KI-Konzepte für das Erlernen nicht deterministischer Spiele am Beispiel von „EinStein würfelt nicht!“

Bacherloarbeit zur Erlangung des Bachelorgrades
Bachelor of Science im Studiengang Informatik
an der Fakultät für Informatik und Ingenieurwissenschaften
der Technischen Hochschule Köln

vorgelegt von:	Julian Rolf Siegfried Cöln
Matrikel-Nr.:	11117175
Adresse:	Seestraße 8
	50374 Erftstadt
	julian.coeln@hotmail.de
eingereicht bei:	Prof. Dr. Wolfgang Konen
Zweitgutachter/in:	Prof. Dr. Frank Victor

Gummersbach, 10.01.2022

Kurzfassung

Das nichtdeterministische Spiel „EinStein würfelt nicht!“ mit perfekten Informationen stellt für die verschiedenen KI Methodiken eine Herausforderung dar. Durch Hilfe des General Board Game Frameworks wurde im Vorfeld der Arbeit die Umgebung für „EinStein würfelt nicht!“ mit den verschiedenen zwei und vier Spieler Varianten erstellt. Die zur Verfügung stehenden Reinforcement Learning Agenten, Monte Carlo, Monte Carlo Tree Search Expectimax (MCTSE) und der Temporal Difference (TD) konnten in der klassischen zwei Spieler Variante, alle Gewinnraten gegen den Zufallsagenten von über 79 % erzielen. Der TD Agent konnte gegen den MCTSE eine Gewinnrate von 44 % erzielen. In der 4 Spielervariante entfiel der Lernerfolg jedoch für den Temporal Diffference Agenten aufgrund von einer zu kleinen Anzahl von Gewichtungen. Die erfolgreiche Implementierung des Replaybuffers für das Framework entfiel. Durch weitere Analyse konnten aber vielversprechende Lösungsansätze entwickelt werden. Die Ergebnisse der Datenmessungen zeigen, dass der Monte Carlo Tree Search Expectimax Agent alle Varianten mit einer Gewinnrate von mindestens 73 % gegen den Zufallsagenten, dominiert.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Künstliche Intelligenz (kurz KI) ist allen Lebensbereichen von zunehmender großer Bedeutung und aus dem Alltag der meisten Mitmenschen nicht mehr herauszudenken. So genügt heutzutage zur Entsperrung des Smartphones ein einfacher Blick in Richtung der Frontkamera. Ein weiteres Beispiel sind soziale Netzwerke, die gezielt Beiträge anzeigen, die wahrscheinlich die höchste Resonanz mit dem Benutzer erzielen. Auch im Spielbereich ist künstliche Intelligenz stark präsent. Algorithmen erzielen Spielstärken, die höher als die von Experten sind. So titelt der SiliconAngle „DeepMind's new MuZero AI develops 'superhuman' chess skills by making plans"Deutscher (2020). Im weiteren Artikelverlauf wird erwähnt, dass MuZero unter anderem in der Lage ist, weitere bekannte Algorithmen in dem Atari57-Test, ein Umgebung bestehend aus 57 weitverbreiteten Atarispielen, die zur Benchmarkspieler von neuronalen Netzen genutzt werden, übertrifft. Das deterministische Spiel Schach ist mit einer Zustandsraum-Komplexität von 10^{50} (Allis et al., 1994, S. 171) viel zu groß als das alle Spielzustände berechnet und gespeichert werden könnten. Hätte ein Agent jedoch die Möglichkeit diese Zustände alle zu berechnen und korrekt zu bewerten, würde dieser Agent perfekt spielen. Eine ähnliche schwere Aufgabe im Game Learning Bereich stellen nichtdeterministischen Spiele dar. Neben der Aufgabe eine geeignete Strategie zu finden, werden Agenten zusätzlich Zufallsereignissen ausgesetzt, die sich außerhalb ihrer Kontrolle befinden und maßgeblich für die Findung einer optimalen Strategie sind. In dieser Arbeit werden KI-Methodiken am Beispiel des nichtdeterministischen Brettspiels „EinStein würfelt nicht!" (EWN) in der Game Learning Umgebung General Game Board (kurz GBG), entwickelt von Wolfgang Konen, untersucht (Konen (2019)). Dabei sollen der Monte Carlo (MC), Monte Carlo Tree Search Expectimax (MCTSE) und der Temporal Difference NTupel (TD-Ntuple) Agent für das Spiel analysiert werden. Erstmalig wird eine Form des Experience Replays in Form eines Replaybuffers für die Lernumgebung implementiert und der TD-Ntuple Agent um diese erweitert. Anschließend sollen die Lernerfolge der benannten Agenten miteinander verglichen werden. Es ergeben sich die folgenden Forschungsfragen im Hinblick auf die Untersuchung:

- Welche messbaren Lernerfolge können durch die Agenten für die zwei und vier Spielervarianten von „EinStein würfelt nicht!" erzielt werden?
- Welche Unterschiede in den Lernerfolgen ergeben sich für die betrachteten verschiedene Spielvarianten von „EinStein würfelt nicht!"?
- Kann im GBG durch Experience Replay bessere messbare Lernerfolge für den TD-Ntuple Agenten realisert werden?

Ausgehnd vom Stand der Forschung wird zuerst das Spiel EWN vorgestellt und die Spielregeln erklärt. Das nächste Kapitel befasst sich mit der Game Learning Umgebung GBG und der Implementierung von EWN und dessen Besonderheiten. Anschließend werden die Agenten und deren Funktionsweisen erläutert. Danach folgt der Aufbau des Replaybuffers und dessen Funktionsweise, sowie ein Funktionstest. Zuletzt werden die

Messergebnisse präsentiert und die aufgetretenen Probleme und Messungen reflektiert und zusammengefasst. Die Rohdaten der Messungen sind in einem Github Repository[1] jederzeit öffentlich zugänglich.

1.1 Stand der Forschung

Stable-Baselines3

Stable-Baselines3 ist eine Open-Source-Software[2] in der Programmiersprache Python, die deep reinforcement learning Algorithmen anbietet. Die Algorithmen folgen strengen Interface-Richtlinien und sind mit der bereitgestellten Dokumentation leicht zu implementieren. 95% der Code-Basis sind mit automatischen Unit Tests abgedeckt (Raffin et al., 2021, S. 1). Unteranderem sind auch Implementationen von Experience Replay bzw. Replaybuffern in der Code-Basis enthalten.

An Agent for EinStein würfelt nicht! Using N-Tuple Networks

Chu et. al. zeigt, dass durch die Nutzung von NTuple Netzwerken in Kombination mit Monte Carlo Learning Gewinnraten in Höhe von 89,10 % gegen einen Zufallsspieler und 42,8 % gegen einen Baseline MCTS Agenten zu erzielen sind (Chu et al., 2017, S. 3). Der MCTS Agent verwednet als Selektionsfunktion UCB (upper-bounds-selection) und wählt den Knoten mit dem höchsten Wert aus gemäß UCB aus (Chu et al., 2017, S. 2). Evaluiert wurde alle 10000 Trainingsspiele gegen einen Zufallsagenten mit 2000 Testspielen. Nach den ersten 10000 Trainingsepisoden erreichte der Agent bereits eine Gewinnrate von mehr als 70 %.

[1] https://github.com/Thanked93/Messdaten
[2] https://github.com/DLR-RM/stable-baselines3

2 „EinStein würfelt nicht!“

Das Brettspiel „EinStein würfelt nicht!“ wurde von Ingo Althöfer 2004 erfunden und im selben Jahr auf der „Games Convention“ in Leipzig veröffentlicht (Althöfer (2011)). EWN entwickelte sich aus einem Fussball-Brettspiel mit Zufallselementen namens „BallaBalla“, welches im Juli 2002 für „Zillions of Games“,während der FIFA-Fussball-WM entworfen wurde.

2.1 Überblick

In diesen Kapitel wird auf die Spielregeln von „EinStein würfelt nicht!“ am Beispiel der klassischen Variante eingegangen. Anschließend wird die vier Spieler Variante vorgestellt und eine Einordnung des Spiels bezüglich der Charakteristiken unternommen.

2.2 Aufbau des Spiels

Das Original „EinStein würfelt nicht!“ ist auf zwei Spieler ausgelegt und wird auf einem quadratischem Spielbrett mit 25 Feldern gespielt. Beide Spieler starten in gegenüberliegenden Ecken und platzieren ihre sechs Spielsteine auf den ersten sechs freien Spielfeldern ausgehend von der Spielbrettecke, wie in Abbildung 1 veranschaulicht. Jeder Spielstein besitzt eine Nummer im Bereich von 1 bis 6. Der erste Spieler würfelt und rückt seinen Spielstein, entsprechend der gewürfelten Augenzahl, um ein Feld in die Richtung einer der anderen Spielbrettecken. Ein Zug in die Richtung der eigenen Spielbrettecke ist ausgeschlossen. Das Schlagen der eigenen und gegenerischen Spielsteine ist erlaubt. Sollte ein Spieler keinen der gewürfelten Augenzahl entsprechenden Spielstein mehr besitzen, muss dieser den nächst höheren oder tieferen Spielstein ziehen. Ein Überschlag von sechs auf eins oder von eins auf sechs ist nicht erlaubt. Sollte der Spielstein mit der Nummerierung drei fehlen und der Spieler besitzt noch alle anderen Spielsteine und würfelt die Augenzahl drei. So kann er sich zwischen dem Spielstein mit der Nummerierung zwei und vier entscheiden. Siegreich ist der Spieler, welcher entweder alle gegenerischen Spielsteine schlägt oder mit einem beliebigen seiner Spielstein in die Spielbrettecke des Gegeners gelangt Althöfer (2004).
Abbildung 1 visualisiert den ersten Zug des weißen Spielers und seine Entscheidungsmöglichkeiten. Der Spieler ist gezwungen den Spielstein mit der Nummer vier zu ziehen. Als Zugmöglichkeit schlägt er entweder einen seiner beiden Spielsteine mit der Nummerierung fünf oder sechs oder zieht auf das freie Spielfeld zwischen beiden Spielsteinen. Schlägt der Spieler den Spielstein mit der Nummer fünf, hat er im nächsten Spielzug bei einem Würfelergebnis von fünf die Möglichkeit, sich zwischen dem Spielstein mit der Nummer vier oder sechs zu entscheiden. Die zweite Abbildung 2 ist das Resultat des Zuges des ersten Spielers seinen Spielstein mit der Nummer fünf zu schlagen. Der schwarze Spieler würfelt anschließend eine eins und zieht den Spielstein mit der Nummer eins.

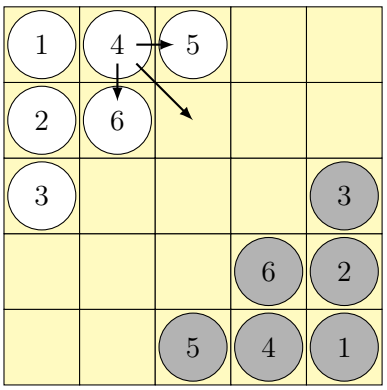


Abbildung 1 Zug 1, weiß würfelt 4

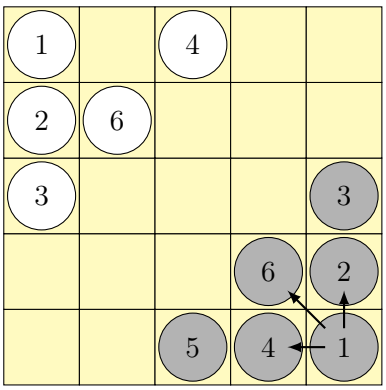


Abbildung 2 Zug 2, schwarz würfelt 1

2.3 4 Spieler Variante

Zusätzlich zu der Standardversion gibt es noch eine drei und vier Spielervariante. In dieser Arbeit werden die zwei und vier Spielervariante von „EinStein würfelt nicht!“ untersucht.

2.3.1 4 Spielervariante

In der 4 Spielervariante bilden zwei Spieler ein Team. Genauso wie in der zwei Spielervariante ist die Siegeskondition, dass entweder alle Spielsteine eines gegenerischen Spielers geschlagen werden oder die gegenüberliegende Ecke mit einem Spielstein erreicht wird. Die restlichen Spielregeln entsprechen der zwei Spieler Variante, so dürfen die eigenen Spielsteine nur in die drei Richtungen der anderen Ecken gezogen werden. Abbildung 3 zeigt eine mögliche Startaufstellung der vier Spielervariante. In der implementierten Form von Ewn bildet im General Board Game Framework der erste Spieler mit dem dritten Spieler und der zweite Spieler mit dem vierten Spieler ein Team. Es wird der Reihenfolge der Startplätze nach gezogen.

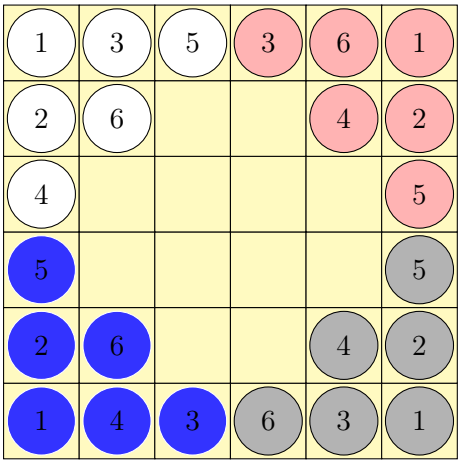


Abbildung 3 Startaufstellung für vier Spieler

2.3.2 Reduzierte Varianten

Jede Variante bietet zusätzlich die Möglichkeit den Suchraum und die Komplexität des Spieles, entweder durch Verkleinerung des Spielbrettes oder Reduzierung der Spielsteine, weiter einzuschränken. Die Abbildungen 4 und 5 stellen mögliche Startaufstellungen der reduzierten Varianten dar.

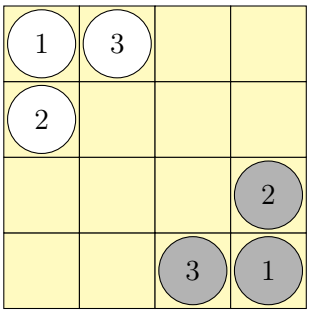


Abbildung 4 Startaufstellung: 2 Spieler 4x4 Spielbrett

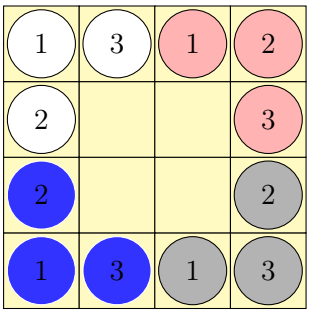


Abbildung 5 Startaufstellung: 4 Spieler 4x4 Spielbrett

2.4 Einordnung des Spiels

Ewn fällt in die Kategorie der nichtdeterministischen Spiele mit vollständigen Informationen. Jeder Spieler hat alle Informationen, die er für die Entscheidung seines Zuges benötigt. Die Eigenschaft des Nichtdeterminismus erhält Ewn durch das Zufallselement des Würfelwurfes. Die Tabelle 1 zeigt die obere Schranke der Spielzustandskomplexität (SSC - State Space Complexity) für die verschiedenen Varianten und der jeweiligen reduzierten Form. Zur Berechnung wird folgende Formel verwendet:

$$SSC \approx \sum_{k=1}^{n} \frac{f!}{(f-k)!}$$

wobei n die Anzahl aller Spielsteine und f die Anzahl der Spielfelder bestimmt. Das Ergebnis inkludiert auch unmöglich zu erreichende Spielzustände und ist deshalb maximal als obere Schranke zu berücksichtigen.

Spielbrettgröße	Spieleranzahl	Spielsteine (gesamt)	SSC
4x4	2	6	6337216
4x4	4	12	1079933185216
5x5	2	12	$\approx 2.68 \cdot 10^{15}$
6x6	4	24	$\approx 8.409 \cdot 10^{32}$

Tabelle 1 Spielzustandskomplexität nach Varianten

2.4.1 Reward - Belohnung

Eine weitere Besonderheit im Spiel von „ EinStein würfelt nicht!“ ist dass es als Belohnung für die Agenten nur zwei Möglichkeiten gibt. Im Falle eines gewonnen Spiels erhält der Agent eine positive Belohnung $+1$ und im Falle eines verlorenen Spiels eine negative Belohnung oder Bestrafung mit dem Wert -1. Aufgrund der mehreren Gewinnbedinungen und dem nicht aussetzen eines Zuges, fällt die Bewertung von 0, die bspw. im Spiel von „Tic Tac Toe“ bei einem Unentschieden vorhanden ist weg.

3 General Board Game

Das General Board Game Framework (kurz GBG) ist eine Open-Source-Software[1] und wurde von Prof. Dr. Wolfgang Konen in Java entwickelt. Es bietet Studierenden die Möglichkeit, sich erstmals mit dem Thema Game Learning auseinanderzusetzen. Ein weiterer Punkt ist es Forschenden ein Framework zu bieten durch das verschiedene KI-Agenten im Wettbewerb stehen, um die Forschung für solche Agenten voranzutreiben (Konen, 2019, S. 1). Die bereitgestellten Agenten im GBG abstrahieren ihre Funktionsweise, sodass sie auf mehrere verschiedene Spiele anwendbar sind. Eine Einführung einer neuen Spielumgebung in das GBG ist im technischen Report[2] genaustens beschrieben. Das Spiel Ewn wurde im Rahmen des Praxisprojekts im GBG implementiert.

3.1 Überblick

In diesem Kapitel wird auf die wichtigsten verschiedenen Klassen des GBG Frameworks eingegangen und die verschiedenen Optionsmöglichkeiten, speziell für EWN, erläutert.

3.2 Die Klassen und Interfaces des GBG

Das GBG stellt eine Vielzahl von Klassen zur Verfügung. Die bereits über die Methoden mit dem GBG Kern kommunizieren.

ObserverBase - Abtrakte Klasse und StateObsNondeterministic - Interface

Die Observerbase bildet einen bestimmten Spielzustand ab und dient dazu alle notwendigen Informationen für diesen in einem Objekt zu speichern beispielsweise alle möglichen anwendbaren Aktionen, die einen Spielzustand in einen Folgezustand erweitern oder die Informationen der einzelnen Spielsteine auf einem Spielbrett. Das StateObsNondeterministic-Interface ist eine Erweiterung für die abstrakte Klasse ObserverBase und bietet eine tiefere Aufteilung der *advance*-Methode in einen determinisitschen Teil, der von außen durch eine Aktion beeinflussbar ist, und einen nichtdeterminisitschen, der intern abgewickelt wird und sich dem Einfluss von Spielern entzieht.

Evaluator - Interface

Durch den Evaluator werden die Trainingsschritte eines KI-Agenten erst messbar. Vergleichsagenten, die im Idealfall eine feste Spielstärke besitzten, evaluieren die Spielstärke des zu trainierenden Agenten. Es können die verschiedenen Startpositionen des Spieles, sowie die Anzahl der getätigten Evaluationen vor Trainingsbeginn bestimmt werden. Das Resultat wird in Form eines Liniendiagramms meistens im Wertebereich -1 und 1 dargestellt. Diese Evaluation hat aber keinen Einfluss auf die Entwicklung

[1] Der Quellcode kann unter https://github.com/WolfgangKonen/GBG eingesehen werden
[2] Abrufbar unter https://www.gm.fh-koeln.de/ciopwebpub/Konen22a.d/TR-GBG.pdf

der Spielstärke des Agenten und dient rein der Messbarkeit der Spielstärke. Es können mehrere Agenten als Vergleichsagent vordefiniert werden.

Gameboard - Interface

Das Gameboard Interface koppelt die grafische Nutzeroberfläche an die Kernfunktionen des GBG und muss spielspezifisch implementiert werden. Dabei wird durch das Interface auf jede Änderung des Stateobserver geachtet und kann dann visuell repräsentiert werden.

Arena - Abtrakte Klasse

Diese Klasse erlaubt die Einstellungsmöglichkeiten für Agenten und die Agentenauswahl und hält Referenzen zur grafischen Nutzeroberfläche und erzeugt alle benötigten Objekte.

3.3 Besonderheiten im Spiel EWN

Die Klasse des Stateobervers wird weiter in die Subklassen *Player* und *Token* unterteilt. Die *Player*-Klasse verwaltet die Spielsteine eines Spielers und hilft den oder die korrekten Spielsteine (Token) auszuwählen. Zusätzlich bietet diese Klasse alle benötigten Funktionalitäten an um die Spielsteine zu entfernen, hinzuzufügen und zu aktualisieren. Die Klasse *Token* kennt den zugehörigen Spieler und beinhaltet die Logik zu Berechnung eines erlaubten Spielzuges. Das Klassendiagramm veranschaulicht die Beziehung und setzt sie in den entsprechenden Kontext zum GBG.

Settings

Durch diese Option können die verschiedenen Spielvarianten von Ewn ausgewählt und gespielt werden. Es stehen insgesamt die folgenden sechs verschiedenen Varianten bereit:

- 3x3 Spielbrett für zwei Spieler mit jeweils 3 Spielsteinen je Spieler
- 4x4 Spielbrett für zwei Spieler mit jeweils 3 Spielsteinen je Spieler
- 5x5 Spielbrett für zwei Spieler mit jeweils 6 Spielsteinen je Spieler
- 6x6 Spielbrett für drei Spieler mit jeweils 6 Spielsteinen je Spieler
- 4x4 Spielbrett für vier Spieler mit jeweils 3 Spielsteinen je Spieler
- 6x6 Spielbrett für vier Spieler mit jeweils 6 Spielsteinen je Spieler

Positionswerte

Die Positionswerte bilden für den TD-NTuple Agenten eine Form, die verschiedenen Spielsteine genauer zu unterteilen und helfen durch die spezifischere Betrachtungsweise die Spielstärke zu erhöhen.

- $[0, ..., n]$ unterscheidet nur zwischen der Spielerzugehörigtkeit und den leeren Feldern.
- $[[0, 1], [2, 3], [4, 5]]$ unterscheidet zwischen der Spielzugehörigkeit, den leeren Spielfeldern und gruppiert die Spielsteine eines Spielers in drei Gruppen.
- $[[0], [1], [2][3], [4], [5]]$ unterscheidet zwischen der Spielerzugehörigkeit, den leeren Spielfeldern und gruppiert jeden nummerischen Wert eines Spielsteins.

Random starting

Die Option *Random Starting* kann die Werte *true* oder *false* annehmen und deint dazu bei jedem Spielbeginn eine neue Startposition für die Spielsteine zu wählen. Dadurch abstrahiert ein zu trainierender Agent mehrere Startaufstellungen und ist nicht nur auf eine Startaufstellung begrenzt.

4 Agenten im GBG

Das GBG Framework bietet bereits eine Vielzahl von stark abstrahierten Agenten an, die in den unterschiedlichsten Spielen anwendbar sind. Die Gemeinsamkeit von allen Agenten ist, dass Sie das Interface *Playagent* implementieren, wordurch sie Zugriff auf die Methode *getNextAction2* erhalten (Konen, 2020, S. 11). Die Methode *getNextAction2* erlaubt es allen Agenten für einen gewissen Spielzustand die bestmögliche Aktion auszuwählen und den Spielzustand damit zu erweitern.

4.1 Überblick

In diesem Kapitel wird auf die Funktionsweise der in dieser Arbeit verwendeten Agenten eingegangen und erläutert. Neben dem Zufallsagenten (Randomagent) und dem Monte Carlo Agenten stehen besonders die Reinforcement Learning Agenten Monte Carlo Tree Search Expectimax (MCTSE) und der Temporal Difference N-Tuple (TD-Ntuple) im Fokus dieser Arbeit.

4.2 Randomagent

Der Randomagent basiert auf einer zufälligen Auswahl einer verfügbaren Aktion a aus der Aktionsmenge A im Spielzustand s zum Zeitpunkt t. Während des Agenten Trainings und der Parametersuche wird dieser Agent aufgrund der schnellen Aktionsbestimmung als Gegenspieler ausgesucht. Bei einer genügend hohen Anzahl von Spielen, sieht man, dass der Agent eine Gewinnrate gegen sich selbst von 50 % erzielt, sofern alle Startbedingungen gleich sind.

4.3 Reinforcement Learning

Im Reinforcement Learning (Bestärkendes Lernen) interagieren Agenten durch Aktionen mit ihrer Umgebung. Anhand von ausgeschütteten Belohnung oder Bestrafung der Umgebung ist der Agent in der Lage sein Verhalten anzupassen. Das Ziel des Agenten ist es die Belohnung zu maximieren und die Bestrafungen zu minimieren. Es wird auf Monte Carlo Tree Search Expectimax und den Temporal Difference Agenten und deren Funktionsweisen eingegangen. Der Unterschied zwischen den Agenten liegt in der Strategie, wie sie die Belohnungen maximieren.

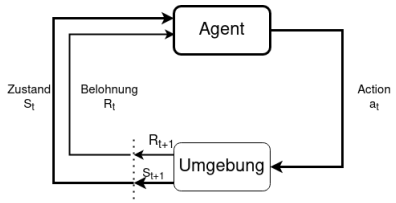


Abbildung 6 Reinforcement Learning (Sutton and Barto, 2018, S. 48 modifiziert)

4.3.1 Monte Carlo

Der Monte Carlo (MC) Agent basiert bestimmt seine Aktionen basiernd auf einer Monte-Carlo-Simulation. Bei einer hohen Spielmenge ist der Agent in der Lage, seine Strategie basiernd auf den Zufallszügen zu ermitteln.
Dafür übernimmt der Agenten den Spielzustand s und überführt diesen mit jeder verfügbaren Aktion in seinen Folgezustand s'. Der Folgezustand wird anschließend durch zufällig gewählte Aktionen bis zum finalen Zustand oder bis zur Beschränkung der Rollout Depth erweitert. Durch die Evalutation der erhaltenen Messungen, erkennt der Agent welche Aktion an die Spielumgebung als beste Aktion wiedergegeben werden soll (Kutsch, 2017, S. 8).

4.4 Monte Carlo Tree Search Expectimax

Der Monte Carlo Tree Search Expectimax (MCTSE) Agent basiert auf dem Monte Carlo Tree Search Agenten und löst das Problem, das der normale MCTS Agent mit nichtdeterministischen Spielen hat, die Zufallsereignisse im Baum zu repräsentieren. Der Agent wurde von Johannes Kutsch für das Spiel 2048 im GBG erstellt. Der klassische MCTS Agent traversiert in einer Iteration pro möglicher Aktion nur einen Folgezustand und verwirft somit weitere nichtdeterminisitschen Ereignisse, die auftreten könnten. Das Problem wird durch die Einführung einer neuen Knotenart, den Chanceknoten behoben. Dieser Knotentyp ist der fehlende Zwischenschritt, um die nichtdeterministische Ereignisse darzustellen. So können jedem Expectimaxknoten alle Chancenknoten angebunden werden, die eine anderes Zufallsereignis abbilden. Im Falle von EWN sind das die verschiedenen Ergebnisse des Würfelwurfs. Dabei durchläuft der Agent genau wie der normale MCTS Agent die vier Phasen Selektion, Expansion, Simulation und Backpropagation. Abbildung stellt die vier Phasen schematisch für dem normalen MCTS Agenten 7 dar. Nachstehend sind die einzelnen Schritte der Phasen genauer erklärt.

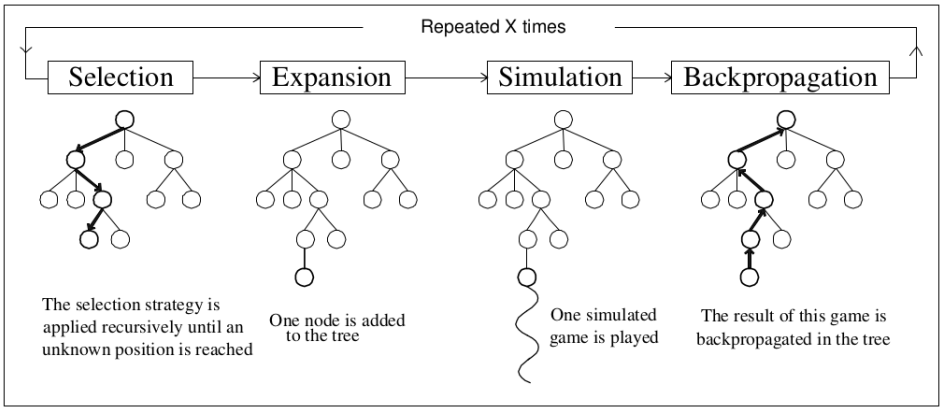


Abbildung 7 Phasen der Monte Carlo Tree Search, (Chaslot, 2010, S. 18)

Selektion Während der Selektionsphase sucht der Agent einen Chanceknoten für die Expansionsphase anhand einer bestimmten Funktion aus. Dazu wird jede verfüg-

bare Aktion in Form eines Expectimaxknoten[1] and den Chanceknoten angehangen. Sollte sich eine Aktion finden, die noch keinen Expectimaxknoten hat, so wird in die Expansionsphase übergegangen (Kutsch, 2017, S. 40). Sobald allen Aktionen ein Expectimaxknoten angehangen wurde, wird anhand der Selektionsfunktion ein Knoten ausgesucht und anschließend die Aktion innerhalb dieses Knoten auf den Zustand angewendet. Für den Folgezustand s' wird anschließend ein neuer Chanceknoten angehangen. Die Selektionsphase startet erneut. Die Selektionsfunktion *Upper Confidence Bound for Trees* (kurz UCT) ist wie folgt definiert und wählt den nächsten Knoten anhand des höchsten Wertes aus:

$$UCT = \bar{X}_c + K * \sqrt{\frac{ln(n+1)}{n_c} + r}$$

Dabei ist $\bar{X}_c$ der Mittelwert aller Belohnungen des Kindknotens c. K die Explorationsrate des MCTSE Agenten. n ist die Besuchsanzahl des Vaterknotens und n_c dem entsprechend die Besuchsanzahl des Kindknotens. r ist ein sehr kleiner Wert, der lediglich eine Entscheidungshilfe bei Gleichstand zweier Knoten ist.

Expansion Solange die maximale Baumtiefe noch nicht erreicht ist und ein Chanceknoten eine noch nicht erweiterte Aktion besitzt, wird für die Aktion a ein neuer Expectimaxknoten an den ausgewählten Chanceknoten angehangen. Die Verbindung beider Knoten erstellt wiederum einen neuen Chanceknoten, der der Folgezustand s' ist. Also die Erweiterung aus der Verbindung des ausgewählten Chanceknotens mit dem Zustand s durch die Aktion a des Expectimaxknotens.

Simulation Während der Simulationphase, werden zufällig gewählte Aktionen auf den in der Expansionsphase ausgewählten Knoten angewendet bis die Rolloutdepth oder ein finaler Spielzustand erreicht sind (Chaslot, 2010, S. 22).

Backpropagation In der Phase der Backpropgation traversiert der Agent alle Vorgängerknoten und addiert den Reward des Endknotens zu diesen. Zudem, werden verschiedene Werte, wie die Anzahl der Besuche aktualisiert (Chaslot, 2010, S. 23).
Die Wiederholungsanzahl für einen Durchlauf lässt sich im GBG über den Parameter *Iterations* bestimmen. Die maximale Baumtiefe, sowie die maximale Anzahl an Expectimaxknoten, kann über die Parameter *Treedepth* und *MaxNodes* eingestellt werden. Die Simulationstiefe wird über den Parameter *Rolloutdepth* gesetzt. Die Explorationsrate entspricht dem Wert von *K(UCT)*.
Die Abbildung 8 zeigt eine schematische Abbildung des MCTSE-Suchbaums für das Spiel EWN.

[1] Es wird sich auf die Verwendung des Begriffs Expectimaxknotens auch für die Verwendung von Expectiminknoten beschränkt

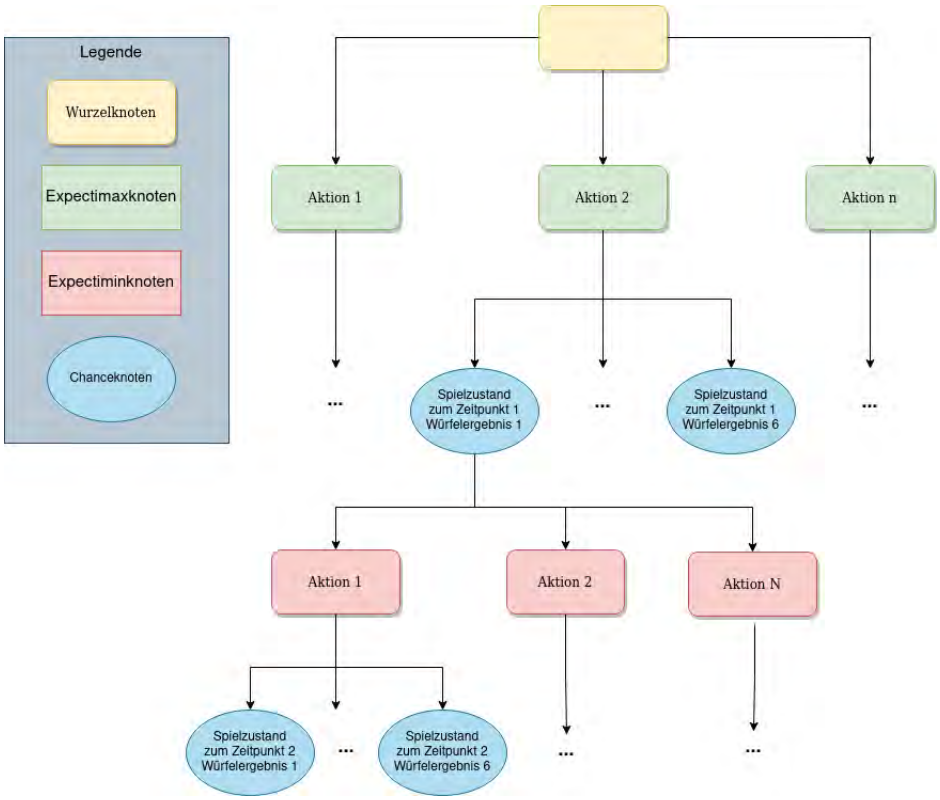


Abbildung 8 Schematische Darstellung des MCTSE-Suchbaums für EWN:

4.5 Temporal Difference

Temporal Difference Learning ist eine Sonderform des bestärkenden Lernens (Reinforcement Learning). Alle nichtterminalen Spielzustände innerhalb einer Epsiode beeinflussen die zukünftige Belohnung oder Bestrafung, die ein Agent im finalen Zustand erhält. Deshalb soll der Agent, die Spielzustände so bewerten können, dass eine positive Belohnung ausgeschüttet wird. Dazu soll der Agent alle zukünftigen Belohnungen abschätzen und diese Abschätzung zu einem späteren Zeitpunkt korrigieren. Diese Korrektur ist der Lernprozess des Agenten. Im Trainingsprozess spielt der Agent gegen sich selbst.

4.5.1 Funktionsweise

Der TD-Agent verwendet eine Wertefunktion (Value Function) $V(s_t)$ zum bewerten eines Spielzustandes s zum Zeitpunkt t. Mit Hilfe dieser Funktion bestimmt der Agent dann die zukünftigen Belohnungen für den Spielzustand s_t. Meistens erhält der Agent aber erst nachdem ein finaler Spielzustand erreicht wurde eine Belohnung oder Bestrafung aus der er Schlüsse ziehen kann. Das stellt ein Problem dar, denn eine Bewertung des jetzigen Spielzustands wirkt sich auf den weiteren Spielverlauf und die Entscheidungen, die der Agent trifft aus. Damit der Agent zum Zeitpunkt t rückwirkend in der Lage ist zu merken, ob er einen Fehler getätigt hat, wird wie folgt ein Fehlersignal für $V(s_t)$ definiert:

$$\delta_t = r(s_{t+1}) + \gamma V(s_{t+1}) - V(s_t)$$

Dabei wartet der Algorithmus den nächsten Zustand s_{t+1} ab und prüft, ob für diesen Zustand bereis ein Wert der Value Function oder die Belohnung/Bestrafung bekannt ist. Ist dies der Fall kann der Wert für $V(s_t)$ angepasst werden und das Fehlersignal korrigiert werden. Ansonsten wird auf den Spielzustand s_{t+2} gewartet und erneut geprüft. Das Fehlersignal für $V(s_t)$ verschwindet sobald $r(s_{t+1}) + \gamma V(s_{t+1})$ erreicht wird. Dadurch kann eine Belohnung oder Bestrafung auch vorherige Spielentscheidungen beeinflussen (Konen, 2015, S. 4).
Ein weiteres Problem bildet für viele Spiele die hohe Anzahl an Spielzuständen, die je nach Spiel weder alle gespeichert noch während des Trainings besucht werden können. Es wird versucht dieses Problem durch eine Näherungsfunktion $f(w; s_t)$ mit freiem Parameter w für Gewichte (weights) zu lösen (Sutton and Barto, 2018, S. 198), sodass $V(s_t) = f(w; s_t)$ gilt . Dadurch wirkt sich eine Änderung in w, die das Fehlersignal verkleinert auf alle anderen Spielzustände aus.Die Auswirkung der Änderung für den Spielzustand s_t auf einen ähnlichen, auch unbekannten, Spielzustand s_u wird als Generalisierung bezeichnet und stellt einen Wissenstransfer dar (Konen, 2015, S. 4).
Zusätzlich kann ein Spielzustand s_t auch nach den Merkmalen sogenannten „Features“ $g(s_t)$ gerneralisiert werden. Diese Merkmale können dem Agenten in Form von N-Tupeln bereitgestellt werden. Der in dieser Arbeit verwendete TD-Ntuple Agent benutzt solche Merkmale.

4.5.2 N-Tupel

Woody Bledsoue und Iben Browning entwickelten in der 1950er Jahren das Konzept der N-Tupel, um Muster maschinell zu erkennen und zu kategorisieren (Bledsoe and Browning, 1959, S. 225). N-Tuple sind verkettete Punkte auf einem Spielbrett welche entweder vordefiniert oder zur Laufzeit erstellt werden können. Durch mehrere Tupel bilden sich Netzwerke durch die ein Spielzustand bewertet wird. Dafür werden die verschiedenen Tupelbewertungen in einer Lookup Tabelle gespeichert und können dann in Kombination mit Temporal Difference angepasst werden. Das GBG Framework bietet verschiedene Möglichkeiten solche N-Tupel-Netzwerke zu erstellen.

Random Walk Im Random Walk werden prozedural zur Laufzeit N-Tupel gebildet. Es erfolgt eine zufällige Auswahl eines Spielbrettpunktes, welches anschließend um einen Nachbarpunkt erweitert wird. Dieses Vorgehen wiederholt sich bis eine bestimmte Anzahl an Punkten dem Tuple hinzugefügt wurden und eine bestimmte Anzahl an Tuplen gefunden wurde. Ein Tupel kann dabei jeden Spielbrettpunkt einmal beinhalten.

Random Point Dieser Algortihmus wählt zufällig Punkte auf dem Spielbrett aus. Dadurch muss kein Spielwissen in der Konstruktion der Punkte vorgegeben werden. Jedoch ist es so schwer, lokale Muster für die Merkmale zu bilden. Im Gegensatz zu dem Algorithmus „Random Walk“ werden die Punkte nicht in eine Nachbarschaftsbeziehung gesetzt.

fixed mode Der fixed mode ermöglicht es dem Entwickler festgesetzte Spielbrettpunkte vorab zu definieren. Dadurch wird gewährleistet, dass bereits identifizierte spielrelevante Bereiche immer betrachtet werden. So bilden in bspw. Othello die Spielbrettecken sehr wichtige Spielfelder, da eine Rückeroberung dieser Spielfelder ausgeschlossen ist. Dennoch verliert der Agent durch die festgesetze N-Tupel-Architektur die Fähigkeit neue wichtigere Spielpunktverbindungen zu erkennen und sich dann verstärkt auf diese zu fokusieren.

4.5.3 N-Tupel Netzwerk

Der Hauptkern dieser Arbeit beschäftigt sich mit dem fixed Mode, so wird das Spielbrett vorab in festgesetzte Bereich eingeteilt und als Merkmal für das Spiel bewertet. Chu hat dafür ein N-Tupel Netzwerk, bestehend aus 24 · 6-er Tupeln für die klassische Version von EWN, vorgeschlagen (Chu et al., 2017, S. 187). Dabei wird jedes Tupel bis bis zum Spielfeldrand nach rechts oder unten verschoben. Die Abbildungen 9, 10 und 11 zeigen die verwendete N-Tupel-Architektur

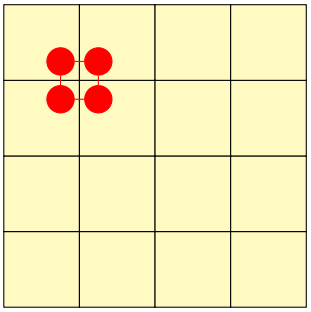


Abbildung 9 N-Tupel Architektur: 9 · 4-Tupel

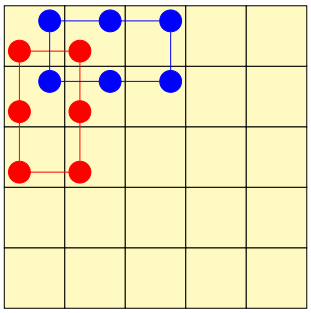


Abbildung 10 N-Tupel Architektur: 24 · 6-Tupel, Chu et al. (2017) S. 187, modifiziert

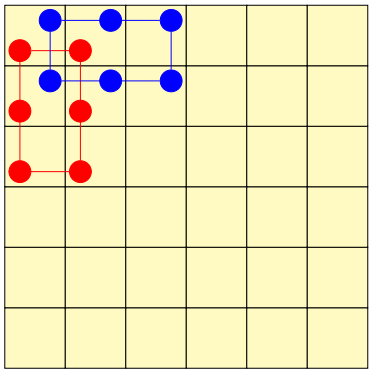


Abbildung 11 N-Tupel Architektur: 40 · 6-Tupel

Anzahl an Gewichtungen Die Kombination der N-Tupel Architektur mit den Positionswerten bilden die Anzahl der Gewichtungen nach folgender Formel

$Gewichtungen = Tupelanzahl \cdot Positionwerte_{gesamt}^{Tupelgre}$
Die Tabelle 2 für die zwei und vier Spielervariante an. In der zwei Spielervariante wird sich auf die 13 Positionwerte bezogen. Für die 4 Spielervariante wird sich aufgrund der hohen Anzahl an Gewichtungen auf 5 Positionswerte bezogen.

Variante	Tupelanzahl	Tupelgröße	Positionswerte$_{gesamt}$	Gewichtungen
4x4 2 Spieler	9	4	13	257049
4x4 4 Spieler	9	4	5	5625
5x5 2Spieler	24	6	13	115843416
6x6 4 Spieler	40	6	5	625000

Tabelle 2 Gewichtungen nach Varianten

4.5.4 Parameter

Der Agent lässt sich über mehrere Parameter justieren.

Alpha Die Lernrate *alpha* ist eine Variable, die die Veränderung der Wertefunktion beeinflusst und gibt an, wie hoch die Schrittweite für den Gradienten ist um die Fehlerfunktion in Richtung eines Minimums anzunähern.

Epsilon Epsilon bestimmt die Explorationsrate des Agenten und bietet eine Wahrscheinlichkeit mit der zufällige Aktionen ausgewählt werden, durch die eine Chance besteht, dass der Agent einen unbekannten Zustand ausgesetzt ist. Dadurch wird der Agent gezwungen während seines Trainings immer wieder neue Herausforderungen zu bewältigen. Je öfter der Agent verschiedene Zustände gesehen hat, desto wahrscheinlicher wird er die Spielfunktion anpassen und kann daraus lernen.

Gamma Der Wert Gamma bestimmt wie vorausschauend ein Agent seine Aktionen wählt. Ist der Wert gering, so wählt der Agent Aktionen aus, die innerhalb kurzer Zeit das beste Ergebnis darstellen. Im Gegensatz dazu liefert ein Wert von 1, dass der Agent die Aktion wählt, welche in dem besten Endzustand endet. Gamma wird auch als Discount-Faktor bezeichnet.

Lambda Der Parameter Lamba bestimmt die Zerfallsrate der *eligibility traces*. Ein Wert von 1 kommt dem Verhalten des MCTS Agenten nahe und versucht die Belohnung oder Bestrafung des finalen Spielzustandes auf alle vorherigen Spielzustände anzuwenden. Ein Wert von 0 hingegen verhindert den Einfluss der Belohnungen auf die vorherigen Spielzustände.

5 Experience Replay

Experience Replay wurde von Lin 1992 für den die reinforcement learning architecture AHC (adaptive heuristic critic) eingeführt (Lin (1992)). Das Konzept ist eine Transition, bestehend aus einem Zustand, einer angewandeten Aktion, einer Belohnung und dem Folgezustand, zu speichern, um Sie einem Agenten zu einem späteren Zeitpunkt wieder zu Trainingszwecken vorzusetzen. Implementiert wird der Replaybuffer zu testzwecken erstmals für den TD-Ntuple Agenten. Ziel ist es, dass durch das wiederholte Training mit einer Transition zu unterschiedlichen Zeitpunkten der Generalisierungseffekt verstärkt wird und durch die erhöhte Anzahl an Trainingszuständen die Wertefunktion schneller ein Optimum erreicht.

5.1 Überblick

Dieses Kapitel befasst sich mit den Details des Replaybuffers im GBG. Es wird zuerst auf die genauen Ziele des Replaybuffers eingegangen und anschließend der Aufbau im GBG Framework erläutert. Danach wird der Entwicklungsverlauf eines Agenten mit verschiedener Parametrisierung für das deterministische Spiel Othello[3] untersucht.

5.2 Ziel

Das wichtigste Ziel ist es, dass der Agent durch die Speicherung von Transitionen bereits erlebte Erfahrungen (Transitionen) beliebig oft wiederholen kann und dadurch verschiedene Trainingsziele in Form von Benchmarks erzielt werden können. Ein zusätzlicher Effekt ist dass die temporale Korrelarität der aufeinanderfolgenden Spielzustände gebrochen wird, da der Agent keine vollständigen Episoden in aufeinanderfolgenden Zeitpunkten, sondern nach einer bestimmten Methodik aus dem Transitionsbestand zu Trainingszwecken erhält (Lin, 1992, S. 314). Zusätzlich vermutet, dass durch die erhöhte Anzahl an Traininszuständen, ein Agent schneller eine gewisse Spielstärke erreicht.

5.3 Aufbau

Der Replaybuffer im GBG Framework besteht aus 4 Komponenten. Durch deren Beziehungen während des Trainings der Replaybuffe gefüllt. Dabei werden die gespeicherten Transitionen über eine Schnittstelle wieder zur Verfügung gestellt werden. Der Grundgedanke ist, dass ein *Pusher* die Zuständigkeit übernimmt, welche Transition überhaupt speicherbar ist. Der *BaseSelektor* während des Trainings entscheided, welche Transition aus dem Buffer für Trainingszwecke genutzt werden soll. Der *BaseBuffer* ist die Komponente, welche die Verwaltung über die Transitionen übernimmt und eine *Transition* das zu speichernde Objekt ist, welches den Zustand darstellt, durch den ein Spielzustand in den nächsten gewandelt wird. Über die Methode *void addTransition(ITransition t)* und *ITransition[] selectBatch()* soll jeder Agent in der Lage sein, den Replaybuffer zu implementieren.

[3] Die Spielregeln sind unter https://www.spielregeln.de/othello.html abrufbar

Die Integration in den TD-Ntuple Agenten wird im nachstehenden Pseudocode erklärt.

.

vlined 1 TD-FARL-Episode-using-Replaybuffer. (Konen and Bagheri, 2021, S. 7) modifiziert. Starting from state s_0, r_t is the delta reward for p_t when taking action a_t in state s_t We connect afterstate s' via player p_t with the previous afterstate $slast[p_t]$ of this player. Note that s_tast and r are vectors of length N .

```
1:  function TD-FARL-Episode-using-Replaybuffer(s_o)
2:      t ← 0
3:      s_last[p] ← null ∀ player p = 0,...,N − 1                    ▷ last afterstates
4:      repeat
5:          p_t = player to move in state s_t
6:          Choose Action a_t from s_t using policy derived from V    ▷ e.g. ϵ-greedy
7:                  ▷ r is the delta reward tuple form the perspective of all players p
8:          (r,s',s'') ← MakeAciton(s_t,a_t)                  ▷ s': afterstate (after taking a_t)
9:          if replaybuffer then
10:             AdaptAgentV2(s_last[p_t], r[p_t], s')
11:         else
12:             AdaptAgentV(s_last[p_t], r[p_t], s')
13:         end if
14:         s_last[p_t] ← s'        ▷ the afterstate generated by p_t when taking action a_t
15:         t ← t+1
16:         s_t ← s''
17:     until (s'' is terminal)
18:     if replaybuffer then
19:         FinalAdaptAgents2(p_t,r,s')
20:     else
21:         FinalAdaptAgents(p_t,r,s')
22:     end if
23: end function
24:
25: function AdaptAgentV2(p_t, r', s')
26:     if s_last[p_t] ≠ null then
27:         addTransition(p_t,s',s_last[p_t],rLast,r,0)
28:                                   ▷ 0 implements state from perspective of p_t
29:         learnFromReplayBuffer()
30:     end if
31: end function
32:
33: function FinalAdaptAgents2([p_t], r, s')
34:     for p = 0,...,N-1, but p ≠ p_t do
35:         if (s_last[p_t] ≠ null) then
36:             addTransition(p,s',s_last[p],rLast,r,1)
37:                               ▷ 1 implements terminal state from perspective of p
38:             learnFromReplayBuffer()
39:         end if
40:     end for
41:     addTransition(p_t,s',s'',rLast,r,2)
42:                               ▷ 2 implements terminal state from perspective of p_t
43:     learnFromReplayBuffer()
44: end function
45:
46: function learnFromReplayBuffer
47:     if !replaybuffer.isEmpty() then
48:         Transition[] batch ← replaybuffer.selectBatch()
49:         for Transition transition of batch do
50:             Target T ← calculate Target t using transition
51:             s' ← get s' from transition
52:             Use NN to get V(s')
53:             Adapt NN by backpropgating error δ = T − V(s')
54:         end for
55:     end if
56: end function
57:
```

Transition Eine Transition implementiert das Interface *ITransition*, indem die benötigten Methodenaufrufe vordefiniert sind. Die einzelnen Attribute der Transition sind aus dem Funktionsaufruf *updateNet* aus der Wertefunktionsklasse der TD-NTuple4 Klasse abgeleitet. Die Update funktion benötigt eine Instanz der Klasse *StateObsWithBoardVector*, die aus dem Spielzustand s' (Afterstate) der Vorrunde von dem Spieler und einem Bordvektor besteht, der dem der N-Tupel-Funktion Repräsentation entspricht. Also ein Vektor bestehend aus den gemappten Positionswerten eines Spielzustandes. Einen Integerwert *curPlayer*, der den aktuellen Spieler repräsentiert und die zwei double Werte *vLast* und *target*, wobei *vLast* die Abschätzung der Belohnung durch die Wertefunktion des Spielzustandes ist und *target* dem delta reward von der Bewertung des letzten Spielzustandes. aus dem der Spieler gekommen ist entpricht. Die Transitionklasse hingegen besteht aus dem akutellen Spieler *player*, einem *Next-*

state4 Objekt, welches dem letzten Spielzustand des Spielers nach dem tätigen der Aktion entspricht. *sLast* der vorherige Spielzustand von *Nextstate4*. Der letzten Belohnung der Spielers *player* und einem Integerwert *isFinalTransition*, der eine angibt, wie *target* gebildet werden muss.

BaseBuffer Die Klasse des Basebuffer implementiert die Verwaltung der Transisitonen und erzeugt die zwei Hilfsklassen, die das Interface ISelector und IPusher implementieren. Die Transitionen werden in Form eines Arrays mit einer festdefinierten Größe festgelegt. Über Zeiger wird der aktuelle maximale Bestand der Transition, sowie der nächste verfügbare Speicherplatz beobachtet.Sobald das Array voll gelaufen ist, wird der Zeiger für den freien Platz auf 0 zurückgesetzt. Dies bewirkt, dass der Buffer immer bezogen auf die Kapazität neue Transitionen zur Verfügung stellen kann. Über den Methodenauruf *addTransition()* kann eine Transition dem Bestand hinzugefügt und über die Methode *selectBatch()* eine anhand der Vorgehensweise für des Selektors eine bestimmte Anzahl an Transitionen zur Verfügung gestellt werden.

Abstrakte Klasse Baseselektor und Randomselektor Die Klasse des Baseselectors implementiert die benötigte Funktionalität, die einer Erweiterung der Klasse benötigt. So hat diese Zugriff auf den Zeiger, der den akutellen Bestand des Buffers ausgibt, sowie auf die vorher definierte Batchgröße. Über die abtrakte Methode *public ITransition[] selectBatch()* können die Erweiterungen dieser Klasse nach einer beliebigen Vorgehensweise Transitionen aus dem Bestand selektieren. Die Randomselector Klasse überschreibt diese Methode mit der Vorgehensweise zufällig Transtiionen aus dem Bestand zu wählen und in Form eines Array der BaseBuffer Klasse zur Verfügung zu stellen.

BasePusher Der Basepusher implementiert das Interface IPusher und dient bevor eine Transition gespeichert wird, zu überprüfen ob diese Transition nützlich ist. Dazu überschreibt diese Klasse die Methode *boolean pushTransition(ITransition t)* und signalisiert über einen boolean Wert, ob die Transition brauchbar ist. Der BasePusher stuft vorerst alle Transitionen als nützlich ein.

5.4 Othello

Die beschriebene Implementation wird für erstmals für das deterministische Spiel Othello verwendet. Die Buffergröße wird bei 5000 fest gesetzt. Die Batchgröße wird mit 5, 25 und 50 getestet. Als Vergleich wird ein Training ohne Replaybuffer ausgeführt. Jede Messung trainiert 10 Agenten mit jeweils 100000 Epsioden. Der hellblaue Bereich in den Abbildungen 12, 13, 14 und 15 bestimmt die Standardabweichung. Der Agent wird mit den Parametern wie folgt initialisert: $\alpha_{start} = 0.2$, α_{final}=0.1, $\epsilon_{start} = 0.1$, ϵ_{final}=0.1, $\gamma = 1.0$, $\lambda = 0$
Die Abbildungen mit höherer Batchgröße zeigen einen starken Abwärtstrend bei steigender Batchgröße. Wohingegen bei kleinerer Batchgröße der Lernkurvenvergleich fast gleich ausfällt. Die beste durchschnittliche Spielstärke erreichen die Agenten ohne Re-

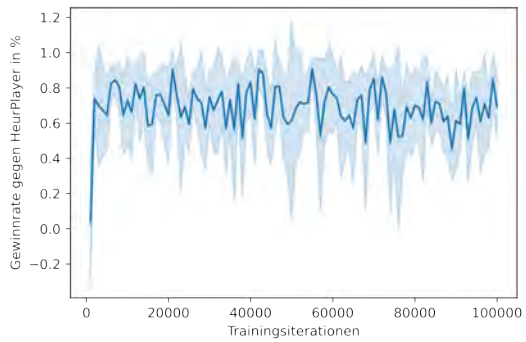


Abbildung 12 Othello ohne Replaybuffer

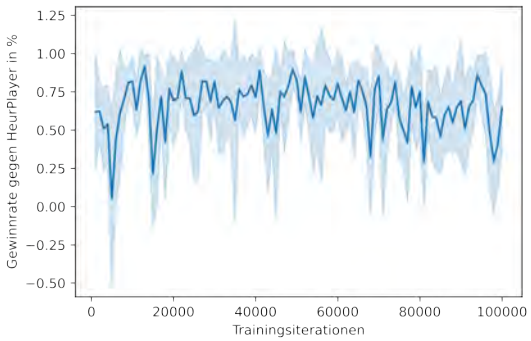


Abbildung 13 Othello mit Replaybuffer und einer Batchgröße von 5

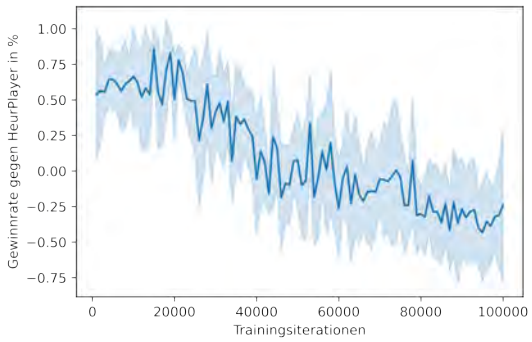


Abbildung 14 Othello mit Replaybuffer und einer Batchgröße von 25

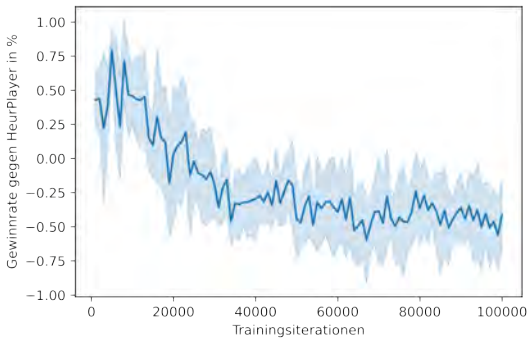


Abbildung 15 Othello mit Replaybuffer und einer Batchgröße von 50

playbuffer. Ein Fehlverhalten des Replaybuffers wird nicht ausgeschlossen. Aufgrund des ausbleibenden Lernerfolges wird der Replaybuffer und seine Eigenschaften nicht weiter für das Spiel „EinStein würfelt nicht!“ untersucht.

5.5 Probleme und Lösungsansätze

Die schlechte Performance des Replaybuffers kann, neben Overfitting, auch ein Hinweis dafür sein, dass die gesammelten Transitionen durch zufälliges selektieren zwar den Zweck erfüllen, die temporale Korrelation zu brechen, aber der Agent keinen großen Nutzen aus den Transitionen ziehen kann. Ein möglicher Lösungansatz ist es , die Transitionen innerhalb des Replaybuffers anhand des aktuellen Fehlersignals zu bewerten. Dadurch könnten die Transitionen mit hohem Nutzen öfters an den Agent übergeben werden. Schaul et. al. zeigt, dass durch eine stochastische Sampling Methode genau dieses Problem gelöst werden kann (Schaul et al., 2015, S. 4). Dafür wird die Priorisierungsfunktion wie folgt definiert:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

wobei $P(i)$, die Priorität der Transition i und k die Summe aller Prioritätsgewichte ist. α ist eine Prioritätsskalierungskonstante, die im Wertebereich $0 \leq \alpha \leq 1$ liegt. p_i entspricht dem $|\delta_i| + \epsilon$. ϵ ist ein kleines positives offset, das verhindert, dass eine Transition mit $|\delta| = 0$ überhaupt nicht gewählt wird. Der δ Wert kann über die Wertefunktion

für die Transition i bestimmt werden. Eine weitere Möglichkeit eine Verbesserung zu erzielen wäre es die Gewichtungen der Wertefunktion vor Trainingsbeginn mit dem Replaybuffer zu kopieren und nach einer bestimmten Anzahl an Trainingsepisoden beide Gewichtungen durch den Agenten bezüglich der Performance zu vergleichen. Stellen sich die frisch trainierten Gewichte als eine bessere Approximation heraus, können diese durch die alten ersetzt werden.

6 Evaluation der Messdaten

In diesem Kapitel werden die Messungen der verschiedenen Varianten von „EinStein würfelt nicht!“ mit dem Random, Monte Carlo, MCTSE und anschließend dem TD-Ntuple Agenten untersucht. Als Benchmarkspieler wird erstmals der Random Agent verwendet und für den TD-Ntuple Agenten der MCTSE Agent hinzugezogen. Die Messungen werden anhand des höchsten Mittelwertes der Gewinnraten als optimal angesehen. Dabei spielt der Agent in der vier Spieler Variante immer mit seinem Gegener gleichzeitig im Team.

6.1 Random Agent im Selbstspiel

Der Zufallsagent wird nur der vollständigkeit her mit in die Evaluation aufgenommen Das Spielverhalten im Selbstspiel erreicht aus der Perspektive des eröffnenden Spielers bei 1000000 Testspielen für die verschiedenen Varianten Gewinnraten nahe 50 %. Die 5x5 zwei Spielervariante und die 6x6 Vierspieler Variante lassen mit Gewinnraten von 52,1 % und 49,2 % keinen klaren Bias erkennen, dass ein Spieler oder ein Team einen Vorteil hat. Wohingegen bei den verkleinerten Varianten die Gewinnchance bei zwei Spielern bei 54,1% für den eröffenenden Spieler und bei der kleinen vier Spielervariante bei 53 % liegt.

6.2 Monte Carlo Agent

Der Monte Carlo Agent unterläuft eine Rastersuche in der die Parameter Iterationen und Rollout Tiefe gegenüber gestellt werden, wobei der Wert der Rollout Tiefe in dem festgelegten Bereich $21 \leq x \leq 23$ liegt und die Iterationsanzahl die Werte von 800 bis 2600 in 200 Schritten annimmt. Der Vergleichsagent ist der Random Agent. Ein Messpunkt besteht aus 200 Spielen, die gemittelt werden. Der Agent startet jeweils 100 Spiele in dem Team mit dem Eröffnungszug und 100 Spiele in dem Team, das den zweiten Spielzug tätigt. Jede Parameterkonstellation besteht dabei aus 10 Messungen. Die Ergebnisse der 2000 Spiele werden in tabellarischer Form für die fünf siegreichsten Agent je Spielvariante aufgelistet.

4x4 Spielbrett für 2 Spieler

Die Gegenüberstellung aus der Tabelle 3 zeigt, dass die Messungen alle einen Kernwert im Mittelwert der Gewinnrate von mindestens 82.0 Prozent ergeben. Den höchsten Mittelwert von 82,9 % kann der MC Agent mit einer Parametrisierung von 2000 Iterationen und einer Rollout Tiefe von 23 erreichen. Der Median dieser Messung beträgt 83,5 %. Im Vergleich dazu kann der Agent aber mit der Hälfte der Iterationen und einer Rollout Tiefe von 18 ein Ergebnis von 82,1 % im Mittelwert und 82 % im Median erzielen. Folglich ist eine Verbesserung der Spielleistung um 0,8 % rechenintensiv.
Die Abbildung 16 zeigt die Auswirkung der Iterationen auf den Agenten mit einer Rollout Tiefe von 23. Das Maximum im Mittelwert liegt bei 2000 Iterationen. Das Minimum bei 1800 Iterationen mit einer Siegesrate von 78,1 %. 1200 und 2400 Iterationen

Iterationen	Rollout Tiefe	Mittelwert	Standardabweichung	Min.	Median	Max.
1000	18	0.821	0.029981	0.75	0.820	0.86
	21	0.827	0.038312	0.74	0.835	0.87
1800	19	0.824	0.035653	0.76	0.820	0.87
2000	23	0.829	0.051521	0.70	0.835	0.89
2200	18	0.820	0.041366	0.76	0.815	0.89

Tabelle 3 MC-N gegen Random, 4x4 2 Spieler, die 5 besten Messungen

liegen mit 78,2 % beide somit sehr nahe am Minimum.

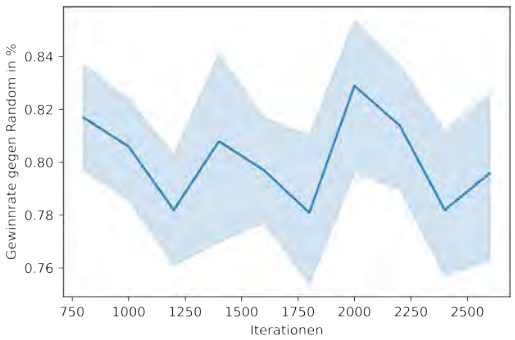


Abbildung 16 MC-N gegen Random, Auswirkung der Iterationsanzahl

5x5-Spielbrett für 2 Spieler

In der klassischen Variante von „EinStein würfelt nicht!“ beträgt die höchste Messung der durchschnittlichen Gewinnrate 80,3 % mit einem Median von 80 %. Die Tabelle 4 zeigt, dass durch eine geringere Iterationszahl und Rollout Tiefe ein höherer Median von 81 % gemessen wurde. Die Messungen der Tabelle erzielen alle eine Gewinnrate im Durchschnitt von 79,6 % und im Median von 79 %. Im Vergleich zu den Messungen der kleineren zwei Spieler Variante ist das Maximum der durchschnittlichen Gewinnrate der klassischen Variante 1,7 % geringer als das Minimum und 2,6 % geringer als das Maximum der durchschnittlichen Gewinnrate der verkleinerten Variante. Wobei die maximalen und minimalen Messpunkte den gleichen Bereich von 70 % bis 89 % abdecken.

		Mittelwert	Standardabweichung	Min.	Median	Max.
Iterationen	Rollout Tiefe					
800	19	0.794	0.039497	0.73	0.810	0.83
1200	17	0.796	0.040056	0.74	0.795	0.89
	20	0.789	0.054252	0.70	0.790	0.87
	22	0.803	0.050563	0.73	0.800	0.88
2400	18	0.798	0.032931	0.72	0.805	0.84

Tabelle 4 MC-N gegen Random, 5x5 2 Spieler, die 5 besten Messungen

4x4-Spielbrett für 4 Spieler

Die Messungen der kleinen vier Spieler Variante haben die optimalen Parameter bei 2600 Iterationen und einer Rollout Tiefe von 20. Im Vergleich sind die Rollout Tiefen 19 und 22 mit der gleichen Iterationsanzahl um 0,4 %-Punkte und 0,15 %-Punkte schlechter. Alle Messungen der durchschnittlichen Gewinnrate überschreiten im Mittelwert 58 %, aber unterschreiten die 59 % Grenze. Agenten mit einer Parametrisierung von 2400 Iterationen und einer Rollout Tiefe von 20, weisen die kleinste Standardabweichung auf. Der höchste Median liegt mit 59 % bei der optimalen Parameterkonstellation.

		Mittelwert	Standardabweichung	Min.	Median	Max.
Iterationen	Rollout Tiefe					
1600	22	0.5845	0.048902	0.510	0.5825	0.675
2400	20	0.5880	0.025188	0.555	0.5825	0.645
2600	19	0.5850	0.043970	0.530	0.5725	0.665
	20	0.5890	0.034059	0.535	0.5900	0.640
	22	0.5875	0.039388	0.525	0.5825	0.660

Tabelle 5 MC-N gegen Random, 4x4 4 Spieler, die 5 besten Messungen

6x6-Spielbrett für 4 Spieler

In der großen Spielvariante für vier Spieler können die Agenten mit der besten Performance eine untere Gewinnrate von 60,3 % erreichen. Den fünften Platz teilen sich zwei Agenten mit der Parametrisierung von 800 Iterationen und einer Rollout Tiefe von 22, sowie ein Agent mit einer Iterationsanzahl von 1000 und einer Rollout Tiefe

von 20. Gleichzeitig ist zu sehen, dass die Agenten aus der Tabelle 6 bei steigender Iterationsanzahl, innerhalb der 6 besten Messungen, eine Steigerung der durchschnittlichen Gewinnraten erzielen können. Im Vergleich zu der kleineren Variante ist zu sehen, dass die Agenten mit der Monte Carlo Methode eine Steigerung der Gewinnrate bei Vergrößerung der Spielkomplexität erzielen können. So ist das Minimum der durchschnittlichen Gewinnraten um 1.4 % höher als das Maximum in der verkleinerten Variante. Eine Überlegung ist, dass aufgrund der höheren Spielkomplexität, Agenten in der Lage sind, sich gegen willkürlichen Spielzüge der Zufallsagenten aus unvorteilhaften Spielsituationen befreien können, während in der verkleinerten Variante die Spielzüge der Zufallsagenten eine größere Auswirkung auf den Spielverlauf haben. Folglich hat der Agent mehr Spielzüge zur Verfügung, um die Episode für sich zu entscheiden.

		Mittelwert	Standardabweichung	Min.	Median	Max.
Iterationen	Rollout Tiefe					
800	22	0.6030	0.026373	0.56	0.6025	0.640
1000	20	0.6030	0.046260	0.54	0.5950	0.700
2000	22	0.6045	0.037227	0.55	0.6050	0.670
	23	0.6060	0.025798	0.57	0.6050	0.640
2200	21	0.6090	0.049205	0.53	0.6125	0.670
	22	0.6185	0.040760	0.56	0.6075	0.685

Tabelle 6 MC-N gegen Random, 6x6 4 Spieler, die 6 besten Messungen

6.3 MCTSE Agent

Die Messungen des MCTSE Agenten wurden mit einer festen Iterationsanzahl von 2000 und variablen Parametern in der Baumtiefe, Rollout Tiefe und Knotenanzahl getätigt. Die Rollout Tiefe ist auf die Werte 125,150,175, die Baumtiefe auf 21 22 23 und die maximale Knotenanzahl auf 450,500 und 550 beschränkt. Jede Messung spielt zehn Spiele gegen den Randomagenten und wird zehnfach ausgeführt. So spielt jede Einstellung 100 mal gegen den Randomagenten. Dabei spielt der Agent 50 mal aus der Perspektive des eröffnenden Teams/Spielers und weitere 50 mal aus der Perspektive des anderen Teams/Spielers. Es werden jeweils die fünf siegreichsten Agenteneinstellungen in tabellarischer Form aufgelistet.

4x4 Spielbrett für 2 Spieler

Diese Spielvariante erreicht der MCTSE Agent einen Mittelwert der Gewinnrate gegen den Randomagenten von 90%. Diese können von 4 verschiedenen Parametriseriungen erreicht werden. Die untere Grenze der Gewinnrate liegt bei 86%. Es zeigt sich, dass der Agent mit der geringsten Knotenanzahl die kleinste Standardabweichung hat. Aus der Tabelle 7 ist zu entnehmen dass der Mittelwert der Gewinnraten auf die zwei verschiedene Werte 86% und 90% fallen.

			Mittelwert	Standardabweichung	Min.	Median	Max.
Rollout Tiefe	Baumtiefe	max. Knotenanzahl					
125	21	550	0.90	0.141421	0.6	1.0	1.0
	23	500	0.86	0.164655	0.6	0.9	1.0
		550	0.86	0.164655	0.6	0.9	1.0
150	21	550	0.90	0.194365	0.4	1.0	1.0
	23	450	0.90	0.105409	0.8	0.9	1.0
		500	0.86	0.134990	0.6	0.8	1.0
175	22	500	0.86	0.164655	0.6	0.9	1.0

Tabelle 7 4x4 - 2 Spieler, MCTSE gegen Random, die 7 besten Agenteneinstellungen

5x5 Spielbrett für 2 Spieler

Die Tabelle 8 zeigt, dass der MCTSE Agent in der klassischen Variante von EWN eine maximale Performance von 96 % im Mittelwert gegen den Zufallsagenten erzielt. Alle Agenten erreichen im Mittelwert mehr als 90 % und übertrumpfen damit die Ergebnisse der kleineren Spielvariante. Durch den starken Einfluss auf das Spielverhalten von den Zufallsereignissen, kann der Agent ähnlich wie der MC Agent, eine bessere Performance erreichen, je mehr Zeitschritte ein Spiel durchschnittlich hat.

			Mittelwert	Standardabweichung	Min.	Median	Max.
Rollout Tiefe	Baumtiefe	max. Knotenanzahl					
125	23	450	0.96	0.084327	0.8	1.0	1.0
150	22	500	0.92	0.103280	0.8	1.0	1.0
	23	500	0.90	0.141421	0.6	1.0	1.0
		550	0.92	0.139841	0.6	1.0	1.0
175	22	550	0.90	0.141421	0.6	1.0	1.0

Tabelle 8 5x5 - 2 Spieler, MCTSE gegen Random, die 5 besten Agenteneinstellungen

4x4-Spielbrett für 4 Spieler

In der kleinen vier Spieler Variante von EWN erzielt der MCTSE Agent in den 5 mti den besten Einstellungen eine durchschnittliche Gewinnrate von $\geq 68\%$. Die maximale Performance wird mit einer Iterationsanzahl von 2000, einer Rollout Depth von 125, einer Baumtiefe von höchstens 22 und einer maximalen Expectimaxknotenanzahl von 550 erreicht. Von den aufgeführten Werten innerhalb der Tabelle 9 erreicht der Agent mit dieser Einstellung einen Median von 70%, obwohl das Minimum in dieser Messreihe einmal 40% erzielt hat. Die geringste durchschnittliche Gewinnrate konnte bei den beiden Agenten mit den Parametertupeln (2000,125,22,450) und (2000,175,21,500) gemessen werden. Folglich ist in der kleinen vier Spieler Variante von EWN die maximale Knotenanzahl ein entscheidender Faktor für die Gewinnraten, denn auch der zweitplatzierte Agent hat eine maximale Knotenanzahl von 550 Knoten.

			Mittelwert	Standardabweichung	Min.	Median	Max.
Rollout Tiefe	Baumtiefe	max. Knotenanzahl					
125	21	450	0.69	0.152388	0.5	0.65	0.9
	22	450	0.68	0.168655	0.3	0.70	0.9
		550	0.73	0.156702	0.4	0.75	1.0
150	21	550	0.71	0.159513	0.5	0.65	1.0
175	21	500	0.68	0.147573	0.5	0.70	0.9

Tabelle 9 4x4 - 4 Spieler, MCTSE gegen Random, die 5 besten Agenteneinstellungen

6x6-Spielbrett für 4 Spieler

In der großen vier Spieler Variante ist das beste Ergebnis für EWN mit einer durchschnittlichen Gewinnrate von 73% zweimal durch die Parametertupel (2000, 125, 22, 500) und (2000, 150, 21, 500) erzielt wurden. Dabei ergibt sich aus der Tabelle 10, dass das Tupel mit einer Rollout Tiefe von 150 den höchsten Median innerhalb der Messungen hat. Die anderen Agentenmessungen haben den gleichen Median und unterscheiden sich großteils in den minimal erzielten Werten einer Messreihe. Das schlechteste Ergbnis bildet der Agent mit dem Tupel (2000, 150, 22, 450) mit einer Gewinnrate im Mittelwert von 69 %. Dabei hat der beste Agent der kleineren Variante eine gleichhohe Gewinnrate, jedoch liegt der Median 5% unterhalb des Medians der großen 4 Spielervariante.

			Mittelwert	Standardabweichung	Min.	Median	Max.
Rollout Tiefe	Baumtiefe	max. Knotenanzahl					
125	21	450	0.72	0.168655	0.4	0.7	1.0
	22	450	0.71	0.137032	0.5	0.7	0.9
		500	0.73	0.082327	0.6	0.7	0.9
150	21	500	0.73	0.240601	0.3	0.8	1.0
175	23	450	0.69	0.179196	0.4	0.7	0.9

Tabelle 10 6x6 - 4 Spieler, MCTSE gegen Random, die 5 besten Agenteneinstellungen

6.4 TD-Ntuple Agent

Es wird zuerst die Parametriseriung zweitens der Versuchsaufbau und zu letzt die erzielten Ergebnisse betrachtet.

6.4.1 Parametriesierung

Die optimale Parametrisierung für einen Agenten zu finden, ist eine zeitaufwendige Aufgabe. Eine reine Rastersuche, die alle Parameterverbindungen untersucht, würde bei 4 verschiedenen Parametern mit 10 Einstellungsmöglichkeiten pro Parameter, unter der Annahme, dass jeder Agent knapp 30 Sekunden benötigt, um 50.000 Trainingsiterationen zu durchlaufen, bereits $\frac{10^5 \cdot 30}{60 \cdot 60} = 833,33$ Stunden dauern, wenn jeder Agent zehn mal trainiert wird. Jedoch werden tatsächlich mindestens 6 Parameter untersucht, was den verfügbaren Zeitaufwand überschreiten würde.
Stattdessen wird jeder Parameter einzeln betrachtet und versucht ein Optimum für diesen zu finden. Die gefundene Einstellung wird dann für die nächste Einstellungssuche des nächsten Parameters übernommen. Besitzt ein Parameter einen Anfangs- und Endwert, sowie α und ϵ, wird dieses Paar in einer Rastersuche herausgefiltert. Die Ergebnisse werden für jeden Parameter gruppiert und der Mittelwert, Median, Minimum, Maximum, sowie die Standardabweichung berechnet. Als Optimum wird der Wert genommen, welcher einen Grenzwert im Median und Mittelwert überschreitet. Zur Evaluierung werden insgesamt 1000 Testspiele aus beiden Perspektiven in der klassischen zwei Spieler Variante gegen den Zufallsagenten getätigt. Die initialen Starteinstellungen werden aus der Tabelle 11 entnommen.

Parameter	Startwert
Alpha initialisiert	0.0
Alpha final	0.0
Epsilon initialisiert	0.0
Epsilon final	0.0
Gamma	1.0
Lambda	0.0

Tabelle 11 Starteinstellung für TD-Ntuple Agenten

Alpha Ein Optimum der Lernrate Alpha ist gegeben durch das Tupel (0.3,0.1) wobei der erste Wert den Startwert und der zweite Wert den finalen Alphawert angibt. Dabei gilt für das Wertetupel $\alpha_s \in$ [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0] und für $\alpha_f \in [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]$, wobei α_s der Startwert und α_f der Finalwert ist. Die fünf besten Ergebnisse können einen Richtwert der Gewinnrate im Median und Mittelwert von 72 % übersteigen. Tabelle 12 zeigt die fünf siegreichsten Alphawerttupel gruppiert.

		Mittelwert	Standardabweichung	Min.	Median	Max.
alpha	alphafinal					
0.1	0.2	0.728	0.015492	0.71	0.725	0.76
0.3	0.1	0.740	0.048762	0.66	0.735	0.81
	0.2	0.722	0.060332	0.62	0.725	0.80
0.4	0.3	0.721	0.047947	0.64	0.730	0.79
1.0	0.2	0.729	0.060083	0.60	0.740	0.81

Tabelle 12 Messungen für das Alphawerttupel

Dabei sticht die Kombination des Tupels (0.3,0.1) besonders heraus. Die durchschnittliche Gewinnrate und der Median liegen beide bei $\geq$ 73,5 %. Der Unterschied zum zweitplatzierten Tupel liegt in der geringeren Gewinnrate im Mittelwert von 1,1 %-Punkten. Der Median liegt 0,5 %-Punkte unterhalb des Medians des zweitplatziertem Wertepaares (1.0,0.2). So erreicht das gefundene optimale Wertepaar (0.3, 0.1) gleich mit dem Tupel (1.0, 0.2) das höchste Maximum in der Gewinnrate. Das Minimum liegt jedoch im Wertepaar (0.3, 0.1) 6 %-Punkte höher. Abbildung 17 zeigt die Verteilung der Datenmessungen je nach Alphatupel. Dabei zeigt sich, dass die Messung mit dem Tupel (1.0, 0.2) einen Ausreißer aufweist, der das Minimum bildet und folglich den Durchschnitt negativ beeinflusst.
Die gefundenen Werte für das Alphawertetupel (0.3,0.1) bilden nun die Startwerte für das Epsilontupel.

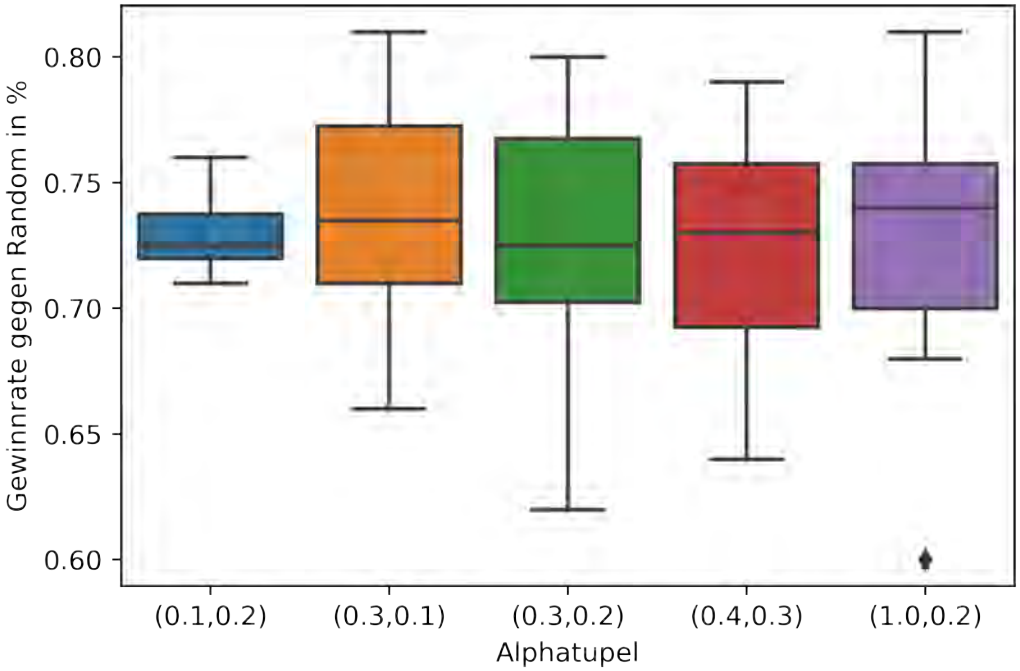


Abbildung 17 5 beste Messungen als Boxplot

Epsilon Das Optimum für das Epsilontupel ist (0.0, 0.3) mit einer durchschnittlichen Gewinnrate von 74,8 % und einem Median von 75,5% überschreitet diese Einstellung den Grenzwert von 74,5%. Epsilon kann für ϵ_s und ϵ_f Werte aus folgender Menge annehmen: $[0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]$. Der unterste Wert der fünf besten Agenteneinstellungen zeigt einen Mittelwert der Gewinnrate gegen den Zufallsagenten von 73.6% mit dem geringsten Median in Höhe von 74 %.

		Mittelwert	Standardabweichung	Min.	Median	Max.
$epsilon_s$	$epsilon_f$					
0.0	0.3	0.748	0.051812	0.64	0.755	0.82
0.1	0.0	0.736	0.044771	0.65	0.740	0.81
	0.2	0.739	0.048865	0.62	0.750	0.80
0.2	0.2	0.752	0.051812	0.67	0.740	0.85
0.3	0.3	0.745	0.045277	0.69	0.745	0.80

Tabelle 13 5 besten Messungen für das Epsilontupel

Die Abbildung 18 zeigt die Tabelle als Boxplots sortiert nach den Epsilonwerten. Die Tupel (0.1, 0.0) und (0.1,0.2) besitzen jeweils drei Ausreißer. Das Maximum bildet ein Agent mit dem Tupel (0.2,0.2) und liegt damit 3 Prozentpunkte über dem stärksten Agenten von dem Tupel (0.0,0.3). Jedoch zeigt sich, dass die Streuung der Datenpunkte innerhalb des Boxplots des Tupels (0.0, 0.3) oberhalb des Medians konzentrierter sind. Eine mögliche Überlegung für die Performance bei steigender Epsilonrate ist, dass der Agent nachdem er eine Basisstrategie erlernt hat, diese durch zufällige Spielzüge im späteren Trainingsverlauf ausbaut und dadurch bessere Resultate erzielen kann.

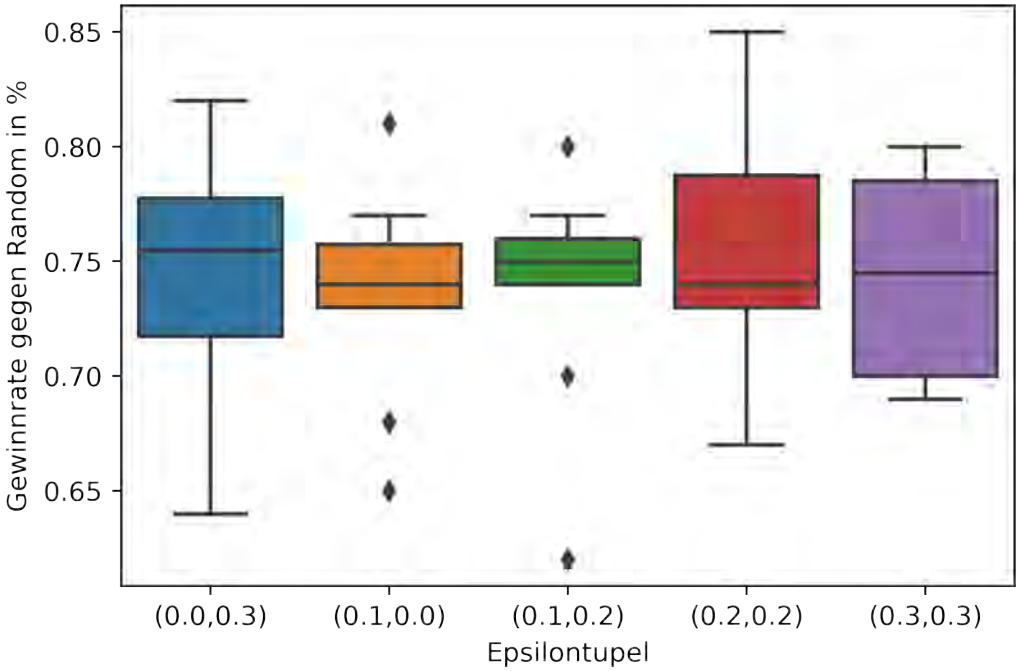


Abbildung 18 5 beste Epsilontupelmessungen als Boxplot

Gamma Für den Wert Gamma hat sich gezeigt, dass eine Reduzierung des Wertes schlechtere Ergebnisse in der Gewinnrate erzielt. Abbildung 19 verdeutlicht, dass sich ein Gamma Wert von 0.0 am schlechtesten auf die Gewinnrate auswirkt. Der Mittelwert der Messungen erreicht durch diese Einstellung lediglich eine Gewinnrate von 14,2 %. erhöht sich Gamma auf 0.1, so ereicht die Gewinnrate mehr als 40 %. Das ist typische, denn bei einem Discountfaktor von 0 wählt der Agent immer die Aktion aus, die den maximalen Erfolg im jetzigen Spielzustand erzielt. Ein Wert von 1.0 erzielt eine Gewinnrate im Median von 75 % und im Mittelwert von 76 %. Eine Ausnahme bilden die Messungen mit dem Wert von 0.5, dort ist ein kleiner Anstieg in der Gewinnrate zu sehen.

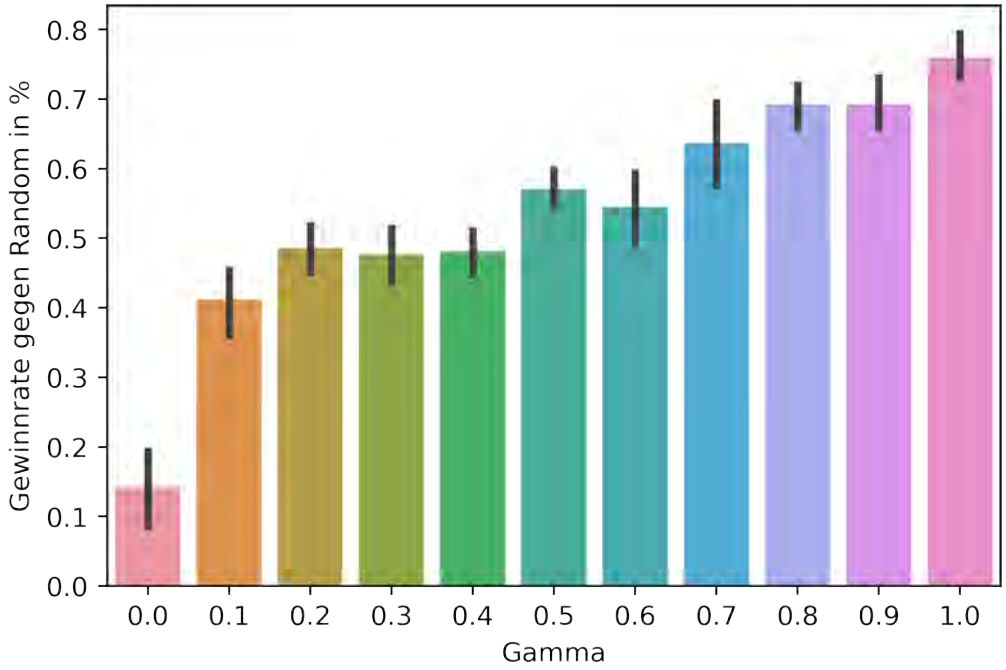


Abbildung 19 Gamma: Entwicklung der Gewinnrate

Lambda Für den Parameter Lambda stellt sich heraus, dass ein kleiner Wert von 0.1 das Ergebnis nochmals verbessert. So kann ein Mittelwert der Gewinnrate von $76,2\% \pm 0,67\%$ erzielt werden. Die Tabelle 14 zeigt, wie sich die Werte von λ auf die Spielperformance der Agenten auswirkt.

		Mittelwert	Standardabweichung	Min.	Median	Max.
lambda						
0.0	0.720	0.063770	0.61	0.735	0.82	
0.1	0.762	0.066966	0.63	0.780	0.87	
0.2	0.726	0.044522	0.65	0.730	0.79	
0.3	0.758	0.069089	0.60	0.770	0.85	
0.4	0.755	0.040620	0.70	0.750	0.82	
0.5	0.740	0.042947	0.67	0.725	0.81	
0.6	0.710	0.051208	0.63	0.705	0.81	
0.7	0.716	0.037771	0.67	0.720	0.79	
0.8	0.670	0.045461	0.60	0.670	0.73	
0.9	0.571	0.092069	0.41	0.580	0.73	
1.0	0.141	0.073401	0.00	0.150	0.24	

Tabelle 14 Übersicht aller Lambdamessungen

Optimale Einstellung Das gefundene Optimum in der Parametrisierung wird nun für die folgenden Messungen verwendet außer es ist explizit anders beschrieben. Die Einstellungen sind aus der Tabelle 15 entnommen.

Parameter	Startwert
Alpha initialisiert	0.3
Alpha final	0.1
Epsilon initialisiert	0.0
Epsilon final	0.3
Gamma	1.0
Lambda	0.1

Tabelle 15 Endeinstellungen für TD-Agenten

6.4.2 Varianten mit höheren Trainingsepsioden

Die Agenten werden mit einer höheren Anzahl an Trainingsspielen trainiert. Dabei werden die finalen Spielstärken, sowie die Trainingsentwicklung für die einzelnen Varianten durch den Random und den MCTSE Agenten gemessen und miteinander verglichen. Ein Agent wird fünfach für jeweils 5.000.000 Epsioden trainiert. Die Evaluation gegen den Randomagenten besteht aus 1000 Testspielen udn gegen den MCTSE Agenten aus 20 Testspielen, dabei werden 50 % der Spiele aus der Perspektive des eröffnenden Teams gespielt.

4x4-Spielbrett für 2 Spieler

In der kleinen zwei Spieler Variante von „EinStein würfelt nicht!“ erzielt der TD-Ntuple Agent als finale Siegesrate gegen den Random Agenten und den MCTSE Agenten im Mittelwert 65,44 % und 39 %, wie Abbildungen 20 und 21 zeigen. Die hellblauen Bereiche stellen die Standardabweichung dar. Dabei wird der maximale Erfolg gegen den Random Agenten nach 4535000 Trainingsspielen von 68,12 % festgestellt. In Abbildung 20 erkennt man einen leichten Aufwärtstrend in der Spielleistung gegen den Random Agenten. Die untere Schranke von 50 % im Mittelwert wird in dieser Variante nie gegen den Random Agenten unterschritten. Die schlechteste Leistung wird nach 120000 und 460000 Trainingsspielen mit 54,4 % erreicht.
Im Vergleich übersteigen die Agenten die Grenze von einer durchschnittlichen Gewinnrate in Höhe von 50 % gegen den MCTSE Agenten fünf mal. Das erste mal nach 330000 Iterationen mit 53 %. Das zweite Mal wird diese Erfolg nach 1.200.000 Trainingsiterationen und einer 55 prozentigen Gewinnrate erreicht. Diese Messung stellt gleichzeitig das Maximum dar. Zwei weitere Messungen können nach nach 1.245.000 und 1.990.000 Trainingsspielen in Höhe von 51 % gemessen werden. Die zweithöchste Messung erfolgt nach 2085000 Spielen mit 54 %. Auffallend ist, dass sich die Ergebnisse alle in der ersten Hälfte des Trainings ereignen. In der zweiten Trainingshälfte wird der Wert von 50 % nur einmal nach 4895000 Episoden Trainingszeit erzielt. In Abbildung 21 ist kein Trend für eine stetige Erhöhung der Gewinnrate gegen den MCTSE Agenten zu erkennen.

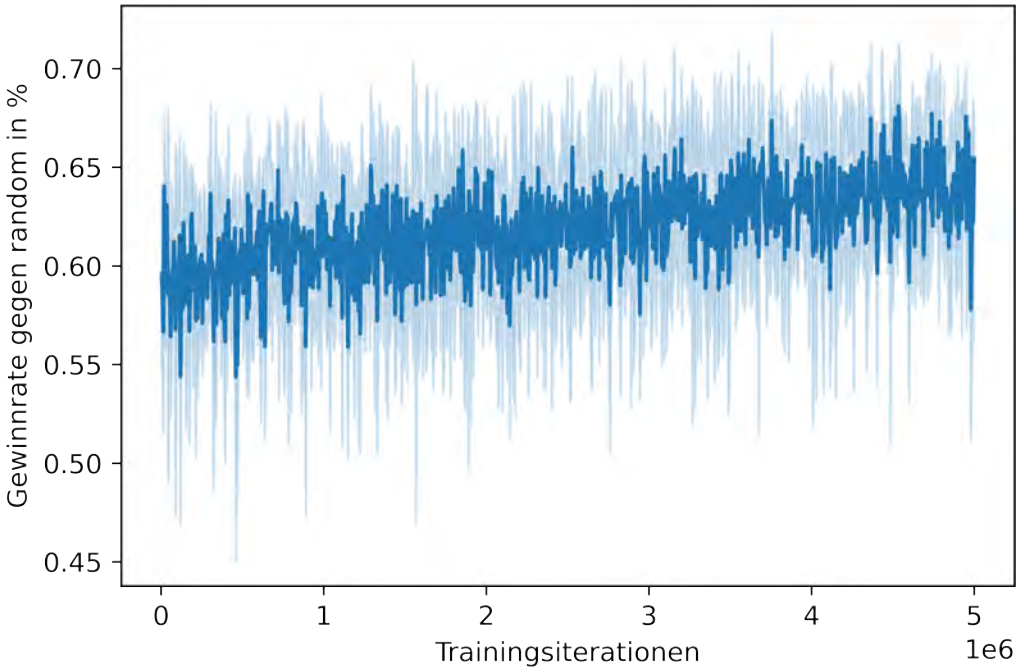


Abbildung 20 4x4 - 2 Spieler Entwicklung: TD-Ntuple gegen Random

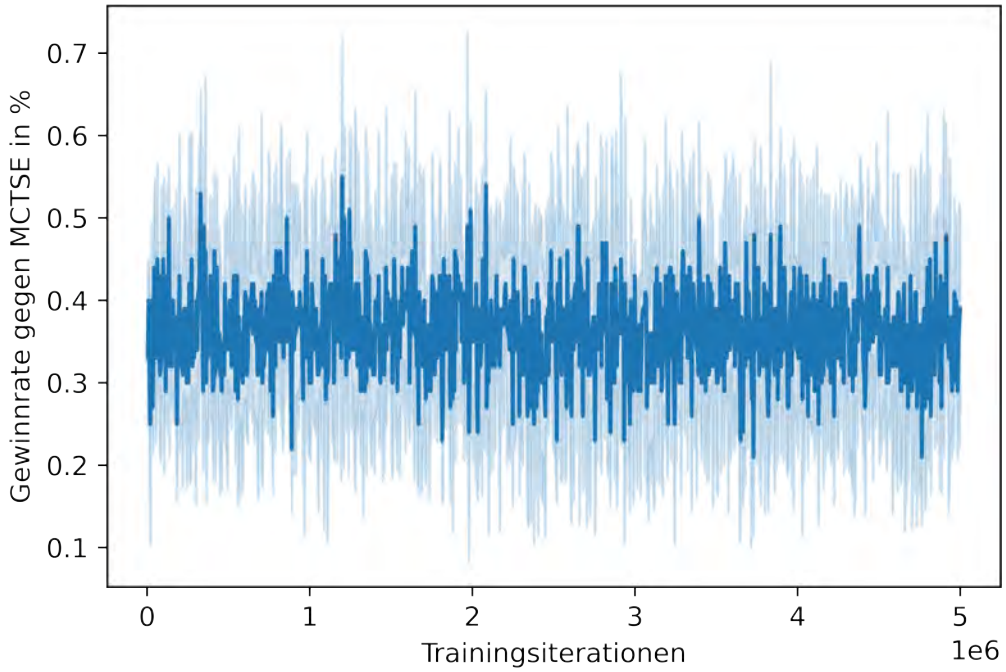


Abbildung 21 4x4 - 2 Spieler, Lernerfolg: TD-Ntuple gegen MCTSE

5x5-Spielbrett für 2 Spieler

Der TD-Ntuple Agent erreicht in der klassischen Version von EWN im Vergleich zu dem Randomagenten gegen Ende des Trainings im Mittelwert eine Siegesrate von

79,52%. Nach 3570000 Episoden wird erstmals und einmalig eine Marke von 81% überschritten. Die Abbildung 22 zeigt nach einem starken Anstieg immer noch einen lecihten Aufwärtstrend im Spielverhalten.

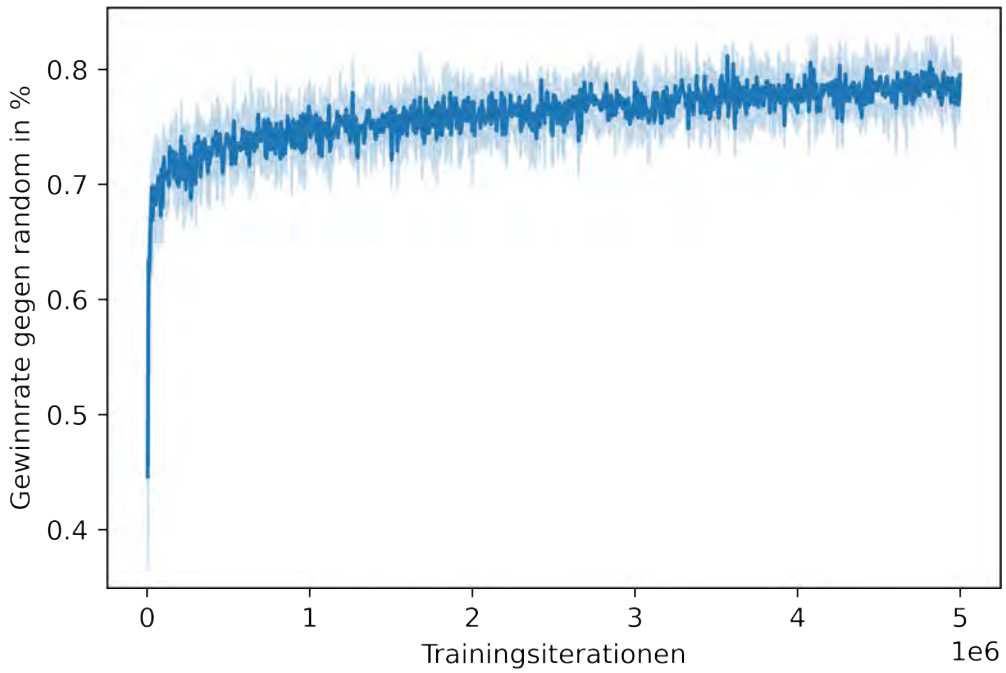


Abbildung 22 5x5 - 2 Spieler Entwicklung: TD-Ntuple gegen Randomagenten

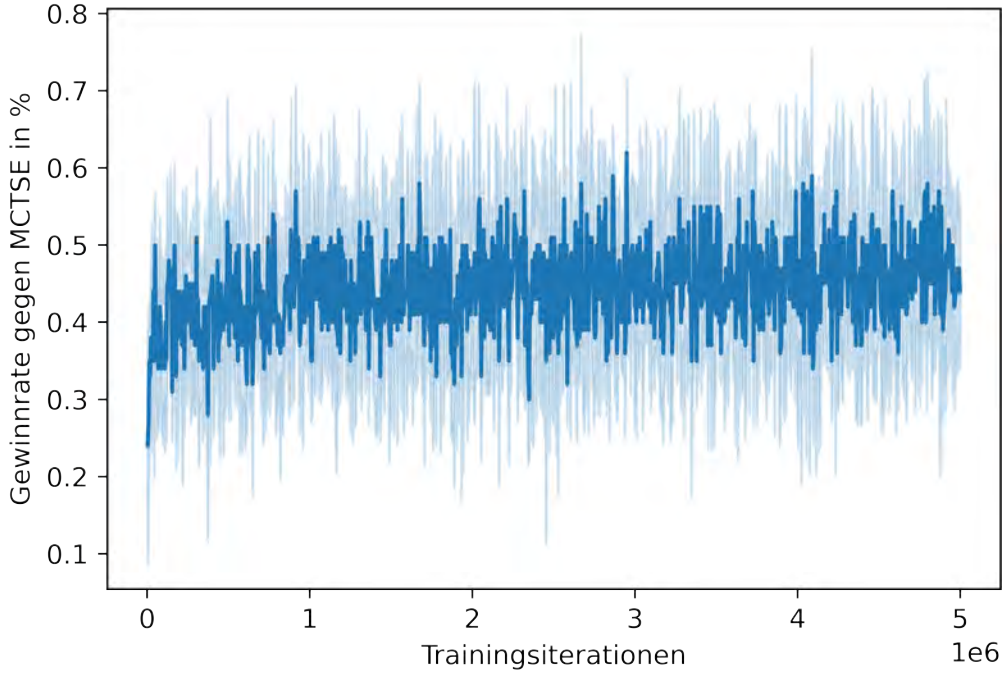


Abbildung 23 5x5 - 2 Spieler, Entwicklung: TD-Ntuple gegen MCTSE

Gegen den MCTSE Agenten wird eine finale Gewinnrate von 44% gemessen. Außerdem erreicht die Agenten im Durchschnitt eine maximale Gewinnrate von 62% nach 2950000 Trainingspspielen. Die Standardabweichung ist jedoch bei dieser Messung mit 10,3% bezüglich der Gewinnrate sehr hoch. Die Messung der Agenten bei 2950000 Trainingsiterationen zeigen ein Minimum der Siegesrate von 45% und ein Maximum von 70%. Der Median liegt bei einer Siegesrate für diese Messung bei 65% und ist damit 3% höher als der Mittelwert. Folglich sind die Messungen für diesen Datenpunkt linkschief.

4x4-Spielbrett für 4 Spieler

In Abbildung 24 ist der Entwicklungsverlauf gegen den Random Agenten für die kleine vier Spieler Variante dargestellt. Auffallend ist, dass der Agent nach 4.750.000 Epsioden eine Gewinnrate von durchschnittlich 55,1% erzielt und nach 5.000.000 Epsioden um 5,08% auf 49,93% abfällt. Es wird erstmals nach 1.250.000 Trainingseinheiten eine Gewinnrate von über 50% erreicht und dies bleibt konstant oberhalb bis auf den Einbruch am Ende des Trainings. Nach weitern 1.750.000 Trainingsspielen steigt die Gewinnrate auf 52,7%. Das Maximum der durchschnittlichen Gewinnraten wird nach 3.250.000 Spielen erreicht und liegt bei 56%. Das zweithöchste Maximum wird nach 4.750.000 Episoden mit 55,1% erreicht.
Im Vergleich zeigt die Abbildung 25 die durchschnittliche Entwicklung der TD-NTuple Agenten gegen den MCTSE Agenten. durchschnittliche wird die 50% Grenze viermal durchbrochen in der Reihenfolge mit folgenden Gewinnraten durchbrochen. Final erreicht der Agent eine Gewinnrate von

- Nach 750.000 Episoden mit einer durchschnittlichen Gewinnrate von 50,5%
- Nach 1.750.000 Episoden mit einer durchschnittlichen Gewinnrate von 50,5%
- Nach 3.500.000 Episoden mit einer durchschnittlichen Gewinnrate von 53%
- Nach 4.750.000 Episoden mit einer durchschnittlichen Gewinnrate von 50,5%

Die niedrigste Gewinnrate erzielen die Agenten nach 1.250.000 Trainingseinheiten mit 37%.

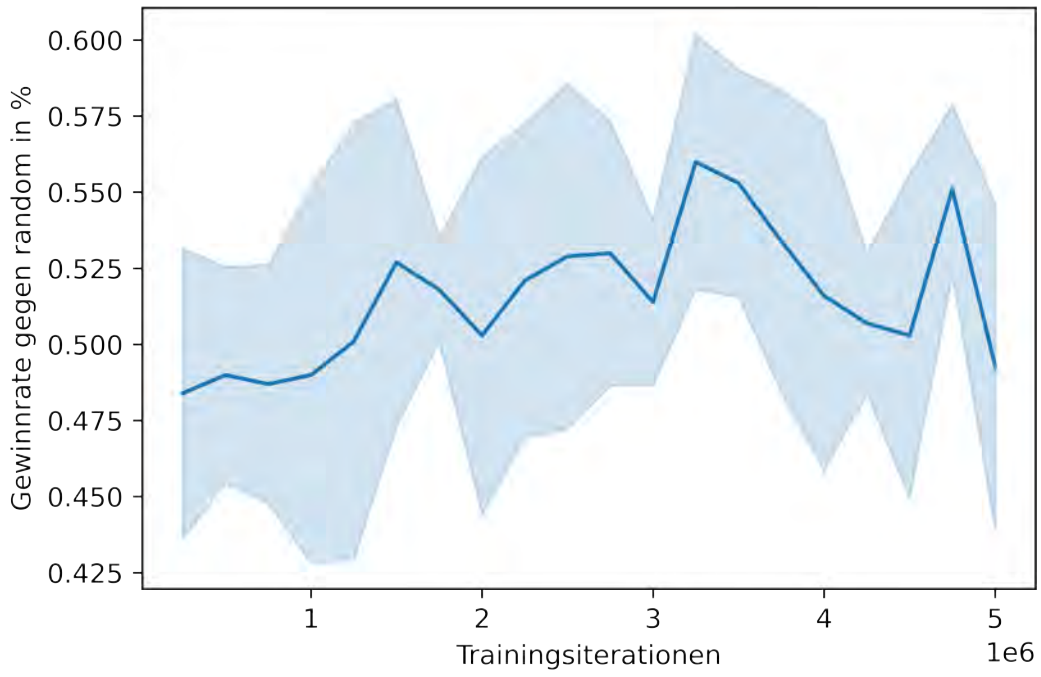


Abbildung 24 4x4-4 Spieler Entwicklung: TD-Ntuple gegen Randomagenten

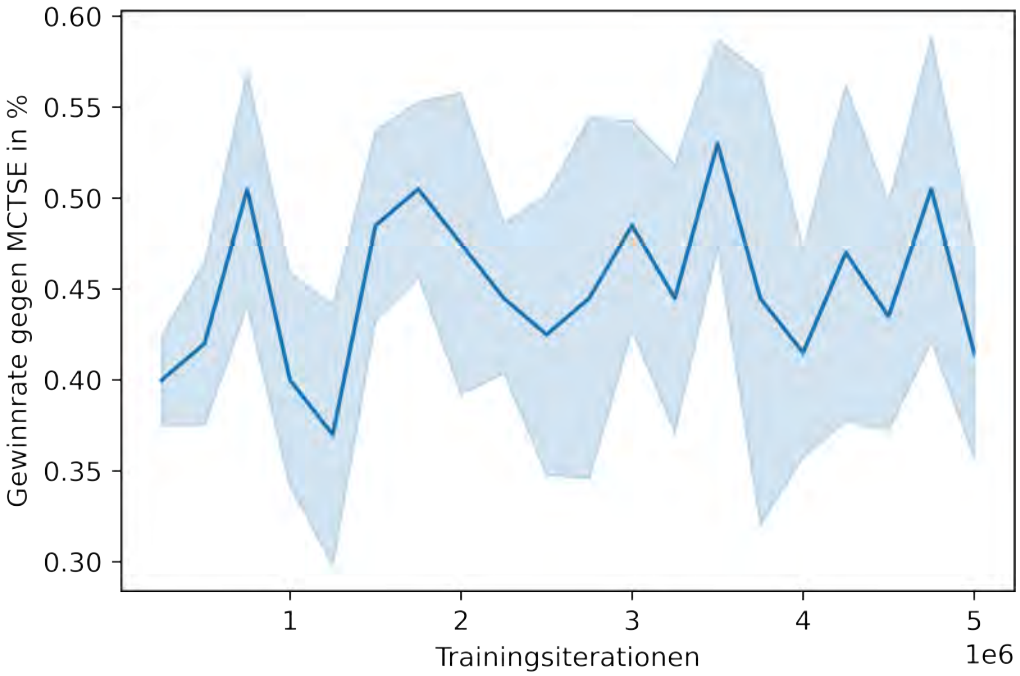


Abbildung 25 4x4 - 4 Spieler, Entwicklung: TD-Ntuple gegen MCTSE

6x6 Spielbrett für 4 Spieler

Die trainierten TD-NTuple Agenten erreichen final gegen den Randomagenten eine Gewinnrate von 50,3% und gegen den MCTSE Agenten eine Gewinnrate von 45%. Der Entwicklungsverlauf ist in der Grafik 26 und 27 als Liniendiagramm abgebildet. Der hellblaue Bereich stellt wieder die Standardabweichung dar. Erstmals wird die 50%-tige Gewinnrate gegen den Randomagenten um 0,2% nach 1.000.000 Trainingsiterationenen überschritten. Nach 4.250.000 Episoden bleibt der durchschnitt immer höher als 50%. Der Mittelwert des Maximums wird nach 4.500.000 Episoden erreicht und liegt dann einmalig bei 53,4%.

Gegen den MCTSE Agenten wird in dieser Variante von Ewn die 50% Grenze nie überschritten. Es wird aber ein Maximum von 49% nach 4250000 Trainingsspielen und ein Minimum in der Spielperformance bei 2.500.000 Epsioden erreicht.

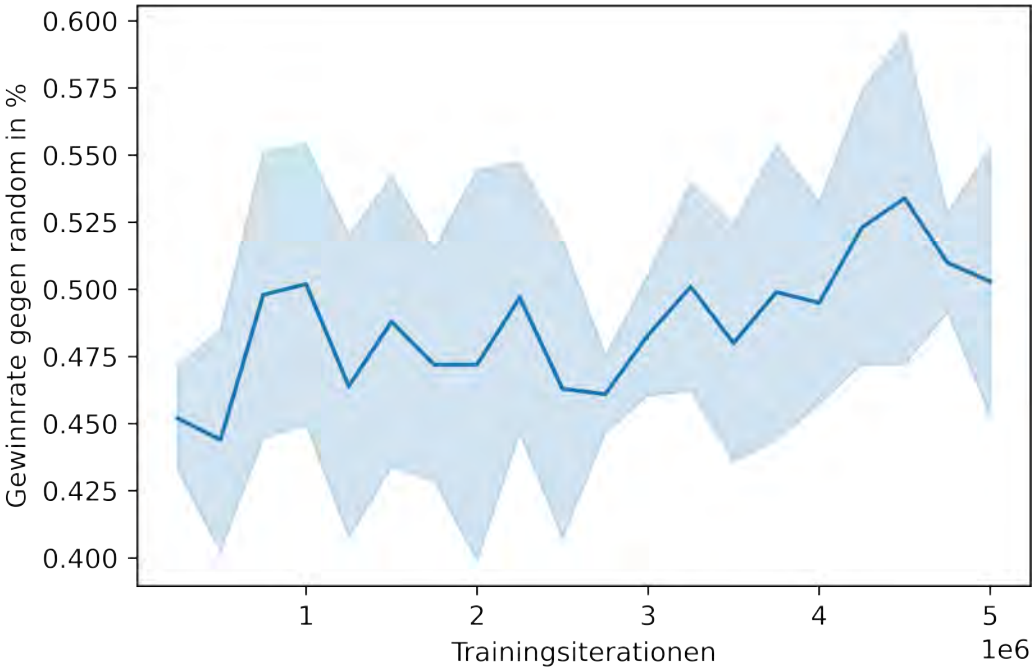


Abbildung 26 6x6-4 Spieler Entwicklung: TD-Ntuple gegen Randomagenten

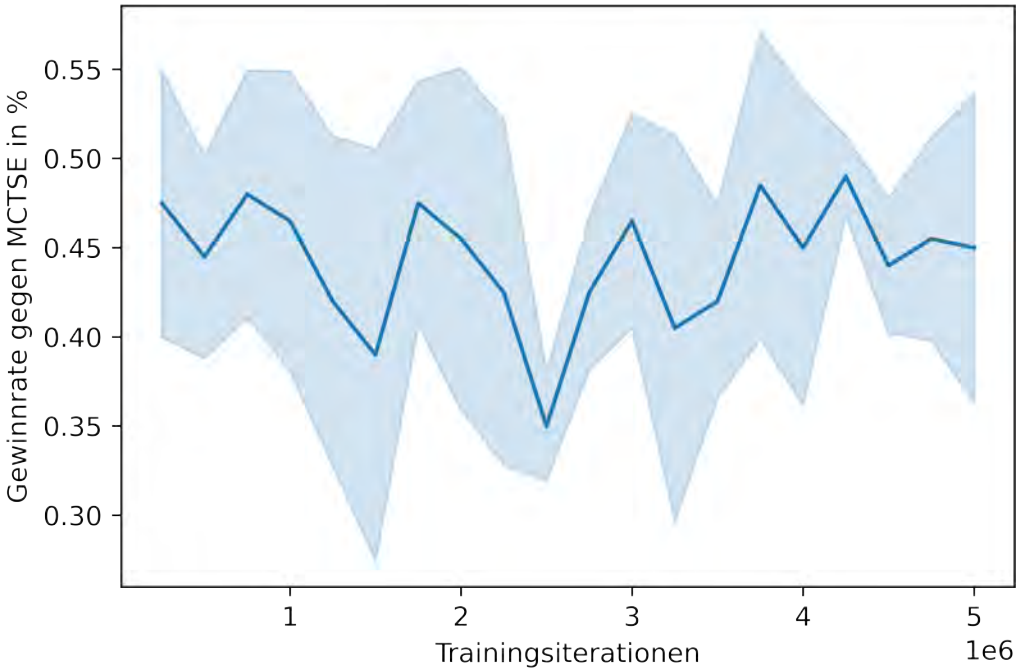


Abbildung 27 6x6 - 4 Spieler, Entwicklung: TD-Ntuple gegen MCTSE

6.4.3 Randomwalk und Randompoint

Für die Generierung von Merkmalen wurde auch die Methode des Randomwalks und des Randompoints ausprobiert. Weder der Randomwalk noch Randompoints erzielte

höhere Performance als der Randomagent. Für den Versuch des Randomwalks werden 25 Tupel mit der Länge vier generiert und jeweils 10 Agenten für 250000 Epsioden trainiert. Gespielt wird das klassischen EWN für 2 Spieler mit 6 Spielsteinen auf einem 5x5 großem Spielbrett. Die Abbildung 28 zeigt die durchschnittliche Entwicklungskurve des Agenten gegen den Zufallsagenten in Prozent.

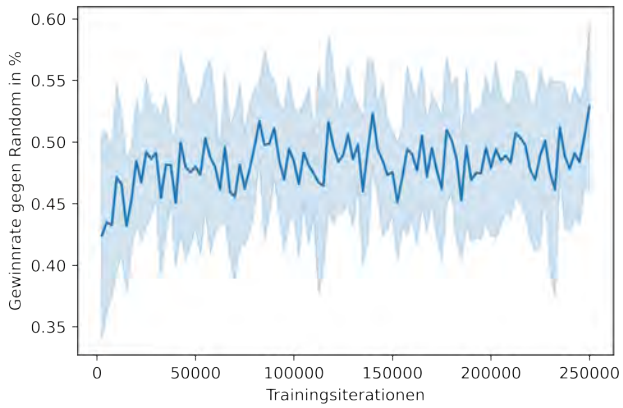


Abbildung 28 Randomwalk - Gewinnrate nach 2500000 Iterationen gegen Random

Die Gewinnrate liegt nach 250000 Trainingsiterationen durchschnittlich bei 52.87%. Größten teils spielt bewegt sich der Agent unterhalb der 50% Grenze. Daraus lässt sich schließen, dass der Agent keine Tupel finden konnte, die für das Spiel Bedeutung haben. Dies kann an dem hohen nichtdeterministischen Teil liegen, den EWN aufweist.

Ebenso erreicht die Methode Randompoint keine Gewinnrate die über die zufälligen Züge des Randomagenten hinausragt, siehe Abbildung 29. Der Mittelwert der Gewinnrate ist nach 250000 Trainingsiterationen bei 47,48% angekommen.

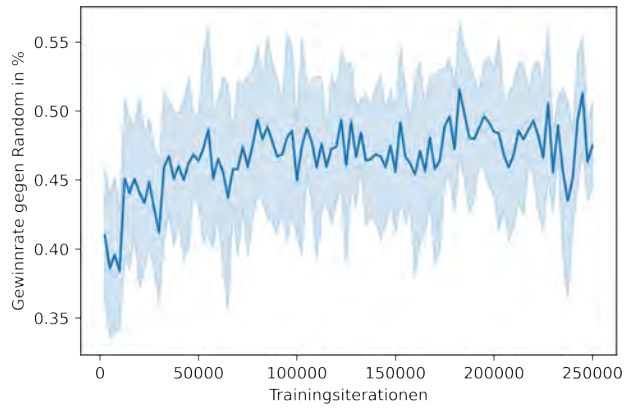


Abbildung 29 Randompoint - Gewinnrate nach 2500000 Iterationen gegen Random

6.5 Vergleich aller Agenten nach Spielvarianten

Die Reinforcement Learning Agenten müssen in den verschiedenen Umgebungen jeweils andere Strategien entwickeln um Gewinnbringende Spielzüge zu finden. Die Tabelle 16 zeigt die Spielstärke der gewinnreichsten Agenten.

		Spielvariante			
Agent	**vs.**	4x4 2 Spieler	5x5 2 Spieler	4x4 4 Spieler	6x6 4 Spieler
Random	Random	54,1 %	52,1 %	53 %	49,2 %
MC	Random	82,9 %	80,3 %	58,9 %	61,85 %
MCTSE	Random	90 %	96 %	73 %	73 %
TD-Ntuple	Random	65,44 %	79,52	49,93 %	50,3 %
	MCTSE	39 %	44 %	41,5 %	45 %

Tabelle 16 Übersicht: Die Spielstärken der Agent

Die Messungen ergeben, dass der MCTSE Agent in allen Spielvarianten mit Abstand die besten Ergebnisse erzielen kann, was auf seine Natur zurückzuführen ist. Er wurde speziell für nichtdeterministische Spiele, wie 2048 implementiert. Aufgrund der Chanceknoten ist es dem Agenten möglich die Zufallsereignisse des Spiels EWN stärker als die anderen Agenten mit in die Spielzustandbewertung einfließen zu lassen. Dabei ist besonders die hohe Gewinnrate in der klassischen zwei Spieler Variante von 96% und in der verkleinerten Variante von 90 % eine hohe Hürde, die der Agent erklimmen kann. Im Vergleich erzielt der TD-Ntuple Agent in der verkleinerten zwei Spieler Variante eine Gewinnrate von 65,44 % gegen den Zufallsagenten und eine Gewinnrate von 39 % gegen den MCTSE Agenten. In der klassischen zwei Spieler Variante ist der TD-Ntuple Agent in der Lage eine deutlich höhere Gewinnrate von 79,52 % zu erzielen. Die Gewinnrate gegen den MCTSE Agenten kann von 39 % auf 44 % gesteigert werden. Der MC Agent ist in der verkleinerten zwei Spieler Version 2,6% stärker als in der klassischen Variante und erzielt eine Gewinnrate von 82 ,9 % im Gegensatz zu 80,3 %. In beiden vier Spieler varianten erzielt der MCTSE Agent eine maximale durchschnittliche Gewinnrate von 73%. Wohingegen der Vergleich mit dem TD-Ntuple Agenten eine Gewinnrate von 41,5% in der kleinen und eine Gewinnrate von 45% in der großen Spielvariante erzielt. Gegen den Zufallsagenten beträgt die maxmale Gewinnrate desr TD-Ntuple Agenten in beiden Fällen knapp 50%. Was bedeutet, dass es dem Agenten nicht gelungen ist ausschlaggebende Spielentscheidungen für die vier Spieler Variante, wie der MC oder der MCTSE Agent, zu treffen und seine gefundene Strategie kaum Einfluss auf das Spielverhalten hat. Der MC Agent hingegen ist auch in der vier Spieler variante stärker als der TD-Ntuple Agent und erreicht Gewinnraten von 58,9% und 61,85%. Er ist aber immer noch deutlich schwächer als der MCTSE Agent. Die schlechte Spielweise des TD-Ntuple Agenten in der vier Spieler Variante kann auf eine zu kleine Netzgröße zurückgeführt werden.

7 Fazit und Ausblick

Das Ziel dieser Arbeit ist mit Hilfe des General Board Game Frameworks, die Methodiken von künstlichen Intelligenzen am Beispiel von EWN zu untersuchen. Vorab wurde im Rahmen des Praxisprojekts die Game Learning Umgebung um das Spiel "EinStein würfelt nicht„ erweitert. Durch Hilfe der Reinforcment Learning Agenten wurde das Spiel EWN unter Bezug der Forschungsfragen untersucht. Durch die erzielten Messergebnissen kann die erste Forschungsfrage

- Welche messbaren Lernerfolge können durch die Agenten für die zwei und vier Spieler Varianten von „EinStein würfelt nicht!“ erzielt werden?

beantwortet werden. Alle Reinforcement Learning Agenten können in den zwei Spieler Varianten messbare Lernerfolge Erfolge erzielen. Der TD-Ntuple Agent erreicht nach 5 Millionen Trainingseinheiten in der klassischen Variante eine Gewinnrate gegen den Zufallsagenten von 79,52 % und gegen den MCTSE Agenten eine Gewinnrate von 44 %. Im direkten Vergleich hat der MCTSE Agent ene deutlich höhere Gewinnrate von 96 % gegen den Zufallsagenten erzielt. Der MC Agent ist auf dem zweiten Platz bezüglich der Gewinnrate mit 82,9 in der kleinen und 80,2 in der großen Variante. Bezüglich der vier Spieler Varianten kann ein messbarer Lernerfolg für den MC und den MCTSE Agenten erzielt werden. Der ausbleibende Lernerfolg für den TD-Ntuple Agenten liegt an der zu kleinen Netzgröße. Eine Erhöhung der Positionswerte ist mit der verwendeten NTuple-Architektur generell möglich, aber enorm rechenintensiv.

Die zweite Kernfrage

- Welche Unterschiede in den Lernerfolgen ergeben sich für die betrachteten verschiedene Spielvarianten von „EinStein würfelt nicht!“?

kann so beantwortet werden, dass der Unterschied zwischen den Varianten in den Lernerfolgen der Einfluss der weiteren Spieler ist. So wird ein Agent, der mit einem anderen dummen Agenten gepaart ist, durch diesen ausgebremst. Dennoch zeigt sich, dass der MCTSE Agent in beiden vier Spieler Varianten gegen den Zufallsagenten trotzdem eine Gewinnrate von 73 % erzielen kann. Verallgemeinert kann gesagt werden, wenn ein Agent stark in der zwei Spieler Variante ist, bleiben die Lernerfolge in der vier Spieler Variante nicht aus, fallen oder steigen aber je nach der Stärke des Mitspielers. So wird die hohe Gewinnrate des MCTSE Agenten durch die schlechte Gewinnrate des Mitspielers TD-Ntuple Agenten reduziert.

Die dritte Forschungsfrage

- Kann im GBG durch Experience Replay bessere messbare Lernerfolge für den TD-Ntuple Agenten realisert werden?

Zum jetzigen Zeitpunkt sind die Evaluationen des TD-Ntuple Agenten im Spiel Othello negativ ausgefallen. Es kann keine Verbesserung des Lernerfolgs erkannt werden. Im Gegenteil, mit steigender Batchzahl entwickelt sich die Lernkurve schlechter und der Agent erlernt keine Strategie. Overfitting durch zuviele einfache Transition

oder schlechte Transitionen können beides Hintergründe des Problems darstellen. Ein beschriebener erster Lösungsansatz ist es, die Transition nach ihrem Nutzen für den Agenten zu bewerten. Alternativ kann aber auch die rechenintensivere Methode der Gewichtseinfrierungen versucht werden.

Literatur

Louis Victor Allis et al. *Searching for solutions in games and artificial intelligence*. Diss. Ponsen & Looijen Wageningen, 1994.

Ingo Althöfer. „EinStein würfelt nicht!“ Ein schnelles Zweierspiel für Jung und Alt. website, 2004. URL `https://www.3-hirn-verlag.de/MasterGame/regel.html`. besucht am 15.12.2021.

Ingo Althöfer. Zu den Anfängen von „EinStein würfelt nicht!“. website, 2011. URL `https://althofer.de/ewn-ursprung.html`. besucht am 15.12.2021.

Woodrow Wilson Bledsoe and Iben Browning. Pattern recognition and reading by machine. In *Papers presented at the December 1-3, 1959, eastern joint IRE-AIEE-ACM computer conference*, pages 225–232, 1959.

Guillaume Maurice Jean-Bernard Chaslot Chaslot. *Monte-carlo tree search*. Maastricht University, 2010. Diss.

Yeong-Jia Roger Chu, Yuan-Hao Chen, Chu-Hsuan Hsueh, and I-Chen Wu. An Agent for EinStein Würfelt Nicht! Using N-Tuple Networks. In *2017 Conference on Technologies and Applications of Artificial Intelligence (TAAI)*, pages 184–189, 2017. doi: 10.1109/TAAI.2017.32.

Maria Deutscher. DeepMind's new MuZero AI develops 'superhuman' chess skills by making plans . online, 2020. URL `https://siliconangle.com/2020/12/23/deepminds-new-muzero-ai-develops-superhuman-chess-skills-making-plans/`. besucht am 06.12.2021.

Wolfgang Konen. Reinforcement Learning for Board Games: The Temporal Difference Algorithm, 08 2015.

Wolfgang Konen. General Board Game Playing for Education and Research in Generic AI Game Learning, 2019.

Wolfgang Konen. The GBG Class Interface Tutorial V2.2: General Board Game Playing and Learning. Technical report, Research Center CIOP (Computational Intelligence, Optimization and Data Mining), Oct, 2020. URL `http://www.gm.fh-koeln.de/ciopwebpub/Konen20d.d/TR-GBG.pdf`.

Wolfgang Konen and Samineh Bagheri. Final adaptation reinforcement learning for N-player games. *arXiv preprint arXiv:2111.14375*, 2021. URL `https://arxiv.org/abs/2111.14375`.

Johannes Kutsch. KI-Agenten fur das Spiel 2048: Untersuchung von Lernalgorithmen für nichtdeterministische Spiele. Master's thesis, TH Köln – University of Applied Sciences, 2017. URL `http://www.gm.fh-koeln.de/~konen/research/PaperPDF/BA_JohannesKutsch_Final-2017.pdf`. Bachelor thesis, 2nd prize in CBC award 2018.

Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

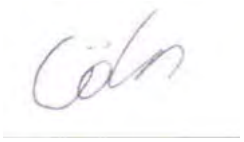
Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL `http://jmlr.org/papers/v22/20-1364.html`.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL `http://incompleteideas.net/book/the-book-2nd.html`.

Erklärung

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



Gummersbach, den 10.01.2022 Rechtsverbindliche Unterschrift