KI-Agenten im Vergleich: Erfolg von Agenten des Ludii General Game Systems in Partien gegen perfekte oder starke Agenten am Beispiel der Spiele Vier Gewinnt, Nim und Othello

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

vorgelegt von:	Meltem Leyal Seven
Matrikel-Nr.:	111 323 93
Adresse:	Nobelstraße 41
	51107 Köln
	meltem_leyal.seven@smail.th-koeln.de
	meltemleyal.seven@gmail.com

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

Gummersbach, 26.01.2023

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.

Köln, 26.01.2023

Ort, Datum	Rechtsverbindliche Unterschrift

Kurzfassung

Diese Arbeit dient der Erfassung der Spielstärke von sogenannten general game-playing KI-Agenten des Ludii General Game Systems im Vergleich zu perfekt- oder sehr stark-spielenden KI-Agenten. Das wird am Beispiel der Spiele Vier Gewinnt, Othello und Nim getan. Für diese Spiele werden die im General Board Game Playing and Learning Framework verfügbaren sehr starken AlphaBeta- und Edax-Agenten und der perfekt-spielende Bouton-Agent den Ludii-Agenten gegenüber gestellt. Es wird erörtert, wie weit die general game-playing Ludii-Agenten von perfektem Spiel sind. Dazu werden in allen drei Spielen eine Menge an Episoden mit unterschiedlichen Spielkonfigurationen gespielt und die Performanz der Ludii-Agenten an ihrer Gewinnrate gemessen. Keiner der Ludii-Agenten zeigt dabei eine besondere Spielstärke gegen diese.

Schlagwörter: GGP, Ludii, GBG, KI, Edax, Bouton

Tabellenverzeichnis

1 Hauptparameter des TCL-Agenten 13
2 Hauptparameter des TCL4-Agenten 13
3 Episoden-Konfiguration für Vier Gewinnt 20
4 Für Vier Gewinnt antretende GBG-Agenten 20
5 Zuglisten der vom UCT-Agenten in sieben Zügen gewonnenen Episoden 24
6 Gewinnraten des AlphaBeta-DL- und UCT-Agenten in Episoden gegeneinander 24
7 Durchschnittliche Längen der von AB-DL- und UCT-Agent gewonnenen Episoden 25
8 Episoden-Konfiguration für Othello 30
9 Für Othello antretende GBG-Agenten 31
10 Gewinnraten der Episoden ohne XOT Openings 32
11 Gewinnraten der Episoden mit XOT Openings 32
12 Episoden-Konfiguration für Nim 34
13 Für Nim antretende GBG-Agenten 35

Abbildungsverzeichnis

1 Formel für die Selektionsphase eines UCT-Agenten 10
2 Gewinn für Gelb 15
3 Ausgangsformation eines Othello-Spielbretts 16
4 Auswirkungen eines Zuges auf das Othello-Spielbrett 16
5 fig:wrapfig 18
6 Ludii Nim Konfiguration mit 5 Haufen 18
7 Gewinnraten des AlphaBeta- und UCT-Agenten in Episoden gegeneinander 21
8 Durchschnittliche Längen der von AB- und UCT-Agent gewonnenen Episoden 21
9 Vergleich von zwei Episoden Vier Gewinnt mit gleichen Eröffnungszügen 22
10 Verlauf einer Episode Vier Gewinnt 23
11 Gewinnraten des TCL- und UCT-Agenten in Episoden gegeneinander 26
12 Durchschnittliche Längen der von TCL- und UCT-Agent gewonnenen Episoden 26
13 Gewinnraten des MWRAP- und UCT-Agenten in Episoden gegeneinander 28
14 Durchschnittliche Längen der von MWRAP- und UCT-Agent gewonnenen Episoden 28
15 Gewinnraten des MCTS- und UCT-Agenten in Episoden gegeneinander 29
16 Durchschnittliche Längen der von MCTS- und UCT-Agent gewonnenen Episoden 29
17 Entwicklung der Gewinnrate des MAST-Agenten gegen Edax über verschiedene Level 31
18 Letzten zwei Züge einer Edax-MAST-Episode auf Level 1 ohne XOT Openings 32
19 Gewinnraten des TCL4- und MAST-Agenten in Episoden gegeneinander 33
20 Gewinnraten des TDNT4- und UCT-Agenten in Episoden gegeneinander 36

Abkürzungsverzeichnis

KI Künstliche Intelligenz

RL Reinforcement Learning

TD Temporal Difference

DLP Digital Ludeme Project

API Application Programming Interface

GGP General Game Playing

GDL Game Description Language

GBG General Board Game

UCT Upper Confidence bound for Trees

TDL Temporal Difference Learning

UCB1 Upper Confidence Bound 1

MCTS Monte Carlo Tree Search

MAST Move Average Sampling Technique

MARL Multi-agent Reinforcement Learning

Inhaltsverzeichnis

Erklärung **I**

Kurzfassung **II**

Tabellenverzeichnis **III**

Abbildungsverzeichnis **IV**

Abkürzungsverzeichnis **V**

1 Einleitung **1**
1.1 Motivation . . . 2
1.2 Ludii Game Playing System . . . 3
1.3 General Board Game Playing and Learning Framework . . . 4
1.4 Ludii-GBG Schnittstelle . . . 6
1.5 Verwandte Arbeiten . . . 6

2 KI-Agenten **8**
2.1 Monte Carlo Tree Search . . . 8
2.2 Agenten des Ludii Game Systems . . . 10
2.2.1 Upper Confidence Bound for Trees (UCT) . . . 10
2.2.2 Biased MCTS (Uniform Playouts) . . . 11
2.2.3 Move Average Sampling Technique (MAST) . . . 11
2.3 Agenten des GBG-Frameworks . . . 12
2.3.1 AlphaBeta . . . 12
2.3.2 Edax . . . 12
2.3.3 Bouton . . . 12
2.3.4 TD-N-Tupel-basierte KI-Agenten . . . 13
2.3.5 MCTS . . . 14

3 Spiele **15**
3.1 Vier Gewinnt . . . 15
3.1.1 Spielstrategie . . . 15
3.2 Othello . . . 15
3.2.1 XOT Openings . . . 17
3.3 Nim . . . 18
3.3.1 Konfigurationen . . . 18

4 Durchführung und Agenten-Evaluation **19**
4.1 Vier Gewinnt . . . 19
4.1.1 Durchführung . . . 19

4.1.2 AlphaBeta und AlphaBeta-DL 20
4.1.3 TCL 26
4.1.4 TCL mit MCTS-Wrapper 27
4.1.5 MCTS 29
4.2 Othello 30
4.2.1 Agenten 31
4.2.2 Edax 31
4.2.3 TCL 33
4.3 Nim 34
4.3.1 Agenten 35
4.3.2 Bouton 35
4.3.3 TDNT4 35

5 Zusammenfassung und Ausblick **38**

6 Literaturverzeichnis **39**

Anhang **44**

1 Einleitung

Künstliche Intelligenz (KI) als Feld der Wissenschaft ist bereits seit circa der Mitte des 20. Jahrhunderts von großem Interesse für Forscher [1]. Die offizielle Einführung der Bezeichnung KI fand im Jahr 1956 in dem Dartmouth-Workshop statt, welcher auch den Zeitpunkt der Etablierung der Künstlichen Intelligenz als akademisches Fachgebiet kennzeichnet [2]. Das Meistern von Strategie-Spielen seitens eines Computers ist fast genauso lange Thema. Noch vor dieser Formalisierung begann Arthur Samuel, bekannt als einer der Gründerväter der KI, im Jahr 1952 ein lernfähiges Computer Programm für das Spiel Dame zu entwickeln, welches seine eigene Bewertungsfunktion lernt und letzten Endes den eigenen Entwickler im Spiel besiegen konnte [2][3][4]. Dieses Programm gelte auch als die „erste bedeutende Anwendung“ des Reinforcement Learning (RL) [2]. Ungefähr 45 Jahre später gelingt es Deep Blue, einem Schachprogramm von IBM, im Jahr 1997 den Schachweltmeister Gary Kasparov zu besiegen [5].

Spiele stellen eine gute Grundlage für das Erforschen und Verbessern künstlicher Intelligenz. KI-Agenten müssen ihr Lernvermögen und ihre „Intelligenz“ im Spiel gegen menschliche Gegner unter Beweis stellen. Je größer der Zustandsraum eines Spiels, desto komplexer ist es. Ein Zustandsraum beschreibt im Beispiel von Brettspielen die Anzahl an möglichen, regelkonformen Kombinationen von Positionen für Spielsteine auf dem Spielbrett, welche von der Ausgangsposition aus erreichbar sind. Allein bei einem „simplen“ Spiel wie Tic-Tac-Toe mit nur neun Spielfeldern, können 1000 verschiedene Zustände erreicht werden. Für Schach beträgt die sogenannte Zustandsraum-Komplexität bereits 10^{43} [6].

Mit Deep Blue ist zwar ein großer Fortschritt für die KI zu verschreiben. Dennoch handelte es sich um ein Expertensystem, das unter anderem Zugriff auf ein Eröffnungsbuch mit etwa 4000 Positionen und einer Datenbank mit 700.000 Großmeisterspielen, aus denen einstimmige Strategie-Empfehlungen entnommen werden konnten [7]. Ein Eröffnungsbuch umfasst eine Vielzahl an Zuständen, die das Spielbrett in Folge von möglichen Zügen ab dem Eröffnungszug des Spiels annehmen kann, die vorteilhaft für den KI-Agenten sind. Sollte beispielsweise das durch den Eröffnungszug des Gegners entstandene Spielbrett im Eröffnungsbuch des KI-Agenten enthalten sein, so steht diesem sofort sein nächster Zug zur Verfügung. Deep Blue hat das Schachspiel also mit Hilfestellungen dank Experteninformationen gelernt.

Dies führt zu der Frage nach KI-Agenten, welche auch ohne Hilfestellungen und nur mit den jeweiligen Spielregeln ausgestattet, eine Menge an Spielen einer Spielkategorie, wie Brettspiele, spielen kann. Aus ähnlichen Überlegungen heraus entstand,

lange vor Deep Mind, die Idee des General Game Playing (GGP).

„General game players are systems able to accept descriptions of arbitrary games at runtime and able to use such descriptions to play those games effectively without human intervention. In other words, they do not know the rules until the games start.“[8]

Bereits im Jahr 1968 stellte Jacques Pitrat seine Realisierung eines „General Game-Playing Program“ vor [9]. Es sollte mehrere Spiele spielen können, nur mit dem Wissen über die jeweiligen Spielregeln ausgestattet [9]. Die Spielregeln sollten mittels einer einheitlichen Sprache beschrieben werden, deren Syntax für die Beschreibung möglichst vieler Spiele geeignet ist [9]. In den 90er Jahren präsentierte Barney Pell eines der ersten GGP Systeme, MetaGamer [10]. Das MetaGamer-System sollte in der Lage sein, die Regeln eines Spiels entgegenzunehmen und mit Hilfe von diesen ein „spezialisiertes Programm“ als Spieler zurückgeben [10]. Die Spielregeln mussten im Format einer vordefinierten Sprache für das System beschrieben werden [10, S. 1379].

Seit 2005 wird weitgehend die Game Description Language (GDL) [11] als der akademische Standard zur Beschreibung von Spielen in GGP-Systemen angesehen. Die GDL ist eine regelbasierte Sprache für das Definieren diskreter Spiele mit vollständiger Information [12]. Ein Spiel wird in Form von Fakten über seine Zustände und Regeln definiert [11, S. 2].

Berühmte Beispiele für general game-playing KI-Agenten sind AlphaGo Zero [13] und AlphaZero [14], Nachfolger von AlphaGo [15]. Während AlphaGo unter anderem mittels Supervised Learning anhand von Experten-Partien das Go-Spiel gemeistert hat, erreichten AlphaGo Zero und AlphaZero ihre „übermenschliche“ Spielstärken in den Spielen Go [13][14], Schach [14] und Shogi [14] ohne jegliches Domänenwissen.

1.1 Motivation

Spielspezifische KI-Agenten, wie AlphaGo, und Expertensysteme, wie DeepBlue, besitzen general game-playing KI-Agenten gegenüber einen klaren Vorsprung beim Lernen von Spielstrategien. Deswegen werden sie gerne als Messlatte für die Spielstärke von general game-playing KI-Agenten verwendet. Ein Beispiel hierfür ist, dass AlphaZero in Partien gegen die nach Stand der Technik stärksten spielspezifischen KI-Agenten für die jeweiligen Spiele getestet wurde [14, S. 4].

Eines der neuesten GGP-Systeme ist das Ludii General Game System [16] mit einer Reihe an general game-playing KI-Agenten. In dieser Arbeit soll die Spielstärke einiger dieser KI-Agenten gemessen werden, indem sie gegen für die jeweiligen Spiele starke oder perfekte KI-Agenten antreten. Ziel ist die Beantwortung der folgenden

Frage:

`Wie weit entfernt von starkem oder perfektem Spiel sind die general game-playing KI-Agenten des Ludii General Game Systems?`

Diese soll am Beispiel der Spiele Vier Gewinnt, Othello und Nim erörtert werden, indem für jedes Spiel ein KI-Agent des Ludii Game Systems gegen starke oder perfekte KI-Agenten spielen wird. Diese werden von dem General Board Game (GBG) Playing and Learning Framework zur Verfügung gestellt. Unter ihnen befinden sich der perfekte Bouton-Agent für Nim und die sehr starken AlphaBeta- und Edax-Agenten für jeweils Vier Gewinnt und Othello.

Es ist naheliegend, dass der Gewinnerfolg der Ludii-Agenten gegenüber ihren spielspezifisch perfekten oder besonders starken Gegnern klein ausfallen könnte. Um die Spielstärke der Ludii-Agenten besser einordnen zu können, werden sie zu Vergleichszwecken auch gegen andere KI-Agenten des GBG eingesetzt.

Die Spielstärke der Ludii-Agenten wird in erster Linie an ihrer erzielten Gewinnrate festgemacht. Je nach Spiel und auffallender Besonderheiten werden ausgewählte Episoden genauer betrachtet und das Vorgehen der KI-Agenten ausgewertet. Für die Spiele Vier Gewinnt und Othello wird die Performanz der KI-Agenten hinsichtlich Spielbretter, auf denen, abweichend von ihrer eigentlichen Ausgangsdarstellung, bereits zufällig gesetzte Spielsteine vorhanden sind, verglichen.

Gegen den sehr starken Edax-Agenten (Kapitel 2.3.2) kann mit steigenden Schwierigkeitsstufen (Leveln) gespielt werden. Die Entwicklung der Gewinnrate des Ludii-Agenten für mehrere Level wird verfolgt.

1.2 Ludii Game Playing System

Das Ludemic Game Playing System [16] ist Teil des Digital Ludeme Project (DLP), ein von Cameron Browne geleitetes Forschungsprojekt der Universität zu Maastricht [17]. Ziel dieses Projekts ist es, die 1000 einflussreichsten Strategie-Spiele der Weltgeschichte zu modellieren und in einer spielbaren Datenbank zusammenzufassen. Diese soll dann für die Analyse der Entwicklung und Verbreitung von Spielen im Verlauf der menschlichen Geschichte, sowie in Beziehung zu den Kulturen gesetzt werden, in denen sie gespielt wurden.

Im Zuge dessen wurde das Ludii General Game System entwickelt, ein Nachfolger des Ludi General Game Systems, das von Cameron Browne für seine Doktorarbeit gebaut wurde. Ludii hebt sich von anderen General Game Playing Systemen ab, da es anstatt der beliebten GDL eine eigene Struktur zum Definieren der Spiele nutzt. Piette et al. stellen sogenannte Ludeme als die Bausteine der von ihnen definierten

Sprache zur Spielbeschreibung vor, welche als konzeptuelle Einheiten spiel-bezogener Informationen beschrieben werden [16][18]. Auf diese Weise sollen Spielbeschreibungen präzise und auch für Menschen leicht verständlich sein. In Ludii entsteht jedes Spiel also aus der Zusammenstellung mehrerer Ludeme, welche jeweils verschiedene Aspekte eines Spieles beschreiben, wie z.B. die Spielregeln.

Hauptgrund für die Entwicklung des Ludii Game Systems sind die Anforderungen die die Forschung im Rahmen des DLPs stellt, welche von GGP Systemen unzureichend erfüllt seien [16]. Ludii solle gegenüber diesen viele Vorteile bieten. Für beispielsweise die Implementierung einer großen Varietät an Spielen und Spielarten sei eine Allgemeinheit und Erweiterbarkeit von einem Game System und folglich auch seiner Spielbeschreibungssprache gefordert, die mit GDL und auch die Regular Board Games (RGB) Sprache [19] nicht geboten sei [16]. Allgemeinheit wird mit Blick auf die Vielfalt an Spielarten beschrieben, die ein Game System unterstützt, ohne dass Erweiterungen der Spielbeschreibungssprache benötigt werden [16]. So mussten für GDL bereits zwei Erweiterungen, GDL-II [20] für Spiele mit versteckten Informationen und GDL-III [21] für epistemische Spiele, entwickelt werden. Im Falle, dass doch mal eine Erweiterung durch eine neue Funktionalität anfällt, spielt die Erweiterbarkeit des Systems eine wichtige Rolle, um den hierfür nötigen Aufwand zu bewerten. Für Ludii mit seiner Spielbeschreibungssprache bedeutet eine Erweiterung das Hinzufügen neuer Klassen zur von dem Game System definierten Ludeme-Bibliothek [16].

Mittlerweile umfasst das Ludii Game System bereits über 1000 Spiele und Spielvarianten. Nutzer können das volle Angebot der Spiele gegen andere Menschen oder eine Auswahl von KI-Agenten spielen. Für Entwickler ist es ebenfalls möglich, eigene Spiele basierend auf der Ludeme Struktur oder eigene KI-Agenten mittels eines Application Programming Interface (API) zu implementieren.

1.3 General Board Game Playing and Learning Framework

Das GBG Playing and Learning Framework ist ein von Wolfgang Konen an der Technischen Hochschule Köln geleitetes Open-Source Projekt, das seit der Veröffentlichung im Jahre 2018 von Konen, seiner Forschungsgruppe und auch Studierenden in diversen Projekten und Arbeiten erweitert wurde [22]. Es handelt sich um ein Java-Framework, welches die gemeinsamen Prozesse der Spielentwicklung und des Game Learning standardisiert. Wie der Name bereits andeutet, liegt dabei der Schwerpunkt auf Brettspielen. Diese können deterministisch sowie nicht-deterministisch sein. Konen definiert ein Brettspiel als ein Spiel mit bekannter, beliebiger Teilnehmeranzahl, bei dem Züge nacheinander ausgeführt werden und, dass auf einem Brett oder Tisch stattfindet [22]. Kartenspiele werden ebenfalls mit in diese Definition aufgenommen.

Das GBG-Framework soll in erster Linie Studierenden den Einstieg in die Bereiche des Game Learning und der KI-Agenten-Entwicklung erleichtern [22]. Der Arbeitsaufwand für die grundlegende Spiel- sowie Agenten-Entwicklung sei im Normalfall zu hoch für die begrenzte Zeit, die interessierten Studierenden für Projekte und Abschlussarbeiten zustehen. Daher biete das GBG-Framework Schnittstellen und abstrakte Klassen für eine zeiteffiziente Implementierung neuer Spiele oder auch KI-Agenten, sodass Studierende sich mit den wesentlichen Aspekten ihrer Forschung befassen können. Mit der Ausnahme weniger, spielspezifischer KI-Agenten können alle im GBG-Framework enthaltenen KI-Agenten auf Spiele, die Gebrauch von diesen Schnittstellen machen, angewendet werden [22].

Anhand der hierdurch wachsenden Auswahl an KI-Agenten und Spielen für Wettbewerbe lassen sich auch bessere Performanz-Vergleiche bezüglich der tatsächlichen Stärke eines KI-Agenten durchführen. Somit sei auch die Forschung nach vielseitigen und starken KI-Agenten, die allgemein für viele verschiedene Spiele einsetzbar sind, erleichtert [22].

Wie bereits angedeutet, umfängt das GBG-Framework auch einige spielspezifisch starke oder sogar perfekte KI-Agenten. Diese sollen dem Evaluieren der Performanz von KI-Agenten dienen, die die Strategien eines Spiels ausschließlich über sogenanntes „self-play“, also durch das Spielen von einer Menge an Partien gegen Instanzen von sich selbst, gelernt haben. Auf diese Weise solle bewertet werden können, wie nah oder weit ein selbst-lernender KI-Agent von starker oder auch perfekter Performanz liegt [22]. Der Begriff „self-play“ stammt aus dem Bereich des Multi-agent Reinforcement Learning (MARL), einer Untergruppe des RL, die sich mit dem Verhalten mehrerer RL-Agenten in einer gemeinsamen Umgebung befasst. AlphaGo Zero [13] und AlphaZero [14] von Google DeepMind sind solche mittels self-play lernenden KI-Agenten.

Ebenfalls wird ein Inspektionsmodus angeboten, über den die Bewertung gewünschter KI-Agenten für einen Spielstand und seine Optionen für den nächsten Zug abgerufen werden können [22]. Dies bietet Einblick in die Genauigkeit der Werte, über die ein KI-Agent seinen nächsten idealen Zug wählt.

Abgrenzung zu GGP Auch wenn die Forschungsbereiche sich überlappen mögen, so gilt es dennoch, zwischen GBG und GGP zu differenzieren. Das GBG zielt, ähnlich zum Ludii Game System, darauf, einige der Begrenzungen des GGP und der GDL zu umgehen [22].

So sind unter anderem Spielsimulationen, welche von bestimmten KI-Agenten zum

Suchen eines geeigneten Zuges verwendet werden, die auf der Beschreibung der GDL basieren, langsamer als die Simulationen der kompilierten GBG-Spiel-Engine, welche 10.000 bis 90.000 Züge pro Sekunde für Temporal Difference (TD)-N-Tupel Agenten ermöglicht [22].

Spiele im GBG-Framework seien dank dem Einpflegen von beispielweise spielspezifischen Symmetrien des Spielbretts auch performanter als in GGP-Systemen, welche mit einer „suboptimalen Performanz für ihre Allgemeinheit zahlen.“ [22] Dennoch biete das GBG-Framework nicht dieselbe Allgemeinheit wie GGP-Systeme. Während es in diesen möglich sei, neue Spiele zur Laufzeit einzuführen, müsse dies im GBG-Framework schon zur Kompilierzeit erfolgen [22].

1.4 Ludii-GBG Schnittstelle

Die Spiele werden innerhalb der Ludii-Umgebung ausgetragen. Dies geschieht mit Hilfe einer Schnittstelle zwischen den zwei Systemen, welche in vorherigen Projekten implementiert [23][24] und erweitert [25] wurde. Diese nutzt die von Ludii gebotene AI-Klasse, welche für die Implementierung und Anwendung eigener KI-Agenten im Ludii Game System gedacht ist. Die Schnittstelle ermöglicht es, die im GBG angebotenen KI-Agenten dem Ludii Game System als Pseudo-Ludii-Agenten vorzustellen. In Wirklichkeit übersetzt dieser jedoch nur die Spielstände in ein für den GBG-Agenten verständliches Format und fragt ihn nach seinem nächsten Zug. Dieser Zug wird dann wieder in das Ludii-Format zurückübersetzt und vom Pseudo-Agenten gespielt. Für den Die folgenden Spiele können derzeit zwischen Ludii- und GBG-KI-Agenten ausgetragen werden:

- Yavalath [24]
- Othello [23][24]
- Vier Gewinnt und Nim [25]

1.5 Verwandte Arbeiten

Im Rahmen eines Informatikprojekt an der Technischen Hochschule Köln vergleicht Johannes Scheiermann trainerte general-purpose RL-Agenten des GBG-Frameworks mit untrainierten General-Game Playing Agenten des Ludii Game Systems [23]. Da der Schwerpunkt von Scheiermanns Arbeit auf dem Aufbau der Schnittstelle zwischen dem GBG und Ludii beruht hat, beschränkt sich ihre Datenerhebung auf das Auswerten von 100 Episoden zwischen einem TD3-Tupel-Agenten (FUSSNOTE, Stand 23.07.2020) auf der Seite des GBG und einem nicht-spezifizierten Ludii-KI-Agenten. Jeder KI-Agent spielte in 50 Episoden den ersten Zug. Den Ergebnissen zufolge habe der Ludii-KI-Agent mehr Partien für sich bestimmt [23]. Aufgrund von

Problemen bezüglich der GBG-Ludii-Schnittstelle, welche noch in ihrem Anfangsstadium war, werden jedoch gewisse Partien auch als ungültig erklärt und Scheiermann stellt die Vermutung auf, dass der TD3-Tupel-Agent unter korrekten Spielbedingungen mehr Partien hätte gewinnen können. Somit seien die Ergebnisse nicht aussagekräftig bezüglich der tatsächlichen Stärke des TD3-Tupel-Agenten [23].

Eine andere Arbeit konzentriert sich mehr auf das Evaluieren spezifischer GBG-Agenten in der im Vergleich zu Ludii-Agenten. So wertet Ann Weitz in einer Bachelorarbeit [26] den Lernerfolg einer Reihe von unterschiedlich parametrisierten RL Agenten des GBG-Frameworks für das Spiel Yavalath aus, indem sie diese gegen den Biased MCTS Agenten des Ludii Game Systems antreten lässt [26]. Der mit den aus den Ergebnissen der Arbeit hervorgegangenen Parametern trainierte TCL3-Agent gewinnt 78,5% der Episoden gegen den Biased MCTS des Ludii Game Systems [26].

2 KI-Agenten

Im folgenden Kapitel werden die in dieser Arbeit zu evaluierenden KI-Agenten des Ludii Game Systems, sowie ihre gegnerischen KI-Agenten im GBG-Framework, vorgestellt und ihre Funktionsweisen kurz erläutert.

2.1 Monte Carlo Tree Search

Der Großteil der im Ludii Game System verfügbaren KI-Agenten basiert auf dem Monte Carlo Tree Search (MCTS)-Algorithmus und auch alle in dieser Arbeit verwendeten Ludii-Agenten sind Varianten dieses Algorithmus. Im folgenden Abschnitt wird daher die Vorgehensweise des MCTS kurz wiederholt, um Begrifflichkeiten aufzufrischen.

Beim MCTS handelt es sich um einen heuristischen Suchalgorithmus. Er baut iterativ einen Suchbaum auf. Im Kontext von Zuständen eines Spiels wird dieser auch Spielbaum genannt. Die Baumstruktur wird „Knoten für Knoten“ aufgebaut [27]. Im Kontext eines Brettspiels repräsentiert jeder Knoten jeweils einen Zustand des Spielbretts. Der Zug des Spielers der aus diesem Zustand heraus an der Reihe ist führt zu einem Folgezustand, welcher durch Kindknoten dargestellt wird.

Für Spiele mit hoher Komplexitat, also mit einem großen Zustandsraum, kann der Spielbaum einen sehr hohen Verzweigungsgrad aufweisen. Oft ist die hinreichende Suche nach einer optimalen Lösung aufgrund von begrenzter Zeit und Rechenleistung nicht möglich [28] oder sehr langsam, da eben so viele Folgezustände untersucht werden müssen.

Der MCTS-Algorithmus grenzt den zu prüfenden Zustandsraum stark ein, indem er dank einer vordefinierten Strategie (Policy) in erster Linie die bisher vielversprechenden Pfade untersucht [28]. Allerdings lässt er die anderen Pfade nicht komplett außer Acht, da es durchaus möglich ist, dass ein Knoten beziehungsweise Zustand zuerst nachteilig erscheint, sich aber bei weiterer Untersuchung ein paar Züge später ein sehr vorteilhafter Folgezustand ergibt. Die daraus folgende Bewertung des Pfades zu diesem Knoten könnte letztlich viel besser als die bisherigen ausfallen.

Jede Iteration des MCTS wird in vier Phasen unterteilt.

Phase 1: Selektion
Die Selektion beginnt jedes Mal beim Wurzelknoten des aufgebauten Suchbaums. Ausgehend von diesem wird der Baum traversiert bis ein Knoten ohne Kindknoten erreicht wird. Für die Entscheidung, an welchen Pfaden entlang traversiert wird, ist eine sogenannte Child Selection Policy oder auch Tree Policy zuständig. Jeder Knoten enthält Informationen über die Menge an Iterationen, in denen er bereits besucht

wurde. Außerdem speichert er auch eine Art „Erfolgsstatistik“ [28] für den Knoten in Bezug auf diese Iterationen. Diese gibt Auskunft über den Gewinnerfolg des Spielers, der in dem vom Knoten repräsentierten Zustand am Zug ist. Anhand dieser Informationen bestimmt die Tree Policy, zu welchem Kindknoten beziehungsweise Folgezustand, weiter-traversiert wird.

Exploration versus Exploitation „Beim vorzeitigen Abbruch der Berechnung nach einem bestimmten Zeitraum entstehen zwei zentrale Probleme: zum ersten ist der bis dahin durchsuchte Bereich eventuell noch nicht richtig bewertet, zum zweiten gibt es noch große Bereiche im Zustandsraum, die überhaupt nicht begangen worden sind.“ [28]

Wie bereits aufgeführt, versucht der MCTS eine Balance zu wahren zwischen dem Ausnutzen von gut-bewerteten Pfaden und dem Erforschen neuer Pfade, die zu besseren Zuständen führen könnten. Dieses bestreben wird als Exploration-Exploitation Trade-Off bezeichnet. Die Tree Policy, welche die zu erforschende Kindknoten und somit Folgezustände bestimmt, ist daher von zentraler Bedeutung.

Phase 2: Expansion
Sollte der erreichte Knoten keinen Endzustand darstellen und noch nicht komplett expandiert sein, so wird in diesem Schritt um einen weiteren Konten expandiert. Es wird für den erreichten Knoten eine noch unbekannte Aktion gewählt und ein neuer Knoten als Kind an den Spielbaum angehängt. Dieser repräsentiert den aus der gewählten Aktion resultierenden Zustand. Je nach Implementierung des Algorithmus kann auch um mehrere neue Knoten expandiert werden.

Phase 3: Simulation
Dieser Schritt wird auch oft als Rollout oder Playout bezeichnet. In diesem Schritt wird von dem neuen Knoten aus eine Simulation des Spiels ausgeführt, bei der zufällige Züge gespielt werden bis ein Endzustand erreicht worden ist. Für diese Züge und die entstehenden Zustände werden keine Knoten. Stattdessen wird im expandierten Knoten das Endergebnis der Simulation aus Sicht des Spielers, der im vom Knoten repräsentierten Zustand am Zug ist, festgehalten. Ebenso wird die Anzahl der Iterationen in denen dieser Knoten durchlaufen wurde gespeichert. In dem Fall eines neu expandierten Knotens ist dies natürlich nur die aktuelle Iteration.

Die simulierten Züge müssen jedoch nicht rein zufällig gewählt werden. Die Nutzung einer Rollout Policy (auch Playout Policy genannt) zum Beeinflussen der gewählten Züge während der Simulation ist ebenfalls möglich. Hierfür werden beispielsweise spielspezifische Heuristiken verwendet [29].

Phase 4: Rückpropagierung
Durch das Resultat der Simulation müssen nun alle Knoten auf dem Pfad vom Wurzelknoten bis zum expandierten Knoten mit den neuen Statistiken aktualisiert werden. Die Zahl an durchlaufenen Iterationen erhöht sich für alle besuchten Knoten und ihre Gewinnstatistik wird je nach dem Spieler in dem Zustand, den der jeweilige Knoten darstellt, am Zug ist. Denn wenn die Simulation über den expandierten Knoten mit einem Gewinn für den einen Spieler endet, so muss sie für den anderen Spieler einen Verlust bedeuten. Verluste oder Gleichstände gehen ebenfalls in die Erfolgsstatistik mit ein. Das Ergebnis der Simulation wird also bis zum Wurzelzustand rückpropagiert, um die Knoteninformationen anzupassen und auf die nächste Iteration vorzubereiten.

2.2 Agenten des Ludii Game Systems

In diesem Abschnitt werden die general game-playing KI-Agenten des Ludii Game Systems vorgestellt, die zu evaluieren sind. Das Ludii Game System bietet in seiner Auswahl von KI-Agenten auch die Option an, automatisch den für das jeweilige Spiel stärksten KI-Agenten verwenden zu lassen. Das Ludii Game System definiert neben den Spielregeln und dem Spielaufbau auch weitere Informationen zu den Spielen im Ludeme-Format. Die gesamten Informationen zu einem Spiel sind in ihren Metadaten vermerkt. Für einige der Spiele ist in den Metadaten des Spiels der am besten geeignete KI-Agent vermerkt und wird von der sogenannten „LudiiAI()“ automatisch aufgerufen. Es werden für die in dieser Arbeit behandelten Spiele die in ihren Metadaten vermerkten KI-Agenten verwendet.

2.2.1 Upper Confidence Bound for Trees (UCT)

Beim Upper Confidence bound for Trees (UCT) handelt es sich im Allgemeinen um eine Variante des MCTS. Der Name des Algorithmus basiert auf der Bezeichnung seiner Tree Policy, der Upper Confidence Bound 1 (UCB1) Formel [30] zur Bewältigung von „Multi-Armed Bandit“-Problemen. Sie wurde von Kocsis und Szepesvari [31] abgeändert und als Tree Policy für ihren UCT-Algorithmus vorgestellt.

$$UCT = \overline{X_j} + 2C_p\sqrt{\frac{2 * lnn}{n_j}}$$

Abbildung 1 Formel für die Selektionsphase eines UCT-Agenten

- n ist die Anzahl an Iterationen, in denen der aktuelle Knoten n besucht wurde

- n_j ist die Anzahl an Iterationen, in denen der Kindknoten n_j von n besucht wurde
- $\overline{X_j}$: durchschnittliche Belohnung des Kindknoten n_j (Erfolgsstatistik über bisherige Iterationen hinweg)
- C_p ist eine Konstante und wird genutzt, um den Grad an Erforschung bei der Wahl des Kindknoten anzupassen

Der UCT-Agent des Ludii Game Systems verwendet die Standard Implementation der UCB1 Formel mit einer Erforschungskonstante $C = \sqrt{2}$ als Tree Policy in der Selektionsphase. Auch wenn es noch nicht besuchte Kindknoten gibt, können bereits besuchte Kindknoten nochmal gewählt werden.

In der Expansionsphase wird immer nur um einen Knoten expandiert. Die Züge in der Simulationsphase werden zufällig gewählt und es dürfen maximal 200 Züge pro Playout, also pro Simulation, gemacht werden. Rückpropagiert wird mittels Monte Carlo Backpropagation. Letzten Endes wird der Kindknoten der Wurzel, mit der höchsten Iterationszahl gewählt, also das „robuste Kind“ [16]. Dem UCT-Agenten ist es erlaubt, relevante Teile seines aufgebauten Suchbaums des vorherigen Zugs, wieder zu verwenden.

2.2.2 Biased MCTS (Uniform Playouts)

Der Biased MCTS-Agent mit Uniform Playouts verwendet, wie der Name bereits andeutet, ebenfalls eine Variante des MCTS-Algorithmus.

In der Selektionsphase wird dieselbe PUCT Strategie wie die von AlphaGo Zero verwendet [13]. Um diese zu beeinflussen wird eine Softmax Funktion verwendet, die einen Logit-Vektor als Input erhält. Jedes State-Action Paar wird von einem Logit repräsentiert, das aus dem Skalarprodukt des Gewicht-Vektors und des Feature-Vektors für dieses State-Action Paar berechnet wird. Diese Vektoren, wenn für das Spiel definiert, werden aus den Metadaten des Spiels entnommen.

Während der Simulationsphase werden zufällige Züge durchgespielt. Dem KI-Agenten sind maximal 200 Züge pro Simulation erlaubt. Auch hier wird Monte Carlo Backpropagation angewendet, um die Bewertungen zurückzureichen und am Ende das robuste Kind für den nächsten Zug gewählt.

2.2.3 Move Average Sampling Technique (MAST)

Der MAST-Agent benutzt, wie der UCT-Agent auch, die UCB1 Tree Policy für seine Selektionsphase. In der Simulationsphase verwendet er allerdings die Move Average Sampling Technique (MAST) zur Optimierung der Wahl der zu simulierenden Züge.

Diese werden also nicht zufällig gewählt. Wie die anderen KI-Agenten, verwendet er die Monte Carlo Backpropagation zum Rückpropagieren der Bewertungen und wählt ebenfalls den „robusten“ [16] Kindknoten.

2.3 Agenten des GBG-Frameworks

In diesem Abschnitt werden die im GBG-Framework angebotenen KI-Agenten vorgestellt, gegen die die Ludii-Agenten spielen werden.

2.3.1 AlphaBeta

Der AlphaBeta-Agent, implementiert von Thill et al. [32], basiert auf dem Minimax-Suchbaum-Algorithmus. Der Minimax-Algorithmus allein ist sehr stark, jedoch betrachtet er mit einem Tiefensuche-Ansatz rekursiv den gesamten Suchbaum und somit den vollständigen Zustandsraum der gegebenen Umgebung. Dies braucht je nach Zustandsraum-Komplexität der Umgebung, also eines Spiels, viel Rechnerleistung und viel Zeit. Zusätzlich zur Minimax-Suche wird daher AlphaBeta-Pruning angewendet, um den Suchbaum zu trimmen und somit den zu untersuchenden Zustandsraum zu verkleinern.

Zur weiteren Optimierung ist der AlphaBeta-Agent mit einem vorbereiteten Eröffnungsbuch ausgestattet [32]. Der AlphaBeta-DL-Agent unterscheidet sich vom AlphaBeta-Agenten nur in einer Hinsicht. Wenn der AlphaBeta-Agent erkennt, dass er einem Verlust nicht ausweichen kann, wählt er seine Züge zufällig. Wenn der AlphaBeta-DL erkennt, dass er verlieren wird, versucht er diesen Verlust soweit wie möglich zu verzögern (Distant Losses). Wenn der gegnerische KI-Agent dann unerwartet einen Fehler macht, kann der AlphaBeta-DL-Agent seinen Verlust sogar in einen Gewinn umwandeln.

2.3.2 Edax

Bei Edax [33] handelt es sich um ein von Richard Delorme entwickeltes, starkes Othello-Programm mit unterschiedlichen Leveln. Er verwendet unter anderem ebenfalls die AlphaBeta-Suche. Die Level entsprechen der Suchtiefe des Suchbaums.

2.3.3 Bouton

Der Bouton-Agent basiert auf der Gewinnstrategie nach Bouton [34], welche nach ihrem Erfinder Charles L. Bouton benannt ist. Diese beschreibt, ob und wie ein Spieler mithilfe von Rechnungen für den aktuellen Spielstand einen Gewinn erzwingen kann. Ihre Verwendung macht den Bouton-Agenten zu einem perfekten KI-Agenten im Nim-Spiel. Er gewinnt jedes Spiel, dass er als Spieler 1 spielt und auch als Spieler 2, sofern der gegnerische KI-Agent nicht perfekt spielt.

2.3.4 TD-N-Tupel-basierte KI-Agenten

Der TD-N-Tupel-Agent verwendet das Temporal Difference Learning (TDL) in Verbindung mit N-Tupel Systemen [35] [36], um seine Wertfunktion zu lernen. In seiner „einfachen“ Form wird er nur für das Nim-Spiel eingesetzt. Des Weiteren wird er in dieser Arbeit als TDNT4-Agent bezeichnet.

TCL-Agenten

Die in dieser Arbeit verwendeten TCL-Agenten wurden der Arbeit von Scheiermann und Konen [37] entnommen. Ihre genaue Vorgehensweise und die Wahl der Agenten-Parameter wird in [37] genauer beschrieben.

Tabelle 1 Hauptparameter des TCL-Agenten

Lernrate α	3,7
Eligibility Trace Faktor δ	0,0
Explorationsrate ϵ	0,1 $\rightarrow$ 0,0
Trainierte Episoden	250.000

TCL-Agent mit MCTS-Wrapper

Inspiriert durch AlphaZero [14], wurde dieser KI-Agent von Scheiermann und Konen [37] implementiert. Es handelt sich um einen TD-FARL Agenten [35] [36] mit Temporal Coherence Learning [38] [32], welchem ein MCTS-Wrapper hinzugefügt wurde [37]. Bei dieser Vorgehensweise wird das TD N-Tupel Netz erst allein trainiert und zur tatsächlichen Spielzeit wird die Monte Carlo Baumsuche angewendet, um dem KI-Agenten das vorausschauende Spielen und Planen seiner Züge zu ermöglichen. Den Ergebnissen aus [37] zufolge sei der TCL-Agent mit MCTS-Wrapper der erste ohne TPU oder GPU trainierte KI-Agent, der Edax bis inklusive Level 7 besiegt. Im weiteren Verlauf dieser Arbeit wird die Variante dieses KI-Agenten für Othello als TCL4-Agent bezeichnet. Seine Hauptparameter sind in Tabelle 2 angegeben.

Tabelle 2 Hauptparameter des TCL4-Agenten

Lernrate α	0,2
Eligibility Trace Faktor δ	0,5
Explorationsrate ϵ	0,2 $\rightarrow$ 0,1
Trainierte Episoden	250.000

Es wurde ebenfalls ein TCL-Agenten mit MCTS-Wrapper für Vier Gewinnt trainiert [37]. Dieser wird ab hier als MWRAP-Agent bezeichnet. Seine Hauptparameter entsprechen denen des TCL-Agenten für Vier Gewinnt aus dem vorherigen Abschnitt.

Der MCTS-Wrapper hat die folgenden Parameter für dem TCL- sowie den TCL4-Agenten [37]:

- Iterationen = 1000
- CPUCT = 1.0
- unbegrenzte Suchtiefe

2.3.5 MCTS

Dies ist ein Standard MCTS-Agent mit 10.000 Iterationen und einer maximalen Suchbaumtiefe von 40, der die UCT Selektionsstrategie und in der Simulationsphase zufällige Playouts anwendet. Der KI-Agent wird ebenfalls als Teil der Arbeit von Scheiermann und Konen Scheiermann.2022 verwendet.

3 Spiele

Im folgenden Abschnitt werden die Spiele erläutert, in denen die KI-Agenten gegeneinander spielen sollen. Zudem werden auch an Ansätze und Aspekte der Spiele herangeführt, die für die Durchführung der Spiele in Abschnitt relevant sind.

3.1 Vier Gewinnt

Eines der Spiele anhand dessen die KI-Agenten getestet werden sollen ist das Zweipersonen-Strategiespiel Vier Gewinnt. Es handelt sich um ein deterministisches Spiel mit vollständiger Information, welches von Victor Allis gelöst wurde [39]. Dieses wurde im Jahre 1974 auf den Markt gebracht. Gespielt wird auf einem aufrechtstehenden Spielbrett, das ausgehöhlt ist. Das Spielbrett besitzt sieben Spalten und sechs Reihen, also 42 besetzbare Positionen. In diese lassen die Spieler abwechselnd ihre gelben oder roten Spielsteine fallen.

Ziel des Spiels ist es, als erstes mindestens vier Spielsteine der eigenen Farbe in eine Reihe zu bringen. Dabei ist unwichtig, ob die Reihe diagonal, waagerecht oder senkrecht entsteht. Abbildung 1 zeigt, dass der Spieler der gelben Spielsteine durch das Setzen einer diagonalen Reihe einen Gewinn erzielt. Das Spiel gilt als unentschieden, wenn keiner der Spieler eine Reihe bilden kann, bevor das Spielbrett komplett befüllt ist.

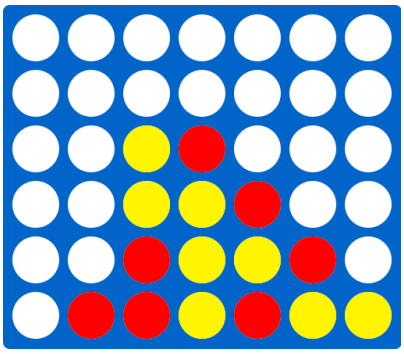


Abbildung 2 Gewinn für Gelb[1]

3.1.1 Spielstrategie

Nach Allis sei bei perfektem Spiel beider Spieler eine Partie für den ersten Spieler immer ein Gewinn, wenn er für seinen ersten Zug die mittlere Spalte des Bretts wählt [39]. Sollte er bei perfektem Spiel eine andere als die mittlere Spalte wählen, so kann der zweite Spieler immer einen Gleichstand erbringen. Eine Partie Vier Gewinnt sei demnach ein theoretischer Gewinn für den ersten Spieler.

3.2 Othello

Othello ist ebenfalls ein deterministisch ablaufendes Zweipersonen-Spiel mit vollständiger Information. Wem die tatsächliche Erfindung zuzuschreiben ist, ist nicht ganz klar. In den den 1880er Jahren wurde das Brettspiel Reversi auf den Markt gebracht, welches auf einem Schachbrett mit 64 Feldern gespielt werden sollte. Die heute bekanntere, etwas abgeänderte Version des Spiels trägt den Namen Othello

[1] Spieldarstellung über Ludii Game System

und wurde fast 100 Jahre später im Jahr 1971 vom Japaner Goro Hasegawa in Japan als Warenzeichen angemeldet.

Othello wird mit abgeflachten Spielsteinen auf einem 8x8 Felder großen Spielbrett gespielt. Die Spielsteine sind auf einer Seite schwarz und auf der anderen Seite weiß bemalt. Die mittleren vier Spielfelder des Brettes sind bereits zu Anfang des Spiels mit zwei weißen und zwei schwarzen Spielsteinen besetzt.

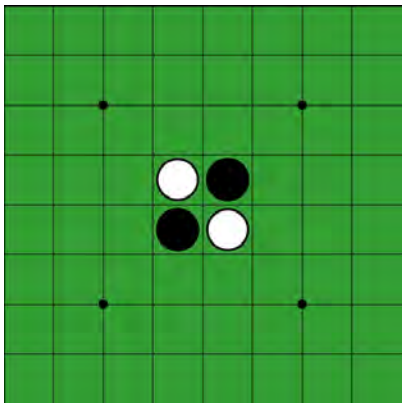
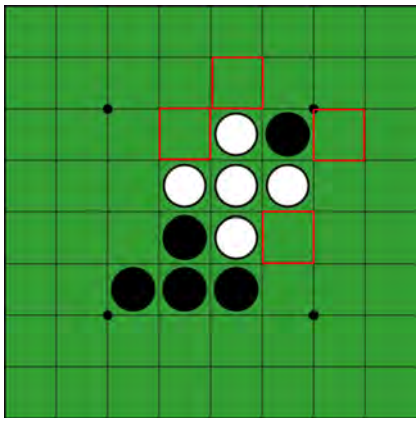
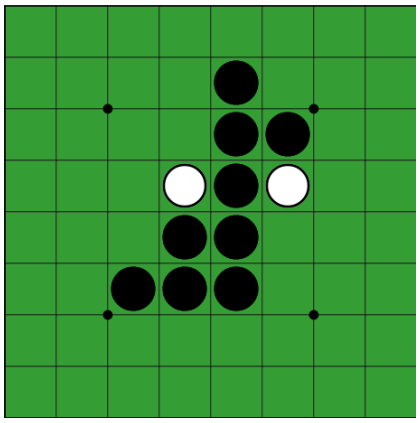


Abbildung 3 Ausgangsformation eines Othello-Spielbretts[2]

Die Spieler legen abwechselnd Spielsteine ihrer eigenen Farbe auf ein leeres Feld des Bretts. Wenn es keine regelkonforme Zugmöglichkeit gibt, ist der Spieler gezwungen zu passen. Spielsteine dürfen nur so gelegt werden, dass sie diagonal, senkrecht oder waagerecht neben einem Stein oder einer Reihe mehrerer Steine der gegnerischen Farbe liegen. Am anderen Ende dieser Reihe muss sich wieder ein eigener Spielstein befinden. Die gegnerischen Spielsteine werden also von beiden Seiten mit eigenen Steinen eingeschlossen.



(a) Zug-Optionen für Schwarz[2]



(b) Spielbrett nach Zug von Schwarz[2]

Abbildung 4 Auswirkungen eines Zuges auf das Othello-Spielbrett

Alle eingeschlossenen Spielsteine werden durch diesen Zug umgedreht, sodass sie die Farbe wechseln und nun dem Spieler gehören, der den Zug getätigt hat. Erst dann ist wieder der andere Spieler an der Reihe. Ein Beispiel zeigt die Abbildung 4. Abbildung 4a zeigt anhand der rot umrandeten Felder die möglichen Züge für Spieler Schwarz an. In Abbildung 4b sind die Folgen seines gewählten Zugs erkennbar. Alle weißen Spielsteine auf der mittleren diagonalen Linie haben sich umgedreht und sind

[2]Spieldarstellung über Ludii Game System

nun schwarz.

Ein Zug kann zum Umdrehen mehrerer Reihen gegnerischer Spielsteine führen, allerdings führen die umgedrehten Steine nicht zum Umdrehen anderer Reihen. Wenn das Spielbrett voll ist, dann gilt das Spiel als beendet. Ebenfalls ist das zu Ende, wenn beide nacheinander Spieler passen. Dies kann auch eintreten, wenn alle Steine auf dem Brett dieselbe Farbe haben, wodurch es nicht mehr möglich ist, sie umzudrehen. Gewonnen hat der Spieler mit mehr Spielsteinen in seiner Farbe auf dem Spielbrett. Sollten beide Spieler gleich viele Steine haben, endet das Spiel als unentschieden.

3.2.1 XOT Openings

Laut Matthias Berg, dem sechszehn-fachen Gewinner der deutschen Othello- Meisterschaften, haben Othello-Spiele das Potenzial monoton zu verlaufen, wenn erfahrene Spieler immer ihre stärksten Eröffnungszüge spielen [40]. Erst ab dem 30. Zug solle eine Partie zum Denken anregen [40]. Daher entwickelten Berg und der vierfache spanische Meister Moreno Borja zwei Listen mit Sequenzen von sogenannten „XOT Openings“ [40]. Dabei handelt es sich um Eröffnungszüge. Die Zugfolgen bestehen jeweils aus acht Zügen [40]. Diese sollen Positionen auf dem Spielbrett erzeugen, mit denen erfahrene Spieler nicht zwingend vertraut sind und die demnach zu Varietät im Spiel führen sollen [40].

3.3 Nim

Bei Nim handelt es sich um ein mathematisches Strategiespiel mit vollständiger Information für zwei Spieler. Der genaue Ursprung des Spiels ist nicht bekannt, jedoch erhielt es seinen jetzigen Namen von Charles L. Bouton [34]. Dieser veröffentlichte im Jahr 1902 die Lösung des Spiels [34], welche auch als Theorie von Bouton bezeichnet wird. Auf dieser Lösung beruht auch der Bouton Agent.

Das Spiel wird mit mehreren Reihen von Gegenständen, beispielsweise Streichhölzern, gespielt. Die Menge an Reihen sowie die Anzahl an Gegenständen kann dabei frei gewählt werden. Die Spieler dürfen nun abwechselnd beliebig viele Gegenstände aus einer Reihe wegnehmen, jedoch immer nur aus jeweils einer Reihe pro Zug. Je nach Variante des Spiels, ist der Spieler, der den letzten verfügbaren Gegenstand nimmt, der Gewinner oder Verlierer. In vielen Online-Auslegungen des Spiels sind es Haufen anstatt Reihen von Gegenständen, auch im Ludii Game System und dem GBG-Framework.

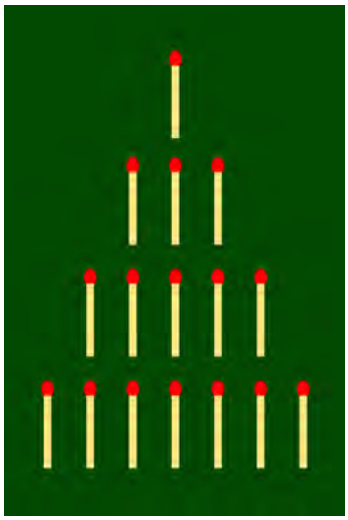


Abbildung 5 Nim mit Streichhölzern[3]

3.3.1 Konfigurationen

Das Ludii Game System bietet verschiedene Konfigurationen für Nim an. So ist es möglich eine beliebige ungerade Anzahl an Haufen für das Spiel zu wählen, jedoch sind es mindestens drei und höchstens 19 Haufen. Die Menge der Gegenstände auf jedem Haufen wird als Zahl ausgedrückt.

Die Anfangsmengen sind in jeder Konfiguration nach demselben Prinzip verteilt. Der mittlere Haufen besitzt die meisten Gegenständen, dessen Menge gleich der Anzahl an Haufen ist. Auf den sich vom mittleren Haufen nach links und rechts aufreihenden Haufen sind immer ein Gegenstand weniger als auf dem zur Mitte näheren Nachbarn.

Abbildung 6 Ludii Nim Konfiguration mit 5 Haufen

[3] https://upload.wikimedia.org/wikipedia/commons/thumb/f/f6/NimGame.svg/285px-NimGame.svg.png

4 Durchführung und Agenten-Evaluation

In diesem Abschnitt werden die Ergebnisse der Spiele, in denen die GGP KI-Agenten des Ludii Game Systems gegen die je nach Spiel stark- oder perfekt-spielenden KI-Agenten im GBG-Framework angetreten sind, präsentiert und ausgewertet.

Die Episoden für die Spiele haben fast alle denselben Grundaufbau. In den Spielen Vier Gewinnt und Othello werden jeweils pro Agenten-Paar 200 Episoden, oder Partien, gespielt. Bei 100 davon macht der Ludii-Agent den ersten Zug und für die restlichen 100 den zweiten. Um verlässlichere Ergebnisse zu erlangen, wird für diese Spiele unter anderem das Spielbrett zum Beginn des Spiels bereits mit Spielsteinen aus manuell getätigten Zügen ausgestattet sein. Diese werden in den relevanten Abschnitten näher erläutert.

Im Spiel Nim werden 100 Episoden pro Agenten-Paar gegeneinander gespielt. In 50 dieser Episoden spielt der Ludii-Agent als Spieler 1 durch und für die anderen 50 als Spieler 2.

Das Hauptkriterium für die Messung der Stärke eines KI-Agenten ist die Gewinnrate für die gespielten Episoden. Je nach Spiel, werden zusätzliche Aspekte betrachtet, die zu genaueren Aussagen über den Erfolg der KI-Agenten hilfreich sein könnten. Diese werden in den entsprechenden Abschnitte beschrieben.

Die Spiele werden mit Hilfe von gesonderten, für den Zweck dieser Arbeit erstellten Klassen, welche die GBG-Ludii-Schnittstelle nutzen, ausgetragen und dokumentiert (siehe Anhang).

4.1 Vier Gewinnt

Im folgenden Abschnitt werden die Ergebnisse der verschiedenen Agenten-Episoden für das Spiel Vier Gewinnt jeweils präsentiert und ausgewertet. Die Kapitel sind aufgeteilt nach den KI-Agenten, gegen die der UCT-Agent angetreten ist. Neben der Gewinnrate werden ebenfalls die durchschnittlichen Längen der von den KI-Agenten gewonnenen Episoden betrachtet. Diese könnte von Interesse sein, da im Fall von Vier Gewinnt, das Erstellen einer 4er-Reihe auch sehr früh möglich ist. Wie früh ein Spiel beendet wird, könnte Einblick in das Spielverhalten der KI-Agenten gewähren.

4.1.1 Durchführung

In Abschnitt 3.1.1 wird beschrieben, dass laut Allis, Vier Gewinn ein theoretischer Gewinn für den ersten Spieler sei [39]. Um diesen Vorteil auszugleichen, spielen die

KI-Agenten abwechselnd als Spieler 1.

Außerdem wird, um ein besseres Bild über die tatsächliche Spielstärke der KI-Agenten zu erlangen, bei der Hälfte der zu spielenden Episoden nicht das leere Spielbrett als Ausgangszustand verwendet. Stattdessen werden vor Beginn jeder dieser Partien eine Anzahl zufälliger Eröffnungszüge manuell durchgeführt, sodass die KI-Agenten die Partien mit einem bereits besetzten Spielbrett beginnen. Für die Länge der Eröffnungszügen wird zufällig zwischen zwei, vier und sechs Zügen entschieden. Die Anzahl an Eröffnungszügen muss gerade sein, damit die Spielerfolge nicht ungewollt verändert wird und kein KI-Agent benachteiligt wird dadurch, dass ihm ein zufälliger Zug mehr als dem Gegner zugeordnet wird. Außerdem wird jede Eröffnungszug-Folge für jeweils zwei Episoden verwendet, sodass die KI-Agenten in beiden Spieler-Rollen für in Bezug auf diese Zugfolge bewertet werden können.

Es werden insgesamt 50 verschiedene Eröffnungszug-Folgen verwendet. Demnach werden also insgesamt 200 Episoden gespielt. Von diesen werden 100 Episoden mit Eröffnungszügen gespielt, bei welchen wiederum 50 Episoden von dem Ludii-Agenten als erster Spieler gestartet werden und die anderen 50 von dem GBG-Agenten. In den anderen 100 Episoden ohne Eröffnungszüge macht der Ludii-Agent ebenfalls für 50 davon den ersten Zug und für die restlichen 50 Episoden den zweiten Zug. Tabelle 3 gibt einen Überblick.

Tabelle 3 Episoden-Konfiguration für Vier Gewinnt

KI-Agent als	Spieler 1	Spieler 2
ohne Eröffnungszüge	50	50
mit Eröffnungszügen	50	50

Der von der LudiiAI()-Klasse empfohlene UCT-Agent tritt im Spiel Vier Gewinnt gegen die in Abbildung 4 KI-Agenten an.

Tabelle 4 Für Vier Gewinnt antretende GBG-Agenten

AlphaBeta- und AlphaBeta-DL-Agent
TCL-Agent
MWRAP-Agent
MCTS-Agent

4.1.2 AlphaBeta und AlphaBeta-DL

Abbildung 7 zeigt die Gewinnraten des UCT-Agenten gegen den AlphaBeta-Agenten für die gespielten 200 Episoden. In den 100 Episoden ohne Eröffnungszüge erlangte

der UCT-Agent eine Gewinnrate von 16,5% gegen den AlphaBeta-Agenten, welcher eine Gewinnrate von 83,5% erreichte. Für die 100 Episoden mit Eröffnungszügen stieg die Gewinnrate des UCT-Agenten auf ungefähr das Doppelte, nämlich 32%. Die Gewinnrate des AlphaBeta-Agenten sank dementsprechend. Der AlphaBeta-Agent ist eindeutiger Gewinner beider Episoden-Sätze. Dennoch konnte der UCT-Agent in den Partien mit Eröffnungszügen größeren Erfolg als in denen ohne solche verzeichnen.

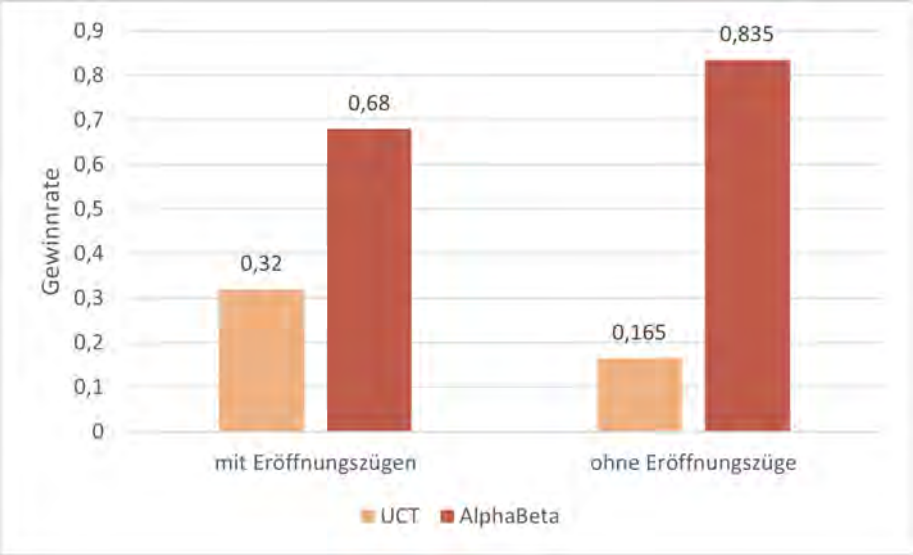


Abbildung 7 Gewinnraten des AlphaBeta- und UCT-Agenten in Episoden gegeneinander

Das Balkendiagramm aus Abbildung 8 illustriert die durchschnittliche Länge von Episoden, die jeweils vom UCT- oder AlphaBeta-Agenten gewonnen wurden. Das Einsetzen von Eröffnungszügen scheint keinen bemerkenswerten Unterschied zu machen. Ob mit oder ohne Eröffnungszüge, die vom UCT-Agenten gewonnenen Episoden sind im Durchschnitt um circa 25 Züge kürzer als die vom AlphaBeta gewonnenen Episoden.

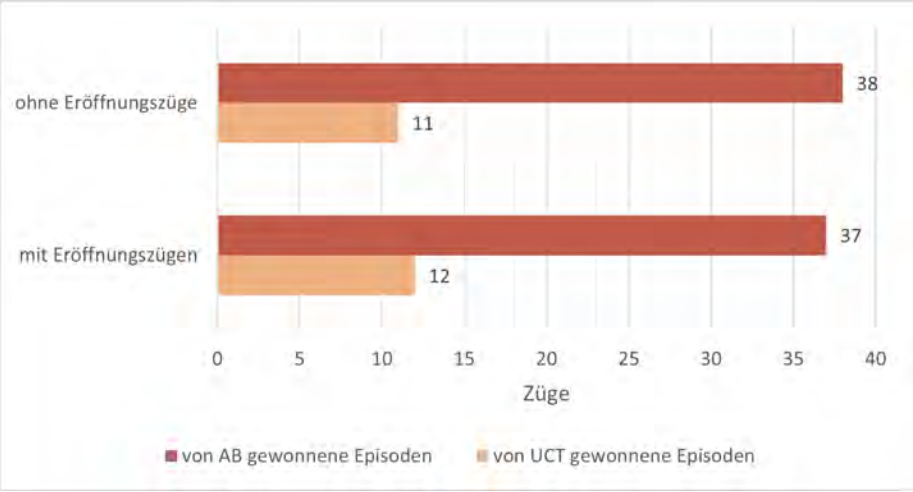


Abbildung 8 Durchschnittliche Längen der von AB- und UCT-Agent gewonnenen Episoden

Episoden Betrachtung

Angesichts der auffälligen Kürze der von dem UCT-Agenten gewonnenen Partien wurden mit Hilfe der Spieldaten (siehe Anhang) einzelne Episoden exemplarisch betrachtet. Dies geschah mittels des vom GBG-Framework bereitgestellten Inspektionsmodus.

Episoden mit Eröffnungszügen

Die Episodenlängen sind auffällig für die Episoden mit denselben Eröffnungszügen, in denen beide KI-Agenten jeweils einmal als Spieler 1 gewinnen. Der UCT-Agent, wenn er denn gewinnt, tut dies in beachtlich weniger Zügen als der AlphaBeta-Agent. Bei Betrachtung der Eröffnungszüge wird klar, dass einige von ihnen eine sehr gute Gewinnvorlage für den ersten oder den zweiten Spieler bieten. Das bedeutet allerdings für keinen der KI-Agenten einen Nachteil, da jede Eröffnungszug-Folge zweimal gespielt wird, sodass beide KI-Agenten in beiden Spielerrollen spielen. Dennoch scheint der AlphaBeta-Agent in Episoden, die er gewinnt, das nicht so schnell zu tun, wie der UCT-Agent, wenn er in derselben Spielerrolle die Episode gewinnt. Ein Beispiel hierfür zeigen die Grafiken in Abbildung 9.

(a) Sieg für UCT als Spieler 1 (Gelb)

(b) Sieg für AB als Spieler 1 (Gelb)

Abbildung 9 Vergleich von zwei Episoden Vier Gewinnt mit gleichen Eröffnungszügen

Die Grafiken in der Abbildung stellen zwei Episoden mit denselben Eröffnungszügen dar. In Abbildung 9a war der UCT-Agent Spieler 1 (Gelbe Disks) und gewann. In Abbildung 9b gewann der AlphaBeta-Agent, ebenfalls als Spieler 1. Die Ziffern auf den Disks kennzeichnen die Reihenfolge der Züge. Oben sind die Spalten- beziehungsweise Zugnummern für das Setzen einer Disk in die jeweilige Spalte notiert. Die zum Gewinn führenden 4er-Ketten sind grün umrandet. Die abgebildeten Episoden beginnen mit sechs Eröffnungszügen, die rot umrandet sind. Während der UCT-Agent als Spieler 1 diese ausnutzt und nach insgesamt drei weiteren Zügen einen Sieg erzielt, verpasst der AlphaBeta-Agent, wenn selber Spieler 1, diese Chance. Anstatt wie der UCT-Agent in Grafik 9a mit Zug 7 eine gelbe 3er-Kette zu bilden und sich einen Gewinn zu sichern, wählt der AlphaBeta-Agent die Spalte 1 für sei-

nen Zug. Der UCT-Agent blockiert in Zug 8 die Spalte 2 und der AlphaBeta-Agent entscheidet die Partie erst nach insgesamt 31 weiteren Zügen (siehe Anhang).

Episoden ohne Eröffnungszüge

Interessanter sind die Episoden ohne Eröffnungszüge. Alle Spiele, die der UCT-Agent für sich entscheiden konnte, waren Episoden in denen er als Spieler 1 tätig gewesen ist. Etwas mehr als die Hälfte dieser Episoden hat der UCT-Agent außerdem nach nur 7 Zügen gewonnen. Beim Nachspielen dieser Episoden im Inspektionsmodus fällt ein gewisses Muster auf.

Abbildung 10 Verlauf einer Episode Vier Gewinnt

Abbildung 10 zeigt den Durchlauf einer dieser Episoden nachgestellt im GBG. Der UCT-Agent als Spieler 1 spielt für Gelb und der AlphaBeta-Agent demnach für Rot. Die Ziffern auf den Disks kennzeichnen die Reihenfolge in der sie gesetzt wurden. Oben sind die Spalten- beziehungsweise Zugnummern für das Setzen einer Disk in die jeweilige Spalte notiert. Der UCT-Agent macht in der Mitte des Spielbretts seinen ersten Zug. Der AlphaBeta-Agent bewertet daraufhin alle Züge, die ihm nun zur Verfügung stehen als gleich schlecht. In so einem Fall wählt der AlphaBeta-Agent einen freiwilligen Zug. Dieser fällt auf Spalte 6. Der UCT-Agent wählt für seinen nächsten Zug die Spalte 2, also sind nun zwei gelbe Disks nebeneinander. Der AlphaBeta-Agent versucht die Bildung einer möglichen 3er-Kette von gelben Disks nicht zu verhindern und setzt seinen nächsten Zug in Spalte 5 neben seine andere rote Disk. Der UCT-Agent nutzt dies aus und legt seine Disk so, dass er drei gelbe Disks angereiht hat und auf beiden Seiten eine leere Spalte ist. Somit ist der AlphaBeta-Agent in einer Zwickmühle, denn egal ob er nun eine der beiden Seiten der gelben Disk-Reihe blockiert, kann der UCT-Agent im nächsten Zug die andere Spalte besetzen und gewinnt trotzdem. Stattdessen legt der AlphaBeta-Agent seine nächste Disk in dieselbe Spalte wie in seinem vorherigen Zug. Der UCT-Agent bildet ist wieder an der Reihe und beendet die Partie mit einem Sieg.

Ein Vergleich der Zuglisten für alle diese Episoden in Tabelle 5 zeigt, dass ein Zusammenhang zwischen ihnen besteht. Der UCT-Agent wählt stets die mittlere Spalte für seinen ersten Zug. Der AlphaBeta-Agent wählt (zufällig) entweder die Spalte ganz rechts oder ganz links. Abhängig davon wählt der UCT-Agent seinen nächsten Zug. Er setzt seine nächste Disk immer auf die entgegengesetzte Seite der Disk des AlphaBeta-Agenten. Es ist nochmal anzumerken, dass der UCT-Agent nicht deterministisch handelt. Ab dem insgesamt 4. Zug der Episoden weichen die Zuglisten

Tabelle 5 Zuglisten der vom UCT-Agenten in sieben Zügen gewonnenen Episoden

Zeile	UCT	AB	UCT	AB	UCT	AB	UCT
1	3	6	2	2	4	5	1
2	3	6	2	5	1	5	0
3	3	6	2	5	1	0	4
4	3	0	4	3	2	0	5
5	3	6	2	3	4	5	1
6	3	0	4	6	2	6	5
7	3	6	2	6	1	2	0
8	3	6	2	3	1	1	0

eher voneinander ab, da der AlphaBeta-Agent wieder einen zufälligen Zug wählt. Dennoch bleiben die Züge des UCT über einen Teil der Episoden gleich. So sind in den Zeilen 2,6,7 und 8 und nochmal in den Zeilen 1 und 5 die Züge des UCT-Agenten identisch. Die Zugliste in Zeile 2 entspricht der zuvor in Abbildung 10 beschriebenen Episode.

Aus den einzelnen Episoden-Betrachtungen geht auch heraus, dass der UCT-Agent besonders in den frühen Phasen des Spiels mögliche Gewinnstellungen für den AlphaBeta-Agenten erkennt und diese blockiert. Je länger das Spiel allerdings andauert, desto schlechter wird der UCT-Agent darin.

AlphaBeta-DL

Die abgebildeten Gewinnraten in Tabelle 6 für die Episoden zwischen dem AlphaBeta-DL- und dem UCT-Agenten zeigen keine nennenswerte Abweichung zu denen des AlphaBeta-Agenten.

Tabelle 6 Gewinnraten des AlphaBeta-DL- und UCT-Agenten in Episoden gegeneinander

	UCT	AlphaBeta-DL
ohne Eröffnungszüge	0,185	0,815
mit Eröffnungszügen	0,31	0,69

Auch die durchschnittlichen Längen gewonnener Episoden des UCT- und AB-DL-Agenten sind ähnlich. Nur die Länge der vom UCT-Agenten gewonnenen Episoden ohne Eröffnungszüge scheint sich weiter verkürzt zu haben (Tabelle 7).

Tabelle 7 Durchschnittliche Längen der von AB-DL- und UCT-Agent gewonnenen Episoden

	von UCT gewonnen	von AlphaBeta-DL gewonnen
ohne Eröffnungszüge	7	38
mit Eröffnungszügen	11	38

Auswertung

Der AlphaBeta-Agent, unabhängig von der Verwendung von Eröffnungszügen, ist eindeutig stärker als der UCT-Agent. Mit Gewinnraten von 83,5% und 68% gewinnt der AlphaBeta-Agent sehr verlässlich, auch wenn es länger dauert. Dies entspricht den Erwartungen an den stark-spielenden KI-Agenten. Dennoch ist bemerkenswert, wie schnell der UCT-Agent zumindest bis ungefähr zur Mitte des Spiels Fehler des AlphaBeta-Agenten erkennt und ausnutzt. Allerdings liegt die Vermutung nah, dass die Fehler des AlphaBeta-Agenten sich auf eine Lücke in seinem Eröffnungsbuch zurückführen lässt. Das ist erkennbar daran, dass die in Tabelle 5 beispielhaft aufgelisteten Zugfolgen sehr ähnlich zueinander sind. Wenn der Gegner des AlphaBeta-Agenten als Erstes eine Disk in die mittlere Spalte legt, bewertet er alle Züge die darauf folgen als gleich schlecht und wählt seinen Zug zufällig. Wenn dieser zufällige Zug ein faktisch schlechter ist, macht der AlphaBeta-Agent es dem UCT-Agenten einfacher, für sich selbst eine Gewinnstellung aufzubauen. Dies lässt sich in den Episoden, die der UCT-Agent mit 7 Zügen gewinnt, gut erkennen. Solch eine Situation tritt aber anscheinend nicht oft auf, da der AlphaBeta-Agent ja dennoch meistens gewinnt.

Wenn der UCT-Agenten nicht das Glück eines derartigen Fehlers in den frühen Phasen des Spiels hatte, gewann er auch nur selten. Dass er später im Spiel auch nicht mehr in der Lage war, die Gewinnstelllungen des AlphaBeta-Agenten zu blockieren, lag vermutlich am höheren Verzweigungsgrad des UCT-Spielbaums zu den Zeitpunkten. Somit lässt sich auch die Deckungsgleichheit der Gewinnraten zwischen dem AlphaBeta- und AlphaBeta-DL-Agenten erklären. Der AlphaBeta-DL-Agent hat „bewusst“ versucht, den Gewinn des UCT-Agenten soweit zu verschieben wie möglich, welcher dann später im Spiel einen Fehler begangen hat. Zwar hat der AlphaBeta-Agent das nicht getan. Jedoch musste er nur die Anfangsphase des Spiels überstehen und warten bis der UCT-Agent nicht mehr in der Lage war, seine Disks zu blockieren. - Uniform Random Playouts nicht so verlässlich wie ein Minimax mit AlphaBeta Purning der auch noch trainiert ist

4.1.3 TCL

Abbildung 11 zeigt, dass in Episoden ohne Eröffnungszüge die beiden KI-Agenten fast gleich aufziehen. Allerdings ist die Gewinnrate des UCT-Agenten mit 57% für die Episoden mit Eröffnungszügen sichtbar höher als die des TCL-Agenten.

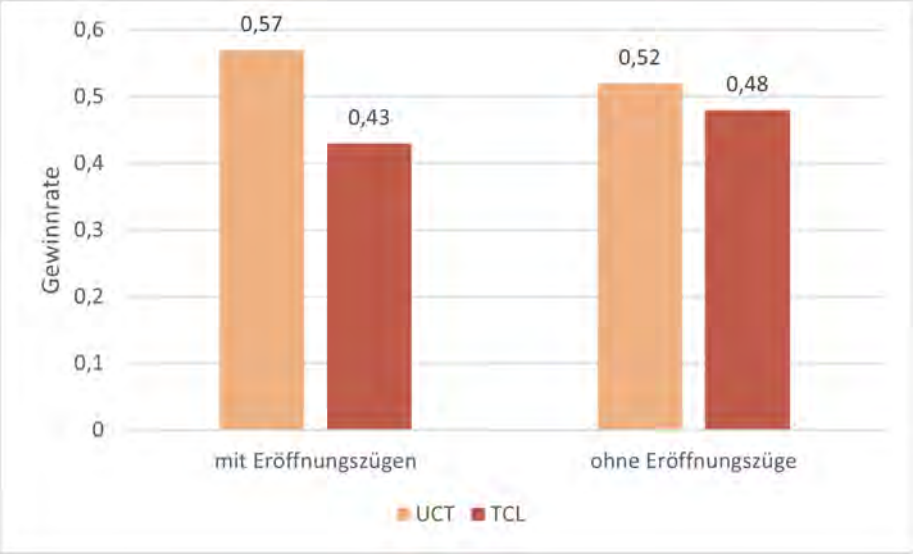


Abbildung 11 Gewinnraten des TCL- und UCT-Agenten in Episoden gegeneinander

Es wird aus Abbildung 12 ersichtlich, dass die vom UCT-Agenten gewonnenen Episoden durchschnittlich kürzer sind als die vom TCL-Agenten gewonnenen. Während der TCL-Agent im Schnitt mehr als 30 Züge für gewonnene Episoden verzeichnet hat, sind es für den UCT-Agenten in Episoden ohne Eröffnungszüge durchschnittlich etwa 16 Züge. In Episoden mit Eröffnungszügen steigt dieser Schnitt etwas an.

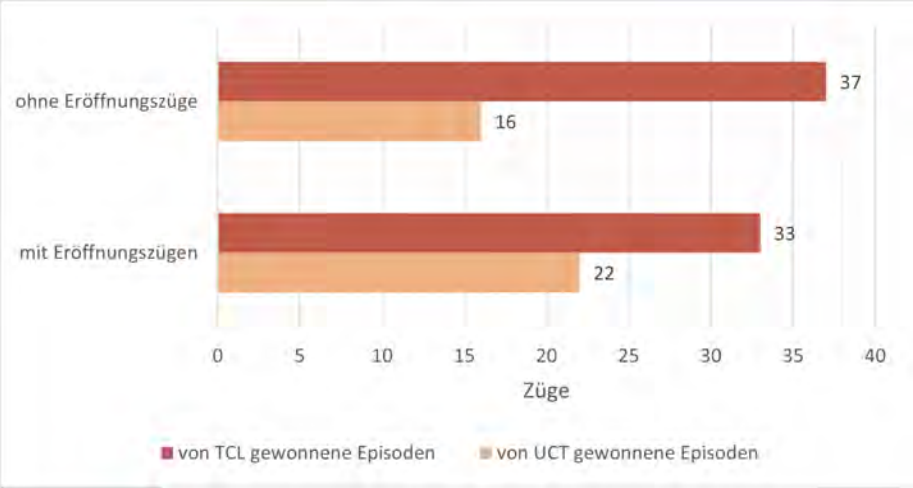


Abbildung 12 Durchschnittliche Längen der von TCL- und UCT-Agent gewonnenen Episoden

Auswertung

Der UCT-Agent scheint besser mit den durch die Eröffnungszügen zufällig generierten Spielbrettern umgehen zu können als der TCL-Agent. Das ergibt Sinn in Anbetracht der Tatsache, dass der TCL-Agent beim Trainieren strategisch sinnvolle Zugfolgen gelernt haben sollte. Die vermutlich oft strategisch schlechten Eröffnungszüge könnten den Spielverlauf in eine Richtung gelenkt haben, mit der der TCL-Agent während des self-play gar nicht oder selten konfrontiert wurde. Da der UCT-Agent untrainiert ist, kann er besser mit unerwarteten Zügen und Spielbrett-Zuständen umgehen, besonders zu Beginn des Spiels, wenn der aufgebaute Suchbaum noch nicht besonders tief ist. Die zufälligen Eröffnungszüge nehmen daher vermutlich keinen Einfluss auf die Zug-Wertungen und somit auch -Wahl des UCT-Agenten. Ohne Eröffnungszüge scheinen die beiden KI-Agenten ungefähr gleich-auf zu sein bezüglich ihrer Spielstärke. Den Vorteil der Planungsfähigkeit seiner Züge, den der UCT-Agent dem TCL-Agenten gegenüber haben sollte, scheint nur einen nicht nennenswerten Unterschied zu machen.

Für den Anstieg der Durchschnittslänge der vom UCT-Agenten gewonnenen Episoden mit Eröffnungszügen gegenüber denen ohne Eröffnungszüge konnte kein Zusammenhang zwischen der Länge der Eröffnungszüge oder ein besonderes Muster erkannt werden. Es lässt sich vermuten, dass die generierten Eröffnungszüge in diesen Instanzen besonders ungelegen für den UCT-Agenten waren. Ein Blick auf die Vorkommnisse der verschiedenen Längen der Eröffnungszüge zeigt, dass Eröffnungszüge der Längen 4 und 6 öfter vorgekommen sind, als solche mit der Länge 2. Da der UCT-Agent mehr als die Hälfte der Episoden mit Eröffnungszügen gewonnen hat, könnte sich die durchschnittliche Verlängerung seiner gewonnenen Episoden darauf zurückführen lassen.

4.1.4 TCL mit MCTS-Wrapper

Während der UCT-Agent mit einer Gewinnrate von 56% die Episoden ohne Eröffnungszüge dominiert hat, wendet sich das Blatt bei den Episoden mit Eröffnungszügen (Abbildung 13). Wenn auch die Differenz zum UCT-Agenten nicht hoch ist, ist der Anstieg der Gewinnrate des MWRAP-Agenten um 8,5% dennoch nennenswert. Zum ersten Mal führt der Einsatz von Eröffnungszügen zum Wechsel von Gewinnern.

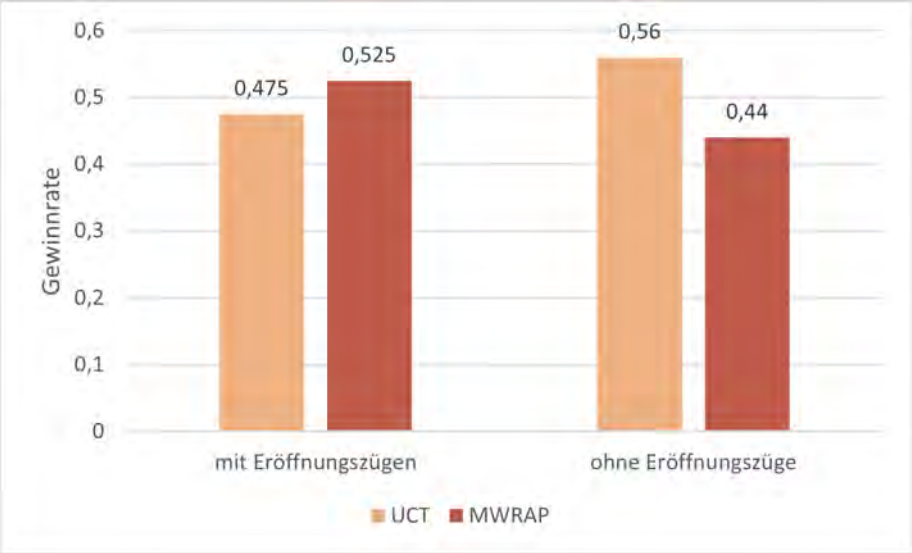


Abbildung 13 Gewinnraten des MWRAP- und UCT-Agenten in Episoden gegeneinander

Abbildung 14 verdeutlicht, dass der Einsatz von Eröffnungszügen die vom UCT-Agenten gewonnenen Episoden im Durchschnitt verlängert, während er sie für den MWRAP-Agenten verkürzt.

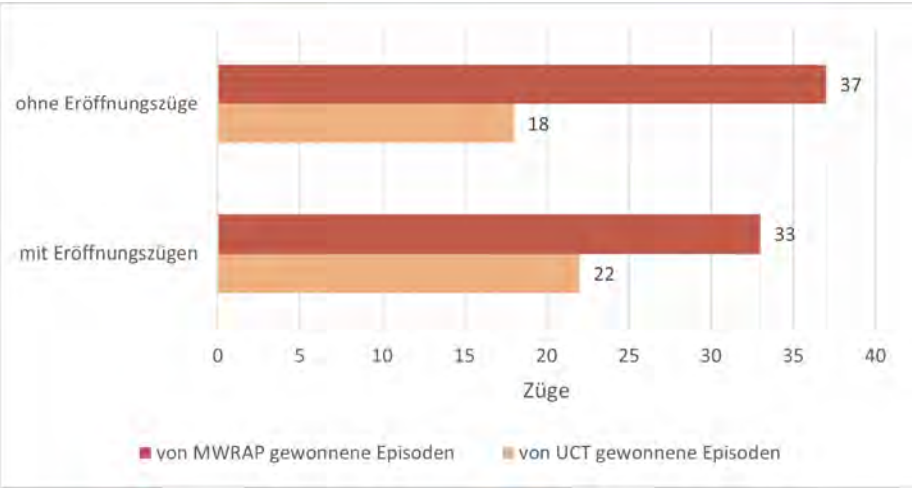


Abbildung 14 Durchschnittliche Längen der von MWRAP- und UCT-Agent gewonnenen Episoden

Episoden-Betrachtung

Alle Episoden, die mit 7 Zügen beendet werden konnten, sind Episoden mit Eröffnungszügen gewesen (siehe Anhang). Diese waren teilweise 6 Züge lang und sehr vorteilhaft für den ersten Spieler. In diesen Episoden haben der UCT- und der MWRAP-Agent immer abwechselnd als Spieler 1 gewonnen.

Auswertung

Es widerspricht Erwartungen, dass der MWRAP-Agent in Episoden ohne Eröffnungszüge gegen den UCT-Agenten schlechter als der TCL-Agent abschneidet. Vor-

teilhafte Eröffnungszüge sind vermutlich der Grund dafür, dass die Gewinnraten der beiden KI-Agenten etwa bei 50% liegen. Es würde bedeuten, dass sie abwechselnd als Spieler 1 die Eröffnungszüge gleich gut ausnutzen. Zum ersten Mal verschlechtert sich die Gewinnrate des UCT-Agenten unter dem Einsatz von Eröffnungszügen im Vergleich zu Episoden ohne diese.

4.1.5 MCTS

Die abgebildeten Gewinnraten (Abbildung 15) unterstreichen die hohe Differenz in Gewinnraten zwischen dem UCT- und MCTS-Agenten. Ob mit oder ohne Eröffnungszüge, hat der UCT-Agent mindestens drei Viertel der Episoden für sich entschieden.

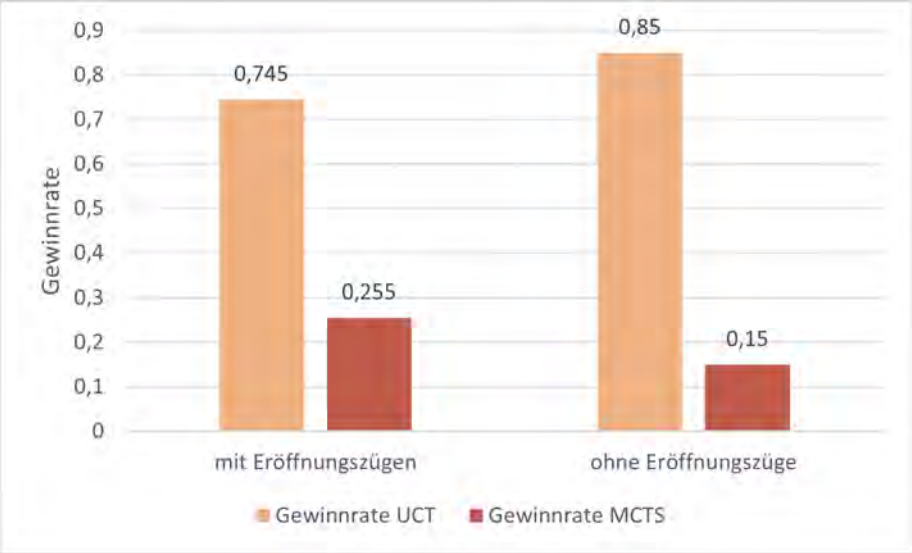


Abbildung 15 Gewinnraten des MCTS- und UCT-Agenten in Episoden gegeneinander

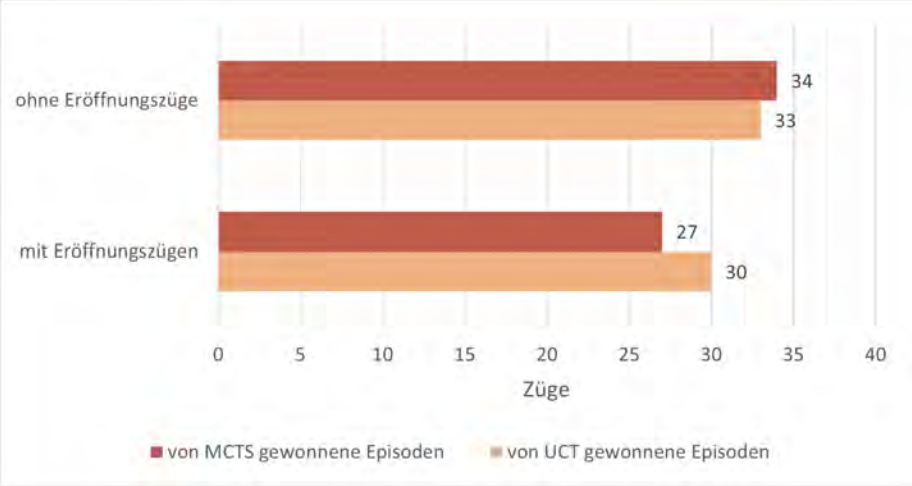


Abbildung 16 Durchschnittliche Längen der von MCTS- und UCT-Agent gewonnenen Episoden

Das Diagramm aus Abbildung 16 verzeichnet, dass der UCT-Agent mit im Durch-

schnitt circa 33 Zügen zum ersten Mal durchschnittlich mehr als 22 Züge zum Erfolg in seinen gewonnene Episoden ohne Eröffnungszüge gebraucht hat. In den gewonnenen Episoden mit Eröffnungszügen hat der UCT-Agent ebenfalls das erste Mal durchschnittlich mehr Züge zum Sieg benötigt als sein Gegner.

Auswertung

Der UCT- sowie der MCTS-Agent sind sehr ähnlich in ihrer Implementierung (siehe Kapitel 2.2.1 und 2.3.5). Deswegen ist die große Differenz zwischen ihrer Spielstärke besonders auffällig. Auch bemerkenswert ist, dass der UCT-Agent erstmals einen Gegner hat, der nicht so früh im Spiel verliert. Auch wenn der UCT-Agent sehr verlässlich gewinnt, braucht er hierfür viel länger als in den Episoden gegen die anderen KI-Agenten. In den frühen bis mittleren Phasen des Spiels scheinen beide KI-Agenten gleich auf zu sein und selten Fehler zu machen. In den späteren Phasen des Spiels ist der UCT-Agent wohl besser in seiner Zug-Wahl als der MCTS-Agent und nutzt dessen Fehler aus.

4.2 Othello

In diesem Abschnitt werden die Resultate der für Othello gespielten Episoden zwischen den KI-Agenten dargelegt und bewertet. Er wird nach den KI-Agenten aufgeteilt, gegen die der MAST-Agent getestet wurde. Außer der Gewinnrate werden besonders kurze Episoden ebenfalls exemplarisch untersucht.

Um die Stärke der KI-Agenten, die ausgehend von dem anfänglichen Aufbau des Spielbretts mit vier Spielsteinen trainiert, verlässlicher bewerten zu können, genauso wie die Leistung der general-purpose KI-Agenten, welche ohne Vorkenntnisse spielen, werden für 100 der 200 Spielepisoden XOT Openings aus der kürzeren Liste eingesetzt (siehe Kapitel 3.2.1). Die Openings werden der Liste zufällig entnommen. Jedes Opening wird zweimal eingesetzt, sodass beide KI-Agenten für jedes Opening eine Episode als erster Spieler und eine als zweiter Spieler bestreiten. Somit werden insgesamt 50 verschiedene Openings verwendet. Die anderen 100 beginnen mit den üblichen vier Spielsteinen auf dem Spielbrett und werden ebenfalls so unterteilt, dass für die Hälfte der Episoden ein KI-Agent den ersten Zug tätigt und für die andere Hälfte der andere KI-Agent. Abbildung 8 verdeutlicht.

Tabelle 8 Episoden-Konfiguration für Othello

KI-Agent als	Spieler 1	Spieler 2
ohne XOT Openings	50	50
mit XOT Openings	50	50

4.2.1 Agenten

Für Othello verweisen die den Metadaten des Spiels auf den MAST-Agent, welcher gegen die folgenden KI-Agenten spielt:

Tabelle 9 Für Othello antretende GBG-Agenten

Edax über mehrere Level
TCL4-100_7_250k-lam05_P4_nPly2-FAm_A

4.2.2 Edax

Abbildung 17 stellt den Verlauf der Gewinnrate des MAST-Agenten gegen den Edax-Agenten von den Leveln 1 bis 4 für Episoden mit und ohne XOT Openings dar. Die dazugehörigen genauen Gewinnraten für den MAST- sowie den Edax-Agenten sind in den Tabellen 10 und 11 einsehbar. Die Gewinnrate des MAST-Agenten ist für Episoden ohne XOT Openings bei Level 2 und 3 höher als für die Episoden mit XOT Openings. Allerdings ist der Fall der MAST-Gewinnraten für beide Episoden-Sätze von Level 1 zu Level 2 beachtlich. Für Episoden ohne XOT Openings sinkt die Gewinnrate von 37,5% auf 6,5%. Das ist eine Differenz von 31%. Für Episoden mit XOT Openings sinkt sie von 24% auf 5%, also um 19%. Von Level 2 zu Level 3 fällt die Gewinnrate jeweils auf 2% und 3%, während auf Level 4 der Edax-Agent alle Episoden gewonnen hat.

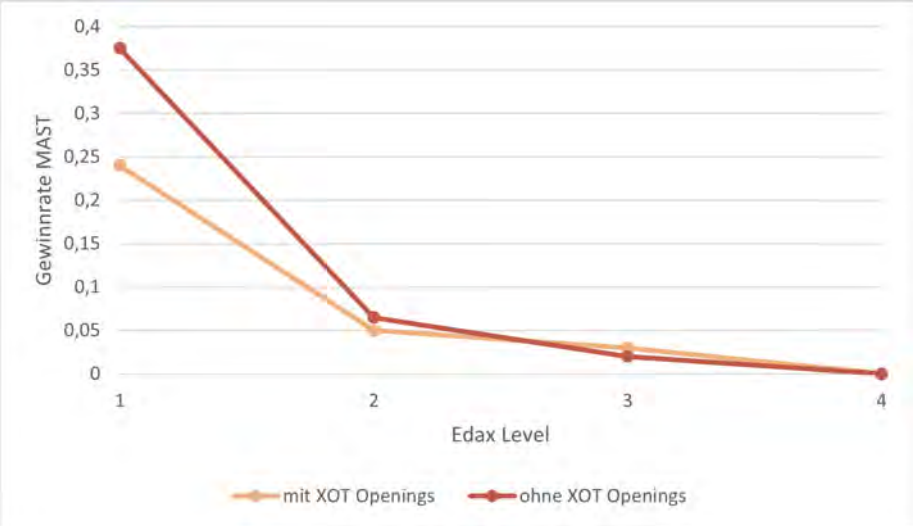


Abbildung 17 Entwicklung der Gewinnrate des MAST-Agenten gegen Edax über verschiedene Level

Tabelle 10 Gewinnraten der Episoden ohne XOT Openings

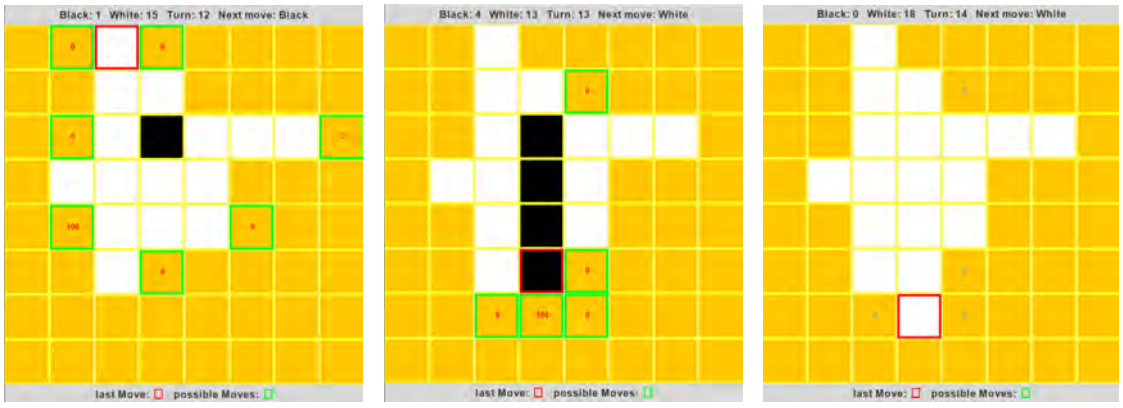
Edax-Level	MAST	Edax
1	0,375	0,625
2	0,065	0,935
3	0,02	0,98
4	0	1

Tabelle 11 Gewinnraten der Episoden mit XOT Openings

Edax-Level	MAST	Edax
1	0,24	0,76
2	0,05	0,95
3	0,03	0,97
4	0	1

Episoden-Betrachtung

Für Edax Level 1 und 2 sind ein Teil der von dem MAST-Agenten gewonnenen Episoden auffällig, da diese Partien in weniger als 20 Zügen beendet worden sind (siehe Anhang). Im Vergleich dazu, ist die kürzeste vom Edax-Agenten auf Level 1 gewonnene Episode 62 Züge lang, unabhängig vom Einsatz von XOT Openings. Das ist verhältnismäßig früh für Othello-Partien, die bis zu 64 Züge lang sein können. Es bedeutet, dass in diesen Fällen, der MAST-Agent alle Edax-Spielsteine umgedreht hat. Untersuchungen im Inspektionsmodus belegen, dass der Edax-Agent, auf Level 1 öfter als auf Level 2, in solchen Instanzen Fehler gemacht hat, die der MAST-Agent ausnutzen konnte.



(a) Edax-Agent (Schwarz) ist am Zug

(b) MAST-Agent (Weiß) ist am Zug

(c) MAST-Agent (Weiß) gewinnt

Abbildung 18 Letzten zwei Züge einer Edax-MAST-Episode auf Level 1 ohne XOT Openings[4]

Ein Beispiel hierfür ist in Abbildung 18 zu sehen. Sie zeigt eine Episode ohne XOT Openings, welche nach insgesamt 10 Zügen der KI-Agenten endete. Die rot umrandeten Felder zeigen den jeweils zuletzt getätigten Zug eines KI-Agenten an. Die grün umrandeten Felder zeigen die Zugmöglichkeiten, die dem KI-Agenten am Zug zur Wahl stehen. Die roten Ziffern geben die Bewertung für die jeweiligen Züge an. Diese wurden seitens eines weiteren Edax-Agenten auf Level 21 bewertet. Edax-Agenten

[4]Spieldarstellung über Inspektionsmodus des GBG-Framework

bewerten ausschließlich einen Zug als den besten, anstatt mehrere gleich-gut bewertete Züge zur Wahl zu haben. Deswegen gibt es nur die Bewertung 100 für den besten Zug und 0 für alle anderen.

In Abbildung 18a ist, nach dem letzten Zug des MAST-Agenten (Weiß), der Edax-Agent (Schwarz) am Zug. Abbildung 18b zeigt seinen gewählten Zug. Dieser führt zu einer Gewinnmöglichkeit für den MAST-Agenten, wie in Abbildung 18c erkennbar ist.

Auswertung

Der MAST-Agent stellt keine Konkurrenz für den sehr starken Edax-Agenten dar. Er hat bei einer Bestehensgrenze von 50% mit seinen 37,5% gegen den Edax-Agenten schon auf Level 1 nicht gewinnen können. Der steile Abfall der Gewinnraten des MAST-Agenten über die steigenden Level ist demnach zu erwarten gewesen. Die Betrachtung der Episoden beweist jedoch, dass er Fehler seines Gegners erkennt und für sich nutzen kann. Aufgrund der Komplexität des Spiels, ist es schwer, genauere Aussagen über das Spielverhalten des MAST-Agenten zu treffen. Die Länge der vom Edax-Agenten gewonnenen Episoden deutet daraufhin, dass der MAST-Agenten zumindest in den frühen Phasen des Spiels keine strategischen Fehler begeht, die der Edax-Agent ausnutzen könnte.

4.2.3 TCL

In Abbildung 19 ist der Kontrast der Gewinnraten zwischen dem MAST- und dem TCL4-Agenten zu erkennen. Die XOT Openings bringen eine Senkung der Gewinnrate des MAST-Agenten um 8%. Während sie für Episoden ohne XOT Openings bei 27,5% liegen, fallen sie für Episoden mit XOT Openings auf 19,5%.

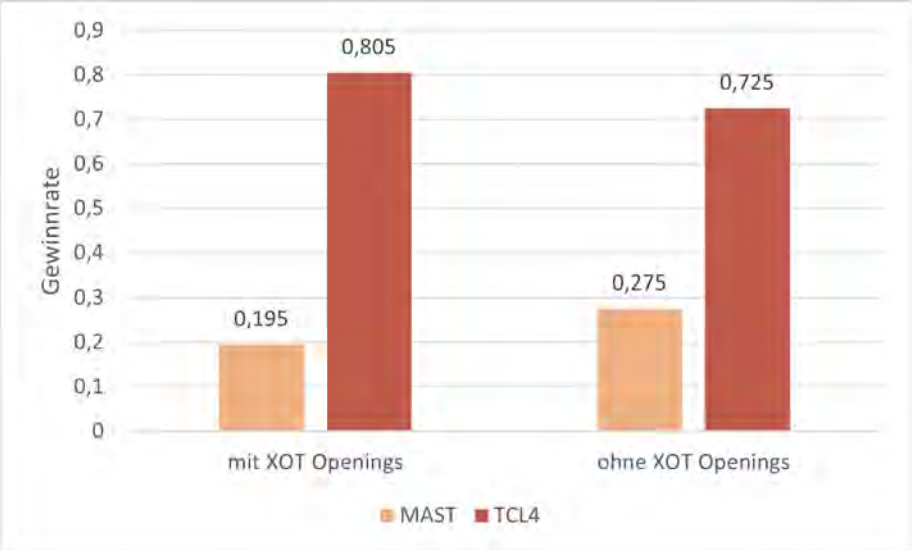


Abbildung 19 Gewinnraten des TCL4- und MAST-Agenten in Episoden gegeneinander

Episoden-Betrachtung

In der Spieldaten zwischen dem MAST- und TCL-Agenten existieren ebenfalls bemerkenswert kurze Episoden, welche vom MAST-Agenten gewonnen wurden (siehe Anhang). Ähnlich zu den Edax-MAST-Episoden, handelt es sich bei diesen Episoden um solche, in denen der TCL4-Agent einen ungünstigen Zug gewählt hat. Aus diesem Grund wurde an dieser Stelle auf eine exemplarische Darstellung verzichtet.

Auswertung

Der Misserfolg des MAST-Agenten gegen den TCL4-Agenten ist absehbar vor dem Hintergrund der von Konen und Scheiermann [37] beschriebenen Stärke des TCL4-Agent. Seine Fähigkeit, Fehler des Gegners auszunutzen, deutet auf eine gewisse Spielstärke hin. Diese ist aber sehr abhängig davon, ob der Gegner überhaupt Fehler begeht.

4.3 Nim

Dieser Abschnitt behandelt die Ergebnisse der Episoden, die von den KI-Agenten im Nim-Spiel bestritten wurden. Aufgeteilt ist der Abschnitt nach den KI-Agenten, die gegen den Biased MCTS-Agenten gespielt haben.

Je mehr Haufen für ein Spiel konfiguriert werden, desto länger dauern die Episoden, da das Nim-Spiel erst beendet ist, wenn alle Haufen leer sind (siehe Kapitel 3.3.1). Aufgrund dieses Zeitaufwandes wird sich auf das Spielen von den Konfigurationen für fünf, sieben und neun Haufen beschränkt. Außerdem wird festgelegt, dass der Spieler, der den letzten Gegenstand nimmt, als Verlierer des Spiels ausgeht. Es werden jeweils 100 Spiele für jede Konfiguration durchgeführt, bei denen nach 50 Episoden die Spieler-Reihenfolge der KI-Agenten vertauscht wird, sodass jeder in der Hälfte der Episoden den ersten Zug tätigt. Die Episodenverteilung wird durch Tabelle 12 visualisiert.

Tabelle 12 Episoden-Konfiguration für Nim

KI-Agent als	Spieler 1	Spieler 2
5	50	50
7	50	50
9	50	50

4.3.1 Agenten

Die LudiiAI()-Klasse gibt lediglich für die Konfiguration der Menge 5 eine Empfehlung zurück, nämlich den Biased MCTS-Agenten. Für die Mengen 5 und 9 tritt der UCT-Agent an.

Tabelle 13 Für Nim antretende GBG-Agenten

5	7	9
Bouton	Bouton	Bouton
TDNT4-20k-project	TDNT4-60k-project	TDNT4-10-4-60k-project
-	-	TDNT4-20-5-60k-project

4.3.2 Bouton

In allen Partien des Biased MCTS-Agenten für 5 Haufen und des UCT-Agenten für 7 und 9 Haufen gegen den Bouton-Agenten, ist der Bouton-Agent ausnahmsloser Gewinner (siehe Anhang).

Auswertung

Dieses Ergebnis ist nicht überraschend. Wie bereits in den Kapiteln 2.3.3 erläutert, ist der Bouton-Agent ein perfekter Spieler für das Nim-Spiel. Dennoch hätte der Biased MCTS- oder der UCT-Agent, der Theorie von Bouton [34] zufolge, die Hälfte der bestrittenen Episoden als Spieler 1 gewinnen können. Das allerdings nur bei perfektem Spiel ihrerseits. Solches lag den Ergebnissen nach aber nicht vor, denn ein Fehler hätte schon gereicht, damit der perfekt-spielende Bouton-Agent seine Verluststellung in eine Gewinngarantie wandeln kann.

4.3.3 TDNT4

Konfiguration mit 5 und 7 Haufen

Der Biased MCTS-Agent verlor für die 5-Haufen-Konfiguration alle gespielten Episoden gegen den TDNT4-Agenten (siehe Anhang).

Konfiguration mit 7 Haufen

Für die 7-Haufen-Konfiguration verlor der UCT-Agent ebenfalls jede Episode gegen den TDNT4-Agenten (siehe Anhang).

Konfiguration mit 9 Haufen

In den Episoden für die Konfiguration mit 9 Haufen gegen die zwei Variationen des TDNT4-Agenten konnte erstmals überhaupt ein Gewinn im Nim-Spiel gesehen werden. Abbildung 20 zeigt die Gewinnraten des UCT- und des TDNT4-Agenten in den zwei Sätzen von Episoden. Der UCT-Agent konnte gegen die mit weniger und kleineren Tupeln trainierte Variante des TDNT4-Agenten eine Gewinnrate von 42% erzielen, welche für die Episoden gegen die Variante mit größeren Tupeln auf 33% sank.

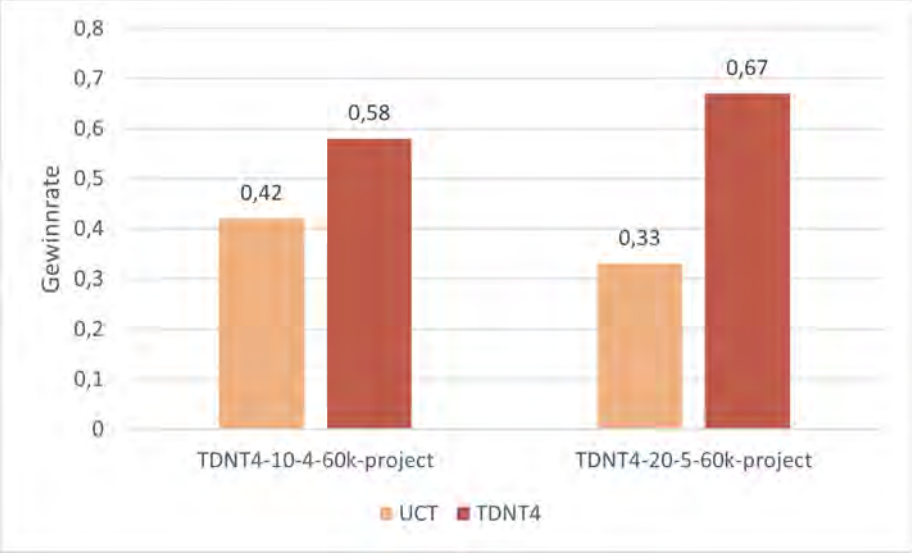


Abbildung 20 Gewinnraten des TDNT4- und UCT-Agenten in Episoden gegeneinander

Episoden-Betrachtung

Da eine Episode bei 9 Haufen aus sehr vielen Zügen besteht und das Auswerten mehrerer solcher Episoden für eine bedeutende Aussage nötig wäre, wurde aufgrund von Zeitgründen die genauere Untersuchung der Episoden an dieser Stelle verzichtet.

Auswertung

Sowohl der Biased MCTS- als auch der UCT-Agent sind dem TDNT4-Agenten unterlegen. Bemerkenswert ist, dass der Biased MCTS-Agent, selbst mit der Verwendung der vordefinierten Features für den vergleichsweise kleinen Zustandsraum der 5-Haufen-Konfiguration, keine einzige Partie für sich entscheiden kann. Es lässt sich vermuten, dass der große Zustandsraum für das Nim-Spiel einen Nachtteil für die MCTS-basierten KI-Agenten mit sich bringt, der für den trainierten TDNT4-Agenten aufgrund seiner unterschiedlichen Vorgehensweise kein Problem ist.

Jedoch scheint der noch größere Zustandsraum des Nim-Spiels mit 9 Haufen auch für den TDNT4-Agenten schwer zu meistern zu sein. Auch wenn für diese Konfiguration keine Episoden-Betrachtung durchgeführt wurde, ist die Annahme berechtigt, dass

der UCT-Agent in diesem Fall Gewinne verbuchen konnte, weil der TDNT4-Agent, im Gegensatz zu zuvor, Fehler gemacht hat.

5 Zusammenfassung und Ausblick

Das Ziel dieser Arbeit war die Evaluation der Spielstärke mehrerer general game-playing KI-Agenten des Ludii Game Systems. Diese wurde in Relation zu perfekten oder stark-spielenden KI-Agenten beispielhaft an den Spielen Vier Gewinnt, Othello und Nim gemessen werden. Als Grundlage zur Ausführung der Auswertungen wurde die in vorherigen Projekten implementierte Schnittstelle zwischen dem GBG-Framework und dem Ludii Game System [24] [25] verwendet. Somit beantwortet die Arbeit die zentrale Frage, welche zu Beginn formuliert wurde.

`Wie weit entfernt von starkem oder perfektem Spiel sind die general game-playing KI-Agenten des Ludii General Game Systems?`

Die gemessenen Gewinnraten der KI-Agenten waren dabei ausschlaggebendes Kriterium. Keiner der drei Ludii-Agenten konnte eine bedeutsame Spielstärke gegen die für die Spiele als sehr stark oder perfekt geltenden AlphaBeta-, Edax- und Bouton-Agenten beweisen. In Partien gegen weitere im GBG-Framework implementierte RL-Agenten konnten vereinzelte Erfolge im Spiel Vier Gewinnt festgestellt werden. Derartige Erfolge blieben für die komplexeren Spiele Othello und Nim aus. Als general game-playing KI-Agenten ohne Vorwissen über das Spiel sind die KI-Agenten des Ludii General Game System im Vergleich zu den vewendeten KI-Agenten weit von starkem oder perfektem Spiel für Spiele mit hoher Zustandsraum-Komplexität entfernt.

Durch die zeitliche Begrenzung dieser Arbeit wurde nicht allen aufgetretenen Ideen nachgegangen. Einige Ansätze für Erweiterungen dieser Arbeit werden im Folgenden aufgeführt:

- Es wurden in dieser Arbeit nur die vom Ludii Game System für die jeweiligen Spiele empfohlenen KI-Agenten evaluiert. Die Auswertung weiterer KI-Agenten für die Spiele Vier Gewinnt, Othello und Nim wären eine Option.
- Ebenfalls könnten die Ludii-Agenten auch für andere Spiele mit den starken KI-Agenten des GBG-Frameworks verglichen werden. Hierfür wäre allerdings die Erweiterung der Ludii-GBG-Schnittstelle um weitere Spiele erforderlich.
- Das Ludii Game System bietet die Möglichkeit, eigene MCTS-Agenten mit den gestellten Strategien zusammenzustellen. Es wäre interessant zu versuchen, auf diese Weise einen stärkeren KI-Agenten umzusetzen und seine Spielstärke zu evaluieren.

6 Literaturverzeichnis

[1] Wolfgang Ertel. *Grundkurs Künstliche Intelligenz: Eine praxisorientierte Einführung*. 4., überarbeitete Auflage. Lehrbuch. Wiesbaden: Springer Vieweg, 2016. ISBN: 978-3-658-13548-5.

[2] Stuart J. Russell und Peter Norvig. *Artificial intelligence: A modern approach*. 3. ed. Prentice-Hall series in artificial intelligence. Upper Saddle River, NJ: Prentice-Hall, 2010. ISBN: 978-0-13-604259-4.

[3] A. L. Samuel. „Some Studies in Machine Learning Using the Game of Checkers“. In: *IBM Journal of Research and Development* 3.3 (1959), S. 210–229. ISSN: 0018-8646. DOI: 10.1147/rd.33.0210. (Besucht am 16. 01. 2023).

[4] A. L. Samuel. „Some Studies in Machine Learning Using the Game of Checkers. II—Recent Progress“. In: *IBM Journal of Research and Development* 11.6 (1967), S. 601–617. ISSN: 0018-8646. DOI: 10.1147/rd.116.0601. (Besucht am 16. 01. 2023).

[5] D. Goodman und R. Keene. „Man Versus Machine: Kasparov Versus Deep Blue“. In: *ICGA Journal* 20.3 (1997), S. 186–187. ISSN: 1389-6911. DOI: 10.3233/ICG-1997-20308. URL: https://content.iospress.com/articles/icga-journal/icg20-3-08 (besucht am 17. 01. 2023).

[6] Claude E. Shannon. „XXII. Programming a computer for playing chess“. In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 41.314 (1950), S. 256–275. ISSN: 1941-5982. DOI: 10.1080/14786445008521796. (Besucht am 13. 01. 2023).

[7] Murray Campbell, A.Joseph Hoane und Feng-hsiung Hsu. „Deep Blue“. In: *Artificial Intelligence* 134.1-2 (2002), S. 57–83. ISSN: 00043702. DOI: 10.1016/S0004-3702(01)00129-1. (Besucht am 16. 01. 2023).

[8] Michael Genesereth und Michael Thielscher. *General Game Playing*. 1st ed. 2014. Synthesis Lectures on Artificial Intelligence and Machine Learning. Cham: Springer International Publishing und Imprint Springer, 2014. ISBN: 9783031015694. DOI: 10.1007/978-3-031-01569-4. (Besucht am 18. 01. 2023).

[9] Jacques Pitrat. „Realization of a general game-playing program“. In: *Information Processing, Proceedings of IFIP Congress 1968, Edinburgh, UK, 5-10 August 1968, Volume 2 - Hardware, Applications*. Hrsg. von A. J. H. Morrel. 1968, S. 1570–1574. URL: https://aitopics.org/download/classics:140D7211 (besucht am 16. 01. 2023).

[10] Barney Pell. „A STRATEGIC METAGAME PLAYER FOR GENERAL CHESS-LIKE GAMES“. In: *Computational Intelligence* 12.1 (1996), S. 177–198. ISSN: 0824-7935. DOI: 10.1111/j.1467-8640.1996.tb00258.x. (Besucht am 18. 01. 2023).

[11] Nathaniel Love, Timothy Hinrichs und Michael Genesereth. „General Game Playing: Game Description Language Specification“. In: 4. Apr. 2006. URL: https://www.cs.uic.edu/~hinrichs/papers/love2006general.pdf (besucht am 16. 01. 2023).

[12] Michael Genesereth, Nathaniel Love und Barney Pell. „General Game Playing: Overview of the AAAI Competition“. In: *AI Magazine* 26.2 (2005), S. 62. ISSN: 2371-9621. DOI: 10.1609/aimag.v26i2.1813. URL: https://ojs.aaai.org/index.php/aimagazine/article/view/1813 (besucht am 18. 01. 2023).

[13] David Silver u. a. „Mastering the game of Go without human knowledge“. In: *Nature* 550.7676 (2017), S. 354–359. ISSN: 1476-4687. DOI: 10.1038/nature24270. URL: https://www.nature.com/articles/nature24270 (besucht am 18. 01. 2023).

[14] David Silver u. a. „A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play“. In: *Science (New York, N.Y.)* 362.6419 (2018), S. 1140–1144. DOI: 10.1126/science.aar6404. (Besucht am 18. 01. 2023).

[15] David Silver u. a. „Mastering the game of Go with deep neural networks and tree search“. In: *Nature* 529.7587 (2016), S. 484–489. ISSN: 1476-4687. DOI: 10.1038/nature16961. (Besucht am 18. 01. 2023).

[16] Éric Piette u. a. „Ludii – The Ludemic General Game System“. In: 2019. URL: https://arxiv.org/pdf/1905.05013 (besucht am 16. 01. 2023).

[17] Cameron Browne. „Modern Techniques for Ancient Games“. In: *Proceedings of IEEE Computational Intelligence and Games* (2020). URL: https://arxiv.org/pdf/2101.10066 (besucht am 13. 01. 2023).

[18] David Parlett. „What's a Ludeme?“ In: *Game & Puzzle Design* 2.2 (2016), S. 81–84. URL: http://gapdjournal.com/issues/issue-2-2/issue-2-2-sample-parlett.pdf (besucht am 14. 01. 2023).

[19] Jakub Kowalski u. a. „Regular Boardgames“. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33.01 (2019), S. 1699–1706. ISSN: 2159-5399. DOI: 10.1609/aaai.v33i01.33011699. (Besucht am 16. 01. 2023).

[20] S. Schiffel und M. Thielscher. „Representing and Reasoning About the Rules of General Games With Imperfect Information“. In: *Journal of Artificial Intelligence Research* 49 (2014), S. 171–206. ISSN: 1076-9757. DOI: 10.1613/jair.4115. URL: https://www.jair.org/index.php/jair/article/view/10862 (besucht am 18. 01. 2023).

[21] Michael Thielscher. „GDL-III: A Description Language for Epistemic General Game Playing“. In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*. 2017, S. 1276–1282. DOI: 10.24963/ijcai.2017/177. (Besucht am 13. 01. 2023).

[22] Wolfgang Konen. „General Board Game Playing for Education and Research in Generic AI Game Learning“. In: *Conference on Games (CoG), London, 2019*. URL: http://arxiv.org/pdf/1907.06508v1 (besucht am 20. 01. 2023).

[23] Johannes Scheiermann. „Sind (trainierte) General-Purpose-RL-Agenten im Brettspiel Othello stärker als (untrainierte) General-Game-Playing Agenten?“ In: 2020. URL: http://www.gm.fh-koeln.de/~konen/research/PaperPDF/INF-Prj-Scheiermann-2020-08.pdf (besucht am 10. 01. 2023).

[24] Ann Weitz. „Entwicklung einer allgemeinen Schnittstelle zwischen Ludii und dem GBG Framework“. In: 2022. URL: https://www.gm.fh-koeln.de/~konen/research/PaperPDF/PP-Doku-Weitz-2022-02.pdf (besucht am 13. 01. 2023).

[25] Meltem L. Seven. „Erweiterung der Schnittstelle zwischen Ludii und dem GBG-Framework um die Spiele Connect Four und Nim“. In: 2022.

[26] Ann Weitz. „Untersuchung von selbstlernenden Reinforcement Learning Agenten im computergenerierten Spiel Yavalath“. Magisterarb. TH Köln – University of Applied Sciences, 2022. URL: https://www.gm.fh-koeln.de/~konen/research/PaperPDF/BA-Weitz-final-2022.pdf (besucht am 13. 01. 2023).

[27] Aske Plaat. *Learning to Play: Reinforcement Learning and Games*. 1st ed. 2020. Springer eBook Collection. Cham: Springer International Publishing und Imprint Springer, 2020. ISBN: 978-3-030-59237-0. DOI: 10.1007/978-3-030-59238-7.

[28] Uwe Lorenz. *Reinforcement Learning: Aktuelle Ansätze verstehen - mit Beispielen in Java und Greenfoot*. Berlin und Heidelberg: Springer Vieweg, 2020. ISBN: 978-3-662-61650-5. DOI: 10.1007/978-3-662-61651-2.

[29] Sylvain Gelly, Yizao Wang und Olivier Teytaud. *Modification of UCT with Patterns in Monte-Carlo Go*. 2006. URL: https://www.researchgate.net/publication/238378872_Modification_of_UCT_with_Patterns_in_Monte-Carlo_Go.

[30] Peter Auer, Nicolò Cesa-Bianchi und Paul Fischer. „Finite-time Analysis of the Multiarmed Bandit Problem“. In: *Machine Learning* 47.2/3 (2002), S. 235–256. ISSN: 1573-0565.
DOI: 10.1023/A:1013689704352. URL: https://link.springer.com/article/10.1023/A:1013689704352 (besucht am 18. 01. 2023).

[31] Levente Kocsis und Csaba Szepesvári. „Bandit Based Monte-Carlo Planning“. In: *Machine learning: ECML 2006*. Hrsg. von Johannes Fürnkranz, Tobias Scheffer und Myra Spiliopoulou. Bd. 4212. Lecture notes in computer science Lecture notes in artificial intelligence. Berlin und Heidelberg: Springer, 2006, S. 282–293. ISBN: 978-3-540-45375-8.
DOI: 10.1007/11871842{\textunderscore}29. (Besucht am 18. 01. 2023).

[32] Markus Thill u. a. „Temporal Difference Learning with Eligibility Traces for the Game Connect-4“. In: *CIG'2014, International Conference on Computational Intelligence in Games, Dortmund*. Hrsg. von Mike Preuss und Günther Rudolph. 2014, S. 84–91. URL: http://www.gm.fh-koeln.de/%C2%A0konen/Publikationen/ThillCIG2014.pdf (besucht am 25. 01. 2023).

[33] Richard Delorme. *Edax, version 4.4*. 2017. URL: https://github.com/abulmo/edax-reversi (besucht am 20. 01. 2023).

[34] Charles L. Bouton. „Nim, A Game with a Complete Mathematical Theory“. In: *Annals of Mathematics* 3.1/4 (1901), S. 35. ISSN: 0003486X.
DOI: 10.2307/1967631. URL: http://www.jstor.org/stable/1967631 (besucht am 20. 01. 2023).

[35] Wolfgang Konen und Samineh Bagheri. „Final Adaptation Reinforcement Learning for N-Player Games“. In: 29. Nov. 2021. URL: http://arxiv.org/pdf/2111.14375v1 (besucht am 25. 01. 2023).

[36] Wolfgang Konen und Samineh Bagheri. „Reinforcement Learning for N-Player Games: The Importance of Final Adaptation“. In: *Bioinspired Optimization Methods and Their Applications: 9th International Conference, BIOMA 2020, Brussels, Belgium, November 19–20, 2020, Proceedings*. Berlin, Heidelberg: Springer-Verlag, 2020, S. 84–96. ISBN: 978-3-030-63709-5.
DOI: 10.1007/978-3-030-63710-1{\textunderscore}7. (Besucht am 25. 01. 2023).

[37] Johannes Scheiermann und Wolfgang Konen. „AlphaZero-Inspired Game Learning: Faster Training by Using MCTS Only at Test Time“. In: *IEEE Transactions on Games* (2022), S. 1–11. ISSN: 2475-1502. DOI: 10.1109/TG.2022.3206733. URL: http://arxiv.org/pdf/2204.13307v3 (besucht am 25. 01. 2023).

[38] Donald F. Beal und Martin C. Smith. „Temporal Coherence and Prediction Decay in TD Learning“. In: *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*. IJCAI '99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc, 1999, S. 564–569. ISBN: 1558606130.

[39] Victor Allis. „A Knowledge-Based Approach of Connect-Four“. In: *ICGA Journal* 11.4 (1988), S. 165. ISSN: 1389-6911. DOI: 10.3233/ICG-1988-11410.

[40] Matthias Berg. 2010. URL: https://berg.earthlingz.de/xot/aboutxot.php?lang=en (besucht am 20. 01. 2023).

Anhang

Der GBG-Quellcode befindet sich auf `https://github.com/WolfgangKonen/GBG/tree/master/src/ludiiInterface`.

Nach aktuellem Stand verwendet die Ludii-GBG-Schnittstelle die Ludii Version 1.3.0.

Die für diese Arbeit erhobenen Spieldaten befinden sich auf einem USB-Stick, welcher der Arbeit beigefügt ist. Auf diesem befindet sich ebenfalls eine Anleitung zur Reproduktion der Ergebnisse mit den .gamelog Dateien der exemplarisch dargestellten Episoden, welche auch auf dem USB-Stick zu finden sind.

Namen der verwendeten KI-Agenten innerhalb des GBG-Frameworks

Verwendeter Bezeichner	Datei-Name des Agenten im GBG
AlphaBeta	1-AB.agt.zip
AlphaBeta-DL	1-AB-DL.agt.zip
TCL	2-TCL-EXP-NT3-al37-lam000-6000k-epsfin0.agt.zip
MWRAP	0-MWrap1000-TCL-EXP-NT3.agt.zip
MCTS	3-MCTS10000-40.agt.zip
Edax	edax2-d*.agt.zip (* =1,2,3,4)
TCL4	TCL4-100_7_250k-lam05_P4_nPly2-FAm_A.agt.zip
Bouton	Bouton.agt.zip
TDNT4	TDNT4-20k-project.agt.zip (für 5 Haufen) TDNT4-60k-project.agt.zip (für 7 Haufen) TDNT4-10-4-60k-project.agt.zip (für 9 Haufen) TDNT4-20-5-60k-project.agt.zip (für 9 Haufen)