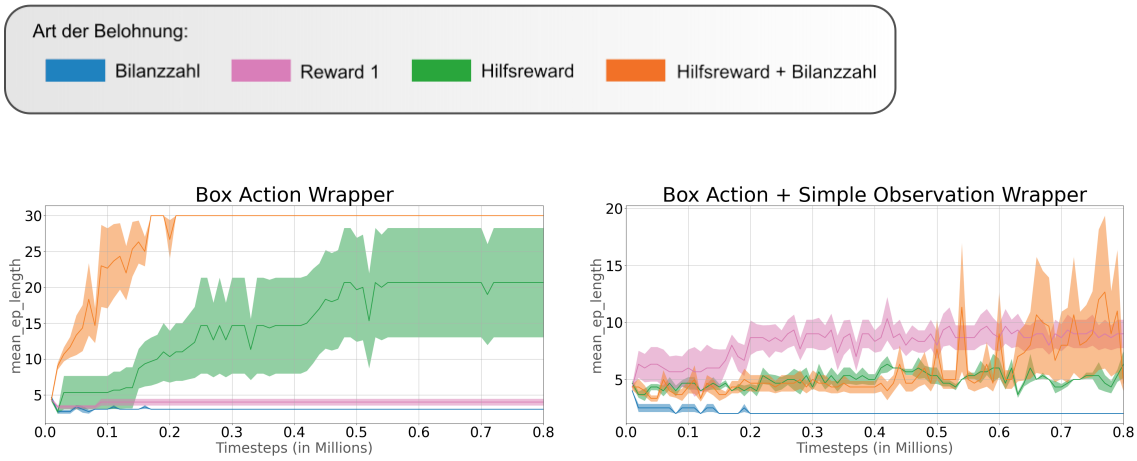


Abbildung 17 TD3: durchschnittliche Episodenlänge je Wrapper

Abbildung 17 ist zu entnehmen, dass der Reward Hilfsreward + Bilanzzahl in Kombination mit dem Action Box Wrapper am schnellsten zum Erreichen der maximalen Rundenanzahl führt. Im Fall des Hilfrewards ist eine große Streuung unter der den damit trainierten Agenten zu beobachten, was die Anzahl der gespielten Runden betrifft. Der Observation Wrapper scheint hier den Erfolg der Agenten zu behindern.

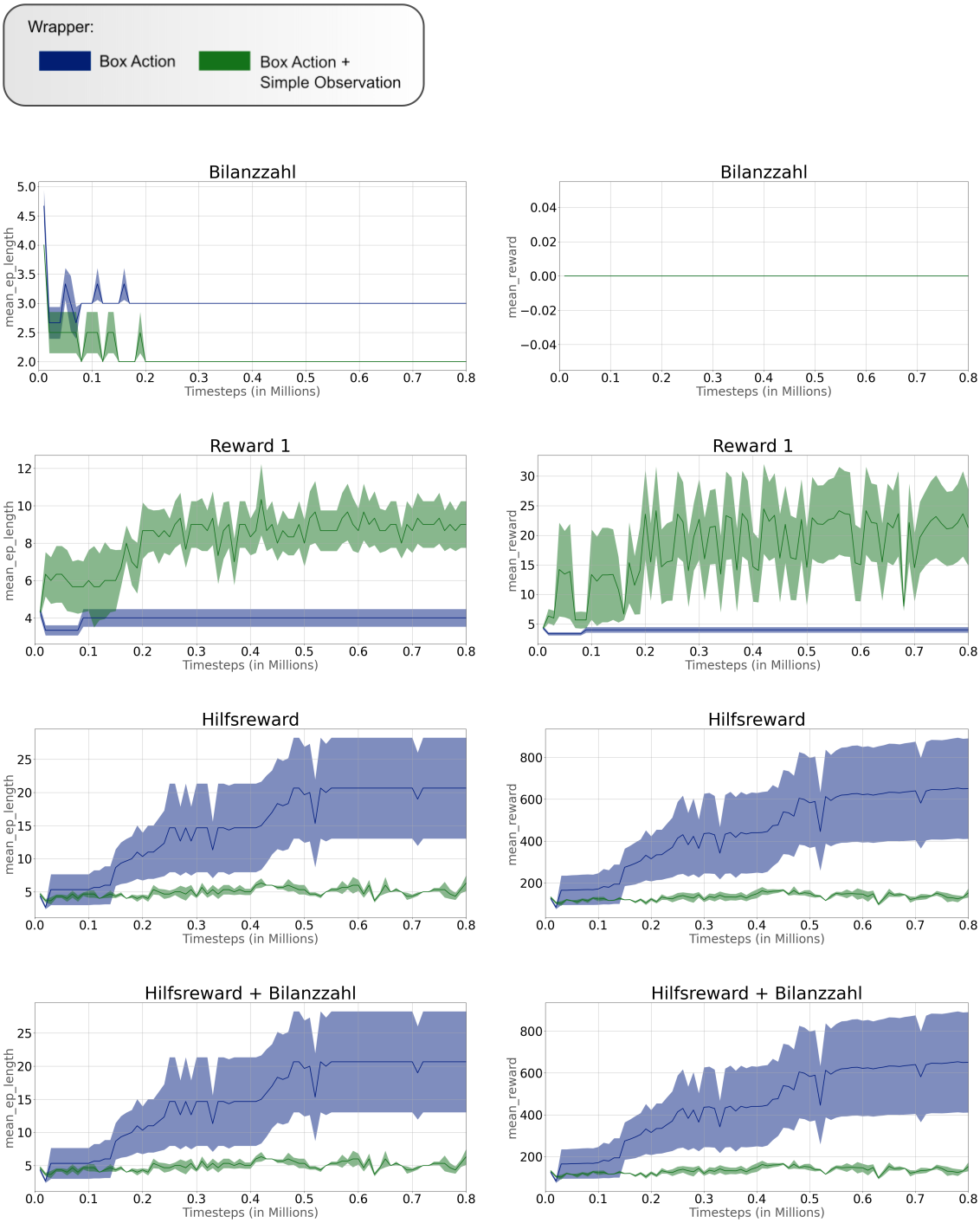


Abbildung 18 TD3: durchschnittliche Episodenlänge und Summe kumulierter Rewards je Belohnung

Anhand von Abbildung 18 ist festzustellen, dass die Agenten unter Einfluss der Bilanzzahl als Reward das Spiel nicht erfolgreich lernen können. Wenn Reward 1 und die beiden Wrapper eingeführt werden, verbessern sich die Ergebnisse erheblich. Die Hilfsrewards arbeiten hingegen nur mit dem Box Action Wrapper gut zusammen. Bei ihnen ist jedoch eine größere Schwankung in den Werten zu beobachten.

Wrapper	Bilanzzahl	Reward 1	Hilfsreward	Hilfsreward + Bilanzzahl
Box Action	3 ± 0 Runden	4 ± 1	20.7 ± 16.7	**30 ± 0**
	0 ± 0 Bilanz	0 ± 0	8.5 ± 7.4	**11.2 ± 0**
Box Action + Simple Observation	2 ± 0 Runden	**9 ± 2.6**	6.7 ± 2.1	4.7 ± 1.5
	0 ± 0 Bilanz	**12.9 ± 11.8**	0 ± 0	0 ± 0

Tabelle 5 TD3: durchschnittlich erreichte Rundenanzahl und Bilanz

Aus Tabelle 5 lässt sich ablesen, dass die Kombination der beiden Wrapper für das Lernen der Agenten nicht erfolgreich ist. Als einzige Ausnahme erweist sich in diesem Falle der Reward 1 als hilfreich. Die Hilfsrewards hingegen zeigen eine positive Wirkung bei der alleinigen Verwendung des Box Action Wrappers. Bezüglich der großen Streuung bei den Agenten, die mit Box Action Wrapper und dem Hilfsreward trainiert sind, muss erwähnt werden, dass zwei aus der gesamten Gruppe 30 Runden spielen und eine Bilanz von circa 13 erzielen. Ähnlich ist bei den Agenten, die mit der Kombination der beiden Wrapper und Reward 1 gelernt haben – einer davon ist komplett erfolglos, während der Rest Bilanzen von 23 bzw. 15.7 für 10 Runden bzw. 11 Runden erreicht.

5.3.3 SAC Ergebnisse

SAC ist ein weiterer Off-Policy Algorithmus und unterstützt nur kontinuierliche Aktionsräume. Da der Einsatz von TD3 überall zu unbefriedigenden Ergebnissen geführt hat, werden an dieser Stelle weitere Untersuchungen durchgeführt. Es sollte dann festgestellt werden, ob Off-Policy Algorithmen mit den für diese Arbeit implementierten Wrapper allgemein weniger erfolgreich sind oder ob das Problem selbst an TD3 selbst liegt.

Hier können nur die zwei Wrapper Box Action und Simple Observation erforscht werden. Wegen der dieser Arbeit gesetzten Zeitbeschränkung sind 3 Agenten je Konfiguration für 800000 Zeitschritte trainiert.

Abbildung 19 SAC: durchschnittliche Episodenlänge je Wrapper

Den Grafiken in Abbildung 19 kann man entnehmen, dass SAC mit dem Box Action Wrapper, dem riesigen Zustandsraum von Ökolopoly und den Hilfsrewards erheblich mehr Spielrunden überlebt. In Gegensatz dazu behindert der modifizierte Zustandsraum die Performance der Agenten.

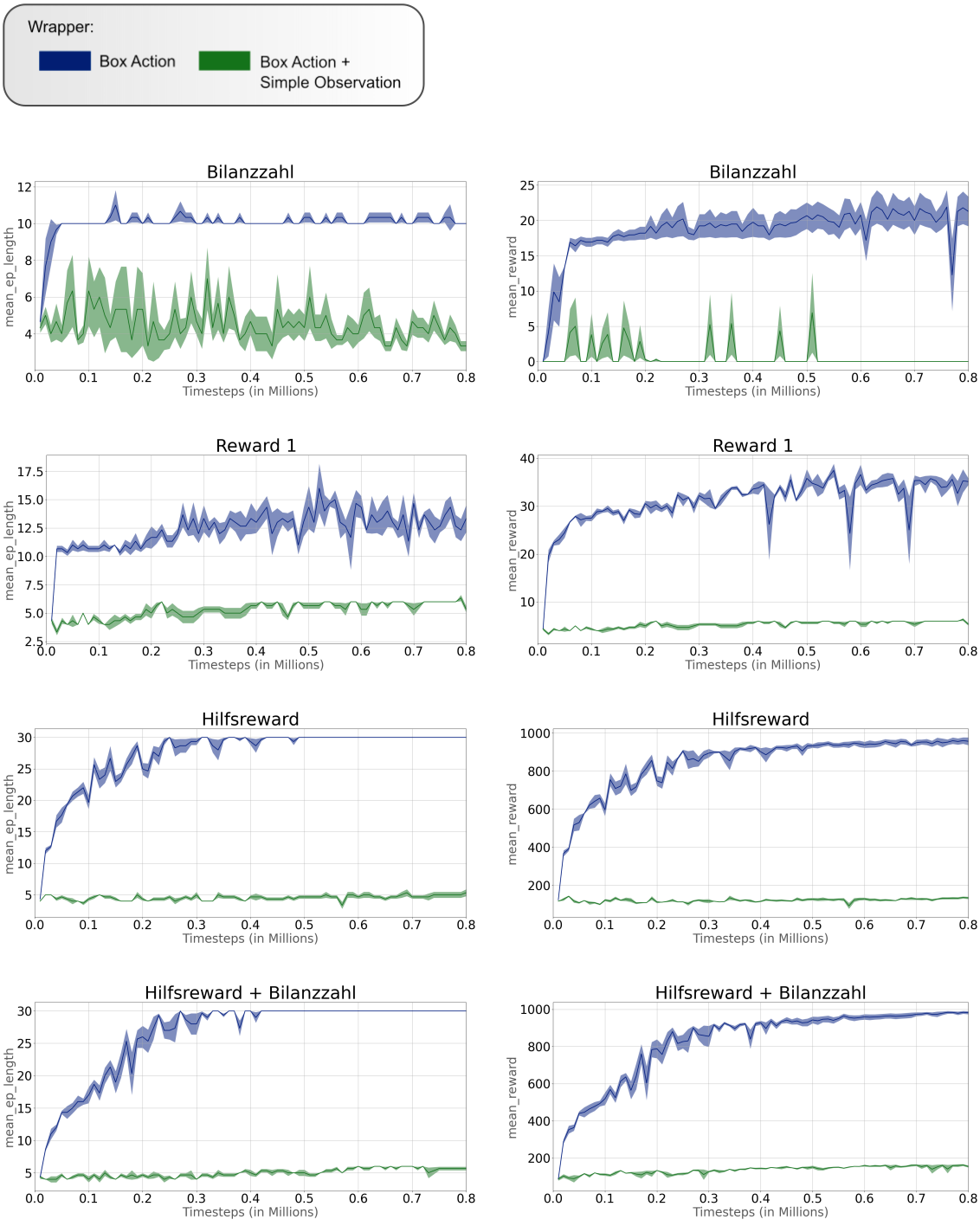


Abbildung 20 SAC: durchschnittliche Episodenlänge und Summe kumulierter Rewards je Belohnung

Basierend auf Abbildung 20 kann festgestellt werden, dass der Box Action Wrapper unabhängig von der Reward-Variante sehr gute Ergebnisse erzielt. Bemerkenswert ist, dass nur geringe Variation unter den trainierten Agenten zu beobachten ist.

Wrapper	Bilanzzahl	Reward 1	Hilfsreward	Hilfsreward + Bilanzzahl
Box Action	10 ± 0 Runden	13.7 ± 2.3	**30 ± 0**	30 ± 0
	18.97 ± 2.5 Bilanz	**22.5 ± 2.9**	12.7 ± 0.9	11.7 ± 0.8
Box Action + Simple Observation	3.3 ± 0.6 Runden	5.7 ± 0.6	5 ± 1	5.7 ± 0.6
	0 ± 0 Bilanz	0 ± 0	0 ± 0	0 ± 0

Tabelle 6 SAC: durchschnittlich erreichte Rundenanzahl und Bilanz

Tabelle 6 zeigt, dass sehr hohe Bilanzzahlen allein mit einem kontinuierlichen Aktionsraum und Reward 1 erzielt werden. Die Hilfsrewards hingegen tragen hauptsächlich dazu bei, mehr Spielrunden zu erreichen.

5.4 Laufzeit

Zu Beginn dieser Arbeit wurden die Experimente auf einem üblichen Rechner durchgeführt. Während PPO wenige Minuten für das Trainieren eines Agenten benötigt, hat sich der Prozess als besonders zeitaufwendig für die Off-Policy Algorithmen erwiesen. Alle Experimente mit PPO, TD3 und SAC wurden daher auf einer von der TH Köln bereitgestellten virtuellen Maschine durchgeführt. Ihr durchschnittlicher Dauer (in Minuten) je nach angewandtem Algorithmus und angewandter Konfiguration (Wrapper, Reward) ist in einer Tabelle im Anhang erfasst (s. Anhang).

Daraus lässt sich ablesen, dass PPO konsistent die kürzesten Zeiten erreicht (durchschnittlich circa 4,3 Stunden für alle trainierte Agenten). Der große Unterschied bzgl. der Dauer bei Simple Action Wrapper und Reward 1 kann sich dadurch erklären, dass vermutlich während der Trainingssessions dieser Agenten ein anderer Prozess auf der Maschine lief, und diese verlangsamte.

Die Off-Policy Algorithmen TD3 und SAC verlangen für das Training von je 3 Agenten dagegen mehr Zeit und gewiss mehr Rechenleistung. Das Abschließen aller Experimente in SAC benötigt ungefähr 3,3 Tagen im Durchschnitt. Bei diesem Algorithmus werden noch große Schwankungen in den Trainingszeiten derjenigen Agenten beobachtet, die mit dem Box Action Wrapper gelernt haben. In Gegensatz dazu dauern die Trainingssessions der Agenten mit der Konfiguration Box Action und Simple Observation Wrapper grob halb so lange. Dafür erzielt der Agent aber keinen Erfolg im Spiel wie in Tabelle 6 zu sehen ist.

TD3 liegt in der Mitte und seine Experimente dauern rund 1,5 Tage.

Zusammenfassend lässt sich noch erkennen, dass der Simple Observation Wrapper entgegen den Erwartungen die Trainingszeiten in den meisten Fällen nicht verkürzt.

5.5 Analyse der besten Agenten

Basierend auf den Ergebnissen aus dem vorherigen Kapitel (s. Tabelle 4) zeichnen sich 2 Arten von Agenten besonders stark aus und werden für weitergehende Untersuchungen

berücksichtigt. Beide bekommen sehr hohe Bilanzzahlen für ihre gelernte Strategien und wurden mit dem Algorithmus PPO und Reward 1 trainiert. Als eine gute Kombination erweist sich der Simple Action mit dem Simple Observation Wrapper. Der gewählte Agent spielt 11 Runden und bekommt dafür Bilanz 22.14 (Agent 1).

Die andere erfolgreiche Konfiguration für das Training ist der Box Action Wrapper alleine. Hier erlangt der konkrete Agent eine Bilanz von 21.67 für 21 Runden (Agent 2).

Agent 1 erreicht Ergebnisse, die einer menschlichen Strategie vergleichbar sind. Im Unterschied dazu überlebt Agent 2 mehr als 20 Runden im Spiel und weist trotzdem eine sehr hohe Bilanzzahl auf. Beide Agenten wecken aus diesen Gründen Interesse. Ihre jeweilige Taktik wird im nächsten Abschnitt aufgezeigt, mit dem Ziel aus ihr zu lernen.

Lebensbereich	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
verfügbare Aktionspunkte	8	11	12	11	10	12	14	17	16	17	24
Sanierung	3	4	4	-	-	-	-	-	6	6	7
Produktion	-	-	-	-	-	-	-	-	-	-	-
Aufklärung	-	-	-	-	-	-	-	12	-	-	-
Lebensqualität	2	3	4	11	10	12	11	5	10	7	8
Vermehrungsrate	-	-	-	-	-	-	-	-	-	-	-
Speziallfall (Aufklärung > Vermehrungsrate)	-	-	-	1	1	1	1	3	3	3	-2

Tabelle 7 Strategie Agent 1

- Grund für Spielende: Bevölkerung überschreitet ihre obere Grenze
- Bilanz: 22.14

Agent 1 (s. Tabelle 7) verteilt zu Beginn seine Aktionspunkte zwischen Sanierung und Lebensqualität. Letztere wird von relativ vielen Lebensbereichen beeinflusst - Umweltbelastung, Aufklärung und Bevölkerung. Aus diesem Grund kann sich ihr Wert rasant ändern. Der Agent investiert dann für die nächsten 5 Runden alle seine Punkte, um diesen Bereich zu stabilisieren. Wenn sich die Lebensqualität in einer guten Lage befindet, werden auch die Anzahl der Aktionspunkte für die nächste Runde positiv beeinflusst. Somit können diese neben der Lebensqualität anderen Bereichen geschenkt werden – Aufklärung und Sanierung. Die vergebenen Punkte beim Spezialfall haben keine Auswirkung, da Aufklärung bis zum Ende unter 21 bleibt. Der Grund für das Spielende, der Aufschwung der Bevölkerung, wird letztendlich von der hohen Lebensqualität verursacht.

Lebensbereich	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**
verfügbare Aktionspunkte	8	9	9	8	8	9	9	11	11	15	15	13	15	18	17	15	19	24	17	25	29
Sanierung	2	-	-	3	-	-	-	-	-	-	-	-	-	3	5	-	5	-	-	6	4
Produktion	3	-3	-3	2	-2	-2	-2	2	-2	-5	7	-3	-5	-4	6	7	-6	-11	2	4	-
Aufklärung	3	3	3	3	3	3	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Lebensqualität	-	3	3	-	3	3	2	4	3	-	-	4	-	4	-	-	2	-	-	-	-
Vermehrungsrate	-	-	-	-	-	1	2	5	3	5	8	4	5	3	6	8	-	10	7	6	8
Spezialfall (Aufklärung > Vermehrungsrate)	-5	3	3	-5	-4	-5	-5	-5	-4	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5

Tabelle 8 Strategie Agent 2

- Grund für Spielende: Politik überschreitet ihre obere Grenze
- Bilanz: 21.67

Agent 2 (s. Tabelle 8) verteilt im Lauf des Spiels gleichmäßig seine verfügbaren Aktionspunkte und konzentriert sich hauptsächlich auf die Produktion. Dieser Bereich hat Einfluss auf die Anzahl der Aktionspunkte pro Runde und die Umweltbelastung, die sich auf die Lebensqualität auswirkt. Bereits in der 5. Runde erhält der Agent Kontrolle über den Spezialfall, was ihm gewisse Freiheit gibt, die Erhöhung der Bevölkerung zu begrenzen. Die Aufklärung befindet sich nun in einer sehr guten Lage. Aus diesem Grund fängt die Lebensqualität an, sich zu erhöhen, was sich als positiv für die Lage der Politik und die Anzahl der Aktionspunkte pro Runde erweist. Der Agent steuert noch die Entwicklung der Bevölkerung indirekt durch die Produktion. Aus der angewandten Strategie kann gefolgert werden, dass der Agent erfolgreich die Rückkopplungen im Spiel erkannt hat. Dies erlaubt ihm mehrere Runden zu überleben und dabei auch eine sehr hohe Bilanz zu bekommen. Das Spielende weist darauf hin, dass sich die Politik sich auf einen sehr hochentwickelten Niveau befindet.

Lebensbereich	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
verfügbare Aktionspunkte	8	9	11	12	10	11	11	14	21	22	21
Sanierung	-	-	-	-	-	9	8	3	2	3	2
Produktion	-	-	-	-6	-2	2	2	2	-5	-5	5
Aufklärung	-	9	11	-	2	-	-	-	-	-	-
Lebensqualität	8	-	-	6	6	-	-	-	-	-	-
Vermehrungsrate	-	-	-	-	-	-	-	2	5	5	-
Spezialfall (Aufklärung > Vermehrungsrate)	-	-	-4	-4	-5	-	-	-	-5	-5	-2

Tabelle 9 Strategie eines Menschen

- Grund für Spielende: Umweltbelastung unterschreitet ihre untere Grenze
- Bilanz: 20

Bei der menschlichen Strategie (s. Tabelle 9) fokussiert sich die Strategie darauf, die Aufklärung so zu erhöhen, dass noch am Anfang Kontrolle über die Vermehrungsrate (Spezialfall) und somit auf die Lage der Bevölkerung gewonnen wird. Als nächstes wird die Produktion so gesteuert, dass sie sich positiv auf die Umweltbelastung auswirkt. Da die Umweltbelastung direkt auf die Lebensqualität wirkt, bleibt Letztere auf diese Weise in einer guten Lage. Die verfügbaren Aktionspunkte werden in Sanierung investiert, die auch die Umweltbelastung beeinflusst. Mit der Lebensqualität und Produktion auf einem gewünschten Niveau werden auch viele Aktionspunkte für jede Runde errechnet. Wenn sie die maximale Grenze von 36 überschreiten, können sie das frühzeitige Spielende verursachen. Dafür werden Punkte der Vermehrungsrate vergeben und danach wieder durch den Spezialfall abgezogen. Umweltbelastung gelangt am Ende an ihre untere Grenze, was bedeutet, dass das Land eine ökologisch nachhaltige Lage erreicht hat.

Es lässt sich zusammenfassen, dass die Strategie von Agent 1 und dem Menschen einem ähnlichen Motiv folgen. Beide konzentrieren sich zu jeder gegebenen Zeit auf sehr wenige oder konkret einen Lebensbereich und investieren die Aktionspunkte so, dass die daraus resultierenden Effekte unmittelbar in der nächsten Runde beobachtet werden. In dieser Hinsicht erweisen sich die beiden Strategien als etwas kurzsichtig, obwohl sie am Ende eine hohe Bilanzzahl erlangen. Im Unterschied dazu konzentriert sich Agent 2 darauf, indirekt die Entwicklung der Lebensbereiche zur gewünschten Richtung zu führen. Auf diese Weise ergibt sich eine sich allmählich und sicher entwickelnde Spielleistung. Die menschliche Strategie kann damit verfeinert werden, um neben einer guten Bilanz auch höhere Rundenanzahl zu erreichen.

Test Robustheit

In diesem Abschnitt werden die beiden besten Agenten auf ihre Robustheit geprüft, indem geänderte Initialwerte im Analyseprogramm eingestellt werden. Dafür werden die Ergebnisse am Ende des Spiels für die Bewertung in Betracht genommen. Die dafür verwendete Ausgangslagen sind in Tabelle 10 zu finden.

Lebensbereich	**Startposition 1**	**Startpos. 2**	**Startpos. 3**	**Startpos. 4**
verfügbare Aktionspunkte	8	8	8	8
Sanierung	1	1	1	1
Produktion	12	9	12	*14*
Aufklärung	4	2	4	4
Lebensqualität	10	7	*12*	*12*
Vermehrungsrate	20	24	20	*22*
Umweltbelastung	13	8	13	13
Bevölkerung	21	24	21	21
Politik	0	0	0	*2*

Tabelle 10 Vergleich verschiedener Ausgangslagen

Die ersten beiden stammen direkt aus der Spielanleitung. Startpositionen 3 und 4 basieren hingegen auf dem originalen Startzustand des Spiels und enthalten einige leicht abgeänderte Werte (markiert in Tabelle 10).

Tabelle 11 veranschaulicht die gespielte Rundenanzahl und erhaltene Bilanz von Agenten 1 und 2 in Abhängigkeit der eingegebenen Startposition.

	Startposition 1	**Startposition 2**	**Startposition 3**	**Startposition 4**
Agent 1	11 Runden 22.14 Bilanz **Grund für Spielende:** Bevölkerung überschreitet ihre obere Grenze	12 16.67 Umweltbelastung unterschreitet ihre untere Grenze	5 0 Vermehrungsrate überschreitet ihre obere Grenze	4 0 Vermehrungsrate überschreitet ihre obere Grenze
Agent 2	21 Runden 21.67 Bilanz **Grund für Spielende:** Politik überschreitet ihre obere Grenze	3 0 Lebensqualität unterschreitet ihre untere Grenze	6 0 Lebensqualität unterschreitet ihre untere Grenze	3 0 Vermehrungsrate überschreitet ihre obere Grenze
Agent 1 (neu)	13 Runden 20 Bilanz **Grund für Spielende:** Lebensqualität überschreitet ihre obere Grenze	11 16.43 Lebensqualität überschreitet ihre obere Grenze	13 20 Lebensqualität überschreitet ihre obere Grenze	13 20 Lebensqualität überschreitet ihre obere Grenze
Agent 2 (neu)	9 0 **Grund für Spielende:** Lebensqualität unterschreitet ihre untene Grenze	9 0 Lebensqualität unterschreitet ihre untene Grenze	22 20.4 Lebensqualität überschreitet ihre obere Grenze	22 20.4 Lebensqualität überschreitet ihre obere Grenze

Tabelle 11 Robustheit Agenten

Hervorzuheben ist, dass Agent 1 ein robusteres Verhalten als Agent 2 gegeben Ausgangslage 2 aufweist. Dies ist überraschend, da Agent 2 mit einem kontinuierlichen Aktionsraum trainiert wurde und in der Regel besser über die Aktionen generalisieren sollte. Es stellt sich allgemein heraus, dass die Agenten mit einer geringfügig veränderten Ausgangslage nicht so gut zurechtkommen und enttäuschende Ergebnisse erreichen. Das lässt sich darauf zurückführen, dass sie das Training ausschließlich nur mit einer bestimmten Startposition absolviert haben und somit überangepasst sind.

Ein Weg, dies zu vermeiden und die Robustheit der Agenten zu stärken, ist die Einführung einer zusätzlichen Startposition während des Trainings. Diese unterscheidet sich wenig von der originalen und hilft den Agenten dabei, andere ähnliche Zustände zu lernen. Agent 1 und 2 werden für 800000 Timesteps neu trainiert. Für dieses Ziel wird die Ausgangslage eines sogenannten Schwellenlandes eingesetzt (Startposition 2, Tabelle 11), die aus der Spielanleitung entnommen ist. Am Anfang jedes neuen Spiels wird eine der beiden Startpositionen (1 oder 2) willkürlich gewählt. Ausgangslagen 3 und 4 kommen während des Trainings nicht vor und dienen als Vergleich dafür, wie gut die Agenten gelernt haben.

Bemerkenswerterweise wirkt sich das Training mit alternierenden Startzuständen positiv auf die Robustheit von Agent 1 aus. Er erzielt die ausgezeichnete Bilanz von 20 bei den Startpositionen 1, 3 und 4. Die etwas niedrigere Bilanz bei der Ausgangslage 2 kann sich eventuell dadurch erklären, dass diese am weitesten vom Rest der Positionen entfernt liegt und daher mehr Zeit für ihr Erlernen nötig ist. In Gegensatz dazu scheitert überraschenderweise der neue Agent 2 genau an den Startpositionen 1 und 2, mit denen er trainiert wurde. Allerdings erreicht er gute Ergebnisse bei den restlichen Initialzuständen, vermutlich da er während seines Trainings bessere Aktionen für diese gelernt hat. Abschließend

erweist sich Agent 1 viel robuster als Agent 2. Bei den vereinfachten kleinen Räumen (Simple Action + Simple Observation Wrapper) ist der Agent robuster. Das kann man dadurch erklären, dass er während des Trainings genug Zeit hat, die Mehrheit der Aktionen zu besuchen und sie in Kombination mit den verschiedenen Zuständen zu erforschen.

6 Fazit

Ziel dieser Arbeit besteht darin, das Spiel Ökolopoly dem Reinforcement Learning zugänglich zu machen und den Einfluss von Action und Observation Wrapper in Zusammenhang mit unterschiedlichen Reward-Varianten zu untersuchen. Als Basis diente das im Vorfeld durchgeführte Praxisprojekt [Raycheva, 2021], das das Spiel als Gym-Umgebung mit einer entsprechenden grafischen Oberfläche bereitstellt. Die Arbeit beantwortet die zu Beginn gestellten Forschungsfragen.

F 1.1 Kann Reinforcement Learning auf Ökolopoly angewandt werden, gegeben den komplexen Aktions- und Zustandsraum?

Die Anwendung von Reinforcement Learning direkt auf das Environment von Ökolopoly führt zu keinen konkreten Ergebnissen. Um es zu ermöglichen, werden passende Wrapper für den riesigen Aktions- und Zustandsraum des Spiels implementiert. Infolge sind zwei Action und ein Observation Wrapper entstanden.

Der Box Action Wrapper überführt den diskreten Action Space des Spiels in einen kontinuierlichen Raum. Damit generalisiert der Agent besser über die Aktionen und trifft im Endeffekt geeignetere Entscheidungen. Im Unterschied dazu vermindert der Simple Action Wrapper drastisch den Aktionsraum. Die für eine Runde verfügbaren Punkte werden immer gedrittelt unter den gewünschten Lebensbereichen vergeben. Der Agent erforscht dann die meisten vorhandenen Aktionen während seines Trainings. Beide Action Wrapper halten dabei die Constraints bezüglich der Actions ein, die in Kapitel 3.1 erwähnt wurden. Der Simple Observation Wrapper vereinfacht stark den Zustandsraum, indem jeder Lebensbereich in 3 Sektoren „niedrig – mittel – hoch“ eingeteilt wird. Somit wird er zum schnelleren Erlernen unterschiedlicher Zustände beigetragen. Die in Kapitel 3.3 aufgeführten Arten von Belohnungen dienen als weitere Untersuchungspunkte. Eine davon ist der domänenspezifische Hilfsreward, der die Überlebensfähigkeit der Agenten im Spiel verbessert. Dieser wird im Vergleich zu der am Spielende erhaltenen Bilanzzahl, dem simplen Reward 1 und der Summe von dem Hilfsreward und der Bilanz gesetzt. Diese unterstützen den Agenten beim Finden der optimalen Strategie.

F 1.2 Welchen Einfluss haben die verschiedenen Repräsentationen der Räume auf den Erfolg (Lernprozess und Performance) des Agenten?

Für das Testen der Zusammenwirkung der entwickelten Komponenten werden Experimente mit den Algorithmen PPO, TD3 und SAC durchgeführt. Anhand der daraus erhaltenen Ergebnisse lässt sich feststellen, dass sich PPO als am zuverlässigsten für das Erzeugen erfolgreicher Agenten erweist und dass diese mit allen Action und Observation Wrapper gut zusammenarbeitet. TD3 ist auf die Art des Wrappers und der Reward-Variante stark angewiesen. Befriedigende Ergebnisse erreichen konkret die Konfigurationen: Box Action Wrapper mit dem Hilfsreward + Bilanzzahl, die stets zu ähnlich guten Erfolgen führen. SAC erzielt allgemein zufriedenstellende Ergebnisse nur mit dem Box Action Wrapper unabhängig vom eingesetzten Reward.

Über das gesamte Training kumulieren alle Algorithmen die erhaltenen Rewards auf ähn-

liche Weise. Es ist zu beobachten, dass die meisten lange vor dem Ende flachere Kurven aufweisen.

In Hinsicht auf Laufzeit erreicht PPO die kürzesten Trainingszeiten.

Begünstigte Konfigurationen für die Agenten erweisen sich der Simple Action in Kombination mit dem Simple Observation Wrapper oder der Box Action Wrapper allein. Bei beiden werden der simple Reward 1 und der PPO Algorithmus eingesetzt. Daraus werden jeweils Agent 1 und Agent 2 als die besten für weitere Untersuchung gewählt.

F 1.3 Inwieweit beeinflusst der Reward den Erfolg des Agenten?

In allen Szenarien kann beobachtet werden, dass die Belohnungsvarianten Hilfsreward und Hilfsreward + Bilanzzahl immens zu der Überlebensfähigkeit der Agenten während des Spiels beitragen. Dabei muss jedoch beachtet werden, dass eine höhere Rundenanzahl wegen der Definition der Formel 1 zu schlechterer Bilanz führt. Die Bilanzzahl als Rewards hingegen wirkt sich in den meisten Fällen nicht zufriedenstellend aus. Überraschenderweise führt der simple Reward 1 zum Erlangen der höchsten Bilanzzahlen.

F 2.1 Wie robust ist ein trainierter Agent, wenn die Startposition geringfügig geändert wird?

Die beiden Agenten zeigen bei einer alternativen Ausgangslage nicht so gute Robustheit. Sie werden mit einer zusätzlichen Startposition neu trainiert, was sich auf den Agenten 1 erfolgreich auswirkt. Die Robustheit scheint stark von der Größe der Räume während des Trainings abzuhängen.

F 2.2 Wie viele Trainingsrunden braucht der Agent, um zu lernen?

Dies hängt vom gewählten Wrapper und Reward ab. Bei allen Algorithmen sind meistens die angegebenen 800000 Timesteps genug, damit zufriedenstellende Performance erreicht wird.

F 2.3 Kann der Agent bessere Ergebnisse als ein Mensch erzielen?

Beim Vergleich der Strategien hat sich ergeben, dass einer der besten Agenten nicht nur eine sehr hohe Bilanz erzielt, sondern zweimal mehr Runden als der Mensch überlebt. Zudem hat er eine vorausschauendere Strategie entwickelt.

F 2.4 Kann der Mensch etwas von den Agenten lernen?

Die menschliche Strategie kann verbessert werden, indem die Aktionspunkte gleichmäßig verteilt werden und mehr Verständnis für die implementierten Rückwirkungen aufgebracht wird. Durch das Nutzen indirekter Einflüsse auf die Lebensbereiche können im Lauf des Spiels konsistent bessere Ergebnisse erwartet werden.

Abschließend wurde im Verlauf der Arbeit viel Erfahrung mit dem RL Baselines3 Zoo Framework gesammelt, das dem Erstellen der Agenten dient. Interessant dabei war die praxisnahe Auseinandersetzung mit state-of-the-art RL-Algorithmen (PPO, TD3 und SAC) und ihre Ergebnisse in Zusammenhang mit den realisierten Wrapper und Reward-Varianten

im Ökolopoly-Environment zu bewerten. Weiterhin werden die Kenntnisse mit PyQt für Erzeugen von GUIs erweitert, indem ein Analyseprogramm in Verknüpfung mit Stable Baselines3 realisiert worden ist. Dieses veranschaulicht die vom Agenten gelernte Strategie in Abhängigkeit von der eingestellten Startposition und erlaubt weitere Untersuchungen, wie die Analyse des gelernten Verhalten der besten Agenten und das Testen ihrer Robustheit.

7 Ausblick

Das Hauptziel dieser Arbeit, Reinforcement Learning auf Ökolopoly anzuwenden, wurde erreicht. Zur Vertiefung bieten sich folgende Forschungspunkte:

- Der genaue Grund für den Abbruch von PPO, wenn der Algorithmus auf die Ökolopoly-Umgebung ohne Wrapper eingesetzt wird, muss noch herausgefunden werden.
- Weitere Anfangsbedingungen können untersucht werden und für diese berichtet werden, welche Konfigurationen aus Wrapper und Reward sich als geeignet erweisen.
- Es können neue Reward-Varianten erforscht werden. In einer umfassenderen Untersuchung kann der Agent für jede Runde beispielsweise den Zähler aus der Formel der Bilanzzahl (s. Gl. 1) als Belohnung bekommen.
- Anschließend kann noch ein Observation Wrapper erstellt werden, der den Zustandsraum in einen kontinuierlichen überführt. Dessen Wirkung auf die Umgebung kann mit den Off-Policy Algorithmen untersucht werden.

Literatur

[Berner et al., 2019] Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., u a. (2019). Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*.

[Dulac-Arnold et al., 2015] Dulac-Arnold, G., Evans, R., van Hasselt, H., Sunehag, P., Lillicrap, T., Hunt, J., Mann, T., Weber, T., Degris, T., und Coppin, B. (2015). Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679*.

[Fujimoto et al., 2018] Fujimoto, S., Hoof, H., und Meger, D. (2018). Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, Seiten 1587–1596. PMLR.

[Haarnoja et al., 2018] Haarnoja, T., Zhou, A., Abbeel, P., und Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, Seiten 1861–1870. PMLR.

[Hill et al., 2018] Hill, A., Raffin, A., Ernestus, M., Gleave, A., Kanervisto, A., Traore, R., Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., und Wu, Y. (2018). Stable Baselines. `https://github.com/hill-a/stable-baselines`.

[Hollenstein et al., 2020] Hollenstein, J. J., Renaudo, E., Saveriano, M., und Piater, J. (2020). Improving the Exploration of Deep Reinforcement Learning in Continuous Domains using Planning for Policy Search. *arXiv preprint arXiv:2010.12974*.

[Holzbaur, 1991] Holzbaur, U. (1991). Ökolopoly und neuronale Netze - Das kybernetische Grundmodell, Ecolopoly and neural nets - The cybernetic basic model. In *Studium und Praxis, Die Welt, Ein vernetztes System, Workshop, 3*, Seiten 177–186, Ulm. Universitätsverlag Ulm;.

[Huber, 2011] Huber, J. (2011). Die grüne Welle.

[Kanervisto et al., 2020] Kanervisto, A., Scheller, C., und Hautamäki, V. (2020). Action space shaping in deep reinforcement learning. In *2020 IEEE Conference on Games (CoG)*, Seiten 479–486. IEEE.

[Lapan, 2020] Lapan, M. (2020). *Deep reinforcement learning hands-on*. Packt publishing.

[Lillicrap et al., 2015] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., und Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

[Merrick und Maher, 2009] Merrick, K. E. und Maher, M. L. (2009). *Motivated reinforcement learning: curious characters for multiuser games*. Springer Science & Business Media.

[Müggenburg, 2019] Müggenburg, J. (2019). *Kybernetik*, Seiten 280–282. J.B. Metzler, Stuttgart.

[Plaat, 2020] Plaat, A. (2020). *Learning to Play: Reinforcement Learning and Games*. Springer Nature.

[Raycheva, 2021] Raycheva, R. (2021). Erstellung eines Custom Environments in OpenAI Gym für das Spiel Ökolopoly. Technischer bericht, TH Köln – University of Applied Sciences. Praxisprojekt.

[Schulman et al., 2017] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., und Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

[Sutton und Barto, 2018] Sutton, R. S. und Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

[Sutton et al., 2000] Sutton, R. S., McAllester, D. A., Singh, S. P., und Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, Seiten 1057–1063.

[Vester, 1988] Vester, F. (1988). Der blaue Planet in der Krise. *Gewerkschaftliche Monatshefte*, 39(12):713–773.

[Vester und Korte, 1993] Vester, F. und Korte, R. (1993). *Ökolopoly: ein kybernetisches Umweltspiel*. Spieleverlag.

[Zahavy et al., 2018] Zahavy, T., Haroush, M., Merlis, N., Mankowitz, D. J., und Mannor, S. (2018). Learn what not to learn: Action elimination with deep reinforcement learning. *arXiv preprint arXiv:1809.02121*.

[Zhang et al., 2020] Zhang, J., Zhang, Z., Han, S., und Lü, S. (2020). Proximal Policy Optimization via Enhanced Exploration Efficiency. *arXiv preprint arXiv:2011.05525*.

Anhang

Repository: `https://github.com/cherrisimo/oekolopoly-rl`

Videos

- Ökolopoly-GUI: `https://th-koeln.sciebo.de/s/3z4uK3mKmYJqS1k`
- Enjoy.py: `https://th-koeln.sciebo.de/s/qJBhXCGYmPrwsVd`
- Analyseprogramm: `https://th-koeln.sciebo.de/s/7jThJo4mbEDYZQI`

  Im Video vom Analyseprogramm wird zunächst ein Agent (seine entsprechende *.zip*-Datei) ausgewählt und im Programm geladen. Mit dem Predict-Button kann die gelernte Strategie des Agenten Zug für Zug beobachtet werden. Die dadurch erzeugten Einträge in der Tabelle können dann mit Clear gelöscht werden. Bei Anklicken des Play-Buttons wird die gesamte Taktik des Agenten auf einmal gezeigt. Als nächstes kann die Startposition geändert und basierend darauf die vom Agenten getroffenen Aktionen beobachtet werden.

Laufzeit Algorithmen

Wrapper		PPO	TD3	SAC
Box Action	Bilanzzahl	9 ± 1	266 ± 7	886 ± 354
	Reward 1	8 ± 0	267 ± 4	851 ± 344
	Hilfsreward	8 ± 0	218 ± 14	836 ± 344
	Hilfsreward + Bilanzzahl	8 ± 0	212 ± 2	827 ± 339
Box Action + Simple Observation	Bilanzzahl	11 ± 0	220 ± 11	354 ± 4
	Reward 1	10 ± 0	313 ± 14	350 ± 2
	Hilfsreward	11 ± 0	340 ± 70	326 ± 3
	Hilfsreward + Bilanzzahl	11 ± 0	175 ± 5	302 ± 21
Simple Action	Bilanzzahl	13 ± 0	-	-
	Reward 1	99 ± 24	-	-
	Hilfsreward	15 ± 0	-	
	Hilfsreward + Bilanzzahl	14 ± 0	-	-
Simple Action + Simple Observation	Bilanzzahl	12 ± 0	-	-
	Reward 1	12 ± 0	-	-
	Hilfsreward	10 ± 0	-	-
	Hilfsreward + Bilanzzahl	8 ± 0	-	-

Die Zahlen in der Tabelle stellen die durchschnittlichen Trainigszeiten einer Gruppe von Agenten in Minuten dar. Bei PPO werden je Konfiguration 10 Agenten trainiert, wohingegen es bei TD3 und SAC jeweils 3 sind.

Hyperparameterliste

Die für diese Arbeit eingesetzten Hyperparameter sind von den unten angegebenen Umgebungen für die eigenen Experimente übernommen und stammen jeweils aus *ppo.yml, td3.yml und sac.yml*. Diese Dateien sind im Repository von RL Baselines3 Zoo enthalten.

- PPO: CartPole-v1
- TD3: BipedalWalker-v3
- SAC: BipedalWalker-v3

Hyperparameter gruppiert nach Algorithmus und Wert.

Hyperparameter	**Wert**		
	PPO	TD3	SAC
n_timesteps	!!float 8e5	!!float 8e5	!!float 5e5
policy	'MlpPolicy'	'MlpPolicy'	'MlpPolicy'
gamma	0.98	0.98	0.98
learning_rate	lin_0.001	!!float 1e-3	!!float 7.3e-4
batch_size	256		256
ent_coef	0.0		'auto'
learning_starts		10000	10000
buffer_size		200000	300000
gradient_steps		-1	64
train_freq		[1, "episode"]	64
policy_kwargs		"dict(net_arch=[400, 300])"	"dict(log_std_init=-3, net_arch=[400, 300])"
n_envs	8		
n_steps	32		
gae_lambda	0.8		
n_epochs	20		
clip_range	lin_0.2		
noise_type		'normal'	
noise_std		0.1	
tau			0.02
use_sde			True

Erklärung

Ich versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer oder der Verfasserin/des Verfassers selbst entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Gummersbach, 11.01.2022

Ort, Datum

Rechtsverbindliche Unterschrift