|  |
| --- |
| Zusammenfassung der Arbeit Abstract of Thesis |
| Fachbereich:  Department: :  **Elektrotechnik und Informatik**  Studiengang:  University course: :  **Informatik/Softwaretechnik**  Thema:  Subject:  **Spieltheoretische Betrachtung von Reinforcement-Learning-basierten Multiagentensystemen**  Zusammenfassung:  In dieser Arbeit wird anhand einer Computersimulation eines abstrakten spieltheoretischen Szenarios die Wirkung von Agentenbegegnungsschemata und Reinforcement Learning-Lernverfahren auf das Agentenverhalten im Iterierten Gefangenendilemma untersucht.  Abstract :  Verfasser:  Author:  **Leon Rauschecker**  Betreuender Professor/in:  Attending Professor:  **WS 2025/2026**  **Prof. Dr.-Ing. Stefan Krause**  WS / SS : |

# Inhaltsverzeichnis

[Zusammenfassung der Arbeit 1](#_Toc215239353)

[Inhaltsverzeichnis 2](#_Toc215239354)

[1 Einleitung 6](#_Toc215239355)

[2 Theoretische Grundlagen 7](#_Toc215239356)

[2.1 Spieltheorie 7](#_Toc215239357)

[2.1.1 Grundprinzipien 7](#_Toc215239358)

[2.1.2 Das Gefangenendilemma 8](#_Toc215239359)

[2.1.3 Das iterierte Gefangenendilemma 9](#_Toc215239360)

[2.1.4 Memory-One-Strategien 9](#_Toc215239361)

[2.2 Reinforcement Learning 10](#_Toc215239362)

[2.2.1 Grundprinzipien 11](#_Toc215239363)

[2.2.2 Die Herausforderung bei nicht-stationären Umgebungen 13](#_Toc215239364)

[2.2.3 Klassifizierung von RL-Verfahren: Wertebasiert, Policy-Gradient und Actor-Critic 13](#_Toc215239365)

[2.2.4 On-Policy und Off-Policy 15](#_Toc215239366)

[2.2.5 Model-Based und Model-Free 15](#_Toc215239367)

[2.2.6 Aktionsauswahlstrategien: Greedy, Epsilon-Greedy und Softmax 16](#_Toc215239368)

[2.3 Multiagentensysteme 17](#_Toc215239369)

[3 Methodik und Design 17](#_Toc215239370)

[3.1 Agenten 17](#_Toc215239371)

[3.1.1 Lernfähige Agenten 17](#_Toc215239372)

[3.1.2 Agenten mit reinen Strategien 18](#_Toc215239373)

[3.2 Die Begegnungsschemata 19](#_Toc215239374)

[3.2.1 Zufällige Paarung 19](#_Toc215239375)

[3.2.2 Räumliches Gitter 19](#_Toc215239376)

[3.3 Technologisches Setup und Werkzeuge 20](#_Toc215239377)

[3.3.1 Programmiersprache: Python 21](#_Toc215239378)

[3.3.2 PettingZoo als MAS-Umgebungs-Framework 21](#_Toc215239379)

[3.4 Gesamtstruktur der Implementierung der Computersimulation 21](#_Toc215239380)

[3.5 Die Simulationsumgebung 23](#_Toc215239381)

[3.6 Hauptskript und Simulationsschleife 23](#_Toc215239382)

[3.7 Diagramme und Analysetools 26](#_Toc215239383)

[3.8 Setup 0 - Baseline-Setup 29](#_Toc215239384)

[3.9 Setup 1 – Begegnungsschema 30](#_Toc215239385)

[3.10 Setup 2 – RL-Verfahren 30](#_Toc215239386)

[3.11 Setup 4 – Nachbarschaftstypen 30](#_Toc215239387)

[3.12 Setup 3 – Policy 30](#_Toc215239388)

[4 Ergebnisse 31](#_Toc215239389)

[4.1 Setup 0 - Ergebnis des Baseline-Setup 31](#_Toc215239390)

[4.1.1 Diagramme 31](#_Toc215239391)

[4.1.2 Rohdaten 33](#_Toc215239392)

[4.1.3 Analyse-Dashboard 34](#_Toc215239393)

[4.2 Setup 1 - Einfluss des Begegnungsschemas 36](#_Toc215239394)

[4.2.1 Diagramme 36](#_Toc215239395)

[4.2.2 Rohdaten 37](#_Toc215239396)

[4.2.3 Analyse-Dashboard 39](#_Toc215239397)

[4.3 Setup 2 - Vergleichende Betrachtung der Lernalgorithmen und deren Auswirkungen 41](#_Toc215239398)

[4.3.1 Diagramme 41](#_Toc215239399)

[4.3.2 Rohdaten 42](#_Toc215239400)

[4.3.3 Analyse-Dashboard 43](#_Toc215239401)

[4.4 Setup 3.1 - Einfluss der Nachbarschaftsgröße (Von-Neumann-Nachbarschaft) 45](#_Toc215239402)

[4.4.1 Diagramme 45](#_Toc215239403)

[4.4.2 Rohdaten 46](#_Toc215239404)

[4.4.3 Analyse-Dashboard 47](#_Toc215239405)

[4.5 Setup 3.2 – Einfluss der Nachbarschaftsgrößte (Moore-Nachbarschaft) 48](#_Toc215239406)

[4.5.1 Diagramme 48](#_Toc215239407)

[4.5.2 Rohdaten 49](#_Toc215239408)

[4.5.3 Analyse-Dashboard 50](#_Toc215239409)

[4.6 Setup 4 - Einfluss der Policy des Agenten (Vergleich Softmax und Epsilon-Greedy) 50](#_Toc215239410)

[4.6.1 Diagramme 50](#_Toc215239411)

[4.6.2 Rohdaten 50](#_Toc215239412)

[4.6.3 Analyse-Dashboard 50](#_Toc215239413)

[5 Interpretation der Ergebnisse 50](#_Toc215239414)

[5.1 Setup 1 50](#_Toc215239415)

[5.2 Setup 2 50](#_Toc215239416)

[5.3 Setup 3 50](#_Toc215239417)

[5.4 Setup 4 51](#_Toc215239418)

[6 Zusammenfassung und Ausblick 51](#_Toc215239419)

[6.1 Erweiterung des spieltheoretischen Modells 51](#_Toc215239420)

[6.2 Reinforcement Learning spezifische Erweiterungen 51](#_Toc215239421)

[Anhang 52](#_Toc215239422)

[Abbildungsverzeichnis 53](#_Toc215239423)

[Tabellenverzeichnis 55](#_Toc215239424)

[Literaturverzeichnis 56](#_Toc215239425)

# Einleitung

Die Komplexität moderner technischer Infrastrukturen erfordert zunehmend den Übergang von zentral gesteuerten Systemen hin zu verteilten, autonomen Netzwerken, wie etwa bei der Koordination autonomer Fahrzeugflotten, der Lastverteilung in dezentralen Energienetzen (Smart Grids) oder der dynamischen Ressourcenallokation im Internet of Things (IoT), weshalb ein tiefes Verständnis von Multiagentensystem (MAS) relevant ist. In diesen Domänen agieren die Agenten nicht isoliert voneinander, sondern müssen in einer geteilten Umgebung miteinander interagieren, um individuelle oder globale Ziele zu erreichen [1].

Die Herausforderung bei der Entwicklung solcher Systeme liegt nicht allein in der algorithmischen Optimierung eines einzelnen Agenten, sondern in der Bewältigung der strategischen Interdependenz. Da die Agenten, seien es Handelsalgorithmen oder Lieferroboter, oft darauf programmiert sind, ihren eigenen Nutzen zu maximieren, entstehen unvermeidlich Interessenkonflikte um begrenzte Ressourcen. Aus der Perspektive des Multi-Agent Reinforcement Learning (MARL) führt dies zu einer grundlegenden Herausforderung: Die Nicht-Stationarität der Umgebung. Alle Akteure lernen gleichzeitig und passen ihre Strategien stetig an. Was für einen Agenten im Moment als optimale Strategie erscheint, kann durch die Adaption der Nachbarn im nächsten Moment obsolet oder sogar schädlich sein.

Um robuste und effiziente MAS zu entwerfen, ist es daher essenziell zu verstehen, wie Reinforcement Learning (RL), als Methode der adaptiven Strategiefindung, und Spieltheorie, als Werkzeug zur Analyse strategischer Interaktionen, zusammenspielen. Die Frage, wie in einer Population aus lernfähigen, unabhängigen Agenten stabile Kooperation emergieren kann, ist somit nicht nur von theoretischem Interesse, sondern eine Voraussetzung für die Funktionsfähigkeit und Effizient moderner technischer Infrastrukturen.

Das Ziel dieser Arbeit ist es, ein tieferes Verständnis für die Dynamiken in komplexen technischen Multiagentensystemen zu entwickeln. Es soll untersucht werden, unter welchen Rahmenbedingungen sich kooperatives Verhalten in Systemen stabilisieren kann, in denen Agenten mittels Reinforcement Learning adaptiv ihre Strategien anpassen. Dabei dient das Iterierte Gefangenendilemma (IGD) als abstraktes Modell für Konfliktsituationen.

Die Arbeit ist dabei wie folgt aufgebaut:

Zunächst wird eine Baseline geschaffen, in der einfache Q-Learning-Agenten in einer strukturierten, homogenen Umgebung interagieren.

Ausgehend von dieser Baseline werden isolierte Variablen variiert, um deren Einfluss auf die Entstehung von Kooperation zu quantifizieren. Zunächst liegt der Fokus auf dem Vergleich zwischen anonymen Interaktionen (Zufallspaarung) und fixen, lokalen Nachbarschaften (Räumliches Gitter), da die Topologie der Vernetzung in technischen Systemen (lokale Vernetzung vs. globale Vernetzung) eine grundlegende Designentscheidung darstellt [5].

Des Weiteren wird untersucht, wie unterschiedliche Lernalgorithmen und Explorationsstrategien die Stabilität und Kooperationsfreudigkeit der Agenten beeinflussen.

# Theoretische Grundlagen

In diesem Kapitel werden die grundlegenden theoretischen Konzepte, welche in dieser Arbeit angewandt werden, erläutert.

## Spieltheorie

### Grundprinzipien

Die Spieltheorie ist eine mathematische Theorie, welche Entscheidungssituationen mit mehreren, rationalen Spielern formal modelliert. Spieler sind in diesem Kontext die aktionswählenden Entitäten beziehungsweise die Akteure, zum Beispiel ein Wirtschaftsunternehmen, eine politische Partei oder ein autonomes Vehikel. Die Spieltheorie wird auch als die Theorie der strategischen Entscheidungsfindung bezeichnet, wobei die Interdependenz und Interaktivität zwischen den Spielern der Kern der Spieltheorie ist und somit einen eigenständigen Themenbereich der Entscheidungstheorie darstellt. Das Ziel ist dabei, optimale Strategien der Spieler und mögliche Gleichgewichtszustände zu finden, zum Beispiel um die Dynamiken von kooperativen Gemeinschaften zu analysieren, wie etwa deren Entstehung, Entwicklung und Stabilität. Anwendung findet die Spieltheorie unter anderem in den Wirtschaftswissenschaften, den Politikwissenschaften, der Evolutionsbiologie oder in der Informatik [1].  
Man unterscheidet dabei zwischen der kooperativen und der nicht-kooperativen Spieltheorie. Bei der kooperativen Spieltheorie ist es den Spielern möglich, Allianzen, verbindliche Verträge oder Regeln zu erstellen, um kooperatives Verhalten zu fördern [1].  
Im Gegensatz dazu, ist es bei der nicht-kooperativen Spieltheorie nicht möglich, übergeordnete Regeln zu definieren, an welche sich die Spieler halten müssen. Die Entwicklung der Strategien, das Entstehen von potenziellen Gleichgewichtszuständen und die Kooperationsfreudigkeit ist somit ausschließlich abhängig von den Zielen der jeweiligen Spieler. Dabei ist in der klassischen Spieltheorie das alleinige Ziel des Spielers die Maximierung des eigenen Nutzens. Der Nutzen ist hierbei der Gewinn, den ein Spieler erzielen kann, zum Beispiel eine gewisse Punktzahl, Geldeinnahmen oder ein Marktanteil. Dies bedeutet, dass die Spieler in der nicht-kooperativen Spieltheorie primär ihre individuellen Ziele verfolgen. Kooperation kann dennoch entstehen, wenn sie langfristig den eigenen Nutzen der beteiligten Spieler maximiert [1].

Es gibt bereits eine Vielfalt an spieltheoretischen Modellen, welche unterschiedliche Problemstellungen und Dilemmata formalisieren. Eines dieser Modelle aus der klassischen, nicht-kooperativen Spieltheorie, welches den Kern der Untersuchungen in dieser Arbeit darstellt, ist das iterierte Gefangenendilemma.

### Das Gefangenendilemma

Das klassische Gefangenendilemma beschreibt eine Zwei-Spieler-Entscheidungssituation, in welcher sich die beiden Gefangenen, welche in diesem Kontext die Spieler repräsentieren, entscheiden müssen, ob sie mit ihrem Partner, also dem jeweils anderen Spieler, kooperieren oder ob sie diesen verraten. Das Dilemma entsteht dabei durch die charakteristische Rangfolge der Auszahlungen (Payoffs) in der Auszahlungsmatrix (Payoff matrix). Dabei gelten folgende Verhältnisse zwischen Auszahlungen: T > R > P > S sowie 2 \* R > T + S. Dabei ist T = Temptation (Versuchung), R = Reward (Belohnung), P = Punishment (Bestrafung) und S = Sucker (Ausbeutung). Diese Auszahlungsstruktur führt dazu, dass die individuell rationale Entscheidung die Defektion ist und somit im Widerspruch zum kollektiv optimalen Ergebnis, der gegenseitigen Kooperation, steht [2]. Defektion beschreibt hierbei das Gegenstück zur Kooperation, also den Verrat und ist hergeleitet vom englischen Wort „Defection“.

Abbildung 1 zeigt die Auszahlungsmatrix mit den jeweiligen Standardwerten T = 5, R = 3, P = 1 und S = 0. Die Zeilen repräsentieren die beiden möglichen Aktionen von Spieler 1 und die Spalten die Aktionsmöglichkeiten von Spieler 2. In den Zellen der Matrix befinden sich jeweils zwei Werte, wobei der linke Wert jeweils die Auszahlung für Spieler 1 darstellt und der rechte Wert die Auszahlung für Spieler 2. Somit ist T die Auszahlung für einseitige Defektion, S die Auszahlung für einseitige Kooperation, R die Auszahlung für gegenseitige Kooperation und P die Auszahlung für gegenseitige Defektion [2].

Ein Bild, das Text, Schrift, Screenshot, Quittung enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung 1 Auszahlungsmatrix mit Standardwerten im Gefangenendilemma

Beim Gefangenendilemma ist die Defektion die dominante Strategie. Eine dominante Strategie ist immer die rationale Wahl für einen Spieler, unabhängig von der gewählten Strategie des Gegenspielers, da sie eine mindestens gleichwertige wie oder bessere Auszahlung als jede andere eigene Strategie liefert [2].

### Das iterierte Gefangenendilemma

Während das klassische Gefangenendilemma eine einmalige Interaktion beschreibt, erweitert das iterierte Gefangenendilemma (IGD) dieses Modell auf eine mehrfache Wiederholung des Spiels zwischen denselben beiden Spielern. Diese Iteration verändert die Dynamiken der strategischen Entwicklung fundamental, denn anders als im einmaligen Spiel, bei dem Defektion die dominante Strategie darstellt, spielt nun der sogenannte "Schatten der Zukunft" eine entscheidende Rolle [2]. Die Spieler müssen berücksichtigen, dass ihre aktuelle Aktion das zukünftige Verhalten ihres Gegenspielers beeinflussen kann. Dies eröffnet die Möglichkeit für reziproke Strategien, bei denen Kooperation belohnt und Defektion bestraft wird. Die Aussicht auf zukünftige Interaktionen kann somit einen Anreiz schaffen, von der kurzfristig optimalen, aber langfristig suboptimalen Strategie der reinen Defektion abzuweichen und kooperatives Verhalten zu etablieren [2]. Bedingungen zu untersuchen, unter denen Kooperation im IGD entstehen und stabil bleiben kann, ist unter anderem Thema dieser Arbeit.

Man unterscheidet dabei zwischen reinen (pure) und gemischten (mixed) Strategien. Während gemischte Strategien Aktionen auf Basis von Wahrscheinlichkeitsverteilungen wählen, handelt es sich bei reinen Strategien um deterministische Regeln. Sie legen für jede mögliche Spielhistorie eindeutig fest, welche Aktion der Spieler als Nächstes ausführt [2].

Das bekannteste und relevanteste Beispiel für eine solche reine Strategie ist Tit-for-Tat. Diese Strategie folgt einem simplen Prinzip: Sie kooperiert in der ersten Runde und kopiert in allen darauffolgenden Runden die Aktion, welche der Gegenspieler in der Runde zuvor gespielt hat. In zwei Computer-Turnieren von Robert Axelrod im Jahr 1984, in welchem eine Vielzahl an Strategien gegeneinander angetreten sind, ist Tit-for-Tat als klarer Sieger hervorgegangen [2]. Somit bietet Tit-for-Tat eine gute Baseline zur Bewertung der Strategien der lernfähigen Agenten, welche in dieser Arbeit betrachtet werden.

### Memory-One-Strategien

Um im iterierten Gefangenendilemma Entscheidungen treffen zu können, benötigen die Agenten eine Strategie, die festlegt, welche Aktion in Abhängigkeit von der bisherigen Spielhistorie gewählt werden soll. Eine wichtige und häufig untersuchte Klasse solcher Strategien sind die Memory-One-Strategien. Wie der Name impliziert, basiert die Aktionswahl eines Agenten, der einer Memory-One-Strategie folgt, ausschließlich auf dem Ergebnis der vorangegangenen Runde. Da es im Gefangenendilemma vier mögliche Ergebnisse einer Runde gibt, lässt sich eine Memory-One-Strategie formal als ein probabilistischer Vektor darstellen. Dieser Vektor *p* = (P(C|CC), P(C|CD), P(C|DC), P(C|DD)) spezifiziert die Wahrscheinlichkeiten, in der nächsten Runde zu kooperieren, gegeben das Ergebnis der letzten Runde. Reine Strategien wie Tit-for-Tat können durch einen solchen Vektor ebenfalls repräsentiert werden, wobei die Wahrscheinlichkeiten in diesem Fall 0 für reine Defektion und 1 für reine Kooperation sind.

## Reinforcement Learning

Reinforcement Learning (RL) ist neben dem überwachten Lernen (Supervised Learning), dem unüberwachten Lernen (Unsupervised Learning) und dem semi-überwachtem Lernen ein weiters grundlegendes Teilgebiet des Maschinellen Lernens [1].

Im Unterschied zu den anderen Teilgebieten, bei denen aus vorhandenen Datensätzen gelernt wird, entweder mit vorgegebenen Labels wie beim überwachten Lernen oder zur Mustererkennung (Clustering) wie beim unüberwachten Lernen, basiert Reinforcement Learning auf dem Prinzip des Lernens durch Interaktion [1].

Ein Agent interagiert dabei mit einer Umgebung, indem er Aktionen ausführt, die den Zustand der Umgebung verändern. Für jede Aktion erhält der Agent eine Belohnung (Reward) von der Umgebung. Der Agent entwickelt dabei durch Ausprobieren (Trial and Error) eine Strategie (Policy), die den erwarteten kumulativen zukünftigen Gewinn maximiert. Das bedeutet, der Agent muss durch die Interaktion mit der Umgebung die Daten, aus denen er lernen kann, selbst generieren [1].

*Hinweis zur Notation:* Der hier verwendete Begriff „Reward“ beziehungsweise das Symbol r, bezeichnet das allgemeine Feedback-Signal im Reinforcement Learning. Er ist zu unterscheiden vom Parameter R (Reward) aus der Spieltheorie (siehe Abschnitt 2.1.2), welcher die Auszahlung für gegenseitige Kooperation im Gefangenendilemma definiert. Im Folgenden wird der Begriff Reward als das Feedback-Signal verwendet, sofern es nicht explizit anders beschrieben wird.

Dadurch entsteht das Dilemma zwischen Exploration und Exploitation. Exploration bezeichnet das Ausprobieren von Aktionen, die gemäß dem aktuellen Wissensstand des Agenten nicht zwingend die Besten sind. Ziel ist es dabei, neue Informationen über die Umgebung zu sammeln und potenziell bessere Strategien beziehungsweise Zustands-Aktions-Werte zu entdecken. Dies kann in Form von gelegentlichem zufälligem Verhalten auftreten [1].

Exploitation hingegen bedeutet, die Aktion zu wählen, von der der Agent aktuell glaubt, dass sie den höchsten erwarteten zukünftigen Gewinn liefert.  
Dabei ist es wichtig, die Balance zwischen Exploration und Exploitation effizient abzustimmen, um einerseits die Umgebung ausreichend zu erkunden und andererseits das bereits gelernte optimal zu nutzen [1].

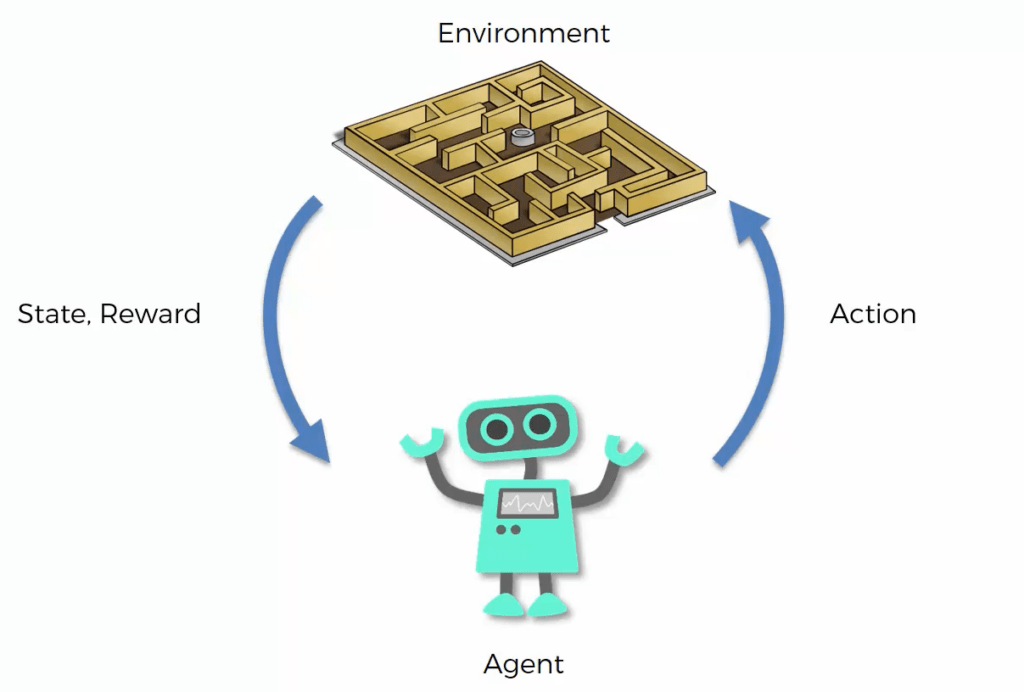


Abbildung 2 (Platzhalter) Visualiserung der Reinforcement Learning Pipeline

### Grundprinzipien

Reinforcement Learning basiert auf verschiedenen mathematischen Konzepten. Das wichtigste und fundamentalste Konzept ist der Markov Decision Process (MDP). Das MDP ist ein 5-Tupel (S, A, P, R, γ) und bietet eine formale Beschreibung von Reinforcement Learning Problemen und ist wie folgt definiert [1]:

S ist die Menge aller möglichen Zustände s ∈ S, auch Zustandsraum genannt. Dieser Zustandsraum kann je nach Problemstellung sowohl kontinuierlich als auch diskret sein.

A ist die Menge der gültigen Aktionen und wird auch als Aktionsraum bezeichnet. Auch der Aktionsraum kann sowohl kontinuierlich als auch diskret sein.

P ist die Übergangsfunktion und liefert die Übergangswahrscheinlichkeit P(s‘| s, a) = Pr(st+1 = s‘| st = s, at = a), welche beschreibt, wie wahrscheinlich es ist, nach Ausführung der Aktion a im Zustand s zum Zeitpunkt t in den Folgezustand s‘ zum Zeitpunkt t + 1 überzugehen.

R ist die Belohnungsfunktion und definiert das Feedback der Umgebung an den Agenten nach Ausführung der Aktion a in Zustand s. R(s, a, s‘) ist die erwartete, unmittelbare Belohnung, die der Agent erhält, basierend auf der Belohnungsfunktion.

γ ist der Diskontierungsfaktor (0 ≤ γ ≤ 1) welcher angibt, wie wichtig zukünftige Belohnungen gewertet werden sollen im Vergleich zur aktuellen Belohnung.

Hinzu kommt die Policy π des Agenten, welche als π(s) definiert ist und das Agentenverhalten repräsentiert. Es beschreibt die Wahrscheinlichkeit, im Zustand s die Aktion a zu wählen.

Zudem nimmt MDP die Markov-Eigenschaft an, welches aussagt, dass die Wahrscheinlichkeit in den Folgezustand s‘ überzugehen und die erwartete Belohnung ausschließlich vom aktuellen Zustand s, sowie der gewählten Aktion a abhängen und nicht von der gesamten Historie der vorherigen Zustände und Aktionen [1].

Des Weiteren ist das Verständnis des Grundprinzips der Bellman-Optimalitätsgleichung, auf welche Teile der Agentenimplementierungen basieren, relevant, denn die Bellman- Optimalitätsgleichung ist das theoretische Fundament vieler wertebasierter RL-Algorithmen wie etwa für Q-Learning und SARSA [3].  
Die Gleichung definiert den optimalen Wert einer Aktion a im Zustand s rekursiv als die Summe der erwarteten unmittelbaren Belohnung Rt+1 und des diskontierten, maximalen erwarteten Q-Wertes des Folgezustandes st+1 [3]:

Da in modellfreien Verfahren wie es bei Q-Learning und SARSA der Fall ist, die Übergangswahrscheinlichkeiten und die Belohnungsfunktion unbekannt sind, wird diese Gleichung durch Temporal Difference (TD) Learning approximiert. Nach einer beobachteten Transition (s, a, s‘) berechnet dieser Algorithmus ein TD-Ziel (TD Target), was eine Schätzung der rechten Seite der Bellman-Gleichung ist, basierend auf der konkreten erhaltenen Belohnung R(s, a, s‘) und der aktuellen Schätzung der optimalen zukünftigen Werte. Für Q-Learning wird das TD Target wie folgt berechnet [1]:

Für SARSA hingegen wird das TD Target basierend auf dem Q-Wert der nächsten, tatsächlich von der Policy gewählten Aktion a‘ berechnet [3]:

Die Differenz zwischen diesem TD-Ziel und dem aktuellen geschätztem Wert Q(s, a) wird als TD-Fehler (TD Error) bezeichnet und ist wie folgt definiert [3]:

Dieser TD-Fehler quantifiziert den Korrekturbedarf basierend auf der neuen Erfahrung (s, a, r, s‘). Er wird dann genutzt, um den Wert Q(s, a) mithilfe der Lernrate α iterativ in Richtung des TD-Ziels anzupassen. Somit ergibt sich folgende TD-Update-Formel [3]:

### Die Herausforderung bei nicht-stationären Umgebungen

Bei vielen klassischen RL-Problemen wird von einer stationären Umgebung ausgegangen. Das bedeutet, dass die Übergangswahrscheinlichkeiten P(s‘| s, a), sowie die Belohnungsfunktion R(s, a) über die Zeit hinweg konstant bleiben [1].

Da die Agenten, welche in dieser Arbeit untersucht werden, lernfähig sind und deren Entscheidungen wechselwirkend auf ihr Verhalten wirkt, handelt es sich bei der Umgebung um eine nicht-stationäre Umgebung. Das heißt, dass es keine optimale Strategie für jeden möglichen Zustand gibt und sich die Übergangswahrscheinlichkeiten ändern können. Die Agenten müssen sich somit dynamisch an die jeweilige Situation anpassen, indem sie aus den Beobachtungen, welche sie bei der Interaktion mit der Umgebung, also den anderen Agenten, machen, lernen, um somit ihren Nutzen zu maximieren [1]. Am Beispiel des IGD bedeutet das, dass wenn ein Agent lernt, eine kooperative Strategie auszunutzen, seine Gegner, welche die Umgebung darstellen, ihre Strategie ebenfalls anpassen werden. Diese Dynamik der stetigen Ausnutzung und Anpassung von Strategien stellt eine besondere Herausforderung in der Untersuchung des Agentenverhaltens dar, da bereits minimale Änderungen an bestimmten Variablen, wie zum Beispiel Nachbarschaftsgröße, Hyperparameter, Startbedingungen und Lernverfahren bereits komplett unterschiedliche Ergebnisse liefern können [4].

### Klassifizierung von RL-Verfahren: Wertebasiert, Policy-Gradient und Actor-Critic

Für die Auswahl geeigneter RL-Verfahren für die Agenten in dieser Arbeit ist das Verständnis der grundlegenden Unterschiede zwischen den verschiedenen Lernansätzen entscheidend. Dieser Abschnitt erläutert zunächst die konzeptionellen Funktionsweisen der drei Hauptkategorien: wertebasierte Verfahren, Policy-Gradient-Verfahren und Actor-Critic-Verfahren, beginnend mit den in dieser Arbeit verwendeten wertebasierten Methoden.

Wertebasierte Verfahren (Value-Based) zielen primär darauf ab, eine Bewertungsfunktion (Value Function) zu erlernen, die den erwarteten zukünftigen Gewinn für bestimmte Zustände oder Zustands-Aktions-Paare schätzt. Die am häufigsten verwendete Funktion ist die Action-Value-Funktion Q(s, a), welche den erwarteten Return bei Ausführung der Aktion a im jeweiligen Zustand s quantifiziert. Die Policy des Agenten leitet sich direkt aus den gelernten Action-Value-Werten ab. In der Regel wählt der Agent die Aktion mit dem höchsten geschätzten Q-Wert im aktuellen Zustand. Zur Sicherstellung der Exploration, also dem gelegentlichen, absichtlichen Abweichen von der aktuell als optimal erachteten Strategie, um neues Wissen über die Umgebungen zu sammeln, werden in der Regel Epsilon-Greedy oder Softmax eingesetzt, welche in einem späteren Abschnitt genauer erläutert werden. Beispiele für wertebasierte Algorithmen, welche auch in dieser Arbeit angewandt werden, sind Q-Learning und SARSA. Diese eignen sich gut für Probleme mit kleinen, diskreten Zustands- und Aktionsräumen, wie sie im modellierten iterierten Gefangenendilemma, also vier mögliche Zustände und zwei mögliche Aktionen, vorliegen, da ein explizites Aktion-Zustand-Mapping in tabellarischer Form (Q-Tabelle) somit möglich ist [3].

Im Gegensatz dazu stehen die Policy-Gradient-Verfahren. Diese optimieren die Strategie direkt, ohne notwendigerweise eine explizite Bewertungsfunktion zu lernen. Die Policy wird dabei oft durch einen parametrisierten Funktionsapproximator (z.B. ein neuronales Netz) repräsentiert, dessen Parameter direkt angepasst werden, um den erwarteten Gesamtgewinn zu maximieren. Dies geschieht typischerweise durch Gradientenaufstieg auf der erwarteten Belohnung. Policy-Gradient-Methoden werden in der Regel in Szenarien mit kontinuierlichen oder sehr hochdimensionalen Aktionsräumen angewandt, in denen wertebasierte Ansätze an ihre Grenzen stoßen, da die Berechnung von Q-Werten für unendlich viele Aktionen nicht praktikabel ist. Grundlegende Beispiele sind REINFORCE und Proximal Policy Optimization [3].

Als dritte Kategorie kombinieren Actor-Critic-Verfahren die Stärken der beiden vorgenannten Ansätze. Sie bestehen aus zwei Komponenten: einem Actor, der die Policy repräsentiert und Aktionen auswählt, und einem Critic, der eine Bewertungsfunktion (oft eine State-Value-Funktion oder eine Action-Value-Funktion) lernt, um die vom Actor gewählten Aktionen zu bewerten. Der Critic liefert dem Actor ein Feedback-Signal (oft den "Advantage", der angibt, wie viel besser eine Aktion war als erwartet), welches der Actor nutzt, um seine Policy durch Gradientenaufstieg zu verbessern. Dieser Ansatz führt häufig zu stabilerem Lernen als reine Policy-Gradient-Methoden und ist die Grundlage vieler moderner RL-Algorithmen wie A2C (Advantage Actor-Critic) oder A3C (Asynchronous Advantage Actor-Critic) [3].

Sowohl Policy-Gradient-basierte Ansätze, also auch Actor-Critic-Verfahren, werden in dieser Arbeit jedoch nicht angewandt, da die gegebene Problemstellung die komplexe Funktionsapproximation nicht erfordert und außerdem ist die Interpretation von Funktionsapproximationen deutlich komplexer, als einfache tabellenbasierte Algorithmen wie Q-Learning geeigneter sind.

### On-Policy und Off-Policy

Eine weitere Kategorie von RL-Lernverfahren ist das Konzept von On-Policy- und Off-Policy. Diese beiden Ansätze unterscheiden sich in der Art und Weise, wie ein Agent Aktionen wählt und seine Policy optimiert. Off-Policy-Verfahren sind in zwei Policies aufgeteilt, die Verhaltenspolicy (Behavioral policy) und die Zielpolicy (Target policy). Die Verhaltenspolicy entscheidet, welche Aktion tatsächlich ausgeführt wird. Mit dieser Policy erkundet der Agent die Umgebung und sammelt Daten. Die Zielpolicy ist die Policy, welche der Agent zu optimieren versucht. Das bedeutet, dass sie nur im Update-Schritt verwendet wird, um den jeweiligen Zielwert zu berechnet [3].

On-Policy hingegen nutzt sowohl für die Aktionswahl als auch für das Update des Wissensstandes, dieselbe Policy [3].

In dieser Arbeit werden sowohl ein Off-Policy-, als auch ein On-Policy-basierter Agent betrachtet.

### Model-Based und Model-Free

Innerhalb des Reinforcement Learnings wird grundlegend zwischen modellbasierten und modellfreien Ansätzen unterschieden [1]. Modellbasierte Verfahren versuchen, ein explizites Modell der Umgebung zu erlernen oder zu nutzen, typischerweise die Übergangsfunktion und die Belohnungsfunktion. Mit diesem Modell kann der Agent dann planen oder zukünftige Konsequenzen seiner Aktionen simulieren, um eine optimale Strategie zu finden [1].

Die in dieser Arbeit verwendeten Algorithmen, Q-Learning und SARSA, gehören hingegen zur Kategorie der modellfreien Verfahren [1]. Modellfreie Agenten lernen kein explizites Modell der Umgebungsdynamik. Stattdessen lernen sie ihr Verhalten direkt aus den erfahrenen Interaktionen. Konkret erlernen Q-Learning und SARSA eine Action-Value-Funktion, welche den erwarteten kumulativen Return für das Ausführen der Aktion im Zustand schätzt [1]. Sie approximieren diese Q-Werte direkt durch TD-Learning aus den beobachteten Belohnungen und Zustandsübergängen, ohne die zugrundeliegenden Wahrscheinlichkeiten oder die Funktion explizit modellieren zu müssen. Die Policy wird dann implizit aus den gelernten Q-Werten abgeleitet, z. B. durch Epsilon-Greedy-Auswahl [1]. Dieser modellfreie Ansatz ist für die vorliegende Arbeit besonders geeignet, da die Umgebungsdynamik durch die Interaktion mit anderen lernenden Agenten komplex und nicht-stationär ist, was die explizite Modellierung erschweren würde.

### Aktionsauswahlstrategien: Greedy, Epsilon-Greedy und Softmax

Wertebasierte RL-Verfahren lernen eine Action-Value-Funktion Q(s, a). Um aus diesen Werten eine konkrete Aktion auszuwählen, also eine Policy zu definieren, werden verschiedene Strategien verwendet, die das Dilemma zwischen Exploration und Exploitation adressieren.

Die einfachste Strategie ist die Greedy-Policy. Sie wählt in jedem Zustand s deterministisch diejenige Aktion a aus, die den aktuell höchsten geschätzten Q-Wert Q(s, a) besitzt: . Dieser Ansatz ist rein exploitativ, das heißt, er nutzt das bereits vorhandene Wissen optimal aus, führt jedoch keine Exploration durch. Somit könnte der Agent in suboptimalen Strategien festsitzen, falls die aktuellen Q-Wert-Schätzungen zu ungenau sind [3].

Um nun Exploration zu ermöglichen, erweitert die Epsilon-Greedy-Strategie (ε-Greedy) den Greedy-Ansatz um eine Zufallskomponente. Mit einer kleinen Wahrscheinlichkeit ε (Epsilon) wählt der Agent eine zufällige Aktion aus allen verfügbaren Aktionen, unabhängig von den Q-Werten. Mit der Wahrscheinlichkeit 1 – ε folgt er der Greedy-Policy und wählt die Aktion mit dem höchsten Q-Wert. Eine übliche Erweiterung ist der Epsilon-Decay, bei dem der Wert von ε im Laufe des Lernprozesses schrittweise multiplikativ mit einem Faktor von < 1 reduziert wird. Dies sorgt dafür, dass der Agent anfangs stark exploriert und später, wenn die Q-Werte zuverlässiger werden, zunehmen exploitiert [3].

Ein etwas anderer Ansatz ist Softmax hingegen. Bei Softmax wird die sogenannte Softmax-Formel verwendet, um die Q-Werte in eine Wahrscheinlichkeitsverteilung umzuwandeln und somit die Größe der Unsicherheit zwischen Q-Werten zu berücksichtigen. Bei Epsilon-Greedy ist es nämlich so, dass Q-Werte nahezu gleich sein können, der größe Wert jedoch mit einer Wahrscheinlichkeit von 1 – ε gewählt wird [3].

Anstatt einer binären Umschaltung zwischen Greedy und Zufall, wandelt Softmax die Q-Werte für alle Aktionen in einem Zustand s mithilfe der folgenden Softmax-Formel in eine Wahrscheinlichkeitsverteilung um [3]:

Hierbei ist β ein positiver Parameter (β > 0), der die Steilheit der Verteilung und somit den Grad der Exploitation steuert. Aktionen mit höheren Q-Werten erhalten eine höhere Auswahlwahrscheinlichkeit. Ein niedriges Beta (β -> 0) führt zu einer fast uniformen, zufälligen Verteilung, was zu hoher Exploration führt, während ein hohes Beta (β -> ∞) das Verhalten der Greedy-Policy annähert, also hohe Exploitation. Ähnlich wie bei Epsilon-Greedy kann man auch das β über Zeit steuern, um in der Regel anfangs eine hohe Exploration zu fördern und später eine höhere Exploitation [3]. Es gibt auch noch den Parameter Temperature T. Beta ist der Kehrwert von Temperature, also . Der Temperature-Parameter wird auch in der Implementierung der Softmax-Formel verwendet.

## Multiagentensysteme

Im Gegensatz zu Single-Agent-Umgebungen, in denen nur ein Agent mit der Umgebung interagiert, sind bei Multiagentensystemen mehrere aktive Agenten beziehungsweise Akteure beinhaltet. Ein MAS besteht somit aus einer Menge autonomer Agenten, die in einer gemeinsamen Umgebung agieren und interagieren. Die zentrale Herausforderung in MAS im Vergleich zu Single-Agent-Umgebungen ist die strategische Interdependenz, das heißt, das optimale Verhalten eines Agenten hängt maßgeblich von den Aktionen der anderen Agenten ab [1]. Somit ist die Spieltheorie ein sehr geeignetes Werkzeug, um Dynamiken und das emergente Verhalten in MAS zu untersuchen. Die Agenten in einem MAS können entweder ihre eigenen Ziele verfolgen oder kooperativ an demselben Ziel arbeiten. In dieser Arbeit wird das MAS als das iterierte Gefangendilemma modelliert und ist somit nicht-kooperativ. Zur Vereinfachung wird zudem angenommen, dass die Agenten sich stets paarweise treffen.

# Methodik und Design

In diesem Kapitel werden grundlegende Designentscheidungen erläutert und begründet.

## Agenten

In diesem Abschnitt wird die Funktionsweise der Agenten erläutert.

Die Agenten sind die aktionswählenden Entitäten der Simulation. Das heißt, sie sind die Spieler des IGD. Die lernfähigen Agenten sind in der Lage, den unmittelbar vorangegangenen Zustand zu nutzen, um daraus ihre Strategie für die zukünftige Aktionswahl anzupassen und zu optimieren. Die Agenten, welche lediglich reine Strategien spielen können, wie Tit-for-Tat oder Always Defect, passen ihre Strategie somit per Definition nicht an.

### Lernfähige Agenten

Es werden in dieser Arbeit zwei lernfähige Agenten betrachtet, welche jeweils auf zwei unterschiedliche RL-Verfahren basieren: Q-Learning und SARSA. Die Hyperparameter wurden so gewählt, dass Kooperation theoretisch begünstigt wird. Sie werden zwischen den Agenten nicht unterschieden und auch nicht dynamisch optimiert, sondern zum Start der Experimente wie folgt festgelegt: Die Lernrate α = 0,05, der Diskontierungsfaktor γ = 0,95, die ε-Startwahrscheinlichkeit ε = 1,0, ε-Decay = 0,9995 sowie den minimalen ε-Wert = 0,001. Eine geringe Lernrate begünstigt stabileres Verhalten, was im instabilen IGD, wo jeder Verrat bereits in eine Defektionsspirale führen kann, von Vorteil ist. Ein hoher Diskontierungsfaktor γ wertet den Schatten der Zukunft, also zukünftige Belohnungen, höher und somit den sofortigen höheren Gewinn, also die Chance auf einseitige Defektion T = 5, geringer [5].

Eine anfänglich hohe-Explorationswahrscheinlichkeit soll den Agenten ermöglichen, zu Beginn der Simulation die Umgebung kennenzulernen, später jedoch zu stabilerem Verhalten führen, weshalb diese Wahrscheinlichkeit schrittweise verringert wird, hin zu einer sehr geringen ε-Wahrscheinlichkeit = 0,001, um ähnlich wie bei der Lernrate, nicht aufgrund von zufälliger Defektion in die sonst dominierende Defektionsspirale überzugehen. Die Wahl der Epsilon-Konfiguration erzeugt dabei ein Dilemma zwischen der Anpassungsfähigkeit der Agenten auf neue potenziell ausbeuterische Agenten und der Stabilität des gesamten Systems, denn chaotische, stark explorative Systeme begünstigen, wie soeben erwähnt, defektives Verhalten [5]. Für diese Arbeit wurde somit ein sehr geringes Minimum von Epsilon gewählt, sodass kooperative Stabilität begünstigt wird, da alle Agenten denselben Hyperparametersatz verwenden.

Auf Basis dieser Hyperparameter werden nun die beiden spezifischen Algorithmen betrachtet.

Bei Q-Learning, dem in dieser Arbeit verwendeten Off-Policy-Verfahren, erfolgt das Update der Q-Werte unabhängig von der Aktion, die in der Verhaltenspolicy Epsilon-Greedy im Folgezustand tatsächlich gewählt wird. Wie bereits erwähnt, optimiert Q-Learning die Zielpolicy greedy. Das TD-Ziel wird unter Verwendung des maximalen Q-Wertes des Folgezustands s‘ berechnet.

SARSA unterscheidet sich dabei durch den On-Policy-Ansatz. SARSA lernt den Wert der tatsächlich ausgeführten Verhaltenspolicy. Die Update-Formel muss daher nicht nur die aktuelle Aktion a, sondern auch die nächste Aktion a‘ kennen, die von der Policy im Folgezustand s‘ tatsächlich gewählt wird.

Die genauen TD-Ziel und TD-Fehler-Formeln der jeweiligen Algorithmen wurden bereits im vorherigen Abschnitt 2.2.1 „Grundprinzipien“ erläutert.

### Agenten mit reinen Strategien

Damit die Simulationsumgebung validiert werden kann, wurden reine Strategien implementiert, welche deutlich einfacher zu interpretieren sind, da klare und einfache Muster entstehen, basierend auf ihrer Strategie. Des Weiteren dienen sie als Testsubjekte, um die Robustheit, Stabilität und mögliche Beschleunigung oder Verhinderung der Entwicklung von kooperativen Gemeinschaften, zu untersuchen.  
Es wurden folgende reine Strategien implementiert:  
Tit-for-Tat, welche in der ersten Runde kooperiert und anschließend die vorangegangene Aktion des Gegenspielers kopiert. Always Defect, welche unabhängig der Strategie des Gegenspielers stets defektiert. Always Cooperate, welche immer kooperiert. Grim Trigger, welche so lange kooperiert, bis der Gegenspieler das erste Mal defektiert. Ab diesen Zeitpunkt defektiert Grim Trigger stets. Und zuletzt Random, welche Aktionen auf reinen Zufall wählt. Diese Strategien wurden gewählt, da sie jeweils relevante Eigenschaften haben.

## Die Begegnungsschemata

Das Begegnungsschemata definiert, welche Agenten miteinander das IGD spielen werden.

### Zufällige Paarung

Das Begegnungsschemata, basierend auf reinem Zufall, dient als statistisches Vergleichsexperiment, um untersuchen zu können, welche Dynamiken aufgrund des Begegnungsschemas entstehen und welche unabhängig davon sind. Es werden hierbei vor jedem Match paarweise Agenten zufällig aus dem gesamten Agentenpool gewählt, welche miteinander das IGD spielen. Ein Match beschreibt also den Moment, in welchem zwei Agenten ausgewählt wurden und nun das IGD über eine bestimmte Anzahl an Runden spielen. Nachdem das Match vorbei ist, wird ein neues Paar definiert, bis die maximale Anzahl an Matches erreicht ist und die Simulation ausgewertet wird.

### Räumliches Gitter

In diesem Abschnitt wird das in dieser Arbeit hauptsächlich untersuchte Begegnungsschemata erläutert. Im Gegensatz zu anonymen zufälligen Agentenpaarung, werden die Agenten im räumlichen Gitter (Spatial Grid) in einer festen Topologie angeordnet. Für diese Simulation wird ein zweidimensionales Gitter verwendet, wobei jede Zelle von genau einem Agenten besetzt ist. Die Position eines Agenten ändert sich während der gesamten Simulation nicht.

Das räumliche Gitter stellt somit ein besonders interessantes Schema dar, da es stabile soziale Netzwerke (Nachbarschaften) modelliert. Ein Agent interagiert ausschließlich mit den Agenten auf seinen benachbarten Feldern. Da ein Agent somit über die gesamte Simulationsdauer hinweg wiederholt auf dieselben, festen Interaktionspartner trifft, wird die Anonymität aufgehoben. Dies ist die entscheidende Voraussetzung für die Emergenz von spieltheoretischen Konzepten wie impliziter Reputation und Reziprozität. Dadurch fällt auch der „Schatten der Zukunft“ stärker ins Gewicht.

Man muss des Weiteren die Art der Nachbarschaft unterscheiden. In dieser Arbeit werden drei verschiedene Nachbarschaftstypen untersucht: Die Von-Neumann-Nachbarschaft, die Moore-Nachbarschaft und eine erweiterte, kreisförmige Moore-Nachbarschaft. Die Von-Neumann-Nachbarschaft zählt die Diagonalen nicht zur Nachbarschaft dazu, wodurch sich eine Gesamtanzahl von vier Nachbarn ergibt. Die Moore-Nachbarschaft erkennt die Diagonalen als Nachbarn an, wodurch sich eine Gesamtanzahl von acht Nachbarn ergibt. Bei der Erweiterung der Moore-Nachbarschaft werden weitere vier Agenten als Nachbar anerkannt. Diese befinden sich jeweils neben dem äußersten, mittigen Agenten, wodurch eine kreisförmige Nachbarschaft entsteht.

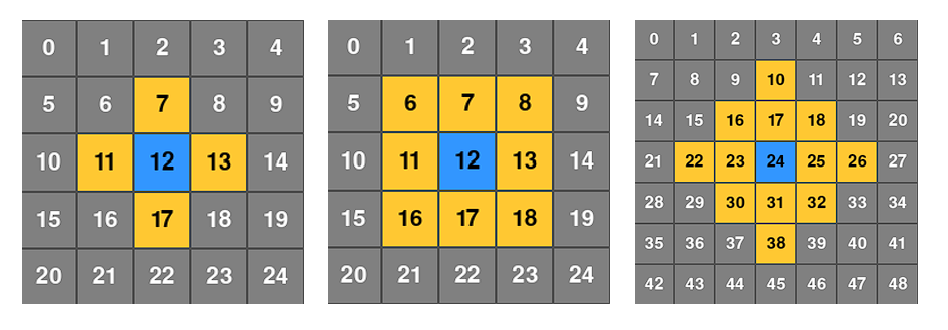


Abbildung 3 – Nachbarschftstypen von links: Von-Neumann-, Moore- und erweiterte Moore-Nachbarschft

In Abbildung 4 sind Visualisierungen der drei Nachbarschaftstypen zu sehen. Ganz links die Von-Neumann-Nachbarschaft, in der Mitte die Moore-Nachbarschaft und ganz rechts die erweiterte Moore-Nachbarschaft.

Der Ablauf der Agentenpaarung wird durch eine interne Generation in Form einer Warteschlange (match\_queue) an Agentenduellen gesteuert. Ein Agentenduell ist also das konkrete Agentenpaar. Bei jeder Anfrage der Hauptschleife nach einem Agentenpaar wird zunächst geprüft, ob diese Warteschlange leer ist.

Sollte die Warteschlange leer sein, wie es sowohl bei der erstmaligen Anfrage zu Beginn der Simulation als auch jedes Mal nach Abschluss einer Generation der Fall ist, wird sie neu befüllt.

Dabei werden alle Agenten iterativ ihren jeweiligen Agentenpaarungen zugewiesen basierend auf dem Nachbarschaftstyp. Das bedeutet, dass in dieser einen Warteschlange alle möglichen Agentenpaarungen für eine Generation gespeichert sind. Diese Liste an Zwei-Tupeln wird anschließend gemischt, bevor das erste Agentenpaar an die Hauptschleife übergeben wird. Die Ränder sind zudem abgeschlossen und nicht überfließend, somit haben Agenten, welche am Rand des Gitters positioniert sind, zwangsläufig eine kleinere Nachbarschaft.

Da der Erstellungsprozess der Warteschlange bidirektional ist, das heißt, invertierte Agentenpaarungen erlaubt sind, spielt jeder Agent pro Generation zweimal gegen alle seine Nachbarn, bis die Warteschlange leer ist.

## Technologisches Setup und Werkzeuge

In diesem Abschnitt wird die Wahl der Technologien, der Bibliotheken, der Frameworks und der Programmiersprache erläutert und begründet.

### Programmiersprache: Python

Als Programmiersprache für die Implementierung der Simulation und Analyse wurde Python gewählt. Diese Entscheidung basiert auf mehreren Vorteilen, die Python für die vorliegende Problemstellung bietet. Python hat sich als dominierende Sprache in den Bereichen des wissenschaftlichen Rechnens, der Datenanalyse und insbesondere des Maschinellen Lernens etabliert.

Ein entscheidender Faktor ist die breite Auswahl an verfügbaren Bibliotheken. Für diese Arbeit waren insbesondere NumPy für numerische Berechnungen und Array-Operationen, Matplotlib zur Erstellung von statischen Ergebnis-Visualisierungen, Pygame für die Entwicklung der interaktiven Visualisierung und PettingZoo als Framework für Multiagentensystemen von zentraler Bedeutung. Des Weiteren gibt es eine breite Auswahl an fortgeschrittenen Machine Learning Bibliotheken wie RLib (Ray), Tianshou und Stable-Basline3, welche in dieser Arbeit zwar nicht angewandt werden, jedoch war die Möglichkeit, Algorithmen aus diesen Bibliotheken nutzen zu können, ein weiterer Grund für die Wahl von Python.

### PettingZoo als MAS-Umgebungs-Framework

Für das Simulationsdesign spielte die Standardisierung und Erweiterbarkeit eine wichtige Rolle. Ziel war es, eine Umgebung (Environment) zu schaffen, die mit verschiedenen Reinforcement-Learning-Algorithmen und -Bibliotheken kompatibel ist, um zukünftige Erweiterungen und Vergleiche zu ermöglichen.

Während für Single-Agent-Umgebungen Gymnasium als De-facto-Standard gilt, ist es für Multiagentensysteme PettingZoo, welches auf Gymnasium aufbaut. PettingZoo ermöglicht es, eine standardisierte API für die Interaktion zwischen Agenten und einer Multi-Agenten-Umgebung zu definieren. Das bedeutet, dass es somit möglich ist, eigene Agentenimplementierungen und Lösungen aus Bibliotheken wie Stable-Baseline3, Tianshou oder RLlib (Ray) nach Belieben zu kombinieren und zu vergleichen.  
PettingZoo bietet bereits eine Auswahl an sowohl offiziellen Umgebungen als auch Umgebungen von Drittanbietern. Für diese Arbeit wurde jedoch eine eigene, benutzerdefinierte PettingZoo-Umgebung des Iterierten Gefangenendilemmas entwickelt, um maximale Kontrolle, Flexibilität und Interpretierbarkeit zu gewährleisten. In dieser Umgebung werden aktuell jedoch lediglich Memory-One-Strategien unterstützt.

## Gesamtstruktur der Implementierung der Computersimulation

Die Architektur der Simulation ist auf die Kernziele der Modularität (Separation of Concerns), Erweiterbarkeit, Lesbarkeit, Testbarkeit und Interpretierbarkeit ausgelegt. Um diese Ziele zu erreichen, wurden Abhängigkeiten zwischen den Modulen möglichst minimiert, Enumerationen anstelle von „Magic Numbers“ verwendet und Vererbung von abstrakten Klassen zur Erhöhung der Flexibilität eingesetzt.

Ein Bild, das Text, Diagramm, Plan, technische Zeichnung enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung 4 - Klassendiagramm der Simulation (vereinfacht)

Abbildung 3 zeigt eine vereinfachte Darstellung des Klassendiagramms, welches die Gesamtstruktur der Computersimulation visualisiert. Es besteht im Kern aus fünf separaten Modulen: Den Agenten, den Begegnungsschemata, der Evaluation, der Multiagentenumgebung (PettingZoo Environment) und dem Skript in der Main, welches alles miteinander verbindet. Die Begegnungsschemata sowie die Agenten sind nach dem Vererbungs-Entwurfsmuster erstellt, bei dem es eine abstrakte Überklasse, jeweils BaseAgent für die Agenten und MatchmakingScheme für die Begegnungschemata, gibt, von welcher die konkreten Agenten beziehungsweise Begegnungsschemata, welche dann zum Einsatz kommen, erben. Des Weiteren werden aus Gründen der Lesbarkeit die beiden Enumerationen Action und PureStrategy verwendet. Action beinhaltet die möglichen Agentenaktionen Kooperieren (COOPERATE = 0) und Verraten (DEFECT = 1). Die PureStrategy-Enumeration enthält die implementierten reinen Strategien. Außerdem gibt es die Gridfactory und Layout Maps. Die Gridfactory ist ein Skript, welches basierend auf Parametern wie Nachbarschaftsgröße oder Agententyp Layout Maps erstellt und diese dann iterativ einliest und dann eine Liste mit den jeweiligen Agenteninstanzen erstellt. Eine Layout Map ist eine Liste an Strategie-Kürzel in Form von Zeichenketten. Dadurch ist es möglich, das Gitter nach Belieben zu gestalten, um verschiedene Untersuchungen durchführen zu können.

Im Folgenden werden die genannten Kernmodule im Einzelnen erläutert.

## Die Simulationsumgebung

Die Simulationsumgebung verwendet wie bereits erwähnt das MAS-Framework PettingZoo. Dieses Framework ist die de-facto-Schnittstelle für Simulationen von Multiagentensystem und ermöglicht die Verknüpfung von verschiedenen Machine Learning Bibliotheken.  
Da bei dieser Arbeit der Anspruch an hohe Flexibilität und Interpretierbarkeit sehr hoch war, wurde auf bereits vorgegebene Drittanbieter-Umgebungen verzichtet und stattdessen eine benutzerdefinierte PettingZoo-Umgebung (Env) implementiert, dessen Funktionsweise im Folgenden nun erläutert wird.  
Die Mindestanforderung an zu implementierenden Methoden für eine funktionstüchtige Umgebung sind die beiden Methoden step() und reset(). Die reset()-Methode setzt den Zustand der Umgebung auf die Startbedingungen zurück und wird vor Beginn der Simulation aufgerufen. Die step()-Methode nimmt die Aktionen der Agenten entgegen, führt diese aus und gibt den neuen, beobachteten Zustand der Umgebung, die zu erhaltenen Belohnungen der Agenten, zwei Variablen, „termination“ und „truncation“, welche angeben, ob dies ein terminaler Zustand ist und optionale weitere Daten über die Umgebung, zurück.  
Die aktuelle Version der Umgebung unterstützt lediglich Memory-One-Strategien.

## Hauptskript und Simulationsschleife

In Hauptskript (Main.py) befindet sich die Simulationsschleife. Das bedeutet, dort werden die Aktionen der Agenten gewählt und der Umgebung übergeben, welche dann die Observation, Reward, etc. zurückgibt und auf dessen Basis die Agenten dann lernen können. Außerdem werden dort auch die für die Diagramme und Rohdaten benötigten Informationen wie Kooperationsrate, Gewinne, Gitterzustand, etc. gesammelt und nach Beendigung der Simulationsschleife geplottet beziehungsweise ausgegeben. Im Folgenden wird nun dieser Vorgang detailliert beschriebenen.  
Bevor die Simulationsschleife gestartet wird, werden alle relevanten Parameter festgelegt. Diese festzulegenden Parameter sind teilweise abhängig von den jeweiligen Experimenten und werden genauer in den Abschnitten 3.8 – 3.12 erläutert.

Anschließend beginnt die Simulation. Zunächst wird das Agentenpaar basierend auf dem Begegnungsschema für das aktuelle Match gewählt. Dieses Agentenpaar spielt dann die angegebene Anzahl an Runden das IGD. Währenddessen werden die Zustände in einem Dictionary, welches hier Experience Buffer genannt wird, gespeichert. Nachdem die Agenten fertig sind, lernen beide auf Basis ihres Experience Buffers, indem schrittweise immer ein Element, also ein aufgezeichneter Zustand, aus dem Experience Buffer eingelesen und entfernt wird und anschließend an die Lernmethode übergeben wird. Das bedeutet, die Anzahl der Lernwiederholungen und der IGD-Runden sind gleich. Wenn die maximale Matchanzahl erreicht ist, werden die gesammelten Daten geplottet beziehungsweise ausgegeben.

Ein Bild, das Diagramm, Entwurf, Origami enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung 5 - Flussdiagramm des Hauptskripts

## Diagramme und Analysetools

Die Auswertung der Simulationsergebnisse erfolgt in drei Teilen: Die Rohdaten, die Zeitreihen-Diagramme und das benutzerdefinierte interaktive Pygame-Analyse-Dashboard.

Die Rohdaten beinhalten ein finales Ranking aller Agenten im Pool. Die Sortierung dieses Rankings erfolgt absteigend nach dem durchschnittlichen Reward pro Match, also dem kumulierten Gesamtgewinn über die gesamte Simulation hinweg dividiert durch die Gesamtanzahl aller gespielten Matches. Somit ist diese Metrik unabhängig von der Position und ermöglicht einen fairen Erfolgsvergleich zwischen den Agenten. Diese Metrik wird im Folgenden als Agenten-Performance bezeichnet. Für jeden Agenten in diesem Ranking werden die folgenden Metriken aufgelistet:

1. Die gelernte Strategie zum Zeitpunkt des Simulationsendes.
2. Die durchschnittliche Kooperationsrate, also der prozentuale Anteil der Kooperationen (Aktion 1) an allen Aktionen über die gesamte Simulationsdauer.
3. Der durchschnittliche Reward pro Match, welcher, wie soeben erwähnt, das Sortierkriterium ist.

Zusätzlich zu diesem Ranking werden die aggregierten Metriken für das Gesamtsystem angezeigt. Diese sind die globale durchschnittliche Kooperationsrate, sowohl über den gesamten Zeitraum als auch als finaler Snapshot, sowie die System-Effizienz, also wie viel Reward alle Agenten über die gesamte Simulationsdauer zusammen gesammelt haben im Verhältnis zum theoretischen Maximum, also wenn alle Agenten immer kooperieren würden. Das theoretische Maximum wird wie folgt berechnet:

R ist dabei die Auszahlung R für gegenseitige Kooperation.

Außerdem werden die Strategievorkommen quantifiziert. Diese Metrik beschreibt die relative Häufigkeit der verschiedenen Strategietypen innerhalb der Gesamtpopulation und gibt somit an, welcher prozentuale Anteil der Gitterfläche von welcher Strategie besetzt wird. Diese Metriken sollen dabei unterstützen, die MAS-Dynamiken im räumlichen Gitter besser zu erkennen und zu quantifizieren.

Neben den Rohdaten werden auch statische Diagramme zur Datenvisualisierung der globalen Trends eingesetzt. Konkret handelt es sich um aggregierte Zeitreihen-Diagramme, welche die Entwicklung über die gesamte Simulationsdauer (alle Matches) abbilden. Dabei werden drei Metriken nach Agententyp betrachtet: die Entwicklung des Strategievektors, die durchschnittliche Kooperationsrate aller Agenten eines Agententyps sowie der durchschnittliche erzielte Reward aller Agenten eines Agententyps.

Eine weitere Datenvisualisierung ist die interaktive Gitter-Visualisierung, welches im Folgenden auch als Analyse-Dashboards bezeichnet wird. Die Interaktivität besteht dabei aus der Möglichkeit, mit den Pfeiltasten links und rechts durch die Match-Historie zu klicken, um den Gesamtzustand des Gitters in jedem Match analysieren zu können. Dabei wird der Agentenpool als ein räumliches 2D-Gitter dargestellt, wobei jede Zelle einen Agenten repräsentiert. Jede Zelle wird basierend auf einer Strategie-Klassifizierung einer bestimmten Farbe zugeordnet. Die Wahl der Strategieklassen wurde zum einen auf Basis von Voruntersuchungen getätigt, in welcher beobachtet wurde, welche Strategien besonders häufig oder konsistent emergieren und zum anderen auf Basis relevanter Strategien aus der spieltheoretischen Literatur wie etwa Tit-For-Tat oder Win-Stay-Lose-Shift. Das Ziel war es grundsätzlich, eine möglichst effektive Erfassung der emergierenden Strategien zu ermöglichen, ohne gleichzeitig zu viele verschiedene Farben für unterschiedliche Strategien definieren zu müssen, da dies schnell unübersichtlich werden könnte und den Fokus auf die relevanten Strategien verzerren könnte. Die Gitter-Visualisierung funktioniert unabhängig vom gewählten Begegnungsschemata, da es zur Erstellung lediglich den Agentenpool iterativ einliest und als ein 2D-Gitter darstellt. Die Strategie-Klassifizierung erfolgt auf Basis des Strategievektors *p* = (P(C|CC), P(C|CD), P(C|DC), P(C|DD)). Für lernfähige Agenten ist es wie folgt definiert:

█ – Rot für Defektoren. Wenn mindestens zwei dieser Werte kleiner oder gleich 0,2 sind, also die Wahrscheinlichkeit, dass nächste Runde defektiert wird, mindestens 80% ist, und gleichzeitig die anderen beiden Werte nicht größer als 0,8 sind, dann wird die Zelle rot angezeigt.

█ – Grün für Kooperatoren. Das Gegenteil der roten Defektoren. Wenn mindestens zwei Werte größer als 0,8 sind und die anderen beiden nicht kleiner als 0,2 sind, wird die Zelle als grün dargestellt.

█ – Orange für Polarisierte Strategien. Wenn mindestens zwei unter 0,2 und die anderen zwei über 0,8 sind, wird die Zelle der Farbe Orange zugeordnet.

Die Farben Lila, Blau und Cyan beschreiben konkrete Strategien.

█ – Lila für Win-Stay-Lose-Shift (WSLS). Bei WSLS bedeutet gewinnen, wenn der Agent die Auszahlungen R oder T bekommt und verlieren, wenn er S oder P bekommt. Wenn er also gewinnt, behält er seine Aktion, welche er zuletzt gespielt hat, bei, ansonsten wechselt er zu der anderen Aktion.

█ – Blau für Tit-for-Tat.

█ – Cyan für eine Vorstufe von Tit-for-Tat. Cyan ist für Agenten, welche nur dann mit einer Wahrscheinlichkeit von 0,8 oder höher kooperieren, wenn der Gegenspieler sowie der Agent selbst letzte Runde beide kooperiert haben. In allen anderen Zuständen kooperiert er nur mit einer Wahrscheinlichkeit von 0,2 oder kleiner.

█ - Gelb für Agenten, welche in keines der vordefinierten Farbkategorieren fällt.

█ - Grau für unbekannte Agententypen.

Für Agenten mit einer reinen Strategie, ist die Farbverteilung wie folgt festgelegt: Blau stets für Tit-for-Tat, Rot für Always Defect, Grün für Always Cooperate und Lila für andere Strategien wie Grim Trigger oder Random.

Zum Gesamtbild des Gitters gehören auch zwei Heatmaps und die scrollbare Agentenliste mit dem jeweiligen Strategievektor eines Agenten. Die erste Heatmap zeigt die Kooperationsrate eines Agenten im Gitter zum Zeitpunkt des jeweiligen Matches. Das Farbspektrum geht von rot für eine geringe Kooperationsrate und grün für eine hohe Kooperationsrate. Die zweite Heatmap zeigt den Mittelwert des aktuellen Gesamtgewinns eines Agenten im Gitter zum Zeitpunkt des aktuellen Matches. Die Obergrenze definiert der aktuell effizienteste Agent und die Untergrenze dementsprechend der aktuell ineffizienteste Agent. Die Heatmaps sind aufgebaut wie das räumliche Gitter wie es in Abbildung 6 zu sehen ist. Das bedeutet, es gibt 200 Zellen innerhalb einer Heatmap, wobei die Zelle ganz oben links die Kooperationsrate von Agent 0 darstellt und die Zelle ganz unten rechts die Kooperationsrate von Agent 200. Die Agentenliste zeigt alle Agenten und ihre jeweilige Strategie zum Zeitpunkt des aktuellen Matches.

Ein Bild, das Screenshot, Farbigkeit, Text, Multimedia-Software enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung 6 - Snapshot des Analyse-Dashboards zum Matchcount 200.000 von Seed 0.

Um ein Gesamtbild über mehrere Seeds beziehungsweise Simulationsdurchläufen zu bekommen, werden die gesammelten Daten der einzelnen Durchläufe nach Durchführung von ausreichend verschiedenen Seeds zusammengefasst.

## Setup 0 - Baseline-Setup

Das folgende Setup beschreibt die Ausgangssituation für die Simulation. Von dieser Baseline-Konfiguration ausgehend wird in den folgenden Untersuchungen eine ausgewählte unabhängige Variable, wie etwa die Nachbarschaftsgrößte, der Agententyp oder das Begegnungsschemata, verändert und betrachtet, um die Auswirkungen auf das Agentenverhalten zu untersuchen.

Die Agentenpopulation ist homogen und besteht ausschließlich aus 200 Q-Learning-Agenten. Die Anzahl an Agenten wurde so gewählt, dass eine Balance zwischen statistischer Aussagekraft und Recheneffizienz beibehalten wird. Denn zu wenige Agenten würden kein akkurates Gesamtbild der Experimente darstellen und die Ergebnisse anfällig für stochastische Ausreißer machen, während zu viele den Rechenaufwand stark erhöhen würden, da mit einer steigenden Anzahl an Agenten auch die maximale Matchanzahl steigen muss, da sonst womöglich die Zeit für die Entwicklung und Emergenz bestimmter Strategien zu kurz wäre.

Es werden keine vordefinierten Strategien verwendet. Die Agenten beginnen mit einer zufällig initialisierten Q-Tabellen mit einem Wertebereich zwischen 1 und 0. Als Verhaltenspolicy wird Epsilon-Greedy eingesetzt. Die Agenten nutzen die in dem Abschnitt 3.4.1 „Lernfähige Agenten“ vorgestellten Hyperparameter. Die Lernart ist Offline-Lernen beziehungsweise Batch-Lernen, das heißt, es wird zunächst eine Liste an Beobachtungen gesammelt über ein Match hinweg, also 200 Beobachtungen, und auf Basis dieser Liste wird dann für jede Beobachtung einzeln gelernt.

Des Weiteren wird als Begegnungsschema das räumliche Gitter verwendet. Als Nachbarschaftstyp wird die erweiterte Moore-Nachbarschaft verwendet. Basierend auf der Agentenzahl von 200 ergibt sich die Gitterform von 20x10.

Die Simulationsdauer ist auf 200.000 Matches festgelegt, wobei jedes Match über 200 Runden läuft.

Die IGD-Rundenanzahl von 200 wurde in Anlehnung an das erste Computer-Turnier von Axelrod gewählt [2].

Die Simulationsdauer ist so gewählt, dass die Entwicklung des Systems ausreichend dargestellt wird. Die Epsilonwahrscheinlichkeit, welche für zufälliges, explorierendes Verhalten sorgt, wird mit jedem Lernschritt um den Faktor EpsilonDecay = 0,9995 verringert. Bei 200 Runden pro Match ergeben sich somit 200 Lernschritte pro Match. Da der minimale Epsilonwert auf Epsilonfinal = 0,001 und das Startepsilon auf EpsilonStart = 1,0 festgelegt sind, ergibt sich über folgende Formel die Rundenzahl N = 13,812, nach welchem Epsilon das Minimum erreicht hat:

Das bedeutet, dass die Lernphase nur sehr kurz ist, nämlich Matches pro Agenten. Da die Umgebung mit zwei Aktionen und vier Zuständen jedoch relativ klein ist und bedenkt, dass 200 Agenten im System existieren, sollte die Länge der Lernphase ausreichend sein, denn bis alle Agenten ihr jeweiliges Epsilon-Minimum erreicht haben dauert es .  
Des Weiteren haben Vorstudien, welche die Entwicklung des Systems betrachtet haben, ergeben, dass ungefähr 200.000 Matches bei den gegebenen Hyperparametern ausreichen, sowie nötig sind, um einen guten Überblick über die Entwicklung des Systems zu bekommen. Diese Testdurchläufe der Vorstudien wurden auf Basis des Baseline-Setups vorgenommen.

## Setup 1 – Begegnungsschema

Die erste Untersuchung beschäftigt sich mir der Frage, welchen Einfluss das Schema, nach welchem die Agentepaare zustande kommen, auf das Agentenverhalten und somit auf die Dynamiken des Multiagentensystems hat. Als Vergleichsschema wird die zufällige Paarung verwendet, da diese den Vorteil der statistischen Voreingenommenheit hat.

## Setup 2 – RL-Verfahren

Das nächste Experiment untersucht die Auswirkungen, welche unterschiedliche RL-Lernverfahren auf das Agentenverhalten und MAS-Dynamiken haben. Das Vergleichsverfahren ist das On-Policy-Verfahren SARSA.

## Setup 4 – Nachbarschaftstypen

Hier soll untersucht werden, ob die Größe des Nachbarschaftstyps einen Einfluss auf das Agentenverhalten und MAS-Dynamiken hat. Dazu werden die Nachbarschaftstypen, welche im Abschnitt 3.5.2 erläutert wurden, miteinander verglichen.

## Setup 3 – Policy

Mit diesem Setup soll untersucht werden, welchen Einfluss eine unterschiedliche Verhaltenspolicy bei Q-Learning auf das Agentenverhalten und MAS-Dynamiken hat. Bei dem Wechsel zu Softmax wird die Größte des Unterschieds beziehungsweise der Unsicherheit des Agenten bezüglich der Q-Werte berücksichtigt. Für dieses Experiment ist der neue Hyperparameter Temperature erforderlich. Dieser wurde in Anlehnung an die Experimente von Sandholm und Crites (1996) gewählt und setzt sich aus der folgenden Konfiguration zusammen [4]:

1. Start-Temperature: 50,0
2. Temperature-Decay: 0,9999
3. Minimalwert der Temperature: 0,1

# Ergebnisse

## Setup 0 - Ergebnis des Baseline-Setup

In diesem Abschnitt werden die Ergebnisse des Baseline-Setups betrachtet und erläutert.

### Diagramme

Beginnend mit den Zeitreihendiagrammen werden die durchschnittliche Entwicklung der Agentenstrategien, die durchschnittliche Kooperationsrate, sowie den duchschnittlichen Reward gezeigt. Diese Diagramme wurden über zehn Durchläufen mit jeweils unterschiedlichen Seeds aggregiert.

Ein Bild, das Text, Diagramm, Screenshot, Reihe enthält.

KI-generierte Inhalte können fehlerhaft sein.

**Abbildung 7 - Zeitreihendiagramm von der aggregierten Strategieentwicklung aller Q-Learning-Agenten gemittelt über zehn verschiedenen Simulationsläufen.**

Der aggregierte Strategieentwicklungsgraph (siehe Abbildung 5) zeigt die durchschnittliche Greedy Policy (p-Vektor) über alle Agenten eines Typs. Die Entwicklung lässt sich in zwei Phasen unterteilen, die durch das Absinken der Explorationswahrscheinlichkeit Epsilon definiert sind.

Der aggregierte Strategieentwicklungsgraph zeigt, dass die durchschnittliche Strategie bei 50% beginnt, dann stark nahe Null abfällt und sich die Strategiewerte in der Explorationsphase, also Match 0 – 13800, mit einem geringen Fehlerband zwischen den Durchläufen kaum unterscheidet. Die Lernphase verläuft wie folgt: p(C|DD) weist den stärksten Anstieg mit einem maximum von ungefähr 0,34 und fällt dann wieder stark ab. Den zweitgrößten Anstieg erfährt der Wert p(C|CD), welcher anschließend ebenfalls stark abfällt. Die Werte p(C|CC) und p(C|DC) haben beide kaum einen Anstieg während der Lernphase und steigen erst zum Ende der Lernphase stark an.

Sobald die Explorationsphase beendet ist, also die Epsilonwahrscheinlichkeit bei allen Agenten das Minimum erreicht hat, ergibt sich ein neues Gesamtbild der Verläufe. Zunächst fällt auf, dass der Wert p(C|CD) stetig einen durchschnittlichen Wert nahe Null hat, anders als die anderen drei Werte, welche sich wellenartig und zyklisch im Bereich zwischen 0,5 und 0,2 bewegen. Dieser Bereich, in welchem sich diese drei Werte bewegen, wird ungefähr ab Match 75.000 schmäler und bewegt sich somit nun in einem Bereich von 0,3 – 0,5. Des Weiteren fällt auf, dass ab Match 25.000 bei den Werten p(C|CC), p(C|DC) und p(C|DD) eine stärkere Fluktuation festzustellen ist, was an den breiteren Fehlerbändern zu bemerken ist.

Ein Bild, das Text, Diagramm, Reihe, parallel enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung 8 - Zeitreihendiagramm der aggregierten Kooperaionsrate aller Q-Learning-Agenten gemittelt über zehn verschiedenen Simulationsläufen.

Der Graph (siehe Abbildung 6) beginnt mit einem Initialwert von 0% bei Match 0 und einem darauffolgenden initialen steilen Anstieg auf das globale Maximum des beobachteten Zeitraums von 43,5% bei Match 300. Anschließend fällt die Rate sofort wieder stark ab, verlangsamt sich jedoch wieder ab Match 3.000 stark und behält bis zum ungefähren Wendepunkt bei Match 7.000 einen flachen Abfall bei. Ab Match 7.000 ist der Abfall somit wieder verstärkt bis zum globalen Minimum des beobachteten Zeitraums von 13,5% bei Match 18.000. Ab diesem Punkt steigt der Graph zunächst stärker an und ab Match 50.000 schwächt die Zunahme ab. Die Kooperationsrate steigt bei Match 50.000 leicht, aber stetig in dem beobachteten Zeitraum bis zu 29% an und hat eine Standardabweichung zum Ende der Simulation von +-1,3%. Die geringe Standardabweichung deutet darauf hin, dass dieses Konvergenzverhalten über alle Simulationsdurchläufe hinweg robust ist.

### Rohdaten

Dieser Abschnitt präsentiert die erfassten Rohdaten gemittelt über zehn verschiedene Simulationsläufen mit jeweils unterschiedlichen Seeds.

Tabelle 1 - Finale Koordinationsrate, System-Effizienz und Agenten-Performance

|  |  |
| --- | --- |
|  | Gesamtzeitraum |
| Kooperationsrate (Durchschnitt) | 24,57% (+-1,10%) |
| System-Effizienz | 57,43% (+-0,94%) |
| Agenten-Performance | 343,80 (+-5,88) |

Tabelle 1 zeigt, wie gut das Gesamtsystem abgeschnitten hat, indem die durchschnittliche Kooperationsrate, der erzielte Gesamtreward aller Agenten sowie die durchschnittliche Agentenperformance betrachtet werden.

Tabelle 2 - Snapshot des Strategievorkommens zum Ende der Simulation

|  |  |
| --- | --- |
| Strategietyp | Anteil am System und Standardabweichung |
| Defektor (Rot) | 44,1% (+-3,3%) |
| Polarisiert (Orange) | 16,7% (+-4,5%) |
| Vorstufe-TFT (Cyan) | 15,4% (+-5,5%) |
| Tit-for-Tat (Blau) | 10,2% (+-1,8%) |
| Kooperator (Grün) | 10,2% (+-2,0%) |
| Win-Stay-Lose-Shift (Lila) | 3,4% (+-0,9%) |

Tabelle 2 zeigt, zu welchem Anteil die jeweiligen Strategietypen zum Ende der Simulation vorkommen. Die Werte in den Klammern sind die jeweiligen Standardabweichungen über alle zehn Durchläufe hinweg.

### Analyse-Dashboard

Ein Bild, das Screenshot, Text enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung 9 - Die Abbildung zeigt oben die Entwicklung der Kooperationsrate anhand von Heatmaps über die Dauer der Simulation hinweg. Die Zahlen über den Heatmaps geben den jeweiligen Matchcount an, zu welchem Zeitpunkt die Heatmap erstellt wurde. Der untere Teil zeigt die Entwicklung des kumulativen Rewards. Diese Verläufe stammen aus dem Durchlauf mit Seed 0. Dieser Durchlauf spiegelt den konsistenten Trend wider, der über alle Simulationsdurchläufe hinweg beobachtet wurde.

In Abbildung 9 sind die beiden Entwicklungen der Kooperationsraten (oben) sowie des kumulativen Rewards (unten) zu sehen. In der Lernphase entwickelt sich das System zu einer Kooperationsrate von ungefähr 50%, wird daraufhin jedoch zunächst stark defektiv. Nachdem die Explorationsphase vorbei ist, ist die Entstehung von defektiven und kooperativen Clustern zu bemerken. Ein Cluster ist in diesem Kontext eine Fläche an Agenten mit ähnlicher Kooperationsrate oder Performance im Gitter, welche ohne Lücken zusammenhängen.

Ein Bild, das Screenshot, Farbigkeit, Text, Multimedia-Software enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung 10 - Gesamtbild des Analyse-Dashboards zum Matchcount 200.000 (Seed 0).

In Abbildung 10 sieht man, dass eine gemischte und große Vielfalt an verschiedenen Strategien emergieren.

Ein Bild, das Text, Screenshot, Essen enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung 11 - Endzustände von zehn verschiedenen Seeds (0-9). Das bedeutet, alle Heatmaps wurden bei Match 200.000 erstellt. Der obere Teil sind die Seeds 0-4, der untere Teil die Seeds 5-9. Die jeweils obere Heatmap zeigt die Kooperationsraten-Heatmap und die Untere zeigt die Heatmap für den kumulativen Reward.

Abbildung 11 zeigt, dass zwar stets kooperative Cluster im MAS entstehen, jedoch immer unterschiedliche Nachbarschaften zu kooperativen Clustern werden.

## Setup 1 - Einfluss des Begegnungsschemas

In diesem Setup wird nun untersucht, wie sich das Begegnungsschema auf die Dynamiken des gegebenen MAS auswirkt.

### Diagramme

Ein Bild, das Text, Screenshot, Diagramm, Reihe enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung 12 - Zeitreihendiagramm von der aggregierten Strategieentwicklung aller Q-Learning-Agenten gemittelt über zehn verschiedenen Simulationsläufen.

Bei diesem Strategieentwicklungsgraph ist die anfängliche Explorationsphase sehr ähnlich im Vergleich mit der Strategieentwicklung im räumlichen Gitter. Nach der Explorationsphase sind dann Unterschiede bemerkbar. Die Wahrscheinlichkeit p(C|DD) bewegt sich in einem Bereich zwischen 0,15 und 0,35 und ist somit durchschnittlich deutlich geringer als im räumlichen Gitter, dafür aber stabiler. Das gleiche gilt für die Werte p(C|CC) und p(C|DC). Diese sind jedoch auf einem höheren Niveau im Bereich zwischen 0,3 und 0,4 ab Match 50.000.

Ein Bild, das Text, Reihe, Diagramm, Zahl enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung 13 - Zeitreihendiagramm der aggregierten Kooperaionsrate aller Q-Learning-Agenten gemittelt über zehn verschiedenen Simulationsläufen.

Die Entwicklung der durchschnittlichen Kooperationsrate in größtenteils analog zur Entwicklung der durchschnittlichen Kooperationsrate im räumlichen Gitter, mit dem Unterschied, dass der Anstieg ab Matchcount 50.000 ungefähr, etwas schwächer ist als im räumlichen Gitter. Die durchschnittliche Kooperationsrate am Ende der Simulation beläuft sich auf 22,42% mit einer Standardabweichung von 0,91%.

### Rohdaten

Tabelle 3 - Finale Koordinationsrate, System-Effizienz und Agenten-Performance

|  |  |  |
| --- | --- | --- |
|  | Gesamtzeitraum | Differenz (∆) zum Baseline-Setup |
| Kooperationsrate (Durchschnitt) | 20,74% (+-0,92%) | ∆ -3,83% (∆ -0,18%) |
| System-Effizienz | 51,58% (+-0,62%) | ∆ -5,85% (∆ -0,32%) |
| Agenten-Performance | 309,47 (+-3,78) | ∆ -34,33 (∆ -2,1) |

Die Zufallspaarung hat in allen System-Metriken (siehe Tabelle 3) schlechter abgeschnitten als das räumliche Gitter.

Tabelle 4 - Snapshot des Strategievorkommens zum Ende der Simulation

|  |  |  |
| --- | --- | --- |
| Strategietyp | Anteil am System und Standardabweichung | Differenz (∆) zum Baseline-Setup |
| Defektor (Rot) | 55,2% (+-2,8%) | ∆ +11,1% (∆ -0,5%) |
| Vorstufe-TFT (Cyan) | 14,3% (+-3,3%) | ∆ -1,1% (∆ -1,2%) |
| Tit-for-Tat (Blau) | 13,2% (+-2,1%) | ∆ +3,0% (∆ +0,3%) |
| Polarisiert (Orange) | 8,9% (+-2,5%) | ∆ -7,8% (∆ -2,0%) |
| Kooperator (Grün) | 7,2% (+-1,7%) | ∆ -3,0% (∆ -0,3%) |
| Win-Stay-Lose-Shift (Lila) | 1,5% (+-0,5%) | ∆ -1,9% (∆ -0,4%) |

Defektoren sind bei der Zufallspaarung um ca. 11% präsenter als im räumlichen Gitter. Die Vorstufe von TFT ist fast identisch vertreten, mit einer Differenz von lediglich 1,1%. Tit-For-Tat hat einen Anstieg von 3,0%, während starke Kooperatoren um 3,0% weniger oft vorkommen. Polarisierte Strategien haben mit -7,8% die zweitgrößte Differenz. Win-Stay-Lose-Shift ist um 1,9% gesunken.

Die Werte in den Klammern zeigen die Differenz der Standardabweichung zwischen den Experimenten.

### Analyse-Dashboard

Ein Bild, das Screenshot, Text, Farbigkeit, Design enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung 14 - Die Abbildung zeigt oben die Entwicklung der Kooperationsrate anhand von Heatmaps über die Dauer der Simulation hinweg. Diese Verläufe stammen aus dem Durchlauf mit Seed 0. Dieser Durchlauf spiegelt den konsistenten Trend wider, der über alle Simulationsdurchläufe hinweg beobachtet wurde.

Die Initialphase ist analog zur Entwicklung der Kooperationsraten und des Rewards der Agenten im räumlichen Gitter: Das System startet mit einer Kooperationsrate von 50%, welche dann anschließend stark abfällt in Richtung Defektion. Nach diesen beiden Phasen kristallisieren sich sehr verstreut dann einzelne Kooperatoren heraus. Das bedeutet, dass im Gegensatz zum räumlichen Gitter, in welchem Cluster entstehen, die kooperative Emergenz deutlich verstreuter und weniger zusammenhängend ist. Diese Beobachtung ist Seed-unabhängig, wie man an Abbildung 16 erkennt.

Ein Bild, das Screenshot, Text, Multimedia-Software, Farbigkeit enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung 15 - Gesamtbild des Analyse-Dashboards zum Matchcount 200.000 (Seed 0).

Auch bei rein zufälliger Agentenpaarung entstehen kooperationsfreudige Agenten und eine allgemein vielfältige Emergenz an unterschiedlichen Strategien, jedoch mit dem Unterschied, dass diese deutlich verstreuter sind, als bei den begrenzten Nachbarschaften im räumlichen Gitter.

Ein Bild, das Text, Screenshot, Muster enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung 16 - Endzustände von zehn verschiedenen Seeds (0-9). Das bedeutet, alle Heatmaps wurden bei Match 200.000 erstellt.

## Setup 2 - Vergleichende Betrachtung der Lernalgorithmen und deren Auswirkungen

In diesem Abschnitt werden Q-Learning- und SARSA-Agenten im IGD-MAS gegenübergestellt und ihr Verhalten vergleichend betrachtet.

### Diagramme

Ein Bild, das Text, Screenshot, Diagramm, Reihe enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung 17 - Zeitreihendiagramm von der aggregierten Strategieentwicklung aller SARSA-Agenten gemittelt über zehn verschiedenen Simulationsläufen.

Die Analyse der aggregierten Strategieentwicklung (siehe Abbildung 17) zeigt in der frühen Explorationsphase (Match 0 bis 13.800) leichte Fluktuationen.

Die Strategien der Agenten haben während der Explorationsphase einen kurzzeitigen und sehr kleinen Höhepunkt bis hin zu 0,13 beim Wert p(C|DD) zum Matchcount 2500 und 0,8 beim Wert p(C|CD) zum Matchcount 5000. Die anderen beiden Werte sind nahe Null. Nach absinken des Epsilons konvergieren die Werte p(C|CC), p(C|CD) und p(C|DD) nahe Null. Eine Ausnahme bildet der Wert p(C|DC), der nicht vollständig gegen Null geht, sondern sich auf einem Niveau von ca. 0,2 stabilisiert.

Ein Bild, das Text, Diagramm, Reihe, parallel enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung 18 - Zeitreihendiagramm der aggregierten Kooperaionsrate aller SARSA-Agenten gemittelt über zehn verschiedenen Simulationsläufen.

Nach einem kurzfristigen Anstieg der durchschnittliche Kooperationsrate aller Agenten während der Explorationsphase, fällt diese wieder stark ab und konvergiert nahe Null. Die Kooperationsrate am Ende der Simulation ist 1,94% mit einer Standardabweichung von 0,39%.

### Rohdaten

Tabelle 5 - Finale Koordinationsrate, System-Effizienz und Agenten-Performance

|  |  |  |
| --- | --- | --- |
|  | Gesamtzeitraum | Differenz (∆) zum Baseline-Setup |
| Kooperationsrate (Durchschnitt) | 2,91% (+-0,23%) | ∆ -21,66% (∆ -0,87%) |
| System-Effizienz | 35,09% (+-0,37%) | ∆ -22,34% (∆ -0,57%) |
| Agenten-Performance | 210,46(+-2,29) | ∆ 133,34 (∆ -3,59) |

Tabelle 6 - Snapshot des Strategievorkommens zum Ende der Simulation

|  |  |  |
| --- | --- | --- |
| Strategietyp | Anteil am System und Standardabweichung | Differenz (∆) zum Baseline-Setup |
| Defektor (Rot) | 96,0% (+-1,8%) | ∆ +51,9% (∆ -1,5%) |
| Polarisiert (Orange) | 2,5% (+-1,5%) | ∆ -14,2% (∆ -3,5%) |
| Tit-for-Tat (Blau) | 1,1% (+-0,6%) | ∆ -9,1% (∆ -1,2%) |
| Kooperator (Grün) | 0,7% (+-0,2%) | ∆ -9,5% (∆ -1,8%) |
| Vorstufe-TFT (Cyan) | 0,5% (+-0,0%) | ∆ -14,9% (∆ -5,5%) |
| Win-Stay-Lose-Shift (Lila) | 0,0% (+-0,0%) | ∆ -3,4% (∆ -0,9%) |

Defektoren sind um 51,9% mehr vertreten als in der Baseline hin zu einem Gesamtvorkommen von 96%. Die anderen Strategietypen haben somit alle and stark Vorkommen verloren.

### Analyse-Dashboard

Ein Bild, das Screenshot, Text, Farbigkeit enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung 19 – Entwicklung der Kooperationsrate sowie des kumulativen Rewards.

Die anfängliche Explorationsrate ist sehr ähnlich zum Setup 0, jedoch erholen sich die SARSA-Agenten nach der anfänglichen Defektionsphase nicht, sondern verweilen mit einer rein defektiven Strategie.

Ein Bild, das Text, Screenshot, Multimedia-Software, Farbigkeit enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung 20 – Snapshot des gesamten Analyse-Dashboards zum Ende der Simulation.

Nahezu alle SARSA-Agenten haben eine rein defektive Strategie erlernt.

Ein Bild, das Text, Screenshot enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung 21 - Endzustände von zehn verschiedenen Seeds (0-9). Das bedeutet, alle Heatmaps wurden bei Match 200.000 erstellt.

In vergleichender Betrachtung fällt auf, dass SARSA-Agenten unter denselben Bedingungen keine kooperativen Cluster bilden. Kooperation allgemein entsteht bei SARSA-Agenten unter den gegebenen Bedingungen kaum.

## Setup 3.1 - Einfluss der Nachbarschaftsgröße (Von-Neumann-Nachbarschaft)

### Diagramme

Ein Bild, das Text, Diagramm, Reihe enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung 23 - Zeitreihendiagramm von der aggregierten Strategieentwicklung aller Q-Learning-Agenten gemittelt über zehn verschiedenen Simulationsläufen.

Auch bei diesem Setup ist die Explorationsphase größtenteils analog zur Baseline. Danach ist jedoch ein starker Anstieg von p(C|DC) und p(C|DD) zu bemerken. Das Maximum von p(C|DC) von ungefähr 0,73 wird ca. bei Match 75.000 erreicht, schwächt danach leicht ab und pendelt sich dann bei ungefähr 0,65 ein. Zudem ist eine hohe Standardabweichung bei p(C|DC) zwischen Match 100.000 und 130.000 auffällig. Der Wert p(C|DD) verläuft analog zu p(C|DC), jedoch auf einem niedrigeren Niveau zwischen 0,55 und 0,6. Der Wert p(C|CC) erfährt einen deutlich schwächeren Anstieg nach der Explorationsphase auf zunächst 0,4, fällt dann leicht ab auf 0,31 bei Match 100.000, steigt jedoch ca. ab Match 120.000 wieder auf 0,44 an. Der Wert p(C|CD) ist wie auch bei der Baseline auf einem deutlich geringeren Niveau nahe Null nach der Explorationsphase, steigt aber in diesem Experiment ab ungefähr match 80.000 leicht an auf ca. 0,07.

Ein Bild, das Text, Diagramm, Reihe, parallel enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung 22 - Zeitreihendiagramm der aggregierten Kooperaionsrate aller Q-Learning-Agenten gemittelt über zehn verschiedenen Simulationsläufen.

Die durchschnittliche Kooperationsrate steigt nach der Explorationsphase stark an und zum Ende der Simulation einen Wert von 40,56%.

### Rohdaten

Tabelle 7 - Finale Koordinationsrate, System-Effizienz und Agenten-Performance

|  |  |  |  |
| --- | --- | --- | --- |
|  | Gesamtzeitraum | Differenz (∆) zum Baseline-Setup | Differenz (∆) zu 3.2 |
| Kooperationsrate (Durchschnitt) | 32,48% (+- 0,76%) | ∆ % (∆ %) | ∆ % (∆ %) |
| System-Effizienz | 68,62% (+- 0,85%) | ∆ +11,19% (∆ -0,18%) | ∆ % (∆ %) |
| Agenten-Performance | 411,49 (+-5,06) | ∆ +67,69 (∆ -0,82) | ∆ (∆ ) |

Tabelle 8 - Snapshot des Strategievorkommens zum Ende der Simulation

|  |  |  |  |
| --- | --- | --- | --- |
| Strategietyp | Anteil am System und Standardabweichung | Differenz (∆) zum Baseline-Setup | Differenz (∆) zu 3.2 |
| Defektor (Rot) | 31,4% (+- 3,5%) | ∆ -12,7% (∆ +0,2%) | ∆ % (∆ %) |
| Polarisiert (Orange) | 20,9% (+- 2,3%) | ∆ +4,2% (∆ -2,2%) | ∆ % (∆ %) |
| Tit-for-Tat (Blau) | 12,8% (+- 3,1%) | ∆ +2,6% (∆ +1,3%) | ∆ % (∆ %) |
| Kooperator (Grün) | 21,9% (+- 2,7%) | ∆ +11,7% (∆ +0,7%) | ∆ % (∆ %) |
| Vorstufe-TFT (Cyan) | 10,2% (+- 3,1%) | ∆ -5,2% (∆ -2,4%) | ∆ % (∆ %) |
| Win-Stay-Lose-Shift (Lila) | 3,0% (+- 1,4%) | ∆ -0,4% (∆ +0,5%) | ∆ % (∆ %) |

### Analyse-Dashboard

Ein Bild, das Screenshot, Text enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung 24 - Entwicklung der Kooperationsrate sowie des kumulativen Rewards.

Ein Bild, das Screenshot, Multimedia-Software, Farbigkeit, Grafiksoftware enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung 25 - Snapshot des gesamten Analyse-Dashboards zum Ende der Simulation.

Der Snapshot am Ende der Simulation visualisiert, dass kooperative Strategie stärker vertreten sind als defektive Strategien.

Ein Bild, das Text, Screenshot enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung 26 - Endzustände von zehn verschiedenen Seeds (0-9). Das bedeutet, alle Heatmaps wurden bei Match 200.000 erstellt.

## Setup 3.2 – Einfluss der Nachbarschaftsgrößte (Moore-Nachbarschaft)

### Diagramme

Ein Bild, das Text, Diagramm, Reihe, Screenshot enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung 27 – Zeitreihendiagramm von der aggregierten Strategieentwicklung aller Q-Learning-Agenten gemittelt über zehn verschiedenen Simulationsläufen.

Ein Bild, das Text, Diagramm, Reihe, parallel enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung 28 - Zeitreihendiagramm der aggregierten Kooperaionsrate aller Q-Learning-Agenten gemittelt über zehn verschiedenen Simulationsläufen.

### Rohdaten

Tabelle 9 - Finale Koordinationsrate, System-Effizienz und Agenten-Performance

|  |  |  |  |
| --- | --- | --- | --- |
|  | Gesamtzeitraum | Differenz (∆) zum Baseline-Setup | Differenz (∆) zu 3.1 |
| Kooperationsrate (Durchschnitt) | % (+- %) | ∆ % (∆ %) | ∆ % (∆ %) |
| System-Effizienz | % (+- %) | ∆ % (∆ %) | ∆ % (∆ %) |
| Agenten-Performance | (+- ) | ∆ (∆ ) | ∆ (∆ ) |

Tabelle 10 - Snapshot des Strategievorkommens zum Ende der Simulation

|  |  |  |  |
| --- | --- | --- | --- |
| Strategietyp | Anteil am System und Standardabweichung | Differenz (∆) zum Baseline-Setup | Differenz (∆) zu 3.1 |
| Defektor (Rot) | % (+- %) | ∆ % (∆ %) | ∆ % (∆ %) |
| Polarisiert (Orange) | % (+- %) | ∆ % (∆ %) | ∆ % (∆ %) |
| Tit-for-Tat (Blau) | % (+- %) | ∆ % (∆ %) | ∆ % (∆ %) |
| Kooperator (Grün) | % (+- %) | ∆ % (∆ %) | ∆ % (∆ %) |
| Vorstufe-TFT (Cyan) | % (+- %) | ∆ % (∆ %) | ∆ % (∆ %) |
| Win-Stay-Lose-Shift (Lila) | % (+- %) | ∆ % (∆ %) | ∆ % (∆ %) |

### Analyse-Dashboard

## Setup 4 - Einfluss der Policy des Agenten (Vergleich Softmax und Epsilon-Greedy)

### Diagramme

### Rohdaten

### Analyse-Dashboard

# Interpretation der Ergebnisse

## Setup 1

Die geringere Kooperationsrate bei der Zufallspaarung bestätigt die These, dass räumliche Struktur Kooperation fördert. Dies geschieht nicht durch ein explizites Reputationsmodell, sondern durch Netzwerk-Reziprozität. Da die Interaktionspartner im Gitter stabil bleiben, erzeugt das Verhalten des jeweiligen Nachbarn einen spezifischen lokalen Zustand (z. B. dauerhafte gegenseitige Kooperation), auf den der Q-Learning-Agent spezifisch reagieren kann. Die Reputation eines Nachbarn ist somit implizit in der gemeinsamen Spielhistorie kodiert, was im anonymen Random-Pairing durch den ständigen Partnerwechsel unmöglich ist.

Dieser Effekt manifestiert sich in der Simulation durch die beobachtete Bildung von kooperativen Clustern, statt verstreute, einzelne Kooperatoren und die dadurch entstandene höhere System-Effizienz, Agenten-Performance und durchschnittliche Kooperationsrate.

Die häufigere Emergenz von Tit-for-Tat-Strategien unter den Agenten ist ebenfalls ein Indikator für die oben beschriebene Begründung, denn Tit-for-Tat ist anders als (nahezu) bedingungslose Kooperatoren (Grün) robuster gegen defektive Agenten.

## Setup 2

Die Ergebnisse zwischen Q-Learning und SARSA lassen sich auf die unterschiedliche Berechnung des TD-Targets in Abhängigkeit von der Explorationsstrategie zurückführen.

Als Off-Policy-Verfahren approximiert Q-Learning den Wert der optimalen Strategie unabhängig von der tatsächlich ausgeführten Verhaltenspolicy. Durch die Verwendung des max-Operators () im Update-Schritt wird der Einfluss der stochastischen Exploration bei der Bewertung zukünftiger Zustände ignoriert. Dies ermöglicht dem Algorithmus, das Potenzial der Kooperation als theoretisches Optimum zu identifizieren, selbst wenn das aktuelle Verhalten der Population noch durch Zufallsaktionen geprägt ist.

Im Gegensatz dazu bewertet SARSA als On-Policy-Verfahren den Wert der tatsächlich verfolgten Strategie, einschließlich der explorativen Aktionen. Das TD-Target basiert auf der im nächsten Schritt real ausgeführten Aktion a‘, welche mit der Wahrscheinlichkeit ε zufällig gewählt wird. Im Iterierten Gefangenendilemma birgt Kooperation das Risiko, durch eine zufällige Reaktion des Gegners oder einen eigenen Fehler ausgebeutet zu werden (S = 0). Da SARSA diese negativen Konsequenzen der Exploration direkt in die Erwartungswertbildung einbezieht, wird die kooperative Strategie während der Lernphase abgewertet.

Dieses Verhalten entspricht dem „Cliff Walking“-Beispiel [3], bei welchem SARSA statt der optimalen Strategie, welche direkt neben dem Abgrund, also dem Cliff, verläuft, eine sicherere Strategie weiter entfernt vom Abgrund lernt, da SARSA eben den Zufall mit einkalkuliert und somit anders wie Q-Learning nicht aufgrund von zufälligen Aktionen in den Abgrund stürzt. Analog dazu identifiziert SARSA die Kooperation als risikobehafteten Zustand und konvergiert folglich zur sichereren Defektionsstrategie, um die durch Exploration verursachten Verluste zu minimieren. Q-Learning hingegen konvergiert zur optimalen Strategie unter der Annahme, dass die Exploration langfristig entfällt.  
Und da zum Zeitpunkt von Epsilon = 0,01 bereits jeder SARSA-Agent eine stark defektive Strategie erlernt hat, wird jeder Versuch zu kooperieren sofort bestraft, weshalb das System zu Defektion konvergiert.

## Setup 3

Die Ergebnisse aus Setup 3 (Variation der Nachbarschaftsgröße) bestätigen und differenzieren den bereits in Setup 1 beobachteten positiven Effekt räumlicher Strukturen auf die Kooperation. Es zeigt sich eine deutliche negative Korrelation zwischen der Nachbarschaftsgröße und der Kooperationsrate: Je kleiner die lokale Nachbarschaft eines Agenten ist, desto höher liegen die erzielte Kooperationsrate sowie die resultierende System-Effizienz.

Im untersuchten Spektrum (von vier Nachbarn bei Von-Neumann bis zu zwölf bei der erweiterten Moore-Nachbarschaft) erwies sich die kleinste Nachbarschaft als die Kooperationsförderndste, was sich durch die Intensität der lokalen Interaktion erklären lässt: In kleineren Nachbarschaften ist der relative Einfluss eines einzelnen, reziproken Nachbarn größer, was die Bildung und Stabilisierung von kooperativen Clustern erleichtert und die Wirkung von einzelnen Defektoren minimiert. Eine größere Nachbarschaft erhöht hingegen die Wahrscheinlichkeit, auf Defektoren zu treffen, was die Q-Table destabilisieren kann.

## Setup 4

# Zusammenfassung und Ausblick

[Zusammenfassung]

## Erweiterung des spieltheoretischen Modells

Um die Komplexität realer Interaktionen präziser abzubilden, bietet das zugrundeliegende Modell diverse Erweiterungsmöglichkeiten. Ein möglicher Ansatz wäre die Integration evolutionärer Mechanismen. Während Agenten aktuell ausschließlich aus eigener Erfahrung lernen, könnte ein zusätzlicher Imitationsschritt, bei dem Strategien erfolgreicher Nachbarn übernommen werden, die Ausbreitung kooperativer Cluster beschleunigen und stabilisieren.

Eine weitere vielversprechende Erweiterung wäre die Integration der verbleibenden Mechanismen aus Nowaks „Five Rules fort he Evolution of Cooperation“ [8]. Diese Arbeit hat bereits die direkte Reziprozität und die Netzwerk-Reziprozität integriert. Die Verwandtenselektion, also Ähnlichkeitsgrade zwischen den Agenten, Gruppenselektion, also Wettbewerb auf kollektiver Ebene, sowie indirekte Reziprozität, also ein explizites Reputationssystem, zum Beispiel ein „Trust Score“, welcher von anderen Agenten abgerufen werden kann, könnten noch ergänzt werden.

Darüber hinaus lassen sich manche technischen Probleme, wie etwa Smart Grids, womöglich besser durch N-Spieler-Spiele wie Public Goods Games beschreiben als durch paarweise Begegnungen, weshalb eine Ausweitung der Interaktion auf mehr als zwei gleichzeitige Spieler ein möglicher nächster Schritt wäre. Abschließend könnte die Robustheit der entstehenden kooperativen Strukturen gezielt validiert werden, indem ausbeuterische Agenten in eine bestehende Population injiziert werden, um die Resilienz des Systems gegenüber solchen Störungen zu analysieren.

## Reinforcement Learning spezifische Erweiterungen

Neben den spieltheoretischen Aspekten stellt auch die Beschränkung auf tabellenbasiertes Q-Learning und diskrete Aktionen eine Vereinfachung dar, die durch moderne Verfahren des Reinforcement Learnings erweitert werden kann. Da Kooperation in realen Anwendungsfällen oft kein binärer Zustand, sondern ein Kontinuum ist, beispielsweise bei der Zuteilung von Bandbreite, wären Verfahren für das Continuous Action Iterated Prisoner's Dilemma (CAIPD) von größerer Relevanz. Hierfür müssten tabellenbasierte Ansätze durch Policy-Gradient-Methoden oder Actor-Critic-Ansätze wie A2C oder PPO unter Einsatz neuronaler Netze zur Funktionsapproximation ersetzt werden.

Ein weiterer wesentlicher Schritt betrifft das Gedächtnis der Agenten. Um über die aktuellen Memory-One-Strategien hinauszugehen, würde der Einsatz von Deep Q-Networks (DQN) oder rekurrenten neuronalen Netzen es ermöglichen, längere Historien zu analysieren und komplexe, zeitlich gestreckte Muster, wie etwa periodische Ausbeutung, zu erkennen. Zudem behandelt Q-Learning andere Agenten lediglich als Teil einer stationären Umgebung, was im Multi-Agenten-Kontext theoretisch ungenau ist. Abhilfe könnten hier spezialisierte Algorithmen schaffen, wie beispielsweise WoLF-PHC, das die Lernrate an die eigene Performance anpasst, oder QMIX, welches für kooperative Szenarien zur Maximierung gemeinsamer Belohnungen entwickelt wurde.

Anstatt dass alle Akteure dasselbe Hyperparameter-Set nutzen, könnten sie individuelle Lernraten oder Diskontierungsfaktoren besitzen. Denkbar wäre in diesem Kontext auch eine dynamische Hyperparameter-Optimierung.

# Anhang

# Ein Bild, das Text, Diagramm, Plan, parallel enthält. KI-generierte Inhalte können fehlerhaft sein.Abbildungsverzeichnis

[Abbildung 1 Auszahlungsmatrix mit Standardwerten im Gefangenendilemma 7](#_Toc215239164)

[Abbildung 2 (Platzhalter) Visualiserung der Reinforcement Learning Pipeline 10](#_Toc215239165)

[Abbildung 3 – Nachbarschftstypen von links: Von-Neumann-, Moore- und erweiterte Moore-Nachbarschft 19](#_Toc215239166)

[Abbildung 4 - Klassendiagramm der Simulation (vereinfacht) 21](#_Toc215239167)

[Abbildung 5 - Flussdiagramm des Hauptskripts 25](#_Toc215239168)

[Abbildung 6 - Snapshot des Analyse-Dashboards zum Matchcount 200.000 von Seed 0. 27](#_Toc215239169)

[**Abbildung 7 - Zeitreihendiagramm von der aggregierten Strategieentwicklung aller Q-Learning-Agenten gemittelt über zehn verschiedenen Simulationsläufen.** 30](#_Toc215239170)

[Abbildung 8 - Zeitreihendiagramm der aggregierten Kooperaionsrate aller Q-Learning-Agenten gemittelt über zehn verschiedenen Simulationsläufen. 31](#_Toc215239171)

[Abbildung 9 - Die Abbildung zeigt oben die Entwicklung der Kooperationsrate anhand von Heatmaps über die Dauer der Simulation hinweg. Die Zahlen über den Heatmaps geben den jeweiligen Matchcount an, zu welchem Zeitpunkt die Heatmap erstellt wurde. Der untere Teil zeigt die Entwicklung des kumulativen Rewards. Diese Verläufe stammen aus dem Durchlauf mit Seed 0. Dieser Durchlauf spiegelt den konsistenten Trend wider, der über alle Simulationsdurchläufe hinweg beobachtet wurde. 33](#_Toc215239172)

[Abbildung 10 - Gesamtbild des Analyse-Dashboards zum Matchcount 200.000 (Seed 0). 34](#_Toc215239173)

[Abbildung 11 - Endzustände von zehn verschiedenen Seeds (0-9). Das bedeutet, alle Heatmaps wurden bei Match 200.000 erstellt. Der obere Teil sind die Seeds 0-4, der untere Teil die Seeds 5-9. Die jeweils obere Heatmap zeigt die Kooperationsraten-Heatmap und die Untere zeigt die Heatmap für den kumulativen Reward. 34](#_Toc215239174)

[Abbildung 12 - Zeitreihendiagramm von der aggregierten Strategieentwicklung aller Q-Learning-Agenten gemittelt über zehn verschiedenen Simulationsläufen. 35](#_Toc215239175)

[Abbildung 13 - Zeitreihendiagramm der aggregierten Kooperaionsrate aller Q-Learning-Agenten gemittelt über zehn verschiedenen Simulationsläufen. 36](#_Toc215239176)

[Abbildung 14 - Die Abbildung zeigt oben die Entwicklung der Kooperationsrate anhand von Heatmaps über die Dauer der Simulation hinweg. Diese Verläufe stammen aus dem Durchlauf mit Seed 0. Dieser Durchlauf spiegelt den konsistenten Trend wider, der über alle Simulationsdurchläufe hinweg beobachtet wurde. 38](#_Toc215239177)

[Abbildung 15 - Gesamtbild des Analyse-Dashboards zum Matchcount 200.000 (Seed 0). 39](#_Toc215239178)

[Abbildung 16 - Endzustände von zehn verschiedenen Seeds (0-9). Das bedeutet, alle Heatmaps wurden bei Match 200.000 erstellt. 39](#_Toc215239179)

[Abbildung 17 - Zeitreihendiagramm von der aggregierten Strategieentwicklung aller SARSA-Agenten gemittelt über zehn verschiedenen Simulationsläufen. 40](#_Toc215239180)

[Abbildung 18 - Zeitreihendiagramm der aggregierten Kooperaionsrate aller SARSA-Agenten gemittelt über zehn verschiedenen Simulationsläufen. 41](#_Toc215239181)

[Abbildung 19 – Entwicklung der Kooperationsrate sowie des kumulativen Rewards. 42](#_Toc215239182)

[Abbildung 20 – Snapshot des gesamten Analyse-Dashboards zum Ende der Simulation. 43](#_Toc215239183)

[Abbildung 21 - Endzustände von zehn verschiedenen Seeds (0-9). Das bedeutet, alle Heatmaps wurden bei Match 200.000 erstellt. 43](#_Toc215239184)

[Abbildung 23 - Zeitreihendiagramm von der aggregierten Strategieentwicklung aller Q-Learning-Agenten gemittelt über zehn verschiedenen Simulationsläufen. 44](#_Toc215239185)

[Abbildung 22 - Zeitreihendiagramm der aggregierten Kooperaionsrate aller Q-Learning-Agenten gemittelt über zehn verschiedenen Simulationsläufen. 44](#_Toc215239186)

[Abbildung 24 - Entwicklung der Kooperationsrate sowie des kumulativen Rewards. 46](#_Toc215239187)

[Abbildung 25 - Snapshot des gesamten Analyse-Dashboards zum Ende der Simulation. 46](#_Toc215239188)

[Abbildung 26 - Endzustände von zehn verschiedenen Seeds (0-9). Das bedeutet, alle Heatmaps wurden bei Match 200.000 erstellt. 47](#_Toc215239189)

[Abbildung 27 – 47](#_Toc215239190)

[Abbildung 28 - 48](#_Toc215239191)

# Tabellenverzeichnis

[Tabelle 1 - Finale Koordinationsrate, System-Effizienz und Agenten-Performance 32](#_Toc215239152)

[Tabelle 2 - Snapshot des Strategievorkommens zum Ende der Simulation 32](#_Toc215239153)

[Tabelle 3 - Finale Koordinationsrate, System-Effizienz und Agenten-Performance 36](#_Toc215239154)

[Tabelle 4 - Snapshot des Strategievorkommens zum Ende der Simulation 37](#_Toc215239155)

[Tabelle 5 - Finale Koordinationsrate, System-Effizienz und Agenten-Performance 41](#_Toc215239156)

[Tabelle 6 - Snapshot des Strategievorkommens zum Ende der Simulation 41](#_Toc215239157)

[Tabelle 7 - Finale Koordinationsrate, System-Effizienz und Agenten-Performance 45](#_Toc215239158)

[Tabelle 8 - Snapshot des Strategievorkommens zum Ende der Simulation 45](#_Toc215239159)

[Tabelle 9 - Finale Koordinationsrate, System-Effizienz und Agenten-Performance 48](#_Toc215239160)

[Tabelle 10 - Snapshot des Strategievorkommens zum Ende der Simulation 48](#_Toc215239161)

# Literaturverzeichnis

|  |  |
| --- | --- |
| [1] | A. Dorri, S. S. Kanhere und R. Jurdak, „Multi-Agent Systems: A Survey,“ *IEEE Access,* Bd. 6, pp. 28573-28593, 2018. |
| [2] | M. A. Nowak und R. M. May, „Evolutionary games and spatial chaos,“ *Nature,* Bd. 359, Nr. 6398, pp. 826-829, 1992. |
| [3] | S. J. Russel und P. Norvig, Artificial Intelligence: A Modern Approach 4th ed., Global Ed., Harlow, England: Pearson, 2021. |
| [4] | R. Axelrod, The Evolution of Cooperation, New York, NY: Basic Books, 1984. |
| [5] | R. S. Sutton und A. G. Barto, Reinforcement Learning: An Introduction, 2nd ed., Cambridge, MA: MIT Press, 2018. |
| [6] | J. García und M. Van Veelen, „No strategy can win in the repeated prisoner's dilemma: linking game theory and computer simulations,“ *Frontiers in Robotics and AI,* Bd. 5, p. 102, 2018. |
| [7] | T. W. Sandholm und R. H. Crites, „Multiagent reinforcement learning in the Iterated Prisoner's Dilemma,“ *BioSystems,* Bd. 37, Nr. 1-2, pp. 147-166, 1996. |
| [8] | M. A. Nowak, „Five Rules for the Evolution of Cooperation,“ *Science,* Bd. 314, Nr. 5805, pp. 1560-1563, 2001. |

|  |
| --- |
|  |