Master Medieninformatik | Abschlussarbeit

Implementierung und Untersuchung eines Turniersystems für KI-Agenten in Brettspielen

vorgelegt von

Felix Barsnick, B. Eng.

Matrikelnummer: 11083708

zur Erlangung des akademischen Grades

Master of Science (M. Sc.)

Prof. Dr. Wolfgang Konen (Technische Hochschule Köln)

Prof. Dr. Christian Kohls (Technische Hochschule Köln)

Gummersbach, 15. April 2019

Fakultät für Informatik und Ingenieurwissenschaften

Technology
Arts Sciences
TH Köln

Masterarbeit

Titel: Implementierung und Untersuchung eines Turniersystems für KI-Agenten in Brettspielen

Gutachter:

- Prof. Dr. Wolfgang Konen (Technische Hochschule Köln)
- Prof. Dr. Christian Kohls (Technische Hochschule Köln)

Zusammenfassung:
Diese Arbeit behandelt die Implementierung und Untersuchung eines Turniersystems für KI Agenten, die gegeneinander Brettspiele spielen. Das Turniersystem soll dabei helfen, Aussagen über die Spielstärke der Agenten im Kampf gegeneinander zu treffen. Dafür können vor dem Turnier verschiedene Einstellungen getroffen werden. Während des Turniers misst das System, wie schnell die Agenten spielen und welche Ergebnisse sie erzielen. Dabei werden die Wertungssysteme Elo und Glicko zur Bestimmung der Spielstärke der Agenten untersucht. Am Ende werden die Messergebnisse ausgewertet und dem Nutzer übersichtlich präsentiert. Weiterhin werden in dieser Arbeit verschiedene Forschungsfragen zum Verhalten der Agenten in verschiedenen Turnieren untersucht.

Stichwörter: Künstliche Intelligenz, Brettspiele, Turniere, Elo-Rating, Glicko-Rating
Datum: 15. April 2019

Inhaltsverzeichnis

Abstract II

1 Einleitung 1
1.1 Motivation . . . 1
1.2 Begriffskonvention . . . 2

2 Grundlagen 3
2.1 Ursprung von Brettspielen . . . 3
2.2 Determinismus in Brettspielen . . . 3
2.3 Künstliche Intelligenzen spielen Brettspiele . . . 4
2.3.1 Schach . . . 5
2.3.2 Go . . . 5
2.3.3 General Game Playing . . . 6
2.3.4 General Video Game Playing . . . 6
2.4 General Board Gaming Framework . . . 7
2.5 GGP vs GBG . . . 7

3 Brettspiel-Wettbewerbe 8
3.1 Turnierformen . . . 8
3.2 Ermitteln von Spielerstärke . . . 10
3.2.1 Elo . . . 11
3.2.2 Glicko . . . 13
3.3 Ermitteln des Gewinners . . . 15
3.4 Stand der Forschung . . . 16

4 Die Turnier-Software 19
4.1 Anforderungen . . . 19
4.2 Architektur . . . 21
4.3 Klassen . . . 22
4.3.1 GUI Klassen . . . 23
4.3.2 TSAgentManager . . . 27
4.3.3 TSAgent . . . 28
4.3.4 TSResultStorage . . . 28
4.3.5 TSTimeStorage . . . 28
4.3.6 Hilfsklassen . . . 29

4.4 Bibliotheken 30
4.4.1 JHeatChart 30
4.4.2 Elo 30
4.4.3 Glicko2 31
4.5 Integration in das bestehende GBG Framework 31
4.5.1 XArenaMenu - GUI 31
4.5.2 Arena - Hauptschleife 32
4.5.3 XArenaFuncs - Wettkampffunktionen 32
4.5.4 AgentIO - Speichern und Laden 33

5 Herausforderungen der Implementierung 35
5.1 Zeitmessung in Java 35
5.2 Umgang mit Determinismus 36
5.3 Visualisierung der Ergebnisse 37
5.4 Zeitliche Dauer der Turniere 38
5.5 Auslastung des Arbeitsspeichers und Nutzung des Java Garbage Collectors 38

6 Messungen 40
6.1 Methodik 40
6.2 Messreihen 44
6.2.1 M1 - Kommutativität 44
6.2.2 M2 - Zufällige Startzustände 49
6.2.3 M3 - Approximative Rankings 52
6.2.4 M4 - Reproduzierbarkeit 57
6.2.5 M5 - Grenzen der Zeitmessung in Java 59

7 Evaluation/Diskussion 65
7.1 M1 - Kommutativität 65
7.2 M2 - Zufällige Startzustände 65
7.3 M3 - Approximative Rankings 66
7.4 M4 - Reproduzierbarkeit 67
7.5 M5 - Grenzen der Zeitmessung in Java 68

8 Zusammenfassung und Ausblick 70

Literaturverzeichnis 72

Anhang **76**
Tabellen zu Kapitel 6.1 76
Tabellen zu Kapitel 6.2 77

Eidesstattliche Erklärung **85**

1 Einleitung

1.1 Motivation

Menschen spielen miteinander gern Brettspiele und versuchen dabei natürlich auch zu gewinnen. Besonders motivierte Spieler treten auch in Turniere gegeneinander an. Dort können sie mit verschiedenen Taktiken gegeneinander antreten und nach verschiedenen Runden erfahren, wie gut sie sich behaupten konnten. Seit einigen Jahren können auch künstliche Intelligenzen (KI's) entwickelt werden, die solche Brettspiele spielen können. Verschiedene solcher Brettspiel-KI's können auch in virtuellen Turnieren gegeneinander antreten, um die beste der KI's zu bestimmen. Entwickler können so den Fortschritt ihrer Arbeit messen und bei verschiedenen Implementierungen die jeweilige Effizienz bestimmen. Auch können menschliche Spieler gegen diese KI's antreten, und so ihr Können erweitern und vielleicht auch untypische Spielarten oder Taktiken erlernen.
In dieser Ausarbeitung soll ein Konzept für eine Turnierfunktion für das General Board Game Playing Software Framework (GBG) entworfen werden. Mit dessen Hilfe sollen KI-Agenten gegeneinander antreten können, um eine Aussage über ihre Qualität und "Stärke" treffen zu können. Diese Qualität soll sich aus einer Reihe von Faktoren zusammensetzen. Dazu gehört aktuell, wie oft ein Agent gegen andere gewinnt oder verliert, wie schnell er seine Züge berechnet und die Dauer des Trainings vor dem Einsatz. Das General Board Game Framework (GBG) bietet die Möglichkeit künstliche Intelligenzen und Brettspiele zu implementieren. Bisher war es möglich, einzelne Messungen mit den Agenten in diesem Framework durchzuführen, aber es war kein umfangreiches Turnier- und Messsystem implementiert. Es war Ziel dieses Arbeit ein solches Turnier- und Messsystem zu entwickeln und Untersuchungen durchzuführen. Mithilfe dieses Messsystems sollen verschieden viele Agenten in den Spielen des GBG's gegeneinander antreten können und dann mithilfe der Messergebnisse Aussagen über die Qualität der einzelnen Agenten zu treffen.

Mit einem solchen Turniersystem sollen folgende Fragen untersucht werden:

F1 Wie stark „nicht-kommutativ“ ist eine Matrix aus Ergebnissen der Matches eines Turniers, d.h. wie sehr weichen W_{ij} und (1-W_{ij}) voneinander ab?

F2 Verändern Rankings sich stark, wenn man nicht nur vom Default-Board anfängt, sondern von zufälligen Startzuständen?

F3 Kann man aus wenigen Spielen mit Hilfe der Wertungssystemen Elo und Glicko2 gut auf die Rankings eines vollständigen Round-Robin-Turniers schätzen?

F4 Ist eine Reproduzierbarkeit der Messergebnisse gegeben?

F5 Wo liegen die Grenzen der Zeitmessungen bei sehr kurzen Intervallen in Java?

F6 Wie ermittelt man aus der Round-Robin-Matrix eine Kennzahl S_i für jeden Player 1, . . . , N, die ein faires Ranking darstellt?

Dafür werden in dieser Arbeit verschiedene Aspekte von klassischen Turnieren, Bewertungsschemata und Herausforderungen bei der Umsetzung dieses Projekts dargelegt.

1.2 Begriffskonvention

Während der Entwicklung des Turniersystems wurde es notwendig, eine Konvention für die Beschreibung von Vorgängen in Brettspielen zu definieren. Die Begriffe Spiel und Runde sind nicht eindeutig genug und können nicht präzise verwendet werden. Um diese Mehrdeutigkeiten zu verhindern, wurde folgende Konvention entwickelt:

- Agent: Digitaler Spieler eines Spiels (Software)
- Spiel: Das Spiel selbst (Hex, Tic Tac Toe, Vier Gewinnt, ...)
- Match: Wettkampf von Agenten in einem Spiel
- Episode: Wiederholung eines Wettkampfes der Agenten
- Zug: Aktion eines Agenten während eines Wettkampfes

Diese Konvention findet in dieser Ausarbeitung und auch der Softwareimplementation Anwendung.

2 Grundlagen

In diesem Kapitel werden die Grundlagen von realen und virtuellen Brettspielen erörtert. Darüber hinaus werden das General Board Gaming Framework und das General Game Playing vorgestellt.

2.1 Ursprung von Brettspielen

Ein Brettspiel ist laut Duden ein "Unterhaltungsspiel, das mit Figuren oder Steinen auf einem Spielbrett gespielt wird" [Dud18a]. Neben festen Regeln gibt es hier auch ein definiertes Spielfeld, auf dem die Spielzüge stattfinden. Brettspiele sind ein Teil der Obergruppe der Gesellschaftsspiele und begleiten die Menschen seit jeher. Solche Spiele wurden nicht nur im alten China, Ägypten, Griechenland und Afrika sondern auf der ganzen Welt gespielt. Diese dienten zum Zeitvertreib, aber auch zum Erwerb von Wissen [Web18].

2.2 Determinismus in Brettspielen

Brettspiele gibt es in den verschiedensten Variationen, Regeln und Komplexitäten. Ein Kriterium, um die Menge dieser Spiele zu kategorisieren ist die Unterscheidung in deterministische und nichtdeterministische Spiele. Determinismus beschreibt, dass Ereignisse exakt kausal vorbestimmt sind [Dud18b]. Es geschieht also nichts zufällig, sondern nur nach klar nachvollziehbaren und reproduzierbaren Regeln. Diese Unterscheidung kann große Auswirkungen in einem Turnier virtuell gegeneinander spielender Agenten haben.

Nichtdeterministische Spiele

Ein Spiel ist nichtdeterministisch, wenn es zufällige Elemente enthält. Dazu können Würfel oder gemischte Kartenstapel zählen, die den Spielfluss beeinflussen. Jedes Mal, wenn ein solches Spiel gespielt wird, entstehen neue Spielabläufe, und der spielende Agent muss auf die immer neuen Situationen reagieren. Wenn ein digital spielender Agent ein solches nichtdeterministisches Spiel wiederholt spielt und seine Reaktionen auf diese neuen Situationen beobachtet werden, können genauere Aussagen über sein "Können" und seine Qualität getroffen werden.

Deterministische Spiele

Ein Spiel ist deterministisch, wenn es keine zufälligen Elemente enthält. Der Spiel-

ablauf ist bei diesen Spielen nur von den Regeln des Spiels und den Entscheidungen des Spielers abhängig. Die umfangreiche Evaluation der Qualität eines Agenten in einem deterministischen Spiel kann nicht einfach dadurch ermittelt werden, dass er ein Spiel wiederholt spielt. Der Agent wird bei demselben Gegenspieler in jeder Runde wahrscheinlich dieselben Züge tätigen. Dadurch kann keine Aussage über sein Verhalten in verschiedenen Situationen getroffen werden. Um solche Agenten trotzdem in verschiedenen Spielverläufen evaluieren zu können, muss in den Spielablauf eingegriffen werden. Darauf wird in den folgenden Kapiteln noch näher eingegangen werden.

Determinismus in Agenten

Nicht nur die Spiele können deterministisch sein, auch die Agenten. Die Implementierung des Agenten entscheidet, ob ein Agent deterministisch spielt. Wenn in die Berechnung der Züge vom Zufall beeinflusste Berechnungen einfließen, besitzt er ein nicht deterministisches Verhalten.

2.3 Künstliche Intelligenzen spielen Brettspiele

Informatiker waren schon immer daran interessiert, das Potential vorhandener Hardware auszuschöpfen und damit beeindruckende Anwendungen zu realisieren. Dazu gehörten auch Programme der künstlichen Intelligenz (KI). Ziel dieser Programme ist es, menschliche Intelligenz nachzubilden. Auf diese Weise wurde es Computern auch ermöglicht, gegeneinander oder gegen Menschen (Brett-)Spiele zu spielen. Die Herausforderung ist dabei, einem Computer die Regeln und das Spielen eines Brettspiels beizubringen und ihm erfolgreiche Spielweisen zu vermitteln. Bei einfachen Spielen war es mit hoher Rechenleistung möglich, alle denkbaren Züge zu berechnen, um die Besten auszuwählen. Bei komplexeren Spielen, wie etwa Schach und besonders Go, ist das nicht mehr möglich. Deshalb stellen diese Spiele besondere Herausforderungen dar und verlangen neue Lösungen. Menschen versuchen an dieser Stelle den Gegner und seine Taktik zu analysieren und dann selbst entsprechend zu handeln. In Wettbewerben gibt es auch oft zeitliche Limitationen, wie viel Zeit ein Agent für die Berechnung seiner Züge oder das Match zur Verfügung hat. Diese Limitationen und die mit der Zeit zunehmende verfügbare Rechenkraft von Computern ermöglichten viele verschiedene Lösungsansätze in diesem Feld der Informatik.

2.3.1 Schach

Das Spiel Schach war die erste große Brettspiel-Herausforderung für KI Systeme. Im Jahr 1996 besiegte der Computer Deep Blue das erste Mal einen Schachweltmeister. Deep Blue war aus vielen speziellen Prozessoren gefertigt und von IBM programmiert worden. Er berechnete in jeder Sekunde eines Matches viele Millionen möglicher Züge und konnte so mithilfe seiner Rechenkraft seine Gegner besiegen [IBM12]. Deshalb ist es kein lernendes System, also streng genommen auch keine künstliche Intelligenz, da es nur versucht, so viele mögliche Züge zu berechnen wie es kann, um zu gewinnen. Es besaß keine Möglichkeit, aus seinen Fehlern zu lernen oder über viele Matches hinweg bessere Taktiken zu entwickeln.

2.3.2 Go

Das strategische Brettspiel Go stammt aus Asien und besteht aus einem 19x19 Linien großen Gitterfeld mit 361 Schnittpunkten. Gespielt wird mit weißen und schwarzen Spielsteinen. Es besitzt nur wenige einfache Regeln, aber aufgrund der großen Spielfeldgröße ist die Anzahl der möglichen Spielzüge enorm hoch. Ein menschlicher Spieler benötigt hier wenig Zeit, um die Regeln zu erlernen, aber sehr lang, um einen erfolgreichen Spielstil zu entwickeln. Aufgrund dieser hohen Komplexität entwickelte die Firma Google DeepMind die KI AlphaGo. Ziel war es, eine KI zu schaffen, die Go nicht programmatisch spielt, sondern eine KI, die eine erfolgreiche Spielweise erlernt. Mit diesem Ansatz hat AlphaGo in den letzten Jahren erfolgreich die weltweit besten Go-Spieler geschlagen. Bei dieser KI lassen sich selbst antrainierte neue Spieltechniken beobachten, die bisher noch nicht von Menschen gespielt wurden. Dadurch ergab sich die Situation, dass Go Spieler gegen AlphaGo spielen wollten, um von der KI zu lernen und ihren eigenen Spielstil zu verbessern. Aufgrund der enorm großen Anzahl an möglichen Spielzügen während einer Partie war es nicht möglich, einen Brute Force Ansatz wie Deep Blue im Schach zu verfolgen. Deshalb besteht AlphaGo aus mehreren neuronalen Netzen basierend auf Google's Tensorflow-Software.
Die neueste Version der nur auf das Spiel Go spezialisierten Software AlphaGo heißt AlphaGo Zero. Diese benötigt im Gegensatz zu AlphaGo keine Datenbank menschlicher Go-Spiele mehr sondern lernt nur aus Spielen gegen sich selbst ("from zero"). Sie benötigt ferner nur noch ein einzelnes Computersystem und keine verteilte Architektur und ist sehr viel stärker als AlphaGo. In 100 Runden, die AlphaGo Zero

gegen die AlphaGo Version spielte, die Lee Sedol besiegte, gewann AlphaGo Zero alle 100 Runden. AlphaGo besitzt ein Elo von 3739 und AlphaGo Zero 5185.
Diese Leistung wird nur noch von DeepMinds neuester Software AlphaZero übertroffen. Die verschiedenen AlphaGo Versionen wurden noch mit Spielen gegen Computer und Menschen trainiert, wohingegen AlphaGo Zero und AlphaZero nur die Regeln von Go kannte und dann immer wieder gegen sich selbst gespielt hat. Durch seine neue allgemeinere und verbesserte Architektur konnte AlphaZero in kurzer Zeit Go lernen und so stark werden, dass es AlphaGo Zero nach wenigen Tagen schlagen konnte. Da AlphaZero nur die Regeln eines Spiels benötigt und danach gegen sich selbst antritt, um zu lernen, ist es auch in der Lage erfolgreich Schach und Shogi zu spielen [SHS+].

2.3.3 General Game Playing

In den Beispielen von Deep Blue und AlphaGo wurden KIs vorgestellt, die nur ein spezielles Spiel sehr gut spielen konnten. Im Feld des General Game Playings (GGP) ist es das Ziel, Agenten zu entwickeln, die beliebige Spiele selbstständig erlernen und spielen können. Entwickelt wurde diese Art von Agenten von der Stanford Logic Group der Stanford Universität. Um einem Agenten ein Spiel "beizubringen" wurde die Game Description Language (GDL) entwickelt. Ein beliebiges, in der GDL codiertes Spiel, kann so von jedem GGP Agenten erlernt und gespielt werden. Die amerikanische Non-Profit-Organisation Association for the Advancement of Artificial Intelligence (AAAI) wurde 1979 gegründet und befasst sich seitdem mit der Erforschung und Förderung im Feld der künstlichen Intelligenz [AAA18]. Jährlich hält sie seit 1980 die AAAI Konferenz ab [AAA], bei der auch seit 2005 ein großer GGP Wettbewerb abgehalten wird. Die Stanford Universität führt über das Jahr den International General Game Playing Competition (IGGPC) Wettbewerb durch. Die besten Teilnehmer des IGGPC können am GGP Wettbewerb der AAAI teilnehmen [Gen18].

2.3.4 General Video Game Playing

Seit dem Jahr 2014 gibt es auch eine neue Art der Wettkämpfe, in denen Agenten Videospiele spielen können. Dabei liegt der Fokus auf 2D Arcade-Spielen. Auch hier ist die Motivation, dass ein Agent sich in einer breiten Auswahl an Spielen behaupten kann und nicht auf ein spezielles Spiel spezialisiert ist. In [PLST+16] wird die General Video Game Playing Competition vorgestellt. Dieser Wettkampf

ermöglicht es einzeln Agenten, in verschiedenen Videospielen anzutreten. Ähnlich der GDL in GGP Spielen wird für die Video-Spiele die video game description language (VGDL) genutzt. Seit 2016 gibt es ein solches Turnier auch für Spiele mit zwei Spielern mit der Two-Player GVGAI Competition [GPLL16]. In der Arbeit [GCS+18] wird das zugehörige Turnierframework vorgestellt.

2.4 General Board Gaming Framework

Das General Board Gaming Framework (GBG) wird seit Ende 2016 von Wolfgang Konen in der Programmiersprache Java entwickelt [Kon18a]. Es bietet die Möglichkeit, Brettspiele und Agenten dafür zu implementieren. Das Framework ist modular aufgebaut und stellt verschiedene Klassen und Interfaces zur Verfügung. Mithilfe dieser können Spiele selbst, Spiellogik und die Agenten implementiert werden. In [Kon17, Kon18b] steht ein Tutorial zur Benutzung des Frameworks und ein Technischer Report, der den wissenschaftlichen und ausbildungstechnischen Mehrwert von GBG beschreibt, zur Verfügung. Das GBG ist auf der Plattform GitHub unter der General Public License frei verfügbar und verfügt bereits über einige Spiele und Agenten. Der Großteil dieser Agenten verwendet dabei verschiedene KI Verfahren.

2.5 GGP vs GBG

Wie zuvor in Kapitel 2.3.3 vorgestellt, befasst sich das General Game Playing mit der Implementierung generischer Agenten, welche verschiedenste Spiele bestmöglich spielen können. Dadurch können verschiedenste Spiele mittels der Game Description Language für alle Agenten implementiert werden, die Agenten werden aber immer Generalisten und keine Spezialisten für konkrete Spiele bleiben.
Das General Board Gaming Framework befasst sich auf der anderen Seite mit der Implementierung von Agenten für einzelne konkrete Spiele. Die Implementierungen der Agenten können also spezielle Anpassungen an die jeweiligen Spiele enthalten. Dadurch sind diese Agenten des GBG in der Lage, einzelne Spiele sehr gut und sehr performant zu spielen, aber zur Laufzeit auch nur diese. Will man dem GBG neue Spiele hinzufügen, sind kleine programmtechnische Anpassungen für die Agenten vorzunehmen. Der Vorteil ist aber, dass der größte Teil des Agenten-Codes unverändert übernommen werden kann. Ferner kann GBG für ein spezifisches Spiel stärker spielende Agenten erzeugen als GGP. GGP Agenten können allerdings ein ihnen unbekanntes GDL kodiertes direkt spielen.

3 Brettspiel-Wettbewerbe

In diesem Kapitel werden verschiedene Turniersysteme zur Organisation eines Turniers und Algorithmen zur Bestimmung der Stärke einzelner Spieler behandelt. Mithilfe dieser Algorithmen können während oder nach einem Turnier vergleichbare Aussagen getroffen werden, wie gut ein einzelner Spieler ist.

3.1 Turnierformen

Es haben sich mit der Zeit einige verschiedene Turnierformen entwickelt. Diese unterscheiden sich in der Art und Weise, wie sie Spieler gegeneinander antreten lassen. Grundsätzlich lässt sich jede dieser Turnierformen mit jeder Menge an Spielern durchführen. Aber speziell bei größeren Mengen an Spielern kann dies bei manchen Turnierformen zu unpraktikabel langen Spielzeiten führen.
Die folgende Tabelle 1 soll veranschaulichen, wie viele Matches die verschiedenen Turnierformen dieses Kapitels mit verschieden vielen Spielern benötigen.

Anzahl Spieler	K.-o.	Einfaches RT	Doppeltes RT
2	1	1	2
4	3	6	12
8	7	28	56
12	11	66	132
20	19	190	380
n	$n-1$	$\frac{1}{2} * (n * (n-1))$	$n * (n-1)$

Tabelle 1: Anzahl Matches bei K.-o. System und einfachem/doppeltem Rundenturnier (RT)

Rundenturnier System

Eine sehr simple und weit verbreitete Turnierform ist das Rundenturnier (engl. Round Robin). Hier treten alle Spieler gegeneinander an. Die Siege, Unentschieden und Niederlagen jedes Spielers werden dabei erfasst. So kann am Ende ein Gewinner ermittelt werden. Dieses Verfahren wird beispielsweise im Schach verwendet, mit einer Darstellung der einzelnen Rundenergebnisse in einer Matrix (siehe Abbildung 1). Da in dieser Turnierform alle Spieler gegeneinander antreten, ist es nicht nötig die einzelnen Spielstärken vor dem Turnier zu kennen. Der Vorteil ist hier, dass alle Spieler gegeneinander spielen können und so die Chance haben ihre Stärken in jeder

[1] https://chesscentral.files.wordpress.com/2010/10/crosstable.jpg

WCSC 2010 Kanazawa 2010

		1	2	3	4	5	6	7	8	9	
1	**Shredder**	*	½	1	½	1	1	1	1	1	7.0 / 8
2	**Rondo**	½	*	½	1	½	1	1	1	1	6.5 / 8
3	**Thinker**	0	½	*	½	1	½	1	1	1	5.5 / 8
4	**Pandix Breakthrough**	½	0	½	*	½		1	1	1	4.5 / 7
5	**Deep Junior**	0	½	0	½	*		1	1	1	4.0 / 7
6	**Jonny**	0	0	½			*		1	1	2.5 / 5
7	**Darmenios**	0	0	0	0	0		*	1	1	2.0 / 7
8	**Fridolin**	0	0	0	0	0	0	0	*	1	1.0 / 8
9	**Hector For Chess**	0	0	0	0	0	0	0	0	*	0.0 / 8

(33 Games)

Abbildung 1: Beispiel einer Schach Spielergebnis-Matrix nach einem Rundenturnier[1]

Situation einsetzen zu können. Der klare Nachteil ist, dass mit diesem Verfahren bei vielen Spielern ein Turnier viel zu lang dauern würde.
Rundenturniere lassen sich in zwei Typen unterscheiden: Das **einfache/reine Rundenturnier** und das **Doppelrundenturnier**. Im einfachen Rundenturnier treten zwei Spieler einmal gegeneinander an. Es besitzt also nur einer der beiden die Möglichkeit einen Startzug zu setzen. Die Anzahl der zu spielenden Matches im einfachen Rundenturnier beträgt $\frac{1}{2} * (n * (n-1))$ wenn n die Anzahl der Spieler bezeichnet. Im Fall des Doppelrundenturniers spielt jedes Spielerpaar zweimal. Jeder Spieler erhält also einmal die Chance zu beginnen, woraus die doppelte Anzahl an Matches resultiert. Die Anzahl der zu spielenden Matches im Doppelrundenturnier Beträgt $n * (n-1)$ wenn n die Anzahl der Spieler bezeichnet. Deshalb ist speziell das Doppelrundentunier nur für eine "übersichtliche" Anzahl von Spielern geeignet (siehe Tabelle 1).

K.-o.-System / Pokalsystem

In diesem Turniersystem treten die Spieler paarweise gegeneinander an. Die jeweiligen Gewinner treten in den folgenden Runden gegeneinander an, bis nur noch ein Spieler übrig ist. Auf diese Art kann ein solches Turnier in kürzerer Zeit ausgetragen werden, allerdings können nicht alle Spieler gegeneinander antreten (siehe Abbildung 2). Die Anzahl der zu spielenden Matches im einfachen Rundenturnier Beträgt $n-1$ wenn n die Anzahl der Spieler bezeichnet. Die Aufteilung der Spieler in dieser

Turnierform kann also einen direkten Einfluss auf den Erfolg der einzelnen Spieler haben. Wenn möglich, sollte hier versucht werden, starke mit schwachen Spielern zu gruppieren, damit starke Spieler nicht zu früh ausscheiden; wodurch schwache Spieler länger im Turnier bleiben könnten als sie "sollten". Faire Spielerpaare kön-

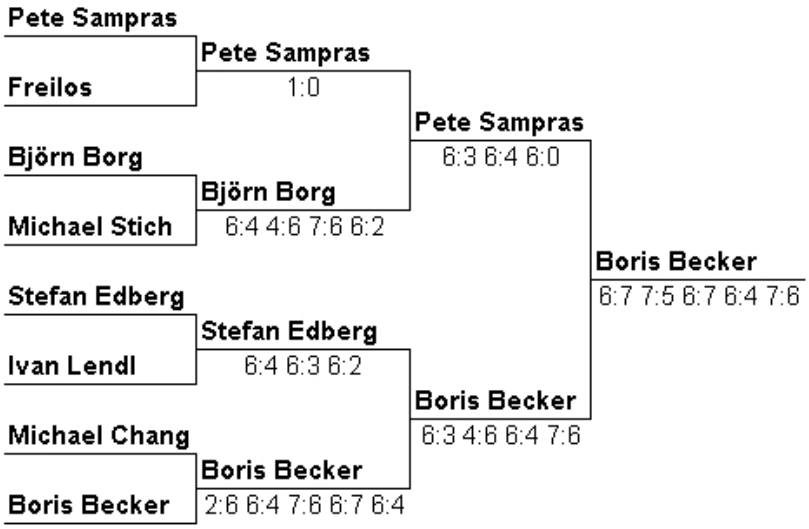


Abbildung 2: Beispiel eines Turnierablaufes nach dem Pokalsystem[2]

nen mithilfe der im folgenden Kapitel vorgestellten Algorithmen bestimmt werden, wenn sie vor dem Turnier schon probeweise gespielt haben.

3.2 Ermitteln von Spielerstärke

Für manche Turnierformen oder Ergebnisauswertungen ist es notwendig, die Stärke eines realen oder virtuellen Spielers zu bestimmen. Dafür lassen sich einfache Metriken nutzen, wie das bisherige Verhältnis aus gewonnenen und verlorenen Spielen. Eine solche Metrik erlaubt allerdings keine Vergleichbarkeit mit anderen Spielern, die gegen ganz andere Spieler gespielt haben. Um eine solche Vergleichbarkeit zu ermöglichen, entwickelten sich verschiedene Algorithmen, die die Stärke eines Spieles als Zahl darstellt und vergleichbar macht. Die Grundlage bildet hier die Elo-Zahl aus dem Jahr 1960 [Elo78]. Dieses Verfahren wurde in den folgenden Jahren von verschiedenen Personen und Firmen weiterentwickelt und schuf weitere Verfahren wie Bayes-Elo oder Glicko(2). Diese Algorithmen werden in den folgenden Kapiteln 3.2.1 und 3.2.2 vorgestellt.

[2]https://www.sport-software.de/fileadmin/screen/ko-e.gif

3.2.1 Elo

Der Physiker Arpad Elo entwickelte ab 1959 ein Wertungssystem, mit dessen Hilfe konnten Spieler aufgrund ihrer bisherigen Turnierergebnisse statistisch bewertet werden [Elo78]. Das Wertungssystem wurde im Schach 1960 [Elo61] von der United States Chess Federation (USCF) und 1971 [BDH+] von der internationalen FIDE übernommen. Mit den Jahren wurde es auch in einigen anderen Sportarten eingesetzt, um Spieler zu bewerten. Im Schach besitzen Anfänger ein Elo von unter 1000, Amateure bewegen sich im Bereich bis 2000, darüber befinden sich die Meister und ab 2500 die internationalen Großmeister. Die weltweit besten Schachspieler besitzen ein Elo von ca. 2800 [FID18]. Die Elo-Zahl eines neuen Spielers wird geschätzt. Mit jedem Spiel wird sein Elo aus den Stärken beider Spieler neu berechnet. Je nach Siegen oder Niederlagen werden die Elo-Zahlen beider Spieler nach jedem Spiel angepasst. In diese Anpassung des Spieler Elos fließt der sogenannte *k-Faktor* mit ein. Umso größer das Elo eines Spielers wird, umso kleiner wird sein k-Faktor. Dies hat die Konsequenz, dass Spieler mit niedrigem Elo und hohen k-Faktor bei einem Sieg ihr Elo stärker verbessern können als Spieler mit einem hohen Elo und niedrigem k Faktor.

Der Elo Algorithmus

Die Elo Zahl eines Spielers wird zu Beginn seiner Laufbahn geschätzt. Im allgemeinen wird hier ein Startwert von 1000 bis 1500 verwendet, auf der Website *chess.com* können Nutzer sich in diesem Spektrum selbst einschätzen. Nach der USCF wird das Rating eines neuen Spielers nach den Ergebnissen seiner ersten Spiele und der Elo-Werte seiner Gegner bestimmt [Uni]. Der bereits erwähnte k-Faktor ist zu Beginn relativ hoch, nach der FIDE bei 30 und bei der USCF 32. Nach jedem Spiel wird das Elo beider Spieler aktualisiert, dafür wird das bisherige Elo beider Spieler benötigt und wer von beiden gewonnen hat. Die Änderung des Elo-Wertes wird dann mithilfe eines Erwartungswertes und des k-Faktors bestimmt. Der Erwartungswert E wird wie folgt bestimmt:

$$E = \frac{1}{1 + 10^{\frac{Elo_{Gegner} - Elo_{Spieler}}{400}}} \tag{1}$$

Die Veränderung des Spieler Elos erfolgt mit folgender Formel:

$$Elo_{neu} = Elo_{alt} + k * (S - E) \tag{2}$$

Dabei beschreibt k den k-Faktor des Spielers, S das Spielergebnis (Spieler gewinnt S=+1, verliert S=–1, unentschieden S=0) und E den zuvor genannten Erwartungswert. Der k-Faktor verringert sich in Stufen mit steigender Anzahl an Spielen (FIDE) bzw. mit steigendem Elo (USCF) und erreicht sein Minimum mit 10 in der FIDE Implementierung und 16 nach der USCF.

$$k_{FIDE} = \begin{cases} 30 & \text{für} < 30 \text{ Spiele} \\ 15 & \text{für} > 30 \text{ Spiele \& Elo} < 2400 \\ 10 & \text{für} > 30 \text{ Spiele \& Elo} > 2400 \end{cases} \tag{3}$$

$$k_{USCF} = \begin{cases} 32 & \text{für Elo} < 2100 \\ 24 & \text{für } 2100 < \text{ Elo} < 2400 \\ 16 & \text{für Elo} > 2400 \end{cases} \tag{4}$$

Dieser Algorithmus findet vielfach Anwendung in verschiedenen Sportarten und Wettkampf-Online-/Computerspielen. Je nach Implementierung unterscheiden sich die Bestimmungen des k-Faktors, des Startwertes für neue Spieler und von Parametern im Erwartungswert. Nach jedem Spiel muss das Elo jedes Spielers aktualisiert werden, um die neue Spielstärke abzubilden. Durch die sinkenden k-Faktoren bei steigender Spielstärke kann ein Spieler zu Beginn seiner Laufbahn sein Elo schneller erhöhen als zu einem späteren Zeitpunkt mit höherem Elo.

Wertungsin-/deflation

Die Phänomene der Deflation und der Inflation beschreiben in der Volkswirtschaftslehre einen anhaltenden Trend der Zu- und Abnahme der Preise bei konstanter Kaufkraft der Konsumenten. Dies führt zu einem steigenden oder auch sinkenden Wert von Produkten, mit der negativen Folge, dass sich Kunden weniger Produkte leisten können und/oder Unternehmen weniger verdienen. Dieses Phänomen lässt sich auf die Elo-Zahl übertragen. Abhängig vom k-Faktor, erhält der eine und ver-

liert der andere Spieler nach einem Match Punkte. Bei ähnlichem k-Faktor sind dies annähernd gleich viele Punkte. Hat eine gewisse Menge an Spielern wiederholt gegeneinander gespielt, entsteht so eine Rangordnung unter ihnen mit den jeweiligen Elo Zahlen. Die Menge der Elo-Punkte der Spieler ist konstant, bei weiteren Spielen "tauschen" die Spieler nur noch Punkte untereinander. Kommt nun ein neuer Spieler neu dazu, beginnt er mit einem niedrigen Elo und senkt das Elo derjenigen, gegen die er gewinnt. Dadurch entnimmt er dem System Punkte und senkt das Elo der anderen geringfügig, obwohl sich deren Spielstärke nicht verändert haben muss. Diese Entwicklung kann im Schach über die letzten Jahrzehnte beobachtet werden, wo die Ratings von Top Spieler aus der nahen Vergangenheit nicht mehr mit den heutigen vergleichbar sind. Für eine vergleichbare damalige Platzierung werden heute mehr Punkte benötigt [Che].

3.2.2 Glicko

Der 1995 von Mark E. Glickman entwickelte Glicko Algorithmus [Gli13, Gli99] ist eine Weiterentwicklung des Elo-Ratings. Die Motivation, Glicko zu entwickeln, bestand darin, eine *Bewertungsabweichung* einzuführen. Diese verändert sich abhängig davon, wie häufig ein Spieler spielt. Der Elo-Score soll das Können eines Spielers abbilden, dieser kann aber nur aktualisiert werden, wenn auch Spiele erfolgen. Der Elo-Score eines häufig antretenden Spielers bildet sein Können sehr viel genauer ab, als der Score eines Gelegenheitsspielers. Spielen zwei Spieler mit demselben Elo-Score gegeneinander, wobei der eine selten und der andere regelmäßig spielt, würden beide im Fall einer Niederlage die selbe Menge Punkte verlieren. Im Glicko fließt die Spielhäufigkeit mit ein, dementsprechend würde der selten spielende Spieler weniger Punkte verlieren, als der regelmäßig spielende. Durch diese zusätzliche Berechnung der Bewertungsabweichung kann dieser Algorithmus realistischere Spielerbewertungen berechnen. Der Glicko-Algorithmus wurde bereits in einigen Online Schach Plattformen und Online-Computerspielen implementiert.
Als Weiterentwicklung entwarf Glickman den Glicko2 Algorithmus mit einer zusätzlichen *rating volatility*, was sich als Bewertungs-Sprunghaftigkeit übersetzen lässt. Dieser Parameter ermöglicht Rückschlüsse, wie konstant ein Spieler spielt.

Der Glicko2 Algorithmus

Zu Beginn erhält ein neuer Spieler ein Startrating von $r = 1500$ in der Elo-Skalierung und eine Startabweichung (Rating Deviation) von $RD = 350$. Die Sys-

temkonstante τ sollte laut Glickmann einen Wert zwischen 0.3 und 1.2 besitzen, Standartwerte sind hier 0.6 und 0.75. Wird ein kleiner Wert für die Systemkonstante gewählt reagiert das Rating träger auf Schwankungen in den Spielergebnissen. Zuletzt besitzt die Volatilität σ den Standardwert 0.06. Die Berechnungen werden in der Glicko Skalierung der Werte durchgeführt, dafür wird ein Umrechnungsfaktor von 173.7178 benötigt.
Der Algorithmus wird in [Gli12] im Detail beschrieben, allgemein wird er in dieser Arbeit wie folgt beschrieben:

1. Zu Beginn müssen für die einzelnen Spieler die oben genannten Werte gesetzt werden. Neue Spieler erhalten die Standartwerte, schon bestehende nutzen RD und ϕ aus vorherigen Runden.

2. Es folgt die Umrechnung des Ratings und der RD in die Glicko-Skalierung mithilfe des genannten Umrechnungsfaktors. In der Glicko-Skalierung erhält das Rating r die Bezeichnung μ und RD erhält ϕ.

3. Berechnung der Menge ν. Dies ist die geschätzte Varianz des Spielers Bewertung basiert nur auf Spielergebnissen.

4. Berechnung der Menge Δ, die geschätzte Verbesserung der Bewertung durch Vergleich der Bewertung vor der Periode für die Leistungsbewertung nur basierend auf den Spielergebnissen.

5. Berechnung der neuen Volatilität σ'. Diese Berechnung erfordert eine Iteration.

6. Aktualisierung der Bewertungsabweichung auf den neuen Wert für die Vorbemessungsdauer ϕ.

7. Aktualisierung der Bewertung und RD auf die neuen Werte μ' und ϕ'.

8. Umrechnung von Rating und RD in die ursprüngliche Elo-Skalierung.

An diesem Ablauf ist zu erkennen, dass der Glicko intern mit einem eigenen Wertesystem arbeitet, dies aber einfach in das System des Elo umgerechnet werden kann. Diese Eigenschaft ermöglicht es Elo und Glicko Ratings miteinander vergleichen zu können.

3.3 Ermitteln des Gewinners

Der Gewinner eines Turniers kann im Fall eines Rundenturniers am Ende anhand der gewonnen, unentschiedenen und verlorenen Spiele ermittelt werden. Ein Tripel aus diesen drei Kennzahlen für jede Partie, dargestellt in einer Tabelle, gestaltet sich allerdings recht unübersichtlich (siehe Abbildung 3). Die Diagonale dieser Matrix bleibt leer, da ein Agent nicht gegen sich selbst spielt. Dies ist hier mit dem Wort *null* gekennzeichnet. Eine Zusammenfassung dieser Kennzahlen zu einer ein-

Y vs X	Random	Minimax	MC-N	MCTS
Random	null	W:0 | T:0 | L:3	W:0 | T:0 | L:3	W:0 | T:0 | L:3
Minimax	W:3 | T:0 | L:0	null	W:3 | T:0 | L:0	W:3 | T:0 | L:0
MC-N	W:3 | T:0 | L:0	W:3 | T:0 | L:0	null	W:2 | T:0 | L:1
MCTS	W:3 | T:0 | L:0	W:2 | T:0 | L:1	W:3 | T:0 | L:0	null

Abbildung 3: Spielergebnis Matrix mit Win, Tie und Loss Einträgen

zigen Zahl ermöglicht es, schnell auf einen Blick in die Ergebnismatrix zu erkennen, welcher Spieler wie gut abgeschnitten hat. Ein solcher Score kann gebildet werden, indem die Summe der gewonnen Spiele mit dem Faktor 1.0, die unentschiedenen mit dem Faktor 0.5 und die verlorenen mit dem Faktor 0.0 multipliziert wird. Diese drei Werte werden dann addiert, um die Kennzahl (WTL Score) des Spielers zu erhalten (siehe Abbildung 4). Die Darstellung kann noch weiter vereinfacht werden, indem die einzelnen Scores in einer Heatmap farblich kodiert werden, ähnlich wie in [SPLR16]. So kann auf einen Blick identifiziert werden, welche die besten Agenten sind (siehe Abbildung 5 für ein Beispiel).

Y vs X	Random	Minimax	MC-N	MCTS
Random	null	0.0	0.0	0.0
Minimax	3.0	null	3.0	3.0
MC-N	3.0	3.0	null	2.0
MCTS	3.0	2.0	3.0	null

Abbildung 4: Spielergebnis Matrix nur mit Score Einträgen

Die Visualisierung der Messergebnisse des Turniers mittels Heatmap und Ergebnistabellen bieten eine einfache und schnelle Möglichkeit sich einen Überblick über die Turnierergebnisse zu verschaffen. Bei Turnieren mit größerer Menge an Teilnehmern kann es bei wachsender Größe der Heatmaps und Tabellen unübersichtlich werden. Deshalb werden für jeden Agenten auch die zuvor vorgestellten Elo und Glicko2 Scores berechnet. Mit ihrer Hilfe sollen detaillierte Informationen zur Spielerstärke dargestellt werden können.

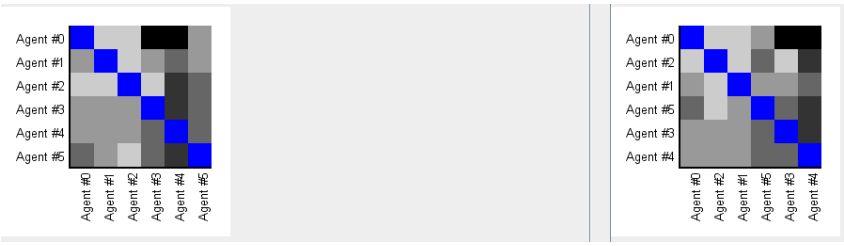


Abbildung 5: Spielergebnis Heatmap auf Basis von Score Einträgen (links unsortiert, rechts vertikal sortiert nach Spielstärke)

3.4 Stand der Forschung

Wie bereits vorgestellt, wurden die frühen Wettbewerbe von künstlichen Intelligenzen in Spielen wie Schach durchgeführt. Dort traten Computer erst gegeneinander und dann auch gegen Menschen an (siehe Kapitel 2.3.1). In den Jahren danach folgten Wettbewerbe in anderen Brettspielen wie MS Packman, diese wurden dabei innerhalb der einzelnen Spiele abgehalten [PLST+16]. Dies änderte sich mit der Entwicklung des General Game Playings und der zugehörigen Game Description Language [LHH+08] (siehe Kapitel 2.3.3). Diese erlaubten es, die gleichen Agenten in verschiedenen Spielen antreten zu lassen. Seit 2005 wird jährlich die International General Game Playing Competition (IGGPC) abgehalten [GLP05]. Dort treten viele Teams gegeneinander an, die besten werden für das Turnier der jährlichen AAAI Konferenz zugelassen [Gen18]. Im Zuge dieser Turniere wurden verschieden Verfahren zur Bestimmung der Stärke von Spielern erprobt. Der erste dieser Algorithmen ist die Elo Zahl (siehe Kapitel 3.2.1), ursprünglich entwickelt im Schach. In den folgenden Jahren wurde Elo unter verschiedenen Namen weiterentwickelt, um den komplexen Ansprüchen der Turniere gerecht zu werden. Zu den bekanntesten dieser Nachfolger gehören Glicko/Glicko2 von Mark Glickmann (siehe Kapitel 3.2.2) und TrueSkill der Firma Microsoft [HG06]. Untersuchungen der verschiedenen Algorithmen kommen zu dem Ergebnis, dass aktuell Glicko2 das effektivste Wertesystem ist in diesem Feld [SPLR16]. Die Weiterentwicklung der GGP sind die General Video Game Playing Wettbewerbe (GVGP), in denen verschiedene Videospiele gespielt werden. Im internationalen GVGAI Wettbewerb treten Agenten seit 2014 gegeneinander an [PLST+16]. Die GVGP Wettbewerbe gibt es als Einzelspieler oder auch mit zwei Spielern, die gegeneinander antreten [GPLL16, GCS+18].

GGP Wettkämpfe

Wie in [GLP05] beschrieben, besteht das Wettbewerbssystem im GGP aus einem

Hauptcontroller, genannt Game Manager, der die Spiele und Agenten verwaltet und steuert. Die Kommunikation erfolgt dabei über TCP/IP. Vor einem Match wird der Game Manager konfiguriert. Dazu erhält er das Spiel selbst, Anzahl der Agenten, die Vorbereitungszeit am Anfang in Sekunden (*startclock*) und die Zeit pro Zug im Spiel (*playclock*). Zu Beginn eines Matches erhalten die Agenten einen Start-Befehl, um ihre Initialisierung durchzuführen. Danach ruft der Game Manager nacheinander die Agenten auf, übergibt ihnen den aktuellen Zustand des Spiels und wartet dann auf den Zug des Agenten. Wenn der Zug nicht innerhalb eines vorgegeben Zeitlimits erfolgt, wird dem ein Agenten, als Strafe, ein zufälliger legaler Zug vorgegeben. Am Ende des Matches sendet der Game Manager ein Stop-Signal an alle Spieler, woraufhin diese sich beenden. Der Game Manager speichert den Verlauf und das Ergebnis des Matches in seiner Arcade Datenbank.

GVGP Wettkämpfe

In [PLST+16] beschreiben die Autoren der General Video Game Playing Competition ihr Wettkampfsystem und dessen Funktion. Auch in diesem Turnier fungiert ein Computer als Hauptcontroller und steuert per TCP/IP Befehle die Abläufe der einzelnen Matches. Die Agenten werden in diesem Wettbewerb Controller genannt. Jeder Controller muss dabei über zwei Methoden erreichbar sein, eine zum initialisieren und eine weiter für die einzelnen Spielzüge mit dem Namen *act*. Nach dem Aufruf der Initialisierung hat der Controller eine Sekunde CPU Zeit, um sich zu initialisieren und turnierbereit zu werden. Die Berechnung der Züge mittels der *act* Funktion steht eine Zeit von 40ms zur Verfügung. Dauert die Berechnung zwischen 40 und 50ms Sekunden wird der Zug nicht ausgeführt, aber der Controller bleibt im Spiel. Wird auch das zweite Limit von 50ms verletzt, wird ein Controller aus dem Match des Spiels disqualifiziert. Beiden Methoden werden bei jedem Aufruf Informationen mittels einer StateObservation übergeben. Darin finden sich Informationen zum Spielzustand, der Zeitdauer in der die Methode ausgeführt werden muss, die erlaubten Spielzüge des Controllers und weitere Informationen zum Spiel selbst. Ähnlich der Arcade Datenbank im GGP Wettbewerb besitzt auch das GVGP Turniersystem eine Datenbank mit Informationen zu den einzelnen Wettkämpfen. Weiterhin wird dort gespeichert, ob die einzelnen Controller ordnungsgemäß funktionieren und ob Fehler in der Ausführung existieren. Diese Informationen können in einem Browser abgerufen werden. Das Turnierframework besitzt 30 Spiele mit jeweils fünf Level, von denen 10 zum trainieren vor dem Wettbewerb, 10 zum va-

lidieren im Wettbewerb und 10 zum Bestimmen der final Rankings gedacht sind. Um den Gewinner zu ermitteln, werden die Controller nach erreichten Punkten und Zielen in den Spielen sortiert. Wenn mehrere Agenten dasselbe Ergebnis erreichen gewinnt der, der die kürzeste Zeit zur Berechnung seiner Züge benötigt hat.

Weitere Spiele

Neben den GGP und GVGP Wettbewerben aus den letzten Jahren gab es davor auch schon einige (größere) Wettbewerbe in Brett- und Videospielen. In der Recherche zu dieser Arbeit tauchten dazu immer wieder die Spiele *Pac-Man*, die neuere Version *MS Pac-Man* und *MS Pac-Man vs Ghost* auf. In den Datenbanken wie IEEE und ACM finden sich viele Arbeiten zu Agenten und Algorithmen, die zum Einsatz kamen, um in den Turnieren dieser Spiele teilzunehmen. Es war dem Autor allerdings nicht möglich, Informationen zu den Wettbewerben selbst oder Details zu den Wettbewerbsframeworks zu finden. Rückschlüsse auf deren innere Funktion sind so also nicht möglich.

4 Die Turnier-Software

n diesem Kapitel werden die Implementierung und der Aufbau der Turnier-Software erläutert. Es erfolgt die Darstellung der Funktionsanforderungen und der Programmarchitektur. Weiterhin werden die Klassen des Turniersystems sowie deren Funktionen, die verwendeten Bibliotheken und die Integration in das bestehende GBG Framework erläutert.

4.1 Anforderungen

Die Implementierung dieses Projekts soll folgende Anforderungen erfüllen, um ein umfangreiches Turnier- und Messsystem zu realisieren:

A.1 **Übersichtliche graphische Oberflächen:** Die graphischen Oberflächen zum Konfigurieren des Turniers und Anzeigen der Ergebnisse sollen übersichtlich und schlicht gestaltet sein. Aufgrund der hohen Menge an Messwerten ist es weder sinnvoll noch hilfreich alle darzustellen. Die Darstellung von relevanten Auswertungen und Analyseergebnissen ist an dieser Stelle zielführend. Schaltflächen können hier dem Nutzer die Möglichkeit bieten die Menge der dargestellten Informationen zu erweitern.

A.2 **Beliebig viele Agenten aus dem Speicher laden können:** Es soll möglich sein beliebig viele, mit dem GBG-Framework erzeugte, Agenten aus dem Speicher in das Turniersystem laden zu können. Auf diese Weise können verschiedene Parametrisierungen der Agenten gegeneinander getestet werden. Außerdem wird so eine Reproduzierbarkeit von Messungen ermöglicht, da mithilfe dieser Funktionalität die gleichen Agenten wiederholt verwendet werden können.

A.3 **Zufällige Starthalbzüge:** In den Turnieren soll es möglich sein, eine beliebige Anzahl an Starthalbzügen zufällig zu generieren und den Agenten vorzugeben. Diese sollen zufällig vor dem Turnier erstellt werden. Dabei sollen so viele Zustände wie eingestellte Episoden (Wiederholung eines Matches) erstellt und anschließend für alle Matches verwendet werden. Dadurch soll sichergestellt sein, dass die gleichen Bedingungen für jedes Match gelten und eine Vergleichbarkeit gegeben ist.

A.4 **Verschiedene Turniermodi:** In der Konfiguration des Turniers soll es möglich sein, verschiedene Turniermodi zu verwenden. Dadurch kann gemessen werden, wie sich Agenten bei verschiedenen und verschieden vielen Matches behaupten können. Auch bieten diese eine Flexibilität, beispielsweise bei der Verwendung von sehr vielen Agenten, um die Dauer des Turniers zu begrenzen (siehe Kapitel 3.1).

A.5 **Match mit beliebig vielen Episoden wiederholen:** Jedes Match der Agenten soll beliebig oft wiederholt werden können, um zufällige Schwankungen zu verhindern. Eine einzelne Messung ist oft nicht aussagekräftig, speziell wenn das Spiel und/oder die verwendeten Agenten nicht deterministisch sind. Wiederholungen der Matches mithilfe beliebig vieler Episoden soll helfen diese Schwankungen auszugleichen.

A.6 **Abspeichern der Turnierergebnisse:** Am Ende des Turniers soll es möglich sein, alle Messergebnisse des Turniers als Datei abzuspeichern. Dadurch können Ergebnisse im Nachhinein wiederholt betrachtet oder weitergehend analysiert werden. Die Turnierergebnisse sollen sämtliche Messergebnisse des Turniers enthalten.

A.7 **Detaillierte Messung bis hin zum einzelnen Zug:** Während des Turniers sollen viele detaillierte Informationen erfasst werden, um umfangreiche Analysen zu ermöglichen. Deshalb sollen für jede Episode jedes Matches erfasst werden, welcher Agent gewonnen hat und wie lang jeder einzelne Zug gedauert hat (so präzise wie möglich). Auf Basis dieser Messdaten werden umfangreiche Analysen und Auswertungen möglich.

A.8 **Visualisierung der Messergebisse:** Anforderung A.7 gibt vor, dass eine große Menge an Messwerten erfasst werden soll, aber Anforderung A.1 gibt vor, dass die graphischen Ausgaben übersichtlich sein sollen. Zu viele Informationen würden den Nutzer überfordern, deshalb ist es wichtig, dass geeignete Visualisierungen und Vorverarbeitungen der Messdaten erfolgen.

A.9 **Implementierung von Elo und Glicko:** Zusätzlich zu den Messwerten aus Anforderung A.7 sollen über das Turnier hinweg Spielstärken der Agenten bestimmt werden. Dafür sollen die weit verbreiteten Algorithmen Elo und Glicko2 implementiert werden (siehe Kapitel 3.2.1 und 3.2.2). Diese sollen auch Teil der abgespeicherten Turnierergebnisse aus Anforderung A.6 sein.

Diese genannten Anforderungen stellen eine hinreichende Bedingung für die Durchführung und technische Realisierung dieses Projekts dar. Diese wird in den folgenden Kapiteln vorgestellt.

4.2 Architektur

Das Turniersystem wurde als Bestandteil des GBG-Frameworks implementiert. Es lässt sich über einen eigenen Unterpunkt im Hauptmenü der GBG GUI starten. Das Turniersystem setzt sich dabei aus Klassen zur Erzeugung der GUI, Ausführung der Hauptlogik, Hilfsklassen und Bibliotheken zusammen. Diese Klassen werden im folgenden Kapitel 4.3 und 4.4 im Detail erläutert. In Kapitel 4.5 wird weitergehend auf die Integration des Turniersystems in das bestehende GBG Framework eingegangen. Sämtliche Klassen sind, wie der Rest des GBG-Frameworks, in der Programmiersprache Java geschrieben. Dabei wird mindestens die Java Version 1.8 benötigt, um das Turniersystem betreiben zu können. Die Implementierung erfolgte mit der Entwicklungsumgebung IntelliJ der Firma JetBrains in der Ultimate 2018 Version.
Der Aufruf des Menüs des GBG-Framworks wurde in die games.XArenaMenu Klasse eingefügt, hier wurde die neue Methode generateTournamentMenu() hinzugefügt. Mit ihrer Hilfe werden zwei Menüpunkte erzeugt. Der erste startet die Turniersystem-Konfigurations-GUI, der zweite ermöglicht es, alte gespeicherte Turnierergebnisse zu laden und anzuzeigen. Die Konfigurations-GUI-Klasse TSSettingsGUI2 initialisiert dann zum Start eines Turniers die Klasse TSAgentManager, welche mit rund anderthalbtausend Zeilen den Hauptteil der Turnierlogik enthält. Hier werden alle weiteren Turnierklassen initialisiert und verwaltet. Die Klasse TSAgent verwaltet dabei alle Informationen rund um die spielenden Agenten, TSResultStorage speichert sämtliche Spielergebnisse und TSTimeStorage verwaltet die detaillierten Zeitmessungen. Zur Durchführung von Turnieren verwendet das GBG eine Hauptschleife in der games.Arena Klasse, dieser wurde ein neuer Modus für das Turnier hinzugefügt, in dem alle Schritte des Turniers durchgeführt werden. Am Ende des Turniers wird die zweite GUI TSResultWindow aufgerufen, um die Spiel- und Messergebnisse zu visualisieren. Darüber hinaus gibt es einige Hilfsklassen als dedizierte und innere Klassen, die beim Transferieren und Erfassen von Daten helfen.

4.3 Klassen

Im vorherigen Kapitel 4.2 wurden die Klassen des GBG Turniersystems vorgestellt. Dieses Kapitel liefert nun einen detaillierteren Einblick in die Implementierung der einzelnen Klassen. Dabei werden zuerst die beiden GUI Klassen behandelt (Kapitel 4.3.1), danach folgen die Hauptklassen TSAgentManager (Kapitel 4.3.2), TSAgent (Kapitel 4.3.3), TSResultStorage (Kapitel 4.3.4) und TSTimeStorage (Kapitel 4.3.5). Darauf folgen dann die dedizierten Hilfsklassen in Kapitel 4.3.6 und zuletzt die Bibliotheken in Kapitel 4.4. Abbildung 6 veranschaulicht das Zusammenspiel dieser Klassen schematisch. Einige der Klassen implementieren das Java

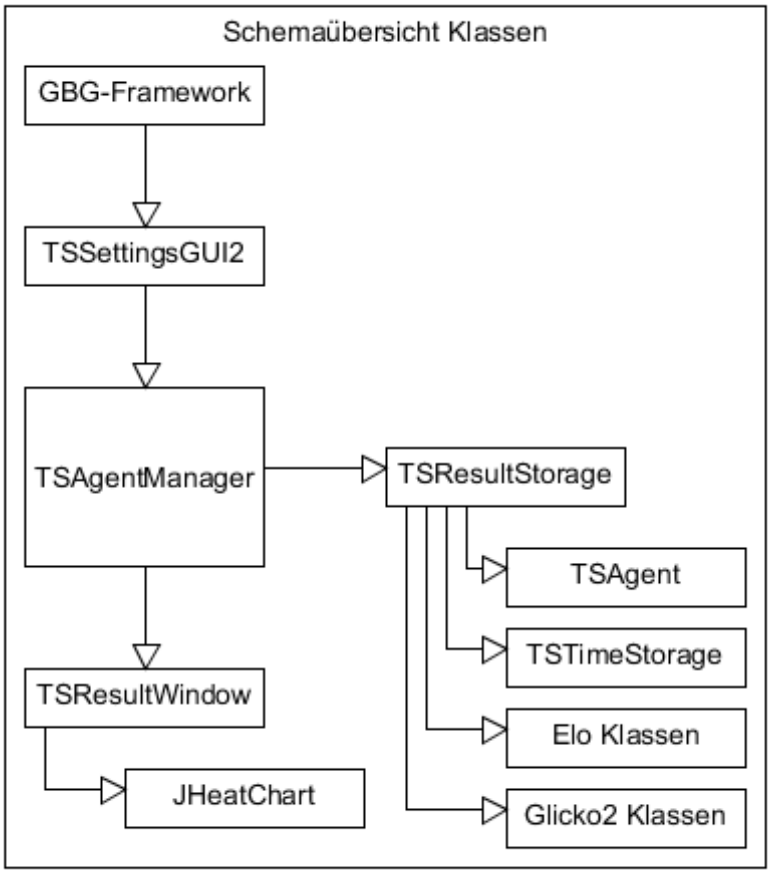


Abbildung 6: Schematische Übersicht der Klassen des Turniersystems ohne intern verwendete Hilfsklassen

Interface zur Serialisierung, um als Datei gespeichert werden zu können und somit die Anforderung A.6 zu erfüllen. Methoden zum Speichern und Laden der Turnierergebnisse wurden dem GBG Framework hinzugefügt (siehe Kapitel 4.5.4).

4.3.1 GUI Klassen

Das Turniersystem besitzt zwei graphische Oberflächen (GUIs), die im vorherigen Kapitel erwähnt wurden. Beide wurden mit dem GUI Designer der IntelliJ Entwicklungsumgebung umgesetzt. Dieser GUI Designer generiert eine Java Klasse für die Logik und eine .form Datei im XML Format zur Bearbeitung der GUI mittels graphischer Oberfläche. Aus dieser form Datei wird anschließend der Swing GUI Java-Code automatisch generiert. Mithilfe dieses Plugins konnten die GUIs schnell und einfach entworfen werden, ohne den Swing Code selbst schreiben zu müssen.

TSSettingsGUI2

Zu Beginn des Turniers wird ein Fenster zur Konfiguration der Parameter geöffnet

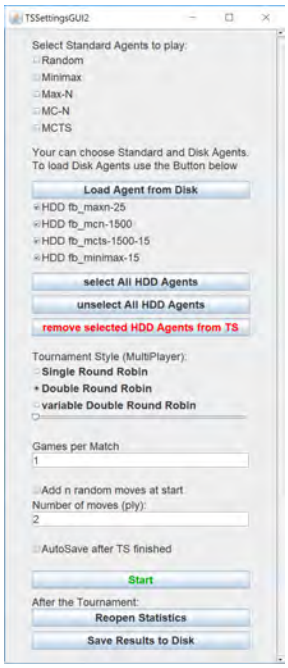


Abbildung 7: Einstellungen für ein Turnier in der graphischen Oberfläche des Turniersystems

(TSSettingsGUI2, siehe Abbildung 7). Ganz oben in diesem Fenster können die vorhandenen Standard Agenten des GBG Frameworks ausgewählt werden, diese Auswahl verändert sich abhängig davon, ob das gewählte Spiel deterministisch ist. Darunter befindet sich die Möglichkeit auf der Festplatte gespeicherte Agenten in das Turnier zu laden, auszuwählen und auch wieder zu löschen. Diese müssen vorher im GBG parametrisiert und gespeichert werden. Angezeigt werden Agenten aus dem Speicher mit dem Dateinamen.

Unter dem Bereich der Agenten aus dem Speicher befindet sich die Konfiguration des Turniermodus. Wie in Kapitel 3.1 vorgestellt, kann hier ein einfaches Rundenturnier (Single Round Robin) oder ein Doppeltes Rundenturnier (Double Round Robin) gewählt werden. Unterhalb des Double Round Robin gibt es einen weiteren Punkt des variable Round Robin. Dahinter verbirgt sich ein doppeltes Rundenturnier, dessen Anzahl an Matches mit einem Schieberegler reduziert werden kann. Es muss allerdings jeder Agent einmal spielen können, deshalb kann die Anzahl an Matches nicht beliebig weit eingegrenzt werden.
Darauf folgt die Eingabe, wie viele Episoden pro Match gespielt werden sollen und darunter die Möglichkeit zufällige Halbzüge zu aktivieren. Diese zufälligen Halbzüge werden vor Turnierbeginn errechnet und dann für alle Matches angewandt, damit jedes Match dieselben Startzustände besitzt (entsprechend Anforderung A.5).
Im unteren Bereich des Fensters befindet sich eine Checkbox, um das automatische Abspeichern der Turnierergebnisse auf die Festplatte zu aktivieren.
Darunter befindet sich der Startknopf, der das Turnier startet, wenn alle benötigten Eingaben vorhanden sind. Zuletzt gibt es noch Schaltflächen, um das Statistikfenster nach Ende des Turniers wiederholt zu öffnen und um ein fertiges Turnier in einem beliebigen Ordner auf der Festplatte zu speichern.

TSResultWindow

Die Ergebnisse des Turniers werden direkt nach dem Turnier oder mittels der GUI aus dem Speicher in der TSResultWindow Klasse visualisiert (siehe Abbildung 8). Diese besteht aus einigen Tabellen und Heatmaps zur Darstellung der Matchergebnisse und Zeitmessungen.
Am unteren Rand des Fensters befinden sich zwei Buttons zum Ausblenden und Einblenden aller Tabellen. Nach Klicken zum Anzeigen aller Tabellen sieht der Nutzer alle im folgenden beschriebenen Tabellen. Jede Tabelle kann darüber hinaus einzeln mit einem eigenen Button ein- und ausgeblendet werden.
Die Tabellen und Heatmaps beinhalten alle nur eine kurze Bezeichnung für jeden Agenten, um die Namen kurz zu halten. In der vorletzten Tabelle erfolgt die Zuweisung der Kurznamen zu den Dateinamen der Agenten.
Ganz am oberen Rand des Fensters befindet sich eine Zeile mit Informationen zum Turnier. Diese beinhaltet den Startzeitpunkt des Turniers, Anzahl der Matches, die Anzahl der Episoden pro Match und die Anzahl der zufälligen Halbzüge zu Beginn jedes Matches. Die oberen beiden Tabellen stellen jeweils eine Matrix dar, in denen

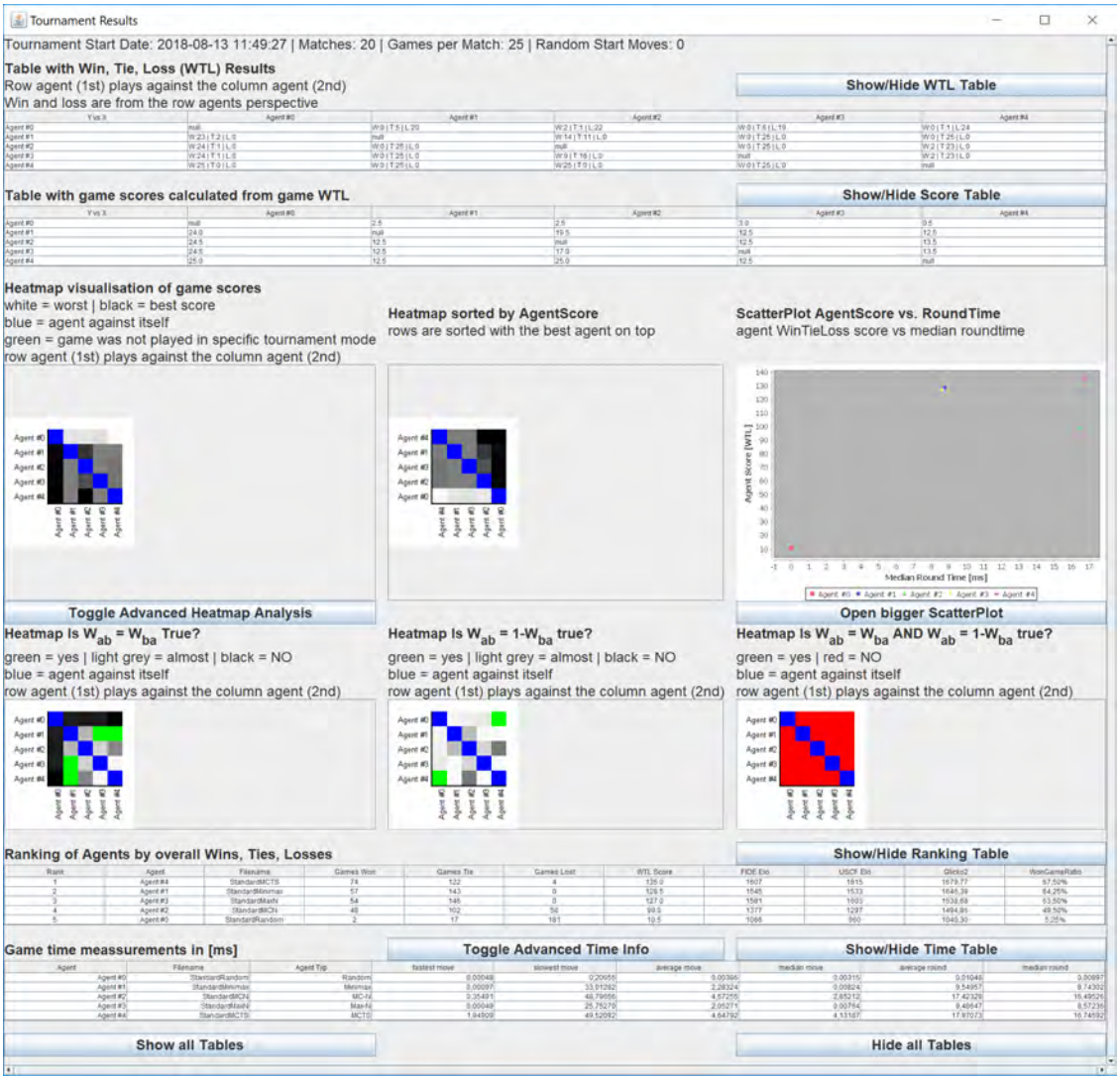


Abbildung 8: Ergebnisse und Statistiken eines Turniers

die Ergebnisse der einzelnen Episoden der Matches zu finden sind. Diese sind in der oberen Matrix als WTL aufgeschlüsselt, also Win (Sieg), Tie (Unentschieden) und Loss (Niederlage). Die Zellen stellen dabei die Ergebnisse des Matches dar, in denen der Agent der Zeile gegen den Agenten der Spalte spielt und die WTL Ergebnisse aus Sicht des Zeilenagenten zu lesen sind. Dabei ist zu erkennen, dass die Diagonale von oben links nach unten rechts keine Ergebnisse enthält, dies sind die Matches, in denen ein Agent gegen sich selbst gespielt hätte, diese Matches haben nicht stattgefunden. Die Untere der oberen beiden Tabellen zeigt eine kompaktere Darstellung, in der die einzelnen W, T und L Werte zu einem WTL Score verrechnet sind. Dazu wird die Anzahl der Siege mit dem Faktor 1,0, die Unentschieden mit dem Faktor

0,5 und die Niederlagen mit dem Faktor 0,0 multipliziert. Anschließend wird die Summe dieser drei Ergebnisse gebildet. Diese einzelne Zahl ermöglicht eine schnelle und einfache Übersicht über die Ergebnisse aller Matches.
Mittig im Ergebnisfenster befinden sich Visualisierungen der oberen Matrizen in Form von Heatmaps. Eine Heatmap stellt Datenpunkte farblich dar und kodiert die Höhe des Datenpunktes farblich. Die Heatmaps visualisieren die Daten der WTL Score Matrix. Die linke Heatmap stellt die Daten genauso dar wie in der darüber befindlichen Matrix, in der rechten Heatmap wurde die y-Achse nach WTL Scores sortiert und die x-Achse so, dass die Diagonale wiederhergestellt ist. In dieser Ansicht sind die Agenten zeilenweise von oben nach unten sortiert. Rechts in dieser Zeile befindet sich ein (Scatter-)Plot, in dem die Zeit pro Match im Median gegen den WTL Score aufgetragen ist. Ein bestmöglicher Agent würde sich hier im linken oberen Bereich befinden, also eine sehr hohe Spielstärke mit sehr kurzer Zeit zum Berechnen der Spielzüge. Der Plot lässt sich, durch einen Klick auf den darunter befindlichen Button, auch größer in einem neuen Fenster öffnen, zur besseren Lesbarkeit.
Unter der linken Heatmap befindet sich ein Button mit der Aufschrift "Toggle Advanced Heatmap Analysis" (Umschalten fortgeschrittene Heatmap Analyse). Ein Klick auf diesen Button öffnet eine weitere Zeile mit drei weiterführenden Heatmaps. Diese drei analysieren Zusammenhänge innerhalb der darüber liegend linken unsortierten Heatmap. Links wird untersucht ob Hin- und Rückrunde eines Matches das gleiche Ergebnisse hat (grün gefärbt) oder ob es eine Abweichung gibt (hellgrau bedeutet eine geringe Abweichung bis hin zu schwarz mit (fast) keinem Zusammenhang). Daneben befindet sich die Untersuchung, ob die Rückrunde eines Spiels das genaue Gegenteil ergab als die Hinrunde, die Farbkodierung ist die gleiche wie zuvor. Die rechte Heatmap stellt eine Zusammenfassung der vorherigen beiden Untersuchungen dar, wenn beide vorherigen Bedingungen voll erfüllt sind für ein Feld, dann wird es grün gefärbt, sonst rot. Mithilfe dieser Analysen lässt sich beispielsweise feststellen, ob immer der gleiche Agent gewinnt oder immer der beginnende Agent eines Matches gewinnt.
Die zweite Tabelle von unten beinhaltet Informationen über die Agenten und ihre Spielstärke. Sie ist nach dem WTL Score absteigend sortiert und beinhaltete für jeden Agenten die Bezeichnung im Turnier, den Dateinamen, und Detailinformationen wie viel Spiele gewonnen/verloren/unentschieden gespielt wurden und die einzelnen Scores in WTL, Elo und Glicko2 System.

Die letzte Tabelle unten in diesem Fenster beinhaltet Auswertungen der detaillierten zeitlichen Messungen. Der Nutzer hat hier die Möglichkeit abzulesen, welcher Agent wie schnell oder langsam Züge getätigt hat und wie schnell Matches gespielt wurden. Mit einem Klick auf den Toggle Button können auch erweiterte Informationen zu jedem einzelnen Match angezeigt werden.

4.3.2 TSAgentManager

Die Klasse TSAgentManager ist der Mittelpunkt der Logik des Turniersystems. In dieser Klasse werden alle Agenten verwaltet, Zeit- und Spielergebnismessungen gespeichert, der Turnierablauf gesteuert und am Ende die Messdaten ausgewertet und das Ergebnisfenster erstellt. Für die Initialisierung dieser Klasse benötigt der Konstruktor die Anzahl der Spieler des Spiels. Vor Beginn eines Turniers übergibt die Einstellungsoberfläche die vorhandenen Standardagenten. Alle Standard- und Festplattenagenten werden in eigenen TSAgent Objekten im TSResultStorage gespeichert. Diese Gliederung in einzelne Klassen erfolgt, damit das Ergebnis eines Turniers abgespeichert werden kann. Dafür muss nur die Klasse TSResultStorage serialisiert werden. Wenn der Nutzer in der Einstellungsoberfläche auf die Start-Schaltfläche klickt, werden alle gesetzten Einstellungen an den TSAgentManager übergeben und in Variablen für das Turnier gespeichert. Mithilfe dieser Einstellungen wird dann der Turnierplan innerhalb von TSResultStorage erstellt und sämtliche Variablen und Datenstrukturen für die Messungen vorbereitet. Zuletzt wird in der GBG Arena-Instanz des Spiels der Modus der Hauptschleife von IDLE zu TRNEMNT gewechselt und die Eingabefelder und Schaltflächen des Einstellungsfensters für Eingaben gesperrt. Innerhalb des Turnierbereichs der Hauptschleife beginnt darauf eine weitere Schleife, die Schritt für Schritt die Matches des Turniers durchführt, bis alle Matches und Episoden gespielt und alle Ergebnisse erfasst sind. Bei jedem Durchlauf der Schleife wird dabei geprüft, ob noch eine weitere Episode verfügbar ist. Wenn ja, dann werden die erforderlichen TSAgent und dazugehörigen TSTimeStorage Objekte abgerufen und dann den Wettkampf Methoden singleCompeteBaseTS und competeTS in XArenaFuncs übergeben. In diesen Methoden tragen die Agenten alle Episoden-Kämpfe aus, erfolgen die Zeitmessungen und erfolgt die Rückgabe des Ergebnisses. Das Ergebnis wird in den TSAgent Objekten der spielenden Agenten gespeichert und danach die nächste Runde begonnen. Wenn alle Episoden aller Matches gespielt sind, wird die Auswertung und Visualisierung der Messungen ausgeführt und das Ergebnis automatisch gespeichert, wenn

der Nutzer das eingestellt hat. Am Ende wird das Turnier im TSAgentManager als beendet markiert und die Eingabefelder und Schaltflächen des Einstellungsfensters wieder für Eingaben freigegeben.

4.3.3 TSAgent

Die Objekte der TSAgent Klasse speichern den spielenden Agenten selbst und die Anzahl der Spiele, die gewonnen/verloren oder unentschieden gespielt wurden. Darüber hinaus kann das TSAgent-Objekt Auskunft über den Namen des Agenten geben und ob es sich um einen Agenten aus dem lokalen Speicher handelt. Bei Einzelspielerspielen werden die erzielten Scores erfasst, um am Ende statistische Aussagen zu ermöglichen. Die Objekte der Klasse TSAgent werden in der Klasse TSResultStorage in einer dynamischen ArrayListe verwaltet und sind zum Abspeichern serialisierbar. Nur die PlayAgent Variable, in der das Agentenobjekt der Festplattenagenten gespeichert ist, ist transient und somit nicht Teil der Serialisierung.

4.3.4 TSResultStorage

Die Klasse TSResultStorage beinhaltet alle Messergebnisse, Eigenschaften und Agenten des Turniers. Die Agenten befinden sich in einer dynamischen ArrayListe. Der Spielplan, die Spielergebnisse und die Zeitmessungen befinden sich jeweils in mehrdimensionalen Arrays und erfassen alle Messungen des Turniers. Der String *startDate* beinhaltet den Startzeitpunkt des Turniers. Die Klasse TSResultStorage ist auch serialisierbar, um die Ergebnisse eines Turniers vollständig abspeichern zu können.

4.3.5 TSTimeStorage

Die Klasse TSTimeStorage verwaltet die detaillierten Zeitmessungen eines jeden Zuges eines Agenten über alle Episoden eines Matches. Anhand der Größe des Spielplans erstellt die Klasse TSAgentManager im TSResultStorage für jeden Agenten jedes Matches eine Instanz der TSTimeStorage Klasse. Für die Zeitmessungen wurden drei ArrayLists vom Typ Long implementiert, *measuredTimesInNS* speichert die Messungen aller Züge eines Agenten des Matches. Zusätzlich wird in *tmpRoundTimesInNS* nur die Messungen der aktuellen Episode erfasst. Nach beenden einer Episode muss die Methode *roundFinished()* aufgerufen werden, wodurch alle Werte

der *tmpRoundTimesInNS* zusammengefasst werden und als Rundenzeit in *roundTimesInNS* gespeichert wird. Danach wird *tmpRoundTimesInNS* für die nächste Episode geleert. Da alle Messungen im Turniersystem in der Einheit Nanosekunden erfolgen, bietet diese Klasse Methoden zur Umrechnung der Nanosekundenmesswerte in Millisekunden. Die Darstellung von Messwerten in Millisekunden erleichtert das Verständnis für den Nutzer. Für die Erstellungen der statistischen Auswertung bietet diese Klasse Methoden zur Bestimmung der kürzesten/längsten Zugdauer und die durchschnittliche/Median Zeit für die Episoden und das ganze Match. Auch diese Klasse ist serialisierbar und ist Teil der abspeicherbaren Turnierergebnisse. Dank der detaillierten Erfassung der Messungen können vielseitige Auswertungen vorgenommen werden.

4.3.6 Hilfsklassen

Im Java Package TournamentSystem.tools befinden sich die Hilfsklassen TSDiskAgentDataTransfer, TSGameDataTransfer und TSHeatmapDataTransfer. Diese Klassen wurde für die kompakte Übergabe von Daten zwischen Methoden konzipiert. TSDiskAgentDataTransfer wird verwendet, um beliebig viele Agenten von der Festplatte in das Turniersystem zu importieren. Die Klasse wurde implementiert, um die Agenten selbst als PlayAgent Objekte und die zugehörigen Dateinamen als Strings übergeben zu können. TSGameDataTransfer wurde implementiert, um die nötigen Daten eines Matches aus dem TSResultStorage an die Wettkampfmethoden zu übergeben. Mithilfe dieser Klasse können die Objekte für Agenten, Zeitmessung und Startzustand zusammen mit einzelnen Eigenschaften zusammen übergeben werden. Sämtliche Daten innerhalb dieser Klasse werden in Arrays gespeichert, somit kann sie in Turnieren mit beliebig vielen gegeneinander spielenden Agenten eingesetzt werden. TSHeatmapDataTransfer wird in der Erstellung der Visualisierung eingesetzt, um sämtliche Heatmaps in einem an die Ergebnis-GUI-Klasse zu übergeben. Durch den Einsatz dieser Hilfsklasse konnte die Signatur der Heatmap-Setter-Methode deutlich vereinfacht werden. Darüber hinaus besitzt die Klasse TSAgentManager drei innere Hilfsklassen TSHMDataStorage, TSHM2DataStorage und TSSimpleTimeTableHelper. Alle drei Klassen werden innerhalb der statistischen Auswertung für die graphische Ausgabe der Ergebnisse verwendet. TSHMDataStorage und TSHM2DataStorage werden bei dem Sortieren der Datenreihen vor dem Erstellen der Heatmaps eingesetzt. TSSimpleTimeTableHelper kommt bei der Datenauswertung für die vereinfachte Tabelle zur Zeitmessung zum Einsatz.

4.4 Bibliotheken

In den folgenden Kapiteln 4.4.1 bis 4.4.3 werden die drei Bibliotheken vorgestellt, die bei der Implementierung des Turniersystems verwendet wurden. Die JHeatChart und Glicko2 Bibliotheken wurden dabei geringfügig angepasst. Die Elo Bibliothek ist eine Kombination verschiedener Bibliotheken, da zum Zeitpunkt der Entwicklung keine einzelne Java Bibliothek mit zufriedenstellendem Funktionsumfang gab.

4.4.1 JHeatChart

Die verwendete JHeatChart Bibliothek [Cas10] stammt vom Autor Tom Castle aus dem Jahr 2010. Diese Bibliothek erlaubt es, auf Basis von mehrdimensionalen Datensätzen, HeatMaps zu generieren. Dabei wird standardmäßig die Skala der Datenpunkte in Grautöne zwischen Schwarz und Weiß umgesetzt. Für den Einsatz im Turniersystem wurde die Bibliothek dahingehend angepasst, dass ein Turniermodus implementiert wurde, der über einen neuen Konstruktor aktiviert werden kann. In diesem Turniermodus werden keine Datenpunkte kleiner als 0 interpretiert, sondern Werte kleiner 0 als Anweisung für eine gesonderte farbliche Darstellung umgesetzt. Auf diese Weise können spezielle Felder in der Visualisierung mit definierten Farben gesetzt werden. Diese speziellen Felder umfassen dabei die Hauptdiagonale (dort finden keine Matches statt, weil Agenten nicht gegen sich selbst spielen) und Felder der erweiterten Datenanalyse, die in besonderen Fällen gesondert eingefärbt werden. Abbildung 8 veranschaulicht diese Funktion anhand der erweiterten Heatmapanalysen, in denen reguläre Daten und verschiedene spezielle Felder vorkommen.

4.4.2 Elo

Die im Turniersystem verwendete Bibliothek zur Berechnung der Elo Ratings basiert in weiten Teilen auf der java-elo Bibliothek [Nos13] von Mariusz Nosiński aus dem Jahr 2013. Die Funktionalitäten waren allerdings auf viele Klassen verteilt und der Einsatz erschien umständlich. Mit der Struktur der verwendeten Glicko2 Bibliothek als Beispiel wurde der Programmcode dann auf eine Berechnungs- und eine abstrakte Spielerklasse reduziert. Mithilfe der Spielerklasse wurden zwei verschiedene Parameterimplementierungen des Elo Ratings nach USCF und FIDE implementiert. Die Spielerklasse beinhaltet das Ranking des Agenten und implementiert das Interface Serializable, um dieses Rating, als Bestandteil des TSResultStorage, in einer Datei abspeichern zu können.

4.4.3 Glicko2

Die Bibliothek Glicko2 [Goo13] stammt vom Autor Jeremy Gooch aus dem Jahr 2013. Diese Bibliothek ermöglicht die einfache Nutzung des Glicko2 Algorithmus und orientiert sich in seiner Implementierung am Paper des Algorithmus [Gli12]. Für den Einsatz im Turniersystem wurden nur die Namen der Klassen angepasst, indem Glicko2 vor die ursprünglichen Namen eingefügt wurde. Weiterhin wurde in den Klassen Rating und RatingCalculator das Interface Serializable implementiert, um das Glicko2 Rating, als Bestandteil des TSResultStorage, in einer Datei abspeichern zu können.

4.5 Integration in das bestehende GBG Framework

Im folgenden Kapitel wird dargestellt, welche Klassen des GBG Frameworks modifiziert werden mussten, um das Turniersystem zu integrieren.

4.5.1 XArenaMenu - GUI

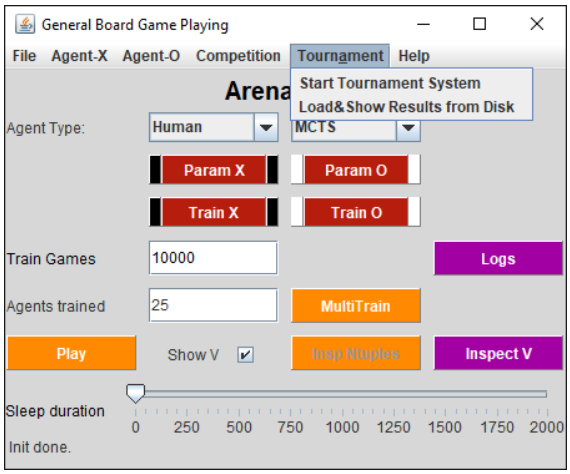


Abbildung 9: Turniermenüpunkt in der GBG GUI

Die Klasse XArenaMenu im Package games ist für die Generierung der Menüs im GBG Framework zuständig. In dieser Klasse wurde die neue Methode generateTournamentMenu() erstellt, um das Turniersystem in die GBG Menüs zu integrieren. Mithilfe der Methode werden zwei Menüpunkte erstellt, der erste öffnet die Einstellungsoberfläche des Turniersystems. Der zweite Punkt öffnet einen Dialog zum Öffnen eines abgespeicherten Turnierergebnisses, um dessen Ergebnisfenster wieder zu öffnen (siehe Abbildung 9).

4.5.2 Arena - Hauptschleife

Innerhalb der Klasse Arena im Package games befindet sich in der Methode run() eine While-Endlosschleife. Diese Schleife besteht aus einem Switch-Case, welches, je nach TaskState der Arena, die verschiedenen Funktionen des GBG ausführt, oder sich im IDLE Modus im Standby befindet. Für das Turniersystem wurde hier ein neuer TaskState definiert mit dem Namen TRNEMNT. Wenn dieser Zustand gesetzt wird, ermöglicht diese Endlosschleife das Durchlaufen des Turniers. Zu Beginn eines Turniers werden zuerst die einzelnen Datenstrukturen zur Zeitmessung, Ergebniserfassung und dem Spielplan initiiert und die Oberflächenelemente der Einstellungsoberfläche gesperrt. Dadurch werden Eingaben durch den Nutzer verhindert, während ein Turnier läuft. Danach wird die Fortschrittsanzeige initiiert und dann das Turnier mittels einer weiteren Schleife Spiel für Spiel durchgeführt, wie bereits in Kapitel 4.3.2 vorgestellt.

4.5.3 XArenaFuncs - Wettkampffunktionen

In der Klasse XArenaFuncs befinden sich die Methoden des GBG Frameworks, um zwei Agenten gegeneinander spielen zu lassen und die Spielergebnisse zurück zu geben. Die dafür benötigten Methoden competeBase(...) und compete(...) hätten stark verändert werden müssen, um auch mit dem Turniersystem verwendet werden zu können. Deshalb wurden beide Methoden unter den Namen singleCompeteBaseTS(...) und competeTS(...) dupliziert. In diesen neuen Methoden konnten alle nötigen Änderungen und neuen Datentypen des Turniersystems implementiert werden. Aus der Arena Klasse wird zuerst die Methode singleCompeteBaseTS(...) mit allen nötigen Daten der Agenten und Messungen aufgerufen und darin für den Wettkampf vorbereitet. In dieser Methode folgt dann ein Aufruf der competeTS(...) Methode, in welcher die beiden Agenten gegeneinander spielen und die Zeit Messungen stattfinden, bis ein Spielergebnis erzielt wurde. Wenn der Nutzer zufällige Startspielzüge eingestellt hat, werden diese in dieser Methode an die Agenten übergeben. Beide Methoden dieser Klasse werden für jede Episode des Matches neu aufgerufen, um die Agenten jedes Mal erneut aufzurufen und neu zu messen. Das Ergebnis, welcher Agent gewonnen hat, wird an die Arena zurückgegeben und dort in den Messergebissen im TSResultStorage und den TSAgent-Objekten der Agenten gespeichert.

4.5.4 AgentIO - Speichern und Laden

Im GBG Framwork gab es im Package agentIO die Klasse LoadSaveGBG mit der Methode saveGBGAgent(PlayAgent pa). Mithilfe dieser Methode konnten Agenten über das Menü in der GUI auf die Festplatte abgespeichert werden. Dies konnte eingesetzt werden, um beispielsweise speziell trainierte/parameterisierte Agenten abzuspeichern, und so mit denselben Eigenschaften immer wieder verwenden zu können. Mit dem Turniersystem sollte auch eine Möglichkeit geschaffen werden, um Turnierergebnisse lokal abspeichern zu können. Da eine eigene Methode zum Abspeichern der Turnierergebnisse fast exakt denselben Javacode wie saveGBG-Agent enthalten hätte, wurde diese nicht dupliziert. Stattdessen wurde saveGBG-Agent(...) in saveGBGHelper(...) umbenannt und mit weiteren Übergabevariablen versehen. Die Methode wurde dann so erweitert, dass anhand der übergebenen Parameter zu erkennen war, ob ein Agent oder ein Turnierergebnis abgespeichert werden soll und welche dafür spezifischen Einstellungen gesetzt werden mussten. Zusätzlich wurde wieder eine saveGBGAgent(...) Methode implementiert, die nur ein PlayAgent Objekt übernimmt und dann die saveGBGHelper Methode mit den entsprechenden Parametern aufruft. Zum Speichern eines Turnierergebnisses wurden zwei Methoden mit dem Namen saveTSResult(...) implementiert. Die erste übernimmt nur ein TSResultStorage Objekt und ruft dann die saveGBGHelper Methode auf mit Parametern, die den Nutzer dann, durch einen sich öffnenden Dialog, selbst bestimmen lassen, wo und unter welchem Namen abgespeichert werden soll. Die zweite Methode nimmt zusätzlich zum TSResultStorage Objekt einen boolean Wahrheitswert entgegen. Mit diesem boolean kann gesetzt werden, ob der Nutzer Speicherpfad und Dateinamen eingeben darf oder ob der Standardordner TSR im Spielverzeichnis des GBG mit einem automatisch auf Datum und Uhrzeit generierten Dateinamen verwendet werden soll. Mithilfe dieser Lösung konnten weitreichende Javacode-Duplikate vermieden werden, die entstanden wären, wenn saveTSResult als Kopie der saveGBGAgent entstanden wäre.
Zum Laden eines Turnierergebnisses wurde eine Kopie der loadGBGAgent Methode mit dem Namen loadGBGTSResult(...) verwendet, da ähnlich wie in der XArenaFuncs, zu weitreichende Änderungen hätten vorgenommen werden müssen. Die Methode nimmt als Wert den Dateipfad des abgespeicherten Turnierergebnisses entgegen und gibt dann ein Objekt der Klasse TSResultStorage zurück. Diese Methode wird über den zweiten Menüpunkt Turniersystems in der GBG GUI ausgelöst. Das TSResultStorage Objekt wird dann an eine Instanz des TSAgentManagers

übergeben, in dem dann die Messergebnisse neu analysiert und verarbeitet werden, wonach das Ergebnisfenster erstellt und geöffnet werden kann.
Weiterhin wurde in der Klasse LoadSaveGBG die Methode loadMultipleGBG-Agent() implementiert. Diese kann über die Turniereinstellungsoberfläche aufgerufen werden, um einzelne oder mehrere Agenten aus dem lokalen Speicher in das Turniersystem zu laden. Der Rückgabewert ist hier ein Objekt der, zuvor in Kapitel 4.3.6 beschriebenen, Hilfsklasse TSDiskAgentDataTransfer. Diese beinhaltet die geladenen Agenten und die zugehörigen Dateinamen, welche im Turniersystem als Name des Agenten verwendet werden.

5 Herausforderungen der Implementierung

Dieses Kapitel befasst sich mit den besonderen Herausforderungen in der Implementierung des Tuniersystems.

5.1 Zeitmessung in Java

Die verwendete Programmiersprache Java bietet verschiedene Möglichkeiten, um Zeit zu erfassen und messen. Die bekanntesten Methoden sind dabei die *System .currentTimeMillis()*[3] und *System.nanoTime()*[4]. Diese liefern beide Zeitstempel in Milli- und Nanosekunden Auflösung. Problematisch kann hier die Vergleichbarkeit von Messwerten auf demselben oder verschiedenen Computersystemen sein. Verschieden hohe Auslastungen auf einem Computersystem, verursacht durch andere Programme, mindern die für Java zur Verfügung stehende Rechenkraft. Dadurch kann es passieren, dass ein Programm mit derselben Aufgabe wiederholt läuft, aber die ermittelte verstrichene Zeit abweichend ausfällt. Ein weiterer Ansatz ist die Klasse *ThreadMXBean*[5], welche seit Java 1.6 verfügbar ist. Diese bietet mit der Methode *getCurrentThreadCpuTime()* die Möglichkeit, die aktuelle CPU-Zeit des Threads in Nanosekunden anzugeben. Mit Hilfe dieser Methode hätte es möglich sein können, nur die verstrichene Zeit des Arena-Threads zu messen. In Java Version 1.8 wurde die neue Zeitklasse *Instant*[6] eingeführt. Auch mit ihr sind Zeitmessungen möglich.
Im Spiel Tic Tac Toe wurden dazu Messungen in einem Turnier durchgeführt, um die Eignung der verschiedenen Methoden und Klassen zu untersuchen. Die Wahl fiel auf Tic Tac Toe, da Turniere in diesem Spiel sehr kurz dauern und einzelne Züge sehr schnell ausgeführt werden. Eine Zeitmessung mit hoher Auflösung wird dafür benötigt. Es wurden Turniere mit zwei Standardagenten durchgeführt und dabei die Dauer jeden Zuges mit den genannten Klassen und Methoden gemessen. Diese Messungen haben ergeben, dass oft keine brauchbaren Messergebnisse erfasst werden konnten. Die Zeitmessungen ergaben überwiegend den Wert 0, was nahelegt, dass die meisten Methoden nur eine Messgenauigkeit im Millisekunden Bereich zur Verfügung stellen. Für viele Messungen ist das viel zu langsam. Nur *System.nanoTime()* lieferte für jeden Zug ein Messergebnis größer Null. Zu

[3] https://docs.oracle.com/javase/7/docs/api/java/lang/System.html#currentTimeMillis()
[4] https://docs.oracle.com/javase/7/docs/api/java/lang/System.html#nanoTime()
[5] https://docs.oracle.com/javase/7/docs/api/java/lang/management/ThreadMXBean.html
[6] https://docs.oracle.com/javase/8/docs/api/java/time/Instant.html

den Messungen mit *System.nanoTime()* ist wichtig anzumerken, dass die Methode zwar einen Wert in Nanosekunden Auflösung zurückgibt, aber keine Messgenauigkeit von einzelnen Nanosekunden besitzt. Die Dokumentation der Methode weist darauf explizit hin. Aleksey Shipilëv hat dazu detaillierte Untersuchungen durchgeführt [Shi14]. Darin zeigt er, dass die Genauigkeit vom verwendeten Betriebssystem und auch der Hardware des Computers abhängig ist. Im schlechtesten Fall besitzt *System.nanoTime()* nur eine Genauigkeit von Millisekunden. Eine Untersuchung der Messgenauigkeit von *System.nanoTime()* wird in Kapitel 6.2.5 in Messreihe M5 durchgeführt.

5.2 Umgang mit Determinismus

Wie in Kapitel 2.2 dargelegt, besitzen deterministische Spiele in ihrem Ablauf keinen Zufall. Ein Würfelwurf kann einen solchen Zufall darstellen, da das Ergebnis nicht vorhersehbar ist. Der Verlauf des Spieles ist nur von den Entscheidungen der spielenden Agenten und den Spielregeln selbst abhängig. Ein spielender Agent berechnet seine Züge nach verschiedenen Algorithmen, manche dieser Algorithmen können auch mittels Training verbessert werden. Neben dem Spiel können auch die Agenten selbst deterministisch sein, und ihre Züge immer nach dem selben Schema berechnen. Dann verhalten sie sich in der gleichen Spielsituation immer exakt gleich, da sie im Gegensatz zum Menschen keine Intuition oder Kreativität besitzen. Das kann zu einem Problem führen, wenn zwei deterministische Agenten in einem deterministischen Spiel gegeneinander spielen. Dann werden beide immer dieselben Züge tätigen, den einzigen Unterschied im Spielverlauf macht dann, wer von beiden beginnt zu spielen. Ein Wiederholen der Partie wird also keine neuen Züge oder "Taktiken" offenbaren. Eine Lösung könnte sein, die beiden Agenten eine Partie nicht mit einem leeren Spielbrett beginnen zu lassen. Es sollen also eine Anzahl an n Startzügen vorgegeben werden, um die Agenten damit weiter spielen zu lassen (deshalb Anforderung A.3 an die Software). Auf diese Art kann von außen Varianz erzwungen werden, um das Können der Agenten in verschiedenen Spielsituationen zu überprüfen. Ein anderer Ansatz könnte sein, sicherzustellen, dass in einem Match immer mindestens ein Agent ein nicht-deterministisches Verhalten besitzt.

5.3 Visualisierung der Ergebnisse

Die Herausforderung bei der Visualisierung der Messergebnisse ist es, den Nutzer nicht zu überfordern und zu überladen (siehe Anforderung A.1 in Kapitel 4.1). Aus der Masse an erfassten Daten lassen sich viele Tabellen, Zahlenreihen und Grafiken generieren, das würde aber keinen nennenswerten Mehrwert liefern und den Nutzer überfordern. Relevante Informationen würden in der Flut der Daten untergehen. Deshalb ist es sehr wichtig zu identifizieren, welche Daten und Ergebnisse aus der Analyse der Daten relevant sind. Deshalb finden sich im Ergebnisfenster des Turniersystems nur sechs Plots und vier Tabellen. Drei Plots und drei Tabellen enthalten detailliertere Informationen und sind minimiert, wenn sich das Fenster öffnet. Der Nutzer kann sie bei Bedarf ausklappen. Fünf der Plots sind Heatmaps, um Match- und Analyseergebnisse farblich kodiert darzustellen. Der sechste Plot ist ein Scatterplot, um die Spielstärke der Agenten im Turnier gegen ihre Geschwindigkeit auftragen. Die einzige sofort sichtbare Tabelle befindet sich unterhalb der Plots und beinhaltet die wichtigsten Informationen zu den Agenten. Hier lässt sich deren Typ, Spielstärke und Alias im Turnier ablesen. Der Nutzer kann mithilfe dieser Tabelle auf einen Blick erkennen, welcher Agent wie gut abgeschnitten hat. Die weiteren Tabellen und Grafiken ermöglichen darüber hinaus detailliertere Aussagen über das Verhalten der Agenten im Turnier. Als übersichtliche graphische Lösung fiel die Wahl auf Heatmaps, mit ihrer Hilfe können drei Dimensionen von Daten als farbige Matrix dargestellt werden (welche Agenten gespielt haben und eine Aussage über das Match). Dabei erhalten die darzustellenden Ergebniswerte verschiedene Grauwerte. Nicht stattgefundene Matches erhalten gesonderte Farben. Aus den Heatmaps können keine detaillierten Werte abgelesen werden, aber sie ermöglichen einen schnellen Überblick. Die detaillierten Informationen zu den ersten beiden Heatmaps finden sich in den oberen beiden Tabellen des Ergebnisfensters. Der Scatterplot ermöglicht Aussagen wie schnell und gut ein Agent gespielt hat. Der optimale Agent wäre dabei sehr gut und sehr schnell (oben links im Plot). Umso langsamer ein Agent ist, desto weiter rechts befindet er sich und umso besser er spielt, umso weiter oberhalb der horizontalen Achse. Zusätzliche detaillierte Auswertungen der Zeitmessung finden sich in der untersten Tabelle im Fenster. Diese besitzt eine vereinfachte und eine detaillierte Darstellung. Darin wird pro Match und Agent zusammengefasst, wie schnell oder langsam Züge berechnet und Episoden gespielt wurden.

5.4 Zeitliche Dauer der Turniere

Ein weiterer Faktor ist die für ein Turnier benötigte Zeit. Bei einem Rundenturnier mit vielen Spielern und einem komplexen Spiel kann es sich schnell um viele Minuten bis Stunden handeln. Lösungen können organisatorische Lösungen wie die Verwendung eines alternativen Turniersystems sein oder eine technische, wie der parallele Ablauf von Spielen auf mehreren Threads. Ein paralleler Ablauf von Spielen würde allerdings die verfügbare Rechenkapazität pro Thread reduzieren und könnte wieder die Messergebnisse negativ beeinflussen. Messungen mit und ohne parallel betriebenen anderen Programmen wie Browsern wurden durchgeführt. Diese zeigten, dass die Nutzung andere Programme zeitgleich mit einem Turnier die Zeitmessungen negativ beeinflussen. Ein ähnlicher negativer Einfluss wird für Turniere auf parallelen Threads erwartet.

5.5 Auslastung des Arbeitsspeichers und Nutzung des Java Garbage Collectors

Während der Implementierung des Turniersystems wurde festgestellt, dass während eines laufenden Turniers eine sehr hohe Arbeitsspeicherauslastung durch Java beobachtet werden konnte. Der Java Prozess nahm fast 4GB in Anspruch unter Windows 10. Die hohe Auslastung verursachte keine merklichen Veränderungen des Turnierablaufs oder der Messungen. Da dies keine zu erwartende Beobachtung ist wurde versucht die Ursache zu identifizieren. Scheinbar wurden im Verlauf des Turniers sehr viele Speicher belegt, der nach Ende einer Episode nicht richtig freigegeben wurde. Eigentlich sollte der Java Garbage Collector (GC) automatisch in regelmäßigen Intervallen automatisch von Java ausgeführt werden, um nicht mehr benötigenden Speicher freizugeben. Als Lösung wurde nach Abschluss jeder Turnierepisode der GC manuell gestartet mit dem Befehl *System.gc()*. Dies führte zu einer signifikanten Minderung des Arbeitsspeicherverbrauchs auf nur noch rund 1GB. Zu diesem Zeitpunkt wurde allerdings nicht bemerkt, dass diese Lösung ein großes Problem darstellt. Bei Turnieren mit vielen Episoden und kurzen Spielzeiten wurde der GC in kurzer Zeit oft aufgerufen. Jeder Aufruf benötigt etwas Zeit, in welcher der Programmcode angehalten wird, was bei kleinen Turnieren nicht auffällt. Bei großen Turnieren mit großer Anzahl an Episoden fiel auf, dass die Summe der Matchzeiten sehr viel kleiner war als die gesamte Dauer des Turniers, z.B. ein 40 Sekunden Turnier dauerte hier vier Minuten. Es entstand der Verdacht,

dass diese Differenz durch den GC verursacht wurde, als ein Turnier mit dem Fehler *java.lang.OutOfMemoryError: GC overhead limit exceeded* abgebrochen wurde. Dieser Fehler besagt, dass der GC 98% der CPU Zeit in Anspruch nimmt und das Programm selbst nur noch 2%. Nach Entfernen des manuellen GC Aufrufs lag die Arbeitsspeicherauslastung bei großen Turnieren wieder bei rund 4GB, aber die Dauer der Turniere deckt sich so wieder mit der Summe der einzelnen Messungen. Da durch den hohen Verbrauch von Arbeitsspeicher keine negativen Folgen für Turnier und Messung beobachtet werden konnten und sich die vermeintliche Lösung als sehr problematisch herausstellte, wird die hohe Auslastung des Arbeitsspeichers vorerst in Kauf genommen.

6 Messungen

In den vorherigen Kapiteln wurde vorgestellt, wie das Turniersystem implementiert wurde und wie es funktioniert. In diesem Kapitel soll nun darauf eingegangen werden, wie die eingangs gestellten Forschungsfragen mittels Messungen untersucht werden können. Die Messreihen, Durchführung der Messung und Auswertung werden in diesem Kapitel vorgestellt. Im anschließenden Kapitel 7 (Evaluation/Diskussion) findet eine Diskussion der Messergebnisse statt.

6.1 Methodik

In Kapitel 1 (Einleitung) wurden verschiedene Forschungsfragen formuliert. Davon lassen sich fünf mithilfe von Messreihen im Turniersystem untersuchen und beantworten. Für diese Messungen werden nichttriviale Spiele und verschiedene Agenten benötigt. Als Spiel kommen dafür Hex [Gal17] und Vier Gewinnt (basierend auf [Thi12] und [BTKK16]) in Frage. Tic Tac Toe ist aufgrund der wenigen möglichen Züge und einfachen Regeln zu trivial und in 2048 können nicht alle Forschungsfragen untersucht werden, da es nur einen Spieler ermöglicht und keine Paare. Das Spiel Hex kann in verschiedenen Spielfeldgrößen verwendet werden. In den Messungen dieser Arbeit wird die Standardgröße 4x4 verwendet, da die verfügbare Zeit nicht für weitere größere Felder ausreichte. Für die Messungen wurden Agenten im GBG Framework mit verschiedenen Parametern erstellt und abgespeichert. Durch verschiedene Parameterkombinationen wird sichergestellt, dass sich nicht alle Agenten eines Typs gleich verhalten. Das Abspeichern der parametrisierten Agenten in den lokalen Speicher stellt sicher, dass die gleichen Agenten mit demselben Können wiederholt verwendet werden können. Dadurch können Messungen mit den gleichen Rahmenbedingungen wiederholt durchgeführt werden. Die Messergebnisse der einzelnen Messungen pro Reihe werden auch im lokalen Speicher gesichert, um die Ergebnisse im Nachhinein wieder abrufen zu können.

Agenten

Das GBG Framework stellt insgesamt zehn verschiedene Typen von Agenten zur Verfügung (siehe Tabelle 2). In den Turnieren lassen sich allerdings nur sieben davon nutzen. Das liegt zum einen daran, dass der Human Agent den Nutzer spielen lässt und keinen eigenen Agenten darstellt. Zum anderen können nicht alle Agenten in deterministischen und auch nicht deterministischen Spielen eingesetzt werden.

Deshalb werden im GBG und auch GBG-Turniersystem, je nach Spieltyp, neben dem allgemeinen Agenten nur die kompatiblen Agenten angezeigt. Zusätzlich gibt es zwei trainierbare Agenten, die nur verwendet werden können, wenn sie trainiert und abgespeichert wurden. Deshalb sind sie in der Liste der Standardagenten im Turniersystem nicht gelistet.

Agent	Turnier geeignet	Trainierbar	Geeignet für det. Spiele	Geeignet für nicht det. Spiele
Random	JA	NEIN	JA	JA
Minimax	JA	NEIN	JA	JA
Max-N	JA	NEIN	JA	NEIN
ExpectimaxN	JA	NEIN	NEIN	JA
MC-N	JA	NEIN	JA	JA
MCTS	JA	NEIN	JA	NEIN
MCTSExpectimax	JA	NEIN	NEIN	JA
Human	NEIN	NEIN	JA	JA
TD-NTuple-2	JA	JA	JA	JA
TDS	(JA)	(JA)	JA	JA

Tabelle 2: Übersicht der Agenten im GBG Framework (det. = deterministisch)

Für die Messreihen in den Spielen Hex und Vier Gewinnt (beide deterministisch) wurden, von den zur Verfügung stehenden sieben Agententypen, nur vier(Hex) bzw. drei(Vier Gewinnt) verwendet. Das liegt daran, dass der Max-N Agent eine allgemeine Version des Minimax darstellt (die Verwendung des Minimax also keinen Mehrwert darstellt), der TDS Agent zum Zeitpunkt der Messungen aus technischen Gründen[7] seitens des GBG Frameworks nicht verwendet werden kann und der Random Agent nur zufällige Züge tätigt. Für das Spiel Hex wurden 26 Agenten des Typs Max-N, MCN, MCTS, TD-NTuple-2 mit verschiedenen Parametrisierungen erstellt. Für Vier Gewinnt wurden 36 Agenten der Typen MCN, MCTS, TD-NTuple-2 erstellt, die Anzahl je Typ war abhängig von der Menge der einstellbaren Parameter. Max-N wurde nicht verwendet, da er selbst bei kleinen Turnieren eine mehrstündige Laufzeit verursachte. Dies war in Anbetracht der großen Anzahl an Turnieren und Episoden in dieser Arbeit nicht praktikabel. Alle so erstellten Agenten wurden zusammen mit den Agenten, die schon im Github Repository des GBG vorhanden waren (TD-NTuple-2 und TCL (Temporal Coherence Learning; siehe [BTKK16])),

[7] Die Arbeit [Gal17] beschreibt die Probleme der Bestimmung von Features des TDS Agenten in Hex im GBG Framework. Dem Agenten war es ab Spielfeldern der Größe 4x4 nicht möglich die Spielfunktion erfolgreich zu erlernen. Für Felder kleiner 4x4 konnten Features erfolgreich gefunden werden.

in drei Turnieren getestet. Die Turniere wurden alle mit zehn Episoden pro Match und 0/1/2 zufälligen Startzügen durchgeführt. Zur Auswertung wurden die WTL Scores der Agenten über die drei Turniere gemittelt und die vier besten jedes Typs ausgewählt. Dies resultiert im Spiel Hex (vier Typen) in 16 ausgewählten Agenten und einem 17ten Random Agenten und im Spiel Vier Gewinnt (drei Typen) in 12 ausgewählten Agenten und einem 13ten Random Agenten.
Die ausgewählten Agenten des Spiels Hex finden sich im Anhang in Tabelle 10 und die des Spiels Vier Gewinnt in Tabelle 11. In diesen Tabellen finden sich die Agentennamen in den Messreihen, Dateinamen, mittlerer WTL aus den drei Auswahlturnieren und die Standardabweichung (SD) der WTL. Die Reihenfolge der Agenten in der Tabelle und Benennung erfolgt nicht sortiert nach der Stärke der Agenten, sondern nach der Reihenfolge, in der das Turniersystem die Agenten aus dem Speicher lädt. Aus den Dateinamen lässt sich ablesen, um welchen Typ Agenten es sich handelt und welche Parameter genutzt wurden. Wenn ein verfügbarer Parameter nicht im Dateinamen genannt wird besitzt er den Standardwert des GBG Frameworks.

Testcomputer

Die Messreihen werden auf einem Computer mit dem Modell *Microsoft Surface Book 2 15"* durchgeführt. Dieser verfügt über einen Intel Core i7-8650U Prozessor und 16GB Arbeitsspeicher. Das Betriebssystem ist Windows 10 Pro in 64 Bit, die Java Version ist 1.8.0_181 64 Bit. Die Messungen werden in der Entwicklungsumgebung IntelliJ IDEA Ultimate 2018.2.4 64 Bit durchgeführt.

Ablauf der Messungen

Zu Beginn jedes Turniers werden die Agenten in das Turniersystem geladen und die nötigen Einstellungen gesetzt (Anzahl der Episoden pro Match, Anzahl der zufälligen Startzustände und Aktivieren des automatischen Speicherns der Ergebnisse). Danach wird das Turnier gestartet. Nach Beendigung wird der generische Name der abgespeicherten Turnierergebnisdatei durch einen aussagekräftigen ersetzt. Wenn eine weitere Messung folgen soll, müssen alle Agenten aus dem Turniersystem entfernt werden, da bei Testmessungen beobachtet wurde, dass Messergebnisse sonst nicht konstant sind. Dieser Ablauf wird wiederholt, bis alle Messungen durchgeführt wurden. Zur Bestimmung der Rankings in den Messreihen wird der Glicko Score der einzelnen Agenten über die Turniere genutzt. Dazu werden die Glicko

Rang	Agent	WTL	FIDE Elo	USCF Elo	Glicko2
1	Agent #4	210.0	1226	1194	1661,17
2	Agent #0	204.0	1111	1109	1641,83
3	Agent #15	196.0	1717	1895	1616,04
4	Agent #5	191.0	1219	1205	1599,92
5	Agent #9	191.0	1420	1423	1599,92
6	Agent #8	188.0	1345	1331	1590,25
7	Agent #16	188.0	1795	2073	1590,25
8	Agent #1	186.0	1159	1165	1583,81
9	Agent #10	181.0	1478	1549	1567,69
10	Agent #7	179.0	1296	1281	1561,24
11	Agent #6	172.0	1215	1186	1538,68
12	Agent #11	169.0	1443	1504	1529,01
13	Agent #12	166.0	1497	1611	1519,34
14	Agent #13	161.0	1528	1636	1503,22
15	Agent #3	91.0	1033	988	1277,59
16	Agent #2	45.0	1005	1033	1129,31
17	Agent #14	2.0	718	599	990,71

Tabelle 3: Beispiel Ergebnisse der Wertungssysteme WTL, Elo und Glicko2 nach einem Doppelrundenturnier mit 10 Episoden im Spiel Hex

Scores der Wiederholungen eines Turniers gemittelt. Die WTL, Elo und Glicko Rankings der Agenten nach einem Turnier im Spiel Hex mit 10 Episoden findet sich in Tabelle 3. Dieses Beispiel zeigt, dass die WTL und Glicko Rankings in der Reihenfolge gleich sind. Die Rangfolgen der beiden implementierten Elo Varianten unterschieden sich dabei in einem Großteil der Platzierungen im Vergleich zur WTL. Dies zeigt die Unterschiede in den Algorithmen von Elo und Glicko. Das Elo eines Agenten wird nach jeder Episode aktualisiert. Wie stark ein Agent sein Elo bei einem Sieg erhöhen kann hängt auch von dem Elo des Gegners ab. Wenn dieser Gegner in diesem Turnier noch zu wenig Spiele absolviert hat, stellt sein Elo noch nicht seine Stärke dar, obwohl er stark sein kann. Wäre das Elo des Gegners schon höher gewesen, hätte der ursprüngliche Agent bei einem Sieg mehr Punkte erhalten. Der Glicko Algorithmus besitzt dieses Problem nicht, da er die Wertung eines Agenten nur nach gewissen Perioden aktualisiert. Innerhalb dieser Periode werden Wettkampfergebnisse nur gesammelt. Das Turniersystem ist so implementiert, dass das ganze Turnier eine Wertungsperiode ist, die Glicko2 Werte werden also erst am Ende des Turniers basierend auf den Ergebnissen aller Matches bestimmt. Mit dieser Erkenntnis kann die eingangs gestellte Forschungsfrage F6, was eine gute

Ranking-Kennzahl für einen Agenten ist, beantwortet werden. Der WTL Score ist eine einfache, aber nicht sehr aussagekräftige Lösung und die Elo-Zahl liefert keine zuverlässigen Rankings in den beobachteten Turnieren. Einzig das Glicko2 Wertungssystem ist eine geeignete Lösung und liefert Ratings, die die Spielstärke der Agenten zuverlässig abbilden.

6.2 Messreihen

Zur Untersuchung der Forschungsfragen sollen die Messreihen M1 - M5 durchgeführt werden.

6.2.1 M1 - Kommutativität

Mithilfe der Messreihe M1 soll die Forschungsfrage F1 aus Kapitel 1 untersucht werden. Kommutativität in den Ergebnissen erlauben Rückschlüsse auf die Agenten und ihre Eigenschaften im Turnier.

Durchführung

Es spielen die 17(Hex)/13(Vier Gewinnt) zuvor bestimmten Agenten in einem Doppel Rundenturnier mit zehn Episoden pro Match gegeneinander. Dieses Turnier wird zehn Mal wiederholt, um die Messgenauigkeit bei zufälligen Schwankungen zu verbessern. Die Ergebnisse jedes Turniers werden automatisch abgespeichert.

Auswertung

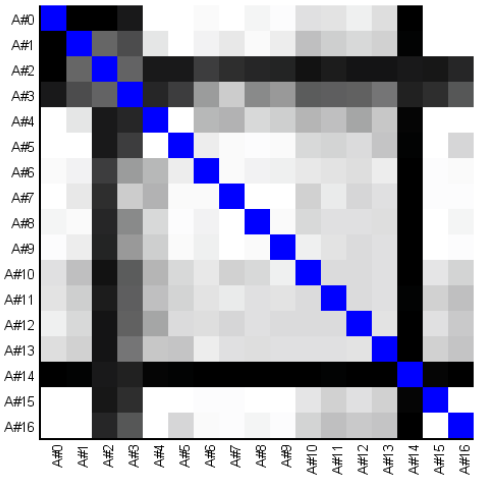
Die zehn Turniere dieser Messreihe besitzen jeweils eine Heatmap zur Visualisierung der Zusammenhänge $W_{ab} = W_{ba}$ und $W_{ab} = 1 - W_{ba}$. Dabei beschreibt der Wert W_{ab} den normierten WTL Score in dem Match zwischen Agent A (Zeile) und B (Spalte), die auf den Achsen der Heatmap abgelesen werden können. Diese Heatmaps wurden bereits in Kapitel 4.3 im Abschnitt *TSResultWindow* als fortgeschrittene Analysen der Heatmapdaten vorgestellt. Mithilfe der Untersuchung $W_{ab} = W_{ba}$ kann untersucht werden, ob immer der Agent gewinnt, der als Erster/Zweiter beginnt. Die andere Untersuchung $W_{ab} = 1 - W_{ba}$ bestimmt, ob einer der beiden Agenten eines Paares immer gewinnt. Diese Daten der zehn Turniere jeden Turnierpaares werden gemittelt, um jeweils eine Ergebnisheatmap zu erhalten. Die dahinterliegenden detaillierten Zahlen der gemittelten Matches und der Standardabweichung über alle Turniere finden sich als Tabellen im Anhang. Umso

heller ein Feld in der Heatmap dargestellt ist, desto höher ist die Übereinstimmung des untersuchten Zusammenhangs.

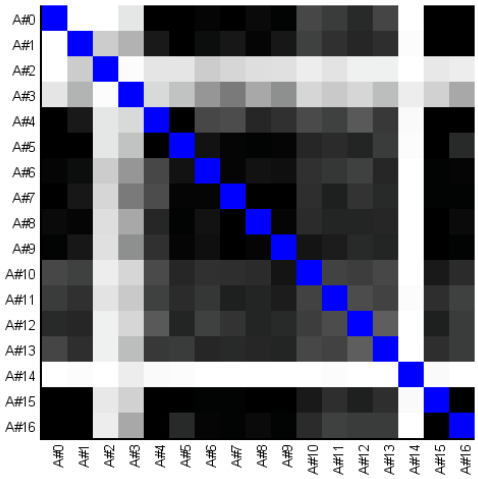
Ergebnis

Für das Spiel Hex finden sich die Ergebnisheatmaps in Abbildung 10(a) bis 10(d). Dort ist zu erkennen, dass es kein klares Ergebnis für einen der beiden Zusammenhänge gibt, sondern eine Mischung. Manche der Agenten gewinnen bei Hin- und Rückrunde der Matches, manche immer dann, wenn sie zuerst anfangen können. Dies ist an den wenigen vollständig weißen und schwarzen Feldern zu erkennen. Diese stellen eine volle Erfüllung(weiß)/Wiederlegung(schwarz) des Zusammenhangs in dem Match dar. Es ist aber auch an den überwiegenden Grautönen zu erkennen, dass ein klares Ergebnis die Ausnahme ist und in den meisten Fällen die Ergebnisse variieren. Im Anhang finden sich die Werte, auf denen die Heatmaps basieren und die zugehörige Standardabweichung für jedes Match in den Tabellen 12 und 13. Die beiden besten Agenten dieser Messreihe sind Agent#4 (*maxn-TD08-hashed*) und Agent#0 (*TDNTuple2_3P-3ply*). Für beide Agenten ist in Abbildung 10(a) an den überwiegend hellen Feldern zu erkennen, dass der dort als erstes spielende Agent überwiegend gewinnt. Für den MaxN Agenten#4 ist zu erkennen, nur im "Kampf" gegen die anderen MaxN-Agenten #2 und #3 und den Random-Agenten #14 ist anhand der dunklen Felder zu erkennen, dass dort nicht bei Hin- und Rückrunde der jeweils zuerst spielende Agent gewinnt. Dies deckt sich mit dem Ranking des Turniers, da die Agenten #2, #3 und #14 die schlechtesten sind. Sie verlieren also in der Regel, egal, ob sie als erstes oder zweites beginnen zu spielen. Für den TDNTuple2 Agenten #0 ist auch zu erkennen, dass er überwiegend als zuerst spielender gewinnt, nur nicht gegen die Agenten #1-#3 und #14. Agent #1 ist dabei ein anderer TDNTuple2. In Abbildung 10(b) kann für die beiden besten Agenten #4 und #0 abgelesen werden, dass die Felder ihrer Turniere überwiegend sehr dunkel sind. Es ist also nicht immer der gleiche Agent ruhmreich, sondern der zuerst spielende. Nur die schon zuvor aufgefallenen Agenten wurden auch laut dieser Visualisierung zuverlässig in Hin- und Rückrunde besiegt.

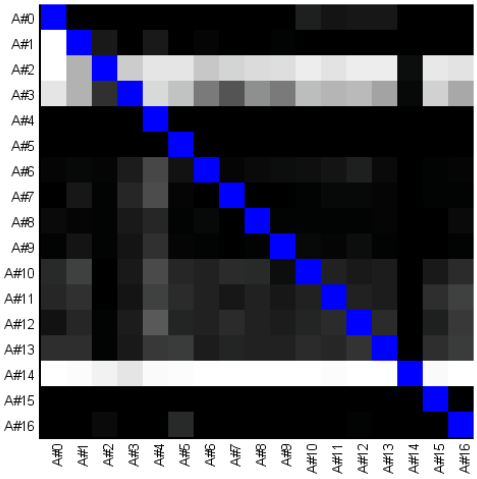
Für das Spiel Vier Gewinnt finden sich die Ergebnisheatmaps in Abbildung 11(a) bis 11(d). Dort ist ein ähnlicher Trend zu erkennen. Auch in diesem Spiel gibt es keine eindeutige Erfüllung der einen oder anderen Untersuchung für alle Matches. Es lassen sich auch hier gewisse Muster für einzelne Agenten erkennen, aber das Ergebnis jedes Matches hängt von den jeweiligen Agenten ab. Im Anhang finden sich die



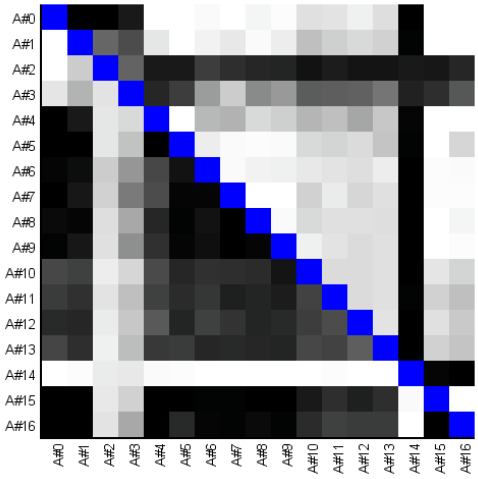
(a) $|W_{ab} = W_{ba}|$ - Umso kleiner der Betrag der Differenz zwischen W_{ab} und W_{ba} desto heller das Feld.



(b) $W_{ab} = 1 - W_{ba}$ - Umso kleiner die Differenz zwischen W_{ab} und $1 - W_{ba}$ desto heller das Feld.



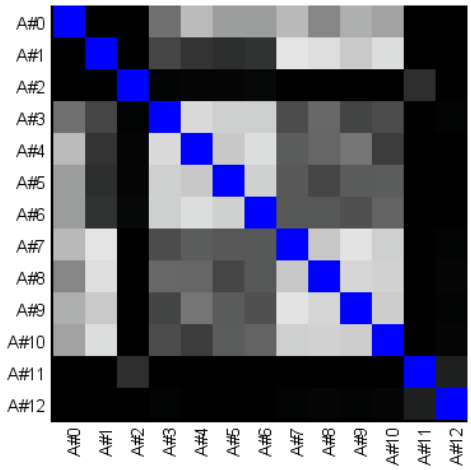
(c) WTL Scores der Matches. Umso dunkler ein Feld desto höher der WTL Score. Es spielt Zeilen- gegen Spaltenagent



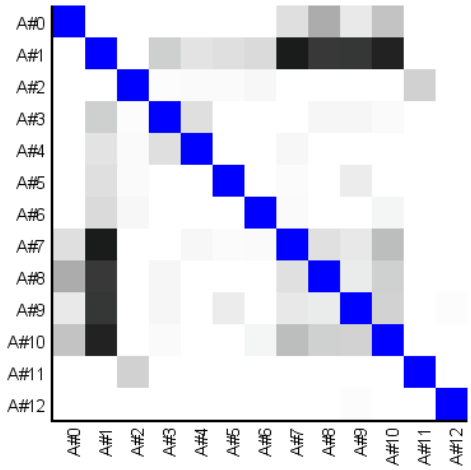
(d) links von der Diagonale sind die Werte aus Abb.10(b) und rechts von der Diagonale die Werte aus Abb.10(a)

Abbildung 10: Heatmaps zu Messreihe M1 im Spiel Hex.

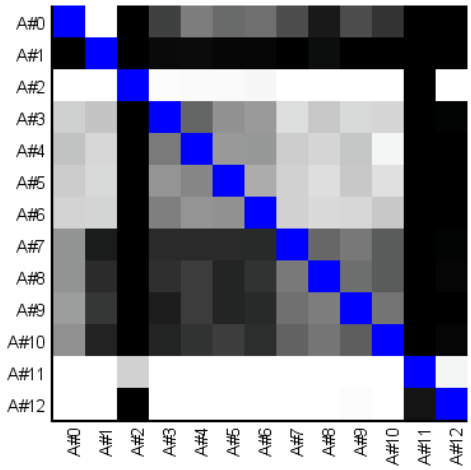
Werte, auf denen die Heatmaps basieren und die zugehörige Standardabweichung für jedes Match in den Tabellen 14 und 15. Auch in diesem Spiel ist eine Symmetrie um die blaue Diagonale zu erkennen, allerdings stellt diese das Gegenteil zu der im Spiel Hex dar. Bei Hex ist eher eine Erfüllung des Zusammenhangs $W_{ab} = W_{ba}$ zu erkennen, bei Vier Gewinnt ist es eine stärkere Erfüllung von $W_{ab} = 1 - W_{ba}$. Die beiden besten Agenten dieser Messreihe sind Agent#1 (*TCL-EXP-al20-lam05-500k-HOR010-T*) und Agent#7 (*mcts-IT2000-TD10-RD200-V0*). Für beide Agenten ist in Abbildung 11(a) an den überwiegend dunklen Feldern zu erkennen, dass der dort als erstes spielende Agent überwiegend verliert. Ob er dabei den ersten Zug einer Episode tätigt hat keinen großen Einfluss auf das Ergebnis. In Abbildung 11(b) visualisiert, dass in diesem Spiel überwiegend der bessere von zwei Agenten die Hin- und Rückrunde eines Matches gewinnt.



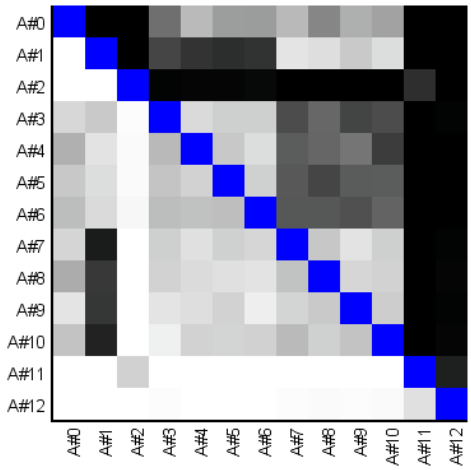
(a) $|W_{ab} = W_{ba}|$ - Umso kleiner der Betrag der Differenz zwischen W_{ab} und W_{ba} desto heller das Feld.



(b) $W_{ab} = 1 - W_{ba}$ - Umso kleiner die Differenz zwischen W_{ab} und $1 - W_{ba}$ desto heller das Feld.



(c) WTL Scores der Matches. Umso dunkler ein Feld desto höher der WTL Score. Es spielt Zeilen- gegen Spaltenagent



(d) links von der Diagonale sind die Werte aus Abb.11(b) und rechts von der Diagonale die Werte aus Abb.11(a)

Abbildung 11: Heatmaps zu Messreihe M1 im Spiel Vier Gewinnt.

6.2.2 M2 - Zufällige Startzustände

Mithilfe der Messreihe M2 soll die Forschungsfrage F2 aus Kapitel 1 untersucht werden. Das Turniersystem bietet die Möglichkeit, die ersten Züge eines Matches zufällig zu generieren und den spielenden Agenten vorzugeben. Dadurch kann untersucht werden, ob ein Agent auch perfekt spielt, wenn er von einem anderen Zustand als dem Anfangszustand aus startet.

Durchführung

Es spielen die 17(Hex)/13(Vier Gewinnt) zuvor bestimmten Agenten in einem Doppel Rundenturnier mit zehn Episoden pro Match gegeneinander. Es werden Turniere mit null, einem und zwei zufälligen Startzügen durchgeführt, jedes Turnier wird zehn Mal wiederholt, um die Messgenauigkeit bei zufälligen Schwankungen zu verbessern. Die Ergebnisse jedes Turniers werden automatisch abgespeichert.

Auswertung

Die zehn Turniere jeder der drei Reihen werden ausgewertet und zusammengefasst zu einem gemittelten Ranking. Dazu werden die zehn Glicko2 Werte jedes Agenten gemittelt und dann alle Agenten nach diesem Wert sortiert. Die Ergebnisse dieser Auswertung findet sich im Anhang in den Tabellen 16-18. Diese Tabellen beinhalten das gemittelte Ranking mit Glicko2 Werte und dazugehöriger Standardabweichung jedes Agenten. Weiterhin wurden die Glicko2 Werte aller Turniere pro Agent zusammengefasst und in Box-Whisker-Plots visualisiert. Diese Plots wurden in Excel 2016 erstellt und visualisieren das Minimum, das untere Quartil, den Median, das obere Quartil und das Maximum. Zusätzlich werden die Rankings jedes Agenten dieser Messreihe mit dem Ranking des Turniers ohne zufällige Startzüge (RSM0) verglichen. Dabei wird für jeden Agenten erfasst, wie viele Platzierungen sich das Ranking in der Turnierreihe verschoben hat. Für die Ergebnisse werden zwei Metriken erfasst:

- Mittel aller Verschiebung > 0
- Anzahl Verschiebungen > 0

Ergebnis

Für das Spiel Hex findet sich der Box-Whisker-Plot in Abbildung 12 und die Tabelle der Verschiebungen in Tabelle 4. Die Tabelle zeigt, dass sich die Reihenfolge

Anteil gespielter Matches:	RSM0	RSM1	RSM2
Mittel der Verschiebungen > 0	0	3,5	5,57
Anzahl Differenzen > 0	0	12	14

Tabelle 4: Mittel der Anzahl der Verschiebungen eines Ranges in Relation zur Position im Turnier mit keinem zufälligen Startzug (RSM0) der Matches und die Anzahl der verschobenen Rankings in Messreihe M2 im Spiel Hex

des Rankings in RSM1 mit einem zufälligen Startzug stark verändert. In diesem Turnier besitzen 12 der 17 Agenten ein verändertes Ranking mit einer mittleren Abweichung von 3,5 Platzierungen bei den verschobenen Agenten. Bei den Turnieren mit zwei zufälligen Startzügen verstärkt sich dieser Trend noch zu 14 von 17 veränderten Rankings mit einer mittleren Verschiebung von 5,57 Platzierungen.

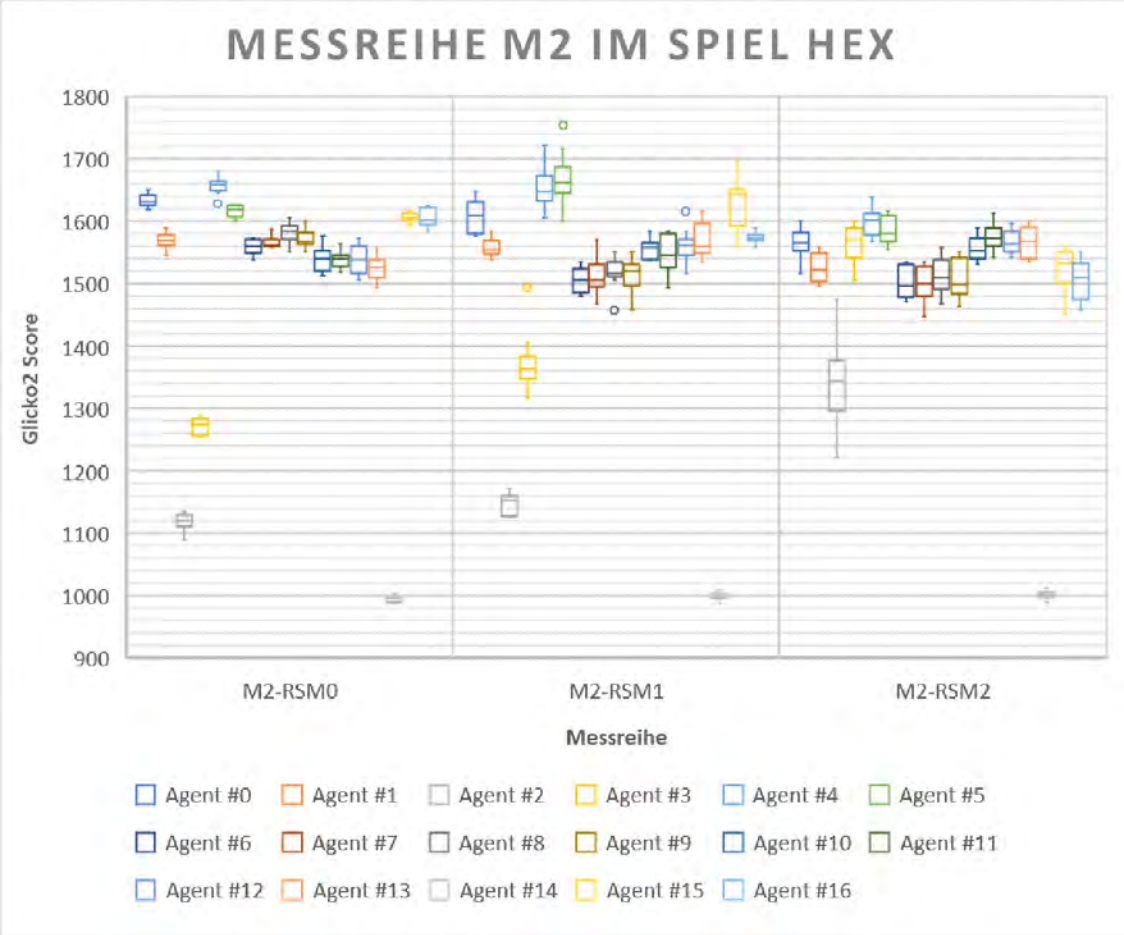


Abbildung 12: Visualisierung der Messungen aus Messreihe M2 im Spiel Hex mit 0/1/2 zufälligen Startzuständen (RSM)

Der Box-Whisker-Plot in Abbildung 12 visualisiert, dass sich die Glicko2 Rankings

der Agenten über die Episoden der Turniere mit steigender Anzahl von zufälligen Startzügen immer weniger konstant werden. Bei RSM0 sind die einzelnen Boxen noch sehr klein, es sind sich also alle Glicko2 Werte sehr ähnlich. Bei den anderen beiden Reihen RSM1 und RSM2 kann beobachtet werden, dass die Boxen vertikal länger werden und auch die Minimum/Maximum Ausschläge größer werden.

Anteil gespielter Matches:	RSM0	RSM1	RSM2
Mittel der Verschiebungen > 0	0	2,75	2,4
Anzahl Differenzen > 0	0	8	5

Tabelle 5: Mittel der Anzahl der Verschiebungen eines Ranges in Relation zur Position im Turnier mit keinem zufälligen Startzug (RSM0) der Matches und die Anzahl der verschobenen Rankings in Messreihe M2 im Spiel Vier Gewinnt

Für das Spiel Vier Gewinnt findet sich der Box-Whisker-Plot in Abbildung 13 und die Tabelle der Verschiebungen in Tabelle 5. Die Tabelle zeigt, dass sich die Reihenfolge des Rankings in RSM1 mit einem zufälligen Startzug stark verändert. In diesem Turnier besitzen 8 der 13 Agenten ein verändertes Ranking mit einer mittleren Abweichung von 2,75 Platzierungen bei den verschobenen Agenten. Bei den Turnieren mit zwei zufälligen Startzügen vermindert sich dieser Trend zu 5 von 13 veränderten Rankings mit einer mittleren Verschiebung von 2,4 Platzierungen. Im Gegensatz zum Spiel Hex vermindert sich hier die Anzahl der veränderten Platzierungen und die Stärke der Verschiebung.
Der Box-Whisker-Plot in Abbildung 13 visualisiert einen ähnlichen Trend wie Tabelle 5. Bei keinem zufälligen Starthalbzug in RSM0 besitzen die Boxen des Plots eine kleine vertikale Ausdehnung, die Glicko Werte der einzelnen Episoden liegen also nah beieinander. Bei einem zufälligen Starthalbzug in RSM1 wächst die vertikaler Ausdehnung der Boxen an, bei *Agent*#1 sogar stark. Wenn die Anzahl der zufälligen Starthalbzüge in RSM2 auf zwei steigt, bleibt die vertikale Auslenkung der Boxen im Plot fast gleich und verringert sich bei einige sogar leicht. Speziell *Agent*#1 reduziert seine vorherige starke Auslenkung wieder, nur *Agent*#0 zeigt eine stärkere Auslenkung als zuvor.

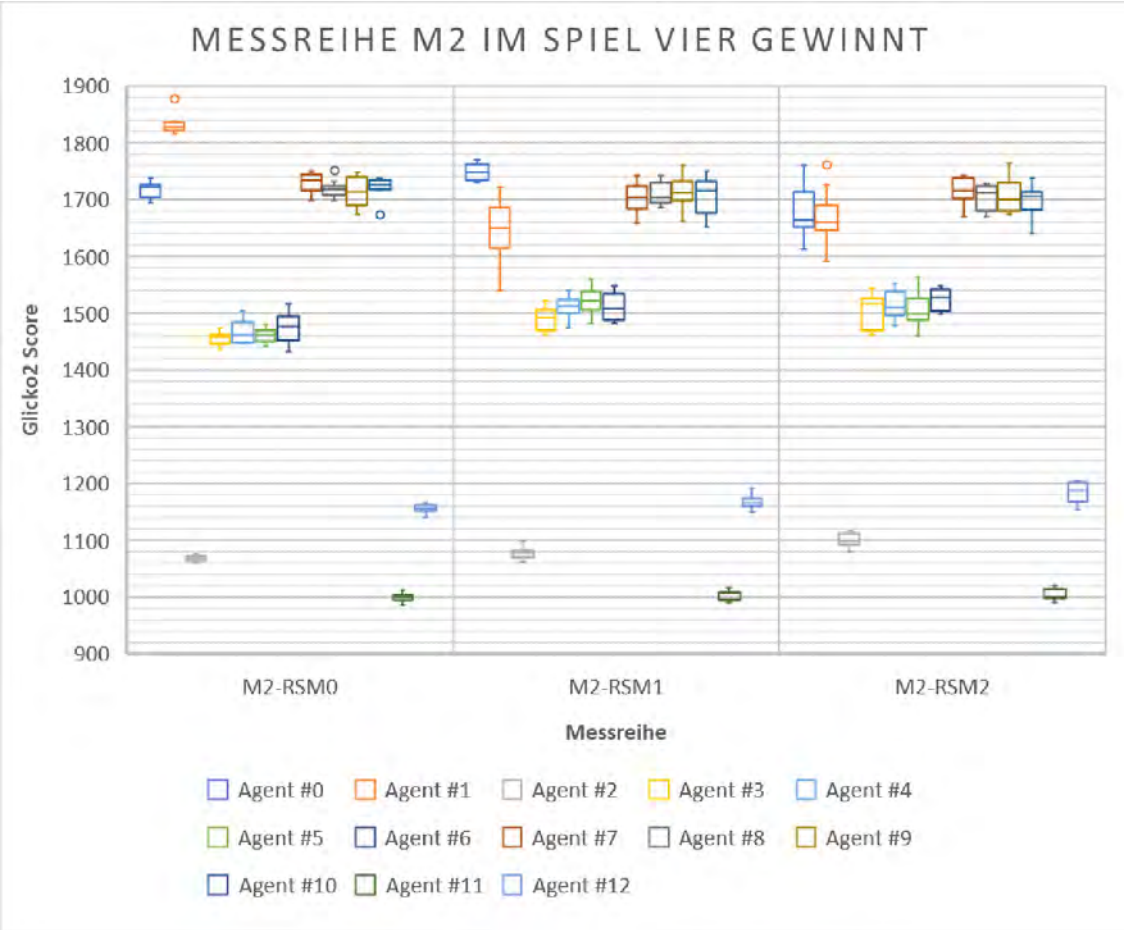


Abbildung 13: Visualisierung der Messungen aus Messreihe M2 im Spiel Vier Gewinnt mit 0/1/2 zufälligen Startzuständen (RSM)

6.2.3 M3 - Approximative Rankings

Mithilfe der Messreihe M3 soll die Forschungsfrage F3 aus Kapitel 1 untersucht werden. Dafür wird die Funktion des variablen Doppelrundenturniers verwendet. Es soll untersucht werden, wie gut das Ranking eines vollen Doppelrundenturniers mittels Glicko2 angenähert werden kann, wenn die Anzahl der Matches reduziert wird.

Durchführung

Es spielen die 17(Hex)/13(Vier Gewinnt) zuvor bestimmten Agenten in einem Doppel-Rundenturnier mit 272/156 Matches mit jeweils zehn Episoden gegeneinander. Darauf folgen Turniere mit schrittweise reduzierter Anzahl an Matches (75%, 50%, 25%, 12%, 6%). Dafür werden aus dem vollständigen Turnier zufällig Mat-

ches entfernt, bis die gewünschte Menge übrig ist. Bei der Auswahl der zufällig zu entfernenden Matches wird darauf geachtet, dass bei den übrigen Matches jeder Agent mindestens einmal spielt. Jedes Turnier wird zehn Mal wiederholt, um die Messgenauigkeit bei zufälligen Schwankungen zu verbessern. Die Ergebnisse jedes Turniers werden automatisch abgespeichert.

Auswertung

Nach den Messungen werden die zehn Ergebnisse jedes Turniers zu einem Ergebnis zusammengefasst, um Schwankungen auszugleichen. Dafür wird der durchschnittliche Glicko2 Wert jedes Agenten eines Turniers bestimmt. Zur Auswertung dieser Messreihe werden die Rankings jedes Agenten mit dem Ranking der 100% Reihe verglichen. Dabei wird für jeden Agenten erfasst wie viele Platzierungen sich das Ranking in der Turnierreihe verschoben hat. Für die Ergebnisse werden zwei Metriken erfasst:

- Mittel aller Verschiebung > 0
- Anzahl Verschiebungen > 0

Diese Metrik wird für alle der fünf Reihen (75%, 50%, 25%, 12%, 6%) erfasst. Weiterhin wird bestimmt, bei wie vielen der zehn Episoden sich der beste Agent des Turniers auf dem ersten Platz befindet.

Ergebnis

In Abbildung 14 sind die Ergebnisse der Messreihe M3 im Spiel Hex visualisiert.Bei 100% der gespielten Spiele betragen alle Werte 0, da dies der Referenzwert der anderen Messungen ist. In den Messungen 75% und 50% ist zu erkennen, dass 10 bzw. 11 der 17 Rankings nicht mehr mit der 100% Reihe übereinstimmen. Die Verschiebungen der abgewichenen Rankings beträgt im Mittel allerdings nur 1,6 bzw. 2,18 Platzierungen. Bei 25% und 12% der gespielten Matches steigt die Abweichung stärker an, hier sind 13 der 17 Platzierungen des Rankings verschoben, um im Mittel 4,15 Ränge. In der letzten Reihe mit nur 6% der gespielten Matches findet sich die höchste Abweichung von einem Turnier mit 100% der gespielten Matches. Hier haben sich 14 der 17 Platzierungen um im Mittel 5,14 Platzierungen verschoben. In Tabelle 6 kann abgelesen werden, in wie vielen der zehn Episoden jedes Turniers sich der beste Agent des 100% Turniers auf dem ersten Platz befindet. Im ersten Turnier, mit 100% der Matches, ist es in 90% der Episoden der Fall, danach sinkt

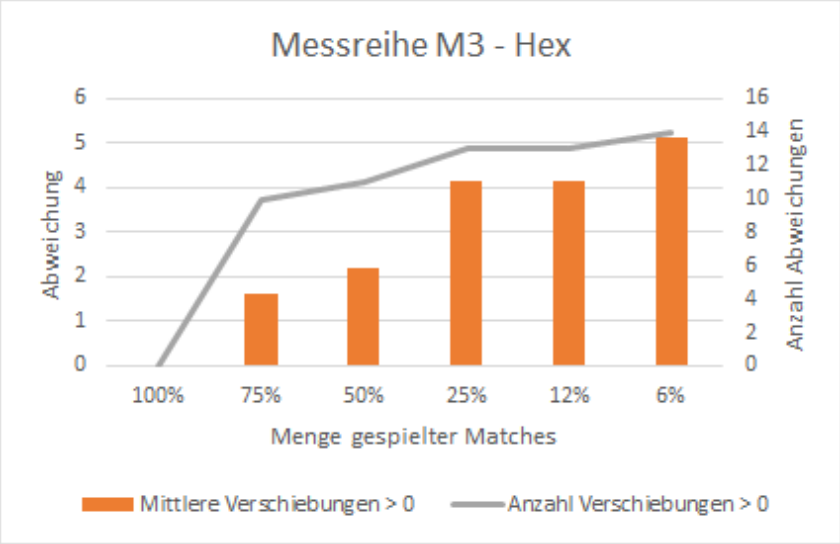


Abbildung 14: Visualisierung der Messungen aus Messreihe M3 im Spiel Hex

die Rate stark ab auf 40% bzw 20%. Weiterhin wird in Abbildung 15 dargestellt, wie sich die Glickoscores der besten drei Agenten aus dem 100% Turnier in den Messungen verhalten. Es ist zu erkennen, dass mit fortschreitender Reduzierung der Matches die Glicko Ratings immer weiter streuen aber im Mittel recht ähnlich bleiben. In Tabelle 7 ist beispielhaft dargestellt, wie die Glicko Ratings der besten

Gespielte Matches	100%	75%	50%	25%	12%	6%
Bester Agent auf Platz 1	90%	40%	20%	20%	0%	20%

Tabelle 6: Prozentualer Anteil, wie oft der beste Agent in den Episoden der Turniere von Messreihe M3 im Spiel Hex auf Platz 1 ist.

drei Agenten über die zehn Episoden des Turniers mit 50% der gespielten Matches ausfallen. Agent#4 besitzt dabei eine Standardabweichung von 76, Agent#0 von 80 und Agent#5 von 77. Es ist eine Schwankung in den Ratings eines Agenten zu erkennen, aber diese ist nicht stark ausgeprägt.

	Ep. 1	Ep. 2	Ep. 3	Ep. 4	Ep. 5	Ep. 6	Ep. 7	Ep. 8	Ep. 9	Ep. 10
Agent #4	1690.04	1572.37	1615.25	1759.30	1735.68	1642.50	1538.42	1667.98	1743.54	1596.05
Agent #0	1576.84	1543.23	1730.51	1551.22	1650.12	1465.88	1680.69	1463.48	1662.84	1554.28
Agent #5	1567.59	1638.72	1602.53	1519.21	1618.99	1699.93	1506.40	1689.89	1728.01	1548.94

Tabelle 7: Glicko Werte der besten drei Agenten in allen Episoden im Spiel Hex in Messreihe M3 im 50% Turnier

Die Ergebnisse der Messreihe M3 im Spiel Vier Gewinnt sind in Abbildung 16

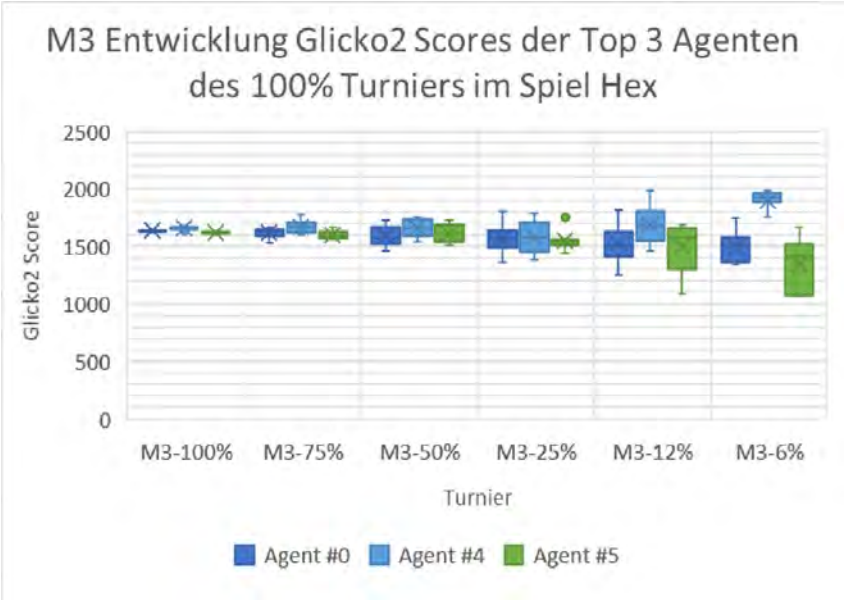


Abbildung 15: Visualisierung der Top 3 Agenten Glicko Ratings der einzelnen Turniere aus Messreihe M3 im Spiel Hex

visualisiert.Bei 100% der gespielten Spiele betragen alle Werte 0, da dies der Referenzwert der anderen Messungen ist. In den Messungen 75%, 50% und 25% ist zu erkennen, dass 8 der 13 Rankings nicht mehr mit der 100% Reihe übereinstimmen. Die Verschiebungen der abgewichenen Rankings beträgt im Mittel allerdings nur 1,75 bzw. 1,5 Platzierungen und fällt somit sehr niedrig aus. In den Reihen mit nur 12% und 6% der gespielten Matches sinkt die Anzahl der verschobenen Rankings erst auf sieben und steigt dann auf elf. Die mittlere Verschiebung liegt dabei bei 2,0 und 1,8 Platzierungen. In Tabelle 8 kann abgelesen werden, in wie vielen der zehn Episoden jedes Turniers sich der beste Agent des 100% Turniers auf dem ersten Platz befindet. Im ersten Turnier, mit 100% der Matches, ist es in 100% der Episoden der Fall, danach halbiert sich die Rate auf 50% und sinkt danach auf 40% bzw. 30%. Weiterhin wird in Abbildung 17 dargestellt, wie sich die Glicko-Ratings der besten drei Agenten aus dem 100% Turnier in den Messungen verhalten. Es ist zu erkennen, dass mit fortschreitender Reduzierung der Matches die Glicko Ratings relativ konstant bleiben und nicht so stark streuen wie im Spiel Hex.

Gespielte Matches	100%	75%	50%	25%	12%	6%
Bester Agent auf Platz 1	100%	50%	50%	40%	30%	40%

Tabelle 8: Prozentualer Anteil, wie oft der beste Agent in den Episoden der Turniere von Messreihe M3 im Spiel Vier Gewinnt auf Platz 1 ist.

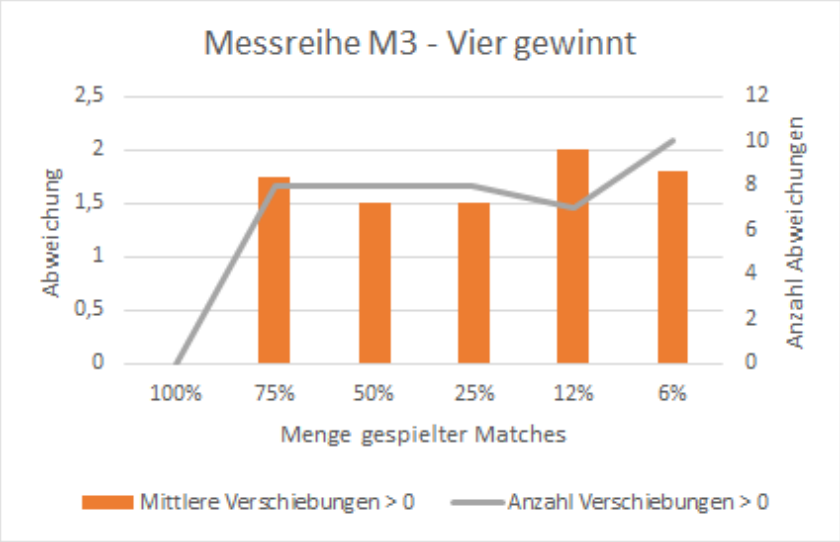


Abbildung 16: Visualisierung der Messungen aus Messreihe M3 im Spiel Vier Gewinnt

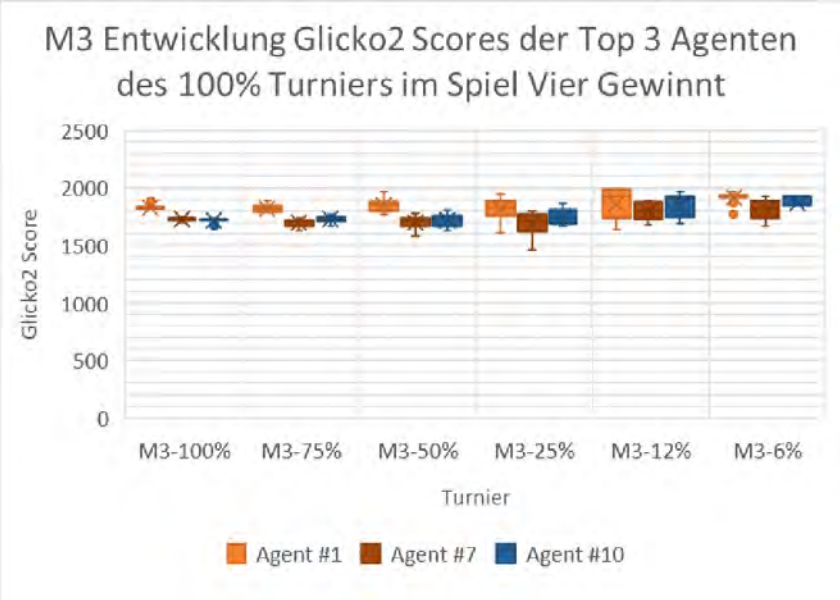


Abbildung 17: Visualisierung der Top 3 Agenten Glicko Ratings der einzelnen Turniere aus Messreihe M3 im Spiel Vier Gewinnt

In Tabelle 9 ist beispielhaft dargestellt, wie die Glicko Ratings der besten drei Agenten über die zehn Episoden des Turniers mit 50% der gespielten Matches ausfallen. Agent#1 besitzt dabei eine Standardabweichung von 60, Agent#7 von 55 und Agent#10 von 57. Es ist eine Schwankung in den Ratings eines Agenten zu erkennen, aber diese ist nicht stark ausgeprägt.

	Ep. 1	Ep. 2	Ep. 3	Ep. 4	Ep. 5	Ep. 6	Ep. 7	Ep. 8	Ep. 9	Ep. 10
Agent #1	1865.31	1763.47	1877.36	1806.75	1888.87	1832.96	1962.38	1763.81	1826.03	1872.48
Agent #7	1682.59	1668.90	1767.11	1671.24	1719.11	1583.99	1716.72	1779.77	1704.26	1726.78
Agent #10	1731.41	1653.38	1744.67	1627.81	1767.11	1739.61	1669.98	1759.22	1810.31	1680.69

Tabelle 9: Glicko Werte der besten drei Agenten in allen Episoden im Spiel Vier Gewinnt in Messreihe M3 im 50% Turnier

6.2.4 M4 - Reproduzierbarkeit

Mithilfe der Messreihe M4 soll die Forschungsfrage F4 aus Kapitel 1 untersucht werden. Durch wiederholte Messungen mit gleichen Einstellungen wird untersucht, wie stark die Messergebnisse dabei variieren.

Durchführung

Es spielen die 17(Hex)/13(Vier Gewinnt) zuvor bestimmten Agenten in einem Doppel-Rundenturnier mit zehn Episoden pro Match gegeneinander. Dieses Turnier wird zehn Mal wiederholt, um die Messgenauigkeit bei zufälligen Schwankungen zu verbessern. Die Ergebnisse jedes Turniers werden automatisch abgespeichert. Da dieses Vorgehen dem in Messreihe M1 entspricht werden die Messergebnisse aus M1 verendet, um eine Wiederholung der selben Messungen zu vermeiden.

Auswertung

Die zehn Glicko Ratings der einzelnen Agenten werden in einem Box-Whisker-Plot visualisiert. Darin lassen sich Aussagen über die Streuung und Verteilung der einzelnen Ratings treffen. Weiterhin werden die Mittelwerte und Standardabweichungen der Ratings in einer Tabelle zusammengefasst. Diese konkreten Zahlen erlauben weitergehende Aussagen, wie konstant die Ratings der Agenten in den zehn Turnieren sind.

Ergebnis

In Abbildung 18 ist zu erkennen, dass die Glicko-Ratings der Agenten im Spiel Hex recht konstant ausfallen. Die Boxen im Plot besitzen für jeden Agenten eine geringe vertikale Ausdehnung. Diese Beobachtung wird durch die Standardabweichungen (SD) der Glicko Werte unterstrichen. Im Spiel Hex besitzt der Agent#12 die höchste SD von 24,6 bei einem mittleren Rating von 1539, alle Hex Agenten dieser Messungen besitzen zusammen eine mittlere SD von 13,8. Der Großteil der Agenten befinden sich mit ähnlich hohen Rankings beieinander, nur Agenten #2,

#3 und #14 weichen stark davon ab. Agent#14 ist der Random Agent, bei ihm ist dieses Ergebnis zu erwarten, da er nur zufällige Züge tätigt. Die anderen beiden Agenten sind vom Typ Max-N mit geringer Baumtiefe. In Abbildung 19 kann

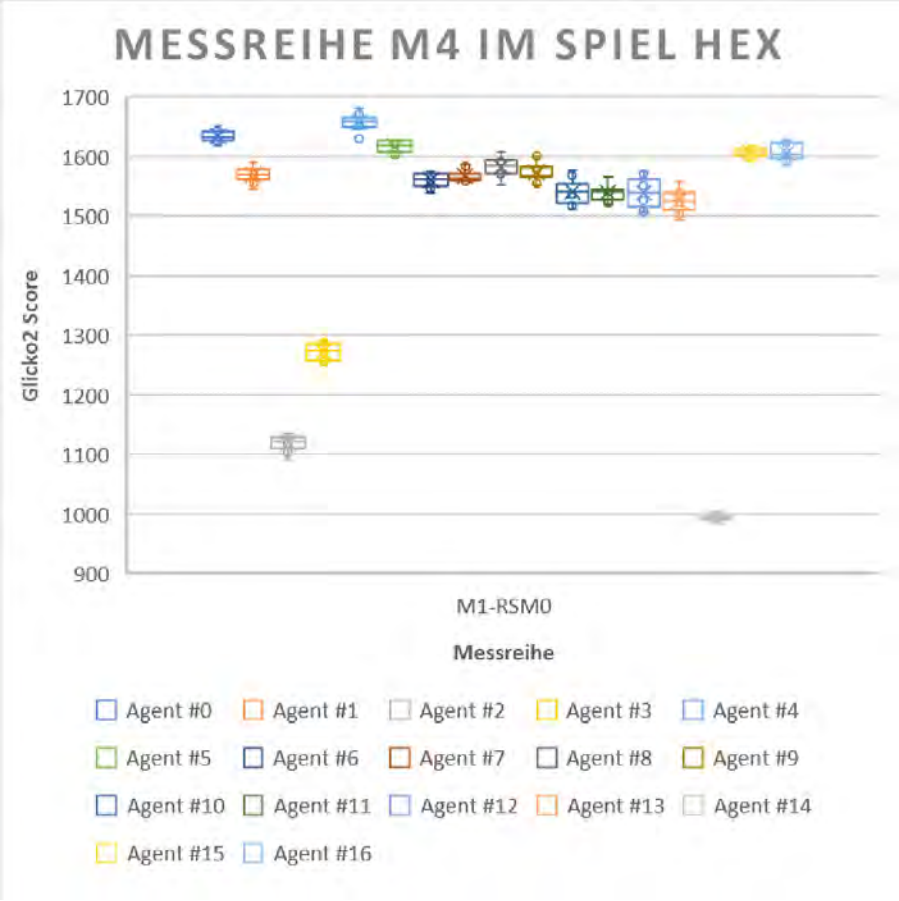


Abbildung 18: Visualisierung der Messungen aus Messreihe M4 im Spiel Hex

für das Spiel Vier Gewinnt der gleiche Trend erkannt werden, dass sich die Glicko Rankings jedes Agenten nicht stark voneinander unterscheiden. In diesem Spiel besitzt Agent#6 mit die höchste SD in dieser Messung von 25,5 bei einem Glicko von 1473,8. Die mittlere SD aller Agenten beträgt 15,5. Es kann weiterhin beobachtet werden, dass sich die Rankings der Agenten mit gleichem Typ wenig unterscheiden. Dies ist in der Abbildung an der Gruppierung der Boxen zu erkennen. Nur die TD-NTuple-2 Agenten im linken und rechten Bereich des Plots besitzen niedrigere Glicko-Ratings als die anderen Agenten.
Im Anhang in Tabelle 19 (Seite 84) befinden sich die detaillierten Glicko Werte der besten drei Agenten aus beiden Spielen zusammen mit der jeweiligen Standardabweichung. Im Spiel sind die Standardabweichungen jeweils unterhalb von 14 und

in Vier Gewinnt jeweils unter 20. Dies belegt die Beobachtung aus den Plots, dass sich die Glickowerte nah beieinander befinden.

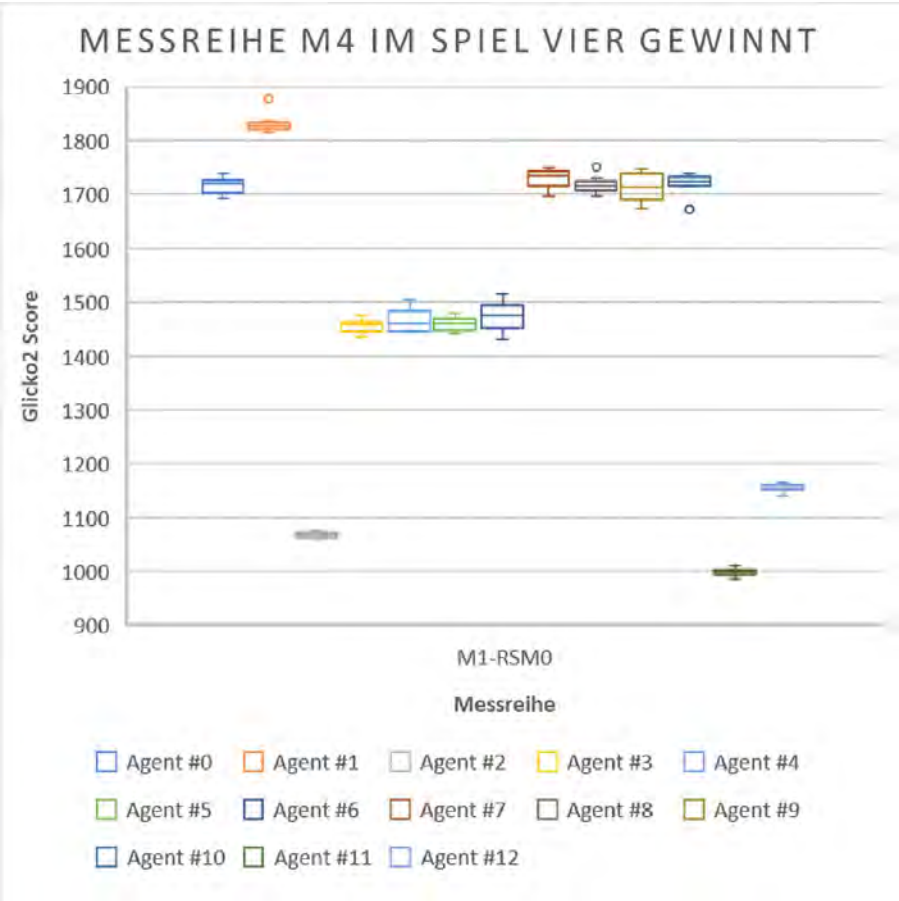


Abbildung 19: Visualisierung der Messungen aus Messreihe M4 im Spiel Vier Gewinnt

6.2.5 M5 - Grenzen der Zeitmessung in Java

Mithilfe der Messreihe M5 soll die Forschungsfrage F5 aus Kapitel 1 untersucht werden. Dafür wird mit einem Testprogramm erfasst, wie hoch die Präzision der Java-Zeitmessung im Mikro- und Nanosekundenbereich ist. Diese Messung wird auf verschiedenen Computern und Betriebssystemen ausgeführt und anschließend verglichen.

Durchführung

Ein Testprogramm misst in einer Schleife 75 Mal die Zeit in Nanosekunden mittels *System.nanoTime()*, die verstreicht während ein Integer Array wiederholt sortiert

wird. Als Sortieralgorithmus kommt dabei eine Java-Implementation des Insertionsort zum Einsatz. Dieser ist mit einer Laufzeit von $O(n^2)$ nicht effizient, aber verbraucht so reproduzierbar Zeit. Das Sortieren der gleichen Menge an Zahlen sollte immer gleich lang dauern. Für diese Untersuchung wird die Anzahl an zu sortierenden Elementen schrittweise reduziert und die Dauer jeder Messung gespeichert. Folgende Mengen an Zahlen sollen sortiert werden: 3000, 2000, 1000, 750, 500, 400, 300, 200, 100. Die genutzten Variablen werden einmalig vor der Messung initialisiert, um das Eingreifen des Java Garbage Collectors zu minimieren. Diese Messung wird neben dem bisher genutzten Testsystem unter Windows 10 zusätzlich unter Ubuntu 18.04 und auch auf einem Lenovo ThinkPad mit Intel i5-4200M Hauptprozessor und 8GB Arbeitsspeicher unter Ubuntu 14.04 64bit durchgeführt, um Verhalten in verschiedenen Umgebungen zu untersuchen.

Auswertung

Für jede Messung der Messreihe werden die gemessenen Zeiten in einem Boxplot visualisiert. Anhand dieser Plots kann erkannt werden, wie sich die Konsistenz der Zeitmessungen in den Reihen entwickelt. Solange die vertikale Ausdehnung der Boxen im Plot gleichmäßig ausfällt, ist die Messung noch präzise. Wenn sich die vertikale Ausdehnung und Messwertausreißer verändern, ist dies in Indiz für sinkende Präzision. Im Anschluss daran wird verglichen, wann diese Grenze der Präzision auf verschiedenen Computer und Betriebssystem erreicht wird.

Ergebnis

In den Ergebnissen des Surface Books unter Windows 10 ist in Abbildung 20 zu erkennen, dass sich der Plot für die Schritte 3000 bis 500 recht konstant verhält. Bis zu diesem Schritt besitzen die Messungen eine gewisse Streuung und auch Fehler, danach werden die zeitlichen Messungen scheinbar sehr viel präziser ohne Ausreißer und sehr geringer Streuung. An den Messwerten direkt lässt sich ablesen, dass sich die Messwerte vor allem im Bereich von 300 bis 100 kaum unterscheiden und sich wenige konkrete Werte immer wieder wiederholen. Diese wenigen, sich immer wieder wiederholenden Fehler sprechen für systematische Messfehler, die sich hier häufen. Die somit letzte zuverlässige Messung bei 400 Elementen entspricht einer Dauer von im Mittel rund 33 Mikrosekunden. Zeitliche Messungen mit einer Dauer von diesem Wert und darunter sollten mit Vorsicht genossen werden, da sie nach dieser Messreihe nicht präzise sind. In Abbildung 21 ist der relative Fehler der

Messreihen aufgetragen. Dort ist zu erkennen, dass der Fehler erst sinkt (zu lesen von rechts nach links) und dann ab (V)500 steigt. Dies belegt die vorherige Beobachtung. In dieser Abbildung wurden Ausreißer der V3000 Messung, die stark vom Median abweichen, nicht abgebildet. Fast alle Werte bei V3000 liegen um rund 1500, deshalb sind die drei Ergebnisse von 3393, 6419 und 8060 deutlich als Ausreißer zu erkennen.

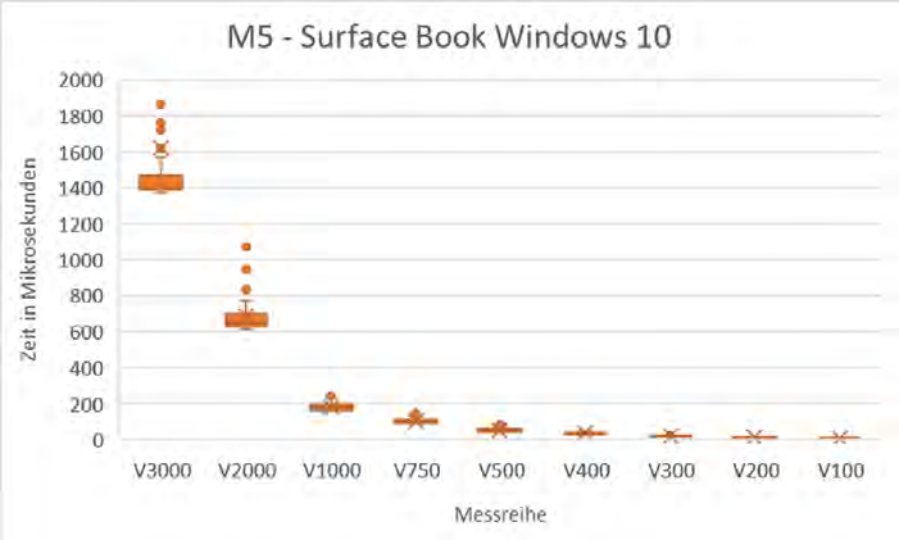


Abbildung 20: Visualisierung der Messungen aus Messreihe M5 auf dem Surface Book unter Windows 10. Zur besseren Lesbarkeit wurden bei V3000 die Ausreißer bei 3393, 6419 und 8060 ausgeblendet.

Die gleichen Messungen wurden auf dem Surface Book mit einem Live Ubuntu 18.04 64bit Betriebssystem wiederholt. Die Ergebnisse in Abbildung 22 und 23 sind sich bis auf ein paar Messausreißer sehr ähnlich. Nur der relative Fehler fällt in der V1000 Reihe sehr viel höher aus. Die letzte zuverlässige Messung liegt hier bei V400 mit relativem einem Fehler von 22%, da bei V500 der relative Fehler auf 47% steigt. Das Sortieren dauert bei V400 im Mittel rund 31 Mikrosekunden. In Abbildung 23 wurden Ausreißer der V3000 und V2000 Messung, die stark vom Median abweichen, nicht abgebildet.
Die Messung wurde auch auf einem Lenovo ThinkPad mit schwächerem Prozessor und Ubuntu 14.04 64bit durchgeführt. Die Ergebnisse in Abbildung 24 zeigen einen ähnlichen Trend wie auf dem Surface Book, allerdings mit sehr viel größerer Streuung. Auch in dieser Messreihe ist eine sehr starke Veränderung der scheinbaren Präzision der Messung ab V500 zu erkennen. Der Fehler in Abbildung 25 zeigt diese Entwicklung ebenfalls. In dieser Messung dauert das Sortieren bei V500

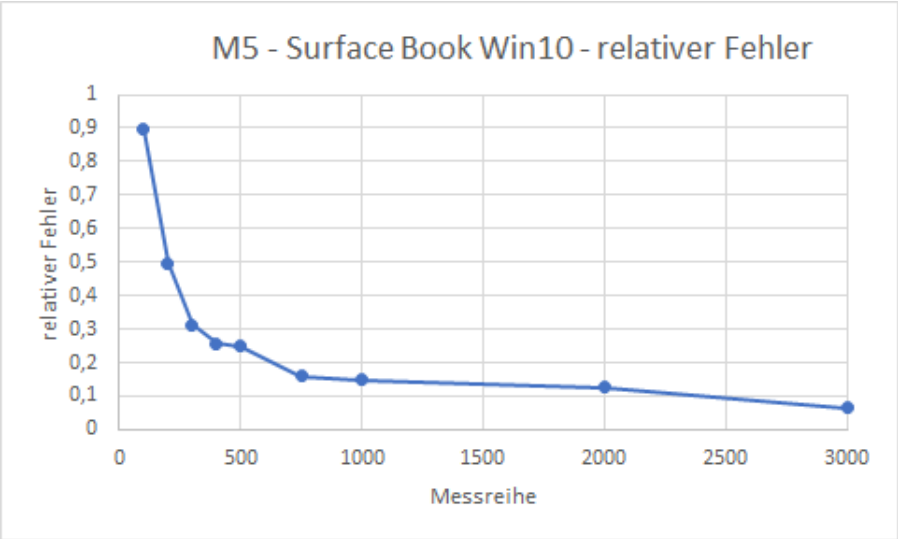


Abbildung 21: Relativer Fehler (Standardabweichung / Mittelwert) aus Messreihe M5 auf dem Surface Book unter Windows 10. Ausreißer bei 3393, 6419 und 8060 wurden aus der Auswertung genommen.

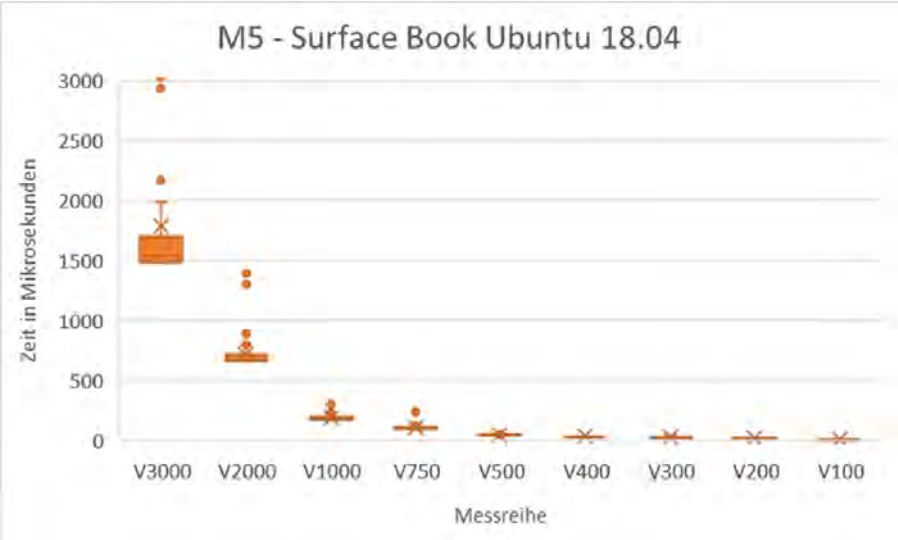


Abbildung 22: Visualisierung der Messungen aus Messreihe M5 auf dem Surface Book unter Ubuntu 18.04. Zur besseren Lesbarkeit wurden bei V3000 die Ausreißer 3393, 6419 und 8060 ausgeblendet.

im Mittel rund 237 Mikrosekunden. Dieser Unterschied in der Dauer kann daran liegen, dass der Prozessor in diesem Laptop nur über zwei Kerne (vier im Surface Book) mit niedrigerer Frequenz verfügt.

Die Messungen wurden auf beiden Systemen mehrfach durchgeführt. Dabei konnte festgestellt werden, dass besonders bei V3000 fast immer mindestens eine Messung

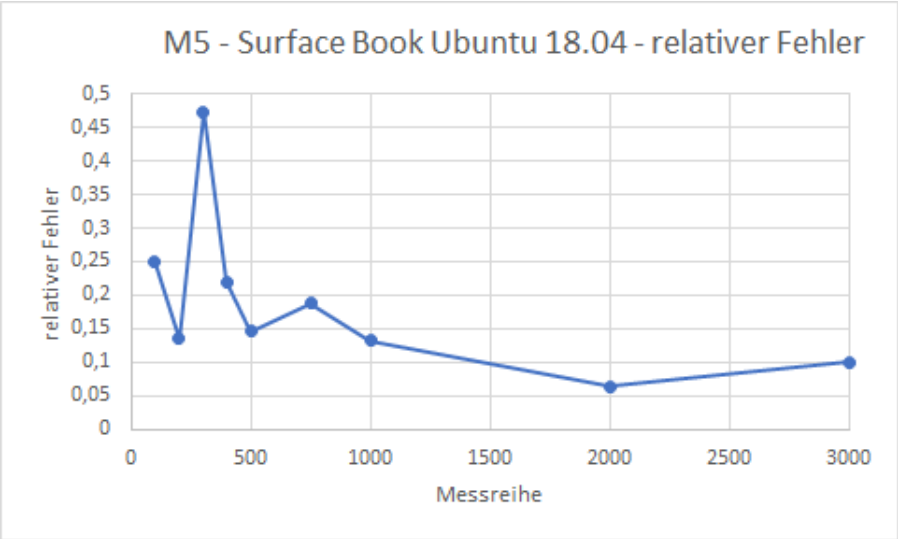


Abbildung 23: Fehler (Standardabweichung gegen mittleren Messwert) aus Messreihe M5 auf dem Surface Book unter Ubuntu 18.04. Zur besseren Lesbarkeit wurden folgende Ausreißer ausgeblendet: V3000: 2937, 3033, 3850, 5680, 6092 | V2000: 1302, 1336, 1391

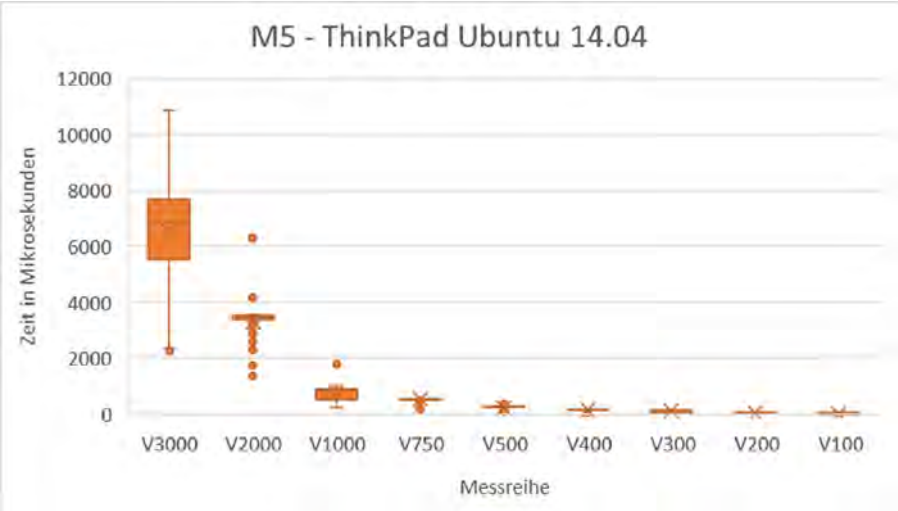


Abbildung 24: Visualisierung der Messungen aus Messreihe M5 auf dem ThinkPad unter Ubuntu 14.04

gegen Ende um rund Faktor 10 von den anderen abgewichen ist. Dies könnte vielleicht durch den Java Garbage Collector verursacht worden sein. Durch Nutzen der gleichen Variablen ohne neue Initialisierung innerhalb der Messschleife konnte dieser Fehler minimiert aber nicht verhindert werden.

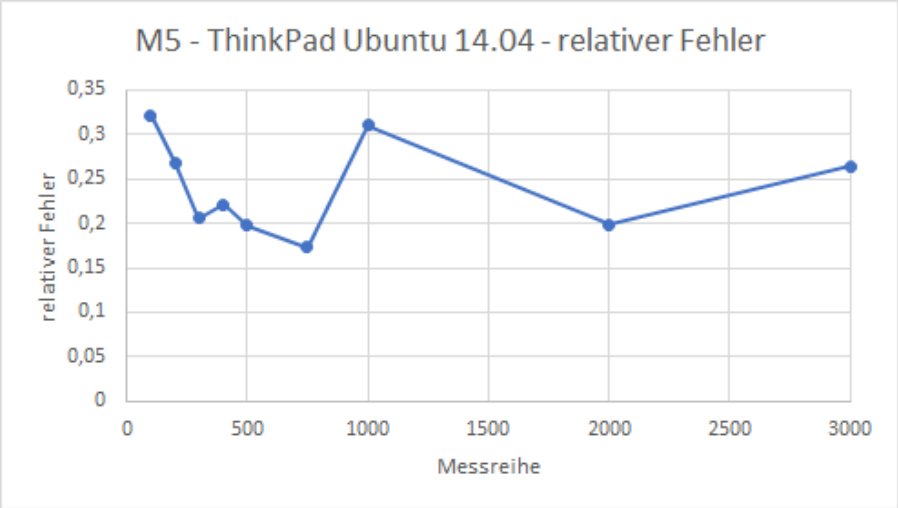


Abbildung 25: Fehler (Standardabweichung gegen mittleren Messwert) aus Messreihe M5 auf dem ThinkPad unter Ubuntu 14.04

7 Evaluation/Diskussion

In diesem Kapitel werden die Ergebnisse der Messungen aus dem vorherigen Kapitel evaluiert und in einer Diskussion in den Kontext eingeordnet.

7.1 M1 - Kommutativität

In Messreihe M1 wurde untersucht, ob zwischen den Wettkampfergebnissen der untersuchten Agenten kommutative Zusammenhänge bestehen. Das Augenmerk lag dabei darauf, ob immer der zuerst ziehende Agent gewinnt oder ob immer derselbe Agent eines Paares gewinnt. Die Ergebnisse der Messungen zeigen, dass in einem Turnier nie nur das eine oder andere erfüllt wird. Es kann beobachtet werden, dass beide Bedingungen für manche Paare voll erfüllt werden, es überwiegend aber nur zu einer Annäherung kommt. Die vielen Grautöne in den Heatmaps der Plots visualisieren das. Es lässt sich allerdings erkennen, dass die Ergebnisse sehr oft einem der beiden Zusammenhänge zugeordnet werden können. Selten gab es bei den untersuchten Agenten Paare, die keinen der beiden Fälle erfüllt haben. Aus den Plots lässt sich als Trend ablesen, dass im Spiel Hex eher der zuerst ziehende Agent gewinnt. Im Spiel Vier Gewinnt gewinnt hingegen eher der gleiche Agent eines Paares, unabhängig davon, ob er beginnt.

7.2 M2 - Zufällige Startzustände

In der zweiten Messreihe wurde untersucht, wie stark zufällige vorgegebene Starthalbzüge die Spielstärke der Agenten beeinflussen. Die Ergebnisse zeigen einen ähnlichen Trend für beide Spiele mit verschieden starken Veränderungen der Agentenrankings in den Turnierergebnissen. Im Spiel Hex fallen die Ergebnisse stärker aus, bei einem zufälligen Halbzug zu Beginn verändern sich 12 der 17 Platzierungen (70,5%), bei Vier Gewinnt sind es nur 8 von 13 (61,8%). Das Mittel der Verschiebung beträgt im Spiel Hex 3,5 Platzierungen (eine Verschiebung um 20,6% der Platzierungen) und im Spiel Vier Gewinnt 2,75 Platzierungen (21,2%). Bei zwei zufälligen Halbzügen zu Beginn des Spiels verhalten sich beide Spiele unterschiedlich. In Hex verschieben sich noch mehr Platzierungen um mehr Ränge (14 Stück = 82,4%) um 5,57 Platzierungen. Das entspricht einer Verschiebung um 32,8% der Platzierungen. Im Spiel Vier Gewinnt verschieben sich weniger Platzierungen als vorher, nur 5 (38,5%) statt vorher 8. Auch verschieben sich die Platzierungen we-

niger stark, nur 2,4 Ränge (18,5%) statt der vorherigen 2,75. Daraus lässt sich erkennen, dass die Agenten im Spiel Hex stärker durch zufällige Starthalbzüge beeinflusst werden als die Agenten im Spiel Vier Gewinnt. Auch die Betrachtung der Boxplots zeigt, dass sich die Verteilung der einzelnen Spielstärken pro Turnier im Spiel Hex weniger konstant entwickeln im Vergleich zu denen in Vier Gewinnt.

7.3 M3 - Approximative Rankings

In der dritten Messreihe wurde untersucht, wie stark sich die Rankings der Agenten verändern, wenn die Anzahl der Matches schrittweise reduziert wird. Die Reduzierung betrug dabei 75%, 50%, 25%, 12% und 6% der Matches eines vollständigen Doppelrundenturniers, welches als 100% Referenz diente. Im Spiel Hex ist zu erkennen, dass sich die Rankings mit abnehmender Menge an Matches zunehmend verschlechtern. Bei 75% und 50% verschieben sich rund 60% der Platzierungen um rund 10% der Ränge. Danach steigert sich die Anzahl der Verschiebungen bis zuletzt auf rund 82%. Da sich ab 25% 13 und mehr Verschiebungen um mehr als 4 Platzierungen in den 17 Agenten messen lassen, ist hier keine wirkliche Ähnlichkeit mehr zum ursprünglichen Ranking zu erkennen. Im Bereich von 75% und 50% liegen mit 10 und 11 Verschiebungen auch viele Veränderungen vor, aber mit Rangverschiebungen von rund 2 Platzierungen fallen die Veränderungen hier weniger stark aus. Turniere im Spiel Hex können also in der Anzahl der Matches reduziert werden, um eine Annäherung an das finale volle Ranking zu bekommen. Diese Reduzierung darf für eine aussagekräftige Platzierung nicht zu stark ausfallen.
Im Spiel Vier Gewinnt lässt sich dagegen ein sehr konstanter Trend erkennen. Bei 75% bis 12% werden fast konstant 8 Vertauschungen von Platzierungen gemessen (61%), die sich zwischen 1,5 und 2,0 verschobenen Platzierungen bewegen. Erst bei der letzten Messung mit 6% steigt die Menge der Vertauschungen auf 10. Bei diesem Spiel können also weit weniger Spiele gespielt werden, allerdings liegt die Menge der vertauschten Platzierungen mit mehr als 60% sehr hoch. Da die Anzahl der verschobenen Ränge bei nur ca. 2 liegt, kann ein solches Turnier mit reduzierter Anzahl an Matches in Vier Gewinnt als sehr grobe Interpolation für das finale Ergebnis genutzt werden. Es kann aber für beide Spiele geschlossen werden, dass eine Reduzierung der Matchanzahl verschieden starke Auswirkung auf das Ranking hat. Mit den gemessenen Ergebnissen ist es möglich, Vorhersagen mithilfe der Glicko2 Ratings auf die zu erwartenden Endergebnisse zu treffen. Die Anzahl der Matches sollte aber nicht zu stark reduziert werden.

Die für beide Spiele ebenfalls beispielhaft eingebrachten Tabellen mit Glickoratings der besten drei Agenten veranschaulichen, dass die Ratings zwar schwanken, diese Schwankung aber nicht sehr stark ausfällt. Die Platzierung der Agenten in den Rankings verschiebt sich, wenn kein vollständiges Doppelrundenturnier mehr durchgeführt wird, aber die Glickoratings der Agenten fällt dennoch relativ konstant aus innerhalb der Turniere.
In der Arbeit [SPLR16] untersuchen die Autoren die Eignung der Algorithmen Bayesian-Elo und Glicko zur Vorhersage von Rankings eines Turniers. Der verwendete Bayesian-Elo ist eine weiterentwickelte Version des in dieser Arbeit verwendeten ursprünglichen Elos. Die Autoren untersuchen, wie gut die beiden Wertungssysteme die Rankings von Agenten in einem Pac-Man vs. Ghost Turnier bestimmen können. Beide Algorithmen produzieren in diesen Messungen gut Ergebnisse, aber auch hier schneidet der Glicko besser ab und produziert zuverlässigere Ergebnisse. Der Bayesian-Elo erreicht bessere Ergebnisse, als der hier verwendete ursprüngliche Elo. Dies liegt wahrscheinlich daran, dass der ursprüngliche Elo keine Aussagen über die Zuverlässigkeit eines Ratings treffen kann, wofür der Bayesian-Elo Wahrscheinlichkeitsverteilungen in seine Berechnungen einfließen lässt. Die Autoren kommen, genau wie diese Arbeit, zu dem Schluss, dass Glicko für die Bestimmung von Spielerstärke und Approximation von Rankings in KI-Turniere sehr gut geeignet ist. Der Glicko Algorithmus konnte sich in vollständigen und auch unvollständig durchgeführten Turnieren behaupten und die Spielstärke von Agenten zuverlässig bestimmen.

7.4 M4 - Reproduzierbarkeit

In der vierten Messreihe wurde untersucht, wie reproduzierbar die Rankingergebnisse der Agenten in einer Messung unter gleichen Bedingungen ausfallen. Eine Reproduzierbarkeit ist wichtig, da dies zeigt, dass die Agenten nicht einmal zufällig gut oder schlecht gespielt haben, sondern eine konstante Spielstärke besitzen. Die Ergebnisse der Messungen zeigen, dass die Agenten eine konstante Spielstärke besitzen. Es ist eine geringe Schwankung in den Rankings der Agenten zu erkennen, die Messwerte in Tabelle 19 im Anhang zeigen, dass diese sehr gering ausfallen. Auch die gering ausfallende Standardabweichung der Rankings in den Wiederholungen der Turniere bestätigt dies. Diese Schwankungen können sich natürlich erhöhen, wenn ein Turnier mit zufälligen Startzügen durchgeführt wird und nichtdeterministische Agenten untersucht werden. Für jedes Turnier muss dann eine Untersuchung

der Konsistenz der Spielstärke durchgeführt werden, an deren Ende die Aussage über die Konsistenz der Spielstärke eines Agenten eine Aussage über seine Qualität ermöglicht. Dies wäre nicht möglich ohne die Erkenntnis dieser Messreihe, dass eine grundsätzliche Reproduzierbarkeit gegeben ist.

7.5 M5 - Grenzen der Zeitmessung in Java

Die fünfte Messreihe untersucht die Grenzen der Zeitmessung in Java. Mithilfe dieser Ergebnisse können Aussagen getroffen werden, wie zuverlässig die Messwerte des Turniersystems sind. Viele der Züge erfolgen sehr schnell, speziell bei trivialen Spielen wie Tic Tac Toe. Deshalb ist es wichtig, die Grenzen der Präzision des Turniersystems zu untersuchen. Die Messergebnisse folgen den Ergebnissen aus der Literatur, dass die minimal messbare Präzision in Java auch von der Stärke des verwendeten Computers und vom Betriebssystem abhängt. Das Testsystem erreicht die Grenze der Präzision bei rund 33 Mikrosekunden unter Windows und bei rund 31 Mikrosekunden unter Ubuntu. Das ältere und langsamere ThinkPad erreicht seine Grenze der Präzision bei rund 237 Mikrosekunden und ist damit wesentlich langsamer. Die zu beobachtende scheinbare Steigerung der Präzision in den Box-Plots anhand sinkender vertikaler Ausdehnung der Boxen ist auf den ersten Blick widersprüchlich. Eigentlich wäre zu erwarten, dass die Präzision bei längeren Messungen höher ist und dann ab einem gewissen Punkt abnehmen würde. Zusätzlich wurde die Entwicklung des relativen Fehlers der Messung betrachtet. Damit war es möglich zu bestimmen, ab wann die gemessenen Zeitintervalle zu kurz für eine präzise Messung wurden. Bei genauerer Betrachtung der einzelnen Messwerte unter Windows fällt auf, dass in den letzten kurzen Messungen wenige diskrete Messwerte immer wieder auftreten, dazwischen aber keine wirklich weiteren Werte auftreten. Im Fall des Surface Books sind hier ganzzahlige Vielfache von 485 Nanosekunden zu erkennen. Dieses Verhalten kann durch die Implementierung der Zeitmessung in Java unter Windows verursacht werden, da dieses Phänomen unter Ubuntu nicht beobachtet werden kann. Die Messergebnisse dieser Reihen zeigen, dass die Wahl des Computers und Betriebssystems Auswirkungen auf die Präzision der Messergebnisse hat. Somit wäre es erforderlich, vor der Nutzung auf einem neuen Computer eine Art Benchmark durchzuführen, um die Präzision des Computers zu bestimmen. Dieses Ergebnis muss dann durch den Nutzer bei der Interpretation der Messergebnisse beachtet werden. Diese Grenze beeinflusst allerdings nicht jede Messung, da speziell die komplexeren Spiele oft längere Berechnungen besitzen und

die hohe Präzision nur bei sehr kurzen Zeiträumen zum tragen kommt.

8 Zusammenfassung und Ausblick

In dieser Arbeit wurde die Implementierung und Untersuchung eines Turniersystems für KI Agenten in Brettspielen betrachtet. Das Turniersystem ist eine Erweiterung des GBG-Frameworks und nutzt die darin implementierten Brettspiele und Agenten. Das Turniersystem besitzt verschiedene Funktionen, um Agenten unter verschiedenen Bedingungen zu testen und deren Spielstärke zu messen. Die Messergebnisse werden dem Nutzer nach dem Turnier in übersichtlicher Weise präsentiert. Mithilfe der einstellbaren zufälligen Startzüge und verschiedener Turniermodi kann der Agent vielseitig evaluiert werden. Aufgrund der wenigen verfügbaren Literaturquellen zu Turniersystemen im KI Bereich gab es wenig Orientierungen bei der Planung und Konzeptionierung der Software.
Die Messreihen ergaben wertvolle Erkenntnisse zum Verhalten der Agenten in verschiedenen Turniersituationen und auch den Limitationen des Turniersystems. Die in Kapitel 4.1 gestellten Anforderungen wurden erfüllt, somit stellt das vorgestellte Turniersystem eine wertvolle Erweiterung des GBG-Frameworks dar. Die Abhängigkeit der Messwertpräzision von dem verwendeten Computer und Betriebssystem ist eine wichtige Erkenntnis und muss von Nutzern beachtet werden. Eine Implementierung eines einfachen Benchmarks zur Bestimmung der zeitlichen Präzision wäre eine wertvolle Ergänzung des vorhandenen Funktionsumfanges.
Insgesamt stellt das vorgestellte Turniersystem ein vielseitiges Werkzeug zur Untersuchung von Agenten in Brettspielen dar und ermöglicht umfangreiche Untersuchungen. Die graphische Oberfläche besitzt noch Potential für Verbesserungen und Erweiterungen, speziell für kleine Displays und auch für Displays mit sehr hoher Auflösung. Hier sind die graphischen Elemente noch recht statisch und skalieren kaum. Weiterhin bietet die Implementierung des Turniers noch Spielraum zur Verbesserung. Bei der Implementierung wurde, aufgrund der Komplexität des Projekts, noch keine Optimierung der Effizienz durchgeführt. In diesem Kontext sollte speziell die hohe Arbeitsspeicherauslastung im laufenden Turnier weitergehend analysiert werden. Die sinnvolle Nutzung des Garbage Collectors und weitere Analysen der Ursache sollten vorgenommen werden, damit auf schwächeren Systemen oder bei großen Turnieren keine Beeinträchtigungen auftreten. Während der Implementierung und Nutzung konnten auf dem Testsystem keine negativen Effekte durch die hohen Arbeitsspeicherauslastung festgestellt werden.
Eine zusätzliche Erweiterung könnte ein neuer dynamischer Turniermodus sein, in

dem nicht die Anzahl der Matches zufällig reduziert wird, sondern anhand der temporären Glicko-Ratings nach einigen Matches. So kann ausgenutzt werden, dass Glicko das Rating eines Agenten, mit reduzierter Match-Anzahl, recht zuverlässig prognostizieren kann. Ab einer gewissen Menge an Matches könnten so die schlechtesten Agenten vorzeitig disqualifiziert werden. So könnte eine hohe Turnierlaufzeit geschickt verkürzt werden. Darüber hinaus sollte auch der Turniermodus der Heatchart Bibliothek erweitert werden. In ihrer jetzigen Implementierung sind die Einfärbungen abhängig vom niedrigsten und höchsten Wert der darzustellenden Daten. Die Heatmaps von zwei verschiedene Turnierdatensätze sind also aktuell nicht vergleichbar. Dies könnte behoben werden, indem eine feste obere und untere Grenze implementiert wird, in der sich die zu visualisierenden Werte befinden können. Dadurch würden die gleichen Werte immer mit dem gleichen Grauwert dargestellt werden. Da aus Zeitgründen in dieser Arbeit die Messreihen im Spiel Hex nur mit einer Spielfeldgröße von 4x4 durchgeführt werden, konnten auf diesem verhältnismäßig kleinen Feld manche Agenten ihr volles Potential nicht entfalten, da Matches schnell beendet werden können. Turniere und Messreihen mit größeren Spielfeldern müssten durchgeführt werden, um zu untersuchen, welche Agenten mit welchen Parametern auf diesen Feldern siegreich sind. Ebenfalls hat die Vorbereitung und die Dauer des Trainings der Agenten bisher keinen Einfluss auf die Turnierergebnisse, da die Agenten zu diesem Zeitpunkt keine Informationen dazu speichern können. Wenn ein neuer Agent erstellt und trainiert wurde, sollte die Dauer des Trainings mit im Agenten abgespeichert werden. Diese Dauer kann dann als weitere Metrik mit in die Auswertung der Turnierergebnisse einfließen, wenn eine dementsprechende Erweiterung des GBG-Frameworks vorgenommen worden ist.

Literaturverzeichnis

[AAA] AAAI: *AAAI Conference on Artificial Intelligence*. https://aaai.org/Conferences/AAAI/aaai.php, Abruf: 13.11.2018

[AAA18] AAAI: *Association for the Advancement of Artificial Intelligence*. https://www.aaai.org/home.html. Version: 27.09.2018, Abruf: 13.11.2018

[BDH+] Berry, Jonathan ; Dobre, Claudiu ; Howard, Robert ; Petronic, Jovan ; Weeks, Mark: *History of Elo ratings 1971-2001*. http://www.olimpbase.org/Elo/summary.html, Abruf: 11.10.2018

[BTKK16] Bagheri, Samineh ; Thill, Markus ; Koch, Patrick ; Konen, Wolfgang: Online Adaptable Learning Rates for the Game Connect-4. In: *IEEE Transactions on Computational Intelligence and AI in Games* 8 (2016), Nr. 1, S. 33–42. http://dx.doi.org/10.1109/TCIAIG.2014.2367105. – DOI 10.1109/TCIAIG.2014.2367105

[Cas10] Castle, Tom: *Create Java Heat Maps | JHeatChart*. http://www.javaheatmap.com/. Version: 2010, Abruf: 13.11.2018

[Che] ChessBase GmbH: *Nigel Short on being number one in Britain again*. https://en.chessbase.com/post/nigel-short-on-being-number-one-in-britain-again, Abruf: 13.11.2018

[Dud18a] Duden: *Duden | Brettspiel | Rechtschreibung, Bedeutung, Definition*. https://www.duden.de/rechtschreibung/Brettspiel. Version: 2018, Abruf: 06.07.2018

[Dud18b] Duden: *Duden | Determinismus | Rechtschreibung, Bedeutung, Definition*. https://www.duden.de/rechtschreibung/Determinismus. Version: 2018, Abruf: 06.07.2018

[Elo61] Elo, Arpad E.: New USCF rating system. In: *Chess Life* 1961 (31.12.1961), Nr. 16, S. 160–161

[Elo78] Elo, Arpad E.: *The rating of chessplayers, past and present*. New York : Arco Pub, 1978

[FID18] FIDE: *Chess Ratings Standard Top 100 Players July 2018*. https://ratings.fide.com/top.phtml?list=men. Version: 2018, Abruf: 06.07.2018

[Gal17] Galitzki, Kevin: *Selbstlernende Agenten für das skalierbare Spiel Hex: Untersuchung verschiedener KI-Verfahren im GBG-Framework*. Gummersbach, TH Köln, Bachelorarbeit, 2017. http://www.gm.fh-koeln.de/~konen/research/PaperPDF/BA-KevinGalitzki-final-2017.pdf

[GCS+18] Gaina, Raluca D. ; Couetoux, Adrien ; Soemers, Dennis J. N. J. ; Winands, Mark H. M. ; Vodopivec, Tom ; Kirchgesner, Florian ; Liu, Jialin ; Lucas, Simon M. ; Perez-Liebana, Diego: The 2016 Two-Player GVGAI Competition. In: *IEEE Transactions on Games* 10 (2018), Nr. 2, S. 209–220. http://dx.doi.org/10.1109/TCIAIG.2017.2771241. – DOI 10.1109/TCIAIG.2017.2771241. – ISSN 2475–1502

[Gen18] Genesereth, Michael: *IGGPC - Overview*. http://logic.stanford.edu/ggp/iggpc/index.php. Version: 2018, Abruf: 13.11.2018

[Gli99] Glickman, Mark E.: Parameter estimation in large dynamic paired comparison experiments. In: *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 48 (1999), Nr. 3, S. 377–394

[Gli12] Glickman, Mark E.: *Example of the Glicko-2 system*. http://www.glicko.net/glicko/glicko2.pdf. Version: 2012, Abruf: 9.10.2018

[Gli13] Glickman, Mark E.: *Welcome to Glicko ratings*. http://www.glicko.net/glicko.html. Version: 2013, Abruf: 06.07.2018

[GLP05] Genesereth, Michael ; Love, Nathaniel ; Pell, Barney: General game playing: Overview of the AAAI competition. In: *AI magazine* 26 (2005), Nr. 2, S. 62

[Goo13] Gooch, Jeremy: *Java implementation of the Glicko-2 rating algorithm: OSGi-compatible Java implementation of the Glicko-2 rating algorithm*. https://github.com/goochjs/glicko2. Version: 2013, Abruf: 13.11.2018

[GPLL16] Gaina, Raluca D. ; Perez-Liebana, Diego ; Lucas, Simon M.: General Video Game for 2 players: Framework and competition. In: *2016 8th Computer Science and Electronic Engineering Conference (CEEC)*. Piscataway, NJ : IEEE, 2016. – ISBN 978–1–5090–2050–8, S. 186–191

[HG06] Herbrich, Ralf ; Graepel, Thore: *TrueSkill(TM): A Bayesian Skill Rating System*. https://www.microsoft.com/en-us/research/publication/trueskilltm-a-bayesian-skill-rating-system-2/. Version: 2006

[IBM12] IBM: *Deep Blue*. http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/. Version: 2012, Abruf: 15.07.2018

[Kon17] Konen, Wolfgang: *The GBG Class Interface Tutorial: General Board Game Playing and Learning*. https://github.com/WolfgangKonen/GBG/blob/master/resources/TR-GBG.pdf. Version: 2017, Abruf: 06.07.2018

[Kon18a] Konen, Wolfgang: *General Board Game Playing*. https://github.com/WolfgangKonen/GBG. Version: 2018, Abruf: 15.07.2018

[Kon18b] Konen, Wolfgang: General Board Game Playing as Educational Tool for AI Competition and Learning. (2018). http://www.gm.fh-koeln.de/ciopwebpub/Kone18a.d/ToG-GBG.pdf

[LHH+08] Love, Nathaniel ; Hinrichs, Timothy ; Haley, David ; Schkufza, Eric ; Genesereth, Michael: General game playing: Game description language specification. (2008)

[Nos13] Nosinski, Mariusz: *marioosh / java-elo / source / — Bitbucket*. https://bitbucket.org/marioosh/java-elo/src/master/. Version: 2013, Abruf: 13.11.2018

[PLST+16] Perez-Liebana, Diego ; Samothrakis, Spyridon ; Togelius, Julian ; Schaul, Tom ; Lucas, Simon M. ; Couetoux, Adrien ; Lee, Jerry ; Lim, Chong-U ; Thompson, Tommy: The 2014 General Video Game Playing Competition. In: *IEEE Transactions on Computational Intelligence and AI in Games* 8 (2016), Nr. 3, S. 229–243. http://dx.doi.org/10.1109/TCIAIG.2015.2402393. – DOI 10.1109/TCIAIG.2015.2402393

[Shi14] SHIPILEV, Aleksey: *Nanotrusting the Nanotime*. https://shipilev.net/blog/2014/nanotrusting-nanotime/. Version: 2014, Abruf: 13.11.2018

[SHS+] SILVER, David ; HUBERT, Thomas ; SCHRITTWIESER, Julian ; ANTONOGLOU, Ioannis ; LAI, Matthew ; GUEZ, Arthur ; LANCTOT, Marc ; SIFRE, Laurent ; KUMARAN, Dharshan ; GRAEPEL, Thore ; LILLICRAP, Timothy ; SIMONYAN, Karen ; HASSABIS, Demis: *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*. http://arxiv.org/pdf/1712.01815v1

[SPLR16] SAMOTHRAKIS, Spyridon ; PEREZ, Diego ; LUCAS, Simon M. ; ROHLFSHAGEN, Philipp: Predicting Dominance Rankings for Score-Based Games. In: *IEEE Transactions on Computational Intelligence and AI in Games* 8 (2016), Nr. 1, S. 1–12. http://dx.doi.org/10.1109/TCIAIG.2014.2346242. – DOI 10.1109/TCIAIG.2014.2346242

[Thi12] THILL, Markus: *Reinforcement Learning mit N-Tupel-Systemen für Vier Gewinnt*. Gummersbach, TH Köln, Bachelorarbeit, 2012. http://www.gm.fh-koeln.de/ciopwebpub/Theses.d/Thill12.d/BA-Thill-2012.pdf

[Uni] UNITED STATES CHESS FEDERATION: *CC Ratings Explanation*. http://www.uschess.org/content/view/7520/393/, Abruf: 26.11.2018

[Web18] WEBER, Michael: *Die Geschichte der Brettspiele - Brettspiel | brettspiel.net*. https://www.brettspiel.net/die-geschichte-der-brettspiele.html. Version: 2018, Abruf: 06.07.2018

Anhang

Tabellen zu Kapitel 6.1

Agentname	Dateiname	mittlere WTL	SD WTL
Agent #0	TDNTuple2_3P-3ply	377,3	17,47
Agent #1	TDNTuple2_3P-MODE2	368,6	21,54
Agent #2	maxn-TD04-hashed	133,6	67,03
Agent #3	maxn-TD06-hashed	284,3	90,53
Agent #4	maxn-TD08-hashed	404,6	4,16
Agent #5	maxn-TD10-hashed	406,6	18,14
Agent #6	mcn-IT1000-RD20-NA1	343,6	18,14
Agent #7	mcn-IT2000-RD10-NA1	342,6	16,28
Agent #8	mcn-IT2000-RD20-NA1	342,0	27,73
Agent #9	mcn-IT2000-RD40-NA1	342,0	19,92
Agent #10	mcts-IT2000-TD10-RD100-V0	361,3	17,61
Agent #11	mcts-IT2000-TD10-RD300-V0	367,3	20,79
Agent #12	mcts-IT2000-TD40-RD200-V0	365,0	17,52
Agent #13	mcts-IT2000-TD40-RD300-V0	366,6	9,07
Agent #14	random	31,6	1,15
Agent #15	tdn2-RM-PWn-NumT20-TS-12-SymY	366,0	44,30
Agent #16	tdn2-RM-PWn-NumT20-TS-6-SymY	358,6	10,01

Tabelle 10: Turnieragenten des Spiels Hex mit Agentennamen in den Messreihen, Dateinamen, mittlerer WTL aus den drei Auswahlturnieren und die Standardabweichung (SD) der WTL

Erklärungen der Abkürzungen der selbst erstellten Agenten in Tabelle 10 und 11:

- **TD** - Tree Depth
- **hashed** - Nutzung einer Hashmap
- **IT** - Iterations
- **RD** - Rollout Depth
- **NA** - Number Agents

- **V** - Verbosity (MCTS)

Agentname	Dateiname	mittlere WTL	SD WTL
Agent #0	TCL-EXP-al20-lam05-500k-HOR001	511,33	42,29
Agent #1	TCL-EXP-al20-lam05-500k-HOR010-T	503,33	49,97
Agent #2	TDNT-al10-lam03-500k	34,50	5,50
Agent #3	mcn-IT1000-RD30-NA1	296,17	18,10
Agent #4	mcn-IT2000-RD30-NA1	306,83	24,97
Agent #5	mcn-IT2000-RD40-NA1	311,17	3,18
Agent #6	mcn-IT2000-RD50-NA1	309,67	27,02
Agent #7	mcts-IT2000-TD10-RD200-V0	506,00	23,44
Agent #8	mcts-IT2000-TD20-RD300-V0	510,17	3,21
Agent #9	mcts-IT2000-TD40-RD200-V0	512,33	7,09
Agent #10	mcts-IT2000-TD40-RD400-V0	510,33	13,90
Agent #11	random	4,33	0,58
Agent #12	tdn2-standard	62,17	13,53

Tabelle 11: Turnieragenten des Spiels Vier Gewinnt mit Agentennamen in den Messreihen, Dateinamen, mittlerer WTL aus den drei Auswahlturnieren und die Standardabweichung (SD) der WTL

Tabellen zu Kapitel 6.2

	A#0	A#1	A#2	A#3	A#4	A#5	A#6	A#7	A#8	A#9	A#10	A#11	A#12	A#13	A#14	A#15	A#16
A#0	NaN	1,00 (0,00)	1,00 (0,00)	0,90 (0,00)	0,00 (0,00)	0,00 (0,00)	0,02 (0,04)	0,00 (0,00)	0,04 (0,07)	0,01 (0,03)	0,04 (0,15)	0,07 (0,13)	-0,02 (0,09)	0,09 (0,21)	1,00 (0,00)	0,00 (0,00)	0,00 (0,00)
A#1	-1,00 (0,00)	NaN	0,60 (0,00)	0,70 (0,00)	-0,10 (0,00)	0,00 (0,00)	0,01 (0,07)	0,09 (0,06)	0,02 (0,04)	0,07 (0,07)	0,25 (0,13)	0,19 (0,10)	0,15 (0,14)	0,18 (0,16)	0,99 (0,03)	0,00 (0,00)	0,00 (0,00)
A#2	-1,00 (0,00)	-0,60 (0,00)	NaN	-0,61 (0,15)	-0,90 (0,00)	-0,90 (0,00)	-0,76 (0,14)	-0,82 (0,13)	-0,85 (0,13)	-0,86 (0,13)	-0,93 (0,08)	-0,89 (0,06)	-0,92 (0,06)	-0,92 (0,10)	0,90 (0,09)	-0,91 (0,07)	-0,85 (0,13)
A#3	-0,90 (0,00)	-0,70 (0,00)	0,61 (0,15)	NaN	-0,85 (0,14)	-0,76 (0,16)	-0,37 (0,21)	-0,18 (0,15)	-0,46 (0,20)	-0,40 (0,13)	-0,64 (0,12)	-0,63 (0,14)	-0,62 (0,15)	-0,54 (0,16)	0,87 (0,08)	-0,82 (0,12)	-0,66 (0,17)
A#4	0,00 (0,00)	0,10 (0,00)	0,90 (0,00)	0,85 (0,14)	NaN	0,00 (0,00)	0,28 (0,12)	0,30 (0,14)	0,15 (0,13)	0,19 (0,14)	0,29 (0,07)	0,25 (0,11)	0,35 (0,12)	0,22 (0,08)	0,98 (0,04)	0,00 (0,00)	0,00 (0,00)
A#5	0,00 (0,00)	0,00 (0,00)	0,90 (0,00)	0,76 (0,16)	0,00 (0,00)	NaN	0,07 (0,07)	0,02 (0,06)	0,01 (0,03)	0,02 (0,04)	0,15 (0,11)	0,17 (0,09)	0,14 (0,14)	0,23 (0,14)	0,99 (0,03)	0,00 (0,00)	0,16 (0,12)
A#6	-0,02 (0,04)	-0,01 (0,07)	0,76 (0,14)	0,37 (0,21)	-0,28 (0,12)	-0,07 (0,07)	NaN	-0,02 (0,04)	-0,01 (0,09)	-0,04 (0,08)	0,07 (0,11)	0,05 (0,15)	0,01 (0,15)	0,07 (0,08)	1,00 (0,00)	-0,01 (0,03)	-0,02 (0,04)
A#7	0,00 (0,00)	-0,09 (0,06)	0,82 (0,13)	0,18 (0,15)	-0,30 (0,14)	-0,02 (0,06)	0,02 (0,04)	NaN	0,00 (0,00)	0,00 (0,00)	0,16 (0,14)	0,06 (0,10)	0,14 (0,19)	0,12 (0,09)	1,00 (0,00)	-0,01 (0,03)	-0,01 (0,03)
A#8	-0,04 (0,07)	-0,02 (0,04)	0,85 (0,13)	0,46 (0,20)	-0,15 (0,13)	-0,01 (0,03)	0,01 (0,09)	0,00 (0,00)	NaN	0,00 (0,05)	0,15 (0,08)	0,12 (0,10)	0,12 (0,10)	0,11 (0,14)	1,00 (0,00)	0,00 (0,00)	-0,04 (0,07)
A#9	-0,01 (0,03)	-0,07 (0,07)	0,86 (0,13)	0,40 (0,13)	-0,19 (0,14)	-0,02 (0,04)	0,04 (0,08)	0,00 (0,00)	0,00 (0,05)	NaN	0,02 (0,08)	0,07 (0,13)	0,06 (0,17)	0,12 (0,16)	1,00 (0,00)	0,00 (0,00)	-0,01 (0,03)
A#10	-0,04 (0,15)	-0,25 (0,13)	0,93 (0,08)	0,64 (0,12)	-0,29 (0,07)	-0,15 (0,11)	-0,07 (0,11)	-0,16 (0,14)	-0,15 (0,08)	-0,02 (0,08)	NaN	0,00 (0,20)	0,04 (0,18)	0,06 (0,20)	1,00 (0,00)	-0,10 (0,09)	-0,17 (0,14)
A#11	-0,07 (0,13)	-0,19 (0,10)	0,89 (0,06)	0,63 (0,14)	-0,25 (0,11)	-0,17 (0,09)	-0,05 (0,15)	-0,06 (0,10)	-0,12 (0,10)	-0,07 (0,13)	0,00 (0,20)	NaN	0,04 (0,19)	0,04 (0,14)	0,99 (0,03)	-0,18 (0,12)	-0,25 (0,08)
A#12	0,02 (0,09)	-0,15 (0,14)	0,92 (0,06)	0,62 (0,15)	-0,35 (0,12)	-0,14 (0,14)	-0,01 (0,15)	-0,14 (0,19)	-0,12 (0,10)	-0,06 (0,17)	-0,04 (0,18)	-0,04 (0,19)	NaN	0,03 (0,13)	1,00 (0,00)	-0,12 (0,10)	-0,21 (0,17)
A#13	-0,09 (0,21)	-0,18 (0,16)	0,92 (0,10)	0,54 (0,16)	-0,22 (0,08)	-0,23 (0,14)	-0,07 (0,08)	-0,12 (0,09)	-0,11 (0,14)	-0,12 (0,16)	-0,06 (0,20)	-0,04 (0,14)	-0,03 (0,13)	NaN	1,00 (0,00)	-0,18 (0,09)	-0,23 (0,15)
A#14	-1,00 (0,00)	-0,99 (0,03)	-0,90 (0,09)	-0,87 (0,08)	-0,98 (0,04)	-0,99 (0,03)	-1,00 (0,00)	-1,00 (0,00)	-1,00 (0,00)	-1,00 (0,00)	-1,00 (0,00)	-0,99 (0,03)	-1,00 (0,00)	-1,00 (0,00)	NaN	-0,98 (0,04)	-1,00 (0,00)
A#15	0,00 (0,00)	0,00 (0,00)	0,91 (0,07)	0,82 (0,12)	0,00 (0,00)	0,00 (0,00)	0,01 (0,03)	0,01 (0,03)	0,00 (0,00)	0,00 (0,00)	0,10 (0,09)	0,18 (0,12)	0,12 (0,10)	0,18 (0,09)	0,98 (0,04)	NaN	0,00 (0,00)
A#16	0,00 (0,00)	0,00 (0,00)	0,85 (0,13)	0,66 (0,17)	0,00 (0,00)	-0,16 (0,12)	0,02 (0,04)	0,01 (0,03)	0,04 (0,07)	0,01 (0,03)	0,17 (0,14)	0,25 (0,08)	0,21 (0,17)	0,23 (0,15)	1,00 (0,00)	0,00 (0,00)	NaN

Tabelle 12: Daten der Heatmap Messreihe M1-$W_{ab} = W_{ba}$ Ergebnisse im Spiel Hex mit Standardabweichung in Klammern

	A#0	A#1	A#2	A#3	A#4	A#5	A#6	A#7	A#8	A#9	A#10	A#11	A#12	A#13	A#14	A#15	A#16
A#0	NaN	0,00 (0,00)	0,00 (0,00)	0,10 (0,00)	1,00 (0,00)	1,00 (0,00)	0,98 (0,04)	1,00 (0,00)	0,96 (0,07)	0,99 (0,03)	0,72 (0,16)	0,77 (0,17)	0,84 (0,16)	0,73 (0,21)	0,00 (0,00)	1,00 (0,00)	1,00 (0,00)
A#1	0,00 (0,00)	NaN	0,20 (0,00)	0,30 (0,00)	0,90 (0,00)	1,00 (0,00)	0,95 (0,05)	0,91 (0,06)	0,98 (0,04)	0,91 (0,07)	0,75 (0,13)	0,81 (0,10)	0,85 (0,14)	0,82 (0,16)	0,01 (0,03)	1,00 (0,00)	1,00 (0,00)
A#2	0,00 (0,00)	0,20 (0,00)	NaN	0,01 (0,15)	0,10 (0,00)	0,10 (0,00)	0,20 (0,12)	0,16 (0,16)	0,13 (0,12)	0,12 (0,09)	0,07 (0,08)	0,11 (0,06)	0,06 (0,08)	0,06 (0,07)	0,00 (0,11)	0,09 (0,07)	0,07 (0,12)
A#3	0,10 (0,00)	0,30 (0,00)	0,01 (0,15)	NaN	0,15 (0,14)	0,24 (0,16)	0,41 (0,14)	0,52 (0,21)	0,34 (0,18)	0,44 (0,14)	0,16 (0,12)	0,21 (0,21)	0,16 (0,20)	0,26 (0,13)	0,07 (0,11)	0,18 (0,12)	0,34 (0,17)
A#4	1,00 (0,00)	0,90 (0,00)	0,10 (0,00)	0,15 (0,14)	NaN	1,00 (0,00)	0,72 (0,12)	0,70 (0,14)	0,85 (0,13)	0,81 (0,14)	0,71 (0,07)	0,75 (0,11)	0,65 (0,12)	0,78 (0,08)	0,02 (0,04)	1,00 (0,00)	1,00 (0,00)
A#5	1,00 (0,00)	1,00 (0,00)	0,10 (0,00)	0,24 (0,16)	1,00 (0,00)	NaN	0,93 (0,07)	0,98 (0,06)	0,99 (0,03)	0,98 (0,04)	0,85 (0,11)	0,83 (0,09)	0,86 (0,14)	0,77 (0,14)	0,01 (0,03)	1,00 (0,00)	0,84 (0,12)
A#6	0,98 (0,04)	0,95 (0,05)	0,20 (0,12)	0,41 (0,14)	0,72 (0,12)	0,93 (0,07)	NaN	0,98 (0,04)	0,93 (0,08)	0,94 (0,07)	0,81 (0,11)	0,79 (0,11)	0,75 (0,13)	0,85 (0,07)	0,00 (0,00)	0,99 (0,03)	0,98 (0,04)
A#7	1,00 (0,00)	0,91 (0,06)	0,16 (0,16)	0,52 (0,21)	0,70 (0,14)	0,98 (0,06)	0,98 (0,04)	NaN	1,00 (0,00)	1,00 (0,00)	0,82 (0,11)	0,88 (0,10)	0,80 (0,23)	0,84 (0,10)	0,00 (0,00)	0,99 (0,03)	0,99 (0,03)
A#8	0,96 (0,07)	0,98 (0,04)	0,13 (0,12)	0,34 (0,18)	0,85 (0,13)	0,99 (0,03)	0,93 (0,08)	1,00 (0,00)	NaN	0,98 (0,04)	0,83 (0,12)	0,86 (0,12)	0,86 (0,10)	0,85 (0,13)	0,00 (0,00)	1,00 (0,00)	0,96 (0,07)
A#9	0,99 (0,03)	0,91 (0,07)	0,12 (0,09)	0,44 (0,14)	0,81 (0,14)	0,98 (0,04)	0,94 (0,07)	1,00 (0,00)	0,98 (0,04)	NaN	0,92 (0,09)	0,89 (0,09)	0,84 (0,12)	0,86 (0,16)	0,00 (0,00)	1,00 (0,00)	0,99 (0,03)
A#10	0,72 (0,16)	0,75 (0,13)	0,07 (0,08)	0,16 (0,12)	0,71 (0,07)	0,85 (0,11)	0,81 (0,11)	0,82 (0,11)	0,83 (0,12)	0,92 (0,09)	NaN	0,74 (0,19)	0,76 (0,14)	0,72 (0,11)	0,00 (0,00)	0,90 (0,09)	0,83 (0,14)
A#11	0,77 (0,17)	0,81 (0,10)	0,11 (0,06)	0,21 (0,21)	0,75 (0,11)	0,83 (0,09)	0,79 (0,11)	0,88 (0,10)	0,86 (0,12)	0,89 (0,09)	0,74 (0,19)	NaN	0,70 (0,19)	0,74 (0,20)	0,01 (0,03)	0,82 (0,12)	0,75 (0,08)
A#12	0,84 (0,16)	0,85 (0,14)	0,06 (0,08)	0,16 (0,20)	0,65 (0,12)	0,86 (0,14)	0,75 (0,13)	0,80 (0,23)	0,86 (0,10)	0,84 (0,12)	0,76 (0,14)	0,70 (0,19)	NaN	0,63 (0,16)	0,00 (0,00)	0,88 (0,10)	0,77 (0,19)
A#13	0,73 (0,21)	0,82 (0,16)	0,06 (0,07)	0,26 (0,13)	0,78 (0,08)	0,77 (0,14)	0,85 (0,07)	0,84 (0,10)	0,85 (0,13)	0,86 (0,16)	0,72 (0,11)	0,74 (0,20)	0,63 (0,16)	NaN	0,00 (0,00)	0,82 (0,09)	0,77 (0,15)
A#14	0,00 (0,00)	0,01 (0,03)	0,00 (0,11)	0,07 (0,11)	0,02 (0,04)	0,01 (0,03)	0,00 (0,00)	0,00 (0,00)	0,00 (0,00)	0,00 (0,00)	0,00 (0,00)	0,01 (0,03)	0,00 (0,00)	0,00 (0,00)	NaN	0,02 (0,04)	0,00 (0,00)
A#15	1,00 (0,00)	1,00 (0,00)	0,09 (0,07)	0,18 (0,12)	1,00 (0,00)	1,00 (0,00)	0,99 (0,03)	0,99 (0,03)	1,00 (0,00)	1,00 (0,00)	0,90 (0,09)	0,82 (0,12)	0,88 (0,10)	0,82 (0,09)	0,02 (0,04)	NaN	1,00 (0,00)
A#16	1,00 (0,00)	1,00 (0,00)	0,07 (0,12)	0,34 (0,17)	1,00 (0,00)	0,84 (0,12)	0,98 (0,04)	0,99 (0,03)	0,96 (0,07)	0,99 (0,03)	0,83 (0,14)	0,75 (0,08)	0,77 (0,19)	0,77 (0,15)	0,00 (0,00)	1,00 (0,00)	NaN

Tabelle 13: Daten der Heatmap Messreihe M1-$W_{ab} = 1 - W_{ba}$ Ergebnisse im Spiel Hex mit Standardabweichung in Klammern

	A#0	A#1	A#2	A#3	A#4	A#5	A#6	A#7	A#8	A#9	A#10	A#11	A#12
A#0	NaN	-1,00 (0,00)	1,00 (0,00)	0,57 (0,17)	0,27 (0,18)	0,39 (0,10)	0,39 (0,17)	0,28 (0,15)	0,47 (0,14)	0,32 (0,11)	0,37 (0,11)	1,00 (0,00)	1,00 (0,00)
A#1	1,00 (0,00)	NaN	1,00 (0,00)	0,73 (0,10)	0,80 (0,07)	0,82 (0,13)	0,81 (0,12)	0,10 (0,07)	0,12 (0,18)	0,21 (0,08)	0,14 (0,07)	1,00 (0,00)	1,00 (0,00)
A#2	-1,00 (0,00)	-1,00 (0,00)	NaN	-0,99 (0,03)	-0,98 (0,04)	-0,98 (0,04)	-0,97 (0,05)	-1,00 (0,00)	-1,00 (0,00)	-1,00 (0,00)	-1,00 (0,00)	0,82 (0,09)	-1,00 (0,00)
A#3	-0,57 (0,17)	-0,73 (0,10)	0,99 (0,03)	NaN	0,08 (0,17)	0,01 (0,25)	-0,11 (0,20)	-0,70 (0,05)	-0,59 (0,21)	-0,73 (0,13)	-0,70 (0,16)	1,00 (0,00)	0,99 (0,03)
A#4	-0,27 (0,18)	-0,80 (0,07)	0,98 (0,04)	-0,08 (0,17)	NaN	-0,07 (0,24)	-0,02 (0,18)	-0,64 (0,16)	-0,60 (0,19)	-0,54 (0,16)	-0,77 (0,13)	1,00 (0,00)	1,00 (0,00)
A#5	-0,39 (0,10)	-0,82 (0,13)	0,98 (0,04)	-0,01 (0,25)	0,07 (0,24)	NaN	-0,10 (0,19)	-0,66 (0,17)	-0,73 (0,05)	-0,64 (0,11)	-0,64 (0,13)	1,00 (0,00)	1,00 (0,00)
A#6	-0,39 (0,17)	-0,81 (0,12)	0,97 (0,05)	0,11 (0,20)	0,02 (0,18)	0,10 (0,19)	NaN	-0,66 (0,19)	-0,66 (0,23)	-0,68 (0,20)	-0,61 (0,19)	1,00 (0,00)	1,00 (0,00)
A#7	-0,28 (0,15)	-0,10 (0,07)	1,00 (0,00)	0,70 (0,05)	0,64 (0,16)	0,66 (0,17)	0,66 (0,19)	NaN	0,07 (0,24)	-0,03 (0,12)	0,03 (0,22)	1,00 (0,00)	0,99 (0,03)
A#8	-0,47 (0,14)	-0,12 (0,18)	1,00 (0,00)	0,59 (0,21)	0,60 (0,19)	0,73 (0,05)	0,66 (0,23)	-0,07 (0,24)	NaN	0,06 (0,20)	0,10 (0,22)	1,00 (0,00)	0,98 (0,04)
A#9	-0,32 (0,11)	-0,21 (0,08)	1,00 (0,00)	0,73 (0,13)	0,54 (0,16)	0,64 (0,11)	0,68 (0,20)	0,03 (0,12)	-0,06 (0,20)	NaN	-0,08 (0,24)	1,00 (0,00)	0,99 (0,03)
A#10	-0,37 (0,11)	-0,14 (0,07)	1,00 (0,00)	0,70 (0,16)	0,77 (0,13)	0,64 (0,13)	0,61 (0,19)	-0,03 (0,22)	-0,10 (0,22)	0,08 (0,24)	NaN	1,00 (0,00)	0,98 (0,04)
A#11	-1,00 (0,00)	-1,00 (0,00)	-0,82 (0,09)	-1,00 (0,00)	-1,00 (0,00)	-1,00 (0,00)	-1,00 (0,00)	-1,00 (0,00)	-1,00 (0,00)	-1,00 (0,00)	-1,00 (0,00)	NaN	-0,88 (0,11)
A#12	-1,00 (0,00)	-1,00 (0,00)	1,00 (0,00)	-0,99 (0,03)	-1,00 (0,00)	-1,00 (0,00)	-1,00 (0,00)	-0,99 (0,03)	-0,98 (0,04)	-0,99 (0,03)	-0,98 (0,04)	0,88 (0,11)	NaN

Tabelle 14: Daten der Heatmap Messreihe M1-$W_{ab} = W_{ba}$ Ergebnisse im Spiel Vier gewinnt mit Standardabweichung in Klammern

	A#0	A#1	A#2	A#3	A#4	A#5	A#6	A#7	A#8	A#9	A#10	A#11	A#12
A#0	NaN	0,00 (0,00)	0,00 (0,00)	-0,07 (0,19)	-0,25 (0,28)	-0,21 (0,15)	-0,26 (0,14)	0,12 (0,15)	0,33 (0,14)	0,08 (0,11)	0,24 (0,11)	0,00 (0,00)	0,00 (0,00)
A#1	0,00 (0,00)	NaN	0,00 (0,00)	0,19 (0,15)	0,11 (0,11)	0,12 (0,12)	0,14 (0,10)	0,89 (0,07)	0,78 (0,18)	0,79 (0,08)	0,86 (0,07)	0,00 (0,00)	0,00 (0,00)
A#2	0,00 (0,00)	0,00 (0,00)	NaN	0,01 (0,03)	0,02 (0,04)	0,02 (0,04)	0,03 (0,05)	0,00 (0,00)	0,00 (0,00)	0,00 (0,00)	0,00 (0,00)	0,18 (0,09)	0,00 (0,00)
A#3	-0,07 (0,19)	0,19 (0,15)	0,01 (0,03)	NaN	0,12 (0,29)	-0,15 (0,24)	-0,10 (0,32)	-0,03 (0,22)	0,03 (0,21)	0,03 (0,14)	0,02 (0,09)	0,00 (0,00)	-0,01 (0,03)
A#4	-0,25 (0,28)	0,11 (0,11)	0,02 (0,04)	0,12 (0,29)	NaN	-0,12 (0,19)	-0,17 (0,27)	0,03 (0,15)	-0,07 (0,16)	-0,01 (0,20)	-0,15 (0,14)	0,00 (0,00)	0,00 (0,00)
A#5	-0,21 (0,15)	0,12 (0,12)	0,02 (0,04)	-0,15 (0,24)	-0,12 (0,19)	NaN	-0,24 (0,14)	0,02 (0,23)	-0,01 (0,15)	0,07 (0,22)	-0,12 (0,15)	0,00 (0,00)	0,00 (0,00)
A#6	-0,26 (0,14)	0,14 (0,10)	0,03 (0,05)	-0,10 (0,32)	-0,17 (0,27)	-0,25 (0,14)	NaN	0,02 (0,18)	-0,05 (0,12)	-0,00 (0,09)	0,04 (0,21)	0,00 (0,00)	0,00 (0,00)
A#7	0,12 (0,15)	0,89 (0,07)	0,00 (0,00)	-0,03 (0,22)	0,03 (0,15)	0,02 (0,23)	0,02 (0,18)	NaN	0,12 (0,29)	0,09 (0,19)	0,26 (0,21)	0,00 (0,00)	-0,01 (0,03)
A#8	0,33 (0,14)	0,78 (0,18)	0,00 (0,00)	0,03 (0,21)	-0,07 (0,16)	-0,01 (0,15)	-0,05 (0,12)	0,12 (0,29)	NaN	0,08 (0,23)	0,19 (0,14)	0,00 (0,00)	-0,02 (0,04)
A#9	0,08 (0,11)	0,79 (0,08)	0,00 (0,00)	0,03 (0,14)	-0,01 (0,20)	0,07 (0,22)	-0,01 (0,09)	0,09 (0,19)	0,08 (0,23)	NaN	0,17 (0,24)	0,00 (0,00)	0,01 (0,03)
A#10	0,24 (0,11)	0,86 (0,07)	0,00 (0,00)	0,02 (0,09)	-0,16 (0,14)	-0,12 (0,15)	0,04 (0,21)	0,26 (0,21)	0,19 (0,14)	0,17 (0,24)	NaN	0,00 (0,00)	-0,02 (0,04)
A#11	0,00 (0,00)	0,00 (0,00)	0,18 (0,09)	0,00 (0,00)	0,00 (0,00)	0,00 (0,00)	0,00 (0,00)	0,00 (0,00)	0,00 (0,00)	0,00 (0,00)	0,00 (0,00)	NaN	-0,04 (0,16)
A#12	0,00 (0,00)	0,00 (0,00)	0,00 (0,00)	-0,01 (0,03)	0,00 (0,00)	0,00 (0,00)	0,00 (0,00)	-0,01 (0,03)	-0,02 (0,04)	0,01 (0,03)	-0,02 (0,04)	-0,04 (0,16)	NaN

Tabelle 15: Daten der Heatmap Messreihe M1-$W_{ab} = 1 - W_{ba}$ Ergebnisse im Spiel Vier gewinnt mit Standardabweichung in Klammern

Rang	Agent	Dateiname	∅ Glicko2	σ
1	Agent #4	maxn-TD08-hashed	1656,65	13,86
2	Agent #0	TDNTuple2_3P-3ply	1633,77	9,76
3	Agent #5	maxn-TD10-hashed	1616,68	9,47
4	Agent #15	tdn2-RM-PWn-NumT20-TS-12-SymY	1606,69	7,36
5	Agent #16	tdn2-RM-PWn-NumT20-TS-6-SymY	1606,05	14,45
6	Agent #8	mcn-IT2000-RD20-NA1	1582,52	16,38
7	Agent #9	mcn-IT2000-RD40-NA1	1573,17	16,30
8	Agent #1	TDNTuple2_3P-MODE2	1569,30	13,27
9	Agent #7	mcn-IT2000-RD10-NA1	1566,72	10,31
10	Agent #6	mcn-IT1000-RD20-NA1	1559,63	11,89
11	Agent #10	mcts-IT2000-TD10-RD100-V0	1540,94	21,00
12	Agent #12	mcts-IT2000-TD40-RD200-V0	1539,00	24,61
13	Agent #11	mcts-IT2000-TD10-RD300-V0	1538,04	13,40
14	Agent #13	mcts-IT2000-TD40-RD300-V0	1525,14	19,39
15	Agent #3	maxn-TD06-hashed	1272,75	13,78
16	Agent #2	maxn-TD04-hashed	1119,00	14,16
17	Agent #14	random	993,93	5,48

Tabelle 16: Messreihe M2-RSM0 Ergebnisse im Spiel Hex mit durchschnittlichem Glicko2 der Agenten und der Standardabweichung σ der Glicko2 Werte

Rang	Agent	Dateiname	∅ Glicko2	σ
1	Agent #5	maxn-TD10-hashed	1667,94	43,71
2	Agent #4	maxn-TD08-hashed	1652,79	32,28
3	Agent #15	tdn2-RM-PWn-NumT20-TS-12-SymY	1631,51	40,31
4	Agent #0	TDNTuple2_3P-3ply	1608,62	25,97
5	Agent #16	tdn2-RM-PWn-NumT20-TS-6-SymY	1573,81	8,38
6	Agent #13	mcts-IT2000-TD40-RD300-V0	1570,27	27,38
7	Agent #12	mcts-IT2000-TD40-RD200-V0	1561,57	27,45
8	Agent #1	TDNTuple2_3P-MODE2	1558,66	14,48
9	Agent #10	mcts-IT2000-TD10-RD100-V0	1555,76	15,87
10	Agent #11	mcts-IT2000-TD10-RD300-V0	1548,35	29,89
11	Agent #8	mcn-IT2000-RD20-NA1	1517,08	24,78
12	Agent #9	mcn-IT2000-RD40-NA1	1513,22	27,20
13	Agent #7	mcn-IT2000-RD10-NA1	1512,25	28,13
14	Agent #6	mcn-IT1000-RD20-NA1	1505,80	20,60
15	Agent #3	maxn-TD06-hashed	1374,61	48,68
16	Agent #2	maxn-TD04-hashed	1148,33	16,11
17	Agent #14	random	999,41	6,63

Tabelle 17: Messreihe M2-RSM1 Ergebnisse im Spiel Hex mit durchschnittlichem Glicko2 der Agenten und der Standardabweichung σ der Glicko2 Werte

Rang	Agent	Dateiname	∅ Glicko2	σ
1	Agent #4	maxn-TD08-hashed	1598,63	21,92
2	Agent #5	maxn-TD10-hashed	1583,81	22,12
3	Agent #11	mcts-IT2000-TD10-RD300-V0	1575,10	21,11
4	Agent #12	mcts-IT2000-TD40-RD200-V0	1567,05	18,47
5	Agent #13	mcts-IT2000-TD40-RD300-V0	1565,11	24,21
6	Agent #0	TDNTuple2_3P-3ply	1564,79	23,36
7	Agent #3	maxn-TD06-hashed	1564,47	29,34
8	Agent #10	mcts-IT2000-TD10-RD100-V0	1556,73	18,56
9	Agent #1	TDNTuple2_3P-MODE2	1526,11	22,66
10	Agent #15	tdn2-RM-PWn-NumT20-TS-12-SymY	1524,50	33,26
11	Agent #8	mcn-IT2000-RD20-NA1	1512,25	29,38
12	Agent #9	mcn-IT2000-RD40-NA1	1509,67	30,99
13	Agent #16	tdn2-RM-PWn-NumT20-TS-6-SymY	1506,77	31,34
14	Agent #6	mcn-IT1000-RD20-NA1	1501,29	25,48
15	Agent #7	mcn-IT2000-RD10-NA1	1500,00	30,24
16	Agent #2	maxn-TD04-hashed	1341,41	67,70
17	Agent #14	random	1002,31	7,16

Tabelle 18: Messreihe M2-RSM2 Ergebnisse im Spiel Hex mit durchschnittlichem Glicko2 der Agenten und der Standardabweichung σ der Glicko2 Werte

Spiel	Hex			Vier Gewinnt		
Top 3 Agenten	Agent #4	Agent #0	Agent #5	Agent #1	Agent #7	Agent #10
Glicko2 Werte	1661,17	1641,83	1599,92	1877,35	1748,71	1737,99
der zehn	1645,05	1625,71	1616,04	1823,75	1697,25	1731,55
Turniere	1670,84	1638,60	1603,15	1830,18	1720,83	1727,27
	1651,50	1641,83	1622,49	1815,17	1746,56	1716,55
	1680,51	1619,26	1625,71	1834,47	1740,13	1671,52
	1628,93	1651,50	1619,26	1817,32	1725,12	1718,69
	1657,94	1625,71	1619,26	1823,75	1742,28	1716,55
	1657,94	1628,93	1625,71	1832,32	1742,28	1735,84
	1654,72	1628,93	1625,71	1836,61	1697,25	1725,12
	1657,94	1635,38	1609,59	1823,75	1727,27	1725,12
SD	13,86	9,76	9,46	17,59	19,07	18,81

Tabelle 19: Glicko2 Werte und Standardabweichung SD der besten Agenten im Spiel Hex und Vier Gewinnt in der Messreihe M4

Eidesstattliche Erklärung

Ich versichere hiermit, die vorgelegte Arbeit in dem gemeldeten Zeitraum ohne fremde Hilfe verfasst und mich keiner anderen als der angegebenen Hilfsmittel und Quellen bedient zu haben.

Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Köln, den 15. April 2019

Unterschrift
(Felix Barsnick)