SpookyJS

Ein multiagentenbasiertes JavaScript-Framework zur flexiblen Implementation digitaler browserbasierter Brettspiele und spielübergreifender künstlicher Intelligenz

Inauguraldissertation

zur Erlangung des Doktorgrades

der Philosophischen Fakultät der Universität zu Köln

im Fach Historisch-Kulturwissenschaftliche Informationsverarbeitung

Vorgelegt von Jan Gerrit Wieners.

Datum der Disputation: 14.01.2015

1. Referent: Prof. Dr. Manfred Thaller

2. Referent: Prof. Dr. Reinhard Förtsch

Vergleiche: *wissen* und *sagen*:

wieviele m hoch der Mont-Blanc ist –
wie das Wort „Spiel“ gebraucht wird –
wie eine Klarinette klingt.

Wer sich wundert, daß man etwas wissen könne, und nicht sagen, denkt vielleicht an einen Fall wie den ersten. Gewiß nicht an einen wie den dritten.

- *Wittgenstein: Philosophische Untersuchungen, §7*

Kurzdarstellung

Künstliche Intelligenz in digitalen Spielen ist zumeist Anwendungsdomäne komplexer spielspezifischer Softwarelösungen mangelnder Erweiterbarkeit. Die vorliegende Arbeit beschäftigt sich mit der Konzeption und Realisierung des JavaScript-Frameworks *SpoookyJS*, das die vereinfachte Erstellung browserbasierter digitaler Brettspiele ermöglicht. Entwickelt als Multiagentensystem, bietet *SpoookyJS* artifizielle Gegner in den umgesetzten Spielen und fungiert als Test- und Entwicklungsumgebung für die Forschung um spielübergreifende artifizielle Entscheidungsfindung.

Schlüsselwörter: Digitale Brettspiele, Künstliche Intelligenz, Computational Intelligence, General Game Playing, Monte Carlo Spielbaumsuche.

Inhaltsverzeichnis

1 Einleitung 14

1.1 Fragestellung und Ziel der Arbeit 15

1.2 Aufbau und Verlauf der Arbeit 18

2 Von einer ehrenhaften Täuschung – Zum Gegenstand der künstlichen Intelligenz 21

2.1 Über den Schachspieler des Herrn von Kempelen 23

2.2 Fokussierungsweisen künstlicher Intelligenz 27

2.2.1 Menschliche vs. nichtmenschliche Intelligenz – Der Turing-Test 30

2.2.2 Schwache vs. starke künstliche Intelligenz 34

2.2.3 Künstliche Intelligenz und das Konzept der Rationalität 36

2.3 Intelligente Agenten 38

2.3.1 Agentenfunktion und Agentenautonomie 41

2.3.2 Agentenumwelt 42

2.3.3 Brettspielumgebungen 45

2.3.4 Aufbau und Struktur von Agenten 46

2.3.5 Lernvermögen 48

2.4 Brettspiele, Drosophila künstlicher Intelligenz 51

2.5 Methodendifferenzierung – Künstliche Intelligenz und Computational Intelligence 54

2.6 Zusammenfassung und Ausblick 58

3 Von Huizinga nach Γ – Theorien des Spieles 59

3.1 Spielarten von Spieltheorien 61

3.2 Huizingas Homo Ludens 64

3.3 Juuls klassisches Spielmodell 66

3.3.1 Spielregeln 68

3.3.2 Spiel- und Zugregelatome 69

3.3.3 Zusammenfassung und Ausblick 70

3.4 Spieltheorie nach von Neumann und Morgenstern 72

3.4.1 Spiel und Spielpartie, Zug und Zugwahl 73

3.4.2 Das Spiel Γ in Normalform 75

3.4.3 Das Gefangenendilemma 78

3.4.4 Dominante Strategien und das Nash-Gleichgewicht 79

3.4.5 Kooperatives und egoistisches Agieren 81

3.4.6 Strategien und Entscheidungen 82

3.4.7 Wie du mir, so ich dir: das wiederholte Gefangenendilemma 84

3.4.8 Nullsummenspiele 86

3.4.9 Sequentielle Entscheidungen: Spielbaum und Extensivform 87

3.5 Zusammenfassung und Ausblick 93

4 Artifizielles Spielvermögen in digitalen Brettspielen – Methoden und Implementationen 94

4.1 Künstliche Intelligenz als Suche nach optimalen Spielentscheidungen 95

4.1.1 Problemlösen durch Suchen – Elementare Suchverfahren 96

4.1.1.1 Breiten- und Tiefensuche 98

4.1.1.2 Iterativ vertiefende Tiefensuche 99

4.1.2 Minimax-Algorithmus 101

4.1.3 Negamax-Algorithmus 106

4.1.4 Alpha-Beta Kürzung 107

4.1.5 Berücksichtigung dediziertes Spielwissens – die Heuristische Evaluierungsfunktion 114

4.1.6 Optimierungen und Ergänzungen der Alpha-Beta Kürzung 117

4.1.6.1 Zugvorsortierung 118

4.1.6.2 Killerzüge und Killerheuristik 119

4.1.6.3 Nullzug-Suche 119

4.1.6.4 Ruhesuche 121

4.1.6.5 Zugumstellungstabellen 122

4.1.7 State of the Art: Ausgewählte KI-Implementationen 124

4.1.7.1 Perfektes Damespiel: Chinook 124

4.1.7.2 Der erste nichtmenschliche Schachweltmeister: Deep Blue ... 127

4.1.7.3 Meisterliches Schachspiel mit beschränkten Ressourcen: Stockfish 129

4.1.8 Zusammenfassung und Ausblick 130

4.2 Künstliche Intelligenz und das Moment des Zufalls – Monte Carlo Verfahren 132

4.2.1 Monte Carlo Simulation 134

4.2.2 Monte Carlo Spielbaumsuche 135

4.2.3 Mehrarmige Banditen, UCB und UCT 140

4.2.4 AMAF und RAVE 146

4.2.5 Laufzeitoptimierung: Parallelisierung und Multithreading 148

4.2.6 Monte Carlo Verfahren und digitale Brettspiele 150

4.2.7 Zusammenfassung und Ausblick 155

4.3 Computational Intelligence und digitale Brettspiele 156

4.3.1 Künstliche neuronale Netze 157

4.3.1.1 Aufbau und Aktivierungsfunktion eines Neurons 158

4.3.1.2 Netztopologien 161

4.3.1.3 Ausbreitungsregel 164

4.3.1.4 Lernregeln 164

4.3.1.5 Lernmethoden 165

4.3.1.6 Fehlerrückführung 166

4.3.1.7 Mehrschichtige Perzeptren und Backgammon 167

4.3.2 Lernen durch Verstärkung 171

4.3.2.1 Markov-Entscheidungsprozesse 172

4.3.2.2 Dynamische Programmierung und Wertiteration 173

4.3.2.3 Lernen anhand temporaler Differenz 175

4.3.2.4 TD(0) und TD(λ) 176

4.3.2.5 Q-Lernen 178

4.3.2.6 Q-Lernen – Ein Anwendungsbeispiel 183

4.3.2.7 State of the Art: Meisterhaftes Backgammonspiel auf Grundlage verstärkenden Lernens 189

4.3.2.8 Verstärkendes Lernen in unterschiedlichen Brettspieldomänen 191

4.3.3 Zusammenfassung und Ausblick 194

5 Spielübergreifende künstliche Intelligenz(en) – General Game Playing 196

5.1 Zillions of Games 197

5.2 General Game Playing 202

5.2.1 Game Description Language 204

5.2.1.1 Praxisbeispiel – Tic Tac Toe in der Game Description Language 204

5.2.1.2 GDL und das Moment des Zufalls – GDL-II 207

5.2.2 General Game Playing Competition 210

5.2.2.1 Cluneplayer 212

5.2.2.2 Fluxplayer 213

5.2.2.3 Cadiaplayer 213

5.3 Zusammenfassung und Ausblick 215

6 SpoookyJS – Ein webbasiertes universelles Spielsystem 218

6.1 Spiele spielen mit SpoookyJS – Ein Überblick über das Framework ...220

6.1.1 Projekthomepage und Bezugsquellen 220

6.1.2 Systemanforderungen 221

6.1.3 Seitenaufbau und Interfaces 221
6.2 Systemarchitektur – Kernkomponenten von SpoookyJS 227
6.2.1 Spoooky.Game 229
6.2.2 Spoooky.GridWelt 231
6.2.3 Spoooky.Game.loop 234
6.2.4 Spoooky.MetaAgent 237
6.3 Spielerstellung mit SpoookyJS 238
6.3.1 Download und Ordnerstruktur 238
6.3.2 Das Regelwerk des Schachspieles 241
6.3.2.1 Spielfiguren und ihre Bewegungsmöglichkeiten 242
6.3.2.2 Spielverlauf, Spielregeln und Spielregelatome 249
6.3.3 Entitätenblaupausen – Implementation von Spielsteinen und Zugmöglichkeiten 251
6.3.4 Spielwelt- und Spielregeldefinition 257
6.3.4.1 Schachspielregel Unentschieden 260
6.3.4.2 Schachspielregel König steht im Schach 261
6.3.4.3 Schachspielregel Schachmatt 263
6.3.4.4 Schachspielregel Patt 264
6.3.5 Spielebaukasten – Zur Definition unterschiedlicher Spielarten 265
6.3.5.1 Interaktionsmodi 266
6.3.5.2 Lineare und netzartige Spielwelten 266
6.3.5.3 Spielbereiche jenseits der Spielwelt mit Spoooky.Areas 270
6.3.5.4 Spiele mit Zufallselementen – Spielwürfel mit Spoooky.Dice 271
6.4 Spielübergreifende künstliche Intelligenz in SpoookJS 273
6.4.1 Spoooky.MetaAgent und Agentenensemble 274
6.4.1.1 Agentenfokus 276
6.4.1.2 Agentengehirn 279

6.4.1.3 Memorieren und agieren – Das Lernvermögen des Meta Agenten 283
6.4.1.4 Agentenfitness 285
6.4.1.5 Agentenensembles und ihre Tradierung 287
6.4.2 Evaluation und Diskussion der spielübergreifenden KI 289
6.4.2.1 Tic Tac Toe 291
6.4.2.2 Schach und Dame (Checkers) 295
6.4.2.3 Gomoku 299
6.4.2.4 Spiel der Amazonen (Amazons) 302
6.4.2.5 Backgammon 305
7 Zusammenfassung und Ausblick 307
7.1 Vereinfachte Spielerstellung und General Game Playing 308
7.2 Artifizielles Spielvermögen und Erweiterungen 310
7.3 Entwicklungsumgebung und multiagentenbasierter Ansatz 311
8 Literaturverzeichnis 315
9 Abbildungsverzeichnis 340
10 Tabellenverzeichnis 346
11 Abkürzungsverzeichnis 347
12 Verzeichnis der Online-Ressourcen 349
13 Anhang 350
13.1 Listing – Vollständige Beschreibung des Tic Tac Toe Spieles in der GDL 350
13.2 Kommunikationsprotokoll einer vollständigen GGP-Partie des Tic Tac Toe Spieles 353
13.3 Regelwerke 357
13.3.1 Dame International, Checkers, Draughts 357
13.3.2 Backgammon 361

14 Tabellarischer Lebenslauf 365

15 Register 369

1 Einleitung

> „‘Kannst Du zuzählen?‘ fragte die Weiße Königin.
> ‘Was ist eins und eins und eins und eins und eins und eins und eins und eins und eins und eins?‘
> ‘Ich weiß nicht‘, sprach Alice. ‘Ich hab‘ mich verzählt.‘“[1]

Wäre Alice ein Computer, sie würde die Frage der Königin innerhalb eines Lidschlages beantworten: Eins und eins und eins und eins und eins und eins und eins und eins und eins und eins ist eine Folge von zehn Symbolen. Zehn ist sowohl durch fünf als auch durch zwei geradzahlig teilbar. Der Dezimalzahl 10 entspricht die Binärzahl 0000 1010. Um die Zahl 10 anzunähern, muss die Gleitkommazahl 3,162277660168379331998893544432 quadriert werden. Alice jedoch scheint es an Konzentration zu mangeln. Alice zählt eins und eins (und eins) zusammen – und verzählt sich. Die Lösung der an sie gestellten Aufgabe dennoch im Kopf, mag Alice vorgeben, sich verzählt zu haben, weil sie weiß, dass sie mit der Weißen Königin ein Spiel zu spielen hat. Weil sie mutmaßt, dass ihre Entscheidung, der Königin im jetzigen Moment etwas vorzugaukeln, einen positiven Einfluss auf den Gesprächsverlauf mit der Monarchin habe.

Menschliches Problemlösen ist mal kreativ und phantasievoll, mal geradlinig berechnend, Gedankengänge mäandrierend, gewohntes transzendierend, Neues erschließend und Altes tradierend. Computerbasiertes algorithmisches Problemlösen ist zunächst binär, manifestiert sich auf seiner grundlegendsten Ebene in angelegter elektrischer Spannung oder nicht angelegter Spannung, in 1 oder 0, wahr oder falsch, Ja oder Nein. Computern das Rechnen oder die Symbolmanipulation beizubringen, ist trivial; Computern das Problemlösen zu lehren, um sich in einem abgegrenzten Anwendungs- und Problemfeld wie Alice zu gerieren, ist jedoch ein schwieriges Unterfangen.

[1] Carroll 2000, S. 148.

Besonders deutlich offenbart sich die Differenz zwischen menschlichem und computerbasiert artifiziell realisiertem Problemlösungsverhalten in klassischen Brettspielen wie Schach, Dame, Gomoku oder dem Go-Spiel. Klassische Brettspiele zeichnen sich durch ihre wenigen Regeln aus, die jedoch hochkomplexe Entscheidungssituationen zu generieren vermögen. Um eine Aussage über die Güte eines Spielzuges zu einem bestimmten Spielzeitpunkt zu treffen, wenden menschliche Spielerinnen und Spieler heuristische Methoden an, um ihre – im Vergleich zu Computern – langsame Verarbeitungsgeschwindigkeit und geringe Speicherkapazität zu kompensieren. Computer hingegen vermögen Abermillionen Spielrunden vorauszuberechnen, sie memorieren Spielwissen in Form von umfangreichen Eröffnungs- und Endspieldatenbanken, sind menschlichem Schlussfolgern und menschlichem Abstraktionsvermögen, besonders jedoch menschlicher Kreativität unterlegen.

1.1 Fragestellung und Ziel der Arbeit

Die vorliegende Arbeit beschäftigt sich mit der Frage, wie sich artifizielle Spielerinnen und Spieler umsetzen lassen, die in der stark ressourcenbeschränkten Umgebung von Webbrowsern in unterschiedlichen Spielen sinnvolle Entscheidungen zu treffen vermögen. Das im Rahmen der Ausarbeitung entwickelte JavaScript-Framework *SpoookyJS* formuliert eine Antwort auf die gestellte Frage nach einem universellen Spielsystem und leistet drei Beiträge zur Forschung um spielübergreifende artifizielle Intelligenz in browserbasierten digitalen Brettspielen:

- Schnelle und einfache Spielerstellung: Das quelloffene Framework *SpoookyJS* ist kostenlos auf seiner Projekthomepage unter http://www.spoookyjs.de verfügbar und ermöglicht die vereinfachte Erstellung browserbasierter Brettspiele anhand basaler Kenntnisse der Programmiersprache JavaScript.
- Browserbasiertes universelles Spielsystem: Für jedes Spiel, das mit dem Framework erstellt wird, stellt *SpoookyJS* ohne manuelle Anpassungen artifizielle Gegner bereit.

- Wiederverwendbarkeit und Testumgebung: Aufgrund seiner modularen Architektur und der hohen Wiederverwendbarkeit seiner Systembestandteile bietet *SpoookyJS* eine Test- und Entwicklungsumgebung für Experimente, die sich mit der Forschung um artifizielle Spielentscheidungen in digitalen Brettspielen beschäftigen.

Zur Entscheidungsfindung in differierenden Spielsituationen wurde ein Proof of Concept Ansatz verfolgt und umgesetzt, bei dem nicht eine einzige isolierte artifizielle Intelligenz über Zugentscheidungen reflektiert, sondern ein Team von artifiziellen Intelligenzen. So basiert die Entscheidungsfindung im Rahmen des realisierten Multiagentensatzes auf der Analysetätigkeit mehrerer artifizieller Spieler, die Zugmöglichkeiten auf individuelle Art und Weise untersuchen. Als Arbeitswerkzeug wurde jedem Agenten die Monte Carlo Spielbaumsuche beigegeben, mit der sich spielunabhängig Aussagen über die Güte von Zugmöglichkeiten treffen lassen. Als ein frühes Anschauungsbeispiel der Monte Carlo Spielbaumsuche, wie sie in Kapitel 4.2 noch ausführlich referiert wird, mag der folgende Bildschirmausschnitt aus einem der Visualisierungsmodule des Frameworks dienen, anhand dessen die Analysetätigkeit eines artifiziellen Spielers in einer Runde des Tic Tac Toe Spieles veranschaulicht sei. Praktisch nachvollziehen lässt sich das Beispiel auf der entsprechenden Spieleseite unter dem URL http://www.spoookyjs.de/games/tictactoe.

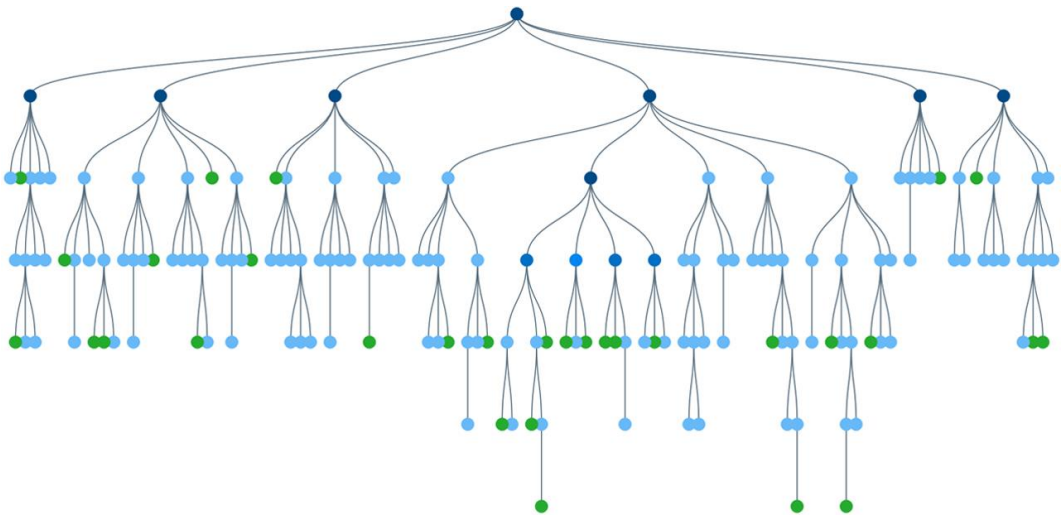


Abbildung 1: Screenshot eines laufzeitgenerierten Monte Carlo Spielbaumes, wie er zur Entscheidungsfindung eines artifiziellen Spielers im Tic Tac Toe Spiel aufgebaut und verwendet wurde.

In der initialen Entscheidungssituation, wie sie mit dem obersten Knoten – dem Wurzelknoten – repräsentiert ist, bieten sich dem artifiziellen Spieler sechs Zugmöglichkeiten, die von ihm teils intensiv, teils weniger eingehend betrachtet werden. Intensiv analysierte Zugmöglichkeiten sind in der Darstellung dunkelblau und weniger intensiv untersuchte Zugmöglichkeiten hellblau dargestellt. Der Wert einer Zugmöglichkeit wird im Rahmen der Monte Carlo Spielbaumsuche abgeschätzt, indem die entsprechende Zugmöglichkeit virtuell ausgeführt wird und anschließend zahlreiche Spielpartien simuliert werden, in denen die Spieler ihre Zugwahlen zufallsbedingt treffen. Je häufiger der Spieler die simulierten Spielpartien gewinnt, desto besser scheint ihm die entsprechende Zugmöglichkeit.

Der Aufbau des Spielbaumes resultiert aus der mehrschrittigen Verfahrensweise der Monte Carlo Spielbaumsuche: Ausgehend von den analysierten Zugmöglichkeiten werden die gegnerischen Zugoptionen in den Spielbaum eingehangen und ebenfalls zufallsbedingt zu ihrem Ende gebracht. Nach und nach generiert sich somit ein Spielbaum wie der zuvor wiedergegebene, bei dem finale Zustände der Spielpartie – Zustände, bei denen ein Spieler das Spiel gewinnt oder die Partie unentschieden ausgeht – grün markiert sind. Herausforderungen im Monte Carlo Spielbaumsuchverfahren manifestieren sich im möglichst schnellen Ausspielen der simulierten Spielpartien sowie der sinnvollen Selektion der Zugmöglichkeiten. So steht der artifizielle Spieler vor der Wahl, bereits bekannte Zugwahlen eingehender zu untersuchen, die in früheren Iterationen des Verfahrens als vielversprechend ermittelt wurden oder noch nicht explorierte Zugmöglichkeiten zu ergründen – ein Problem, das als das Problem des *mehrarmigen Banditen* (*Multi-armed bandit*) bezeichnet wird. Sind ihre Zeitressourcen erschöpft, reichen die einzelnen artifiziellen Spieler ihre Untersuchungsergebnisse weiter an einen übergeordneten Metaspieler, der die Zugvorschläge seiner assoziierten Spieler dazu verwendet, um sich schließlich für eine Zugmöglichkeit zu entscheiden.

Mit dem Webbrowser als Zielplattform intendiert das im Rahmen der vorliegenden Arbeit entwickelte JavaScript-Framework *SpoookyJS*, eine Umgebung bereitzustellen, in der sich schnell und ohne großen Aufwand von

unerfahrenen Benutzerinnen und Benutzern alte und neue Brettspiele umsetzen lassen. In seinem modularen Aufbau fokussiert *SpoookyJS* die spielübergreifende Wiederverwendbarkeit seiner Architekturkomponenten und intendiert die schnelle und unkomplizierte Durchführung von Experimenten um spielübergreifende artifizielle Entscheidungsfindung in digitalen Brettspielen. Aufgrund des browserbasierten Ansatzes ist kein Compiliervorgang vonnöten, der die Arbeit mit der Software verzögerte, einzig die Bearbeitung einer zentralen JavaScript-Datei ist notwendig, um Spiele mit dem Framework umzusetzen und zu experimentieren. Verbunden mit dem hohen Maß an Wiederverwendbarkeit und der Praxisnähe des browserbasierten Ansatzes sind einige inhaltliche und praktische Herausforderungen, die in der folgenden Übersicht über den Erläuterungsverlauf der vorliegenden Arbeit skizziert werden.

1.2 Aufbau und Verlauf der Arbeit

Eingeleitet von einer mechanischen Apparatur, die im achtzehnten Jahrhundert durch ihr außerordentlich qualifiziertes Schachspiel zeitgenössische Spielerinnen und Spieler zu verblüffen vermochte – und auch heute noch erstaunt und fasziniert –, leitet das nachfolgende Kapitel 2 in den Forschungskomplex der künstlichen Intelligenz ein, der sich mit dem Turing-Test und Searles Gedankenexperiment des *Chinesischen Zimmers* in starke und schwache künstliche Intelligenz differenziert findet. Interpretationsangeboten künstlicher Intelligenz, die sich mit der Abbildung menschlichen Denkens und menschlichen Handelns beschäftigen, wird anschließend das Konzept der Rationalität gegenübergestellt, das einen Grundbaustein der vorliegenden Arbeit hervorbringt und formt: das Konzept des intelligenten Agenten.

Anhand unterschiedlicher Spielarten von Spieltheorien führt Kapitel 3 in den Theoriekomplex *Spiel* ein. Werden mit Huizingas *Homo Ludens* und Juuls klassischem Spielmodell zunächst Ansätze vorgestellt, das (menschliche) Spiel über seine Erscheinungsformen zu bestimmen und charakteristische Spielmomente herauszuarbeiten, wird anschließend mit der Spieltheorie nach von Neumann und Morgenstern sowohl das theoretische Handwerkszeug in der

Implementation autarker, rational handelnder Spielindividuen als auch Lösungsmechanismen verschiedenartiger Spiele bereitgestellt. In seiner Intention, das Vokabular in der Beschreibung von Spielen zu formulieren, werden Spiele in Kapitel 3 somit nach ihren Informationsgraden unterschieden und werden Spielbestandteile feingliedrig differenziert, um die praktische Implementation von Spielen theoretisch zu grundieren.

Algorithmen, die dazu dienen, Zugmöglichkeiten einer Spielerin oder eines Spielers zu einem bestimmten Spielzeitpunkt zu bewerten, gar bestmögliches artifizielles Spielverhalten zu realisieren, sind Gegenstand des Methodenkapitels 4. Im vierten Kapitel werden zunächst Spielbaumsuchverfahren wie der *Minimax*-Algorithmus und seine Erweiterung, die *Alpha-Beta Kürzung* vorgestellt. Beide Verfahren ermöglichen in der Theorie optimale Spielentscheidungen, scheitern in der Praxis jedoch zumeist an der Komplexität ihres Gegenstandes. Jene hohe Spielkomplexität wird handhabbar durch *Monte Carlo* Methoden, anhand derer die Güte möglicher Spielzüge durch virtuelles Ausspielen angenähert wird.

Gerieren sich Minimax-Algorithmus, Alpha-Beta Kürzung und der betrachtete Monte Carlo Algorithmus als Spielbaumsuchverfahren, approximieren Verfahren des Forschungskomplexes *Computational Intelligence* schrittweise Lösungen für die an sie gestellten Probleme. Künstliche neuronale Netze wie das *mehrschichtige Perzeptron* lassen sich dazu verwenden, um Spielsituationen zu bewerten und Methoden des verstärkenden Lernens wie der *Q-Lernalgorithmus* realisieren die Fähigkeit artifizieller Spielgegner, aus Fehlern zu lernen – Spielsituationen zu memorieren. Die Fähigkeiten und Einsatzmöglichkeiten der referierten Algorithmen und Modelle werden in Unterkapiteln veranschaulicht, wenn mit den Systemen *Chinook*, *Deep Blue* oder *TD-Gammon* State of the Art Implementationen der Algorithmen im Dame-, Schach- und Backgammonspiel vorgestellt werden.

Welche Ansätze sich in der Praxis erfolgreich zur Umsetzung spielübergreifender künstlicher Intelligenz bewährten und bewähren, darüber informiert Kapitel 5 mit einem Überblick über das Forschungsfeld des *General Game Playing* (*GGP*). Der Forschungskomplex des GGP beschäftigt sich mit der Entwicklung von Systemen,

die dazu fähig sind, die Spielregeln ihnen unbekannter Spiele zu interpretieren und jene Spiele ohne menschliches Expertenwissen gut zu spielen vermögen. Wird zunächst eingeführt in die formale Spielbeschreibungssprache *GDL*, so wird anschließend die *General Game Playing Competition* vorgestellt – ein Wettbewerb, in dem universelle Spielsysteme wie *Cluneplaer*, *Fluxplayer* oder *Cadiaplayer* gegeneinander antreten, um sich in unterschiedlichsten Spielen kompetitiv zu behaupten.

Nachdem der eigene Ansatz in Kapitel 5.3 kontextualisiert und existierenden Ansätzen zur Implementation spielübergreifender artifizieller Intelligenz gegenübergestellt wurde, wird das im Rahmen der vorliegenden Arbeit entwickelte JavaScript-Framework *SpoookyJS* in Kapitel 6 erläutert. Basierend auf der Monte Carlo Spielbaumsuche und dem Q-Lernalgorithmus ermöglicht *SpoookyJS* die browserbasierte – und somit plattformunabhängige – Erstellung digitaler Brettspiele einschließlich spielübergreifender künstlicher Intelligenz. Eingeleitet wird das sechste Kapitel mit Informationen zur Projekthomepage, Bezugsquellen des Frameworks und den unterschiedlichen Interaktions- und Seitenelementen, die von dem Rahmenwerk bereitgestellt werden. Im anschließenden Kapitel 6.2 wird die Systemarchitektur von *SpoookyJS* erläutert, bevor in Kapitel 6.3 anhand des Schachspiels auf die praktische Erstellung von Spielen mit *SpoookyJS* eingegangen wird. Kapitel 6.4 trägt vor, wie die spielübergreifende multiagentenbasierte künstliche Intelligenz im Framework realisiert wurde und expliziert den Aufbau der artifiziellen Spielgegner, die in unterschiedlichen Spieldomänen zu agieren vermögen. Abgeschlossen wird das sechste Kapitel mit der Evaluation der spielübergreifenden künstlichen Intelligenz in den Spielen Tic Tac Toe, Schach und Dame, Gomoku, Amazons und Backgammon, wie sie mit dem Framework browserbasiert umgesetzt wurden.

Kapitel 7 fasst die zentralen Methoden und Thesen der Arbeit zusammen, bespricht die Einsatzmöglichkeiten des bereitgestellten Systems und gibt einen Ausblick auf die Erweiterungsmöglichkeiten sowohl des Frameworks als auch der eingesetzten Methoden zur Realisierung spielübergreifender artifizieller Intelligenz.

2 Von einer ehrenhaften Täuschung – Zum Gegenstand der künstlichen Intelligenz

> „Ich weis [sic] nicht, ob alle meine Mitzuschauer so, wie ich, dachten, wenigstens glaubte ich auf einigen Gesichtern etwas zu bemerken, welches den Gedanken von einer übernatürlichen Macht in ihrem Herzen verriet: eine alte Dame aber, die vielleicht die ersten Eindrücke von guten, und bösen Geistern durch ihre Umme erhalten hatte, schlug ein Kreuz, mit einem andächtigen Seufzer vor sich, und schlich an ein etwas entferntes Fenster, um dem bösen Feind den sie unfehlbar bey oder in der Maschine vermuthete, nicht so nahe zu seyn.“[2]

Als sich die Türen des glanzerfüllten Saales öffnen und jene vielbesprochene Maschine sich endlich den ersten skeptischen Blicken der zahlreichen, mitunter von weither angereisten, Besucherinnen und Besucher preisgibt, zögern einige Teilnehmerinnen und Teilnehmer dieser außergewöhnlichen Veranstaltung zunächst, den Raum zu betreten und auf einem der exakt auf die artifizielle Präsenz ausgerichteten Stühle Platz zu nehmen. Außergewöhnlich, das ist bereits die äußere Erscheinung der Maschine:

> „Sie bestehet aus einem drey und ein halben Fuß langen, zwey Fuß breiten, und zwey und ein halben Fuß hohen Kasten, in Form eines Schreibtisches der auf vier Walzen ruht, und von einem Orte zum andern gerücket werden kann. Hinter demselben sitzt eine türkisch gekleidete männliche Figur von mittlerer Menschengröße auf einem hölzernen Stuhle, der an dem Schrank fest gemacht ist, und mit demselben fortgeschoben wird. Mit dem rechten Arme lehnt sie sich auf den Schrank, in der linken Hand aber hält sie eine lange türkische Tabakspfeife, die sie eben vom Munde genommen zu haben scheinet. Der linke ist eben derjenige Arm, mit welchem sie spielt, nachdem ihr das Tabaksrohr aus

[2] Windisch 1783, S. 11f.

> der Hand gezogen worden ist. Vor ihr liegt das an den Schrank geschraubte Schachbrett, auf welches sie die Augen beständig gerichtet hält.“[3]

Rasch füllt sich der Saal, setzen sich mehr und mehr Menschen auf die zur Beobachtung des Geschehens vorgesehenen Sitzgelegenheiten, wagen sich gar manche kühne Beobachterinnen und Beobachter in die Nähe des exotisch gekleideten Mannes aus Holz und Metall, stirnrunzelnd in murmelndes Diskutieren vertieft ob der geraunten Fähigkeiten dieser seltsamen Maschine. Nachdem der letzte Sitzplatz belegt, die Lautstärke der Gespräche auf ein Flüstern reduziert und die Türen des Saales geschlossen sind, geht stolz ein vornehm gekleideter Herr durch die Reihen. Mit starker Stimme richtet er sich an das Publikum, intoniert wieder und wieder seine Frage, wer es mit der Maschine im Schachspiel aufzunehmen vermag. Schweigen. Zögerliche Blicke. Abwarten. Verhalten hebt eine junge Frau ihren Arm, steht endlich auf und nimmt die Herausforderung an.

[3] Windisch 1783, S. 19f. Weitere zeitgemäße Beschreibungen der Maschine Wolfgang von Kempelens sind zu finden in Busch 1821, S. 151f. und Hindenburg 1784, S. 12.

2.1 Über den Schachspieler des Herrn von Kempelen

Bevor das Spiel jedoch seinen Lauf nimmt, öffnet Wolfgang von Kempelen, der Erfinder des aufgrund seines exotischen Äußeren als *Schachtürke* bezeichneten Automaten,[4] die hölzernen Türen und Schubladen und gibt das Innere seiner Schöpfung den Blicken der Menschenmenge preis. Um vorab jegliche Bedenken der Zuschauerinnen und Zuschauer zu zerstreuen, dass es im Folgenden nicht mit rechten Dingen zuginge, offenbaren sich nach und nach feinmechanische Feinheiten, zeigen sich Schräubchen und Zahnräder, enthüllen sich komplex wirkende Mechanismen zur Steuerung des artifiziellen Schachspielers, die unbewegt darauf zu warten scheinen, endlich in Aktion gegen die menschliche Spielerin treten zu dürfen.[5]

Abbildung 2: Im Jahr 2004 der Öffentlichkeit vorgestellter Nachbau des Automaten Wolfgang von Kempelens. Auf der linken Seite in geschlossenem, auf der rechten Seite in geöffnetem Zustand. Bildverwendung mit freundlicher Genehmigung des Heinz Nixdorf MuseumsForum.

Nachdem die Türen des Automaten schließlich geschlossen und alle Zuschauerinnen und Zuschauer von der materiellen Komplexität der ernst dreinblickenden Maschine überzeugt wurden, beginnt das Schachspiel. Bedacht wählt die Spielerin Spielzug um Spielzug, reflektiert ihre Züge und künftige Zugmöglichkeiten im Voraus, stellt ihrem nichtmenschlichen Gegenspieler gar

[4] Ein gutes Jahrhundert vor der Arbeit von Kempelens definiert Hobbes Automaten als „Maschinen, die sich selbst durch Federn und Räder bewegen, wie eine Uhr“ (Hobbes 1984, S. 5). An dieser Stelle und im Folgenden sei ein Automat verstanden als eine Maschine, die selbständig an sie gestellte Aufgaben bearbeitet und erfüllt. Einen Überblick über frühe Automaten wie die 1738 von Jacques de Vaucanson vorgestellte mechanische Ente bietet Nilsson 2010, S. 3–29.

[5] Eine ausführliche Beschreibung des Innenlebens des Automaten ist zu finden u.a. in Windisch 1783, S. 22–28 und Standage 2005, S. 30f.

Fallen und weicht Gefährdungen geschickt aus. Schlussendlich jedoch verliert sie die Schachpartie, wohl aber in beruhigender Gewissheit, dass hier nichts mit rechten Dingen zuging, dass jener außergewöhnlich gekleidete hölzerne Automat erneut das Unfassbare bewältigt hat: Im „schwerste[n] aller Spiele“[6], wie ein Beobachter des Geschehens bewundernd in einem Briefwechsel berichtet, gegen einen intelligent agierenden menschlichen Gegner zu bestehen, gar zu obsiegen.

Als demonstrierte das findige Schachspiel nicht hinreichend gebührend die Fähigkeiten des Automaten, so folgt zum Ende der Darbietung das Sprachspiel des *Schachtürken*. Wie Carl Friedrich Hindenburg in seiner 1784 publizierten Arbeit *Über den Schachspieler des Herrn von Kempelen* ausführt,

> „wird [zuletzt] eine Tafel mit goldenen Buchstaben und Ziffern auf das Schachbret gelegt, vermittelst welcher die Figur eine willkührlich ihr vorgegebene Frage dadurch beantwortet, daß sie die Buchstaben einzeln mit den Fingern zeigt, die zusammengenommen die Antwort ausmachen. Ehe diese Antwort gegeben wird, wird das Werk in der Figur aufgezogen. Nach jedem einzelnen Worte, das Herr Anton aus den angezeigten und, wegen der entfernten Zuschauer, laut nachgesprochenen Buchstaben sammelt, legt die Figur die Hand aufs Kissen und erhebt sie von neuem für die Buchstaben des folgenden Worts.“[7]

Was nun folgt, ist außerordentlich, ist erstaunlich, ist verblüffend: Auf die Frage seines Alters erwidert die schachspielende Maschine „192 Monate“; die Frage, ob er verheiratet sei, beantwortet der Automat Hindenburg zufolge, er „habe viel [sic] Weiber“[8]. Jener Automat, der noch wenige Minuten zuvor die gespannte Menschenmenge durch seine Spielfähigkeit in Staunen versetzte, suggeriert nun, dass er des Menschen Sprache nicht einzig auditiv wahrnehmen könne, vielmehr die an ihn gerichteten Fragen zu verstehen und sich gar mit Hilfe der auf dem Schachbrett befindlichen Buchstaben und Ziffern natürlichsprachig zu äußern vermag. Wenngleich die Erwiderungen des *Schachtürken* in heutigen Ohren mitunter nicht mehr derart sinnerfüllt nachklingen mögen, wie sie Hindenburg

[6] Windisch 1783, S. 3.
[7] Hindenburg 1784, S. 18.
[8] Hindenburg 1784, S. 18.

gegen Ende des 18. Jahrhunderts empfindet, so versetzen die Fähigkeiten des orientalisch gekleideten Automaten selbst zeitgenössische algorithmen- und technikerprobte Annahmen von den Fähigkeiten schlauer Maschinen in Staunen. – Ein Automat, eine Maschine, die dazu fähig ist, hochkomplexe Eigenschaften menschlichen Da- und Soseins perfekt nachzuahmen, kann (und darf?) so etwas tatsächlich existieren?[9]

Selbstverständlich ist denn auch der enorme Erfolg dieser Zauberapparatur nicht von ungefähr: Nach seiner initialen wunderlichen Vorstellung am habsburgischen Hofe Maria Theresias im Jahre 1770 bereist Wolfgang von Kempelen mehrere Jahrzehnte lang Europa, um seinen *Schachtürken* zu präsentieren.[10] Nach dem Tode von Kempelens 1804 erwirbt der Mechaniker und Schausteller Johann Nepomuk Mälzel die Maschine und dehnt die Präsentationsreisen gar aus bis nach Nordamerika.[11] Ein enormer anhaltender Erfolg, wenngleich Wolfgang von Kempelen zeitlebens nicht läugnet, „daß bei dem Spiele dieser Figur Täuschung vorgehe“[12]. Nur wie sich die Täuschung vollziehe, das sei partout nicht herauszufinden. So sinniert Karl Gottlieb von Windisch in einem seiner *Briefe über den Schachspieler des Herrn von Kempelen*:

> „Und doch muß es Täuschung seyn, sagen Sie. O! Das wird Ihnen keine vernünftige Seele, auch selbst der Erfinder nicht abläugnen. – Aber, worinnen besteht dann diese Täuschung? – Das eben ist der gordische Knoten, den selbst Alexander durch keinen Schwertstreich auflösen würde. – Also, eine Täuschung? Ja, aber eine Täuschung die dem menschlichen Verstande Ehre macht […].“[13]

---

[9] Auch von Kempelen muss die gar unerhörte Außergewöhnlichkeit des Sprachspieles seines Automaten bewusst gewesen und bewusst geworden sein, unterließ er es doch in späteren Vorstellungen, die Dialogfähigkeit seiner Erfindung zu demonstrieren (vgl. Standage 2005, S. 206f.), um seinen *Schachtürken* nicht vorzeitig als Täuschung zu entlarven.

[10] Vgl. Standage 2005, S. 28–34.

[11] Vgl. das Kapitel *Der hölzerne Krieger in Amerika* in Standage 2005, S. 131–150 sowie die ausstellungsbegleitenden Materialien des Heinz Nixdorf MuseumsForum Paderborn (Heinz Nixdorf MuseumsForum: Frühe Automaten - Wunder der Technik. Schachtürke wieder zum Leben erweckt. Online verfügbar unter http://www.hnf.de/museum/die-mechanisierung-der-informationstechnik/fruehe-automaten-wunder-der-technik/schachtuerke-wieder-zum-leben-erweckt.html, zuletzt geprüft am 22.02.2013).

[12] Busch 1821, S. 153.

[13] Windisch 1783, S. 9. Standage weist darauf hin, dass die brieflichen Ausführungen Windischs mit Vorsicht zu genießen seien, dienten sie doch alleine dem Zweck, von Kempelens Schachautomaten zu bewerben: „Im Grunde war das ganze Werk [d.h. Windischs Briefe, den *Schachtürken*

Ungeachtet seiner großen Popularität im ausgehenden 18. und im frühen 19. Jahrhundert findet der schachspielende Automat Wolfgang von Kempelens jedoch nur selten oder nur am Rande seine Erwähnung in einführenden Werken in die Fach- und Forschungsdisziplin der künstlichen Intelligenz, die sich im Juli des Jahres 1956 mit einer Fachkonferenz am Dartmouth College in Hanover, New Hampshire gründet[14] und ebensolche Fragen fokussiert, die jener *Schachtürke* vorgeblich spielerisch zu beantworten vermochte: Wie könnte, wie sollte eine des Schachspielens fähige Maschine beschaffen sein? Ist es möglich, eine Maschine zu entwickeln, die im Schachspiel *schlau* agiert? Eine Maschine, die sich als *intelligent* signifizieren ließe?

Dass nun Wolfgang von Kempelens Automat nur selten seine Erwähnung in forschungsrelevanten Kontexten findet, das vollzieht sich aus gutem Grunde. Denn der schachspielende Automat konnte schließlich nicht verbergen, dass seine Kleidung Verkleidung war, dass sein Vermögen, menschliche Eigenschaften nachzuahmen, eine ehrenhafte Täuschung war, die es vermochte, die meisten ihrer Gegner in „weniger als einer halben Stunde“[15] Spielzeit zu besiegen. Als initialer Denkimpuls jedoch ist Wolfgang von Kempelens Automat äußerst dienlich, gar essentiell: Wie konstituiert sich künstliche Intelligenz im Schachspiel und in anderen klassischen Brettspielen wie Backgammon oder dem Damespiel?[16] Was ist mit den Begriffen *künstlich* und *intelligent* signifiziert, was ist zuallererst gemeint und impliziert, wenn von *künstlicher Intelligenz* die Rede ist?

betreffend] nichts anderes als ein geschickt aufgemachter Werbetext für den Türken, der darin als der erstaunlichste Automat, den es je gegeben hat, beschrieben wurde. Es sollte wohl seine Leser animieren, dem so gepriesenen Apparat einen persönlichen Besuch abzustatten.“ (Standage 2005, S. 60).

[14] Vgl. Russell und Norvig 2012, S. 40–54 sowie das Kapitel *Gatherings* in Nilsson 2010, S. 49–61 für einen ausführlichen Überblick über die Gründung und Entwicklung der Forschungsdisziplin der künstlichen Intelligenz.

[15] Standage 2005, S. 34.

[16] Ein Brettspiel sei definiert als ein Spiel, in dessen Verlauf Spielsymbole (Spielsteine, Spielfiguren) auf einem Spielbrett platziert und/oder von dem Spielbrett entfernt und/oder auf dem Spielbrett bewegt werden (vgl. auch He et al. 2013, S. 31f.).

2.2 Fokussierungsweisen künstlicher Intelligenz

Wäre Wolfgang von Kempelens *Schachtürke* keine solch beeindruckende Täuschung gewesen – insgeheim wurde der schachspielende Automat von einem Menschen in seinem Inneren bedient [17] –, so hätte von Kempelen die außerordentliche Meisterleistung vollzogen, eine mechanische Vorrichtung zu kreieren, die sowohl menschliches Handeln als auch menschliches Denken nicht einzig imitieren, sondern gar artifiziell hervorbringen könnte. Eine mechanische Vorrichtung, der nicht einzig durch ihr orientalisch anmutendes humanoides Erscheinungsbild das Adjektiv *menschlich* zugesprochen würde – ein Automat, dazu fähig, die Substanz, das Wesen des Menschlichen zu spiegeln, gar zu reproduzieren[18] – so folgert und so fragt auch Ensmenger mit einem Hinweis auf die Relevanz des Schachspieles:

> "Because chess was historically regarded as such an essentially human endeavor, the ability of machines to play chess seemed to have fundamental metaphysical implications. If a machine could emulate what was widely considered the summit of human intelligence, namely the abstract reasoning associated with chess-playing ability, then was it not possible that the essence, and not just the appearance, of humanity could eventually be reproduced mechanically [...]?"[19]

Wie meisterlich solch ein Meisterstück wäre, die Spielfiguren des Schachspieles auf dem Schachbrett zu bewegen, zuvor gar zu bedenken, welche Spielzüge den Spielverlauf zu einem individuell positiven Spielergebnis leiten könnten, bringt die Reflexion über die Implikationen dieser beiden Fokussierungsweisen in der Nachahmung menschlichen Denkens und menschlichen Agierens hervor: Um die Schachfiguren auf dem Spielbrett bewegen zu können, muss die Maschine zuallererst über Bestandteile, Bauteile und Mechanismen verfügen, um ihre unmittelbare und mittelbare Umgebung sinnlich wahrnehmen zu können. Das mit Hilfe von Sensoren Wahrgenommene wird anschließend fürwahrgenommen,

[17] Vgl. das Kapitel *Die Geheimnisse des Türken* in Standage 2005, S. 165–187.
[18] Beachte auch die Gegenüberstellung der Begriffe *organisch* und *artifiziell* bei Ensmenger 2012, S. 9.
[19] Ensmenger 2012, S. 9.

wenn Prozesse der Mustererkennung Entitäten [20] aus der Wahrnehmung extrahieren, um in daran anschließenden Folgeprozessen Sinnfreies von Sinnerfülltem zu unterscheiden. Nach und nach, von Zahnrad zu Zahnrad, von Mechanismus zu Mechanismus, von Algorithmus zu Algorithmus generieren sich somit Bedeutungen. Sinngehalte, die sich zunächst auf Feststellungen wie „dies ist eine Schachfigur", „diese Schachfigur hat die Form einer Königin" beschränkt; Bedeutungen, deren Kontext zuallererst erschlossen werden muss: „meine Wahrnehmung ist gerichtet auf das Schachspiel", „das Schachspiel verfügt über die folgenden, nicht zu negierenden Regeln". Damit schließlich Handeln auf Grundlage der Mustererkennung und logischen Schlussfolgerns resultiert, muss die Maschine über Aktuatoren verfügen, um mit ihrer Umwelt interagieren und sich aus ihrer isolierten Hülle erheben zu können. Menschliches und artifiziell implementiertes Agieren im Schachspiel setzt dabei jedoch nicht einzig voraus, Spielfiguren nach der ihnen vorgeschriebenen Weise zu bewegen, sondern im Wettstreit mit einer Gegnerin oder einem Gegner flexibel auf unterschiedliche und sich ständig ändernde Anforderungen zu reagieren, die sich nach und nach im komplexen Laufe des Spieles ergeben, gar Züge im Voraus zu berechnen und Strategien zu entwickeln, die das individuelle Spielverhalten auf lange Sicht optimieren.

Mit seinen vermeintlichen Fähigkeiten erdichtet und eint die schachspielende Maschine von Kempelens zwei Fokussierungsweisen, die Russell und Norvig[21] in ihrem einführenden Standardwerk in die Forschungsdisziplin der künstlichen Intelligenz voneinander separieren. So lässt sich die Beschäftigung mit künstlicher Intelligenz als Abbildung und Nachahmung menschlicher Eigenschaften den Autoren zufolge in zwei Theorie- und Anwendungsgebiete unterschieden: Auf der einen Seite der Fragen- und Problemkomplex, ob und wie sich menschliches Denken artifiziell implementieren lässt, um schließlich „Computern das Denken beizubringen" [22], auf der anderen Seite die Bestrebung, menschliches

[20] Der Begriff der *Entität* soll an dieser Stelle und im weiteren Verlauf der Arbeit unbestimmt verstanden sein als „allgemeine Bezeichnung für ein sprachliches bzw. gedankliches Objekt oder für ein außersprachliches Bezugsobjekt" (Prechtl 2008, S. 138).
[21] Russell und Norvig 2012.
[22] Haugeland 1989, zitiert nach Russell und Norvig 2012, S. 23.

Handlungsvermögen abzubilden und künstlich zu repräsentieren, um „Computer dazu zu bringen, Dinge zu tun, bei denen ihnen momentan der Mensch noch überlegen ist.“[23] Wie gelungen, wie erfolgreich eine artifizielle Intelligenz ist, das ist in den Augen dieser beiden Betrachtungsweisen abhängig davon, wie geschickt sie unterschiedliche Facetten menschlicher Eigenschaften und Fähigkeiten wiederzugeben vermag.

[23] Rich und Knight 1991, zitiert nach Russell und Norvig 2012, S. 23.

2.2.1 Menschliche vs. nichtmenschliche Intelligenz – Der Turing-Test

Als Indikator der Fähigkeit eines artifiziellen Systems, menschliches (sprachliches) Agieren nachzuahmen – gar zu fingieren –, lässt sich der Turing-Test anführen.[24] Im Jahr 1950 von Alan M. Turing in seiner Arbeit *Computing Machinery and Intelligence* vorgestellt, basiert der Turing-Test auf einem Imitationsspiel, das von einer Frau (B), einem Mann (A) und einem Fragesteller (C) gespielt wird.[25] Akustisch und visuell von der Frau und dem Mann isoliert, obliegt es dem Fragesteller, das Geschlecht von A und B herauszufinden, so dass er am Ende des Spieles konstatieren kann: „A ist männlich und B ist weiblich“ oder „A ist weiblich und B ist männlich“.[26] Veranschaulicht sei der Versuchsaufbau mit der folgenden Darstellung.[27]

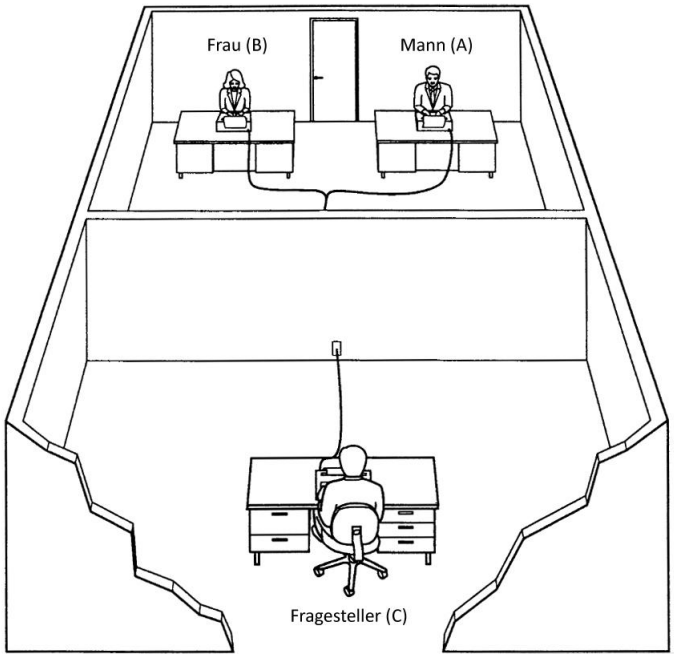


Abbildung 3: Versuchsaufbau des Turing-Tests: Der Fragesteller C befragt die Spielpartner A und B – und versucht herauszufinden, welchen Geschlechts A ist.

[24] Ein System sei verstanden als ein „Zusammenhang von einzelnen Teilen, die voneinander abhängig sind und so ein Ganzes bilden, das einer bestimmten Ordnung unterliegt“ (Prechtl 2008, S. 599). Eine feingliedrige Differenzierung unterschiedlicher Arten von Systemen (*System*, *Komplexes System*, *Kompliziertes System* sowie *Chaotisches System*) findet sich in Kroll 2013, S. 4.

[25] Turing selbst bezeichnet seinen ursprünglichen Versuchsaufbau als „imitation game“ (Turing 1950, S. 433); sehr schnell jedoch fand sich jenes *Imitationsspiel* umbenannt in den *Turing-Test*. Mit dem Loebner-Preis wird seit dem Jahr 1991 jährlich ermittelt, welches Computerprogramm den Test am besten (in unterschiedlichen Kategorien) besteht.

[26] Vgl. Turing 1950, S. 433.

[27] Die Grafik des Turingtests ist wiedergegeben nach Fogel 2001, S. 8. Die Bezeichnungen der Versuchsteilnehmer wurden ins Deutsche übersetzt.

Verbunden über ein Kommunikationsmedium, das keine Rückschlüsse auf das Geschlecht von A oder B zulässt, fragt C in einem Konversationsbeispiel Turings zuerst nach der äußeren Erscheinung von A:

> „C: Will X [im vorliegenden Falle A] please tell me the length of his or her hair?“[28]

Spieler A versucht, den Fragesteller zu täuschen und berichtet, eine kurzgeschnittene Damenfrisur – einen Bubikopf – zu tragen. Wie Turing anmerkt, besteht die Aufgabe der Spielerin B darin, dem Fragesteller zu helfen – und könnte eine sinnvolle Strategie der Spielerin darin bestehen, wahrheitsgemäß zu antworten, um C von sich zu überzeugen:

> „The best strategy for her is probably to give truthful answers. She can add such things as ‘I am the woman, don’t listen to him!‘ to her answers, but it will avail nothing as the man can make similar remarks.“[29]

Was geschähe, würde der Spieler A oder die Spielerin B nun durch einen Computer ersetzt? Würde sich der Fragesteller ebenso häufig richtig oder falsch entscheiden, wie in der eingangs vorgetragenen Spielkonfiguration, in der das Imitationsspiel einzig von menschlichen Spielerinnen und Spielern bestritten wird? Stellt Turing zu Beginn seiner Arbeit die Frage, ob Maschinen denken können, so transformiert sich seine Frage ob der Reflexion um die Folgen des maschinellen Einflusses nun in einen Verhaltensintelligenztest,[30] bei dem ein Mensch Q unter Verwendung einer Computertastatur und eines Monitors[31] mit einer Gesprächspartnerin A kommuniziert, die sich in einem anderen, von Q visuell und akustisch isolierten, Raum befindet. Bei jeder Äußerung von A stellt sich Q die Frage, ob er sich mit einer menschlichen Gesprächspartnerin austauscht oder ob es ein nichtmenschliches, ein artifizielles System ist, das antwortet. Vermag ein solches System, sein menschliches Gegenüber zu täuschen, so besteht es den Turing-Test.

[28] Turing 1950, S. 433.
[29] Turing 1950, S. 433.
[30] Vgl. Russell und Norvig 2012, S. 1177.
[31] Turing verwendet einen *Teleprinter* als Kommunikationsmedium (vgl. Turing 1950, S. 433).

In verschiedenen Konversationsbeispielen, wie sie Turing in seiner Arbeit anführt, offenbaren sich die Schwierigkeiten und Herausforderungen in der Nachahmung des Menschlichen, die sich mit dem gleichnamigen Test verbinden:[32]

„Q: Please write me a sonnet on the subject of the Forth Bridge.
A: Count me out on this one. I never could write poetry."

Q bittet A darum, ein Sonett über eine Brücke zu schreiben, die an einem Meeresarm an der Ostküste Schottlands gelegen ist. Indem A antwortet, dass sie des lyrischen Schreibens nicht fähig sei, aktiviert sie durch die Verknüpfung der beiden Wortformen *Sonett* und *Lyrik* (*poetry*) offensichtlich Hintergrundwissen, das sich in der Frage von Q nicht ausgeführt findet. Ist es einem artifiziellen System möglich, solch eine Konversation zu führen? Ließe sich mit A auch ein Gespräch über andere Themen führen?

„Q: Add 34957 to 70764.
A: (Pause about 30 seconds and then give as answer) 105621."

Q weist A an, zwei ganzzahlige Dezimalzahlen zu addieren. A suggeriert Q, sich für die Berechnung eine halbe Minute Zeit zu lassen und gibt schließlich ein falsches Ergebnis zurück. Ist der Rechenfehler um den Wert von 100 ein menschlicher? Oder täuscht das artifizielle System einzig vor, ein menschlicher Kommunikationspartner zu sein, indem es des Menschen begrenzte Rechenleistung anhand des Fehlers aufzeigt?

„Q: Do you play chess?
A: Yes.
Q: I have K at my K1, and no other pieces. You have only K at K6 and R at R1. It is your move. What do you play?
A: (After a pause of 15 seconds) R-R8 mate."

Q fragt A zunächst, ob A des Schachspieles mächtig sei. A bejaht und beantwortet die von Q gestellte Frage, welche Zugwahl A in einer bestimmten Spielkonfiguration vornehmen würde, mit einem bestimmten Zug. Ist der

[32] Alle wiedergegebenen Beispiele sind wörtlich zitiert aus: Turing 1950, S. 434f.

gewählte Zug ein legitimer Zug? Imaginiert A eine Abbildung des Schachbrettes in der bestimmten Anordnung der Spielfiguren und überlegt sich anhand der Vorstellung des Schachbrettes die beste Zugmöglichkeit?

Mit Turings Konversationsbeispielen klingt an, wie komplex sich die Nachahmung menschlichen Sprachvermögens gestaltet, das auf menschliche Eigenschaften wie die des Denkens verweist – und offenbart sich eine Fokussierungsweise in der Forschung um künstliche Intelligenz, die des Sprachspieles mächtige Systeme hervorzubringen intendiert. Ließe sich nun ein System als *intelligent* bezeichnen, das den Turing-Test nicht besteht, komplexe Herausforderungen wie das *Problem des Handlungsreisenden*[33] jedoch effizient zu bearbeiten vermag?[34] Der Begriff der Intelligenz, wie ihn Pfeifer und Scheier in Ihrer Arbeit *Understanding Intelligence* [35] zunächst deskriptiv annähern, signifiziert einen Komplex unterschiedlicher Fähigkeiten, in dem das menschliche Sprachvermögen nur eines von zahlreichen Definitionspartikeln darstellt. Wesentlich für die definitorische Arbeit der Autoren ist darüber hinaus menschliches Lern- und Gedächtnisvermögen sowie Bewusstsein, Intuition, Kreativität, folgerndes[36] und logisches Denken aber auch Affektionen,[37] motorische Fähigkeiten sowie die Kompetenz, in einer höchst komplexen Umgebung zu bestehen.[38]

---

[33] Das *Problem des Handlungsreisenden* (*Traveling Salesman Problem*) beschreibt die Frage, ob und wie sich die möglichst optimale – d.h. möglichst kurze – Reiseroute des Handlungsreisenden bzw. der Handlungsreisenden auf seinem / ihrem Weg vom Ausgangsort über verschiedene Stationen der Reise bis hin zum Ausgangsort bestimmen lässt.

[34] French beantwortet die Frage nach der Relevanz des Turing-Testes wenn er anwendungsfokussiert ausführt: „building machines that would never pass a turing test, but that can interact with humans in a highly meaningful, informative way, is the way forward in ai.“ (French 2012, S. 74) und ferner spielbezogen: „Computers of the future, even if they never pass a Turing Test, will potentially be able to see patterns and relationships between patterns that we, with all our experience in the world, might simply have missed. The phenomenal computing capacity of computers, along with ever-better data capture, storage, retrieval, and processing algorithms, has given rise to computer programs that play chess, backgammon, Go, and many other highly ‘cognitive‘ games as well, or better, than most humans.“ (French 2012, S. 77).

[35] Pfeifer und Scheier 2001.

[36] Bennett und Hacker differenzieren folgerndes Denken von streng logischem Denken, wenn sie definieren: „[folgerndes Denken, im Englischen ‘Reasoning’] bezeichnet nicht folgern im strengen logischen Sinn, sondern ein von Ausgangspunkten herkommendes Durch- und Weiterdenken einer Sache, die es mehr oder weniger zu Ende denkt. Ein Denken, das, obgleich ihm die Werkzeuge der Logik zur Verfügung stehen, auch fehlgehen und Irrtümern unterliegen kann.” (Bennett und Hacker 2012, S. 18).

[37] Vgl. die Untergliederung von Affektionen in Emotionen, Erregungen und Stimmungen in Bennett und Hacker 2012, S. 266.

[38] Vgl. Pfeifer und Scheier 2001, S. 7–12.

2.2.2 Schwache vs. starke künstliche Intelligenz

"Do they think? Do they *really* think?"[39]

Ob einem Computersystem, das den Turing-Test besteht, tatsächlich menschenähnliche kognitive[40] Fähigkeiten und Zustände zu eigen sind, stellt sich mit der Debatte zwischen schwacher und starker künstlicher Intelligenz, im Folgenden mit *KI* abgekürzt, in Frage. Im Unterschied zur schwachen KI, die Computer als Werkzeuge zur Lösung bestimmter Probleme versteht, attribuiert die These der starken KI Computern menschliche Fähigkeiten: Der starken KI zufolge besitze ein Computer, der den Turing-Test aufgrund bestimmter Fähigkeiten zu bestehen vermag – zum Beispiel durch seine Fähigkeit, sich sprachlich auf materielle oder immaterielle Dinge zu beziehen – jene Fähigkeiten tatsächlich.[41]

Mit seinem Gedankenexperiment des *Chinesischen Zimmers*, das den symbolverarbeitenden Charakter des als künstlich intelligent zu signifizierenden Systems betont, sucht Searle die Ansicht einer starken KI zu widerlegen, wie sie einige Interpretinnen und Interpreten aus dem Turing-Test folgern:[42] Der chinesischen Sprache nicht mächtig, findet sich Searle in seinem Gedankenversuch wieder in einem isolierten Zimmer, in dem sich zahlreiche Körbe voller Kärtchen befinden, auf denen chinesische Symbole aufgedruckt sind. Neben den Symbolkärtchen findet sich ein seitenstarkes Buch, das in Searles Muttersprache Englisch verfasst ist und darüber informiert, nach welchen Regeln die Symbole miteinander kombiniert werden müssen. Nachdem Searle einige Symbole

[39] Dennett 1998, S. 20. Kursivschrift nach dem Autor.

[40] Pfeifer et al. differenzieren die Begriffe *Kognition*, *Denken* sowie *Bewusstsein* und weisen wie im Folgenden wiedergegeben auf die enge Verknüpfung von Kognition und Körperlichkeit hin: „Cognition is sometimes employed as a more general term than thinking because it does not necessarily imply consciousness. However, it is important to keep in mind that despite the more abstract connotations of thinking as compared to cognition, thinking is not a disembodied process: […] it seems to be directly tied to sensory-motor and other bodily (i.e., physiological) processes, as is cognition." (Pfeifer et al. 2007, S. 9).

[41] Vgl. Searle 1990, S. 40.

[42] Searles Gedankenversuch ist wiedergegeben nach Searle 1990, S. 40f. In seinem Zeitungsbeitrag bezeichnet Zimmer das Gedankenexperiment Searles gar als Provokation (Zimmer 1990). Eine erschöpfende Betrachtung des Gedankenexperimentes und seiner Ex- und Implikationen findet sich in Dresler 2009.

betrachtet hat, deren Form sich ihm zwar erschließt, die Bedeutung der einzelnen und miteinander verknüpften Symbole jedoch verborgen bleibt, reicht ihm ein Mensch von außerhalb des Raumes durch einen kleinen Schlitz in einer der Wände des Zimmers einen Stapel mit Symbolen herein. Searle verarbeitet nun den Symbolstapel nach der Handlungsvorschrift, wie sie ihm das Buch vorgibt und gibt nach einigen Minuten zahlreiche Kartenstöße durch eine andere Wandöffnung heraus, die das Ergebnis seiner Arbeit darstellen. Ohne auch nur eines der chinesischen Symbole zu verstehen, war es Searle möglich, durch die Verknüpfung einzelner Zeichen auf Grundlage eines Regelwerkes Symbolfolgen zu generieren, die jenseits des chinesischen Zimmers für einen Interpreten der Symbolfolge mit Bedeutung verbunden sind. Den Bezug zu Turings Test stellt Searle wie folgend her:

> „In die Computersprache übersetzt, wäre also das Regelbuch das Computerprogramm, sein Autor der Programmierer und ich der Computer; die Körbe voller Symbole wären die Daten, die kleinen mir ausgehändigten Stöße die Fragen und die von mir hinausgereichten Stöße die Antworten. Nehmen wir nun an, das Regelbuch sei so verfaßt, daß meine Antworten auf die Fragen von denen eines gebürtigen Chinesen nicht zu unterscheiden sind […] Also hätte ich den Turing-Test für Chinesisch bestanden. Gleichwohl habe ich nicht die geringste Ahnung von dieser Sprache […] Wie ein Computer hantiere ich mit Symbolen, aber verbinde keine Bedeutung mit ihnen.“[43]

Aus seinem Argumentationslauf schlussfolgert Searle, dass die bloße regelkonforme Manipulation von Symbolen kein Indiz für menschliches Denken oder Verstehen sei:

> „Wenn ich kein Chinesisch verstehe, indem ich lediglich ein Computerprogramm zum Verstehen von Chinesisch ausführe, dann tut das auch kein Digitalcomputer. Solche Maschinen hantieren nur mit Symbolen gemäß den im Programm festgelegten Regeln. […] Das bloße Hantieren mit Symbolen genügt nicht für Fähigkeiten wie Einsicht, Wahrnehmung, Verständnis oder Denken. Und da Computer ihrem Wesen nach Geräte zur Manipulation von Symbolen sind, erfüllt

[43] Searle 1990, S. 40f.

> das bloße Ausführen eines Computerprogramms auch nicht die Voraussetzungen einer geistigen Tätigkeit.“[44]

Wie Dresler anmerkt, akzeptiere Searle die Position der schwachen KI und halte „den menschlichen Geist durchaus für *simulierbar* und es somit für möglich, dass die KI das *richtige* Programm mit dem *richtigen* Input-Output-Verhalten konstruieren kann.“[45] Searle lehne jedoch die Ansprüche und Implikationen ab, die jener Simulation des menschlichen Geistes von der starken KI beigemessen würden.[46] Wenngleich Dresler darauf hinweist, dass Searles Unterscheidung zwischen schwacher und starker KI bei Forscherinnen und Forschern der Fachdisziplin *Künstliche Intelligenz* als irrelevant angesehen würde,[47] verorten Russell und Norvig die Debatte um schwache vs. starke KI in einer pragmatisch-ethischen Sphäre:

> „Die meisten KI-Forscher nehmen die schwache KI-Hypothese als gegeben hin und kümmern sich nicht um die starke KI-Hypothese – solange ihre Programme funktionieren, ist es ihnen egal, ob sie es als Simulation der Intelligenz oder als echte Intelligenz bezeichnen. KI-Forscher sollten sich jedoch auch mit den ethischen Auswirkungen ihrer Arbeit beschäftigen.“[48]

2.2.3 Künstliche Intelligenz und das Konzept der Rationalität

„Solange ihre Programme funktionieren“ – Russell und Norvig zufolge finden sich jene Interpretationsangebote künstlicher Intelligenz, die sich mit der Abbildung menschlichen Denkens und menschlichen Handelns beschäftigen, Deutungsweisen gegenübergestellt, die das Konzept der Rationalität fokussieren. „Ein System ist rational“, wie die Autoren ausführen, „wenn es das seinen Kenntnissen entsprechende ‘Richtige‘ macht.“[49] Im Lichte dieses

[44] Searle 1990, S. 41.
[45] Dresler 2009, S. 63 (Kursivschrift nach dem Autor).
[46] Vgl. Dresler 2009, S. 63. Zur Kritik des Gedankenexperimentes vgl. die entsprechenden Kapitel in Dresler 2009 sowie die Arbeiten Rapaport 1988, Rheinwald 1991 und besonders Rapaport 2000, S. 472–475.
[47] Dresler 2009, S. 66.
[48] Russell und Norvig 2012, S. 1176.
[49] Russell und Norvig 2012, S. 22.

Rationalitätsbegriffes [50] betrachtet, verfährt ein System sinnvoll und effektiv, wenn es die Anforderungen erfüllt, die an das System gestellt werden – und nicht, wenn es sich möglichst menschenähnlich in seinen Problemlösungsverfahren geriert. Der vorgetragene Rationalitätsbegriff stellt dabei das Konzept des Agenten in den Mittelpunkt – Agenten, die sich in ihrer Mit- und Umwelt rational verhalten. Agenten, die in einem Multiagentenkontext miteinander kommunizieren, interagieren, kooperieren oder in Wettstreit gegeneinander treten, um ihre Leistung zu maximieren, sprich: die an sie gestellten Anforderungen zu erfüllen.

Das von Russell und Norvig vorgetragene Konzept des Agenten lässt sich auf Wolfgang von Kempelens schachspielenden Automaten übertragen: Situiert in seiner Umgebung, seiner Mit- und Umwelt, verfügt der Agent *Schachtürke* über Mechanismen, um seine Außenwelt wahrzunehmen und auf sie einzuwirken. Das vom Schachautomaten gespielte Schachspiel eröffnet hierbei eine mannigfaltige Anforderungswelt. So muss das Spiel und sein Verlauf zunächst über Sensoren sinnlich wahrgenommen und muss mitunter eine interne Repräsentation der Spielwelt erstellt und fortlaufend aktualisiert werden. Eigene Zugmöglichkeiten werden anschließend analysiert, Strategien entwickelt, angepasst und überdacht und das Verhalten gegnerischer Agenten antizipiert, so dass sich das durch Aktuatoren realisierte Verhalten eines Agenten in der Spielwelt endlich als *schlau*, *geschickt* oder *intelligent* signifizieren lässt.

[50] Vgl. auch den ergänzenden Hinweis der Autoren auf die Unterscheidung zwischen menschlichem und rationalem Verhalten: „Wir wollen darauf hinweisen, dass mit der Unterscheidung zwischen *menschlichem* und *rationalem* Verhalten nicht etwa ausgedrückt werden soll, dass Menschen notwendigerweise ‘irrational‘ im Sinne von ‘emotional instabil‘ oder ‘geistesgestört‘ sind. Man muss nur bedenken, dass wir nicht perfekt sind: Wir sind nicht alle Schachgroßmeister, auch nicht diejenigen von uns, die alle Schachregeln beherrschen […]“ (Russell und Norvig 2012, S. 18, Kursivschrift nach den Autoren). Siehe ferner auch die Erläuterung rationalen Handelns bei Neumann und Morgenstern: „The individual who attempts to obtain these respective maxima is also said to act ‘rationally.‘ But it may safely be stated that there exists, at present, no satisfactory treatment of the question of rational behavior. There may, for example, exist several ways by which to reach the optimum position; they may depend upon the knowledge and understanding which the individual has and upon the paths of action open to him” (Neumann und Morgenstern 2004, S. 9). Jenseits der Spieltheorie ist es mitunter jedoch klug und sinnvoll, eine weniger optimale Entscheidung zu treffen, um den Gegenspieler bzw. die Gegenspielerin zu verwirren oder zu verunsichern. Die Spieltheorie blendet solche Fälle jedoch zunächst aus, wie auch Davis betont: „Spieltheoretiker sind nicht daran interessiert, die Dummheit ihrer Gegner auszunutzen. […] Der Spieltheoretiker macht die pessimistische und häufig unrealistische Annahme, daß sein Gegner fehlerfrei spielen werde“ (Davis 2005, S. 20).

2.3 Intelligente Agenten

Wenngleich sich kein eindeutiger Konsens über das Konzept des Agenten und seine Bestandteile konstatieren lässt, [51] soll ein Agent im Kontext der Forschungsdisziplin der künstlichen Intelligenz zuerst deskriptiv angenähert sein, um die nachfolgende Definition des Agenten zu grundieren. Wie Šalamon sowie Padgham und Winikoff vorschlagen, lässt sich ein Agent über die folgenden Eigenschaften charakterisieren:[52]

▪ Ein Agent ist eingebettet in eine Umgebung.	Kontext
▪ Ein Agent ist selbst- und nicht fremdbestimmt, ein Agent ist autonom.	Autonomie
▪ Ein Agent nimmt seine Umgebung wahr.	Wahrnehmung
▪ Ein Agent reagiert zeitnah mit einem definierten Verhalten auf Änderungen in seiner Agentenumgebung.	Handlungsvermögen
▪ Ein Agent trifft Entscheidungen; er wartet mitunter nicht darauf, zum Agieren an die Reihe genommen zu werden.	Entschlusskraft
▪ Ein Agent verfolgt seine ihm eigenen Ziele. ▪ Ein Agent verfügt über unterschiedliche Wege und Möglichkeiten, seine Ziele zu erreichen.	Zielorientiertheit
▪ Ein Agent vermag mit anderen Agenten zu interagieren.	Kommunikation

Wie Šalamon anmerkt, ist die obige Annäherung an den Agentenbegriff über seine Eigenschaften eine sehr weitläufige, die jedoch den Vorteil bietet, vor einer eingehenden Durchdringung des Agentenkonzeptes zuerst zu differenzieren, welche Arten von Agenten sich ausfindig machen lassen. So unterscheidet

[51] Vgl. u.a. Macal und North 2009, S. 87, Padgham und Winikoff 2004, S. 1 sowie Wooldridge 2009, S. 21.
[52] Vgl. Šalamon 2011, S. 27 sowie Padgham und Winikoff 2004, S. 1.

Šalamon physische Agenten, natürliche Agenten und softwarebasierte Agenten.[53] Physische Agenten manifestieren sich als haptisch wahrnehmbare künstliche Entitäten, die in der menschlichen Wahrnehmungswelt existieren. Physische Agenten, das sind Roboter, intelligente Sensoren, Sonden – oder von Kempelens S*chachtürke*. Natürliche Agenten wie Menschen oder Tiere dienen oftmals als Vorlage für die Umsetzung physischer oder softwarebasierter Agenten. Softwareagenten sind in einer Programmiersprache verfasste Entitäten mitsamt ihren Wahrnehmungs- und Arbeitsdaten sowie Zugriffsmethoden auf ihre Daten.

Den referierten Erscheinungsweisen von Agenten sei mit Russell und Norvig sowie Wooldridge die Definition eines Agenten beigegeben: Ein Agent ist ein System, das sich in einer Umgebung befindet und dazu fähig ist, selbstständig und eigenverantwortlich Handlungen zu vollziehen, um individuell relevante Ziele zu verfolgen und zu erreichen. Seine Umgebung nimmt der Agent wahr über Sensoren; Handlungen in der Umgebung des Agenten vollziehen sich durch Aktuatoren.[54]

Das Konzept des Agenten, wie es sich in der vorgetragenen Definition manifestiert, vereint sowohl artifizielle, d.h. physische und softwarebasierte, als auch natürliche Systeme unter seinem Begriff. So verarbeitet ein menschlicher Agent Daten, die der sinnliche Wahrnehmungsapparat an das menschliche Gehirn liefert und agiert, eingebettet in seine oder ihre Umwelt, durch Stimme und Sprache, Hände und Füße und weitere Aktuatoren mit seiner Außen- und Innenwelt. Der Systemcharakter gründet sich im modularen Aufbau von Agenten: Miteinander verknüpfte und interagierende Module formen das System *Agent*, wie in der folgenden Agentenskizze illustriert.[55]

[53] Vgl. Šalamon 2011, S. 23f.
[54] Die referierte Definition vereint die definitorischen Ausführungen in Russell und Norvig 2012, S. 60 und Wooldridge 2009, S. 21.
[55] Die Agentenskizze ist – in abgewandelter Form – wiedergegeben nach Wooldridge 2009, S. 22.

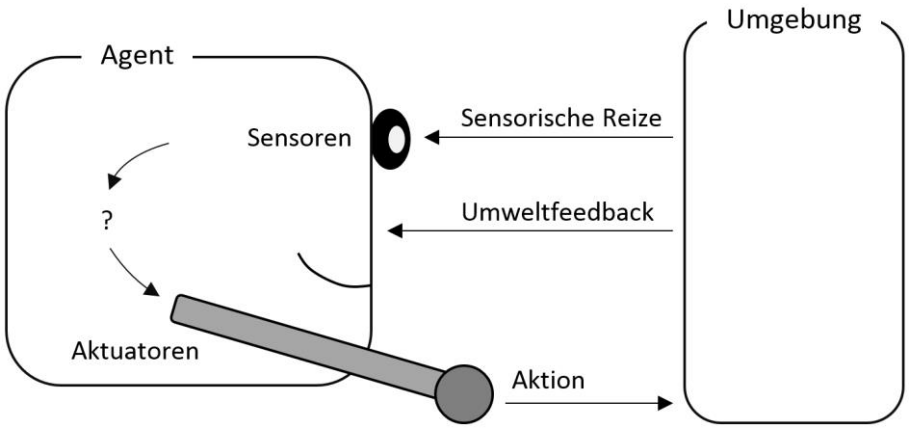


Abbildung 4: Modell eines basalen Agenten: Über seine Sensoren (beispielsweise eine Kamera) nimmt der Agent seine Umwelt wahr und wirkt auf sie ein mit Hilfe seiner Aktuatoren, die auch als *Effektoren* bezeichnet werden. Das Fragezeichen im Inneren des Agenten symbolisiert interne Prozesse, die Agentenaktionen aus sensorisch gewonnenen Daten generieren.

Die Herausforderung in der Modellierung und Implementation artifizieller intelligenter Agenten besteht darin, jene einzelnen Einheiten des Systems zu entwickeln und die Interaktion der Agentenbestandteile zu spezifizieren, um Agenten zu realisieren, die ihre Umwelt wahrnehmen und autark auf sie einwirken können. So muss die Rastergrafik zunächst aufwändig analysiert werden, die ein artifizieller Agent von seinem Wahrnehmungsapparat *Kamera* erhält, so müssen geometrische Figuren, müssen Muster ausfindig gemacht und extrahiert werden – und müssen wahrgenommene Objekte verarbeitet und mit weiteren Daten und Informationen verknüpft werden, die dem Agenten zur Verfügung stehen, um Prozesse anzustoßen, die zweckgemäßes Agieren hervorbringen.

2.3.1 Agentenfunktion und Agentenautonomie

Wie ein Agent handelt, ist durch die Agentenfunktion beschrieben: Die Agentenfunktion bildet die Wahrnehmungsfolge des Agenten – aktuell Wahrgenommenes und in vergangenen Momenten Perzipiertes und Apperzipiertes – auf eine Aktion ab. Realisiert ist die Agentenfunktion durch ein Agentenprogramm, wie Russell und Norvig näher differenzieren:

> „Die Agentenfunktion ist eine abstrakte mathematische Beschreibung; das Agentenprogramm ist eine konkrete Implementierung, die innerhalb eines physischen Systems ausgeführt wird.“[56]

Das Vermögen, Handlungen selbständig und eigenverantwortlich zu verfolgen und zu vollziehen, gar die Auswirkungen der eigenen Handlungen auf die Umwelt zu reflektieren und künftige Handlungen mit empirischen Daten abzugleichen, jene Autonomie des Computersystems *Agent* erschließt sich Wooldridge zufolge in einer Bandbreite individueller Selbstbestimmung:[57] Auf der einen Seite des Spektrums der Mensch mit seiner Freiheit, eigene Überzeugungen, Ziele und Handlungen zu generieren, zu ändern und zu reflektieren, auf der anderen Seite des Spektrums findet sich der triviale Softwareservice, der auf Anfragen nach einem vordefinierten Handlungsschema reagiert. Autonom agiert ein Agent, wenn er Wahrgenommenes in Handlung umsetzt, dabei Vorwissen um sein Dasein in und sein Sosein mit seiner Umwelt aktiviert und zu lernen fähig ist, auch in unsicheren Situationen richtig handeln.[58] Richtiges Handeln eines Agenten ist – zunächst und auch im weiteren Verlaufe der vorliegenden Ausarbeitung – kein moralisch gefärbtes Handeln, vielmehr ein Handeln, das die Bestimmung des in seine Umwelt eingebetteten Agenten erfüllt. Seine Bestimmung, rational zu agieren.

[56] Russell und Norvig 2012, S. 60.
[57] Vgl. Wooldridge 2009, S. 22f.
[58] Vgl. Russell und Norvig 2012, S. 60.

2.3.2 Agentenumwelt

> „Ich bin der Ansicht, dass die Welt sich aus tausenderlei, um nicht zu sagen, aus einer Unendlichkeit von Möglichkeiten zusammensetzt. Und die Auswahl ist zu einem gewissen Grade den die Welt strukturierenden Individuen anheimgestellt. Die Welt ist ein aus kondensierten Möglichkeiten bestehender Kaffeetisch.“[59]

„Die Welt ist ein aus kondensierten Möglichkeiten bestehender Kaffeetisch“ – jene Aussage, die der Protagonist des Romanes *Hard-boiled Wonderland und das Ende der Welt* gleich zu Beginn der Erzählung Haruki Murakamis tätigt, ist zum einen vorausdeutend für die Selektion der Möglichkeiten, der Auswahl von – hier anmutend schönen, dort abstoßend verstörenden – Handlungs- und Daseinsräumen, welche die Agierenden des Romans treffen – freilich, soweit der Autor seine und die Autorin ihre Individuen autark in die Welt setzt und ihnen die Befähigung zu agieren bietet. Der Prozess der Kondensation bindet potenziell Mögliches, manifestiert das Gewusste, Gedachte, Intendierte, macht Handlungsräume zu Handlungswelten, deren Komplexität einzig durch die Imagination sowohl des Autors als auch der Leserin und des Lesers beschränkt scheint.

Am Ende der Erzählung steht schließlich die Frage nach Identität, ist sich der Protagonist der Grenzen seiner Welt sowohl körperlich als auch kognitiv bewusst nach einer langen, sowohl geistigen als auch materiellen Reise:

> „Ich kann jetzt nirgendwo mehr hin und zu nichts mehr zurück. Hier ist das Ende der Welt, und von hier führt kein Weg mehr irgendwohin. Hier tut die Welt ihren letzten Atemzug und steht still.“[60]

Die schließende Stille erfüllt die Aufgabe des Autors, eine Handlung zu berichten, imaginäre Räume zu Erzählungen zu verdichten. Die Fähigkeit, Welten zu

[59] Murakami 2007, S. 12.
[60] Murakami 2007, S. 503.

erschaffen, Geschichten zu erzählen und autordeterminierte Individuen in eine Haltung gegenüber einer fiktiven Welt zu setzen, ist freilich nicht schwarzen Drucklettern auf weißem Papier vorbehalten. Besonders Computer- und Videospiele entwerfen Aktions- und Handlungsräume, bringen erdachte Welten pixelweise zur Existenz, hauchen Agenten Wahrnehmungsvermögen und künstliche Intelligenz ein und fordern Spielerinnen und Spieler dazu auf, sich in Relation zu Spielsubjekten zu setzen, sich mit- und gegenüber Spielsubjekten und Spielobjekten zu verhalten.

Der Spiel- und Handlungsraum, die Aufgabenumgebung, in den sich Agenten eingebettet finden und in dem sie wahrnehmen und handeln, lässt sich über die im Folgenden wiedergegebenen Eigenschaften differenzieren und beschreiben.[61]

- Vollständig beobachtbar vs. teilweise beobachtbar: Eine Aufgabenumgebung ist vollständig beobachtbar, wenn die Sensoren eines Agenten „ihm zu jedem beliebigen Zeitpunkt Zugriff auf den vollständigen Zustand der Umgebung bieten“[62]. Eine Umgebung ist teilweise beobachtbar, wenn der Sensorik des Agenten nur unvollständige Informationen über seine Umwelt vorliegen. Vollständig beobachtbare Aufgabenumgebungen sind der Implementation des Agenten vorteilhaft, da der Agent hierbei keinen internen Zustand über seine Umwelt speichern und verwalten muss, um die Welt zu beobachten.[63]
- Deterministisch vs. stochastisch: Eine Aufgabenumgebung wird als deterministisch bezeichnet, wenn „der nächste Zustand der Umgebung vollständig durch den aktuellen Zustand und die durch den Agenten ausgeführten Aktionen festgelegt wird“[64]. Im anderen Falle wird die Umgebung der Kunst der Vermutung anheimgegeben und als stochastisch bezeichnet. Nichtdeterministisch ist eine Aufgabenumgebung, wenn den Ergebnissen der Agentenhandlungen keine Wahrscheinlichkeiten zugeordnet sind.

[61] Vgl. das Kapitel *Agent environments* in Šalamon 2011, S. 21–23 sowie Wooldridge 2009, S. 24 f.; Russell und Norvig 2012, S. 69–73.
[62] Russell und Norvig 2012, S. 69.
[63] Vgl. Russell und Norvig 2012, S. 69.
[64] Russell und Norvig 2012, S. 70.

- Episodisch vs. sequenziell: In sequenziellen Umgebungen haben die von einem Agenten zu einem Zeitpunkt t getroffenen Entscheidungen und Handlungen kausalen Einfluss auf den Zustand der Umwelt des Agenten zum Folgezeitpunk t_{+1} , somit folglich auch auf die zu treffenden Entscheidungen und die zu vollziehenden Handlungen des Agenten, die der Agent zum Zeitpunkt t_{+1} zu treffen vermag. In einer episodischen Umgebung „ist die Erfahrung des Agenten in atomare Episoden unterteilt“ [65] . Jedes Atom der Episodenfolge besteht aus einer Wahrnehmung und einer Aktion des Agenten. Folgende Episoden sind unabhängig von früheren Episoden.
- Statisch vs. dynamisch: In der Differenzierung zwischen statischer und dynamischer Umgebung ist der Zeitpunkt der Entscheidung des Agenten wesentlich: Vermag sich die Umgebung während des Entscheidungsprozesses des Agenten zu ändern, so wird die Umgebung als dynamisch bezeichnet; ändert sich die Umgebung während der Entscheidung des Agenten nicht, so wird die Umgebung als statisch signifiziert. Ändert sich die Umgebung des Agenten nicht, wohl jedoch seine Leistungsbewertung, so wird die Umgebung als semidynamisch bezeichnet.
- Diskret vs. stetig: Eine Umgebung wird als diskret bezeichnet, wenn sie zu einem Zeitpunkt t, sowie zum Folgezeitpunkt t_{+1} eine endliche Anzahl von Handlungsmöglichkeiten sowie Wahrnehmungsoptionen für den Agenten eröffnet.
- Bekannt vs. unbekannt: Muss der Agent in einer (ihm) unbekannten Aufgabenumgebung zunächst lernen, wie er sich in der Welt verhalten kann und verhalten muss, um seine Leistung zu maximieren, so sind ihm in einer bekannten Umgebung die Folgen seiner Handlungen offenbar.
- Einzelagent- vs. Multiagentenumgebung: Die Distinktion zwischen Einzelagent- und Multiagentenumgebung fokussiert die Auffassung des Agenten über den Anderen in der mittelbaren und unmittelbaren Umgebung des Agenten. Nimmt der Agent in der Einzelagentenumgebung

[65] Russell und Norvig 2012, S. 71.

andere Entitäten seiner Umwelt als jene Umwelt beeinflussenden Wahrscheinlichkeiten wahr, so findet er sich eingebettet in eine Einzelagentenumgebung. Attribuiert der Agent anderen Entitäten seiner Umwelt, Agenten zu sein, das eigene, individuelle Verhalten mit- und gegen die Umwelt mitunter durch die zuvor angedeutete Nutzenfunktion zu orientieren und zu determinieren, so findet sich der Agent eingebettet in eine Mitwelt von Agenten, eine Multiagentenumgebung.

2.3.3 Brettspielumgebungen

Die vorliegende Arbeit beschäftigt sich mit der Frage, wie sich artifizielle Agenten realisieren lassen, die ohne menschliche Expertise in unterschiedlichsten Spielumgebungen sinnvolle Entscheidungen zu treffen vermögen. Die Wahl des Beschäftigungsgegenstandes klassischer Brettspiele gründet in dem besonderen Charakter der Spiele: Klassische Brettspiele konstituieren sich zumeist aus wenigen einfachen Spielregeln, die trotz ihrer Simplizität hochkomplexe Entscheidungsräume zu generieren vermögen. Entsprechend den zuvor referierten Differenzierungen sind Brettspielumgebungen wie die des Schach-, Dame-, Backgammon- oder Gomoku-Spieles, bei dem gewinnt, wer fünf Spielsteine in Folge auf dem Spielbrett abgelegt hat, vollständig beobachtbar, deterministisch oder stochastisch und nehmen einen sequenziellen Verlauf. Die betrachteten Brettspiele ändern sich während der Zugentscheidung des Agenten nicht – sie sind statisch – und sie bieten den Agenten zu einem bestimmten Zeitpunkt des Spielverlaufes eine endliche Anzahl von Zugmöglichkeiten. Zwar ist dem brettspielenden Agenten seine Aufgabenumgebung bekannt, welche Handlung das Spiel zu einem individuell positiven Verlauf führt, ist dem Agenten zunächst jedoch fremd und gestaltet sich als Ergebnis eines Such- und Lernprozesses. Die Brettspielwelten der vorliegenden Ausarbeitung gestalten sich zum einen als konkurrierende Multiagentenumgebung, in der jeder Agent bestrebt ist, das Spiel für sich zu entscheiden. Zum anderen jedoch gründet jede Handlungsentscheidung eines Agenten in dem Zusammenspiel spezialisierter Agenten, die unterschiedliche Aspekte des Spieles und Spielverlaufes betrachten und gemeinsam über sinnvolle Handlungen entscheiden.

2.3.4 Aufbau und Struktur von Agenten

Die Art, wie ein Agent mit seiner Umwelt interagiert – und interagieren sollte –, wurde in den vergangenen Kapiteln mit dem Kriterium der Rationalität spezifiziert: Ein rationaler Agent ist bestrebt, die Erwartungen zu erfüllen, die an ihn gestellt werden. Ein rational handelnder Agent intendiert, seine Leistungsbewertung zu maximieren unter Berücksichtigung aktueller und vergangener Wahrnehmungen, sowie unter Aktivierung seines Wissens um seine Umwelt und den Regeln, die in der Agentenumwelt gelten. Die Agentenfunktion determiniert hierbei, wie sich der Agent in seiner Umwelt verhält. Implementiert ist die Agentenfunktion durch das Agentenprogramm, das gemeinsam mit der Architektur das System Agent konstituiert: Stellt die Architektur des Agenten Zugriffsmöglichkeiten auf die Umwelt (Sensoren) und Mechanismen bereit, um mit der Umwelt zu interagieren (Aktuatoren), so verarbeitet das Agentenprogramm die eingehenden Wahrnehmungsdaten und generiert Handlungen gemäß der Agentenfunktion.[66] Im vorliegenden Falle der webbasierten Implementation von Brettspielen und Brettspielagenten ist die Architektur der Agenten vollständig artifiziell: Sensoren und Aktuatoren des Agenten sind in der Skriptsprache JavaScript softwareseitig als Agentenmodule implementiert, deren Aufbau und Verknüpfung miteinander in Kapitel 6.1 besprochen wird.

Agentenprogramme und Agentenarchitekturen lassen sich so vielfältig gestalten wie die Aufgabenumgebungen, in denen die Agenten eingesetzt werden sollen. So verfügen trivial anmutende Agentenprogramme über eine Tabelle, in der Handlungsreaktionen unterschiedlichen Wahrnehmungen des Agenten zugeordnet sind. Die Tabelle repräsentiert hierbei die Agentenfunktion, die durch das Agentenprogramm umgesetzt wird.[67] Tabellenbasierte Ansätze lassen sich zwar leicht implementieren, sind der Praxis jedoch mitunter undienlich, weil sie keine Flexibilität bieten, auf unterschiedliche Anforderungen zu reagieren: Zum einen sind die meisten Aufgabenumgebungen allzu komplex, um jeden diskretisierten Zeitpunkt im Leben des Agenten auf eine Tabellenzeile abzubilden,

[66] Vgl. Wooldridge 2009, S. 73 f.
[67] Vgl. Russell und Norvig 2012, S. 74 f.

zum anderen muss jede einzelne Wahrnehmung des Agenten gespeichert werden, um Handlungen auf Wahrnehmungsfolgen abbilden zu können – ein äußerst speicherintensives Unterfangen.

Unterschiedliche Agentenarten und Agentenprogramme – von einfachen Reflexagenten bis hin zu lernenden Agenten –, wie sie Russell und Norvig und Wooldridge vorstellen, bieten elaborierte Blaupausen in der Implementation von Agenten:

- Einfache Reflexagenten berücksichtigen bei ihren Handlungen keine vergangenen Wahrnehmungen, sondern reagieren einzig auf aktuell Wahrgenommenes. Formal beschreiben lässt sich ein einfacher Reflexagent als eine Agentenfunktion, die jeder Wahrnehmung eine Handlung zuordnet: Liefert die Umwelt zu einem Zeitpunkt t ein bestimmtes Wahrnehmungsdatum, so reagiert der Agent auf den Umweltimpuls unter Verwendung präformulierter Handlungsregeln. Jene Regeln gestalten sich als triviale „wenn-dann“ Regeln.[68]
- Im Unterschied zu einfachen Reflexagenten verfügen modellbasierte Reflexagenten zum einen über eine Vorstellung darüber, wie die Umwelt des Agenten beschaffen ist – den aktuellen Zustand der Umwelt –, zum anderen ist ihnen bewusst, wie der nächste Zustand ihrer Umwelt aus möglichen Interaktionen mit dem derzeitigen Zustand der Umwelt resultiert. Das Wissen darüber, „wie die Welt funktioniert“[69] wird als Modell der Welt bezeichnet.
- Gegenüber einfachen und modellbasierten Reflexagenten berücksichtigen zielbasierte Agenten zukünftige, mit Absicht hervorzubringende Zustände in ihren Handlungen. Bilden die zuvor betrachteten einfachen und modellbasierten Reflexagenten einzig Aktionen auf Wahrnehmungen ab, so entscheiden zielbasierte Agenten unter Berücksichtigung individuell zu erreichender Ziele, welche Handlung im Entscheidungsmoment sinnvoll ist, um den gewünschten Zustand, das Ziel zu erreichen.

[68] Vgl. Russell und Norvig 2012, S. 76.
[69] Russell und Norvig 2012, S. 78.

- Stellen die individuellen Ziele eines Agenten einzig eine „grobe binäre Unterscheidung“ zwischen den zwei Zuständen „erfüllt“ und „nicht erfüllt“ dar, wie Russell und Norvig anmerken,[70] so ermöglicht das Konzept des Nutzens feinere Differenzierungen in der Bewertung unterschiedlicher Agentenzustände und Agentenziele: Das Verhalten nutzenbasierter Agenten ist charakterisiert durch eine Nutzenfunktion, die unterschiedlichen Zielen des Agenten unterschiedliche Werte zuordnet. Somit lassen sich wünschenswerte Zustände und Ziele des Agenten von weniger wünschenswerten unterscheiden.

Wiedergegeben in aufsteigender Komplexität, stellen die zuvor besprochenen Agententypen zunächst und für sich betrachtet statische Systeme dar: Welche Reaktion auf eine bestimmte Wahrnehmung folgt, inwiefern das eine Ziel des Agenten sinnvoll und das andere Ziel weniger sinnvoll zu verfolgen ist und welchen Nutzen der Agent aus seinen Handlungen zieht, das muss häufig zuallererst vor aller Erfahrung, gar vor aller Erfahrungsmöglichkeit des Agenten manuell festgelegt werden – und das verbleibt im Agenten als sein unerschütterliches Handlungs- und Reaktionsmuster. Weil einfache und modellbasierte Reflexagenten dem Anspruch nicht genügen können, geschicktes Spielverhalten in komplexen Brettspielen zu generieren – eine vollständige Liste, die im Schachspiel jeder Aktion der Gegnerin in jedem Spielzustand eine optimale Gegenaktion zuzuordnen vermag, würde sich als äußerst speicherintensiv gestalten –, seien im Folgenden nutzenbasierte lernfähige Agenten gemeint, wenn von *Agenten* die Rede ist.

2.3.5 Lernvermögen

Dynamisch wird das statische System Agent, indem es angereichert wird durch Mechanismen, die es dem Agenten ermöglichen, intendierte Handlungen abzugleichen mit zuvor Erlebtem und für als sinnvoll erachteten Handlungsmustern. So ergänzt der Modulkomplex des Lernens die in den vergangenen Kapiteln betrachteten Konzepte um die notwendige Flexibilität in der Implementation intelligenter Agenten. Wie die folgende Skizze eines

[70] Vgl. Russell und Norvig 2012, S. 81.

allgemeinen lernenden Agenten illustriert, erfährt sich der Agentenentwurf ergänzt um drei Module, die ein Lernverhalten des Agenten ermöglichen. So ist das Modul des zuvor eingehend betrachteten Leistungselementes, das für die Auswahl rationaler Handlungen zuständig ist, ergänzt um die Kritik- und Lernelemente und den Problemgenerator.[71]

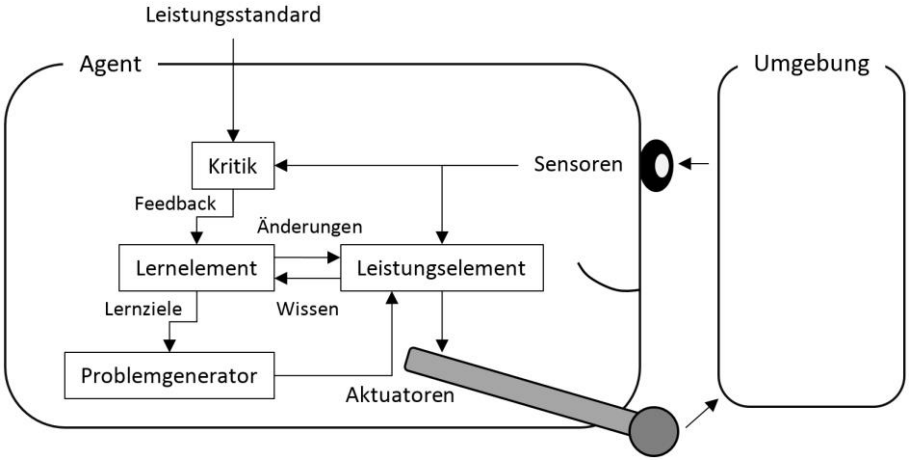


Abbildung 5: Skizze eines allgemeinen lernenden Agenten nach Russell und Norvig.

Orientiert an einem extern formulierten Leistungsstandard wird die Wahl des Leistungselementes, welche Handlung auszuführen ist, beeinflusst durch das Lernelement. Das Lernelement berücksichtigt Rückmeldungen des Kritikelementes, um die Zahnräder des Leistungselementes dynamisch zu justieren. Experimentierfreude ist durch das Element des Problemgenerators vorgesehen: Anstatt immer dieselben für gut befundenen Handlungen auszuführen, schlägt der Problemgenerator alternative Wege zur Lösung eines Problems vor.

Dem Problem- und Anforderungskomplex der Implementation schlau handelnder Agenten in Brettspielen ist das Vermögen unentbehrlich, aus vollzogenen Spielhandlungen zu lernen – ein Agent, der sich immer gleich verhält und immer dieselben Fehler macht, aus vergangenem negativen Feedback folglich nicht zu lernen vermag, ließe sich nur schwerlich als intelligent bezeichnen und erfüllte wohl kaum die an ihn gestellte Forderung der Rationalität. In Brettspielen wie Schach oder Go jedoch optimale Entscheidungen zu identifizieren, stellt sich ob

[71] Die Agentenskizze des allgemeinen lernenden Agenten ist wiedergegeben nach Russell und Norvig 2012, S. 83 f.

der Komplexität der Brettspiele als äußerst schwieriges Unterfangen dar: Konstituiert durch wenige einfache Spielregeln, generieren die betrachteten Spiele außerordentlich komplexe Spielverläufe, in denen die Folgen früher individueller Handlungen schwer abzuschätzen sind – sowohl für menschliches als auch für agentenbasiertes Spielverhalten.

2.4 Brettspiele, Drosophila künstlicher Intelligenz

> "If computers could play chess better than any human, would they be 'intelligent'?"[72]

Eine Vielfalt von Entscheidungsmöglichkeiten auf den ersten, eine ausweglose Situation auf den genaueren zweiten Blick: Weiß ist an der Reihe und kann mit nur einem Zug die im Folgenden wiedergegebene Schachpartie für sich entscheiden – doch wie sollte Weiß agieren, um Schwarz möglichst schnell und effektiv Matt zu setzen? Wie würde Weiß *intelligent* handeln?

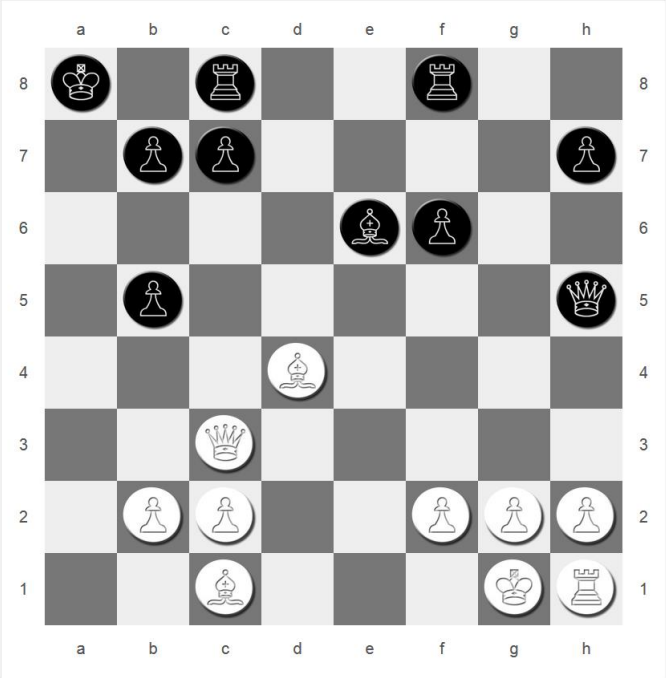


Abbildung 6: Schachmatt in zwei Zügen.

Der Auflösung der Fragen mag mit einem Hinweis gedient sein: Es ist die weiße Dame auf dem Feld c3, die es mit einer Bewegung auf ein bestimmtes Spielfeld vermag, das Schachmatt gegen Schwarz vorzubereiten. Die eingangs mit Jonathan Schaeffer gestellte Frage jedoch, ob sich Computer als intelligent signifizieren ließen, die besser zu spielen vermögen als Menschen, sei zunächst empirisch angenähert. Was es bedeutet, intelligent zu spielen, erschließt sich bei klassischen Gesellschafts- und Brettspielen wie Dame, Schach, Backgammon oder Go aus der

[72] Schaeffer 2009, S. 10.

Erfahrung: Eine intelligent agierende Spielerin handelt überlegt; sie wägt ihre Zugmöglichkeiten gegeneinander ab und reflektiert, ob es sinnvoll ist, dem Gegenspieler eine Falle zu stellen, um das Spielgeschehen mit den nachfolgenden Spielzügen für sich zu entscheiden – all das leistet der menschliche Wahrnehmungsapparat und das menschliche Gehirn in Bruchteilen von Sekunden.

Signifikant für menschliche Intelligenz ist jedoch weniger die Fähigkeit, Abermillionen Runden der Spielpartie vorauszuberechnen, um eine Aussage über die Güte eines Spielzuges zu treffen, vielmehr zeichnet sich menschliche Intelligenz und menschliches Denken im Kontext des Spieles besonders durch seine hohe Flexibilität aus, auf unterschiedliche Spielsituationen kreativ zu reagieren und Mutmaßungen über sinnvolle und weniger sinnvolle Züge in endlicher Zeit anzustellen. Menschen verfahren intuitiv, lösen Spielprobleme kreativ und applizieren heuristische Methoden, um ihre im Vergleich zu Rechnern und Computern verhältnismäßig langsame Verarbeitungsgeschwindigkeit und geringe Speicherkapazität zu kompensieren.[73] Auf der anderen Seite der Gegenüberstellung vollziehen Computer unzählige Rechenoperationen in Nanosekunden, verfügen über schier unbegrenztes Speichervermögen und höchst leistungsfähige Zugriffsmethoden auf das memorierte Gut, sind jedoch menschlichem Schlussfolgern und menschlichem Abstraktionsvermögen, besonders jedoch menschlicher Kreativität[74] unterlegen.

Schon in ihren Gründungsjahren beschäftigte sich die aufkeimende Fachdisziplin der künstlichen Intelligenz mit klassischen Brettspielen und stellte früh das Schachspiel ins Zentrum des Forschungsinteresses. Gar als *Drosophila der künstlichen Intelligenz* bezeichnet,[75] war und ist das Schachspiel ob seiner

[73] Vgl. Mańdziuk 2010, S. XI ff.

[74] Die Definition von Kreativität in Prechtl 2008 ist der Bestimmung des Begriffes dienlich, auch wenn Definitionspartikel wie *Phantasie*, *Intuition* oder *Improvisation* an dieser Stelle nicht tiefergehend betrachtet werden sollen: „K[reativität] ist ein Phänomen, welches sich u.a. aus folgenden Komponenten zusammensetzt: Phantasie, Intuition, Improvisation, Originalität, Flexibilität, Inspiration, divergentes Denken. K[reativität] bringt schöpferische Leistungen, neue, originelle Problemlösungen (in Kunst, Wissenschaft, Technik und Gesellschaft usw.) hervor.“ (Prechtl 2008, S. 317).

[75] Die Fruchtfliege (*Drosophila*) offenbarte sich ob ihrer einfachen genetischen Struktur als sehr dienliches Untersuchungsobjekt der Genetik. Zur Metapher des Schachspieles als *Drosophila* der künstlichen Intelligenz vgl. den gleichnamigen Aufsatz „Is chess the drosophila of artificial intelligence? A social history of an algorithm” (Ensmenger 2012).

Charakteristika der Beschäftigung um die Implementation artifizieller Intelligenz essentiell: Schach ist vollständig durch seine wenigen Regeln beschrieben, die Spielregeln sind gemeinhin bekannt – und das menschliche Vermögen, im komplexen Schachspiel gegnerische Strategien zu identifizieren und vorausschauend den Spielverlauf zu planen, gilt gesellschaftlich zumeist als Indikator menschlicher Intelligenz. [76] Zwar hat das Forschungsinteresse am Schachspiel aufgrund der Erfolge kontinuierlich leistungsfähigerer Computerhardware und immer ausgefeilterer Algorithmen merklich nachgelassen und findet sich nun ersetzt durch das Interesse am Go-Spiel, das einen deutlich komplexeren Spielbaum als das Schachspiel zu generieren vermag – die Herausforderung in der Forschung um schlaue artifizielle Brettspielgegner bleibt jedoch dieselbe und formuliert sich in einer Frage, wie sie Schaeffer stellt: Wenn sich kein intelligentes Computerprogramm in solch einer einfachen Umgebung wie die des Schach- oder Damespieles entwickeln lässt, wie kann dann Hoffnung bestehen, komplexere Probleme in Angriff nehmen und lösen zu können?[77]

[76] Vgl. Ensmenger 2012, S. 18.
[77] Vgl. Schaeffer 2009, S. 10.

2.5 Methodendifferenzierung – Künstliche Intelligenz und Computational Intelligence

Allerspätestens mit Schaeffers im Jahre 2007 getätigten Ausspruch „Checkers is Solved“ [78], der die beachtliche Leistung der Gruppe von Forscherinnen und Forschern rund um Schaeffer und des seit 1989 entwickelten Dame-Programmes *Chinook*[79] treffend zusammenfasst, ist das erreicht, was Mańdziuk in seiner 2010 publizierten Arbeit *Knowledge-Free and Learning-Based Methods in Intelligent Game Playing* als „renaissance of mind game programming“[80] der vergangenen zehn bis fünfzehn Jahre bezeichnet: Computergegner spielen in Spielen, die den Einsatz geistiger [81] Fähigkeiten erfordern, nun häufig besser als menschliche Spielerinnen und Spieler, geschickter als menschliche Großmeisterinnen und Großmeister – und bewegen die eigenen Spielsteine im Falle des Dame-Programmes *Chinook* gar so gut, dass menschliche Spieler nur ein Unentschieden gegen das Programm erzielen können, wenn sie fehlerfrei spielen. – Aber lassen sich jene Computergegner, deren immense Stärke sich allzu häufig in eigens entwickelter Hard- und Software sowie äußerst umfangreichen Eröffnungs- und Endspieldatenbanken gründet, als *intelligent* bezeichnen?

Wie Kroll in seiner Arbeit *Computational Intelligence* einführend anmerkt, existiere keine allgemein anerkannte Definition des Begriffes *Intelligenz*, so dass sich die Annäherung an das Bezeichnete pragmatisch vollziehen lässt: „Intelligenz“, so Kroll, „[ist] die Fähigkeit […], Zusammenhänge zu finden, zu lernen, zu verstehen, Probleme zu lösen sowie Entscheidungen zu fällen.“ [82] Künstlich implementierte Spieler einen die Definitionspartikel; sie intendieren, Zusammenhänge auf dem Spielbrett zu identifizieren („WENN ich Spielfigur A auf

[78] Schaeffer et al. 2007.
[79] Vgl. die Homepage der Software unter http://webdocs.cs.ualberta.ca/~chinook/ (zuletzt erfolgreich aufgerufen am 21.06.2013).
[80] Mańdziuk 2010, S. 41.
[81] In seiner Arbeit lässt Mańdziuk offen, was er meint, wenn er von „mental skills“ schreibt (vgl. Mańdziuk 2010, S. 1). Weil es bei den folgenden Ausführungen um menschliches brettspielbezogenes Problem- und Lösungsdenken geht, wurde an dieser Stelle die Übersetzung „geistige Fähigkeiten“ vorgenommen. Zur Differenzierung der Begriffe *Geist*, *Seele*, *mind* und *mental* vgl. auch Beckermann 2008, S. 4.
[82] Kroll 2013, S. 8

ein bestimmtes Feld bewege, DANN ergibt sich für meine Spielfigur B eine verbesserte Situation“), sie lernen mitunter aus vergangenen Spielpartien und treffen Entscheidungen, die das Spielgeschehen möglichst positiv für sie gestalten.

Sind jene artifiziellen Optimalspieler zumeist hochspezialisiert auf ein einziges Spiel – einem Schachcomputer das Damespiel beizubringen wird in der Regel ein erfolgsloses Unterfangen bleiben – und nicht denk- und umsetzbar ohne höchst leistungsfähige dedizierte Computerhardware, so sind die zugrundeliegenden Algorithmen zumeist so allgemein formuliert, dass sie Lösungsverfahren für unterschiedliche Brett- und Gesellschaftsspiele bieten. Algorithmen zur Suche in Spielbäumen wie der Minimax-Algorithmus und seine Verfeinerungen sowie heuristische Verfahren, wie sie in späteren Kapiteln vorgestellt werden, eignen sich „perfekt“ dazu, künstliche Intelligenz in Brettspielen zu implementieren, wie Millington und Funge betonen.[83]

Wird optimales Problemlösungsverhalten der artifiziellen Spieler angestrebt, so dämpft die Praxis jedoch den euphorischen Ausspruch der beiden Autoren: Verfahren der *klassischen* KI wie der Minimax-Algorithmus oder das Verfahren der Alpha-Beta Kürzung skalieren zumeist äußerst schlecht mit Erhöhung der Komplexität ihres Anwendungsgegenstandes.[84] Jede hinzugefügte Spielfigur, jede Vergrößerung der Spielwelt steigert die Anzahl der zu berücksichtigenden und zu analysierenden Spieleigenschaften mitunter exponentiell.

Wo Algorithmen der klassischen KI in endlicher Zeit keine Lösung finden, bieten sich Konzepte und Mechanismen der Computational Intelligence an, im Folgenden mit CI abgekürzt, die an sie gestellten Probleme näherungsweise zu lösen. Wie Kruse näher ausführt, besteht die Problemlösungsstrategie der Computational Intelligence darin,

> „approximative Techniken und Methoden zu verwenden, die ungefähre, unvollständige oder nur partiell wahre Lösungen zu Problemen finden können, dies aber im Gegenzug in einem akzeptablen Zeit- und Kostenrahmen bewerkstelligen. Diese Ansätze bestehen aus relativ einfachen Teilabläufen, die

[83] Vgl. Millington und Funge 2009, S. 667.
[84] Vgl. Kruse et al. 2011, S. 2.

> im Zusammenspiel zu komplexem und selbstorganisierendem Verhalten führen. Dadurch entziehen sich diese heuristischen Verfahren oft einer klassischen Analyse, sie sind aber in der Lage, schnell ungefähre Lösungen zu ansonsten nur schwierig lösbaren Problemen zu generieren."[85]

Wenngleich Kroll darauf hinweist, dass keine allgemein anerkannte Definition der Computational Intelligence in der Fachwelt existiere[86] und die trennende Linie zwischen den Disziplinen nur eine feine ist, wie Mańdziuk anmerkt,[87] so ist der Gegenstand der CI klar definiert: Als Teilgebiet der künstlichen Intelligenz beschäftigt sich die Forschungsdisziplin der CI mit naturanalogen Methoden. Setzt sich die erweiterte CI mit Schwarmintelligenz und künstlichen Immunsystemen auseinander, so bilden die drei Gebiete *Künstliche Neuronale Netze*, *Fuzzy-Systeme* und *Evolutionäre Algorithmen* den Kern der CI, wie in der folgenden Grafik visualisiert:[88]

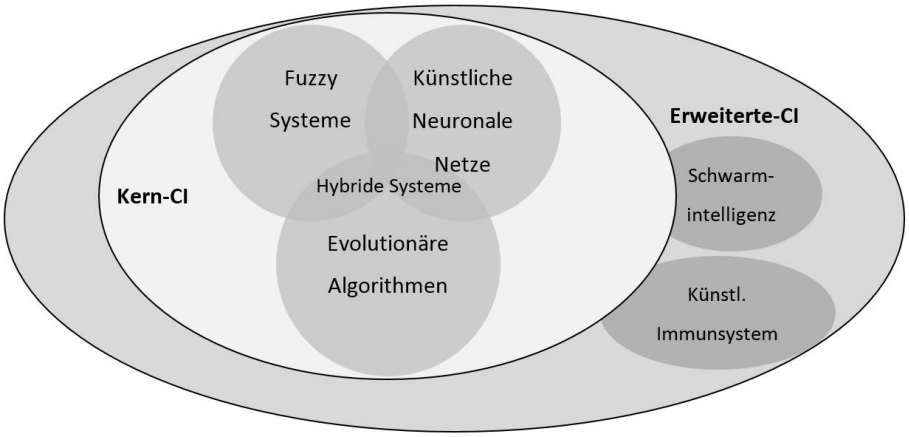


Abbildung 7: Forschungsgebiete der Computational Intelligence nach Kroll.

Die vorliegende Arbeit verfolgt einen agentenbasierten Ansatz zur Modellierung und Implementation schlauer Computergegner und schöpft Ideen und Methoden aus den vorgestellten Fachdisziplinen der KI und CI. Beide Beschäftigungsfelder bieten ihre individuellen Vor- und Nachteile, auf die in späteren Kapiteln eingegangen wird. So werden – nachdem in grundlegende Konzepte der Spieltheorie nach von Neumann und Morgenstern eingeführt wurde – mit dem

[85] Kruse et al. 2011, S. 2.
[86] Vgl. Kroll 2013, S. 12 und Klüver et al. 2012, S. 5.
[87] Vgl. Mańdziuk 2010, S. 4.
[88] Die Übersichtsgrafik ist wiedergegeben nach Kroll 2013, S. 12.

Minimax-Algorithmus und dem Verfahren der Alpha-Beta Kürzung zunächst klassische (Such)verfahren der Künstlichen Intelligenz betrachtet und daran anschließend Methoden der CI besprochen, die der raschen Entscheidungsfindung artifizieller Gegner in digitalen Brettspielen dienen.

2.6 Zusammenfassung und Ausblick

Die vergangenen Kapitel leiteten zunächst ein in das Forschungsgebiet der künstlichen Intelligenz und ihren Grundfragen. Wurden mit dem Turingtest und Searles *chinesischem Zimmer* zuerst Fokussierungsweisen künstlicher Intelligenz betrachtet und zwischen starker und schwacher KI differenziert, konzentrierte sich die Betrachtung auf Grundlage des von Russell und Norvig vorgetragenen Rationalitätsbegriffes auf das Konzept des intelligenten Agenten. Definiert als ein System, das sich in einer Umgebung befindet und dazu fähig ist, selbstständig und eigenverantwortlich Handlungen zu vollziehen, um individuell relevante Ziele zu verfolgen und zu erreichen, verfügt ein Agent über Sensoren, um seine Umgebung sinnlich wahrzunehmen und besitzt Aktuatoren, mit denen der Agent Handlungen in der Agentenumgebung vollzieht. Differenziert wurden die Eigenschaften verschiedener Arten von Umgebungen, in die sich der Agent eingebettet findet – und wurden abschließend unterschiedliche Arten von Agenten vorgestellt. Die Erscheinungsweisen der Brettspielumgebungen des Dame-, Schach-, Gomoku- oder Backgammonspieles, wie sie der vorliegenden Arbeit Gegenstand sind, wurden beschrieben als vollständig beobachtbare sequenzielle statische Multiagentenumgebungen, in denen sich lernfähige Agenten gegen- und miteinander verhalten, um das Spiel für sich zu entscheiden.

Gerierten sich Brettspiele als Katalysatoren und als Schleifstein für die Beschäftigung mit künstlicher Intelligenz, werden in Kapitel 4 Mechanismen der klassischen künstlichen Intelligenz und Verfahrensweisen der Computational Intelligence besprochen, die optimale Agentenentscheidungen in digitalen Brettspielen ermöglichen. Zuerst jedoch wird im folgenden Kapitel erörtert, was das ist, das *Spiel* und durch welche Bestandteile, Eigenschaften und Eigenheiten sich das Spiel auszeichnet. Ausgegangen von anthropologisch-phänomenologischen Definitionsvorschlägen, wie sie Huizinga und Juul leisten, wird mit der Spieltheorie nach von Neumann und Morgenstern anschließend das Handwerkszeug in der Beschäftigung mit der Entscheidungsfindung von Agenten in Spielumgebungen vorgestellt.

3 Von Huizinga nach Γ – Theorien des Spieles

> "Prudence: Well, tell me, Skepticus. What did the Grasshopper say about games?
> Skepticus: First he presented a definition of games or, to be more precise, a definition of game playing. Then he invited me to subject that definition to a series of tests. I was to advance against the definition the most compelling objections I could devise, and he was to answer those objections.
> P: And did the definition withstand your attacks?
> S: He was able, or so it seemed to me, to defend the definition against all of my challenges. [...]
> P: Why, Skepticus, it is almost as though he was –
> S: Playing a game with us?"[89]

Was ist das, dieses Phänomen *Spiel* (*game*), das der Grashüpfer im Dialog zwischen Prudence und Skepticus zum einen zu definieren vermag, zum anderen jedoch jene unterschiedliche alltagssprachliche Verwendung des Spielbegriffes exemplifiziert, die aus der Beschäftigung mit dem Spiel wiederum gespieltes Spiel (play) generiert. Was macht das Spiel zum Spiel? (Wann) Ist die

spie
lerische
Anordnung
von Symbolen,
schwarzen Lettern
auf weißem Untergrund
ein
Spi
el?

[89] Suits 2005, S. 34.

Auf welche Art, auf welche Weise generieren sich aus der Verknüpfung von Spielsymbolen Spieldaten, wie schöpft sich aus jenen Spieldaten – dem Spielbrett, dem Spielstein, dem Spielwürfel – Bedeutung, die aus Materiellem *ein* Spiel, *das* Spiel schafft? Deutungsangebote, die sich dem Spiel des Menschen definitorisch anzunähern versuchen, manifestieren sich in unterschiedlichen Spielarten: Ob die Erscheinungsweisen des Spieles fokussiert werden, oder ob das individuelle Empfinden der Spielerinnen und Spieler oder Fragen nach bestmöglichen Spielentscheidungen im Zentrum der Theoriebildung stehen – der Gegenstand vielfältiger Spieltheorien ist sich immer gleich und doch verschieden und entzieht sich ob seiner Vielfältigkeit und Vielgestaltigkeit häufig eindeutiger Bestimmungen.

3.1 Spielarten von Spieltheorien

Die Beschäftigung mit dem Phänomen Spiel und seinen Erscheinungs-, Interaktions- und Wirkungsformen, die Bestrebungen und Intentionen, bestimmte „Ursachen, Anlässe, Motive und Wirkungen des Spielverhaltens“ [90] aus dem Steinbruch des Alltäglichen herauszumeißeln und als Spiel zu identifizieren sind mannigfaltig und vielfältig. So referiert und benennt Neitzel unterschiedliche Spielarten von Spieltheorien, die differierende Aspekte in ihrer Betrachtung von *Spiel* fokussieren: Funktionalistische Ansätze ergründen der Autorin zufolge das Verhältnis von Spielerinnen und Spielern zum Spiel, anthropologisch-phänomenologisch fundierte Theorien konzentrieren sich auf Charakteristika und Merkmale, die dem Spiel zu eigen sind und die Spieltheorie nach von Neumann und Morgenstern beschreibt, wie Spielerinnen und Spieler innerhalb des Spielraumes agieren, Strategien entwickeln und Entscheidungen treffen.[91]

Wie Thimm und Wosnitza argumentieren, ist der Phänomenkomplex *Spiel* ein wandelbarer und erfährt durch die Abbildung in und mit Hilfe von Computern eine Transformation. Eine Transformation, die klassische Brettspiele und ihre Abbildung als Computerspiele weniger betrifft und tangiert als Computerspiele, die facettenreiche virtuelle Welten ungeahnter Größe generieren – Computerspiele wie *Dishonored*[92], *The Walking Dead*[93] oder *The Last of Us*[94] die Spielerinnen und Spieler gar vor moralische Entscheidungen stellen – und Folgen für gutes oder schlechtes Verhalten in der Spielwelt implementieren. So argumentieren die Autoren, das Spiel erhielte

> „im Kontext digitaler Medien als Topos und als Metapher eine neue Bedeutung […]. Spiele werden zu ‘Spielwelten‘ und damit zu Metaphern neuer, sich schnell entwickelnder Parallelwelten [...] Computerspiele stellen ein bedeutendes, massenhaftes Phänomen der radikalen Moderne dar, indem sie ‘Spiel‘ als

[90] Scheuerl 1991, S. 132.
[91] Vgl. Neitzel, S. 20 ff.
[92] *Dishonored: Die Maske des Zorns* wurde entwickelt von Arkane Studios und 2012 publiziert von Bethesda Softworks.
[93] *The Walking Dead* wurde entwickelt von Telltale Games und 2012 publiziert von Telltale Games.
[94] *The Last of Us* wurde entwickelt von dem Naughty Dog und 2013 publiziert von Sony Computer Entertainment.

anthropologische Konstante und 'Computer' (im Sinne von Informations- und Kommunikationstechnologien) synthetisieren."[95]

Der Weg in der Beschreibung und Durchdringung jener anthropologischen Konstante von klassischen hin zu computerbasierten Spielen wird im Folgenden anhand von Wegpunkten und -markierungen skizziert, wie er sich ausgehend von Huizingas Deutungsangebot vom Spiel des Menschen, das sich Thimm und Wosnitza zufolge als „Grundlage für eine interdisziplinäre Perspektive des Spielens"[96] identifizieren lässt, mäandert hin zum klassischen Spielmodell Jesper Juuls, das Computer und Videospiele in die Theoriebildung integriert.[97] Die nachfolgenden Kapitel intendieren jedoch weniger, Implikationen der computerbasierten Abbildung von Spielen wahrzunehmen, zu besprechen und zu durchdringen,[98] vielmehr beabsichtigen die folgenden Ausführungen, den Gegenstand der vorliegenden Arbeit zu fokussieren und die verwendeten Grundbegriffe und –konzepte zu bestimmen: Was ist zuallererst damit gemeint, wenn vom Spiel gesprochen und geschrieben wird? Über welche Eigenschaften, über welche Eigenheiten verfügen Spiele, im Besonderen Brettspiele? Worin bestehen die Herausforderungen in der computerbasierten Implementation von Spielregeln und Spielumgebungen? Und wie lassen sich schlaue Handlungs- und Zugwahlen der Agierenden des Spieles identifizieren und artifiziell hervorbringen?

Von besonderem Interesse sind der vorliegenden Arbeit zwei Theorieaspekte, die in den folgenden Kapiteln gesondert vorgestellt und besprochen werden: Stellt der den Erscheinungsformen des Spieles verhaftete phänomenologische Ansatz, wie er sich in Huizingas Werk *Homo Ludens* und mit Juuls *klassischem Spielmodell* formuliert, Deutungs- und Zuordnungsoptionen von Handlungs- und

[95] Thimm und Wosnitza 2010, S. 34 f.
[96] Thimm und Wosnitza 2010, S. 37.
[97] Für einen breit gefächerten Überblick über frühe Spieltheorien, die pädagogische, phänomenologische und soziologische Aspekte von Spielen fokussieren, sei auf die beiden Bände *Das Spiel. Band 1: Untersuchungen über sein Wesen, seine pädagogischen Möglichkeiten und Grenzen* (Scheuerl 1990) und *Das Spiel. Band 2: Theorien des Spiels* (Scheuerl 1991) verwiesen. Salen und Zimmerman bieten in ihrer Arbeit *Rules of Play. Game Design Fundamentals* (Salen und Zimmerman 2004) einen umfangreichen Überblick über Spieltheorien; die Aufsatzsammlung *The Game Design Reader: A Rules of Play Anthology* (Salen und Zimmerman 2006) der Autoren stellt einen reichen, vielfältigen Fundus von Texten und Materialien zum Thema bereit.
[98] Einen solchen Überblick bieten beispielsweise Beil 2013, GamesCoop 2012 und auch Stampfl 2012.

Erscheinungsweisen unter dem Begriff des Spieles bereit, so gießt die Spiel- und Entscheidungstheorie nach von Neumann und Morgenstern das theoretische und methodische Fundament für die praktische Implementation problemlösender miteinander konkurrierender und kooperierender Individuen im Kontext der betrachteten Brettspiele.

Auf den folgenden Seiten wird zunächst mit Huizinga und Juul besprochen, wodurch sich das Spiel des Menschen auszeichnet und welche Eigenschaften ihm – den Autoren zufolge – wesenseigen sind. Daran anschließend werden grundlegende Konzepte und Elemente der Spieltheorie nach von Neumann und Morgenstern vorgestellt, um die zu einem späteren Zeitpunkt zu betrachtenden Brettspiele mit dem elaborierten Vokabular der Spieltheorie präzise beschreiben, sowie unterschiedliche Handlungsweisen von Spielerinnen und Spielern in der Welt des Spieles ausdrücken und repräsentieren zu können.

3.2 Huizingas Homo Ludens

> „Wir alle spielen – und dennoch: Wer kann schon sagen, was genau ein Spiel ist?“[99]

Eine äußerst anregende Wirkung auf Beantwortungsversuche der Frage, was das denn sei, das *Spiel des Menschen*, darf der Arbeit Huizingas angemerkt werden.[100] In seinem 1938 veröffentlichten Werk *Homo Ludens – Vom Ursprung der Kultur im Spiel* skizziert Johan Huizinga (1872-1945) sein Vorhaben, das „echte, reine Spiel selbst als eine Grundlage und einen Faktor der Kultur zu erweisen“ [101]. Bereits zu Beginn seiner Ausarbeitung stellt Huizinga die Frage, wie jenes *echte*, jenes *reine* Spiel zu charakterisieren sei und formuliert Antworten unter der Grundannahme, dass der Begriff des Spieles schlecht fassbar sei, gar „in merkwürdiger Weise abseits von allen übrigen Gedankenformen [stehe], in denen wir die Struktur des Geisteslebens und des Gemeinschaftslebens ausdrücken können“ [102]. So müsse vorläufig eine Beschränkung stattfinden, folgert Huizinga, die *Hauptkennzeichen* des Spiels ausfindig zu machen und zu beschreiben. Jene Beobachtung der Erscheinungsweisen des Spieles, jenes Ausfindigmachen der Hauptkennzeichen des menschlichen Spieles und menschlichen Spielverhaltens [103] bündelt sich schließlich in der im Folgenden zitierten Definition:

> „Der Form nach betrachtet, kann man das Spiel also zusammenfassend eine freie Handlung nennen, die als ‘nicht so gemeint‘ und außerhalb des gewöhnlichen Lebens stehend empfunden wird und trotzdem den Spieler völlig in Beschlag nehmen kann, an die kein materielles Interesse geknüpft ist und mit der kein Nutzen erworben wird, die sich innerhalb einer eigens bestimmten Zeit und eines eigens bestimmten Raums vollzieht, die nach bestimmten Regeln ordnungsgemäß verläuft und Gemeinschaftsverbände ins Leben ruft, die

[99] Stampfl 2012, S. 1.
[100] Vgl. Stampfl 2012, S. 4 ff.
[101] Huizinga 2009, S. 13.
[102] Huizinga 2009, S. 15.
[103] Vgl. die Einordnung von Huizingas Fokussierungs- und Betrachtungsweise als *anthropologisch-phänomenologisch* in Neitzel, S. 20 ff.

> ihrerseits sich gern mit einem Geheimnis umgeben oder durch Verkleidung als anders von der gewöhnlichen Welt abheben.“[104]

Huizingas Intention, das „echte, reine Spiel selbst als eine Grundlage und einen Faktor der Kultur zu erweisen“ ist mit der referierten Definition vorbereitet. Menschliches Spiel konzentriere und kondensiere sich Huizinga zufolge als gründende Konstante, als Grundprinzip der Kulturgeschichte.[105] Nach Strupp ist Huizinga die Verknüpfung von Kultur[106] mit Spiel essentiell: Das Spiel „verleiht der Kultur Form und Stimmung. Die Kultur realisiert sich durch und als Spiel, verdrängt aber mit zunehmendem Fortschreiten dessen ursprüngliche Elemente.“[107] Jenes Verdrängen ursprünglicher Spielelemente sei eine – bewusste oder unbewusste – Hinwendung zum Ernst. Nach und nach wandelt sich das Spielerische zum Ernsthaften; Kultur wird ernsthaft und „räumt dem Spiel nur noch eine Nebenrolle ein. Die agonale Periode ist vorbei – oder scheint vorbei“, wie Huizinga ausführt.[108]

Zwar birgt Huizingas Annäherung an den Begriff des Spieles noch Unschärfen und Raum für Anmerkungen – so vernachlässigt Huizinga die vielfältigen unterschiedlichen Arten und Erscheinungsweisen des Spieles[109] –; für die von ihr ausgehende Beschäftigung mit Spiel hat Huizingas Auseinandersetzung mit dem Spiel des Menschen jedoch katalytische Funktion: Huizingas Definition von *Spiel* setzt die neuzeitliche Beschäftigung mit dem Phänomen- und Fragenkomplex Spiel in Gang, sie regt das interdisziplinäre Nachdenken darüber an, was Spiel ist, wodurch sich das Spiel und menschliches Spielen auszeichnet, welche als konstitutiv ausgezeichneten Eigenschaften des Spiels wohl tatsächlich dem Untersuchungsgegenstand wesenseigen sind und führt mit vertiefenden theoriebildenden und -ergänzenden Lesarten und Forschungen der an die

[104] Huizinga 2009, S. 22.
[105] Vgl. Strupp 2000, S. 183.
[106] Für eine Erörterung Huizingas Kulturbegriffes, wie er im *Homo Ludens* anklingt, sei verwiesen auf das Kapitel *Spiel als kulturelle Grundform: Homo Ludens* in Strupp 2000, S. 183 ff.
[107] Strupp 2000, S. 185 f.
[108] Huizinga 2009, S. 88, vgl. auch die Begriffsklärung des Agon als „an feste Regeln gebundener Kampf in geheiligten Formen“ in Huizinga 2009, S. 89.
[109] Vgl. hierzu auch die Erläuterung der unterschiedlichen Spielarten nach Roger Caillois in *Agon*, *Alea*, *Mimikry* und *Ilinx* bei Stampfl 2012, S. 5 f. sowie den kompakten Überblick über die zeitgenössische Kritik an Huizingas *Homo Ludens* in Strupp 2000, S. 185 ff.

Ausführungen Huizingas anknüpfenden Interpretinnen und Interpreten zu verfeinernden Iterationen in der Reflektion und Definition des Spielbegriffes.

3.3 Juuls klassisches Spielmodell

Rund 70 Jahre nach der Veröffentlichung von Huizingas *Homo Ludens* stellt Jesper Juul in seiner 2005 publizierten Arbeit *Half-real: Video Games Between Real Rules and Fictional Worlds*[110] sein *klassisches Spielmodell* vor, das als Ergebnis der Reflektion über sieben Spieldefinitionen früherer Autorinnen und Autoren auch computerrealisierte Spiele in die Theoriebildung einbezieht.[111] Ausgehend von frühen Spieldefinitionen des 20. Jahrhunderts, wie dem zuvor besprochenen Interpretationsangebot Huizingas und nachfolgende Arbeiten berücksichtigend, referiert Juul seine Definition von *Spiel*, die sich aus sechs Bestandteilen konstituiert und intendiert, das Spiel vom Nicht-Spiel abzugrenzen sowie Computerspiele in die Definitionsdiskussion aufzunehmen. So berichtet Juul, dass Spiele seit über tausenden von Jahren[112] einem bestimmten Schema folgten – ein Schema, dem sich Juul mit seiner im Folgenden wiedergegebenen Definition anzunähern sucht:

> „A game is a rule-based system with a variable and quantifiable outcome, where different outcomes are assigned different values, the player exerts effort in order to influence the outcome, the player feels emotionally attached to the outcome, and the consequences of the activity are negotiable.“[113]

Mit seiner Definition beschreibt Juul mehr das Spiel (*game*) und weniger das Gespieltwerden des Spieles (*play*). So stellt Juul das Spiel (*game*) vor als ein regelbasiertes System, in dem sich Spielerinnen und Spieler darum bemühen, den Spielverlauf zu beeinflussen, um das Spiel zu einem bestimmten Spielergebnis zu

[110] Juul 2005.
[111] Juul bespricht in seiner Arbeit sieben Interpretationsangebote von *Spiel*, u.a. von Johan Huzinga, Roger Caillois, Bernard Suits, bis hin zu den Deutungsofferten von Katie Salen und Eric Zimmerman (vgl. auch den Überblick über die sieben Definitionen in der Tabelle 2.1 in Juul 2005, S. 30).
[112] Vgl. Juul 2010, S. 131 f.
[113] Juul 2005, S. 36.

führen.[114] Konstituierende Grundbestandteile des Spieles sind nach Juul zum einen die Spielregeln, zum anderen das veränderliche und quantitativ bestimmbare Spielergebnis sowie das Verhältnis der Spielerinnen und Spieler zum Spiel: Das Spielergebnis ist charakterisiert durch eine individuell-subjektive und transsubjektive Wertigkeit, die Spielerinnen und Spieler dem Ergebnis beimessen – und die Spieler dazu anregt, sich um das Endergebnis des Spieles zu bemühen.[115] Wie bereits Huizinga anmerkt, spielt sich das Spiel nicht im Bereich des *gewöhnlichen* Lebens ab. Vielmehr sei das Spiel „das Heraustreten aus ihm in eine zeitweilige Sphäre von Aktivität mit einer eigenen Tendenz“[116]. Jene Sphäre, jener *magische Zirkel*[117], der betreten und wieder verlassen wird, transformiert Menschen zu Spielerinnen und Spielern und ist Juul zufolge optional verknüpft mit Auswirkungen auf die *wirkliche* Welt der Spielenden. Unternommen wird das Spiel „um seiner selbst willen“[118], es dient keinem bestimmten Zweck. Spiel wandelt das *gewöhnliche* Leben ins Ungewöhnliche indem es sich abspielt, sich selbst Gegenstand ist, sich selbst Grenzen setzt, indem es Aktions- und Handlungsräume für Spielerinnen und Spieler entwirft und bereitstellt.

Zentrales Moment der referierten Spieldefinition ist die Regelhaftigkeit des Systems Spiel, sind die Spielregeln. So verfügt jedes Spiel über seine eigenen, für alle am Spiel Teilnehmenden verbindlichen Regeln. Spielregeln steuern und determinieren den Spielverlauf, sie formen materielle Gegenstände des Alltags wie Murmeln und Steine zu bedeutungstragenden Spielgegenständen. Spielerinnen und Spieler, die sich nicht an die Spielregeln halten, sind Falschspieler, gar Spielverderber, wie Huizinga anmerkt.[119] Werden die Regeln des

[114] Vgl. auch die Definition des Spieles bei Salen und Zimmerman: „A game is a system in which players engage in artificial conflict, defined by rules, that result in quantifiable outcome“ (Salen und Zimmerman 2004, S. 81).
[115] Vgl. Juul 2005, S. 40.
[116] Huizinga 2009, S. 16.
[117] Vgl. u.a. Björk und Holopainen 2006, S. 15 sowie das Kapitel *The Magic Circle* in Salen und Zimmerman 2004, S. 93–99 zur Besprechung des *Magic Circle*, aber auch Günzel 2010 als Kritik an der Arbeit Juuls.
[118] Stampfl 2012, S. 2.
[119] Für eine ausführliche Differenzierung zwischen Falschspieler und Spielverderber sei an dieser Stelle verwiesen auf Huizinga 2009, S. 20.

Spieles missachtet, „stürzt die Spielwelt zusammen. Dann ist es aus mit dem Spiel“[120].

3.3.1 Spielregeln

Eindeutig definieren die Spielregeln sowohl den Handlungsraum und die Handlungsmöglichkeiten der Spielerinnen und Spieler in der Spielwelt, die Interaktions- und Manipulationsmöglichkeiten der Spielobjekte als auch das Verhältnis der Spielobjekte zu- und untereinander. In seiner Arbeit verweist Juul auf den ambivalenten Charakter von Spielregeln: Indem sie definieren, welche Spielsymbole zu einem bestimmten Spielzeitpunkt auf eine vor der Spielpartie festgelegte Art und Weise bewegt werden dürfen, schränken Spielregeln den Handlungsraum der Spielerinnen und Spielern ein; zum anderen eröffnen Spielregeln jedoch unterhaltsame, fordernde und anspruchsvolle Spielpartien, indem sie eine Auswahl treffen aus dem unendlichen Fundus von Möglichkeiten – wie den für das Spiel zur Verfügung stehenden Spielmaterialien, den Spielsymbolen oder Zugmöglichkeiten der Spielfiguren.[121]

Ob vor dem Monitor, ob mit Stift und Papier oder mit holzgeschnitzten Spielfiguren auf einem Spielbrett – Spielregeln sind intermedial, generieren (zumeist) wiederholbare Spielpartien und sind besonders dadurch charakterisiert, dass sie bestimmen, was das Spiel ist und welchen Verlauf die gespielte Partie nehmen kann. So kann eine Spielpartie Schach, die zwei Spielerinnen und Spieler auf einem Blatt mit handgezeichneten Spielmaterialien und -symbolen begonnen haben, notiert und weitergeführt werden auf einem Spielbrett aus Stein oder Holz. Das Schachspiel ist sich hierbei in seinen Regeln immer gleich.

Mit Salen und Zimmerman lassen sich unterschiedliche Arten von Spielregeln differenzieren: Operationale Regeln definieren den Autoren zufolge das Zusammenspiel der Spieler mit dem System *Spiel* (i.e. die auf Papier festgehaltenen und -geschriebenen Regeln[122]), konstitutive Regeln bestimmen die dem Spiel zugrundeliegenden formalen Strukturen und implizite Regeln

[120] Huizinga 2009, S. 20.
[121] Vgl. das Kapitel *Rules* in Juul 2005, S. 55–120.
[122] Vgl. Björk und Holopainen 2006, S. 16.

beschreiben die Etikette, regeln das Miteinander der Spielerinnen und Spieler.[123] Das im Rahmen der vorliegenden Ausarbeitung entwickelte und später noch ausführlich zu besprechende Framework *SpoookyJS* implementiert die operativen Regeln der betrachteten Spiele als Komplex von Spielregelatomen, die kombiniert und als Summe einzelne Spielregeln konstituieren.

3.3.2 Spiel- und Zugregelatome

Besonders in der Entwicklung digitaler Spiele ist es unabdinglich, die Regeln des zu implementierenden Spieles aufzugliedern in kleinste Elemente, im Folgenden als *Regelatome* bezeichnet, die bestimmte Gegebenheiten und Zustände des Spieles repräsentieren und über die logische Verknüpfung mit- und untereinander Ereignisse in der Spielwelt verursachen. Abstrakte Spielregeln wandeln sich somit in konkrete Möglichkeiten mit kausalen Folgen sowohl für die Spielwelt als auch für die Spielerinnen und Spieler des Spieles. So können der König und Turm einer Spielerin im Schachspiel einen Doppelzug, die kurze Rochade vollziehen, wenn bestimmte Bedingungen erfüllt sind. Möchte die Spielerin mit den weißen Spielfiguren eine kurze Rochade durchführen, wie in der folgenden Darstellung visualisiert, so werden fünf Regelatome über eine logische UND-Verknüpfung miteinander verknüpft und nacheinander auf ihre Erfülltheit geprüft.

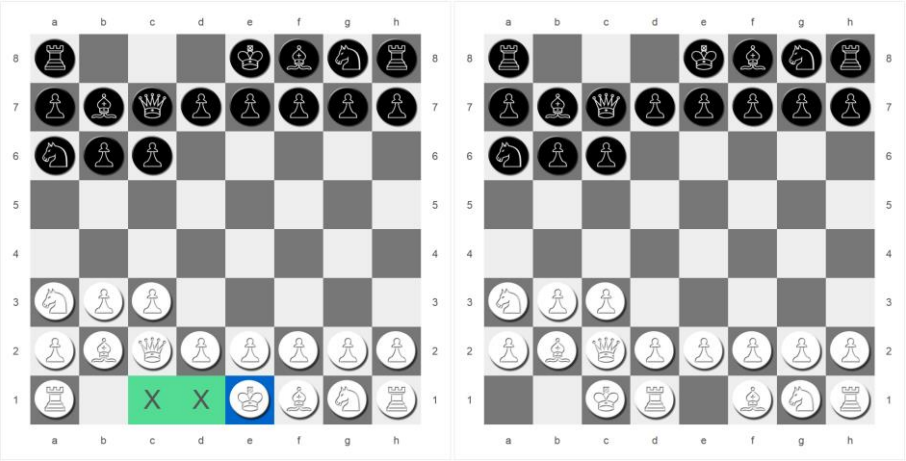


Abbildung 8: Zugregel *kurze Rochade* im Schachspiel auf der linken Seite vor und auf der rechten Seite nach Ausführung des Zuges.

[123] Vgl. Salen und Zimmerman 2004, S. 141–149.

Ist hierbei nur ein Regelatom nicht erfüllt bzw. wahr, so ist auch die vollständige Regelbedingung nicht erfüllt – die Folgen der Spielregel werden nicht aktiviert, die beiden Spielfiguren somit nicht bewegt. Der nachfolgend wiedergegebene Pseudocode exemplifiziert die Zusammensetzung der kurzen Rochade aus fünf Regelatomen und veranschaulicht die Verknüpfung der erfüllten Bedingungen mit den Folgen für die Spielwelt:

Zugregel *Kurze Rochade (Spielerin weiß)*

Wenn der weiße König nicht von einem gegnerischen Spielstein bedroht wird ($Atom_1$) **und** der weiße König noch nicht bewegt wurde ($Atom_2$) **und** der weiße Turm auf Position h2 noch nicht bewegt wurde ($Atom_3$) **und** sich zwischen dem weißen König auf e1 und dem weißen Turm keine eigenen oder gegnerischen Spielsteine befinden ($Atom_4$) **und** sich zwischen dem weißen König und dem weißen Turm kein Feld findet, das von einem gegnerischen Spielstein bedroht wird ($Atom_5$),

dann bewege den weißen König nach g1 **und** bewege den weißen Turm nach f1.

Das im Rahmen der vorliegenden Arbeit entwickelte Framework *SpoookyJS* implementiert Spielregelatome und Spielregeln als JavaScript-Objekte.

3.3.3 Zusammenfassung und Ausblick

Juuls klassiches Spielmodell mag nicht auf jede spieldefinitorische Frage eine Antwort bieten, offeriert jedoch eine sinnvolle – und der vorliegenden Ausarbeitung ausreichend trennscharfe – Differenzierung der unterschiedlichen Spielbestandteile und Spielkonstituenten – und dient den folgenden Kapiteln dazu, relevante Aspekte zu fokussieren: Die vorliegende Arbeit intendiert weniger, das Verhältnis von Spielerinnen und Spielern zum Spiel, noch das Verhältnis des Spiels zum *Rest der Welt* zu betrachten, wie in Juuls Definitionspartikeln *Player effort* sowie *Player attached to outcome* angeklungen, sondern beschäftigt sich vornehmlich mit dem Spiel als System vielfältiger Spielobjekte, Spielbestandteile und Spielfunktionen und fokussiert das für Spiel und Spiele besonders

charakteristische Moment der Spielregeln und die Handlungsmöglichkeiten der im und mit dem Spiel Agierenden. Was mit den Agierenden des Spieles zu verstehen ist und wie sich Spielregeln im Kontext der digitalen webbasierten Umsetzung von Brettspielen abstrakt und wiederverwendbar formulieren lassen, um von den artifiziell implementierten Agierenden des Spiels wahrgenommen und als individuell-eigenes Handlungsziel algorithmisch intendiert zu werden, das sei in den folgenden Kapiteln noch näher betrachtet.

Der zuvor mit der Arbeit Huizingas und Juuls skizzierte Ansatz, das Spiel über seine Erscheinungsformen zu charakterisieren und zu definieren, wird im Folgenden ergänzt um die Spieltheorie nach von Neumann und Morgenstern, die ein elaboriertes Vokabular in der vollständigen Beschreibung und Visualisierung von Spielen und Spielpartien bietet. Darüber hinaus stellt die Spieltheorie nach von Neumann und Morgenstern das theoretische Handwerkszeug in der Implementation autarker, rational handelnder Spielindividuen bereit, die miteinander kooperieren oder gegeneinander wettstreiten.

3.4 Spieltheorie nach von Neumann und Morgenstern

An zwei Abenden einer milden Frühlingswoche treten eine Hobbyköchin und ein Hobbykoch gegeneinander an, um den besten Koch oder die beste Köchin unter ihnen zu ermitteln. Am ersten Abend des Wettbewerbes bereitet die Köchin ein wohlschmeckendes Nachtmahl für sich und ihren Kontrahenten zu; am zweiten Abend des Wettbewerbes kredenzt der Hobbykoch seiner Konkurrentin und sich eine deliziöse Speise. Wohlgenährt und lukullisch verwöhnt, bewerten die Bekochten am Ende des Wettbewerbes das Werk der Rivalen entweder mit gut (das Essen war äußerst vorzüglich) oder schlecht (die Speise hat nicht gemundet). Die Motivation, wohlschmeckend zu kochen, besteht für beide Teilnehmer in der Auszahlung des Wettbewerbes: Wird nur einer der beiden Teilnehmer gut bewertet, so erhält die für gut befundene Kochkunst ein Preisgeld von 100 Euro und der Koch der nicht mundenden Speise geht leer aus. Bei positivem Gleichstand – beide haben die kulinarische Finesse des Anderen für gut befunden – teilen sich die Köchin und der Koch auf dem Siegertreppchen den Betrag: Beide erhalten einen Gewinn von 50 Euro. Bewerten hingegen beide das Mahl ihres Gegenspielers schlecht, so erhält jeder nur eine Aufwandsentschädigung von 20 Euro.

Wie sollten sich die Teilnehmerinnen und Teilnehmer des Wettbewerbes nun verhalten, um den Wettbewerb für sich mit einer möglichst hohen Auszahlung zu entscheiden? Sollten sie in jedem Falle egoistisch verfahren und den anderen Koch schlecht bewerten? Solch ein Agieren würde das soziale Gefüge der Kochgruppe aufs Äußerste strapazieren, so dass der allzu ichbezogene Koch zu einem nächsten Wettbewerb wohl nicht mehr eingeladen werden würde (vorausgesetzt, die Möglichkeit der Wiederholung des Wettbewerbes bestünde). Oder ist es ratsam, den Zufall zu bemühen und die Essensbewertung einem Münzwurf anheim zu geben? Würden die beiden Köche miteinander kooperieren und sich vorab auf eine feste Bewertung einigen, so könnte der Gewinnbetrag durch zwei dividiert werden und jeder und jede erhielte eine passable Auszahlung von 50 Euro – aber stellt solch eine Absprache nicht den Sinn des Wettbewerbes infrage? Und was, wenn sich ein Teilnehmer nun doch nicht an die zuvor getroffene Abmachung

halten und den anderen Koch arglistig schlecht bewerten würde, um die eigene Auszahlung zu maximieren?

Die Spieltheorie, wie sie sich in der 1944 publizierten Arbeit *Theory of Games and Economic Behavior* John von Neumanns und Oskar Morgensterns gründet und im Laufe der Jahrzehnte zahlreiche Verfeinerungen und Ergänzungen erfahren hat,[124] befasst sich mit ebensolchen Fragen und Problemen und beschäftigt sich mit agierenden und interagierenden Individuen, die versuchen, ihre Umwelt zu manipulieren, während die Umwelt im Gegenzug versucht, jene Individuen, die Spielerinnen und Spieler des Spiels, zu manipulieren.

3.4.1 Spiel und Spielpartie, Zug und Zugwahl

In der Vorstellung und Besprechung ihrer Termini Technici[125] unterscheiden von Neumann und Morgenstern zuallererst das Spiel von der Spielpartie: Das Spiel (*game*) ist von Neumann und Morgenstern zufolge die Gesamtheit der Regeln, die das Spiel beschreiben. Eine Spielpartie (*play*) bezeichnet den Spielvorgang, in dem das Spiel gespielt wird.[126] Die Unterscheidung zwischen Spiel und Spielpartie wird angereichert durch die Distinktion von Spielzug und Zugwahl: Eine Partie zu spielen bedeutet von Neumann und Morgenstern zufolge, zu einem bestimmten Zeitpunkt im Verlauf der Spielpartie aus einem Angebot von möglichen Spielzügen (*move*) einen Spielzug zu wählen (*choice*) – das Spiel besteht aus einer Folge von Spielzügen, die Spielpartie aus einer Folge von konkreten Zugwahlen. Welche Spielsymbole zu welchem Spielzeitpunkt von Spielerinnen und Spielern oder von einem Zufallsmechanismus auf welche Art und Weise bewegt werden dürfen, das ist eindeutig vorgeschrieben durch die Regeln des Spieles, wie von Neumann und Morgenstern anmerken:

---

[124] Vgl. den einleitenden Überblick über spieltheoretische Arbeiten, die mit dem Nobelpreis für Wirtschaftswissenschaften ausgezeichnet wurden in Sieg 2010, S. 4, das Kapitel *Entwicklung der Spieltheorie* in Diekmann 2009, S. 16 ff. für eine kurze Übersicht über die Entwicklung der Spieltheorie von ihren Anfängen hin zu interdisziplinären Arbeiten sowie die Abschnitte *Entwicklungsetappen der Spieltheorie* und *Personenkult in der Spieltheorie* in Berninghaus et al. 2010, S. 3–9.

[125] Vgl. Neumann und Morgenstern 2004, S. 49 ff.

[126] Vgl. die englischsprachige Unterscheidung zwischen Spiel und Spielpartie: „The game is simply the totality of the rules which describe it. Every particular instance at which the game is played – in a particular way – from beginning to end, is a play“ (Neumann und Morgenstern 2004, S. 49).

> „A move is the occasion of a choice between various alternatives, to be made either by one of the players, or by some device subject to chance, under conditions precisely prescribed by the rules of the game.“[127]

Charakterisiert ist das Spiel der Spieltheorie nach von Neumann und Morgenstern zum einen durch seine Agierenden, die Spielerinnen und Spieler des Spieles, zum anderen durch die unterschiedlichen Entscheidungen, die von den Spielern zu einem bestimmten Spielzeitpunkt und unter einem bestimmten Grad individueller Informiertheit über das Spiel getroffen werden können sowie der Beweggrund der Spieler, sich im Spiel auf eine bestimmte Weise zu verhalten: die bereits mit Juul angemerkte Auszahlung des Spieles, das Spielergebnis. So lässt sich das zuvor wiedergegebene Kochbeispiel als das Spiel zweier Spieler betrachten, in dem die unterschiedlichen Handlungen der Spieler zu unterschiedlichen individuellen Auszahlungen führen: Bewertet die Köchin die Mahlzeit ihres Gegenspielers schlecht und beurteilt der Koch die Speise seiner Kontrahentin gut, so beträgt die Auszahlung für die Köchin 100 Euro und für den Koch 0 Euro. Würden beide Spieler das Essen ihres Gegenspielers positiv bewerten, so würden sich beide die maximale Auszahlung von 100 Euro teilen – ein wünschenswertes Ergebnis des Spieles, wäre beiden Agierenden doch eine Auszahlung garantiert.

Die Spieltheorie nach von Neumann und Morgenstern betrachtet jene verzwickte Situation der Protagonisten des Kochspieles als ein Spiel zweier Spieler und formuliert Fragen unter anderem nach dem Handeln, der Kommunikationsmöglichkeiten sowie der Rationalität der Spielerinnen und Spieler des Spieles:

- Handeln und Agieren: Welche Handlungsmöglichkeiten stehen den Spielern zu einem bestimmten Zeitpunkt des Spieles zur Verfügung? Für welche Handlungsoption sollten (und werden) sich die einzelnen Spieler entscheiden? Handeln die Spieler gleichzeitig? – und gliedert sich das Spiel in mehrere Runden?
- Anzahl der Spielerinnen und Spieler: Inwiefern verändert sich die Situation, wenn weitere Spieler am Spiel teilnehmen würden?

[127] Neumann und Morgenstern 2004, S. 49.

- Kommunikation: Welchen Einfluss hätte Kommunikation vor Spielbeginn auf die Spielentscheidungen der Spielerinnen und Spieler? Würde das Spiel einen anderen Verlauf nehmen, könnten sich die Spieler vor dem Spiel besprechen?
- Rationalität: Welche Auffassung hat ein Spieler über die Denk- und Handlungsweise seiner Gegenspielerin? Würde es die eigene Handlungsweise beeinflussen, wenn nicht davon ausgegangen werden kann, dass die Gegenspielerin das für sie beste Spielergebnis zu erreichen sucht?
- Wiederholung: Würden die Spieler anders entscheiden, wenn das Spiel nicht nur einmal, sondern wiederholt gespielt würde?

Weil die Spielerin und der Spieler des Kochspieles ihre Entscheidungen unabhängig voneinander treffen und der Köchin sowie dem Koch die Entscheidung des Anderen nicht bekannt sind, wird das Kochspiel als ein Zweipersonen-Spiel in Normalform bezeichnet, das sich vollständig durch eine Auszahlungsmatrix repräsentieren und abbilden lässt.

3.4.2 Das Spiel Γ in Normalform

Spiele der Spieltheorie nach von Neumann und Morgenstern lassen sich auf unterschiedliche Arten und Weisen repräsentieren: Stellt die Extensivform von Spielen den sequentiellen Verlauf der Entscheidungssituationen der Spielerinnen und Spieler im Laufe einer Spielpartie dar, so beschreiben Normalformspiele simultane Konflikt- und Entscheidungssituationen mit einer überschaubaren Anzahl von Elementen. Den nachfolgend betrachteten Spielen gemein sind die folgenden Aspekte und Eigenschaften.[128] Jedes der im Rahmen der vorliegenden Arbeit diskutierten Spiele

- wird gespielt von zwei oder mehr als zwei Spielern,
- nimmt seinen Lauf ausgehend von einer initialen Spielkonfiguration,

[128] Die Skizzierung der Eigenschaften von Spielen der Spieltheorie folgt dem *Versuch einer Spieldefinition*, wie er sich wiedergegeben findet bei Wessler 2012, S. 18.

- bietet den Spielern zu jedem Zeitpunkt des Spieles eine Menge von Entscheidungsmöglichkeiten (den Strategien), sowie Spielregeln, „die festlegen, welcher Spieler sich in welcher Situation für welche seiner Strategien entscheiden kann“[129],
- lässt seine Spieler Strategien gleichzeitig oder rundenbasiert treffen,
- informiert die Spieler über den Zustand der Spielwelt vor und nach getroffenen Entscheidungen der Spielerinnen und Spieler – oder lässt die Spieler im Unklaren über die Folgen von getroffenen Entscheidungen, und
- verfügt über einen Endzustand, auf den die Auszahlung der Spielerinnen und Spieler folgt.

Die Normalform von Spielen der Spieltheorie – also ein Spiel simultaner Entscheidungen – lässt sich formal wie folgend definieren:[130]

Ein Spiel Γ in Normalform ist vollständig beschrieben durch das Tripel $\Gamma = (\mathrm{N}, \mathrm{S}, \mathrm{u})$. Hierbei bezeichnet

- N die finite Anzahl der Spielerinnen und Spieler aus der Spielermenge $N = \{1, \dots, n\}$.
- S den Strategieraum. Der Strategieraum gibt die Menge aller möglichen Strategiekombinationen $s = (s_1, \dots, s_i, \dots, s_n)$ aus den Strategien der einzelnen Spieler an, d.h. $s \in S$. Der Strategieraum S ist definiert als kartesisches Produkt der Strategiemengen $S = S_1 \times \cdots \times S_n$, wobei S_i eine endliche Menge von Entscheidungen und Aktionen für jeden Spieler i in N bezeichnet.

[129] Wessler 2012, S. 18.

[130] Vgl. Holler und Illing 2009, S. 4, vgl. auch die Formalisierung eines Spieles Γ in Normalform bei Shoham und Leyton-Brown 2009, S. 55 f. sowie Prechtl 2008, S. 547 f. und die Kapitel *Explanation oft the Termini Technici* und *The Elements oft the Game* in Neumann und Morgenstern 2004, S. 48 ff. sowie Davis‘ informative Beschreibung von Spiel: „An jedem Spiel nehmen mindestens zwei Spieler teil, von denen jeder eine Strategie wählt, d.h. eine Entscheidung trifft. Das Ergebnis dieser gemeinsamen Wahl – oder auch des Zufalls, falls das Ergebnis einem der Spieler egal ist – führt zu einer Belohnung bzw. Bestrafung eines jeden Spielers, zur sogenannten Auszahlung. Da die Strategie jedes Spielers das Endresultat beeinflußt, muß jeder Spieler darüber nachdenken, welche Strategie wohl jeder andere Mitspieler wählen wird, wohl wissend, daß diese die gleichen Überlegungen anstellen.“ (Davis 2005, S. 17f.).

- $u = (u_1, \dots, u_n)$ die Nutzenfunktionen, wobei $u_i : S \mapsto \mathbb{R}$ die Auszahlungs- oder Nutzenfunktion des Spielers i bezeichnet. Wird die Strategiekombination s gespielt, so lässt sich mit $u_i(s)$ der Nutzen für Spieler i bestimmen.

Das eingangs vorgetragene Kochspiel wird gespielt von dem Koch und seiner Gegenspielerin, die Spielermenge N ist folglich $N = \{1, 2\}$. Reduziert auf die spielabschließende Entscheidungssituation, verfügt jeder Spieler des Kochspieles über zwei Strategien: Kooperation (s_{i1}) und Nicht-Kooperation (s_{i2}). Der Strategieraum S gestaltet sich folglich als

$$S = \{s_{11}, s_{12}\} \times \{s_{21}, s_{22}\} = \{(s_{11}, s_{21}), (s_{11}, s_{22}), (s_{12}, s_{21}), (s_{12}, s_{22})\}.$$

Beide Spieler sind zum Zeitpunkt ihrer Entscheidung nicht darüber informiert, welche Strategie der Gegenspieler wählen wird, um das Spiel für sich zu einem möglichst positiven Ende, d.h. mit einer möglichst hohen Auszahlung zu beenden. Übersichtlich zusammengefasst stellen sich die unterschiedlichen Bewertungsoptionen und die situationsabhängigen monetären Auszahlungen der beiden Köche in der folgenden Auszahlungsmatrix dar:

		Koch	
		Bewertet die Köchin positiv (s_{21})	Bewertet die Köchin negativ (s_{22})
Köchin	Bewertet den Koch positiv (s_{11})	Köchin: 50 Euro Koch: 50 Euro	Köchin: 0 Euro Koch: 100 Euro
	Bewertet den Koch negativ (s_{12})	Köchin: 100 Euro Koch: 0 Euro	Köchin: 20 Euro Koch: 20 Euro

Tabelle 1: Informelle Auszahlungsmatrix des Kochspieles.

Das Besondere des vorgetragenen Kochspieles besteht darin, dass sich die Spielenden eingebettet finden in eine Dilemmasituation: Um das bestmögliche gemeinsame Spielergebnis zu erzielen, müssten beide Spieler kooperieren; auf Kooperation des Gegenspielers ist jedoch kein Verlass, weil sich durch einseitiges Abweichen von der Kooperation die höchste individuelle Auszahlung erzielen lässt. Solche Spiele, die Spielerinnen und Spieler vor die Wahl stellen, ihren

Gegenspieler zu verraten, um das für sie beste Spielergebnis zu erzielen, werden als *Gefangenendilemma* bezeichnet.

3.4.3 Das Gefangenendilemma

Die wohl bekannteste Veranschaulichung der Spieltheorie nach von Neumann und Morgenstern – und das wohl am häufigsten zitierte Spiel der Spieltheorie – ist das *Gefangenendilemma* (*Prisoner's Dilemma*), das sich in seiner ursprünglichen Form wie folgend formuliert:[131] Zwei eines Schwerverbrechens Angeklagte werden in Gewahrsam genommen und getrennt voneinander verhört. Jeder Angeklagte kann sich im Verhör dazu entschließen, das schwere Verbrechen entweder zu gestehen und somit seinen Komplizen zu belasten oder nicht zu gestehen, das schwere Verbrechen begangen zu haben. Das Strafmaß für die unterschiedlichen Handlungen der beiden Straftäter wurde von der Staatsanwältin im Vorhinein festgelegt und den Gefangenen mitgeteilt: Gesteht keiner der Straftäter das schwere Verbrechen, so werden beide wegen eines minder schweren Verbrechens mit jeweils drei Jahren Haft bestraft. Gestehen die zwei Straftäter das schwere Verbrechen, so wird jeder Delinquent zu fünf Jahren Gefängnis verurteilt. Gibt jedoch der eine Gefangene zu, das schwere Verbrechen mit seinem Partner begangen zu haben und schweigt sein krimineller Partner, so tritt eine Kronzeugenregelung in Kraft: Der geständige Verbrecher wird freigelassen und sein Partner hat in den nächsten sieben Jahren mehr als genug Zeit, im Gefängnis über sein Verbrechen nachzudenken.

Wie sollten nun die Spieler des Gefangenendilemmas handeln, vorausgesetzt, beide Spieler sind bestrebt, das für sie bestmögliche Spielergebnis zu erzielen? Jeder der beiden Spieler des Gefangenendilemmas verfügt in dem – an dieser Stelle – nicht wiederholbaren Spiel über die Handlungsmöglichkeiten *Gestehen* sowie *Nicht-gestehen*. Übersichtlich visualisieren lassen sich die unterschiedlichen Folgen der individuellen Entscheidungen in einer Auszahlungsmatrix, die auch als *Normalform* oder *strategische Form* des Spieles bezeichnet wird. In der Auszahlungsmatrix sind alle möglichen Entscheidungskombinationen der

[131] Das Gefangenendilemma ist wiedergegeben nach Luce und Raiffa 1989, S. 94–97, Davis 2005, S. 104–109 und Prechtl 2008, S. 574f.

Spielerinnen und Spieler des Gefangenendilemmas und die individuellen Spielergebnisse als Auszahlungswerte für jede Spielerin und jeden Spieler des Spieles vollständig repräsentiert:

		Spieler 2	
		Nicht-Gestehen (s_{21})	Gestehen (s_{22})
Spieler 1	Nicht-Gestehen (s_{11})	3,3	7,0
	Gestehen (s_{12})	0,7	5,5

Tabelle 2: Auszahlungsmatrix des Gefangenendilemmas

Die Werte in der Auszahlungsmatrix des Spieles *Gefangenendilemma* geben die Jahre der Bestrafung der Spieler im Kontext unterschiedlicher gleichzeitiger Handlungsmöglichkeiten wieder. Der erste Wert, links des trennenden Kommas, steht hierbei für Spieler 1, der zweite Wert für Spieler 2 – in der ursprünglichen Form des Gefangenendilemmas gilt freilich, dass niedrigere Werte für jeden Spieler – folglich ein kürzerer Aufenthalt im Gefängnis – individuell besser sind als höhere Werte.

3.4.4 Dominante Strategien und das Nash-Gleichgewicht

Beide Spieler des Gefangenendilemmas können kooperativ (*welche Entscheidung ist für uns die beste?*) oder nicht-kooperativ (*welche Entscheidung ist für mich die beste Entscheidung?*) agieren. Nicht-kooperatives Handeln wird auch als *Defektion* bezeichnet. Handeln beide Spieler kooperativ, so ist dies im oben angeführten Spiel günstiger, als handelten alle Spieler des Spieles nicht-kooperativ. Da keiner der beiden Spieler jedoch weiß, wie sich der andere Spieler verhält und bindende Verträge zwischen den Spielern nicht möglich sind, besteht die beste Entscheidung jedes Spielers des zuvor wiedergegebenen Gefangenendilemmas darin, zu gestehen (Strategie s_{i2}), da die Summe der Auszahlungen jeweils niedriger, die Anzahl der Jahre im Gefängnis folglich geringer sind als bei Strategie s_{i1}: Gesteht Spieler 1 nicht (s_{11}), so beträgt die Strafe drei oder sieben Jahre Gefängnis für ihn. Gesteht Spieler 1 jedoch (s_{12}), so wird er der Strategiekombination (s_{12}, s_{21})

zufolge entweder straflos freigelassen oder verbringt aufgrund der Strategiekombination (s_{12}, s_{22}) fünf Jahre im Gefängnis.

Weil die Strategie s_{i2} für beide Spieler des Gefangenendilemmas die vorteilhaftere Auszahlung verspricht, wird die Strategie s_{i2} als *dominant* gegenüber der Strategie s_{i1} bezeichnet und als Lösung des Spieles nach dem Kriterium der Dominanz identifiziert. Als ein *Gleichgewicht* des Spieles wird die Strategiekombination (s_{12}, s_{22}) bezeichnet, weil es für keinen der beiden Spieler Sinn macht, eine andere Strategie zu verfolgen, d.h. einseitig von der Kooperation abzuweichen, wenn davon auszugehen ist, dass der Gegenspieler bei seiner (kooperativen) Entscheidung bleibt und nicht defektiert. Eine solche Strategiekombination, bei der es für keinen der Spieler sinnvoll ist, von der gewählten Strategie abzuweichen, wird auch als *Nash-Gleichgewicht* bezeichnet. [132] Um eine Strategiekombination auf ihre Nash-Eigenschaft zu überprüfen, muss „für jeden Spieler einzeln geprüft werden, ob einseitiges profitables Abweichen zu einer alternativen Strategie möglich ist. Wenn dies nicht möglich ist, wird kein Spieler eine Veranlassung haben, einseitig von der gewählten Strategiekonfiguration abzuweichen. Dann liegt ein Nash-Gleichgewicht vor.“ [133] „Allein das Eigeninteresse der Spieler“, so ergänzen Berninghaus et al., „führt dazu, dass die Spieler keinen Anreiz haben, ein Nash-Gleichgewicht einseitig zu verlassen.“[134] Wie in der nachfolgend wiedergegebenen Auszahlungsmatrix veranschaulicht, ist das Nash-Gleichgewicht nicht in jedem Falle mit optimalen Auszahlungen verbunden:

[132] Vgl. Nash 1950 sowie Holler und Illing 2009, S. 9–12.
[133] Berninghaus et al. 2010, S. 24.
[134] Berninghaus et al. 2010, S. 24.

		Spieler 2	
		Nicht-Gestehen (s_{21})	Gestehen (s_{22})
Spieler 1	Nicht-Gestehen (s_{11})	3,3	7,0
	Gestehen (s_{12})	0,7	5,5

Tabelle 3: Auszahlungsmatrix des Gefangenendilemmas mit hervorgehobenem Nash-Gleichgewicht.

Im Gefangenendilemma ist es einem Spieler möglich, seine optimale Strategie ohne Berücksichtigung der Strategie des Gegenspielers zu finden und zu verfolgen – jeder Spieler verfügt über eine strikt dominante Strategie. So gilt für jeden Spieler und jede Spielerin i ungeachtet der Strategiewahl k des Gegenspielers j das Verhältnis $u_i(s_{i2}, s_{jk}) > u_i(s_{i1}, s_{jk})$. Die Auszahlung, die Spieler i mit seiner Wahl der Strategie s_{i2} erhält, ist im vorliegenden Spiel immer größer als die Auszahlung, die er (oder sie) bei Wahl der Strategie s_{i1} erhielte, das Gefangendilemma ist „im Prinzip“[135] gelöst.

3.4.5 Kooperatives und egoistisches Agieren

Das namensgebende Dilemma des Gefangenendilemmas offenbart sich, wenn beiden Spielern die Möglichkeit gegeben wird, sich vor ihrer Aussage abzusprechen: Auch wenn sich beide Spieler versprechen, nicht zu gestehen, sich folglich für die Strategie s_{i1} zu entscheiden, so ist dennoch nicht sichergestellt, dass beide Spieler auch tatsächlich zu ihrem Wort stehen und sich kooperativ verhalten. So könnte Spieler 1 der Versuchung erliegen, seine individuelle Auszahlung zu maximieren, indem er gegen seinen Mitspieler aussagt, weil er davon ausgeht, dass jener sich an die Abmachung hält – und so wird Spieler 2 aufgrund seiner vertrauensseligen Kooperationsbereitschaft die längste Zeit im Gefängnis verbringen, wird folglich in den nächsten Jahren über sehr viel Zeit und Muße verfügen, um über Auszahlungs- und Nutzenfunktion, Kooperation und Nicht-Kooperation, dominante Strategien und das Gefangenendilemma nachzudenken.

[135] Wessler 2012, S. 23.

Weil verbindliche Absprachen im vorgetragenen Beispiel nicht möglich sind, wird das Gefangenendilemma als ein nicht-kooperatives Spiel bezeichnet. Im Gegensatz dazu ist es Spielern in kooperativen Spielen möglich, verbindliche Abmachungen zu treffen. Verbindliche Abmachungen setzen voraus, dass die Spieler miteinander kommunizieren können und dass die Abmachung von einem externen Mechanismus erwirkt werden kann. So ließe sich das Gefangenendilemma um eine Kontrollinstanz ergänzen, die beide Spieler vor Beginn des Spieles dazu verpflichten würde, miteinander zu kooperieren. Hielte sich ein Spieler nicht an die Abmachung, so müsste er eine Strafe bezahlen, die den Nutzen der Defektion seines egoistischen Verhaltens überstiege.

Als eine der mannigfaltigen Grundfragen, mit der sich die Spieltheorie nach von Neumann und Morgenstern beschäftigt, wurde eingangs die Frage gestellt, ob sich die Wiederholbarkeit eines Spieles auf die Entscheidungen der Spielerinnen und Spieler auswirkt. Für das Gefangenendilemma lässt sich die Frage positiv beantworten: Wird das Gefangenendilemma über mehrere Runden gespielt, so ist es notwendig, dass die Spieler ihre Strategien modifizieren und anpassen und so ist es unabdinglich, dass die Spieler Strategien verfolgen, die sich von ihren Gegenspielern nicht leicht durchschauen und antizipieren lassen. Soll im Folgenden kurz eingegangen werden auf das wiederholte Gefangenendilemma, so wird im anschließenden Kapitel zuerst der – bislang umgangssprachlich verwendete – Begriff der Strategie angenähert und so werden zunächst unterschiedliche Arten von Strategien differenziert.

3.4.6 Strategien und Entscheidungen

Wenngleich Wessler darauf hinweist, dass der Begriff der Strategie in der Literatur zur Spieltheorie problematisch ist[136], so lässt sich die Strategie Wessler zufolge beschreiben über die zu Beginn des Kapitels mit von Neumann und Morgenstern vorgenommene Distinktion von Spielzug und Zugwahl: „Die […] *Spielzüge* sind die jeweils zur Verfügung stehenden Aktionen, die durchgeführt werden können […]. Eine Strategie umfasst die *Entscheidung* für einen oder mehrere Spielzüge.“[137]

[136] Vgl. Wessler 2012, S. 19.
[137] Wessler 2012, S. 19, Kursivschreibung nach dem Autor.

Grob unterscheiden lassen sich reine Strategien von gemischten Strategien und Langzeitstrategien. So verfügt jeder Spieler des Gefangenendilemmas über die zwei Strategien *Kooperation* und *Defektion* und entscheidet sich dazu, genau eine bestimmte Strategie zu verfolgen. Die Strategien der Gefangenen werden hierbei als *reine* Strategien bezeichnet. Würden die Gefangenen jedoch ihre Entscheidung zufallsbasiert treffen, so würde jene Strategie, einen Würfel über die Handlung entscheiden zu lassen, als *gemischte* Strategie bezeichnet werden. [138] Langzeitstrategien schließlich setzen mehrere Spielrunden voraus und könnten sich im iterierten Gefangenendilemma wie folgend gestalten:

- $Strategie_1$: Kooperiere in jeder Runde des Spiels
- $Strategie_2$: Defektiere in jeder Spielrunde
- $Strategie_3$: Kooperiere in der ersten Spielrunde und verhalte Dich anschließend nicht kooperativ
- $Strategie_4$: Beginne das Spiel kooperativ und imitiere anschließend das Verhalten des Gegenspielers

Die Spieltheorie nach von Neumann und Morgenstern erwartet, dass jede Spielerin und jeder Spieler des Spiels rational agiert, folglich bestrebt ist, die bestmögliche Entscheidung zu treffen, d.h. sein Handeln am größtmöglichen Nutzen, an der maximalen Auszahlung, zu orientieren. [139] Je nach Informiertheit, je nach Kenntnisstand über die Umwelt der Spieler lassen sich Entscheidungen in unterschiedliche Kategorien klassifizieren:[140]

1. Entscheidungen unter Gewissheit oder Sicherheit: Der Spieler kann alle möglichen Folgen seiner Handlungen voraussagen, d.h. dem Spieler sind die Folgen aller seiner möglichen Handlungen für die Spielwelt bekannt.

[138] Vgl. u.a. das Kapitel *Gemischte Strategien* in Berninghaus et al. 2010, S. 28–34.
[139] Vgl. Davis 2005, S. 19.
[140] Vgl. Diekmann 2009, S. 71 und Prechtl 2008, S. 140. Auf Entscheidungen von Spieler- und Agentengruppen soll an späterer Stelle noch eingegangen werden. Für einen Überblick über die Entscheidungstheorie sei an dieser Stelle zunächst das Kapitel *Entscheidungstheorie, Spieltheorie und rationales Handeln* in Diekmann 2009, S. 69–89 und der Band *Entscheidungstheorie* (Laux et al. 2012) sowie Wessler 2012 empfohlen.

2. Entscheidungen unter Risiko: Die Spielerin ist sich der Wahrscheinlichkeit des Auftretens bestimmter Ereignisse aufgrund ihrer Handlungen in der Spielwelt bewusst.
3. Entscheidungen unter völliger Ungewissheit bzw. Unsicherheit lassen sich in zwei Unterpunkte unterteilen:
    a. Dem Spieler sind nicht alle möglichen Folgen seiner Handlungen für die Spielwelt bekannt; zudem ist es dem Spieler nicht möglich, Wahrscheinlichkeiten im Auftreten bestimmter Ereignisse aufgrund seiner Handlungen auszumachen.
    b. In strategischen Entscheidungssituationen erfährt sich die Spielerin eingebettet in eine Multiagentenumgebung: Die Handlungsintention der Spielerin und das Ergebnis ihres Handelns ist abhängig und wird beeinflusst von dem Agieren anderer Spielerinnen und Spieler in der Spielwelt.

3.4.7 Wie du mir, so ich dir: das wiederholte Gefangenendilemma

Angenommen, das zuvor referierte Gefangenendilemma gestalte sich über mehrere Spielrunden – wie sollten die Spieler des Gefangenendilemmas nun in jeder Runde des Spieles vorgehen? Sollten sie ihre Strategiewahl zu Beginn jeder Runde zufällig treffen oder ließe sich eine Langzeitstrategie ausfindig machen, durch die sich konstant gute Auszahlungsergebnisse erzielen ließen? Eine Antwort auf jene Frage liefert die Strategie *Tit for Tat*, die sich in einem Wettbewerb als äußerst erfolgreiche Strategie offenbarte. So traten 1979 zahlreiche [141] Computerprogramme im Rahmen des von dem US-amerikanischen Politikwissenschaftlers Robert Axelrod ausgerichteten Turnieres gegeneinander und gegen sich selbst an, um das Gefangenendilemma 200 Runden lang zu spielen – und das Spiel möglichst für sich zu entscheiden.[142] Den Wettbewerb entschied Anapol Rapoports Programm *Tit for Tat*, zu Deutsch *Wie du mir, so ich dir*. Rapoport, zum Zeitpunkt des Wettbewerbes Psychologie- und Philosophiedozent

[141] Wessler 2012 weist darauf hin, dass sich die genaue Anzahl der zum Wettbewerb eingereichten Entscheidungsregeln und Computerprogramme nicht genau bestimmen lässt (vgl. Wessler 2012, S. 138f.).
[142] Ausführlich beschreibt Robert Axelrod den Wettbewerb sowie seine Folgen und Implikationen in seiner Monographie *Die Evolution der Kooperation* (Axelrod 2009).

an der Universität Toronto,[143] pflanzte seinem Computerprogramm die folgende Langzeitstrategie ein:

- Beginne das Spiel kooperativ, d.h. gestehe zu Beginn des Spieles das Verbrechen.
- Imitiere in den folgenden Spielrunden immer das Verhalten des Gegners:
  - Hat sich der Gegner in der vergangenen Spielrunde kooperativ verhalten, dann verhalte dich in der aktuellen Spielrunde auch kooperativ.
  - Verhält sich der Gegner in der aktuellen Spielrunde nichtkooperativ, dann wähle in der nächsten Spielrunde Nichtkooperation als Strategie.

Wenngleich simpel, so ist die Langzeitstrategie *Tit for Tat* doch wirkungsvoll: Sie beginnt mit der freundlichen Intention, zu kooperieren und reagiert schnell auf Defektion des Gegners, indem sie das gegnerische Verhalten nachahmt. Zwar ist *Tit for Tat* nicht „der Weisheit letzter Schluss“, wie Wessler anmerkt,[144] jedoch erwies sich die Strategie sowohl im 1979 veranstalteten als auch in später ausgetragenen Turnieren als äußerst erfolgreich. Den Erfolg der Langzeitstrategie *Tit for Tat* sieht Axelrod in den folgenden Eigenschaften begründet:

> „Was den robusten Erfolg von TIT FOR TAT erklärt, ist die Kombination, freundlich zu sein, zurückzuschlagen, Nachsicht zu üben und verständlich zu sein. Freundlichkeit schützt vor überflüssigen Scherereien. Zurückschlagen hält die andere Seite nach einer versuchten Defektion davon ab, diese unbeirrt fortzusetzen. Nachsicht ist hilfreich bei der Wiederherstellung wechselseitiger Kooperation. Schließlich erleichtert Verständlichkeit die Identifikation und löst dadurch langfristige Kooperation aus.“[145]

An späterer Stelle der vorliegenden Arbeit wird noch näher auf Kooperation und Defektion der miteinander Agierenden in Brettspielen einzugehen sein, soll *Tit for Tat* als eine mögliche Langzeitstrategie der zu implementierenden brettspielenden

[143] Vgl. Wessler 2012, S. 93.
[144] Wessler 2012, S. 95.
[145] Axelrod 2009, S. 48.

Agenten besprochen werden. Abschließend jedoch sei der Fokus dieses Kapitels zur Spieltheorie nach von Neumann und Morgenstern gerichtet auf spieltheoretische Konzepte, die der Beschreibung von Brettspielen dienlich sind. Denn anders als das Gefangenendilemma gestalten sich Brettspiele wie Dame, Schach oder Backgammon als rundenbasierte Nullsummenspiele, in denen der Gewinn des einen Spielers der Verlust des anderen Spielers ist.

3.4.8 Nullsummenspiele

In Nullsummenspielen verfolgen die Spielerinnen und Spieler einander entgegengesetzte Interessen; die Minimierung der gegnerischen Auszahlung führt hierbei zur Maximierung der eigenen Auszahlung, des einen Freud' ist des anderen Leid, wie zunächst im folgenden Beispiel-Spiel *Matching-Pennies* nach Shoham und Leyton-Brown [146] veranschaulicht: Zwei Spieler, die jeweils über eine zweiseitige Münze verfügen, entscheiden sich in jeder Spielrunde für eine Seite des Geldstückes und legen die Münze mit einer der beiden Seiten *Kopf* (s_{i1}) oder *Zahl* (s_{i2}), für die sie sich vorab entschieden haben, gleichzeitig auf den Tisch. Welche Seiten der auf den Tisch gelegten Münzen sichtbar sind, entscheidet über das Spielergebnis: Zeigen beide Münzen die gleiche Seite, so erhält Spieler 1 sowohl seine Münze, als auch die Münze des Gegners. Stimmen die beiden Seiten nicht überein, so erhält Spieler 2 beide Münzen.

Wie der Begriff des Nullsummenspieles bereits andeutet, beträgt die Summe der Auszahlungen in jeder Zelle der Auszahlungsmatrix den Wert Null: Was die eine Spielerin gewinnt, das verliert der andere Spieler, somit gilt für zwei Spieler die Gleichung $u_i(s) = -u_j(s)$. In Nullsummenspielen genügt es folglich, einzig die Auszahlung des einen Spielers anzugeben, wie in der folgenden Auszahlungsmatrix für Spieler 1 vorgenommen:

[146] Vgl. Shoham und Leyton-Brown 2009, S. 57f.

		Spieler 2	
		Kopf (s_{21})	Zahl (s_{22})
Spieler 1	Kopf (s_{11})	+1	-1
	Zahl (s_{12})	-1	+1

Tabelle 4: Auszahlungsmatrix des Matching-Pennies Spieles mit reduzierten Auszahlungen

Entscheidet sich Spieler 1 für die Strategie *Kopf* und legt sein Gegenspieler die Münze ebenfalls mit ihrer Kopfseite auf den Tisch, so erhält Spieler 1 die maximale und Spieler 2 die minimale Auszahlung. Verfolgt Spieler 1 jedoch die Strategie s_{12} und sein Gegenspieler die Strategie s_{21}, so verliert Spieler 1 das Spiel mit einer Auszahlung $u_1(s_{12}, s_{21}) = -1$; Spieler 2 erhält in jenem Falle die maximale Auszahlung. Das *Matching-Pennies* Spiel verdeutlicht hierbei, wann es sinnvoll ist, eine reine oder eine gemischte Strategie zu verfolgen: Im Unterschied zum Gefangenendilemma empfiehlt es sich beim *Matching-Pennies* Spiel, keine reine Strategie zu verfolgen, die vom Gegenspieler mitunter schnell zu durchschauen wäre, sondern die Wahl der Münzseite dem Zufall zu überlassen, folglich eine gemischte Strategie zu verfolgen.

3.4.9 Sequentielle Entscheidungen: Spielbaum und Extensivform

Ungemein vielfältiger als die zuvor besprochenen Spiele gestalten sich jene, die über mehrere Runden gespielt werden – Spiele, in denen die Agierenden ihre Strategien nicht gleichzeitig, sondern nacheinander wählen. Mit dem Werkzeug der Normalform ließen sich einzig Spiele betrachten und durchdringen, bei denen die Spielerinnen und Spieler Entscheidungen simultan treffen. Die Normalform von Spielen berücksichtigt jedoch nicht, in welcher zeitlichen Reihenfolge die Spieler ihre Strategien wählen. Brettspiele wie Dame, Schach oder Backgammon gestalten sich deutlich komplexer als die zuvor referierten Spiele und generieren von Spielrunde zu Spielrunde höchst vielfältige Entscheidungssituationen. Solche Spiele lassen sich mit der Extensivform von Spielen beschreiben, die auch als *strategische Form* bezeichnet wird.

Die Extensivform visualisiert den zeitlichen Verlauf eines Spieles in einem Spielbaum und repräsentiert sowohl die Reihenfolge, in der Spieler agieren, als

auch den Informationsstand der Spielerinnen und Spieler zu jedem Zeitpunkt des Spieles. Ein Spielbaum ist ein zusammenhängender endlicher Graph ohne Schleifen, bei dem ein Knoten aus der Gesamtknotenmenge den Wurzelknoten repräsentiert. Zusammenhängend ist ein Graph, wenn „jeder Knoten mit jedem anderen Knoten durch einen Streckenzug verbunden ist.“[147]

Wie in der unten wiedergegebenen Abbildung illustriert, stellt ein Spielbaum die Zugmöglichkeiten der Spieler als Knoten dar und repräsentiert Zugwahlen durch Kanten, die Verbindungen zwischen den Knoten herstellen. Seinen Lauf nimmt das Spiel ausgehend vom Wurzelknoten bis hin zu den End- oder Auszahlungsknoten, die als nicht ausgefüllte Kreise dargestellt sind. Entscheidungen treffen die Spielerinnen und Spieler abwechselnd an den schwarz ausgefüllten Knoten, den Entscheidungsknoten:

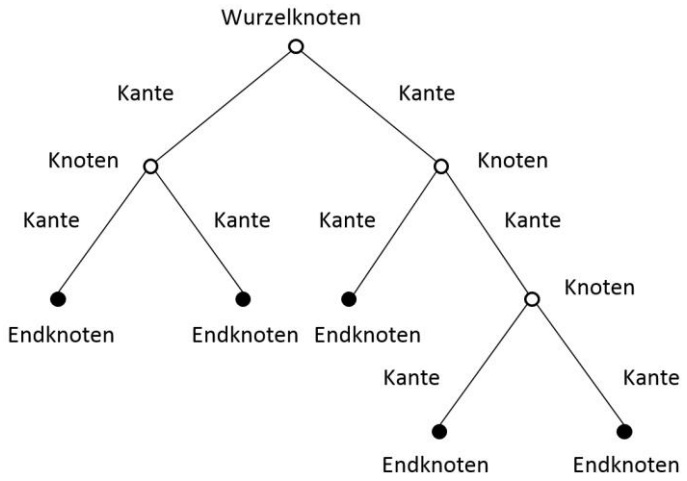


Abbildung 9: Informelle Darstellung des Spielbaumes

Charakterisiert ist der Spielbaum durch die endliche Menge seiner Knoten K.[148] Im zuvor wiedergegebenen Spielbaum beträgt die Menge der Knoten $|K| = 9$. Aus der Gesamtmenge der Knoten K differenzieren sich die Entscheidungsknoten, an denen sich die Spielerinnen und Spieler für eine Strategie entscheiden müssen, von den End- oder Auszahlungsknoten, die als schwarz ausgefüllte Kreise visualisiert sind. Jene Endknoten sind beschrieben durch die Menge Z und stellen

[147] Güth 1999, S. 36.
[148] Die Ausführungen und Erläuterungen der Extensivform folgen Berninghaus et al. 2010, S. 92 ff. Vgl. auch die formale Definition eines Spieles in Extensivform bei Shoham und Leyton-Brown 2009, S. 114.

Ergebnisse der Entscheidungsprozesse im Laufe der Spielpartie dar. Die Menge der Entscheidungsknoten ist bestimmt durch die Differenz der Knotenmenge K von den Endknoten Z. An jedem Entscheidungsknoten aus $X \coloneqq K - Z$ werden von den Spielern Entscheidungen getroffen, die den Spielverlauf generieren und bestimmen.

Welcher Spieler an einem bestimmten Knoten des Spielbaumes an der Reihe ist, lässt sich repräsentieren durch die Spielerzerlegung P. Beiden Spielern des oben dargestellten Spieles ist gemein, dass sie an jedem Entscheidungsknoten aus der Menge X darüber informiert sind, welche Spielzüge und Strategiekombinationen zum jeweiligen Entscheidungsknoten geführt haben. Spiele, in denen jeder Spieler die Strategiemengen S_i und die Auszahlungsfunktionen $u_i(s)$ aller Spieler bekannt sind, werden als Spiele mit *vollständiger Information* bezeichnet. Ein Spieler verfügt über *perfekte Information*, wenn ihm während eines Spielverlaufes alle vorausgehenden Züge der Mitspieler bekannt sind. So weiß im Schach- und im Damespiel jeder Spieler und jede Spielerin zu jedem Zeitpunkt des Spieles, aus welchen vorangegangenen Entscheidungen die aktuelle Spielkonfiguration resultiert und so ist jedem Spieler des Backgammonspieles zu jedem Spielzeitpunkt bekannt, welche früheren Würfelwerte und individuellen Entscheidungen der Spieler das Spiel zu seiner aktuellen Spielkonfiguration formte und führte.

Als Spiele mit *imperfekter Information* hingegen werden Spiele signifiziert, in denen es Spielern nicht möglich ist, die Handlungen der anderen Spieler zu beobachten. Solche Spiele lassen sich unter Berücksichtigung der individuellen Informationsmengen der Spielerinnen und Spieler beschreiben.[149] Im Spielbaum lässt sich imperfekte Information durch eine gestrichelte Linie repräsentieren, die für den Spieler nicht unterscheidbare Knoten kennzeichnet. So visualisiert die horizontale gestrichelte Linie in der folgenden Abbildung, dass Spielerin 2 über keine Kenntnis darüber verfügt, welche Initialentscheidung Spieler 1 am Wurzelknoten x_0 getätigt hat. Folglich muss sich Spielerin 2 für die Strategie s_{21}

[149] Vgl. Berninghaus et al. 2010, S. 93 f.

oder s_{22} entscheiden, ohne zu wissen, ob sie sich zum Zeitpunkt der Entscheidung an Knoten x_1 oder x_2 befindet:

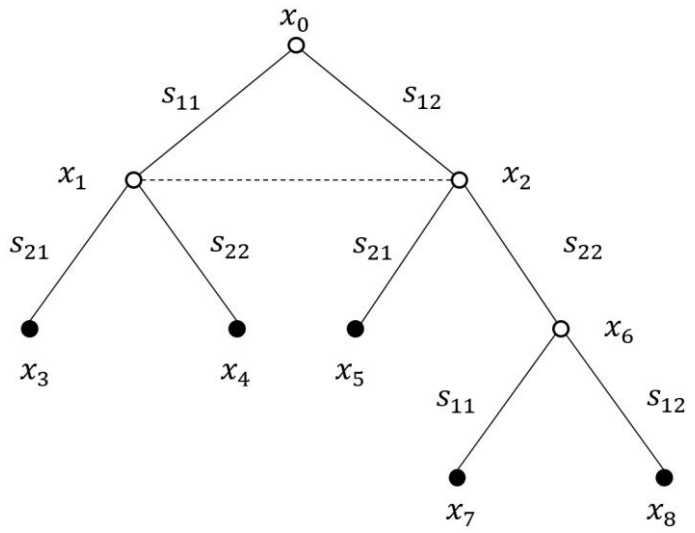


Abbildung 10: Spielbaum eines abstrakten Spieles mit unvollkommener Information.

Abschließend und zusammenfassend sei ein Spiel in Extensivform als Tupel definiert:[150]

Ein Spiel Γ in Extensivform ist vollständig beschrieben durch das Tupel $\Gamma = (N, K, Z, P, I, C, u, P_r)$. Hierbei bezeichnet

- N die finite Anzahl der Spielerinnen und Spieler des Spiels Γ aus der Spielermenge $N = \{1, \dots, n\}$;
- K die Menge aller Knoten;
- Z die Menge der End- oder Auszahlungsknoten;
- P die Spielerzerlegung über alle Entscheidungsknoten $X = K - Z$, die bestimmt, welcher Spieler an einem bestimmten Knoten des Spielbaumes an der Reihe ist;
- H die Informationszerlegung, die den Informationsstand der Spieler repräsentiert;
- C die Aktionsmengen, die den Spielern Handlungsalternativen eröffnen;

[150] Vgl. Berninghaus et al. 2010, S. 92 ff.

- u die Auszahlungsfunktion, definiert als $u : Z \mapsto \mathbb{R}^n$;
- P_r die Menge der Wahrscheinlichkeitsverteilungen des Zufallsspielers. Der Zufallsspieler, der auch als „Natur“ bezeichnet wird, führt Spielzüge mit bekannten Wahrscheinlichkeiten aus.

Zur Annäherung an die Begrifflichkeiten sei schließlich das folgende Schachspiel betrachtet, das in seiner reduzierten Variante auf einem 4×4 Spielfelder kleinen Spielbrett mit den Regeln des Schachspieles gespielt wird. Zu Beginn des Spieles, d.h. an Wurzelknoten x_0 des im Folgenden dargestellten Spielbaumes, steht Spieler 1 (weiß) vor der Entscheidung, seinen König auf das unmittelbar angrenzende linke oder das rechte Spielfeld zu setzen: Spieler 1 kann sich entweder für die Strategie s_{11} oder die Strategie s_{12} entscheiden, die zu dem Terminalknoten $x_1$und dem Kindknoten x_2 führen:

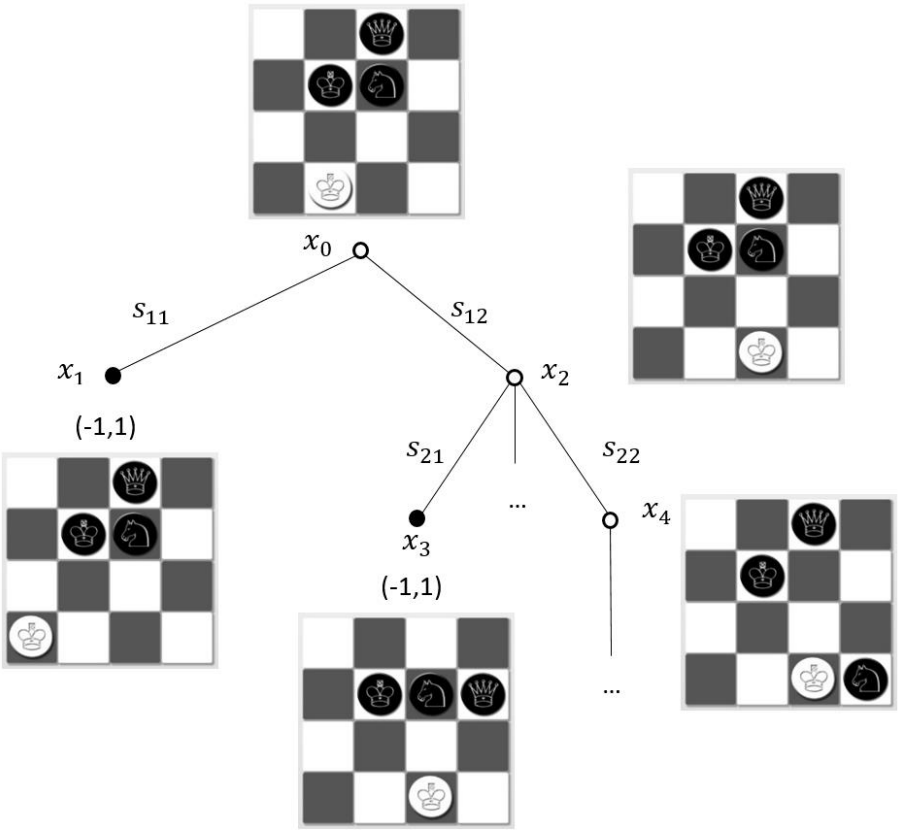


Abbildung 11: Spielbaum des auf eine Spielwelt von 4x4 Felder reduzierten Schachspieles. Um die unterschiedlichen Zugwahlen zu veranschaulichen, finden sich die Knoten des Spielbaumes ergänzt um Bilder der Spielkonfigurationen.

Entscheidet sich Spieler 1 für die Strategie s_{11}, so verliert er das Spiel, weil er seinen eigenen König zugunfähig gesetzt hat. Die Auszahlung für Spieler 1 beträgt

im Nullsummenspiel Schach somit -1; Spielerin 2 (schwarz) gewinnt das Spiel mit einer Auszahlung von 1. Entscheidet sich Spieler 1 am Wurzelknoten x_0 für die Strategie s_{12}, so muss Spielerin 2 entscheiden, welche ihrer Spielfiguren sie auf ein bestimmtes Feld des Spielbrettes bewegt.

Bereits in seiner auf wenige Spielfelder reduzierten Variante offenbart sich die hohe Komplexität des Schachspieles: Mit ihren drei Spielfiguren verfügt Spielerin 2 über rund ein Dutzend Möglichkeiten, den Spielverlauf zu beeinflussen. Um die Übersichtlichkeit zu wahren, ist die Darstellung des oben wiedergegebenen Spielbaumes somit auf wenige Spielkonfigurationen reduziert, mögliche weitere Entscheidungen und Verzweigungen wurden mit drei Punkten abgekürzt. Entscheidet sich Spielerin 2 im rechten Teilbaum an Knoten x_2 für die Strategie s_{21}, so setzt sie den gegnerischen König schachmatt und so gewinnt sie das Spiel mit einer positiven Auszahlung von 1. Wählt Spielerin 2 an Knoten x_2 die Strategie s_{22}, so ist es dem König von Spieler 1 möglich, den gegnerischen Springer zu schlagen – und vielleicht im späteren Verlauf des Spieles ein Unentschieden, d.h. eine Auszahlung von 0, zu erreichen.

3.5 Zusammenfassung und Ausblick

In den vorangangenen Kapiteln wurden zunächst Theorien des Spiels betrachtet und vorgestellt. Auf Huizingas Intention, das Spiel des Menschen zu analysieren und Eigenschaften des Spieles zu extrahieren folgte Juuls definitorische Arbeit, das Spiel in möglichst vielen seiner Facetten als regelbasiertes System mit einem veränderlichen, quantitativ bestimmbaren Spielergebnis zu charakterisieren. Dienten jene einleitenden beiden Kapitel dazu, den Gegenstand der vorliegenden Arbeit anhand seiner Erscheinungsweisen zu beschreiben und zuallererst zu formulieren, was sich hinter dem Spiel des Menschen verbirgt und was sich mit dem Spiel des Menschen verbindet, so bietet die referierte Spieltheorie nach von Neumann und Morgenstern ein effizientes Ausdrucksmittel in der vollständigen Beschreibung und Visualisierung von Spielen. So wurde das Spiel zunächst in seiner Normalform erläutert und mit dem Gefangenendilemma das wohl bekannteste Beispiel-Spiel der Spieltheorie vorgestellt. Daran anschließend wurden Nutzen- und Auszahlungsfunktion und Strategien besprochen, wurde das Nash-Gleichgewicht skizziert sowie auf die Extensivform von Spielen eingegangen.

Gegenstand der nachfolgenden Betrachtungen sind klassische Brettspiele wie Dame, Schach und Backgammon in ihren unterschiedlichen Ausprägungen und Regelvarianten. Mit dem spieltheoretischen Wort- und Werkzeugschatz lassen sich jene Spiele bezeichnen als rundenbasierte Zweispieler-Spiele mit perfekter Information. Was diese Spiele so besonders macht, ist ihre Schlichtheit, die jedoch hochkomplexe Spielwelten eröffnet: Eine äußerst überschaubare Anzahl von Spielregeln und Spielmaterialien führen zu einem höchst anspruchsvollen Spielverlauf, in dem sich mitunter selbst unter Aufwendung hoher Rechenleistung nicht berechnen lässt, welcher Zug der mannigfaltigen Zugmöglichkeiten eines Spielers oder einer Spielerin die beste Zugwahl ist.[151]

[151] Vgl. dazu auch Nilsson 2010, S. 319 ff.

4 Artifizielles Spielvermögen in digitalen Brettspielen – Methoden und Implementationen

> "How many possible checkers positions are there? Joe Culberson figured out the answer: 500,995,484,682,338,672,639. In other words, roughly five-hundred billion billion, or 5×10^{20}. To put a number this big into perspective, imagine the surface of all the earth's land mass as being equivalent to the number of possible checkers positions. Then one position is roughly equivalent to one thousandth of a square inch. Alternatively, pretend that the Pacific Ocean is empty, and you have to fill it using one small cup. The number of cupfuls of water that you pour is roughly the same as the numbers of checkers positions."[152]

Brettspiele wie Dame, Schach, Backgammon oder Go generieren Spielbäume von unermesslicher Komplexität,[153] die sich häufig nur allegorisch imaginieren lässt, wie das oben wiedergegebene Beispiel Schaeffers exemplifiziert. Mit seiner geringen Anzahl von jeweils sechs unterschiedlichen Spielsteinen und einer überschaubaren Menge von Spielregeln verfügt das Schachspiel über einen durchschnittlichen Verzweigungsfaktor[154] von 35. Eine Spiellänge von rund 50 Zügen pro Spieler resultiert somit in einem Spielbaum, der über 35^{100} Knoten verfügt.[155] Schnell wird deutlich, dass die Unterscheidung zwischen sinnvollen und

[152] Schaeffer 2009, S. 12 f.

[153] Der Begriff der Komplexität sei verstanden als quantitative Größe: Mit der Komplexität eines Spielbaumes ist die Gesamtzahl seiner Knoten bezeichnet und ist implizit beschrieben, wie viele Knoten des Spielbaumes besucht werden müssen, um den spieltheoretischen Wert der Initialkonfiguration des Spieles zu berechnen. Feingliedrig nähert sich Mitchell dem Begriff der Komplexität, wenn sie Komplexität (u.a.) als quantitative Größe, als Entropie, als thermodynamische Tiefe oder Komplexität als fraktale Dimension (*Complexity as Fractal Dimension*) bestimmt (vgl. Mitchell 2009, S. 94–111).

[154] Der Verzweigungsfaktor signifiziert die durchschnittliche Anzahl möglicher gültiger Züge in jedem Knoten des Spielbaumes bzw. jeder Spielsituation.

[155] Vgl. Russell und Norvig 2012, S. 206.

weniger sinnvollen Zügen, zwischen schlechten und optimalen Spielzügen zu jedem Zeitpunkt des Spieles eine äußerst diffizile ist.

In den nächsten Kapiteln wird zunächst zu erläutern sein, welche Lösungsmechanismen sich in Brettspielen anbieten und welche Verfahren der Berechnung optimaler Züge, gar zur vollständigen Lösung unterschiedlicher Brettspiele dienen. Ausgehend von der Differenzierung problemlösender Verfahren und Algorithmen in klassische Verfahren der künstlichen Intelligenz und in Verfahren der Computational Intelligence werden mit dem Minimax-Algorithmus und seinen Verfeinerungen zunächst Mechanismen zur performanten Suche im Spielbaum besprochen. Daran anschließend werden Problemlösungsmechanismen vorgestellt, die zufallsbasiert verfahren oder konnektionistische und lernbasierte Ansätze verfolgen, um möglichst optimale Zugmöglichkeiten in Brettspielen zu identifizieren. Ausgehend von verschiedenartigen Einschränkungen und Problemen, die jene Ansätze mit sich bringen, wird an späterer Stelle der eigene Ansatz vorgestellt, der die Implementation einer browserbasierten spielübergreifenden künstlichen Intelligenz auf Grundlage von Monte Carlo Verfahren beschreibt.

4.1 Künstliche Intelligenz als Suche nach optimalen Spielentscheidungen

Künstliche Intelligenz, wie sie sich in diesem Kapitel und seinen Unterkapiteln manifestiert, ist vor allem eines: die möglichst schnelle und tunlichst ressourcensparende Suche nach optimalen Zügen im Spielbaum. Lässt sich bei trivialen Spielen wie Tic Tac Toe in Sekundenbruchteilen vorausberechnen, welche Zugmöglichkeit einer Spielerin oder eines Spielers zum Gewinn führt, so stellen das Dame-, Schach- oder Backgammonspiel ungemein höhere Anforderungen an Such- und Problemlösungsverfahren, die darüber hinaus noch durch umfangreiche Eröffnungsdatenbanken und breit gefächertes Endspielwissen ergänzt werden, um leistungsstark spielen zu können.

Wie gut ein Suchverfahren arbeitet, lässt sich mit Russell und Norvig über vier Charakteristika festmachen. So ist die Güte eines Suchverfahrens bestimmt über die Vollständigkeit der Suche, die Optimalität sowie die Zeit- und Speicherkomplexität des Verfahrens. Sind mit der Zeit- und Speicherkomplexität die Ressourcenanforderungen und die Dauer bis zur Auffindung des Gesuchten signifiziert, so ist mit der Vollständigkeit eines Suchverfahrens die Frage gestellt, ob der Algorithmus eine Lösung für das gestellte Suchproblem findet, sofern eine Lösung existiert. Optimal verfährt ein Suchverfahren, wenn es die schnellste zum Ziel führende Lösung für das Problem findet.[156]

Mit dem Minimax-Algorithmus und seinen Verfeinerungen werden in den nachfolgenden Kapiteln ebensolche Such- und Bewertungsverfahren vorgestellt, mit denen sich – theoretisch – für jede Spielerin und für jeden Spieler zu jedem Zeitpunkt des Spieles optimale Spielzüge[157] identifizieren lassen. Praktisch jedoch ist es in nichttrivialen Spielen wie Schach oder Dame nicht realisierbar, alle möglichen Zugmöglichkeiten und ihre Auswirkungen auf den Spielverlauf *on the fly* zu berechnen. Bevor in den folgenden Unterkapiteln auf jene Problematik eingegangen und das Minimax-Verfahren und seine Erweiterungen anhand des Beispiel-Spieles Tic Tac Toe veranschaulicht werden, seien zunächst drei basale Methoden zur Suche in Spielbäumen besprochen, die ihren Einsatz im Rahmen der zu besprechenden Verfahren finden: die Breiten- und Tiefensuche sowie die iterativ vertiefende Tiefensuche.

4.1.1 Problemlösen durch Suchen – Elementare Suchverfahren

Am Beispiel der einleitend wiedergegebenen Schachpartie offenbart sich, wie knifflig sich die Entscheidungsfindung im komplexen Suchraum des Schachspieles gestaltet: Weiß ist am Zug und verfügt mit seinen zehn Spielfiguren über 33 unterschiedliche Zugmöglichkeiten auf dem Spielbrett. Nach dem Zug von Weiß ist Schwarz an der Reihe, aus den mannigfaltigen Bewegungsmöglichkeiten seiner

[156] Vgl. Russell und Norvig 2012, S. 115.

[157] Aufgrund terminologischer Unschärfe wird von der Gepflogenheit abgesehen, in Brettspielen den Zug eines Spielers als *ply* zu bezeichnen (vgl. Samuel 1963). Jede Spielhandlung eines Spielers, die im Spielbaum von einem Knoten x_m nach x_n führt, soll im Kontext der betrachteten Spiele als Spielzug eines Spielers und als Spielrunde verstanden sein (vgl. hierzu auch Millington und Funge 2009, S. 668).

eigenen Spielfiguren diejenige zu identifizieren, die den Lauf des Spieles zum Positiven wendet – oder ein Schachmatt verhindert.

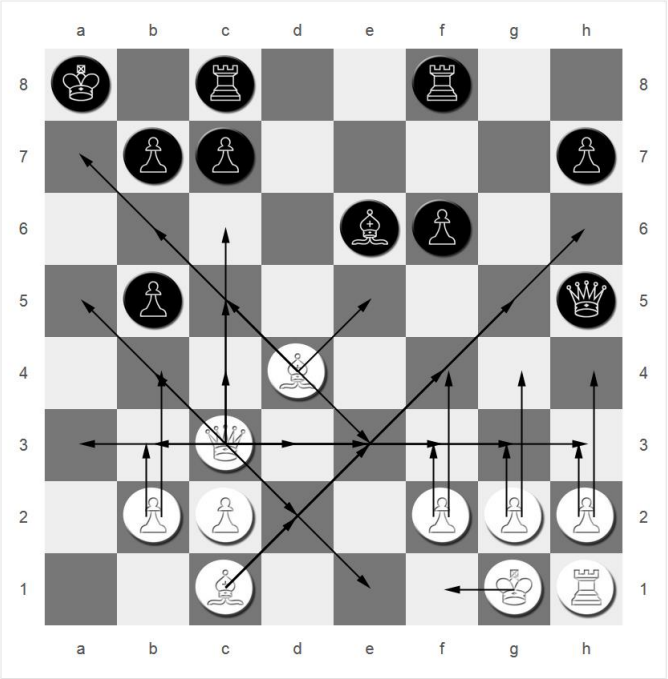


Abbildung 12: Spielpartie *Schachmatt in zwei Zügen* mit eingezeichneten Zugmöglichkeiten von Weiß.

Bei Spielproblemen werden alle legitimen Zugfolgen ausprobiert und der individuelle Nutzen der Züge und Zugfolgen für die Spieler bewertet. Weil die Spielerinteressen in der Spielumgebung konfligieren, werden solche Spielprobleme auch als *adversariale* Suchprobleme bezeichnet.[158] Der Konflikt der Spieler manifestiert sich im Wettbewerb um die Auszahlung des Spieles: In jeder Runde der Spielpartie intendiert der aktuelle Spieler, seinen eigenen Nutzen zu maximieren, folglich die höchste Auszahlung am Ende der Partie zu erhalten. Im Gegenzug dazu beabsichtigt die Gegenspielerin, die eigene Auszahlung zu maximieren, indem sie die Auszahlung des Gegenspielers minimiert. Jeder Spieler A kann somit annehmen, dass seine Gegenspielerin B immer den für Spieler A ungünstigsten Zug ausführen wird – den Zug folglich, der die Auszahlung von Spieler A minimiert. Aufgrund dieser Annahme über die Strategiewahl der Spieler wird das vorliegende Spielproblem als *Minimax*-Problem bezeichnet.

[158] Vgl. Russell und Norvig 2012, S. 206.

Um individuell günstige, gar optimale Züge der Spieler zu identifizieren, wird die Spielpartie ausgehend von ihrer Initialkonfiguration, hin zu den Enden der Spielpartie in einem Spielbaum dargestellt. Auf welche Art, auf welche Weise der Spielbaum vom Wurzelknoten bis zu den Auszahlungsknoten durchquert wird, ist durch verschiedene Traversierungsarten bestimmt, von denen im Folgenden die Breiten- und die Tiefensuche vorgestellt werden.

4.1.1.1 Breiten- und Tiefensuche

Im Unterschied zur Breitensuche, bei der zunächst die Knoten einer Ebene nacheinander betrachtet werden, bevor die nachfolgenden Knoten der nächsten Ebene des Baumes traversiert werden, besucht die Tiefensuche – auch als *Depth First Search* bezeichnet – nacheinander alle Kindknoten des Wurzelknotens. Anschließend geht sie wieder zurück (*Backtracking*), um mit noch nicht besuchten Teilbäumen analog zu verfahren. So wird die Tiefensuche die Knoten des in der folgenden Abbildung dargestellten Binärbaumes[159] in der Reihenfolge A → B → D → E → C → F → G → H → I besuchen, und so werden die Knoten bei der Breitensuche in ihrer alphabetischen Reihenfolge A → B → C → D → E → F → G → H → I traversiert, um die gesuchte Information aufzufinden.

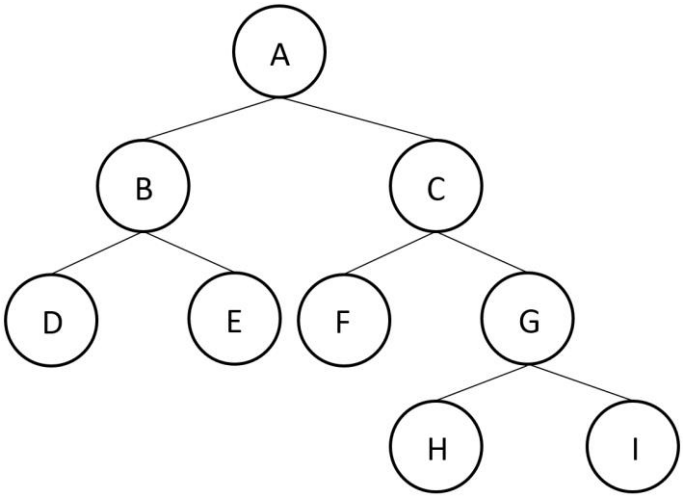


Abbildung 13: Binärbaum zur Illustration der Breiten- und Tiefensuche.

Liefert die Breitensuche die optimale Lösung, weil sie, einheitliche Kosten für den Besuch aller Knoten des Baumes vorausgesetzt, in ihrem Verlauf die Lösung mit

[159] Ein Binärbaum ist ein Baum, bei dem jeder Knoten des Baumes maximal über zwei Nachfolge- oder Kindknoten verfügt.

den geringsten Pfadkosten[160] ausfindig macht, so hat sie zum großen Nachteil, dass ihre Speicherkomplexität je nach Anforderungsszenario enorm hoch ist, weil alle Knoten einer Ebene des Suchbaumes expandiert im Arbeitsspeicher vorgehalten werden müssen und der Speicherplatzbedarf folglich exponentiell mit der Suchtiefe ansteigt [161] – für nichttriviale Spiele wie Schach eine völlig unpraktikable Lösung. Bei der Tiefensuche hingegen kann ein Knoten aus dem Arbeits- und Zwischenspeicher entfernt werden, wenn alle seine Folgeknoten expandiert und betrachtet wurden; der Speicherplatzbedarf steigt mit der Suchtiefe folglich nicht exponentiell, wie bei der Breitensuche, sondern linear. Im Gegensatz zur Breitensuche jedoch liefert die Tiefensuche mitunter nicht die optimale Lösung: Findet sich im oben wiedergegebenen Binärbaum sowohl an Knoten D als auch an Knoten C eine Lösung, so wird die Tiefensuche als ihr Ergebnis nicht den Knoten C zurückliefern, der die geringsten Pfadkosten aufweist, sondern den zuerst gefundenen Knoten D.

Wie eingangs mit dem Beispiel Schaeffers angemerkt, generieren Spiele wie Dame oder Schach Spielbäume äußerst hoher Komplexität. Die Suche nach optimalen Zügen gestaltet sich hierbei in der Regel äußerst zeitaufwändig, so dass es zumeist notwendig ist, die Tiefensuche auf eine maximale Suchtiefe zu beschränken. Liefert die Tiefensuche in einem Spielbaum mit n Knoten im besten Falle nach einer Suchoperation und im schlimmsten Falle nach n Suchvorgängen garantiert eine Lösung, so ist es der tiefenbeschränkten Suche zuweilen nicht möglich, eine Lösung für das gestellte Problem aufzufinden, da die Suche mitunter nicht zu relevanten Knoten vordringt.

4.1.1.2 Iterativ vertiefende Tiefensuche

Als Mittel der Wahl zur Suche in großen Suchräumen und bei Unbekanntheit der Lösungstiefe bietet sich die iterativ vertiefende Tiefensuche an, die Vorteile der Tiefen- und Breitensuche zu kombinieren.[162] So vereint die iterativ vertiefende

[160] Die Güte individueller Entscheidungen ist bestimmt durch die Pfadkostenfunktion, die einem Pfad im Zustandsraum – d.h. einer Aktionsfolge vom Ausgangs- zum Zielzustand – einen numerischen Wert zuweist. Eine optimale Lösung lässt sich bestimmen als eine Problemlösung, welche die geringsten Pfadkosten verursacht.
[161] Vgl. Russell und Norvig 2012, S. 118 f.
[162] Vgl. Russell und Norvig 2012, S. 125.

Suche den geringen Speicherplatzverbrauch der Tiefensuche mit dem Vorteil der Breitensuche, die optimale Lösung für das gestellte Problem zu finden. Die Verfahrensweise der iterativen Suche lässt sich leicht nachvollziehen: Schrittweise wird die Begrenzung der Suchtiefe erhöht, bis die Lösung gefunden oder ein Abbruchkriterium der Suche erfüllt ist (z.B. eine vordefinierte maximale Laufzeit des Verfahrens). Das folgende Beispiel veranschaulicht das Verfahren der iterativ vertiefenden Suche. Als Grundlage der Erläuterung dient der folgende Binärbaum mit acht Knoten:

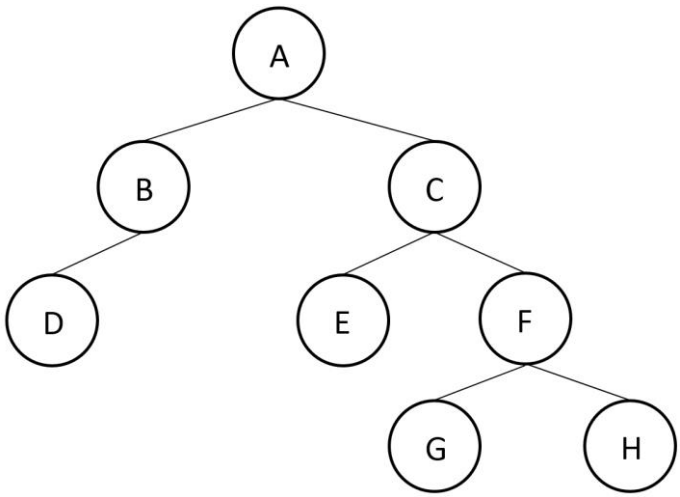


Abbildung 14: Binärbaum zur Verdeutlichung der iterativ vertiefenden Tiefensuche.

In der ersten Iteration der iterativ vertiefenden Suche wird zunächst der Wurzelknoten A betrachtet. Befindet sich am Knoten A die gesuchte Lösung, so wird der Suchvorgang erfolgreich abgeschlossen und der Wert an A zurückgeliefert. Findet sich an Knoten A die gesuchte Lösung nicht, so werden in der folgenden Iteration des Verfahrens die Knoten in der Reihenfolge A → B → C besucht, bis die Lösung gefunden wurde. In der dritten Iteration werden die Knoten des Baumes in der Reihenfolge A → B → D → C → E → F besucht; die vierte und finale Iteration schließlich traversiert die Knoten in der Reihenfolge A → B → D → C → E → F → G → H. Das Verfahren mag ineffizient anmuten, weil mit jeder Iteration bereits zuvor generierte Knoten des Baumes erneut erzeugt werden müssen, jedoch ist der Aufwand hierfür überschaubar, weil sich meisten Knoten

auf der untersten Ebene des Suchbaumes finden, so dass es unerheblich ist, wenn die oberen Ebenen des Baumes mehrfach generiert und betrachtet werden.[163]

Suchstrategien wie die tiefenbeschränkte Tiefensuche oder die iterativ vertiefende Suche finden ihre praktische Anwendung bei dem Minimax-Algorithmus und seinen Verfeinerungen. Wie sich die Suche nach optimalen Zügen gestaltet, das sei in den nachfolgenden Kapiteln zunächst am Beispiel des Tic Tac Toe Spieles betrachtet.

4.1.2 Minimax-Algorithmus

In seiner Grundvariante wird Tic Tac Toe von zwei Spielern auf einem zweidimensionalen Spielbrett mit 3×3 Spielfeldern gespielt. Das Spielbrett ist in seinem Ausgangszustand leer: Auf keinem der neun Spielfelder findet sich ein Spielsymbol einer Spielerin oder eines Spielers. In abwechselnder Reihenfolge des rundenbasierten Spieles legen die Spieler ihre Symbole auf den freien Feldern des Spielbrettes ab. Spielerin 1 verwendet das Kreuzsymbol X, Spieler 2 das Kreissymbol O zur Kennzeichnung der eigenen Spielzüge. Im Laufe der Partie dürfen die Spielfelder nur von einem Spielsymbol eines Spielers belegt werden. Das Spielziel ist erreicht, wenn ein Spieler drei seiner Spielsymbole in einer horizontalen, vertikalen oder diagonalen Folge auf dem Spielbrett abgelegt hat. Der im Folgenden wiedergegebene informell dargestellte vereinfachte Spielbaum visualisiert eine Spielpartie mit den Spielern *MAX* und *MIN*. Jeder Knoten des Baumes stellt einen Zustand des Spieles dar, im Folgenden auch als Spielkonfiguration bezeichnet.

[163] Vgl. Russell und Norvig 2012, S. 124.

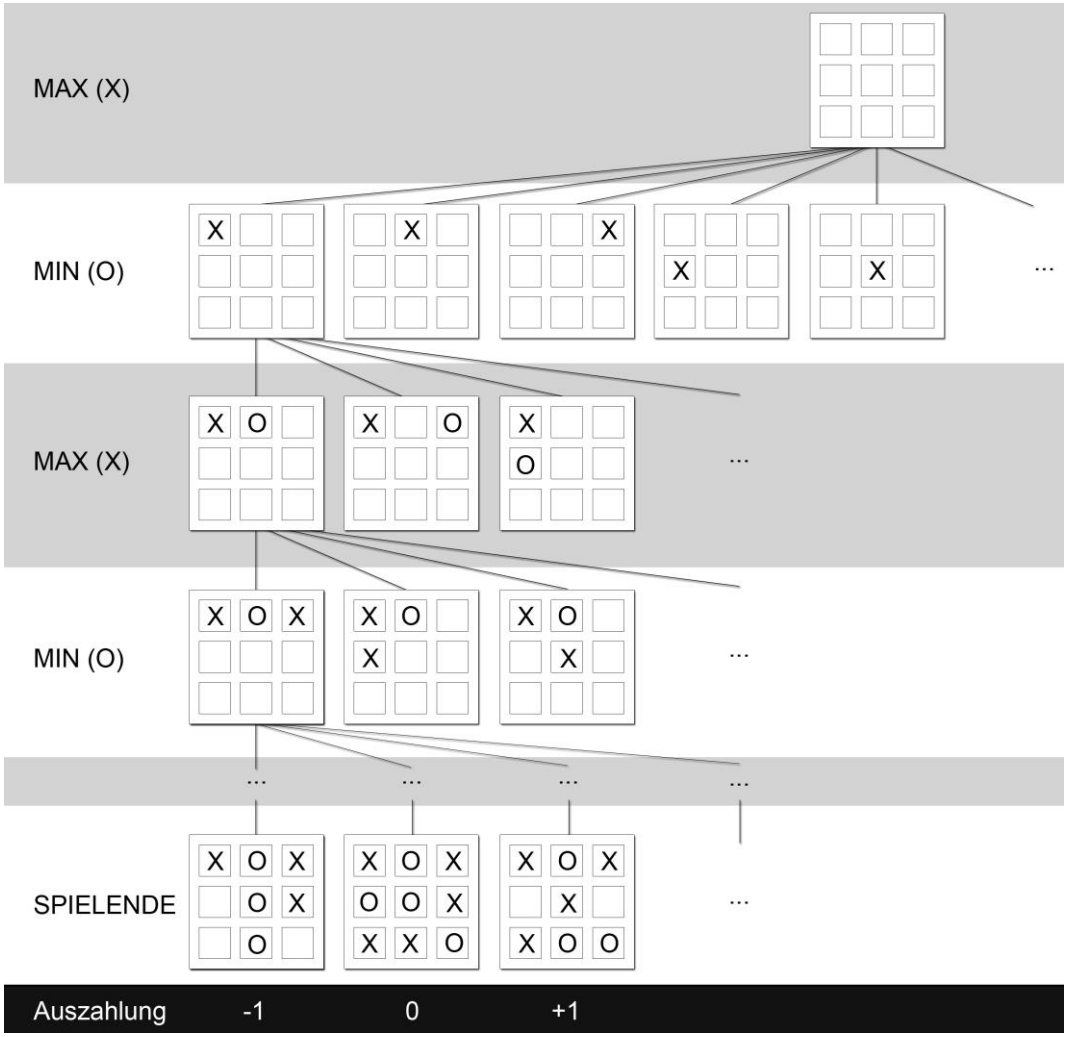


Abbildung 15: Partieller Spielbaum des Tic Tac Toe Spieles, wiedergegeben nach Russell und Norvig 2012, S. 208.

Das Tic Tac Toe Spiel, wie es sich mit dem obigen Spielbaum veranschaulicht, ist ein Nullsummenspiel: Gewinnt Spielerin 1 – in der Grafik als *MAX* bezeichnet –, so verliert Spieler 2 (*MIN*). Weil in Nullsummenspielen des einen Freud‘, was des anderen Leid ist, besteht die dominante Strategie von Spieler 2 darin, die Auszahlung seiner Gegenspielerin 1 zu minimieren, um die eigene Auszahlung höchstmöglich zu gestalten. Aus diesem Blickwinkel wird Spieler 2 folglich als *minimierender* Spieler und Spielerin 1 als *maximierende* Spielerin bezeichnet. So intendiert Spielerin 1 im oben wiedergegebenen Spiel, eine Folge von Entscheidungen zu treffen, die sie zu einem Spielende mit einer Auszahlung von +1 bringt und so bemüht sich Spieler 2, die Auszahlung seiner Gegenspielerin zu minimieren, das Spiel folglich mit einer Auszahlung von -1 abzuschließen. Ein Unentschieden führt zu einer Auszahlung von 0.

Optimale Strategien für Spielerinnen und Spieler lassen sich zu jedem Zeitpunkt des Konkurrenzspieles über das Minimax-Verfahren bestimmen. So traversiert der rekursiv verfahrende Minimax-Algorithmus jeden Knoten des Spielbaumes und ermittelt für jeden Knoten den Minimax-Wert, um eine Aussage darüber zu treffen, welcher Spielzug zu einem optimalen Spielergebnis führt. Um die Übersichtlichkeit in der Erläuterung des Minimax-Verfahrens zu wahren, sei der folgende reduzierte Spielbaum besprochen:[164]

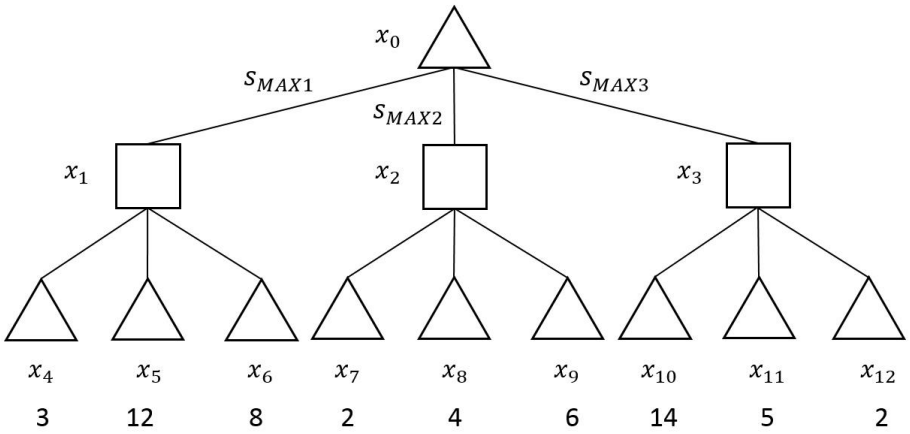


Abbildung 16: Spielbaum mit zwei Entscheidungsschichten. Die als Dreieck visualisierten Knoten bezeichnen Knoten, an denen der maximierende Spieler MAX am Zuge ist; an rechteckigen Knoten ist der Spieler MIN am Zuge. Die an den Blättern x_4 bis x_{12} notierten ganzzahligen Werte repräsentieren die Auszahlungen am Ende des Spieles.

Welche Strategie sollte die das Spiel beginnende Spielerin *MAX* im oben wiedergegebenen Spiel mit zwei Spielrunden verfolgen, um optimal zu agieren und ihre Auszahlung zu maximieren? Ist es für *MAX* sinnvoll, sich zu Beginn des Spieles für die Strategie s_{MAX2} zu entscheiden oder ist es schlauer, die Strategie s_{MAX3} zu verfolgen, da sich hier an Endknoten x_{10} mit einer Auszahlung von 14 die höchste Auszahlung des gesamten Spieles findet?

Die Frage nach der besten Strategie für Spielerin *MAX* ist schnell beantwortet: Weil ihr – rational agierender – Gegenspieler *MIN* bestrebt ist, die Auszahlung von *MAX* zu minimieren, besteht die beste Strategie für die Spielerin *MAX* darin, die Strategie s_{MAX1} zu verfolgen, um sich das Maximum der Minimalwerte zu sichern. Würde sie sich für die Strategie s_{MAX2} oder s_{MAX3} entscheiden, so betrüge die Auszahlung 2 – eins weniger als das Maximum der Minimalwerte bei Strategie

[164] Der Spielbaum orientiert sich am Beispiel, wie es sich in Russell und Norvig 2012, S. 209 findet.

s_{MAX1}. Rationales Agieren beider Spieler vorausgesetzt, kann die Spielerin *MAX* bereits zu Beginn des Spieles mit einem Auszahlungswert von 3 rechnen – das Spiel ist gelöst.

Das Minimax-Verfahren, wie es sich mit der folgenden Darstellung erläutert findet, traversiert den Spielbaum mit dem Verfahren der Tiefensuche bis an seine Blätter, berechnet mit der Auszahlungsfunktion die Wertigkeit des Spielergebnisses und reicht die ganzzahligen Werte an den jeweils darüber liegenden Elternknoten weiter. Ist der Elternknoten ein *MAX*-Knoten, so wird das Maximum der an den Kindknoten identifizierten Auszahlungswerte weitergegeben; ist der Elternknoten ein *MIN*-Knoten, wird das Minimum der Auszahlungswerte an den Elternknoten übergeben. Auf Basis dieses Verfahrens ist es sehr leicht möglich, optimale Strategien zu identifizieren, vorausgesetzt, jedem Spieler sind während des Spielverlaufes alle vorausgehenden Züge seiner Mitspieler bekannt.

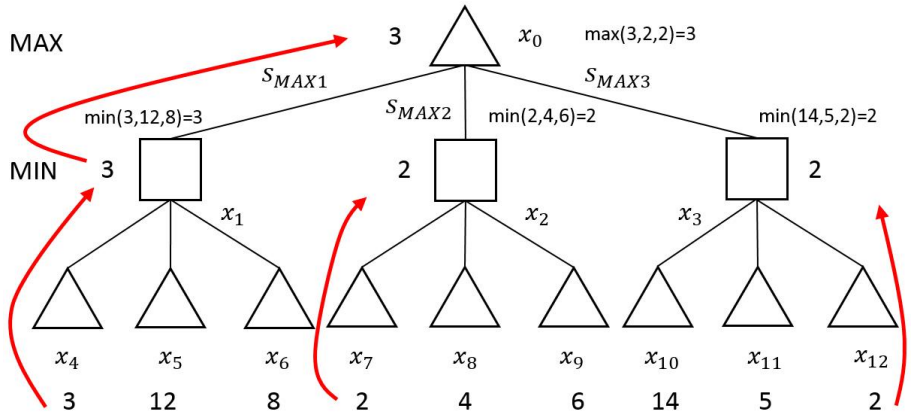


Abbildung 17: Visualisierung des Minimax-Verfahrens.

Zwar lassen sich alle 255.168 Spielverläufe von Tic Tac Toe, wie sie Jesper Juul in seinem Beitrag *255.168 ways of playing Tic Tac Toe*[165] ausfindig macht, an dieser Stelle nicht übersichtlich in einem Spielbaum visualisieren, für leistungsfähige Rechner und Computer ist die Identifizierung siegbringender Züge vom Anfang bis zum Ende der Spielpartie für beide Spieler unter Anwendung des Minimax-Verfahrens jedoch ein allzu leichtes – und für Spiele überblickbarer Spielbaumkomplexität in sekundenbruchteilen geleistet. Die folgende Grafik

[165] Vgl. Juul: 255,168 ways of playing Tic Tac Toe 2003. Online verfügbar unter http://www.jesperjuul.net/ludologist/255168-ways-of-playing-tic-tac-toe, zuletzt geprüft am 24.09.2013.

visualisiert das Minimax-Verfahren anhand einer Partie des Tic Tac Toe Spieles, ausgehend von einem bestimmten Spielzustand, der den Wurzelknoten des Spielbaumes darstellt.

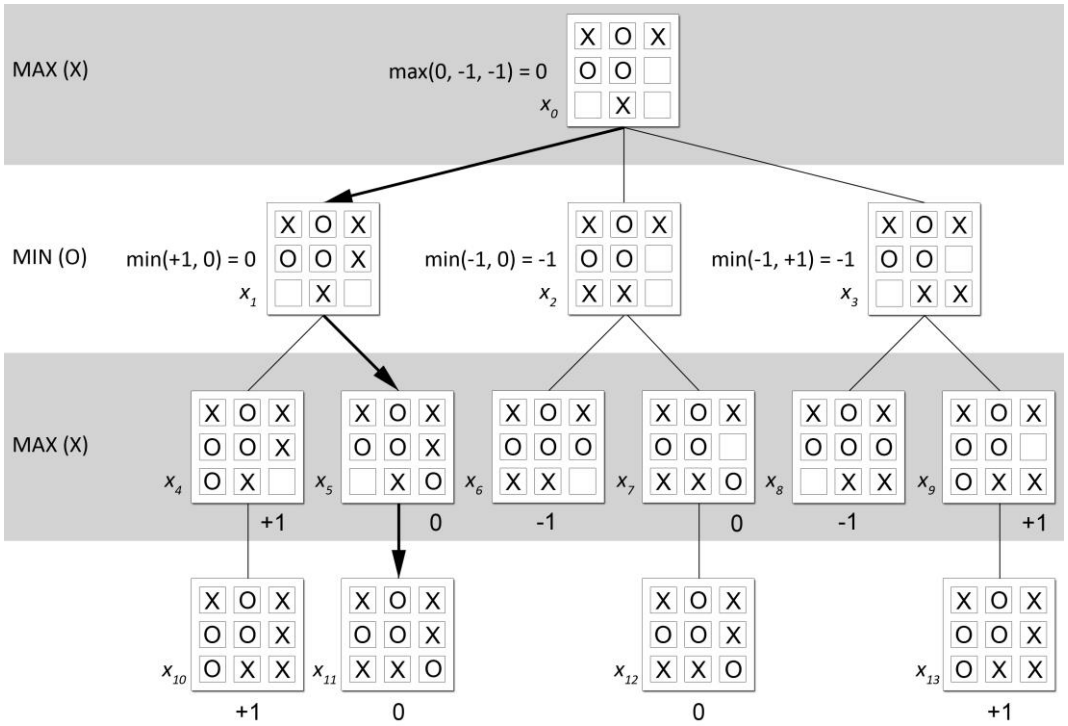


Abbildung 18: Lösung einer Partie des Tic Tac Toe Spieles anhand des Minimax-Algorithmus‘.

An Wurzelknoten x_0 der dargestellten Spielpartie verfügt die maximierende Spielerin über drei Möglichkeiten, ihr Spielsymbol X auf dem Spielfeld abzulegen. Jede Wahl der Spielerin generiert weitere Pfade im Spielbaum, die sich nach Ermittlung der Auszahlung an den Blattknoten bewerten lassen. Verhalten sich beide Spieler rational, d.h. sind beide Spieler zu jedem Zeitpunkt des Spieles bestrebt, ihre individuelle Auszahlung zu maximieren, so führt das Spiel zu einem Unentschieden am Blattknoten x_{11}. Der spieltheoretische Wert des Spieles beträgt folglich 0.

4.1.3 Negamax-Algorithmus

Der Minimax-Algorithmus bewertet die Zugmöglichkeiten aus der Sicht einer Spielerin oder eines Spielers und muss über Mechanismen verfügen, um zu wissen, welcher Spieler am Zuge ist und folglich, ob die Zugbewertung maximiert oder minimiert werden muss. Weil in Zweispieler-Nullsummenspielen des einen Freud‘ ist, was des anderen Leid, die Summe der Minimax-Werte für den Spieler und seine Gegenspielerin immer 0 ist, lässt sich der Minimax-Algorithmus vereinfachen, indem die Bewertungen der Evaluierungsfunktion bei jedem Hochreichen an die Elternknoten invertiert, d.h. mit -1 multipliziert werden und das Maximum ausgewählt wird. Somit muss nicht mehr zwischen maximierendem und minimierendem Spieler unterschieden werden.[166] Dieses Verfahren trägt den Namen *Negamax* und sei mit dem folgenden Pseudocode wiedergegeben.[167]

Algorithmus 4.1 *Der Negamax-Algorithmus im Pseudocode.*

- Parameter:
  - *gameState*: Zustand der Spielwelt.
  - *depth*: aktuelle Verarbeitungstiefe.
- Rückgabe: Bester identifizierter Zug und seine Bewertung.

```
function Negamax(gameState, depth)
    if (gameState is terminal or depth equals 0) then
      return { finalScore: evaluation(gameState),
               finalMove: none }
    else
      bestScore  ← −∞
      bestMove ← none
      for each child node gameState’ of the current game state
          score, move ← Negamax(gameState’, depth--)
          score ← −score
          if (score > bestScore)
```

[166] Vgl. Millington und Funge 2009, S. 678 ff. und Mańdziuk 2010, S. 17f.
[167] Der Pseudocode des Minimax-Algorithmus‘ folgt den Ausführungen, wie sie sich in Millington und Funge 2009, S. 679f. sowie Mańdziuk 2010, S. 18 finden.

```
                bestScore  ← min(score, bestScore)
                bestMove ← move
        return { finalScore: bestScore, finalMove: bestMove }
```

Der Leistungsfähigkeit des Negamax-Algorithmus‘ essentiell ist zum einen die Evaluierungsfunktion `evaluation`, die eine Aussage darüber trifft, wie gut ein terminaler oder nichtterminaler Zustand des Spieles für den betrachteten Spieler ist. Zum anderen wollen die Datenstrukturen und Schnittstellen bedacht umgesetzt sein, mit denen sich der Algorithmus durch die Spielwelt bewegt: die interne Repräsentation des Spielbrettes, die Möglichkeit, virtuelle Züge – also Züge, die nicht in der tatsächlichen Spielwelt ausgeführt werden, sondern im (virtuellen) Spielbaum neue Kindknoten verursachen – zu vollziehen, um sie anschließend zu bewerten, der Zugriff auf den gegenwärtigen Spielzustand sowie Funktionalität, um alle möglichen Spielzüge der Spielerinnen und Spieler des Spieles zu identifizieren.

4.1.4 Alpha-Beta Kürzung

Das wesentliche Problem an dem oben wiedergegebenen Negamax-Algorithmus besteht darin, dass alle Knoten des Spielbaumes bis zu einer vordefinierten Suchtiefe oder bis zum Ablauf einer vorgegebenen maximalen Suchdauer generiert und bewertet werden, die Anzahl der zu bewertenden Züge folglich exponentiell zur Anzahl der Züge steigt.[168] Bei einer maximalen Suchtiefe d mit einer Anzahl von m möglichen Zügen zu jedem Spielzustand gestaltet sich die Zeitkomplexität des Negamax-Algorithmus als $\mathrm{O}(m^d)$; für einen Algorithmus, der alle Zugmöglichkeiten nacheinander expandiert, ist die Speicherkomplexität mit $\mathrm{O}(d)$ beschrieben.[169] Um die Laufzeit des Algorithmus‘ deutlich zu verkürzen, folglich irrelevante Teile des Spielbaumes von der Suche auszuschließen, kommt das Verfahren der Alpha-Beta-Kürzung (auch als *Alpha-Beta Beschneidung* oder *Alpha-Beta-Pruning* bezeichnet) zum Einsatz, das es ermöglicht, den korrekten

[168] Vgl. Russell und Norvig 2012, S. 212.
[169] Vgl. Russell und Norvig 2012, S. 210f.

Minimax-Wert zu bestimmen, ohne tatsächlich jeden Knoten im Spielbaum zu betrachten.

Seinen Namen erhält das in den späten 1950er Jahren von Newell et al. vorgestellte und 1975 ausführlich analysierte Verfahren [170] von den zwei Schwellenwerten Alpha und Beta, die im Laufe des Suchverfahrens aktualisiert werden und über die sich an einem Knoten des Spielbaumes eine Annahme darüber treffen lässt, ob Kindknoten von der Tiefensuche ausgeschlossen werden können. Hierbei bezeichnet

- α die beste, bislang an jedem Knoten entlang des Pfades ausfindig gemachte Zugfolge für den maximierenden Spieler und
- β die beste, bislang an jedem Knoten entlang des Pfades ausfindig gemachte Zugfolge für den minimierenden Spieler.[171]

Die Alpha- und Beta-Werte formen den Rahmen möglicher zu erreichender Auszahlungen: Für den maximierenden Spieler ist es auf der einen Seite unsinnig, Züge mit einer Auszahlung kleiner Alpha zu wählen, auf der anderen Seite wird der rational agierende Gegenspieler bestrebt sein, dem maximierenden Spieler Zugmöglichkeiten zu verbauen, die eine Auszahlung größer Beta zu Folge haben. [172] Von der Tiefensuche ausgeschlossen werden folglich Teile des Suchbaumes, an deren Knoten sich $\alpha \geq \beta$ findet.

Zur Erläuterung des Verfahrens der Alpha-Beta Kürzung – das auch als „elegantestes aller Suchverfahren der KI“[173] referiert wird – sei der im Folgenden wiedergegebene Spielbaum betrachtet und das Verfahren ausführlich Schritt für Schritt besprochen:[174]

[170] Vgl. Newell et al. 1958, Knuth und Moore 1975 sowie den kurzen Überblick über die Forschungsgeschichte rund um die Alpha-Beta-Kürzung in Mańdziuk 2010, S. 19.
[171] Vgl. Russell und Norvig 2012, S. 214.
[172] Vgl. Millington und Funge 2009, S. 681 ff.
[173] Vgl. u.a. den Ausspruch „One of the most elegant of all AI search algorithms is alpha-beta pruning“ (Korf 2010, S. 13).
[174] Vgl. auch die sehr anschauliche schrittweise Erläuterung des Verfahrens unter http://www.youtube.com/watch?v=xBXHtz4Gbdo (zuletzt aufgerufen am 28.04.2014).

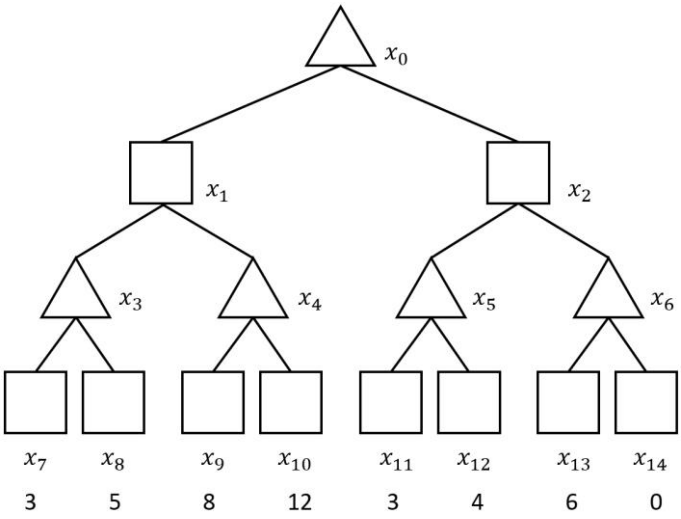


Abbildung 19: Spielbaum zur Verdeutlichung der Alpha-Beta Kürzung.

Die Alpha-Beta Kürzung startet ihre Suche nach der optimalen Strategie für die maximierende Spielerin MAX an Wurzelknoten x_0 mit den Werten $\alpha = -\infty$ und $\beta = +\infty$, weil zu Beginn des Verfahrens noch keine Knoten des Spielbaumes betrachtet wurden. Anschließend wird im Rahmen der Tiefensuche zunächst der Kindknoten x_1 des Wurzelknotens betrachtet und ihm die Werte $\alpha = -\infty$ und $\beta = +\infty$ weitergegeben, weil noch keine Auszahlungsknoten verarbeitet wurden. Analog dazu wird mit Knoten x_3 und x_7 verfahren, so dass sich der Spielbaum wie folgend um die Alpha- und Beta-Werte ergänzt findet:

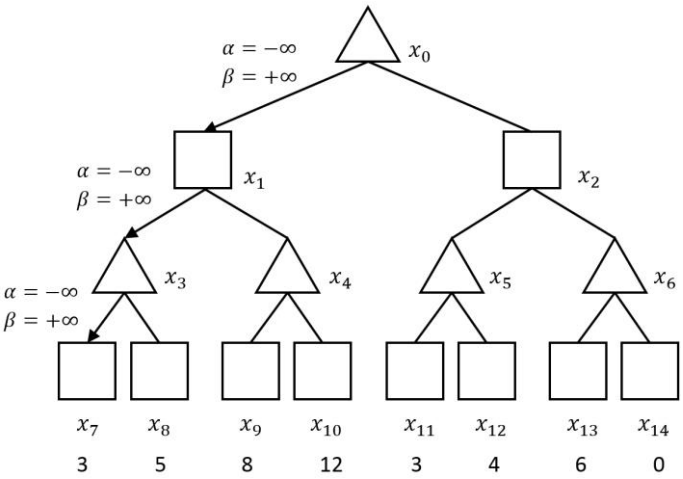


Abbildung 20: Erster Schritt des Verfahrens der Alpha-Beta Kürzung.

Im nächsten Schritt wird die Auszahlung am Blattknoten x_7 bestimmt. Der Auszahlungswert 3 wird zurückgegeben an den maximierenden Knoten x_3 und als Wert des Knotens gesetzt, weil er das bisherige Maximum der Auszahlungen

darstellt. Anschließend wird der Wert verglichen mit dem Beta-Wert. Wäre der Auszahlungswert des soeben betrachteten Blattknotens größer als der Beta-Wert, so müsste der rechte Blattknoten x_8 nicht mehr betrachtet werden. Da dies jedoch nicht der Fall ist, muss der zweite Blattknoten betrachtet werden.

Weil mit der Auszahlung an Blattknoten x_7 ein (zunächst) optimaler Zug gefunden wurde, wird Alpha auf den Wert 3 gesetzt und gemeinsam mit dem Beta-Wert $+\infty$ (der besten Zugoption für den minimierenden Spieler) an den Blattknoten x_8 gereicht. Der Knoten x_8 birgt eine Auszahlung von 5, ist somit höher als das bislang identifizierte Maximum von 3 und wird folglich dem Knoten x_3 zugewiesen. Nachdem nun der linke Teilbereich des Knotens x_1 vollständig generiert und betrachtet wurde, wird dem minimierenden Knoten x_1 der bislang kleinste ausfindig gemachte Auszahlungswert 5 zugewiesen. In der folgenden Darstellung findet sich der Zwischenstand des Verfahrens dokumentiert.

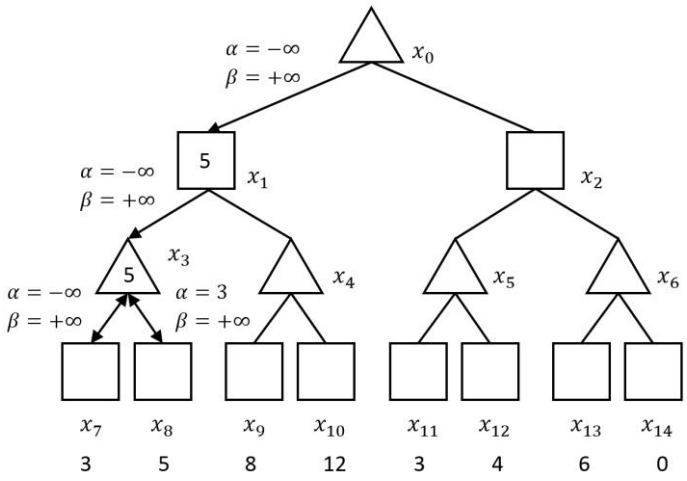


Abbildung 21: Zweiter Schritt des Verfahrens der Alpha-Beta Kürzung.

Im nächsten Schritt wird am minimierenden Knoten x_1 geprüft, ob es sich lohnt, den rechten Zweig und seine Unterknoten zu betrachten. Um zu entscheiden, ob der Kindknoten x_4 aus der Betrachtung ausgeschlossen werden kann, wird der aktuelle Wert 5 an Knoten x_1 verglichen mit der besten Zugfolge, die für die maximierende Spielerin auf höheren Ebenen, d.h. vom aktuell betrachteten Knoten bis zum Wurzelknoten des Spielbaumes identifiziert werden konnte: α. Weil solch eine Zugfolge noch nicht ausfindig gemacht werden konnte ($\alpha = -\infty$ an Knoten x_1), kann der maximierende Knoten x_4 nicht von der Betrachtung ausgeschlossen werden. Weitergereicht an Knoten x_4 wird mit α die beste bislang

gefunde Zugfolge für die maximierende Spielerin. Da noch keine solche Zugfolge ausgehend vom Wurzelknoten bis zum aktuell betrachteten Knoten existiert, wird α der Wert $-\infty$ zugewiesen. β wird die zuvor ermittelte minimale Auszahlung von 5 zugewiesen und weitergereicht. Anschließend wird der Kind- und Blattknoten x_9 betrachtet: x_9 führt zu einer Auszahlung von 8. Weil sich am maximierenden Knoten x_4 noch kein Auszahlungsmaximum findet, wird der Wert von 8 an Knoten x_4 notiert.

In der sich anschließenden Überlegung findet die Kürzung des Suchraumes ihre erste Anwendung: An Knoten x_4 bedenkt die maximierende Spielerin ihr weiteres Vorgehen und vergleicht ihre aktuell ermittelte maximale Auszahlung von 8 mit der besten bislang ermittelten Auszahlung für den minimierenden Spieler $\beta = 5$. An dieser Stelle ist der maximierenden Spielerin gewiss, dass ihr maximierender Gegenspieler bestrebt sein wird, die maximale Auszahlung auf 5 zu beschränken. Weil der ermittelte Alpha-Wert 8 größer ist als der Beta-Wert 5, können alle weiteren Kindknoten des Knotens x_4 von der Suche ausgeschlossen werden – x_{10} muss folglich nicht betrachtet und expandiert werden, wie in der folgenden Darstellung visualisiert.

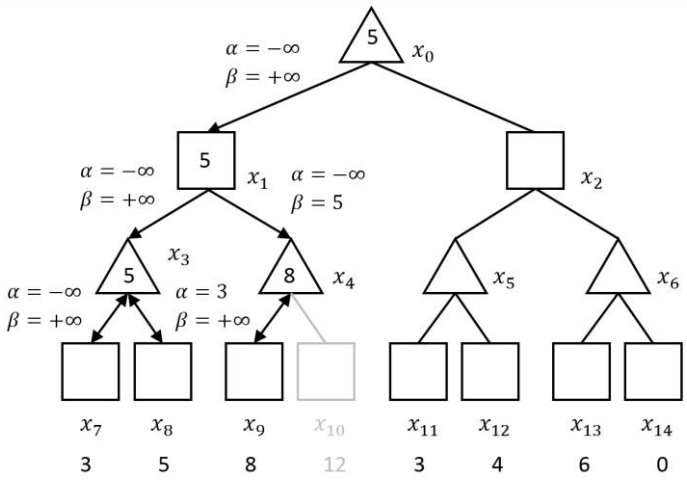


Abbildung 22: Alpha-Beta Kürzung. Grau hinterlegte Knoten werden in der Suche nicht berücksichtigt.

Der linke Teil des Spielbaumes ist nun vollständig betrachtet, so dass am maximierenden Wurzelknoten erneut die Frage gestellt wird, ob der rechte Teilbaum analysiert werden muss. Weil am Wurzelknoten $\alpha = -\infty$ und $\beta = +\infty$ gilt, folglich keine Entscheidungsgrundlage existiert, muss der rechte Teilbaum

betrachtet werden. So wird an Knoten x_2 die beste bislang ermittelte Zugfolge bzw. Auszahlung für die maximierende Spielerin und den maximierenden Spieler weitergereicht. Die beste Auszahlung für die maximierende Spielerin beträgt 5, für den minimierenden Spieler findet sich keine solche Zugfolge, so dass $\alpha = 5$ und $\beta = +\infty$. Anschließend werden die beiden Blattknoten betrachtet. Weil keine Kürzung möglich ist, erhält der maximierende Knoten x_5 mit einem Wert von 4 das Maximum der Auszahlungen zugewiesen, die sich an den Blattknoten x_{11} und x_{12} finden.

Anschließend findet die Alpha-Beta Kürzung ein letztes Mal ihre Anwendung: Am minimierenden Knoten x_2 stellt sich die Frage, ob der rechte Teilbaum untersucht werden muss, oder von der Betrachtung ausgeschlossen werden kann. Weil die soeben ermittelte Auszahlung von 4 kleiner ist als die beste mit $\alpha = 5$ bislang ermittelte Auszahlung bzw. Zugfolge der maximierenden Spielerin, lassen sich weitere Kindknoten von der Untersuchung ausschließen, da die rational agierende maximierende Spielerin niemals einen Zug von x_0 nach x_2 vollziehen würde. Weil alle relevanten Knoten betrachtet wurden, beträgt der Wert des Spieles 5. Die folgende Darstellung visualisiert den finalen Status des Alpha-Beta Verfahrens.

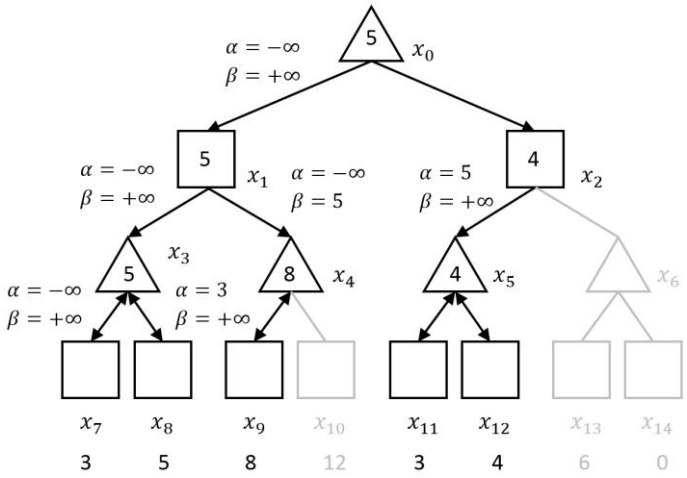


Abbildung 23: Vollständig analysierter Spielbaum im Rahmen des Verfahrens der Alpha-Beta Kürzung. Von der Betrachtung ausgeschlossene Knoten sind grau hinterlegt.

Um nicht bei jedem Zug den Alpha- und Beta-Wert zu prüfen, vertauscht und invertiert der im Folgenden wiedergegebene *Alpha-Beta Negamax* Algorithmus die Werte und prüft nur gegen den Beta-Wert.[175]

Algorithmus 4.2 *Der Alpha-Beta Negamax Algorithmus im Pseudocode.*

- Parameter:
  - *gameState*: Zustand der Spielwelt.
  - *depth*: aktuelle Verarbeitungstiefe.
  - α: Wert der besten Wahl für MAX.
  - β: Wert der besten Wahl für MIN.
- Rückgabe: Bester identifizierter Zug und seine Bewertung.

```
function ABNegamax(gameState, depth, alpha, beta)
    if (gameState is terminal or depth equals 0) then
      return { finalScore: evaluation(gameState),
               finalMove: none }
    else
      bestScore  ← −∞
      bestMove ← none
      for each child node gameState' of the current game state
          score, move ← ABNegamax(gameState', depth--,
                                  −β, −max(α, bestScore))
          score ← −score
          if (score > bestScore) then
              bestScore  ← score
              bestMove ← move
              if (bestScore >= β) then
                  return { finalScore: bestScore, finalMove:
                           bestMove }
      return { finalScore: bestScore, finalMove: bestMove }
```

[175] Vgl. Millington und Funge 2009, S. 682–684 sowie Mańdziuk 2010, S. 21.

4.1.5 Berücksichtigung dediziertes Spielwissens – die Heuristische Evaluierungsfunktion

Lässt sich nicht der vollständige Spielbaum der Spielpartie ausgehend von der aktuellen Spielkonfiguration berechnen und expandieren – ob der Komplexität der betrachteten Spiele ein gängiges Faktum –, muss am aktuellen Knoten eine Aussage darüber getroffen werden, wie gut eine bestimmte Spielkonfiguration im Hinblick auf ein bestimmtes Spielziel fungiert. Die heuristische Evaluierungsfunktion, im Pseudocode der vergangenen Kapitel mit der Funktion `evaluation` repräsentiert, trifft solch eine quantitative Aussage und ist der Kernmechanismus artifizieller Computergegner in klassischen Brettspielen.

Wie Kroll definiert, ist mit dem Term *Heuristik* eine „auf ein bestimmtes Problem angepasste Lösungsstrategie [signifiziert], die problemspezifische Erfahrung ausnutzt, um die Suche so zu lenken, dass diese mit hoher Wahrscheinlichkeit zu einer akzeptablen Näherungslösung führt.“ [176] „Akzeptable“ und „Näherungslösung“ stellen hierbei die entscheidenden Definitionspartikel dar: Gesucht wird keine vollständige, keine optimale Lösung, weil sich eine solche nicht zeitnah auffinden ließe. Gesucht wird ein sinnvoller Vorschlag, ob ein Spielzug einem anderen gegenüber wohl zu einem besseren Spielergebnis führt oder ob es sinnvoll ist, eine völlig andere Zugwahl vorzunehmen, weil sie die Spielpartie zu einem individuell guten Ende zu führen scheint.

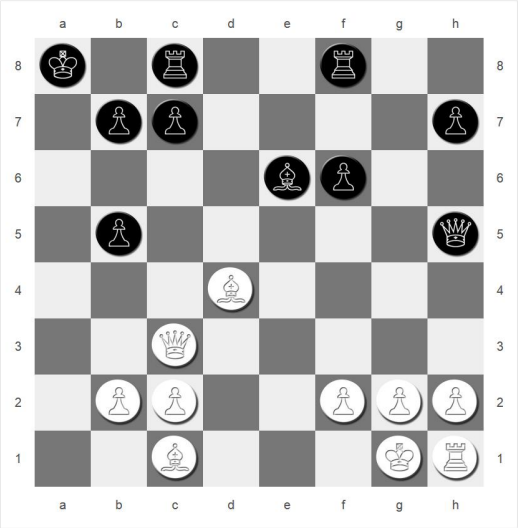
Die Effizienz der Evaluierungsfunktion ist abhängig von zahlreichen Faktoren, die von Spiel zu Spiel differieren, von Spielpartie zu Spielpartie analytisch geschöpft oder über Jahre hinfort als individuelle Erfahrungen der Spielerinnen und Spieler tradiert werden. So finden sich in der Schachliteratur unterschiedliche Materialwerte für die verschiedenen Spielfiguren und so fungieren weitere Faktoren und Merkmale wie die Anordnung und Position der Bauern, die Sicherheit des Königs oder die Summe der möglichen zur Verfügung stehenden Bewegungsmöglichkeiten aller Spielfiguren als Kriterien für die Evaluierungsfunktion.[177]

[176] Kroll 2013, S. 93.
[177] Vgl. Shannon 1950, S. 260.

Mit Russell und Norvig lässt sich die Evaluierungsfunktion formalisieren als gewichtete Linearfunktion:[178]

$$EVAL(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s) = \sum_{i=1}^{n} w_i f_i(s) \qquad (4.1)$$

Hierbei bezeichnet f_i das i-te Merkmal der Spielkonfiguration, z.B. die Anzahl der auf dem Spielbrett befindlichen Bauern, und w_i die entsprechende Gewichtung des einzelnen Merkmales. Mit ganzzahligen Materialwerten von 1 für einen Bauern, 3 für einen Läufer und einen Springer, 5 für einen Turm, 9 für eine Dame und 200 für einen König, wie sie Shannon vorschlägt,[179] lässt sich somit die Gewichtung der eingangs dargestellten Spielkonfiguration für die beiden Spieler berechnen, wie im Folgenden wiedergegeben.[180]



Anzahl der Spielfiguren	
Weiß	Schwarz
Bauer: 5	Bauer: 5
Läufer: 2	Läufer: 1
Springer: 0	Springer: 0
Turm: 1	Turm: 2
Dame: 1	Dame: 1
König: 1	König: 1

Tabelle 5: Übersicht über die Spielkonfiguration *Schachmatt in zwei Zügen*

Die Berechnung erfolgt über den folgenden Term, der den Materialfundus beider Spieler in Beziehung zueinander setzt, indem die Gewichtungen der Spielfiguren

[178] Wiedergegeben nach Russell und Norvig 2012, S. 218.

[179] Vgl. Shannon 1950, S. 260 sowie die umfangreiche Übersicht über Materialwerte der Spielfiguren im Schachspiel in: Chess Programming Wiki: Point Value. Online verfügbar unter http://chessprogramming.wikispaces.com/Point+Value, zuletzt geprüft am 30.09.2013.

[180] Das vorgestellte Beispiel ist ein reduziertes, ein triviales Beispiel. In der Praxis der Schachprogrammierung findet sich die Evaluierungsfunktion ergänzt und erweitert um zahlreiche weitere Merkmale wie die Anzahl blockierter oder freier Bauern oder die Gesamtzahl der legalen Züge aller Spielfiguren. Vgl. hierzu auch Chess Programming Wiki: Evaluation. Online verfügbar unter http://chessprogramming.wikispaces.com/Evaluation, zuletzt geprüft am 30.09.2013.

mit den Differenzen der Anzahl vorhandener Spielfiguren beider Spieler multipliziert werden:

$$EVAL(s) = 1 \times (5-5) + 3 \times (2-1) + 3 \times (0-0) + 5 \times (1-2) + 9 \times (1-1) + 200 \times (1-1) = 0 + 3 + 0 - 5 + 0 + 0 = -2$$

Mit -2 berechnet die Evaluierungsfunktion einen negativen Wert für Weiß. Folglich ist Weiß in der oben wiedergegebenen Spielkonfiguration materiell im Nachteil und Schwarz im Vorteil. Für Schwarz gestaltet sich die materielle Güte der Spielkonfiguration als invertiertes Ergebnis der Berechnung. Um die Situation adäquat analysieren und die nahende Bedrohung des schwarzen Königs identifizieren zu können, offenbart sich die vorliegende Evaluierungsfunktion jedoch schnell als allzu trivial – und offenbart sich zudem der spielspezifische Charakter der betrachteten Evaluierungsfunktion, wie Bewersdorff erläuternd ausführt, kann

> „eine Position im Mittelspiel des Schach[spiele]s […] unter Umständen durch das Verrücken eines einzelnen Bauern um nur ein Feld von einer Gewinn- in eine Verlustposition verwandelt werden. Verrückt man dagegen bei einer Backgammonposition einen Stein um ein Feld, wird dadurch der Wert der Position, bei dem es sich aufgrund des Zufallseinflusses um einen Erwartungswert handelt, relativ wenig geändert. Außerdem ist der Spielbaum des Backgammon[spieles] aufgrund des zusätzlichen Würfeleinflusses stärker verzweigt als der Spielbaum des Schach[spieles]. Backgammon ist damit ein Spiel, bei dem es eher lohnend erscheint, nach guten Schätzfunktionen für den Wert einer Position zu suchen, während beim Schach eher der Fokus auf einer effizienten Durchsuchung des Spielbaums liegen sollte.“[181]

„The most popular AI approach is again the careful hand-crafting process“[182], so fasst Mańdziuk die Arbeit an der spieldedizierten Evaluierungsfunktion zusammen und weist darauf hin, dass die Feinjustierung der Funktion in zwei Phasen stattfindet: In der ersten Phase werden essentielle Spielmerkmale identifiziert und mit groben Gewichtungen versehen; die zweite Phase besteht darin, jene grob

[181] Bewersdorff 2012, S. 195. Mit *AI* ist die *Artificial Intelligence* (Künstliche Intelligenz) abgekürzt.
[182] Mańdziuk 2010, S. 15.

dimensionierten Gewichtungen anzupassen, um die Leistungsfähigkeit der Funktion zu verbessern.[183] Im Unterschied zur manuellen Optimierungspraxis der KI gründet die Verfeinerung der Evaluationsfunktion in der CI nicht in menschlicher Expertise, sondern in dem Lernprozess künstlicher Neuronaler Netze, der Nachahmung von Prinzipien wie Mutation, Rekombination und Selektion der biologischen Evolution (Evolutionäre Algorithmen) oder durch Mechanismen, die es vermögen, aus (virtuellen) Erfahrungen zu lernen (Temporal Difference Learning). Auf jene Methoden und Algorithmen des Lernens sinnvoller Spielcharakteristika zur dynamischen Anpassung der Evaluierungsfunktion wird in Kapitel 0 noch ausführlich eingegangen. Zuvor jedoch werden mit den folgenden Kapiteln zunächst Optimierungen der Alpha-Beta Kürzung vorgestellt und werden mit Monte Carlo Verfahren zufallsbasierte Mechanismen zur Entscheidungsfindung in digitalen Brettspielen besprochen.

4.1.6 Optimierungen und Ergänzungen der Alpha-Beta Kürzung

Als Algorithmus zur Identifikation optimaler Züge in endlichen Zweispieler-Nullsummenspielen mit perfekter Information findet sich die zuvor referierte Alpha-Beta Kürzung ergänzt um Optimierungen, die intendieren, das Laufzeitverhalten des Verfahrens zu verbessern.[184] Die nachfolgenden Subkapitel treffen eine Auswahl aus der Vielzahl von Algorithmen zur Reduktion des Suchraumes und stellen mit der Nullzug- und Ruhesuche, mit Zugvorsortierung und Killerheuristik essentielle Bestandteile zeitgemäßer Computerschach-Engines vor – Verfahren, die sich auch auf andere Brettspiele übertragen lassen. Von der Erläuterung von Eröffnungs- und Endspieldatenbanken wird im Folgenden abgesehen, weil jene Datensammlungen Wissen repräsentieren, das einzig für die jeweiligen Spiele gültig und nicht auf andere Spieldomänen übertragbar ist.

[183] Vgl. Mańdziuk 2010, S. 15.

[184] *SCOUT* (Pearl 1980) und seine Weiterentwicklung *NegaScout* (Reinefeld 1983) sowie der von von Plaat et al. vorgestellte *MTD(f)* Algorithmus (Plaat et al. 1995), der das *TEST*-Verfahren weiterdenkt, wie es Pearl im Kontext seines *SCOUT*-Algorithmus vorschlägt, bieten elaborierte Optimierungen und Alternativen der Alpha-Beta Kürzung. Weil der Fokus der vorliegenden Arbeit auf interagierende Agenten gerichtet ist – und mit dem Alpha Beta Algorithmus ein leistungsfähiges Instrumentarium zur Verfügung steht, seien diese Verfahren an dieser Stelle nur angemerkt und nicht ausführlich betrachtet.

4.1.6.1 Zugvorsortierung

Wie effizient die Alpha-Beta Kürzung verfährt, das hängt besonders davon ab, in welcher Reihenfolge die Auszahlungen bestimmt bzw. die Blattknoten besucht werden. So ließe sich insgesamt nur ein Blattknoten von der Betrachtung ausschließen, wenn die Knoten x_5 und x_6 (und ihre assoziierten Kindknoten) miteinander vertauscht würden, wie nachfolgend dargestellt.

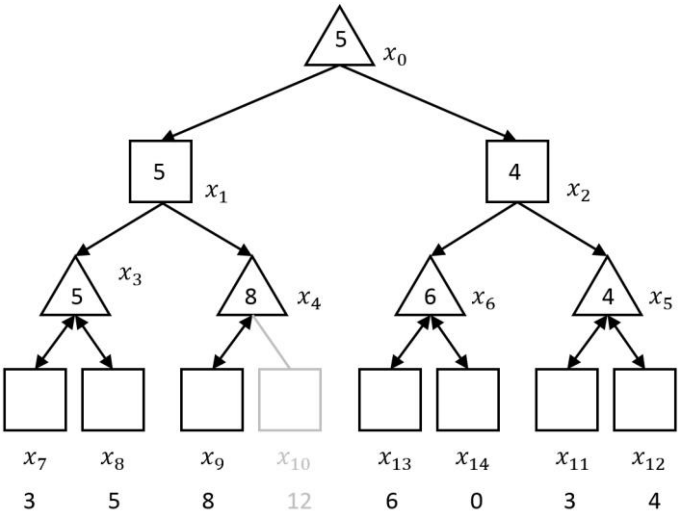


Abbildung 24: Effizienz der Alpha-Beta Kürzung nach Vertauschung der Knoten x_5 und x_6 und ihrer Kindknoten.

Ein naheliegender Ansatz in der Optimierung der Alpha-Beta Kürzung besteht somit darin, die Kindknoten vor Betrachtung zu sortieren, so dass erfolgsversprechende Knoten zuerst aufgesucht und Knoten, die eine geringere Auszahlung garantieren, zuletzt betrachtet und expandiert werden. Im Schachspiel ließe sich eine Vorsortierung der Züge leisten, indem zuerst schlagende, anschließend bedrohende und schließlich einfache Zugmöglichkeiten analysiert würden.[185] Freilich muss hierbei der verwendete Sortieralgorithmus performanter agieren als die Berechnungsprozedur des Minimax-Wertes für den betrachteten Knoten.[186] Eine solche Zugvorsortierung vorab geleistet, reduziert

[185] Vgl. Russell und Norvig 2012, S. 215.

[186] Vgl. Mańdziuk 2010, S. 20. Zur Effizienz der Alpha-Beta Kürzung mit und ohne Zugsortierung treffen u.a. Millington und Funge eine Aussage: „Even without any form of move ordering, the performance of the [Alpha-Beta] algorithm can be 10 times better than minimax alone. With excellent move ordering, it can be more than 10 times faster again, which is 100 times faster than regular minimax.“ (Millington und Funge 2009, S. 685).

sich die Zeitkomplexität auf $O(m^{\frac{d}{2}})$, kann „Alpha-Beta innerhalb derselben Zeit einen etwa doppelt so tiefen Baum gegenüber Minimax lösen.“[187]

4.1.6.2 Killerzüge und Killerheuristik

Eng mit der Vorsortierung von Zügen ist die Berücksichtigung sogenannter *Killerzüge* verbunden. Als Killerzüge werden Spielzüge bezeichnet, die in einem Bereich des Spielbaumes sehr erfolgreich dazu verwendet wurden, Unterknoten von der weiteren Suche auszuschließen. Eine *Killerheuristik*, wie sie 1977 von Akl und Newborn vorgeschlagen wurde,[188] trifft die Annahme, dass jene Killerzüge, die in einem Teil des Spielbaumes gut funktioniert haben, auch in einem anderen Teil des Spielbaumes dazu dienen können, möglichst viele Knoten des Baumes aus dem Suchprozess zu eliminieren. In der Implementation gestaltet sich die Killerheuristik simpel: Für jede Suchtiefe der Alpha-Beta Kürzung wird eine quantitativ beschränkte Liste von Killerzügen vorgehalten, die vor anderen möglichen Zügen betrachtet und analysiert werden. Findet sich ein anderer Zug in entsprechender Suchtiefe, der besser ist als einer der Killerzüge, so wird der Zug in die Liste der Killerzüge aufgenommen und ein Killerzug aus der Liste entfernt.[189] Die *History Heuristic* verallgemeinert den Ansatz der Killerheuristik, indem sie gute Züge im Laufe des Verfahrens der Alpha-Beta Beschneidung identifiziert und unabhängig von der Suchtiefe in einer Liste speichert. Die somit als vorteilhaft identifizierten Züge werden in Folgeläufen des Algorithmus zuerst betrachtet.[190]

4.1.6.3 Nullzug-Suche

Eine besonders im Schachspiel sehr effiziente und häufig genutzte Verfeinerung der Alpha-Beta Kürzung bietet die *Nullzug-Suche*.[191] Bei einem Nullzug gibt der aktuelle Spieler sein Zugrecht ab an seinen Gegenspieler – wenngleich diese erweiterte Zugmöglichkeit in den betrachteten Brettspielen ein regelwidriges

[187] Russell und Norvig 2012, S. 215.
[188] Vgl. Akl und Newborn 1977.
[189] Vgl. Mańdziuk 2010, S. 22.
[190] Vgl. Kantschick 2006, S. 38 sowie für eine Veranschaulichung des Verfahrens im Schachbeispiel: Mańdziuk 2010, S. 22
[191] Vgl. u.a. Goetsch und Campbell 1990. Eine erweiterte Variante der Null-Zug-Suche schlagen Omid und Netanyahu mit ihrem *Verified Null-Move Pruning* vor (Omid und Netanyahu 2002).

Verhalten darstellen würde.[192] Ausgehend von der Annahme, dass sich der aktuelle Spieler eines Zweispieler-Nullsummenspieles mit seinem Zugrecht in einem Vorteil gegenüber seinem Gegenspieler befindet, konstatiert die Nullzug-Heuristik, dass jeder Zug besser ist als ein Nullzug, mit welchem der aktuelle Spieler auf seinen Zug verzichtet, der Nullzug folglich immer bedeutend schlechter ist als der bestmögliche Zug.[193] Eine sehr gute Spielkonfiguration ist folglich eine Konfiguration, die noch als gut bewertet wird, nachdem der aktuelle Spieler auf den eigenen Zug verzichtet und der Gegenspieler darauf folgend einen eigenen Zug getätigt hat.[194]

Mit Omid und Netanyahu gestaltet sich das Verfahren der Nullzug-Suche wie folgend:[195] Ausgehend von der aktuellen Spielkonfiguration führt der aktuelle Spieler einen Nullzug aus – er verzichtet auf seinen Zug. Anschließend ist der Gegenspieler an der Reihe und führt einen legitimen Spielzug aus. Weil sie kein Ergebnis generieren, darf der Gegenspieler den Nullzug des aktiven Spielers nicht mit einem Nullzug beantworten; ferner ist ein einleitender Nullzug des aktuellen Spielers nicht möglich in Spielsituationen, in denen Zugzwang besteht, weil jener Nullzug einen Regelverstoß darstellen würde. Auf den Zug des Gegenspielers folgt die tiefenbeschränkte Alpha-Beta Suche nach besten Zügen im Spielbaum.

Aufgrund der zuvor referierten Annahme, dass der Nullzug immer schlechter ist als der beste Zug, wird der von der tiefenbeschränkten Alpha-Beta Suche zurückgelieferte Wert als untere Grenze verwendet, weil der Wert des besten Zuges besser sein muss als der im Rahmen des Nullzuges identifizierte. Ist der Wert größer oder gleich β, kann der entsprechende Teilbaum von dem Suchprozess ausgeschlossen werden, weil „die [Spielkonfiguration] so stark ist, dass sie bei einer guten Strategie nie erreicht werden kann.“[196] Wie Kantschick anmerkt, lässt sich das Verfahren der Nullzug-Suche nicht in allen Spielsituationen anwenden. Besonders in Spielkonfigurationen, in denen Zugzwang vorliegt, agiert die Nullzug-

[192] Im Go-Spiel hingegen ist es den Spielern erlaubt, von ihrem Zugrecht abzusehen, d.h. zu passen.
[193] Vgl. Goetsch und Campbell 1990, S. 160. Eine Ausnahme stellt der Zugzwang dar, in dem sich der aktuelle Spieler mitunter besser stünde, würde er von seinem Zugrecht absehen.
[194] Vgl. Kantschick 2006, S. 38.
[195] Vgl. Omid und Netanyahu 2002, S. 154.
[196] Kantschick 2006, S. 38.

Suche fehlerhaft und auch bei einem drohenden Schachmatt ist von der Anwendung des Verfahrens abzusehen.[197]

Die Effizienz (und der Geschwindigkeitsvorteil) der Nullzug-Suche beruht auf der Reduzierung der Suchtiefe: Anstatt zahlreiche Ebenen des Spielbaumes mit der Alpha-Beta Kürzung zu untersuchen, wird die Suche in der Praxis auf eine Tiefe von $maxDepth - R - 1$ mit $R = 2$ oder $R = 3$ beschränkt. Häufig gestaltet sich der Faktor R variabel und situationsabhängig (z.B. im Endspiel). [198]

4.1.6.4 Ruhesuche

Die iterativ vertiefende [199] Alpha-Beta Kürzung erweitert die Spielbaumsuche sukzessive, sie schließt in jeder Ebene des Spielbaumes irrelevante Knoten von der Suche aus und bewertet die durch die Knoten und Blätter des Baumes repräsentierten Spielzustände über die Evaluierungs- oder Auszahlungsfunktion – dies im Rahmen der endlichen Laufzeit des Verfahrens. In den meisten Spielen jedoch ist es weniger sinnvoll, stetig die vollständige nächste Ebene des Spielbaumes zu expandieren; vielmehr ist es in den betrachteten Spielen notwendig, vielversprechende Äste des Spielbaumes eingehender und tiefer zu betrachten und weniger lohnende Bereiche des Baumes mit geringer Suchtiefe zu betrachten.[200]

Eine Entscheidung für eine Zugwahl, die sich zunächst als sinnvoll und auszahlungsmaximierend darstellt, kann sich jenseits des Zughorizontes in ihr Gegenteil wandeln und sich als äußerst schlecht für den individuellen Spielverlauf

[197] Vgl. Kantschick 2006, S. 39.

[198] In ihrem kurzen Bericht über die Experimente mit dem Tiefenreduktionsfaktor R deuten Omid und Netanyahu an, wie essentiell die Feinjustierung des Faktors für ein stark spielendes Schachprogramm ist: „In the early days of null-move pruning, most programs used $R = 1$, which ensures the least tactical risk, but offers the least saving in comparison with other R values. Other reduction factors that were experimented with were $R = 2$ and $R = 3$. Research conducted over the years, most extensively by Heinz (1999), showed that overall, $R = 2$ performs better than the too conservative $R = 1$ and the too aggressive $R = 3$. Today, almost all null-move pruning programs, use at least $R = 2$ (Feist, 1999). However, using $R = 3$ is tempting, considering the reduced search effort resulting from shallower null-move searches. […] Experiments conducted by Heinz (1999), in his article on adaptive null-move pruning, suggest that using $R = 3$ in upper parts of the search tree and $R = 2$ in its lower parts can save 10 to 30 percent of the search effort in comparisonwith a fixed $R = 2$, while maintaining overall tactical strength.“ (Omid und Netanyahu 2002, S. 155).

[199] Vgl. das Kapitel 4.1.1.2.

[200] Vgl. Mańdziuk 2010, S. 26.

herausstellen. Die *Ruhesuche* (*Quiescence Search*), von Shannon bereits in seiner 1950 publizierten Arbeit *Programming a Computer for Playing Chess* [201] als Gegenpart zu rechenintensiven Spielbaumsuchverfahren vorgeschlagen, fokussiert dieses Problem, das auch als *Horizonteffekt* bezeichnet wird.

Ein kurzes Beispiel mag der Verdeutlichung des Horizonteffektes dienen: Zu einem bestimmten Zeitpunkt der Schachpartie ist es Schwarz entweder möglich, mit seiner Dame einen gegnerischen Bauern zu schlagen oder auf ein angrenzendes freies Feld zu ziehen. Bei einer Suchtiefe von $maxDepth = 1$ werden die beiden Zugmöglichkeiten von Schwarz unter Verwendung der Evaluierungsfunktion bewertet. Weil der schlagende Zug den Materialwert des Gegners verringert, wird der Zug auf das angrenzende freie Feld schlechter beurteilt als der Zug, der den weißen Bauern vom Spielbrett nimmt. Aufgrund der einzig auf die nächste Ebene des Spielbaumes beschränkten Suche jedoch befindet sich ein weiterer weißer Bauer jenseits des Suchhorizontes, der im nächsten Spielzug die schwarze Dame schlägt und den Materialwert von Schwarz deutlich schwächt.

Bei der Ruhesuche werden Verzweigungen des Spielbaumes eingehender betrachtet, die solche *unruhigen* Spielsituationen repräsentieren – Spielsituationen, in denen ein Schlagabtausch der Spieler stattfindet, die berechneten Werte der Evaluierungsfunktion folglich stark variieren. Die Ruhesuche wartet ab, bis sich die Spielsituation beruhigt hat und evaluiert anschließend die Spielkonfiguration.[202]

4.1.6.5 Zugumstellungstabellen

Im Schachspiel generieren die folgenden vier unterschiedlichen Zugfolgen denselben Spielzustand, in der sich schließlich die beiden Springer von Schwarz und Weiß am Rande des Spielfeldes befinden:

1. b1-a3, b8-a6, g1-h3, g8-h6

   oder

2. b1-a3, g8-h6, g1-h3, b8-a6

[201] Shannon 1950.

[202] Vgl. Carion: Creating a Chess-Playing Computer Program 2013, S. 13. Online verfügbar unter http://www.ulyssecarion.com/ComputerChess.pdf, zuletzt geprüft am 05.11.2013.

oder

3. g1-h3, b8-a6, b1-a3, g8-h6

   oder

4. g1-h3, g8-h6, b1-a3, b8-a6

Zugumstellungstabellen, auch als *Transposition Tables* bezeichnet, machen sich den Umstand zu Nutze, dass sich eine bestimmte Spielkonfiguration durch unterschiedliche Zugfolgen erreichen lässt. Häufig implementiert als Hashtabelle, wird die entsprechende Spielkonfiguration als Hashschlüssel sowie die beste Zugmöglichkeit mitsamt ihrer Bewertung und der Suchtiefe gespeichert, die der Identifizierung jenes besten Zuges diente.[203] Im Laufe des Spieles dient die Zugumstellungstabelle dazu, in ihr bereits gespeicherte Spielkonfigurationen von der Betrachtung auszuschließen. Nähert sich die aktuelle Suchtiefe dem in der Tabelle gespeicherten an, so gilt der Tabelleneintrag als veraltet und so wird die überkommene Spielkonfiguration erneut analysiert.[204] Determiniert sind die Zugumstellungstabellen durch den zur Verfügung stehenden (Arbeits)speicher.

[203] Vgl. Mańdziuk 2010, S. 23.
[204] Vgl. Kantschick 2006, S. 37.

4.1.7 State of the Art: Ausgewählte KI-Implementationen

Computerprogramme und -systeme, die künstliche Intelligenz unter Verwendung der in den vorherigen Kapiteln vorgestellten Mechanismen implementieren, sind so vielfältig und so zahlreich wie die sie umrahmende Forschungsliteratur. Die folgenden Subkapitel treffen eine Auswahl aus der Vielzahl von brettspielenden künstlichen Intelligenzen und geben einen kurzen Überblick, zum einen über *Chinook*, den perfekten Computergegner im Damespiel, zum anderen sei mit *Deep Blue* auf das erste Computersystem hingewiesen, das es 1997 vermochte, einen amtierenden Schachgroßmeister in einem offiziellen Wettbewerb zu schlagen. Abschließend wird mit *Stockfish* eine kostenlose quelloffene Schach-Engine vorgestellt, die es auf zeitgemäßen Rechnern mit eingeschränkten Ressourcen vermag, meisterlich zu spielen. Welche Ansätze sich im Backgammonspiel anbieten, um schlaue artifizielle Gegner zu gestalten, das sei an späterer Stelle ausgeführt, nachdem Mechanismen und Verfahrensweisen der Computational Intelligence wie *künstliche Neuronale Netze* und Methoden des verstärkenden Lernens (*reinforcement learning*) vorgestellt wurden. Eine leisstungsstarke und spielunabhängige Alternative zu Minimax-basierten Verfahren bieten Monte Carlo Methoden, die – neben der besprochenen Alpha-Beta Kürzung – einen weiteren Grundbaustein des im Rahmen der vorliegenden Arbeit entwickelten Frameworks *SpoookyJS* bilden und die in Kapitel 4.2 vorgestellt werden.

4.1.7.1 Perfektes Damespiel: Chinook

Eine Damepartie gegen die Software *Chinook* zu spielen, mag zwar kognitiv durchaus fordernd sein, wer jedoch glaubt, gegen das Computerprogramm gewinnen zu können, irrt: Als die Online-Ausgaben der Tageszeitung *New York Times* und der Wochenzeitschrift *Nature* im Jahr 2007 mit ihren Beiträgen *Death of Checkers, The*[205] und *Checkmate for checkers*[206] das Damespiel mit einem Augenzwinkern zu Grabe trugen, da informierten sie nicht etwa darüber, dass das Interesse am Spiel enorm abgenommen habe. Nein, mit ihren Berichten reagierten

[205] Thompson 2007.
[206] Geller 2007.

Nature und *New York Times* auf den Fachbeitrag *Checkers is solved*[207], den Schaeffer et al. kurz zuvor in der Zeitschrift *Science* publizierten.

Mit ihrem Aufsatz dokumentieren die Autoren ihre Arbeit an der Software *Chinook*[208] und legen dar, dass eine perfekt spielende Spielerin gegen *Chinook* einzig und im besten Falle ein Unentschieden erspielen kann:

> „With this paper, we announce that checkers has been weakly solved. From the starting position […], we have a computational proof that checkers is a draw. The proof consists of an explicit strategy that never loses – the program can achieve at least a draw against any opponent, playing either the black or white pieces. That checkers is a draw is not a surprise; grandmaster players have conjectured for decades.“[209]

Mit ihrem Ausspruch, das Damespiel sei nun schwach gelöst (*weakly solved*), verweisen die Autoren auf die von Allis angebotene hierarchische Klassifizierung von Lösungen, die Zweispieler-Nullsummenspiele mit perfekter Information fokussiert:[210]

1. Ultraschwach gelöst (*ultra-weakly solved*): Bei der ultraschwachen Lösung ist zu Beginn des Spieles der spieltheoretische Wert des Spieles bekannt, angenommen, beide Spieler agierten fehlerfrei. Unbekannt sind jedoch die Strategien, die es den Spielern ermöglichen, die entsprechende Auszahlung zu erhalten.
2. Schwach gelöst (*weakly solved*): In der schwachen Lösung ist der spieltheoretische Wert des Spieles sowie eine Strategie (bzw. eine Strategiefolge) bekannt, die es ermöglicht, die entsprechende Auszahlung ausgehend von der initialen Spielkonfiguration zu erspielen.

[207] Schaeffer et al. 2007.
[208] Die Website http://webdocs.cs.ualberta.ca/~chinook/play (zuletzt erfolgreich aufgerufen am 10.10.2013) informiert umfangreich über die Software und ermöglicht es, gegen *Chinook* über ein Java-Interface zu spielen.
[209] Schaeffer et al. 2007, S. 1518.
[210] Die Hierarchie der Lösungsklassifizierungen ist wiedergegeben nach Allis 1994, S. 7f. und Schaeffer et al. 2005, S. 292.

3. Stark gelöst (*strongly solved*): Stark gelöst sind Spiele wie Tic Tac Toe, bei denen sich für jede Spielkonfiguration und -situation die beste Strategie ermitteln lässt.

Allis weist darauf hin, dass die Lösungsklassifikationen eng miteinander verbunden sind und eine Rangordnung der Definitionen untereinander besteht. So ist jedes stark gelöste Spiel zugleich schwach gelöst und so ist jedes schwach gelöste Spiel auch ultraschwach gelöst.[211]

Bereits 13 Jahre vor Veröffentlichung des Aufsatzes erspielte sich *Chinook* den Weltmeistertitel im Damespiel. So vermochte es die Software, in dem 1994 ausgetragenen Turnier gegen den damaligen amtierenden Dameweltmeister Marion Tinsley, alle sechs Spielpartien des Wettbewerbes mit einem Unentschieden zu beenden. Weil sich Tinsley krankheitsbedingt zurückziehen musste, erhielt *Chinook* den Titel zugesprochen, das erste Computerprogramm zu sein, das im spielerischen Wettstreit gegen menschliche Intelligenz zu bestehen vermochte.

Die enorme Spielstärke von *Chinook* ist das Resultat jahrzehntelanger Arbeit an der Software und gründet zum einen im technischen Fortschritt, der leistungsfähigere Prozessoren, größere Arbeitsspeicher- und Festplattenkapazitäten hervorbrachte und zum anderen in äußerst umfangreichen Eröffnungs- und Endspieldatenbanken sowie effizienten Algorithmen. Um jene Verschiebung der Grenzen Künstlicher Intelligenz zu erreichen, wie sie die Autoren anmerken[212] – und wie sie auch mit den eingangs zitierten Tages- und Wochenzeitschriftsbeiträgen anklingt –, rechneten seit 1989 dutzende Computer ununterbrochen an der Lösung des Damespieles. – Einer Lösung, die in einer Datenbank resultiert, die rund 3.9×10^{13} (39 Billionen)

[211] Vgl. Allis 1994, S. 9.

[212] Vgl. Schaeffer et al. 2007, S. 1518: „The checkers result pushes the boundary of artificial intelligence (AI). In the early days of AI research, the easiest path to achieving high performance was believed to be emulating the human approach. This was fraught with difficulty, especially the problems of capturing and encoding human knowledge. Human-like strategies are not necessarily the best computational strategies. Perhaps the biggest contribution of applying AI technology to developing gameplaying programs was the realization that a search-intensive ('brute-force') approach could produce high-quality performance using minimal application-dependent knowledge."

Spielkonfigurationen mit ≤ 10 Spielfiguren auf dem Spielbrett und ihren spieltheoretischen Bewertungen umfasst und in komprimierter Form mehr als 200 Gigabyte an Festplattenspeicher einnimmt.[213]

4.1.7.2 Der erste nichtmenschliche Schachweltmeister: Deep Blue

Enorme Rechenleistung gegen menschliche Intelligenz – das erste Computersystem, das es vermochte, einen amtierenden Schachgroßmeister in einem offiziellen Wettbewerb zu schlagen, agierte im Jahre 1996 noch zögerlich: *Deep Blue I* errechnete sich im ersten Spiel zwar einen Sieg, verlor die nachfolgenden Spiele des Turnieres jedoch gegen den damaligen amtierenden Schachweltmeister Garri Kasparov. Im darauffolgenden Jahr hingegen bestimmte der Computer den Turnierverlauf und den Wettbewerb gegen Kasparov mit zwei gewonnen Spielen und drei unentschiedenen Spielpartien. Kasparov meisterte und gewann einzig eine Spielpartie gegen *Deep Blue II*.[214]

Der eindrückliche Auftritt des Schachcomputers, den Nilsson im Kapitel *Außergewöhnliche Leistungen* seiner Monographie über die Genese und Entwicklung künstlicher Intelligenz verortet,[215] war zum einen das Resultat hochperformanter Rechenleistung, die in dedizierter Hardware gründet.[216] So manifestierte sich *Deep Blue II* als *IBM RS/6000 SP* Computer mit 30 Prozessorkernen, bei dem jeder Prozessor mit 16 eigens entwickelten Schachchips interagierte, von denen es jeder Schachchip vermochte, zwei bis zweieinhalb Millionen Schachpositionen in der Sekunde zu durchsuchen und hinsichtlich ihrer Güte zu bewerten. Die Suche wurde mit dem iterativ vertiefenden *NegaScout*-Algorithmus – einer Erweiterung der Alpha-Beta Kürzung – implementiert und

[213] Vgl. Schaeffer et al. 2007, S. 1519f. In ihrem 2004 veröffentlichten Aufsatz berichten die *Chinook*-Entwickler über die besonderen soft- und hardwareseitigen Herausforderungen in ihrer Arbeit an der Endspieldatenbank. So implementierten die Forscher ein eigenes Kompressionsverfahren, das auf Lauflängencodierung basiert, um Anfragen an die enorm umfangreiche Datenbank in Echtzeit zu ermöglichen (vgl. Schaeffer et al. 2004).

[214] Die Benennung der unterschiedlichen Versionen des bei IBM entwickelten Computersystems in *Deep Blue I* und *Deep Blue II* folgt der Namensgebung, wie sie sich bei Campbell et al. 2002, S. 57f. – wenngleich dort nicht durchgehend konsistent verwendet – findet.

[215] Vgl. das Kapitel *Extraordinary Achievements* in Nilsson 2010, S. 481–500.

[216] Die Beschreibung der Hardware von *Deep Blue II* ist wiedergegeben nach Campbell et al. 2002, S. 60f.

arbeitete mit Zugumstellungstabellen und Ruhesuche.[217] Jeder Komplex von Prozessor und dedizierten Schachchips, von den Autoren als *Knoten* (*Nodes*) bezeichnet, verfügte über einen Arbeitsspeicher von einem Gigabyte und vier Gigabyte Festplattenspeicher. Das Gros der Prozessoren arbeitete mit einer Taktung von 120 MHz.

Zwar war jeder Rechentakt und jede Speicherzelle des Computersystems auf Kasparovs Spielverhalten maßgeschneidert und repräsentierten umfangreiche Datenbanken das über unzählige Spielpartien gesammelte und analysierte Spielverhalten Kasparovs und weiterer Großmeister.[218] Das entscheidende Moment jedoch für Kasparovs Niederlage gegen *Deep Blue II* will Silver in seiner 2012 publizierten Monographie *The Signal and the Noise – The Art and Science of Prediction*[219] in einer unerhörten Begebenheit ausfindig gemacht haben: Ein temporär in Erscheinung getretener Fehler des Computersystems sei dem Autor zufolge Grund für Kasparovs Niederlage gegen *Deep Blue II*. Wie Campbell, einer der Entwickler von *Deep Blue II* und seinen Vorgängern, in einem Dialog mit Silver berichtet, sei es dem Computersystem zu einem bestimmten Zeitpunkt des Spieles gegen Kasparov nicht möglich gewesen, einen vielversprechenden Zug zu identifizieren, so dass das Computersystem schließlich einen Zufallszug ausgeführt habe – einen Zufallszug, der Kasparov Silver zufolge verwirrte und ihn eine andere Langzeitstrategie und elaborierte Fähigkeiten des Systems vermuten ließ: „A bug occured in the game and it may have made Kasparov misunderstand the capabilities of Deep Blue […] He didnt‘ come up with the theory that the move that it played was a bug.“[220]

[217] Vgl. Mańdziuk 2010, S. 42.

[218] In seinen *Frequently Asked Questions: Deep Blue* führt *IBM* aus, Kasparov habe ob des gesammelten Expertenwissens von *Deep Blue II* nicht gegen eine Maschine gespielt, sondern sei vielmehr gegen den Esprit früherer Großmeisterinnen und Großmeister angetreten: „Deep Blue applies brute force aplenty, but the ‘intelligence‘ is the old-fashioned kind. Think about the 100 years of grandmaster games. Kasparov isn't playing a computer, he's playing the ghosts of grandmasters past. That Deep Blue can organize such a storehouse of knowledge – and apply it on the fly to the ever-changing complexities on the chessboard – is what makes this particular heap of silicon an arrow pointing to the future.“ (IBM: Frequently Asked Questions: Deep Blue. Online verfügbar unter https://www.research.ibm.com/deepblue/meet/html/d.3.3a.html, zuletzt geprüft am 03.11.2013).

[219] Silver 2012.

[220] Silver 2012, S. 288.

Wenn der Indikativ dem *Hörsensagen* anheim gegeben ist, wie es sich mit jenem Fehler des Computersystems verhält, dann sei die kapitelabschließende Frage durch den Konjunktiv bestimmt: Hätte das Turnier zwischen Kasparov und *Deep Blue II* den gleichen Ausgang gefunden, wäre der Fehler in der Software nicht aufgetaucht?[221]

4.1.7.3 Meisterliches Schachspiel mit beschränkten Ressourcen: Stockfish

Dass meisterliches Spielverhalten auch jenseits dedizierter Hardware möglich ist, wie sie im vorangehenden Kapitel mit *Deep Blue II* betrachtet wurde, das offenbart sich mit der quelloffenen kostenlosen Schach-Engine *Stockfish*. *Stockfish*, auf der *Computer Chess Rating Lists Website* in zahlreichen Wettkampfsituationen gelistet unter den leistungsfähigsten Computerschach-Engines,[222] stellt seine Spielstärke graphischen Benutzerschnittstellen über das *Universal Chess Interface* (*UCI*)[223] bereit und liegt als Kompilat für die Plattformen und Betriebssysteme Microsoft Windows, Linux, Mac OS X und als mobile Anwendung für Android (*DroidFish*) und iOS (*SmallFish*) vor. Entwickelt in der objektorientierten Programmiersprache *C++*, sucht *Stockfish* mit einer erweiterten Alpha-Beta Kürzung nach besten Zügen und verwendet (u.a.) Zugumstellungstabellen sowie ein umfangreiches binär vorliegendes Eröffnungsbuch zur Implementation eines schlau agierenden Computergegners.[224]

---

[221] So fragt auch Finley 2012.

[222] Zum Zeitpunkt der vorliegenden Ausarbeitung wurde die Spielleistung von *Stockfish* in den drei Wettbewerben *40/40* (beschränkt auf 40 Züge in 40 Minuten), *40/4* (beschränkt auf 40 Züge in vier Minuten) sowie *FRC 40/4* (die auf 40 Züge in vier Minuten beschränkte Schachvariante *Fischer Random Chess*, bei der die Spielfiguren in der jeweils letzten Reihe (die Bauern behalten ihren ursprünglichen Platz bei) zu Beginn des Spieles zufallsbasiert angeordnet werden) einzig übertroffen von den kommerziellen Schach-Engines *Houdini* und *Komodo* (vgl. computerchess.org.uk: Computer Chess Rating Lists Website. Online verfügbar unter http://computerchess.org.uk/, zuletzt geprüft am 06.11.2013). 2014 gewann *Stockfish* die Thoresen Chess Engines Competition (TCEC), bei der zahlreiche Schach-Engines in regelmäßig ausgetragenen Wettbewerben gegeneinander antreten (vgl. http://www.chessdom.com/stockfish-is-the-winner-of-tcec-season-6-stage-3-houdini-komodo-and-critter-also-qualify, zuletzt erfolgreich aufgerufen am 01.07.2014).

[223] Das *UCI*-Protokoll wurde ab dem Jahr 2000 entwickelt und findet sich beschrieben unter Shredder Computer Schach: UCI-Protokoll. Online verfügbar unter http://www.shredderchess.de/schach-info/features/uci-universal-chess-interface.html, zuletzt geprüft am 07.11.2013.

[224] Der Quellcode von *Stockfish* findet sich frei verfügbar im offiziellen Repository unter https://github.com/mcostalba/Stockfish (zuletzt erfolgreich aufgerufen am 28.04.2014).

4.1.8 Zusammenfassung und Ausblick

Künstliche Intelligenz in digitalen Brettspielen, wie sie sich in den vergangenen Kapiteln manifestierte, gründet in dem spieltheoretischen Lösungskonzept des Minimax-Theorems nach von Neumann und Morgenstern und seinen Implementierungen, dem Mini- bzw. Negamax-Algorithmus und der Alpha-Beta Kürzung zur Suche nach optimalen Entscheidungen im Spielbaum. Gestaltet sich die Einbettung menschlichen Expertenwissens – u.a. in Form breit gefächerter Spieldatenbanken – als unabdinglicher Baustein, um spielstarke artifizielle Gegner hervorzubringen, so erweitern Computersysteme wie *Deep Blue* oder *Chinook* den Blickwinkel auf die betrachteten Mechanismen um ein gewichtiges Moment: Künstliche Intelligenz in brettspielenden Systemen, wie sie im vergangenen Kapitel zum Stand der Dinge besprochen wurden, findet sich zumeist sehr stark verknüpft mit immer leistungsfähigerer, häufig gar dedizierter Hardware, die den Spielbaumsuchverfahren zu schnelleren Ergebnissen verhelfen, zeitnahe Entscheidungsfindung zuallererst gar zu ermöglichen – eine Tendenz zur Hardwarezentrierung, die der Pfiffigkeit um die Forschung künstlicher Intelligenz in digitalen Brettspielen entgegenwirkt?

Gilt das Damespiel mit dem System *Chinook* als schwach gelöst und lassen sich zum einen der Erfolg, zum anderen die massiv verwendeten Hardwareressourcen beim medienwirksamen Turnier Kasparov gegen *Deep Blue* – beim Kampf Mensch gegen Maschine – bestaunen, so bietet das japanische Go-Spiel ob seiner außerordentlichen Spielbaumkomplexität[225] eine besondere Herausforderung in der Implementation schlau agierender Computergegener. Eine Herausforderung, die „gänzlich neue Techniken“[226] zur Suche nach sinnvollen Spielentscheidungen erfordert, weil die enorme Spielbaumkomplexität des Go-Spieles nicht mit den zuvor referierten Suchverfahren nach optimalen Zügen handhabbar ist – und weil

[225] Die Spielbaumkomplexität ist definiert als die Anzahl der Blattknoten des Spielbaumes und repräsentiert folglich die Anzahl aller möglichen Strategiewahlen von der Initialkonfiguration bis zum Ende des Spieles (vgl. Allis 1994, S. 160). Zum Vergleich und zur Einordnung der Spielbaumkomplexität: Die japanische Schachvariante *Shogi* verfügt über eine Spielbaumkomplexität von 10^{226} (vgl. van den Herik, H. Jaap et al. 2002, S. 300) und das Go-Spiel über eine Spielbaumkomplexität von 10^{360}, wird es auf einem 19×19 großen Spielbrett gespielt (vgl. Allis 1994, S. 174) – hier offenbart sich, dass sich die zuvor betrachteten Verfahren zur Suche nach optimalen Strategien in Spielbäumen nur bedingt einsetzen lassen.
[226] Bewersdorff 2012, S. 197.

das Go-Spiel noch immer eines der wenigen klassischen Spiele ist, in dem der Mensch dem Computer weit vorauszudenken fähig ist.[227]

Als besonders leistungsfähige, jedoch erstaunlich schlichte Alternative zu den zuvor besprochenen Verfahren, nimmt sich die Monte Carlo Spielbaumsuche der Herausforderung an, optimale Entscheidungsfindung in unterschiedlichen Spieldomänen zu leisten. Anwendbar nicht einzig in Zweispieler-Nullsummenspielen mit perfekter Information, sondern auch in Spielen mit Zufallselementen wie Backgammon sowie Spielen mit imperfekter Information wie Poker, Scrabble oder Bridge, feierte die Monte Carlo Spielbaumsuche im Jahre 2006 ihren Durchbruch.[228]

[227] Browne et al. 2012, S. 2.
[228] Browne et al. 2012, S. 7.

4.2 Künstliche Intelligenz und das Moment des Zufalls – Monte Carlo Verfahren

Das im Jahre 1949 von Metropolis und Ulam [229] vorgetragene Monte Carlo Verfahren nähert sich der Lösung eines definierten Problemes durch Akkumulation der Ergebnisse zahlreicher zufallsbasierter Simulationsläufe an. Zuerst in der statistischen Physik eingesetzt zur Lösung von Integralen,[230] schlägt Abramson in seinem 1990 veröffentlichten Aufsatz *Expected-outcome: A general model of static evaluation*[231] vor, Monte Carlo Methoden zur Annäherung des spieltheoretischen Wertes einer Zugmöglichkeit zu verwenden. So wird das Spiel bei dem basalen flachen Monte Carlo Verfahren ausgehend von der zu evaluierenden Zugmöglichkeit virtuell und zufallsbasiert mehrere Tausend Male zu Ende gespielt. Zufallsbasiert, das bedeutet, dass die Spieler in den simulierten Spielen ihre Zugwahlen nacheinander jeweils zufällig treffen. Die Wertigkeit der betrachteten Zugmöglichkeit bestimmt sich nach anschließend, nach zahlreichen Läufen des Verfahrens, als die durchschnittliche Auszahlung aller simulierten Spiele. Die nachfolgende Grafik veranschaulicht das flache Monte Carlo Verfahren anhand eines einfachen Spielbaumes:

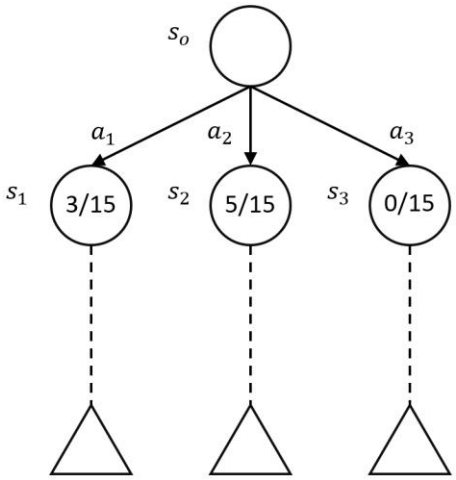


Abbildung 25: Beispielvisualisierung des flachen Monte Carlo Verfahrens zur Identifikation guter Spielzüge. Die Knotenbezeichnungen in der Form n/m informieren über die Anzahl der gewonnenen Spiele (n) aus der Menge der simulierten Spiele (m).

[229] Vgl. Metropolis und Ulam 1949.
[230] Vgl. Browne et al. 2012, S. 4.
[231] Abramson 1990.

In der Beispielgrafik bieten sich dem spielenden Agenten die drei Handlungsmöglichkeiten a_1, a_2 oder a_3, um das Spiel von seinem Initialzustand s_0 zu einem aktionsabhängigen Folgezustand s_1, s_2 oder s_3 zu überführen. Welche Aktion wahrscheinlich die höchste Auszahlung generiert, bestimmt sich bei dem flachen Monte Carlo Verfahren zufallsbedingt: Virtuell, d.h. ohne Folgen für die aktuell gespielte Spielpartie, werden die möglichen Aktionen ausgeführt und auf Grundlage des jeweils resultierenden Spielzustandes wird eine Anzahl von Simulationen ausgeführt – in der Grafik mit gestrichelten vertikalen Linien visualisiert –, in denen das Spiel mit randomisierten Zugwahlen der Spieler bis zu seinem Ende gespielt wird. Durch die Anzahl der gewonnenen Spiele und die Quantität der simulierten Spiele lässt sich anschließend mutmaßen, dass eine Zugmöglichkeit zu bevorzugen ist, die das Spiel in den simulierten Spielen häufiger zu einem individuell positiven Ende führt, als ein Zug, der nur in wenigen simulierten Spielen zum Gewinn führt. Die Beschriftungen der Spielzustände s_1 bis s_3 geben hinter dem trennenden Schrägstrich Aufschluss darüber, wie viele Spiele simuliert wurden und verzeichnen vor dem Schrägstrich die Anzahl der gewonnenen simulierten Spiele. Unter Berücksichtigung der Gewinnrate eines Spielzustandes, die sich als die Anzahl der Gewinne, dividiert durch die Anzahl der simulierten Spiele bestimmt, sollte der Agent die Aktion a_2 wählen, weil sie mit einem Wert von $0{,}3\overline{3}$ die wahrscheinlich beste Aktion der Dreien ist.

Wurde der Mechanismus der Simulation zunächst nur flüchtig betrachtet, sei im folgenden Kapitel näher eingegangen auf Simulation und Simulationsstrategie. Anschließend wird die Monte Carlo Spielbaumsuche vorgestellt, der sich ein besonders starker Einfluss auf die Forschung um künstliche Intelligenz in Problemdomänen sequentieller Entscheidungsfindung wie Brettspielen anmerken lässt.[232] Mit dem Selektionsalgorithmus *Upper Confidence Bounds applied to Trees* und der *All Moves as First* Heuristik werden Erweiterungen der Monte Carlo Spielbaumsuche vorgestellt, die zeitnahe Entscheidungsfindung in unterschiedlichen Spieldomänen ermöglichen. Den Abschluss der Betrachtung

[232] Vgl. Browne et al. 2012, S. 1.

bilden Kapitel zur Laufzeitoptimierung und zum Stand der Kunst in der Implementation schlauer Computergegner auf Basis von Monte Carlo Methoden.

4.2.1 Monte Carlo Simulation

Die essentielle Kernidee des Monte Carlo Verfahrens manifestiert sich in der Simulation.[233] Als die Simulation eines Spieles sei eine Folge von Aktionen definiert, die den Lauf des Spieles von seinem Ausgangs- zu seinem Terminalzustand führt, an dem die Auszahlung z ausgeschüttet wird. Welche Aktion zu einem bestimmten Zeitpunkt der Simulation ausgeführt wird, ist bestimmt von der Simulationsstrategie $\pi(s, a)$, im Englischen als *simulation policy* bezeichnet, die eine Aktion a aus dem endlichen Vorrat legitimer Handlungsmöglichkeiten A auswählt, die in der Spielwelt in ihrem aktuellen Zustand s zur Verfügung stehen.

Den spieltheoretischen Wert des zu evaluierenden Spielzustandes $Q^{\pi}(s_0, a)$ nähert die Monte Carlo Simulation an, indem $N(s)$ vollständige Spiele unter Verwendung der Strategie π vom aktuell betrachteten Spielzustand s_0 bis zum terminalen Zustand gespielt werden. Bestimmen lässt sich der Q-Wert einer Aktion a zu einem Spielzustand s als Auszahlung, die zu erwarten ist, wenn Aktion a ausgeführt wird:[234]

$$Q(s, a) = \frac{1}{N(s, a)} \sum_{i=1}^{N(s)} \mathbb{I}_i(s, a) z_i \tag{4.2}$$

In der Gleichung signifiziert z_i die Auszahlung der i-ten Simulation und mit dem Ausdruck $N(s, a) = \sum_{i=1}^{N(s)} \mathbb{I}_i(s, a)$ ist die Anzahl von Simulationen bestimmt, in der die Aktion a im Spielzustand s ausgeführt wurde. Die Indikatorfunktion $\mathbb{I}_i(s, a)$ gibt den ganzzahligen Wert 1 zurück, wenn die Aktion a bei der i-ten

[233] Browne et al. weisen darauf hin, dass der Begriff der Simulation in der Fachliteratur auf zwei unterschiedliche Arten verwendet wird: „Some authors take it to mean the complete sequence of actions chosen per iteration during both the tree and default policies […] while most take it to mean the sequence of actions chosen using the default policy only“ (Browne et al. 2012, S. 6). Die Erläuterung des Begriffes der Simulation und die Formalisierung des Monte Carlo Verfahrens orientieren sich an den Ausführungen von Gelly und Silver 2011, S. 1857f.

[234] Die Gleichung um die Bestimmung des Monte Carlo Wertes $Q(s, a)$ ist wiedergegeben nach Gelly und Silver 2011, S. 1857; die Erläuterung des Verfahrens orientiert sich an den Ausführungen der Autoren.

Simulation im Spielzustand s ausgeführt wurde und liefert im anderen Falle den Wert 0 zurück.[235] In der vorgetragenen Form dient die Monte Carlo Simulation dem Zweck, die verschiedenen Aktionsmöglichkeiten zu bewerten, die sich zu einem bestimmten Spielzustand bieten; die Simulationsstrategie $\pi(s, a)$ ist hierbei zunächst jedoch fix und unveränderlich,[236] lässt sich jedoch mit den im Folgenden durchgeführten Überlegungen und betrachteten Erweiterungen flexibel gestalten.

4.2.2 Monte Carlo Spielbaumsuche

Wurde bei den zuvor betrachteten Verfahren der Alpha-Beta Kürzung und ihren – mitunter spielspezifischen Optimierungen – der Spielbaum so tief expandiert und mit Hilfe der Evaluierungsfunktion analysiert, wie es die zur Verfügung stehenden Hardwareressourcen in einem praktikablen Zeitrahmen ermöglichen, so ist bei dem Monte Carlo Algorithmus zur Spielbaumsuche der Zufall das konstituierende Moment: Der Wert eines Knotens im Spielbaum wird abgeschätzt, indem ausgehend von dem entsprechenden Knoten zahlreiche Partien simuliert werden, in denen die Spieler ihre Zugwahlen zufallsbedingt treffen. Die folgende Grafik veranschaulicht das Verfahren, in dem die Knotenwerte eines Minimax Spielbaumes durch die Monte Carlo Spielbaumsuche angenähert werden:

[235] Vgl. Gelly und Silver 2011, S. 1858.
[236] Vgl. Gelly und Silver 2011, S. 1858.

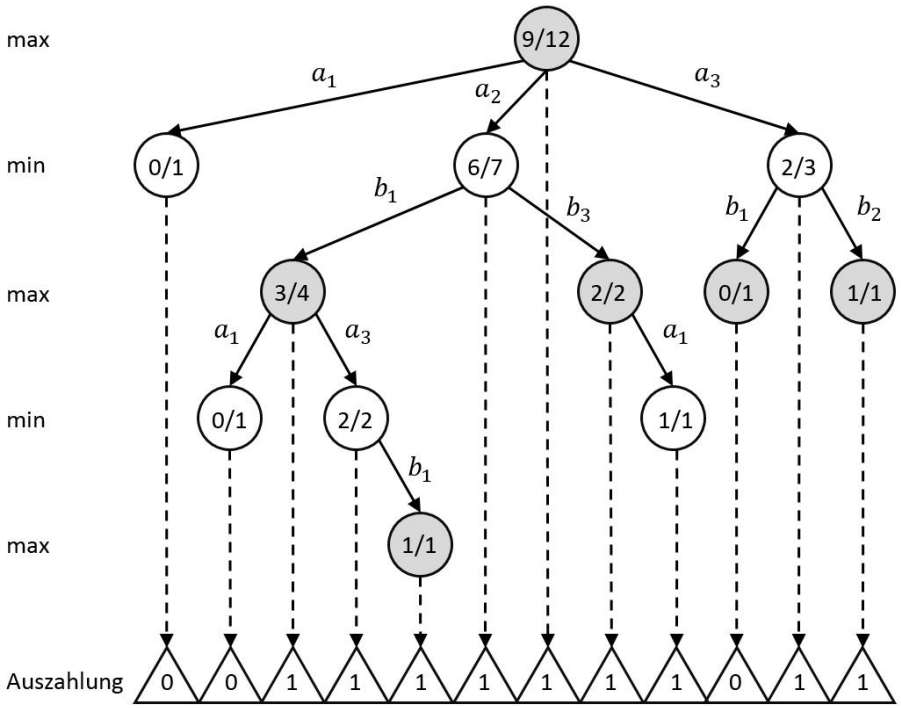


Abbildung 26: Annäherung der Knotenwerte eines Minimax Spielbaumes unter Verwendung der Monte Carlo Spielbaumsuche. Quelle: Gelly et al. 2012, S. 109.

Abgesehen von den Spielregeln und von der Auszahlungsfunktion, die den Wert der terminalen Knoten bestimmt, benötigt die Monte Carlo Spielbaumsuche, im Folgenden mit *MCSS* abgekürzt, kein spielspezifisches Wissen, um leistungsstark zu agieren und eignet sich somit als Lösungsmechanismus im Kontext des *General Game Playing*,[237] auf das im späteren Verlauf noch einzugehen sein wird, wenn das Framework *SpoookyJS* besprochen wird. Ihren Lauf nimmt die MCSS mit den vier Phasen der Selektion, Expansion, Simulation und dem Zurückreichen der Simulationswerte an den Wurzelknoten, wie die folgende Darstellung visualisiert:[238]

[237] Vgl. Finnsson 2012, S. 1550.

[238] Die Darstellung der Monte Carlo Spielbaumsuche ist wiedergegeben nach Chaslot et al. 2008a, S. 216 und Browne et al. 2012, S. 6.

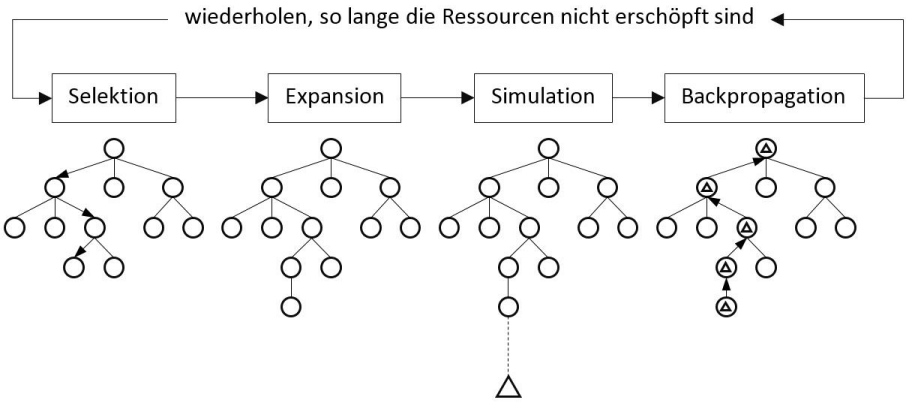


Abbildung 27: Schematische Darstellung der Monte Carlo Spielbaumsuche mit ihren vier Phasen der Selektion, Expansion, Simulation und Rückpropagation.

Im ersten Schritt der rekursiv verfahrenden MCSS wird ein noch nicht analysierter nichtterminaler Knoten ausgehend von dem Wurzelknoten aufgesucht. Welcher Knoten ausgewählt wird, ist abhängig von der Selektionsstrategie (*Tree Policy*), die im folgenden Kapitel mit dem Selektionsalgorithmus *Upper Confidence Bounds applied to Trees* näher betrachtet wird. Wie Chaslot et al. anmerken, obliegt es der Selektionsstrategie, ein Gleichgewicht herzustellen zwischen der Erkundung des Spielbaumes (*exploration*) und der vertiefenden Betrachtung besuchter Zweige des Spielbaumes, die als vielversprechend identifiziert wurden (*exploitation*).[239] An die Selektion schließt sich die Expansion des selektierten Knotens um einen Kindknoten an.

Die auf Selektion und Expansion folgende Phase der Simulation auf Grundlage einer Simulationsstrategie (*Default Policy*) ist das Kernstück der MCSS: Ausgehend von der aktuell betrachteten Spielkonfiguration, die durch den zuletzt expandierten Knoten repräsentiert ist, werden eine Vielzahl von Zufallspartien – oder durch spielspezifische Heuristiken beeinflusste Spielpartien – bis zu ihrem Ende gespielt und mit der Auszahlungsfunktion die Wertigkeit der finalen Spielkonfiguration bestimmt. Der Rückgabewert der Auszahlungsfunktion ist zumeist simpel gehalten: Der positive ganzzahlige Wert 1 steht für einen Sieg, 0 für ein Unentschieden oder -1 für eine Niederlage. Je mehr Simulationen ausgeführt werden, desto genauer gestaltet sich die Vorhersage, wie gut ein

[239] Vgl. Chaslot et al. 2008a, S. 216.

analysierter Spielzug ist.[240] Der Wert der zu evaluierenden Spielkonfiguration bestimmt sich schließlich als Durchschnittswert aller Simulationen und wird im letzten Schritt der Iteration des Algorithmus‘ an die Elternknoten weitergereicht. dieses Verfahren wird als *Rückpropagation* (*backpropagation*) bezeichnet.

Jeder Knoten verfügt über einen Wert, der die Anzahl der gewonnenen simulierten Spiele repräsentiert und verzeichnet einen Zähler, wie häufig der Knoten im Laufe des Verfahrens besucht wurde. Nach jeder Iteration der Prozedur wird der Besuchzähler inkrementiert und der Wert des Knotens aktualisiert. Wurde die letzte Phase abgeschlossen und stehen noch ausreichend Ressourcen wie Arbeitsspeicherplatz und Rechenzeit zur Verfügung, beginnt das vierschrittige Verfahren erneut von vorne. Mit steigender Anzahl der Iterationen des Verfahrens konvergiert der berechnete Auszahlungswert mit der Gewinnwahrscheinlichkeit des Spielers.

Sind die Ressourcen erschöpft oder wurde die Suche unterbrochen, lässt sich die Zugwahl ausgehend von der aktuellen Spielkonfiguration auf unterschiedliche Arten treffen, wie sie Browne et al. referieren:[241]

1. *Max child*: Wähle den Kindknoten mit der besten Bewertung.
2. *Robust child*: Wähle den im Rahmen des Verfahrens meistbesuchten Kindknoten.
3. *Max-Robust child*: Wähle den Kindknoten, der sowohl über die beste Bewertung verfügt als auch am häufigsten besucht wurde.
4. *Secure child*: Wähle den Kindknoten, der die Größe $v + \frac{A}{\sqrt{n}}$ maximiert. Hierbei ist v der Wert des Knotens, A ist der Parameter (z.B. der ganzzahlige Wert 1) und n bezeichnet die Besuchshäufigkeit des Knotens.[242]

Der nachfolgend wiedergegebene Algorithmus veranschaulicht das grundlegende Verfahren der Monte Carlo Spielbaumsuche; auf Details der Selektions- und

[240] Vgl. Gelly et al. 2012, S. 109.
[241] Vgl. Browne et al. 2012, S. 7.
[242] Das Auswahlverfahren *Secure child* wurde vorgeschlagen von Winands et al. (vgl. Winands et al. 2008, S. 31).

Simulationsstrategien sowie des Schrittes der Rückpropagation wird im folgenden Kapitel noch ausführlich eingegangen:[243]

Algorithmus 4.3 *Monte Carlo Spielbaumsuche.*

- Parameter:
  - *gameState*: Ausgangszustand der Spielwelt.
  - *timeLimit*: Maximale Laufzeit des Verfahrens.
- Rückgabe: Aktion, die zum besten Kindknoten des untersuchten Spielzustandes *gameState* führt.

```
function MCTreeSearch(gameState, timeLimit)
    create root node v_0 with state gameState
    while timeLimit > 0 do
      startTime ← Date.now()
      v_l ← TreePolicy(v_0)            // Selektion und Expansion
      Δ ← DefaultPolicy(s(v_l))       // Simulation
      Backup(v_l, Δ)     // Rückreichen der Werte an Elternknoten
      // Aktualisierung der Laufzeitbedingung
      timeLimit ← timeLimit - (Date.now() − startTime)
    return a(BestChild(v_0))
```

Durch möglichst zahlreiche simulierte Zufallsspiele nähert die Monte Carlo Spielbaumsuche die Güte eines individuellen Spielzuges an und erweist sich besonders in Spielen mit imperfekter Information als sehr erfolgreiche Alternative zu klassischen Implementationsansätzen künstlicher Intelligenz in digitalen Brettspielen wie der Alpha-Beta Kürzung. Im Unterschied zur Alpha-Beta Kürzung exploriert die MCSS den Spielbaum heterogen (einige Zweige des Spielbaumes werden eingehender untersucht, andere Verzweigungen des Spielbaumes weniger ausgiebig), funktioniert ohne Expertenwissen sowie Heuristiken und lässt

[243] Der Algorithmus der Monte Carlo Spielbaumsuche ist wiedergegeben nach Browne et al. 2012, S. 6.

sich zu jedem Zeitpunkt der Laufzeit des Verfahrens beenden, um ein praktikables Ergebnis zu erzielen.

4.2.3 Mehrarmige Banditen, UCB und UCT

Die Monte Carlo Spielbaumsuche intendiert, den spieltheoretischen Wert aller ausführbaren Strategiewahlen zu einem bestimmten Zeitpunkt des Spieles anzunähern.[244] Als zentrales zu optimierendes Moment der MCSS offenbarte sich im vergangenen Kapitel die Selektionsstrategie. Die Herausforderung der Selektionsstrategie, zwischen den Strategien *exploitation* und *exploration* zu vermitteln, also entweder bereits bekannte Zugwahlen eingehender zu untersuchen, die in früheren Iterationen des Verfahrens als vielversprechend ermittelt wurden oder noch nicht explorierte Zugmöglichkeiten zu ergründen, ist der Herausforderung gemein, die sich mit dem Problem des *mehrarmigen Banditen* (*Multi-armed bandit*) verbindet.

Bei dem mehrarmigen Banditenproblem, im Jahre 1952 von Robbins beschrieben[245] und im Folgenden mit *MBP* abgekürzt, steht ein Glücksspieler vor einem Spielautomaten mit K nicht identischen Armen und muss eine Entscheidung treffen, welchen der Arme er betätigt, um seinen Gewinn zu maximieren. Zu jedem Zeitpunkt t_1 bis t_n betätigt der Spieler einen der Hebel des Automaten und erhält eine Auszahlung. Weil davon auszugehen ist, dass sich die dem Spieler unbekannte Gewinnverteilung des Automaten unterschiedlich gestaltet, steht der Spieler vor dem Dilemma, ob er möglichst viel Zeit an einem Arm des Spielautomaten ziehen soll, von dem er denkt, dass jener regelmäßig die höchsten Gewinne ausschüttet (*exploitation*) oder ob er einen anderen Arm des Automaten explorieren soll, der möglicherweise eine höhere Ausschüttung böte.[246]

Eine Lösung der Dilemmasituation bietet der Algorithmus *Upper Confidence Bound 1* (*UCB1*), wie ihn Auer et al. 1995 vorschlagen.[247] *UCB1* berücksichtigt die durchschnittlichen Auszahlungswerte für alle Arme des Spielautomaten und trifft

[244] Vgl. Browne et al. 2012, S. 7.
[245] Robbins 1952.
[246] Vgl. Auer et al. 1995.
[247] Auer et al. 2002.

eine Auswahl aus den zur Verfügung stehenden Armen aufgrund der Auszahlungserwartung, die in den individuellen Automatenarm gelegt wird. Die Strategiewahl, die der Algorithmus *UCB1* dekretiert, besteht darin, denjenigen Arm j des Automaten zu betätigen, der den folgenden Ausdruck maximiert:[248]

$$UCB1 = \overline{X}_j + \sqrt{\frac{2\ln(n)}{n_j}} \tag{4.3}$$

Hierbei bezeichnet $\overline{X}_j$ die durchschnittliche Auszahlung des Automatenarmes j; die Anzahl, wie häufig der Arm j gespielt wurde, ist mit n_j signifiziert und n als Funktionsargument des natürlichen Logarithmus' repräsentiert die Gesamtzahl der bereits gespielten Spiele. Wie Browne et al. anmerken, bestärke der Belohnungsterm $\overline{X}_j$ das Vorgehen der Verwertung (*exploitation*) und bekräftige der Summand $\sqrt{\frac{2\ln(n)}{n_j}}$ die Exploration bislang vernachlässigter Strategiewahlen.[249]

Mit ihrem 2006 vorgestellten Algorithmus *Upper Confidence Bounds applied to Trees* (*UCT*) regen Kocsis und Szepesvári[250] an, den Algorithmus *UCB1* auf die Monte Carlo Spielbaumsuche anzuwenden: Jeder Zustand der Spielwelt wird den Autoren zufolge verstanden als ein Bandit und jeder mögliche Spielzug ist betrachtet als ein Arm des Banditen, der den Banditen zu der Ausschüttung einer – zum Zeitpunkt der Betätigung des Armes – noch unbekannten Belohnung führt. Lokalisiert in der Selektionsphase der MCSS, interpretiert UCT die Wahl des zu selektierenden Kindknotens als das Problem eines mehrarmigen Banditen: Jeder Knoten wird als mehrarmiger Bandit begriffen, dessen Arme die Kindknoten darstellen.[251] Sei I die Menge der Kindknoten des aktuell betrachteten Knotens p, so wählt die Selektionsstrategie UCT denjenigen Kindnoten k des Knotens p aus, der den folgenden Ausdruck maximiert:[252]

[248] Vgl. Browne et al. 2012, S. 5.
[249] Vgl. Browne et al. 2012, S. 5.
[250] Kocsis und Szepesvári 2006.
[251] Vgl. Kocsis und Szepesvári 2006, S. 286.
[252] Vgl. Browne et al. 2012, S. 7. Weitere Selektionsverfahren seien an dieser Stelle mit den Algorithmen *UCB1-Tuned* oder *Bayesian UCT* (vgl. Browne et al. 2012, S. 16 und Chaslot 2010, S. 19–22) angemerkt.

$$k \in argmax_{i \in I}\left(v_i + 2 \times C\sqrt{\frac{2 \times \ln(n_p)}{n_i}}\right) \tag{4.4}$$

Wie häufig der Elternknoten p im Laufe der Simulation besucht wurde, ist in dem Term mit n_p repräsentiert; n_i signifiziert, wie oft der Kindknoten i frequentiert wurde. Der Wert des Knotens i ist mit v_i bezeichnet und stellt die durchschnittliche Gewinnhäufigkeit des betrachteten Kindknotens dar. Die Konstante C mit $C \geq 0$ dient dem Zweck, den Grad der Erkundung (*exploration*) noch nicht betrachteter Strategien feinzujustieren,[253] indem der Algorithmus bei klein gewähltem C den zuerst aufgefundenen Kindknoten eingehender betrachtet und den Spielbaum tiefer expandiert als bei einem groß bestimmten Konstantenwert, der zu einer breiten Expansion des Spielbaumes führt. Betonen Chaslot et al. den experimentellen Charakter der Konstante C,[254] so schlagen schlagen Kocsis und Szepesvári mit Verweis auf die Hoeffding-Ungleichung den Wert $\frac{1}{\sqrt{2}}$ für C vor bei Auszahlungen im Intervall von [0, 1].[255] Anwendungsfälle mit Auszahlungswerten jenseits des Intervalles von [0, 1] profitierten von einem anderen Konstantenwert, wie Browne et al.[256] anmerken.

Seine Arbeit am Wurzelknoten des Spielbaumes beginnend, gestaltet sich der UCT-Algorithmus in seiner Verfahrensweise zweischrittig:[257] Im ersten Schritt wird an jedem betrachten Knoten des Spielbaumes derjenige Kindknoten ausgewählt und eingehender betrachtet, der den zuvor wiedergegebenen Ausdruck maximiert. Wurde ein im Laufe des Verfahrens noch nicht besuchter Knoten ausgewählt, beginnt die zweite Phase des Algorithmus‘: die Annäherung des Knotenwertes durch Monte Carlo Simulationen, bei der zahlreiche Zufallsspiele ausgehend vom aktuell betrachteten Knoten bis zu ihrem virtuellen Ende simuliert werden.[258] Der mit den Simulationen identifizierte Wert wird abschließend

[253] Vgl. Browne et al. 2012, S. 8.
[254] Vgl. Chaslot et al. 2008b, S. 347.
[255] Vgl. Kocsis und Szepesvári 2006, S. 288.
[256] Vgl. Browne et al. 2012, S. 8.
[257] Vgl. Marcolino und Matsubara 2011, S. 23.
[258] Die im Laufe der Monte Carlo Simulationen besuchten Knoten des Spielbaumes dienen einzig dazu, den Lauf des simulierten Spieles voranzubringen und finden keinen Eingang in den zu explorierenden Spielbaum, der dem Verfahren grundlegend ist.

verwendet als Annäherungswert des betrachteten Knotens.[259] Mit dem folgenden Pseudocode sei die Arbeitsweise des UCT-Algorithmus‘ veranschaulicht und verdichtet wiedergegeben:[260]

Algorithmus 4.4 *Monte Carlo Spielbaumsuche mit UCT-Algorithmus.*

- Parameter:
  - *gameState*: Ausgangszustand der Spielwelt.
  - *timeLimit*: Maximale Laufzeit des Verfahrens.
- Rückgabe: Aktion, die zum besten Kindknoten des untersuchten Spielzustandes *gameState* führt.

```
function MCTreeSearchWithUCT(gameState, timeLimit)
    create root node v_0 with state gameState
    while timeLimit > 0 do
      startTime ← Date.now()
      v_l ← TreePolicy(v_0)  // Selektion durch UCT und Expansion
      Δ ← DefaultPolicy(s(v_l))       // Simulation
      Backup(v_l, Δ)     // Rückreichen der Werte an Elternknoten
      // Aktualisierung der Laufzeitbedingung
      timeLimit ← timeLimit - (Date.now() − startTime)
    return a(BestChild(v_0, 0))
function TreePolicy(v)
    while v is nonterminal do
      if v not fully expanded then
          return Expand(v)
      else
          v ← BestChild(v, C_P) // Z.B. mit C_P = 1/√2
      return v
```

[259] Zur Erläuterung des Verfahrens vgl. auch Marcolino und Matsubara 2011, S. 23.
[260] Der Pseudocode des *UCT*-Algorithmus‘ ist wiedergegeben nach Browne et al. 2012, S. 9.

```
function Expand(v)
    choose a ∈ untried actions from A(s(v))
    add a new child v' to v
        with s(v') = f(s(v), a)
        and a(v') = a
    return v'
function BestChild(v, c)
    return argmax_{v' ∈ children of v} Q(v')/N(v') + c√(2 ln N(v)/N(v'))
function DefaultPolicy(s)
    while s is non-terminal do
        choose a ∈ A(s) uniformly at random
        s ← f(s, a)
    return reward for state s
function Backup(v, Δ)
    while v is not null do
        N(v) ← N(v) + 1
        Q(v) ← Q(v) + Δ   // Δ: Wert des terminalen Knotens
                              aus sicht des Knotenspielers
        Δ ← −Δ            // Rückpropagation nach Negamax-Logik
        v ← parent of v
```

Der UCT-Algorithmus verfügt über die Eigenschaft, mit steigender Anzahl von Simulationen gegen den besten Zug zu konvergieren.[261] Ergänzt durch die Berücksichtigung spielspezifischen Wissens, habe UCT verschiedene und zahlreiche Ansätze zur Implementation künstlicher Intelligenz in Spielen wie dem komplexen *Spiel der Amazonen*,[262] dem traditionellen Brettspiel Go, diversen

[261] Vgl. Kocsis und Szepesvári 2006, S. 289.
[262] Über die erfolgreiche Anwendung und den Erfolg von *UCT* im *Amazons*-Spiel informiert Lorentz 2008.

Kartenspielen sowie Echtzeit-Strategiespielen übertroffen, wie Gelly und Silver anmerken.[263] Den Erfolg des UCT-Algorithmus in unterschiedlichen Spieldomänen sieht Mańdziuk besonders in drei Eigenschaften des Verfahrens begründet:[264] Zum einen lässt sich die Laufzeit des UCT Algorithmus zu jedem Zeitpunkt beenden unter Sicherung verwertbarer Ergebnisse – bei der Alpha-Beta Kürzung ist dies nicht so einfach möglich. Zum anderen generiert das UCT Verfahren einen inkrementell wachsenden Spielbaum, bei dem vielversprechende Äste und Zweige des Baumes tiefer aufgesucht werden, als bei der Alpha-Beta Kürzung, wie in der folgenden Grafik dargestellt:

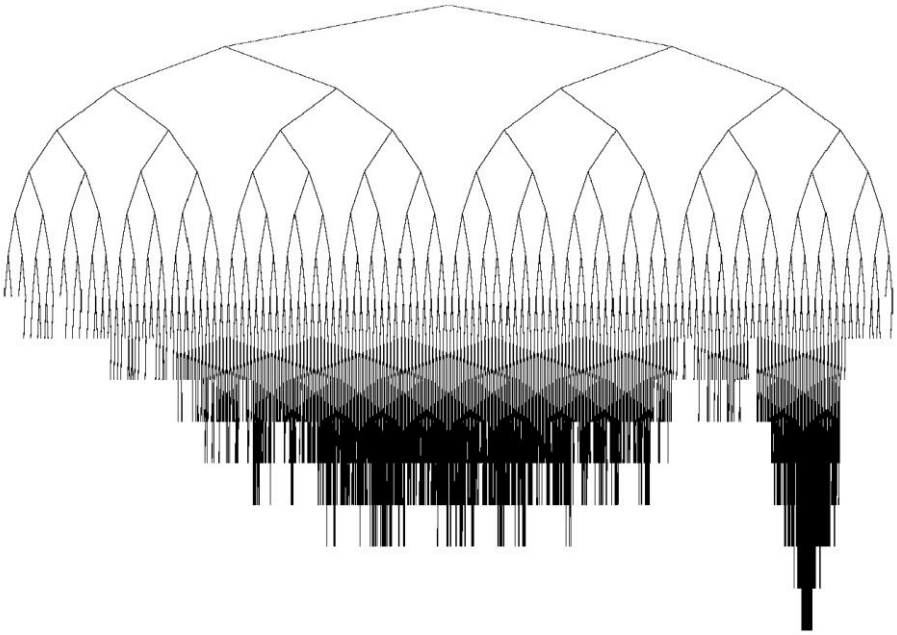


Abbildung 28: Beispielgrafik eines durch das Verfahren der Monte Carlo Spielbaumsuche generierten asymmetrischen Spielbaumes. Quelle: Coquelin und Munos 2007, S. 74.

Ungewissheit über die Qualität eines möglichen Zuges behandelt UCT sinnvoll, weil sich die Gewichtung des Vaterknotens als Auszahlungssummen der Kindknoten darstellt, die durch die Anzahl der Iterationen des Verfahrens beeinflusst sind. So bevorzugt UCT einen Kindknoten gegenüber seinem Geschwisterknoten, wenn der Kindknoten durch eine deutlich höhere Auszahlungssumme gekennzeichnet und der Knoten im Rahmen des Verfahrens häufig aufgesucht worden ist.

[263] Vgl. Gelly und Silver 2011, S. 1856.
[264] Vgl. Mańdziuk 2010, S. 39.

4.2.4 AMAF und RAVE

Die Monte Carlo Spielbaumsuche und der Selektionsalgorithmus UCT nähern den Wert jedes Spielzustandes s und jeder Aktion a im Spielbaum einzeln an und erfordern somit zahlreiche Monte Carlo Simulationen für die Bewertung jedes individuellen Zustandes und für jede Aktion.[265] Häufig jedoch bestimmt sich der Wert eines möglichen Zuges in rundenbasierten Spielen wie Go unabhängig von Zügen, die an anderen Positionen des Spielbrettes ausgeführt werden und ist der Nutzen und der Wert einer Aktion nicht unmittelbar an den aktuellen Zustand der Spielwelt gebunden. [266] Die 1993 von Brügmann [267] für das Go-Spiel vorgeschlagene *All Moves as First* Heuristik (*AMAF*) intendiert, das simulierte Spielverhalten zu beschleunigen, indem sie nicht einzig den Wert des aktuell betrachteten Knotens bestimmt, sondern zusätzlich Geschwisterknoten von Knoten des Spielbaumes aktualisiert, die im Laufe des simulierten Spieles besucht werden.[268] Die grundlegende Idee von AMAF besteht darin, einen Wert für jeden Zug zu bestimmen, unabhängig davon, zu welchen Zeitpunkt und zu welchem Spielweltzustand der Zug ausgeführt wird und vollzieht sich nach dem folgenden Procedere:[269] Für jedes im Rahmen des Monte Carlo Verfahrens simulierte Spiel S_i, das ausgehend von dem Spielzustand p mit dem Zug m weitergeführt wird, bestimmt das AMAF-Verfahren $S_i(p,m) = 1$, wenn der Spieler das simulierte Spiel gewonnen hat und vermerkt im anderen Falle $S_i(p,m) = 0$, wenn der Spieler, der den Zug m ausführte, das Spiel verlor. Der AMAF-Wert für den betrachteten Knoten bestimmt sich abschließend als Durchschnittswert der im Laufe des Verfahrens ermittelten Werte.

Ihre Verknüpfung mit dem UCT Algorithmus erfährt die AMAF-Heuristik mit der *RAVE*-Methode, wie sie Gelly und Silver 2007 vorschlagen.[270] Abgekürzt für *Rapid Action Value Estimation*, berechnet und verwendet der RAVE-Algorithmus nicht den Monte Carlo- oder UCT-Wert jedes Knotens des Spielbaumes, sondern den

[265] Vgl. Gelly et al. 2012, S. 110.
[266] Vgl. Gelly et al. 2012, S. 110.
[267] Brügmann 1993.
[268] Für einen Überblick über Variationen der *All Moves as First* Heuristik sei verwiesen auf Helmbold und Parker-Wood 2009.
[269] Die Beschreibung des *AMAF* Verfahrens ist wiedergegeben nach Chaslot 2010, S. 51.
[270] Gelly und Silver 2007.

entsprechenden AMAF Wert. Welcher Kindknoten k ausgewählt wird, ist dabei bestimmt durch den folgenden Term:[271]

$$k \in argmax_{i \in I}\left(\left(1-\beta(n_p)\right) \times \left(v_i + C\sqrt{\frac{\ln(n_p)}{n_i}}\right) + \beta(n_p) \times AMAF_{p,i}\right) \quad (4.5)$$

Wie im vorangegangen Kapitel besprochen, stellt der Multiplikand $v_i + C\sqrt{\frac{\ln(n_p)}{n_i}}$ den UCT-Teil des Ausdrucks dar. Für den gewichtenden Koeffizienten β schlagen Gelly und Silver $\beta(N) = \sqrt{\frac{k}{3N+k}}$ vor,[272] bei dem es möglich ist, den Wert k auf die Anzahl der Simulationen zu setzen, in denen $\beta = \frac{1}{2}$ sein soll. Die Autoren weisen in ihrer Besprechung des Verfahrens darauf hin, dass sich in der Praxis der Wert $k = 1000$ als besonders effektiv herausgestellt habe.[273] Anhand der folgenden Darstellung sei die Verfahrensweise des RAVE-Algorithmus‘ abschließend verdeutlicht:

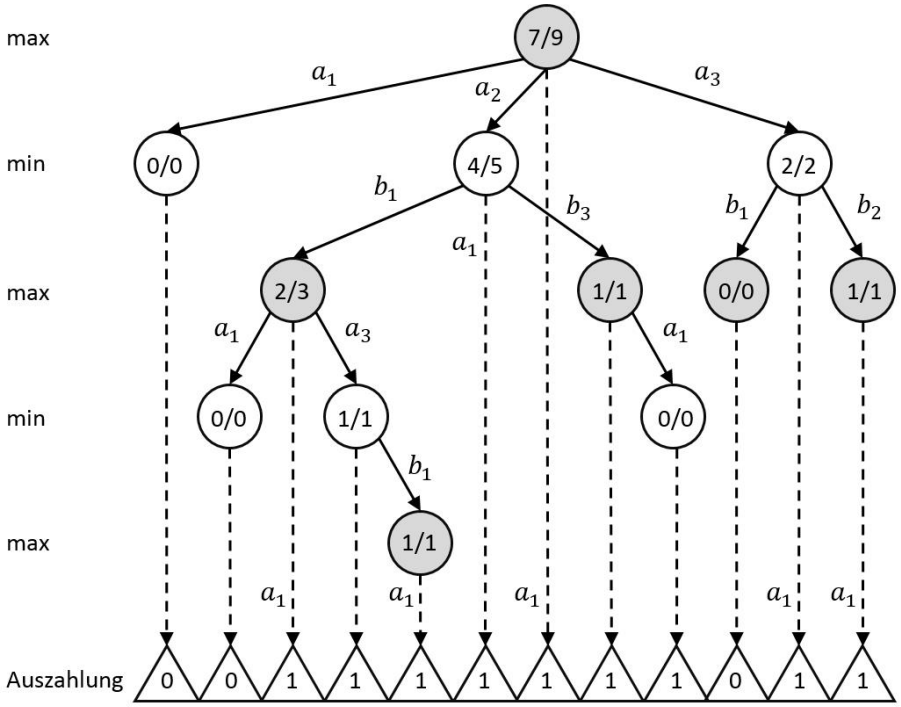


Abbildung 29: Anwendungsbeispiel des RAVE-Algorithmus'. Quelle: Gelly et al. 2012, S. 111.

[271] Vgl. Chaslot 2010, S. 51.
[272] Vgl. Gelly und Silver 2007, S. 278.
[273] Vgl. Gelly und Silver 2007, S. 278.

Wurde im einleitend wiedergegebenen Spielbaum in Abbildung 26 einzig eine Simulation für den Zug a_1 ausgeführt, die in dem Knotenwert 0/1 resultierte, werden mit dem RAVE-Algorithmus neun Simulationen ausgeführt, in denen der maximierende Spieler zu einem beliebigen Zeitpunkt des jeweils simulierten Spieles die Aktion a_1 ausführt. Die einzelnen Knotenwerte werden des Verfahrens entsprechend aktualisiert.

4.2.5 Laufzeitoptimierung: Parallelisierung und Multithreading

In der Beschleunigung der Monte Carlo Spielbaumsuche bieten sich ob der Flexibilität des Verfahrens einige Anknüpfungspunkte, von denen im Folgenden die wohl naheliegendste, die Parallelisierung der Methode besprochen wird.[274] Unterscheiden lassen sich mit Chaslot et al. [275] drei Haupttypen der Parallelisierung, in denen jeweils unterschiedliche Phasen der Monte Carlo Spielbaumsuche auf mehrere Threads[276] verteilt werden: die Blattparallelisierung (*leaf parallelization*), die Wurzelparallelisierung (*root parallelization*) sowie die Baumparallelisierung (*tree parallelization*).

Benötigt die Baumparallelisierung den Autoren zufolge Verfahren, um die individuell arbeitenden Threads davon abzuhalten, ihren Arbeitsgegenstand wechselseitig zu beeinflussen, so gestalten sich die Methoden der Blatt- und Wurzelparallelisierung als einfache, jedoch äußerst dienliche Verfahren zur Parallelisierung der Monte Carlo Spielbaumsuche, weil sie sich schnell und ohne ein Gros an Verwaltungsmechanismen implementieren lassen. Wie in der folgenden Grafik veranschaulicht, werden bei der Blattparallelisierung von jedem selektierten Knoten eine Anzahl von n zu simulierender Spiele auf n Threads verteilt und anschließend simuliert.

[274] Essentiell ist der Performanceverbesserung nicht einzig die Verteilung auf mehrere Prozesse bzw. Threads, sondern auch die Feinjustierung der Selektions- und die Verkürzung der Simulationsphase, z.B. durch Berücksichtigung von Spielspezifika. Auf eigene Optimierungen der Verfahren sei an späterer Stelle eingegangen, wenn die dem Framework *SpoookyJS* zugrunde liegenden Mechanismen erläutert werden.
[275] Chaslot et al. 2008c.
[276] Als Bestandteil des übergeordneten Prozesses des Monte Carlo Verfahrens ist ein *Thread* ein leichtgewichtiger Prozess, der im Adressraum des Prozesses arbeitet.

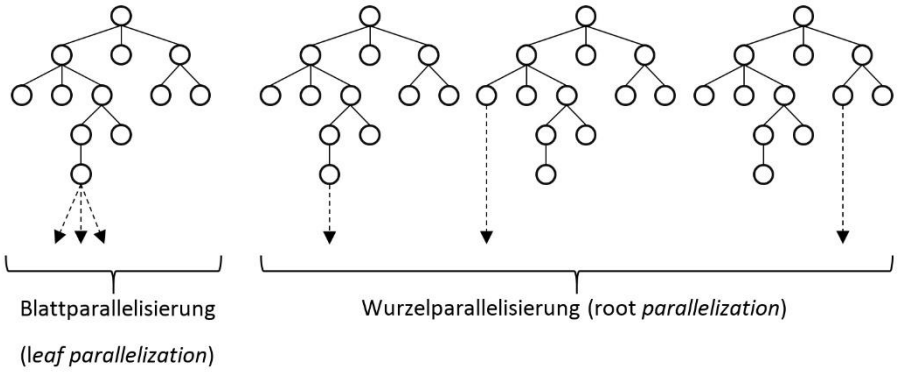


Abbildung 30: Zwei Verfahren der Parallelisierung der Monte Carlo Spielbaumsuche. Die gestrichelten Linien mit Pfeilabschluss signifizieren individuelle Threads für die Simulationsphase des Verfahrens. Quelle: Chaslot et al. 2008c, S. 63.

Zwar bietet die Blattparallelisierung den Vorteil, sich besonders einfach umsetzen zu lassen, die Parallelisierungsmethode hat jedoch den Nachteil, dass der Hauptprozess auf das längste simulierte Spiel warten muss, um seine Arbeit fortzusetzen – die Zeit, die für ein simuliertes Spiel benötigt wird, ist unvorhersehbar.[277] Das alternative Verfahren der Wurzelparallelisierung, von Cazenave und Jouandeau zuerst als *Single-Run Parallelization* vorgestellt,[278] besteht darin, mehrere UCT-Algorithmen parallel laufen zu lassen ohne Kommunikationsmöglichkeit der Threads. Hierbei generiert jeder Thread im Rahmen des Verfahrens einen anderen Spielbaum, z.B. durch individuelle Anpassung der Konstante C. Ist eine maximale Anzahl von Spielen simuliert oder ist die zur Verfügung stehende Rechenzeit erschöpft, sammelt der Hauptprozess die Ergebnisse der Threads ein und vereinigt die Resultate der leichtgewichtigen Prozesse.[279]

In der vorliegenden Arbeit wurde das Verfahren der Wurzelparallelisierung präferiert und implementiert, weil es sich ressourcensparend umsetzen lässt und sich in der Praxis als äußerst leistungsfähiges Verfahren bewährt hat.[280] Auf Details in der Implementation sei an späterer Stelle noch ausführlich eingegangen, wenn das Framework *SpoookyJS* vorgestellt wird. Den Abschluss des gegenwärtigen

[277] Vgl. Chaslot et al. 2008c, S. 63.
[278] Cazenave und Jouandeau 2007, S. 95f.
[279] Vgl. Cazenave und Jouandeau 2007, S. 95f. und Chaslot et al. 2008c, S. 63.
[280] Vgl. u.a. Fern und Lewis 2011, S. 3 Chaslot et al. 2008c, S. 67.

Kapitels bildet eine kurze Betrachtung von Monte Carlo Verfahren in digitalen Brettspielen.

4.2.6 Monte Carlo Verfahren und digitale Brettspiele

Aufgrund seiner enormen Komplexität[281] stellt das Go-Spiel artifizielle Gegner vor eine besondere Herausforderung, die sich mit Allis in der Erkennung und Tradierung von Spielmustern formuliert, wie sie menschliche Spielerinnen und Spieler leisten:

> „The explanation for the low playing strength of current go programs is found in the nature of the game. While the potential branching factor averages 250, human players only consider a small number of these, through extensive knowledge of patterns relevant to go. Similarly, while evaluating a position, humans determine the strengths and weaknesses of groups on the board with pattern knowledge. Thus, either programs must obtain pattern knowledge similar to human experts, or compensate for a lack of such knowledge through search or other means.“[282]

Gespielt auf einem 19×19 Felder großen Spielbrett, legen die Spieler im Go-Spiel ihre schwarzen und weißen Spielsteine nacheinander, Spielrunde für Spielrunde auf den Schnittpunkten der Spielfeldlinien ab. Beide Spieler intendieren, gegnerische Spielsteine einzuschließen, um sie anschließend vom Spielbrett zu entfernen. Jeder eingeschlossene gegnerische Spielstein zählt einen Punkt. Das Spiel gewinnt, wer die meisten Punkte erzielt.[283] Die folgende Fotografie zeigt einen Spielzustand des Go-Spieles:

[281] Vgl. Allis 1994, S. 174f.
[282] Allis 1994, S. 174.
[283] Vgl. Schädler 2007, S. 52 und Dickfeld 2003, S. 12.

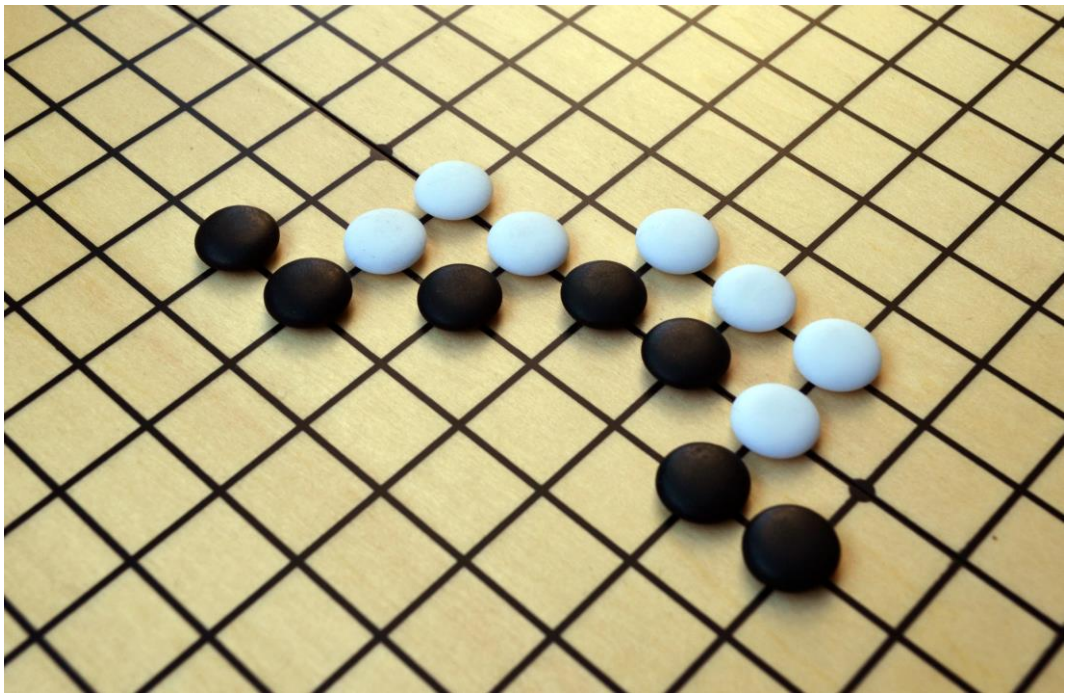


Abbildung 31: Spielzustand des Go-Spieles. Eigene Aufnahme.

Klassische Ansätze in der Implementation künstlicher Intelligenz im Go-Spiel lassen sich mit Marcolino in drei aufeinander folgende Phasen einordnen.[284] In der ersten Phase, die sich mit der 1969 publizierten Arbeit von Zobrist manifestiert,[285] schlug Albert Zobrist, der Erfinder des ersten Go-spielenden Computerprogrammes,[286] vor, das Spielbrett in schwarze und weiße Bereiche einzuteilen und die Relevanz der einzelnen und im Laufe des Spieles miteinander verknüpften Spielsteine mittels Einflussfunktionen (*influence functions*) zu bestimmen. Die zweite Generation Go-spielender Software nutzte abstrakte Repräsentationen des Spielbrettes[287] – und die dritte Phase in der Entwicklung zog aus beobachteten Spielmustern ihre Schlüsse für die Spielentscheidungen.

Alle Ansätze waren der Entwicklung schlau spielender Go-Software dienlich, vermochten es jedoch nicht, starke Computergegner zu implementieren.[288] Erst die in den vergangenen Kapiteln betrachteten Monte Carlo Ansätze und der *Upper*

[284] Vgl. das gleichnamige Kapitel in Marcolino 2011, S. 8f. Die Geschichte der Entwicklung artifizieller Computergegner im Go-Spiel findet sich ausführlich betrachtet in Cai und Wunsch, DonaldC., II 2007.
[285] Vgl. Zobrist 1969.
[286] Vgl. Cai und Wunsch, DonaldC., II 2007, S. 448.
[287] Vgl. Marcolino 2011, S. 8.
[288] Vgl. Marcolino 2011, S. 9.

Confidence Bounds applied to Trees Algorithmus mit seinen Erweiterungen führten das artifizielle Spiel zu meisterhaftem Spielverhalten.[289] Programme wie *Fuego*[290] oder *MoGo*,[291] die mit Monte Carlo Verfahren arbeiten, spielen äußerst leistungsstark im Go-Spiel und ließen sich Marcolino und Matsubara[292] zufolge optimieren zum einen durch immere leistungsfähigere Computerhardware und die Parallelisierung der Monte Carlo Simulationen, zum anderen jedoch durch Verbesserung der zugrundeliegenden Algorithmen und durch ein Weiterdenken der umgesetzten Konzepte. So schlagen die Autoren ein evolutionär verfahrendes Multiagentensystem vor, das die Entscheidungsfindung differierender singulärer Agenten berücksichtigt, um das Spielverhalten des artifiziellen Computergegners zu verbessern. An die Stelle eines isolierten Agenten, der in der Simulationsphase des Monte Carlo Verfahrens die Wertigkeit eines möglichen Spielzuges auf Basis von Ausspielungen verschiedener Zugmöglichkeiten berechnet, setzen Marcolino und Matsubara mehrere Agenten, die über individuell gewichtete Heuristiken und Spielstrategien verfügen und sich in der Simulationsphase abwechseln, um die verfügbaren Heuristiken zu variieren. [293] Die Eigenschaft des Multiagentenansatzes, die Spielstärke ihres artifiziellen Go-Spielers durch das Zusammen- und Gegenspiel einfacher unterschiedlicher Agenten zu erhöhen, bezeichnen die Autoren als ein *emergentes* Phänomen,[294] das die bereits äußerst erfolgreich Go-spielende Software *Fuego* erweitert. Wie zuvor häufig angeklungen, verdeutlicht sich jedoch auch bei Marcolino und Matsubara die Abhängigkeit der implementierten Verfahren von leistungsfähiger Hardware: Um

[289] Vgl. die Übersicht über das Aufkommen und den Erfolg der Monte Carlo Spielbaumsuche in Browne et al. 2012, S. 8.

[290] Das quelloffene *Fuego* (http://fuego.sourceforge.net, zuletzt aufgerufen am 02.04.2014) ist geschrieben in der objektorientierten Programmiersprache *C++* und wird 2010 von den Autoren des Programmes beschrieben als zweitstärkstes Go-Programm auf einem Spielbrett mit 19×19 Spielfeldern (vgl. Enzenberger und Müller 2010, S. 15 sowie den Aufsatz Segal 2011).

[291] Vgl. u.a. die Aufsätze Gelly et al. 2006 und Wang und Gelly 2007. Im Jahr 2008 schlug *MoGo* (https://www.lri.fr/~teytaud/mogo.html, zuletzt aufgerufen am 02.04.2014) den Meister (5. *dan*) Cătălin Ţăranu im Go-Spiel mit 9×9 Feldern. Erzielt wurde die starke Spielleistung der Software durch massiven Hardwareeinsatz.

[292] Marcolino und Matsubara 2011.

[293] Marcolino und Matsubara arbeiten mit der quelloffenen Software *Fuego*, die zum Zeitpunkt der Veröffentlichung des Aufsatzes vorrangig die fünf Heuristiken *Nakade*, *Atari Capture*, *Atari Defend*, *Lowlib* sowie *Pattern* vorsieht (Marcolino und Matsubara 2011, S. 23f.).

[294] Marcolino und Matsubara verstehen den Begriff der *Emergenz* wie im Folgenden wiedergegeben: „we define emergence as a great number of simple iterations that occur in a system, leading to a complex result“ (Marcolino und Matsubara 2011, S. 22).

aussagekräftige Ergebnisse zu generieren, lief die Multiagentenergänzung *Fuegos* auf zahlreichen Rechnern eines japanischen Clusters anderthalb Wochen.[295]

Hat sich die zukünftige Forschung um Monte Carlo Verfahren im Go-Spiel Rimmel et al. zufolge u.a. mit Problemen in Eröffnungsbibliotheken oder dem allzu aggressiven Spielverhalten der artifiziellen Spieler zu beschäftigen,[296] geriert sich die Implementation schlauer Gegner auf Grundlage von Monte Carlo Verfahren im Schach- und Damespiel äußerst zögerlich: Wie Browne et al. berichten, existiere keine Implementation des Monte Carlo Spielbaumsuchverfahrens, die es mit menschlicher Spielstärke im Dame- oder Schachspiel aufzunehmen vermag.[297] Das schlechte Abschneiden schachspielender Monte Carlo Implementationen sehen Ramanujan et al. begründet in Schachfallen (*trap states*), die bei einem falschen Zug innerhalb weniger Züge zu einer Niederlage führen.[298] Jene *trap states*, die dem Schachspiel zu eigen sind, ließen sich mit einem iterativ vertiefenden Minimax-Verfahren leicht identifizieren und umgehen, stellten für die Monte Carlo Spielbaumsuche mit UCT eine allzu große Hürde dar, wie die Autoren berichten.[299]

Sieht sich die einstige *Drosophila* künstlicher Intelligenz im Kontext der Monte Carlo Methoden ihrer Flügel gestutzt, profitiert das *Spiel der Amazonen* (*Amazons*) von den besprochenen Monte Carlo Verfahren zur Implementation spielstarker Computergegner. So nehmen sich Kloetzer et al. der hohen Komplexität des auf einem 10×10 Spielfelder großen Spielbrett gespielten Spieles an mit ihrem Computerprogramm *Campya*[300] und finden sich die Verfahren ergänzt um Ansätze zur Autogenerierung von Eröffnungsdatenbanken mit dem als *Meta-MCTS*[301] betitelten Verfahren oder der Analyse von Endspielen.[302]

[295] Vgl. Marcolino und Matsubara 2011, S. 27.
[296] Vgl. Rimmel et al. 2010, S. 234–237 sowie Browne et al. 2012, S. 29.
[297] Vgl. Browne et al. 2012, S. 32.
[298] Vgl. Ramanujan et al. 2012, S. 4.
[299] Vgl. Browne et al. 2012, S. 10.
[300] Vgl. Kloetzer et al. 2007.
[301] Vgl. Kloetzer 2011.
[302] Vgl. Kloetzer et al. 2008.

Zwar sei bereits an dieser Stelle eine Paradigmenwechsel in der Implementation spielstarker Computergegner im Backgammonspiel angemerkt, der sich ob der äußerst erfolgreichen Anwendung von Methoden des verstärkenden Lernens in den 1990er Jahren vollzieht – wie van Lishout et al.[303] mit ihrer Software *McGammon* demonstrieren, lassen sich jedoch auch Monte Carlo Verfahren erfolgreich in Spielen mit Zufallsmomenten einsetzen: Als *McGammon* im Jahre 2007 an der Computer Olympiade in Amsterdam teilnahm, vermochte es, 6.500 Spiele in der Sekunde durchzuführen. Die erste Zugentscheidung der Software, die auf einem mit 2,6 GHz getakteten *Quad-Operon* System lief, basierte auf 200.000 Monte Carlo Simulationen.[304] Wenngleich *McGammon* mit einem einfachen Monte Carlo Verfahren arbeitete, war das Programm dennoch in der Lage, Expertenzüge zu identifizieren und auszuführen.[305]

[303] van Lishout et al. 2007.
[304] Vgl. van Lishout et al. 2007, S. 182.
[305] Vgl. van Lishout et al. 2007, S. 183.

4.2.7 Zusammenfassung und Ausblick

Künstliche Intelligenz in digitalen Brettspielen präsentierte sich in den vorangegangenen Kapiteln als die Suche nach optimalen Spielentscheidungen im Spielbaum. Wurden mit dem Minimax-Algorithmus und der Alpha-Beta Kürzung zunächst klassische Verfahren zur Suche nach optimalen Entscheidungen in Zweispieler-Nullsummenspielen mit perfekter Information vorgestellt, offenbarte sich die leicht parallelisierbare Monte Carlo Spielbaumsuche mit dem UCT-Selektionsalgorithmus als leistungsfähiges Werkzeug in differierenden Spieldomänen.

Wie geschickt sich die betrachteten Verfahren auch in ihrem Problemlösungsverhalten offenbaren, so ist Ihnen eines gemein: Alle referierten Methoden profitieren sehr stark von der Berücksichtigung und Einbettung Expertenwissens in die Entscheidungsfindung. Wie sich jenes Wissen um sinnvolle einleitende Spielzüge oder die Bevorzugung bestimmter Zugmöglichkeiten gegenüber anderen Zugmöglichkeiten ohne menschliche Expertise autogenerieren lässt, davon wird in den nachfolgenden Kapiteln berichtet, die sich zuerst mit künstlichen neuronalen Netzen beschäftigen und anschließend das Verfahren des verstärkenden Lernens vorstellen, bei dem die spielenden Agenten aus den Erfahrungen mit und in ihrer Agentenumwelt zu lernen vermögen.

4.3 Computational Intelligence und digitale Brettspiele

> "Many people view learning as the core property of intelligence. That learning per se does not make people intelligent, but the capacity to learn, is also a popular view. So learning to learn appears to be the key point."[306]

Lernen und Gedächtnis, Informationsspeicher und Informationsverknüpfung. Aus Erfahrungen zu lernen, das Lernen gar zuallererst zu erlernen, offenbart sich der Reflektion über menschliche und artifizielle Intelligenz als ein Schlüsselkonzept in der Annäherung an den Begriff der Intelligenz.[307] Die folgenden Kapitel beschäftigen sich mit ebensolchen Problemlösungsverfahren, die in dem Vermögen gründen, durch Interaktion mit der Umwelt generierte Erfahrungen auszuwerten, zu memorieren und für spätere Entscheidungsfindungsprozesse zu nutzen. Wird zunächst mit künstlichen neuronalen Netzen ein klassisches Feld der *Computational Intelligence* referiert, das sich durch die komplexitätsreduzierte Ab- und Nachbildung biologischer Gehirne charakterisiert, werden daran anschließend Lernverfahren vorgestellt, in denen schlaues Agieren durch Interaktion mit der Umwelt nach und nach, Erfahrungsepisode für Erfahrungsepisode gelernt wird. Allen betrachteten Ansätzen ist dabei gemein, dass die Eigenschaften des zu lösenden Problems nicht im Vorhinein möglichst vollständig und erschöpfend dem problemlösenden System eingepflanzt werden müssen, um effiziente Entscheidungsfindung in bekannten oder unbekannten Aufgabenumgebungen zu ermöglichen.

Die folgenden Kapitel treffen in ihren Ausführungen eine Auswahl aus dem Methodenkomplex der Computational Intelligence zur Implementation spielstarker Gegner in digitalen Brettspielen. Auf evolutionäre Mechanismen, anhand derer besonders erfolgreiche Agentenkollektive und Agenteneigenschaften tradiert werden, sei an späterer Stelle eingegangen, wenn

[306] Pfeifer und Scheier 2001, S. 9.
[307] Vgl. Pfeifer und Scheier 2001, S. 7–12.

die dem implementierten Framework *SpoookyJS* zugrundeliegenden Mechanismen besprochen werden.

4.3.1 Künstliche neuronale Netze

In der Verarbeitung sinnlicher Wahrnehmungsdaten offenbart sich das menschliche Gehirn als äußerst leistungfähig und fehlertolerant. So generiert das Dnkorgn noh imr Bedetng aus dism Satz, wenngleich einige Buchstaben fehln.[308] Die außerordentlichen Fähigkeiten des menschlichen Gehirns – u.a. in der Mustererkennung, dem Lernen aus empirischen Daten, dem Verknüpfen von Sachverhalten und dem Gedächtnis, das sich als „außergewöhnlichste[s] Phänomen in der Natur“[309] signifizieren lässt – gründet im feingliedrigen Aufbau des massiv vernetzten und parallel arbeitenden Organs. Zwar ist noch immer nicht vollständig ergründet, wie es funktioniert, jenes „komplizierteste System im Universum“[310], die konstituierenden Elemente des Gehirns jedoch sind mit den *Neuronen* benannt. Wie Kruse et al. referieren, verfügt das menschliche Gehirn über rund 100 Milliarden Neuronen, die mittels ihrer Kontaktpunkte, den *Synapsen*, jeweils mit zahlreichen, vielmehr unzähligen anderen Neuronen verbunden sind.[311] Aufgrund seiner feingliedrigen neuronalen Architektur ist es dem biologischen Gehirn möglich, flexibel auf unterschiedliche Anforderungen einzugehen und stellt das Organ „ein massiv parallel-vernetztes System dar, das über die Speicherung von Informationen hinaus auch gleichzeitig deren Verarbeitung bewerkstelligt.“[312]

Künstliche neuronale Netze, auch als *konnektionistische Modelle* bezeichnet und im Folgenden mit KNN abgekürzt, imitieren ihr biologisches Vorbild in deutlich kleinerem Maßstab und dienen vielfältigen Einsatzzwecken wie der Klassifikation von Eingabedaten, der Erkennung von Mustern – oder der Implementation schlauer artifizieller Computergegner in digitalen Brettspielen. Wie Pfeifer und

[308] Vgl. Favre-Bulle 2000, S. 62f.
[309] Thompson 2010, S. 359.
[310] Roth 2009, S. 13.
[311] Vgl. Kruse et al. 2011, S. 9–11. Eine kleine Hirnkunde bietet das gleichnamige Kapitel in Roth 2009, S. 13–32, eine große Hirnkunde Thompson 2010.
[312] Favre-Bulle 2000, S. 63.

Scheier darlegen, sind bei der Umsetzung eines KNN vier grundlegende Charakteristika des Netzes zu besprechen und zu definieren:[313]

1. Die Eigenschaften des individuellen Neurons: Zu definieren ist, wie Eingabeimpulse verarbeitet werden, unter welcher Bedingung das Neuron *feuert*, d.h. Impulse an verknüpfte Einheiten sendet und wie die Ausgabe des Neurons beschaffen ist.
2. Der Grad der Verknüpftheit: Beschrieben werden muss die Topologie des Netzes, d.h. wie die Neuronen angeordnet und miteinander verbunden sind – und sich somit gegenseitig beeinflussen.
3. Die Ausbreitungsregel (*propagation rule*): Bestimmt werden muss, wie sich die Signalweiterleitung zwischen den einzelnen Einheiten des Netzes gestaltet.
4. Die Lernregel des Netzes: Zu bestimmen ist, ob und wie sich die Verknüpfungsgewichtungen im Laufe der Zeit ändern.

Die folgenden Unterkapitel vertiefen den Überblick über jene vier Netzcharakteristika und schließen mit der Betrachtung des mehrschichtigen *Perzeptrons*, das in der Umsetzung künstlicher Intelligenz in computerbasierten Brettspielen seine Anwendung findet.

4.3.1.1 Aufbau und Aktivierungsfunktion eines Neurons

Künstliche neuronale Netze bestehen aus einzelnen Neuronen, auch als *Knoten* bezeichnet, die über vordefinierte Strategien miteinander verknüpft sind. Jedes Neuron nimmt einen oder mehrere Eingabeimpulse entgegen und sendet in Abhängigkeit von den Eingabereizen und der Aktivierungsfunktion des Knotens Impulse an verknüpfte Einheiten. Jeder Ein- und Ausgabeverknüpfung eines Knotens ist eine Gewichtung zugeordnet, welche die Stärke der Verbindung charakterisiert.Das Modell eines basalen künstlichen Neurons sei mit der folgenden Abbildung visualisiert:[314]

[313] Vgl. das Kapitel *The Four or Five Basics* in Pfeifer und Scheier 2001, S. 143–147. Die Autoren führen die Einbettung des Netzes in einen Agenten als Fünfte zu klärende Eigenschaft des KNN an.
[314] Die Darstellung des Neurons ist wiedergegeben nach Pfeifer und Scheier 2001, S. 147.

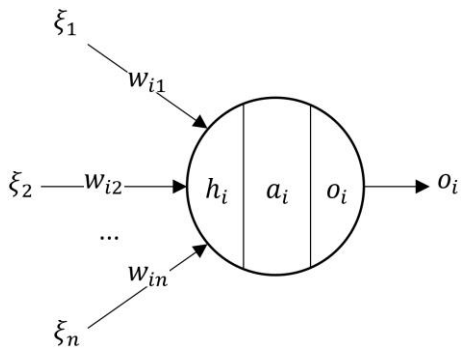


Abbildung 32: Einfaches Modell eines künstlichen Neurons i.

In der Grafik bezeichnet h_i die Summe der gewichteten Eingabeimpulse $w_{i1} \dots w_{in}$, die das Neuron von verknüpften externen Einheiten oder von Sensoren ($\xi_1 \dots \xi_n$) erhält. Das Aktivierungsmaß des Neurons i ist mit a_i ausgedrückt und wird von der Aktivierungsfunktion $g(h_i)$ bestimmt. Die Ausgabe von i ist signifiziert mit o_i.

Die Verarbeitung der Eingabeimpulse $w_{i1} \dots w_{in}$ erfolgt in zwei Schritten: Zuerst wird mit h_i die gewichtete Summe der Eingabeimpulse bestimmt, die an dem Neuron anliegen:

$$h_i = \sum_{j=1}^{n} w_{ij} o_j \tag{4.6}$$

Hierbei signifiziert $w_{ij} o_j$ die Ausgabe o_j des Knotens j, die das betrachtete Neuron i durch eine gewichtete Verbindung w_{ij} erreicht. Ist h_i berechnet, wird anschließend die Aktivierungsfunktion $g(h_i)$ ausgeführt, die das Aktivierungsmaß a_i des Neurons i kalkuliert:

$$a_i = g(h_i) = g\left(\sum_{j=1}^{n} w_{ij} o_j\right) \tag{4.7}$$

Wie Pfeifer und Scheier berichten, wird das Aktivierungsmaß zumeist als Ausgabe des Neurons verwendet (d.h. es gilt $o_i = f(a_i) = a_i$) und wird das Aktivierungsmaß mit jedem Zeitschritt t aktualisiert, so dass sich der Term wie folgend umformen lässt:[315]

[315] Vgl. Pfeifer und Scheier 2001, S. 147.

$$a_i(t+1) = g\left(\sum_{j=1}^{n} w_{ij} a_j(t)\right) \tag{4.8}$$

In der praktischen Umsetzung künstlicher neuronaler Netze gestaltet sich die Aktivierungsfunktion g häufig als Schwellenwertfunktion oder als logistische Funktion. Ein Neuron, das mit einer Schwellenwertfunktion $g_{schwell}$ arbeitet, wird als *Perzeptron* bezeichnet und feuert, wenn h_i einen definierten Schwellenwert Θ erreicht hat:

$$g_{schwell}(h_i) = \begin{cases} 1 \; wenn \; h_i \; \geq \; \Theta \\ 0 \; wenn \; h_i \; < \; \Theta \end{cases} \tag{4.9}$$

Eine weichere Differenzierung im betrachteten Wertebereich zwischen 0 und 1 lässt sich mit einer logistischen Funktion der folgenden Gestalt erreichen:[316]

$$g_{logistisch}(h_i) = \frac{1}{1 + e^{-\frac{h_i - \Theta}{T}}} \tag{4.10}$$

Die logistische Aktivierungsfunktion, auch als *Sigmoid-* oder *Schwanenhalsfunktion* bezeichnet, bietet den Vorteil der Differenzierbarkeit und nähert durch ihr spezifisches Verhalten, zunächst steil anzusteigen und bei starker Erregung gesättigt zu sein, das künstliche Neuron seinem biologischen Vorbild an. Ein Neuron, das mit einer logistischen Aktivierungsfunktion verfährt, wird als *Sigmoid-Perzeptron* bezeichnet. Die beiden referierten Aktivierungsfunktionen veranschaulichen sich mit der folgenden Darstellung:[317]

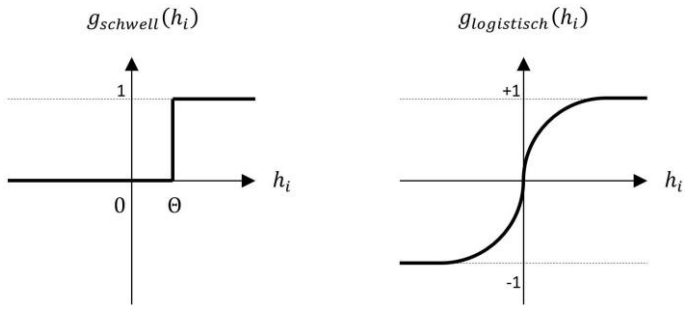


Abbildung 33: Darstellung der Schwellenwertfunktion (links) und Sigmoidfunktion (rechts) mit $T \to 0$.

[316] Vgl. Ertel 2013, S. 252. Das e im Divisor des Terms signifiziert die eulersche Zahl; der Parameter T variiert die Glättung der Sigmoidfunktion.
[317] Die Darstellungen sind wiedergegeben nach Pfeifer und Scheier 2001, S. 148.

4.3.1.2 Netztopologien

Künstliche neuronale Netze implementieren ihre Funktionsweise und ihre individuellen Eigenschaften durch Verknüpfung zahlreicher einzelner Neuronen miteinander. Auf welche Art, auf welche Weise die Neuronen verknüpft sind, ist bestimmt durch unterschiedliche Netztopologien. Wie Kroll anmerkt, besteht ein KNN für gewöhnlich aus drei Schichten:[318] einer Eingabeschicht (*input layer*), die an das Netz angelegte Eingangssignale aufnimmt und weiterleitet an eine oder mehrere versteckte Schichten (*hidden layer*), welche die Netzeingabe transformieren und an die Ausgabeschicht (*output layer*) übergeben. Beispielhaft für eine Netztopologie visualisiert die folgende Darstellung ein *Mehrschichtiges Perzeptron* (*multilayer perceptron*), das sich aus drei Schichten konstituiert: [319]

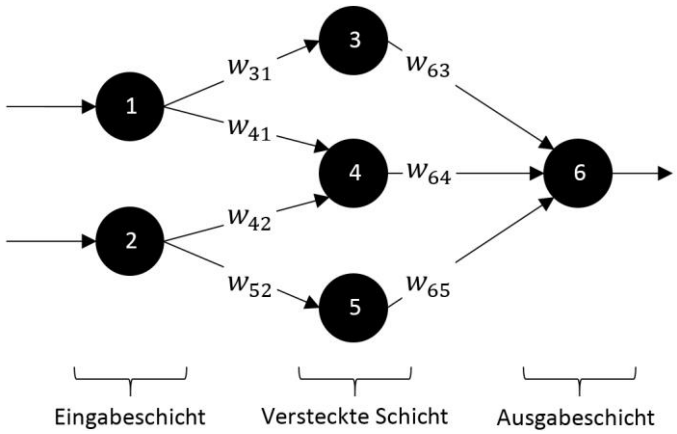


Abbildung 34: Ein vorwärtsgerichtetes künstliches neuronales Netz mit einer Eingabe-, einer versteckten- und einer Ausgabeschicht.

Bei dem dargestellten neuronalen Netz sind alle Neuronen einer Schicht vollständig mit einem oder mehreren Neuronen der nächsten Schicht verknüpft – ob dieser Eigenschaft wird das visualisierte KNN auch als als *vorwärtsgerichtetes* oder *Feedforward-Netz* bezeichnet: Die Neuronen ❶ und ❷ der Eingabeschicht sind verknüpft mit den Neuronen ❸, ❹ sowie ❺ der versteckten mittleren Schicht und die drei Neuronen der versteckten Schicht sind verknüpft mit Neuron ❻ in der Ausgabeschicht. Jeder Neuronenverknüpfung ist eine Gewichtung w_{ij} beigemessen, die bestimmt, wie stark die Neuronen miteinander verbunden sind:

[318] Vgl. Kroll 2013, S. 225.
[319] Die Darstellung des Netzes folgt den Visualisierungen in Pfeifer und Scheier 2001, S. 149 und Russell und Norvig 2012, S. 848.

Eine positive Verbindungsgewichtung signifiziert eine verstärkende (*exzitatorische*) Verbindung, ein negatives Gewicht zwischen den Knoten des Netzes wirkt hemmend (*inhibitorisch*) auf die Signalübertragung und eine 0 stellt keine Verknüpfung der Neuronen dar. Ergänzend zur graphischen Darstellung der Netztopologie lassen sich die Neuronenverknüpfungen in einer Matrix darstellen. Das folgende Beispiel veranschaulicht die Matrixdarstellung mit reellzahligen Beispielgewichtungen für w_{31} bis w_{65}.

$$
\begin{array}{cc}
 & \text{Ausgangsknoten} \\
 & \begin{array}{cccccc} ❶ & ❷ & ❸ & ❹ & ❺ & ❻ \end{array} \\
M = & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0.5 & 0 & 0 & 0 & 0 & 0 \\ 0.7 & \boxed{0.3} & 0 & 0 & 0 & 0 \\ 0 & -0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.9 & \boxed{1.0} & 0.3 & 0 \end{bmatrix} \begin{array}{c} ❶ \\ ❷ \\ ❸ \\ ❹ \\ ❺ \\ ❻ \end{array} \text{ ZielKnoten} \\
 & w_{42} \quad w_{64}
\end{array}
$$

Abbildung 35: Verknüpfungsmatrix eines künstlichen neuronalen Netzes. Zur Veranschaulichung der Matrix sind die Gewichtungen w_{42} und w_{64} hervorgehoben.

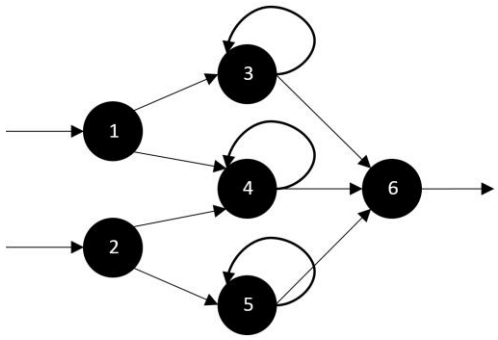
Bei einem vorwärtsgerichteten Netz, wie dem zuvor betrachteten mehrschichtigen Perzeptron, werden die einzelnen Knoten nur in einer Richtung von Schicht zu Schicht des neuronalen Netzes verknüpft – der Ausgabeimpuls des Neurons einer Schicht dient folglich als Eingabeimpuls in der nächsten Schicht. Anders als vorwärtsgerichtete Netze, die eine Funktion der Netzeingabe repräsentieren und über keinen internen Zustand, d.h. kein Gedächtnis verfügen,[320] weisen rückgekoppelte KNN ein dynamisches Verhalten auf und implementieren die Fähigkeit, zu memorieren. Mit Kroll seien die folgenden Rückkopplungsarten differenziert:[321]

a) Bei der direkten Rückkopplung wird die Ausgabe eines Neurons als zeitverzögerter Eingabeimpuls erneut in das Neuron eingespeist.

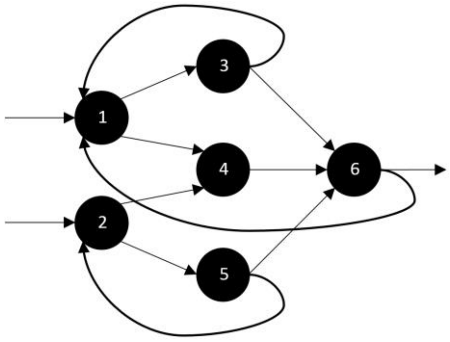
[320] Vgl. Russell und Norvig 2012, S. 847 sowie Kroll 2013, S. 226.
[321] Vgl. Kroll 2013, S. 226f.

b) Wird die Ausgabe eines Neurons als Eingabe in einer früheren Schicht verwendet, so wird die Art der Rückkopplung als indirekt bezeichnet.
c) Lateral ist eine Rückkopplung, wenn Rückkopplungen zwischen Neuronen einer Schicht bestehen.
d) Als extern rückgekoppelte Netze werden neuronale Netze bezeichnet, bei denen die Ausgabe der Ausgabeneuronen (zeitverzögert) in die Eingabeschicht eingekoppelt wird.
e) Sind alle Neuronen miteinander rückgekoppelt, ist das Netz vollständig verbunden.

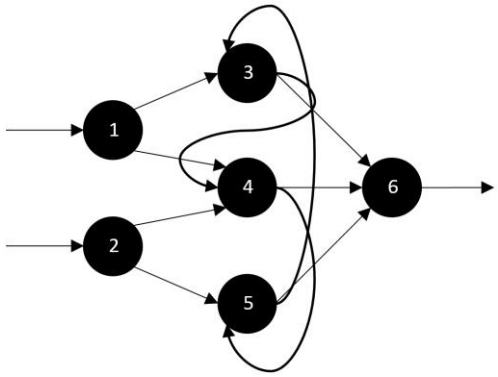
Abschließend seien die verschiedenen Rückkopplungsarten mit den folgenden Darstellungen veranschaulicht:[322]



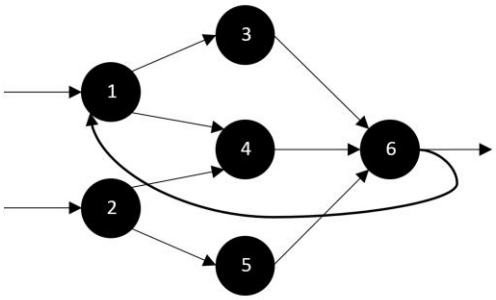
a) Netz mit direkt rückgekoppelten Neuronen in der versteckten Schicht.



b) Neuronales Netz mit indirekten Rückkopplungen.



c) Lateral rückgekoppeltes Netz.



d) Extern rekurrentes Netz.

Abbildung 36: Arten der Rückkopplung in einem Künstlichen neuronalen Netz.

[322] Um die Übersichtlichkeit in der Darstellung zu wahren, wurde von der Visualisierung des Vollständig verbundenen Netzes abgesehen.

4.3.1.3 Ausbreitungsregel

Wie sich die Signalweiterleitung zwischen den Neuronen des Netzes gestaltet, ist mit der Ausbreitungsregel (*propagation rule*, auch als *Propagierungsregel* bezeichnet) formuliert, bei der zumeist die an den Eingängen eines Neurons i anliegenden Ausgabeimpulse o_j verknüpfter Einheiten mit den entsprechenden Verknüpfungsgewichtungen w_{ij} multipliziert werden. Im zuvor betrachteten vorwärtsgerichteten mehrschichtigen Perzeptron gestaltet sich die summierte Eingabe für das Neuron ❻ somit als $h_4 = a_3 w_{63} + a_4 w_{64} + a_5 w_{65}$. Wenngleich die referierte Form der Ausbreitungsregel mit $h_i = \sum_{j=1}^{n} w_{ij} a_j$ in der praktischen Umsetzung künstlicher neuronaler Netze zumeist als gegeben und selbstverständlich angenommen wird, weisen Pfeifer und Scheier darauf hin, dass es sinnvoll sei, die Regel zu explizieren, um die zeitliche Dimension des Ausbreitungsprozesses – z.B. in Form von Verzögerungen – zu berücksichtigen.[323]

4.3.1.4 Lernregeln

Das charakteristische Moment eines künstlichen neuronalen Netzes, sich der Lösung eines gestellten Problems iterativ und adaptiv anzunähern, gründet in den Lernregeln des Netzes, die eine dynamische Anpassung und Feinjustierung der Verbindungsgewichtungen ermöglichen. So gestaltet sich die Arbeit mit einem KNN (zumeist) in zwei Phasen: Zuerst wird das KNN im Lernmodus betrieben, um die Gewichtungen des Netzes auf bestimmte Eingabemuster zu trainieren – wie die Gewichtungen mit jedem Zeitschritt angepasst werden, das ist mit den individuellen Lernregeln des Netzes definiert. Auf die Trainingsphase folgt die Anwendungsphase, in der dem KNN unbekannte, d.h. noch nicht gelernte Eingabemuster zugeführt werden; die Aufgabe des trainierten Netzes besteht nun darin, die unbekannten Eingabemuster einem gelernten Muster zuzuordnen.

Eine basale Lernregel für natürliche Neuronen schlägt Hebb in seiner 1949 publizierten Arbeit *The Organization of Behavior* vor, bei der die Verbindung zwischen zwei Neuronen verstärkt wird, die gleichzeitig aktiv sind:

[323] Vgl. Pfeifer und Scheier 2001, S. 151.

> „When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."[324]

Umgekehrt gilt, dass „Verbindungen zwischen A und B abgebaut werden, wenn keine Aktivierung von B durch A erfolgt (falls vorher Verbindungen bestanden haben)."[325] Mit η als Lernrate, die bestimmt, wie schnell sich die Gewichtungen ändern und unter der referierten Annahme, dass $o_i = a_i$, lässt sich die Hebbsche Lernregel eines künstlichen neuronalen Netzes über den folgenden Term formalisieren:[326]

$$\Delta w_{ij} = \eta a_i a_j \tag{4.11}$$

Wie Klüver et al. anmerken, wird die Lernrate η problembezogen feinjustiert, zumeist mit einem reelzahligen Wert $0 < \eta < 1$.[327] Aktualisiert werden die einzelnen Gewichtungen mit jedem Zeitschritt t nach dem folgenden Ausdruck:

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij} \tag{4.12}$$

4.3.1.5 Lernmethoden

Lernverfahren künstlicher neuronaler Netze lassen sich unterscheiden in überwachte, unüberwachte und verstärkende Methoden. Wird dem künstlichen neuronalen Netz bei überwachtem Lernen (*supervised learning*) sowohl das Ein- als auch das Ausgabemuster vorgegeben und zu jedem Zeitschritt die Abweichung vom Sollwert bestimmt, um anschließend die Gewichtungen entsprechend anzupassen, wählt das KNN im Rahmen des nicht überwachten Lernens besonders aktive Neuronen nach dem *Winner-take-all* Prinzip aus und arrangiert die anderen Neuronen des Netzes in Gruppen (*cluster*) um die Gewinnerneuronen:

> „Nach jeweils einem Eingabesignal wird das Neuron ausgewählt, das den höchsten Aktivierungsstand hat; dies ergibt bei mehreren Eingabesignalen

[324] Hebb 2002, S. 62. Als *Axon* werden die „festen Pfade, auf denen Neuronen miteinander kommunizieren" (Kruse et al. 2011, S. 10) bezeichnet.
[325] Klüver et al. 2012, S. 127.
[326] Vgl. Pfeifer und Scheier 2001, S. 151.
[327] Vgl. Klüver et al. 2012, S. 127.

gewöhnlich auch mehrere selektierte Neuronen. Diese Neuronen bilden dann die Clusterzentren. Dadurch wird insbesondere gewährleistet, dass nach erfolgter Lernphase neue Signale, die einem bereits eingegebenen Signal ähnlich sind, auch wieder den entsprechenden Cluster aktivieren."[328]

Lässt sich bei unüberwachtem Lernen die Hebbsche Lernregel zur automatisierten Anpassung der Verknüpfungsgewichtungen verwenden, so wird bei dem überwachten Lernen häufig die *Delta-Lernregel* verwendet, die auch als *Widrow-Hoff-Regel*[329] bezeichnet wird und die Differenz zwischen Soll- und Istwert berücksichtigt:

$$\Delta w_{ij} = \eta(t_i - a_i)a_j = \eta\delta_i a_j \tag{4.13}$$

Wie zuvor besprochen, signifiziert auch im oben wiedergegebenen Ausdruck Δw_{ij} die Gewichtungsänderung der Verknüpfung von Knoten j zu Knoten i in einem Lernschritt; η bezeichnet die Lernrate und a_i sowie a_j benennen das Aktivierungsmaß der beiden Knoten. Das dem Neuron i zugeordnete Lernmuster – d.h. die gewünschte Ausgabe von i – ist mit t_i angegeben und δ_i signifiziert die Differenz zwischen der erwarteten Aktivierung t_i und der tatsächlichen Aktivierung a_i des Neurons.

4.3.1.6 Fehlerrückführung

Lässt sich die Differenz zwischen Soll- und Istwert in zweischichtigen Netzen über die Deltaregel bestimmen, stellen Netztopologien mit versteckten Schichten wie das mehrschichtige Perzeptron ein Problem für die Deltaregel dar, weil unklar ist, welchen Einfluss die versteckten Neuronen auf die Trainingsdaten haben.[330] Um Lernverhalten in Netztopologien mit versteckten Schichten zu ermöglichen und zu implementieren, wird eine Verallgemeinerung der Deltaregel verwendet: das von Rumelhart et al.[331] vorgeschlagene Verfahren der Fehlerrückführung (*Backpropagation*), bei der als Aktivierungsfunktion die logistische

[328] Klüver et al. 2012, S. 129.
[329] Vgl. Widrow und Hoff 1960.
[330] Vgl. Russell und Norvig 2012, S. 852.
[331] Rumelhart et al. 1985.

Aktivierungsfunktion auf die gewichtete Summe der Eingaben angewendet wird.[332]

Das Verfahren der Fehlerrückführung gestaltet sich in den folgenden Schritten:[333]

- Zuallererst wird das Trainingsmuster in das Netz eingespeist und anschließend durch das vorwärts geknüpfte Netz geleitet.
- Darauf folgend wird das Ausgabemuster mit dem intendierten Muster verglichen und die Differenz zwischen Soll- und Istwert bestimmt.
- Die berechnete Differenz, d.h. der Fehler des Netzes, wird – ausgehend von der Ausgabeschicht – im letzten Schritt des Verfahrens Schicht um Schicht zurückgeleitet, bis die Eingabeschicht erreicht ist. In jedem Zeitschritt werden die Gewichte zwischen den Schichten entsprechend angepasst.

Mit ihren wenigen Schritten leistet die Fehlerrückführung somit eine Minimierung des Fehlers gegenüber den Trainingsdaten, erweist sich in der Praxis häufig jedoch als wenig praktikabel. So weist Fahlman darauf hin, dass die Fehlerrückführung zu inperformant für viele Anwendungen sei und schlecht mit komplexeren Anforderungen skaliere. [334] Erweiterungen und Ergänzungen der Fehlerrückführung seien an dieser Stelle mit den Algorithmen *QuickProp*[335] (auch als *Resilent Backpropagation* bezeichnet) sowie *Rprop* [336] (auch als *Fahlman-Algorithmus* signifiziert) angemerkt.

4.3.1.7 Mehrschichtige Perzeptren und Backgammon

Ihre Anwendung finden künstliche neuronale Netze wie das mehrschichtige Perzeptron – neben zahlreichen weiteren Arbeitsgebieten – in der Implementation schlauer artifizieller Computergegner oder auch in der

[332] Vgl. Ertel 2013, S. 274.
[333] Vgl. Russell und Norvig 2012, S. 852f. In ihren Ausführungen leisten Russell und Norvig eine mathematische Herleitung der Fehlerrückführung (vgl. Russell und Norvig 2012, S. 853f.). Aufgrund des überblickenden Charakters dieses Kapitels sei die Fehlerrückführung einzig beschrieben und nicht erschöpfend hergeleitet.
[334] Vgl. Fahlman 1988, S. 3. Einen kurzen Überblick über Probleme und Erweiterungen der Fehlerrückführung bietet Kroll 2013, S. 247–250.
[335] Vgl. Fahlman 1988. Ein Vergleich zwischen der Fehlerrückführung und dem *QuickProp* Algorithmus bieten Waugh und Adams 1997.
[336] Vgl. Riedmiller und Braun 1992 sowie Riedmiller und Braun 1993. Eine Erweiterung des *Rprop* Algorithmus leisten Igel und Hüsken 2000.

Umsetzung von Systemen, die geschicktes Spielverhalten spielübergreifend realisieren.[337] Wie Mańdziuk anmerkt, werden KNN im betrachteten Kontext der Brettspiele häufig dazu verwendet, die Evaluierungsfunktion zu realisieren.[338] Zumeist umgesetzt als mehrschichtiges Perzeptron, wird die Netztopologie auf das jeweilige Spiel zugeschnitten und werden spielspezifische Gegebenheiten über das neuronale Netz repräsentiert:

> „In many cases some of the weights in the network may be fixed (i.e. non-trainable) – often to make some gamespecific concepts, such as particular spatial arrangements, obvious to the network. The network's input layer is often extended to include not only the raw representation of a game state, but also a number of predefined game features."[339]

Als ein Beispiel für die erfolgreiche Verwendung künstlicher neuronaler Netze im Brettspielen mag das in den späten 1980er Jahren von Tesauro entwickelte und vorgestellte Programm *Neurogammon* dienen,[340] das die im Jahre 1989 ausgetragene erste *Computer Olympiade* im Backgammonspiel gegen andere Computerprogramme für sich zu entscheiden vermochte.[341] Seine Spielstärke erzielte das – nur kurze Zeit später um elaborierte Verfahren wie dem *Temporal Difference Lernen* erweiterte – Programm durch sieben mehrschichtige Perzeptren, die der Bearbeitung unterschiedlicher Aufgaben dienten: Ein Netzwerk beschäftigte sich mit der Frage, ob und zu welchem Zeitpunkt des Spieles dem Gegenspieler der Dopplerwürfel[342] anzubieten sei und sechs neuronale Netze bewerteten die Zugmöglichkeiten in unterschiedlichen Phasen des Spieles. Alle Netze waren umgesetzt als vorwärtsverknüpfte mehrschichtige Perzeptren mit einer Eingabeschicht sowie einer Ausgabe- und einer versteckten

[337] Vgl. den Aufsatz Mańdziuk et al. 2013.
[338] Vgl. Mańdziuk 2010, S. 57.
[339] Mańdziuk 2010, S. 57.
[340] Vgl. Tesauro 1990.
[341] Vgl. Tesauro 1989.
[342] Im Backgammonspiel ist es mit dem Dopplerwürfel möglich, die Auszahlung des Spieles zu erhöhen: Ist sich ein Spieler sicher, dass er das Spiel gewinnen wird, kann er seiner Gegenspielerin den Dopplerwürfel anbieten. Nimmt die Spielerin nicht an, so verliert sie das Spiel. Nimmt die Spielerin den Würfel an und verfestigt sich im Laufe des Spieles ihr Eindruck, sie könne das Spiel für sich entscheiden, ist es ihr möglich, dem Gegenspieler den Dopplerwürfel erneut anzubieten, um die Auszahlung des Spieles abermals zu verdoppeln.

Schicht; die Trainingsphase der mehrschichtigen Perzeptren arbeitete mit dem Verfahren der Fehlerrückführung, um die Neuronengewichtungen dynamisch anzupassen.[343]

Seine Leistungsfähigkeit im Wettstreit gegen andere Computerprogramme erreichte *Neurogammon* jedoch nicht aus sich heraus, sondern durch menschliches Spielwissen: Alleine das neuronale Netz, das der Entscheidungsfindung um den Dopplerwürfel diente, wurde trainiert und evaluiert mit rund 3000 Spielkonfigurationen, die von menschlichen Expertinnen und Experten bewertet wurden.[344] Die Entscheidung, welcher Spielzug sinnvoll und welche Zugwahl weniger schlau sei, basierte ebenfalls auf menschlicher Expertise – und gestaltete sich ob der Komplexität des Arbeitsgegenstandes als „extrem mühsam für den Experten“ [345], wie Russell und Norvig anmerken. Den Herausforderungen,[346] die aus *Neurogammon* erwuchsen, nahm sich Tesauro mit dem 1992 veröffentlichten Nachfolger von *Neurogammon* an: *TD-Gammon* arbeitet mit künstlichen neuronalen Netzen, die mit Verfahren des verstärkenden Lernens trainiert wurden, benötigt sehr wenig Wissen um das Backgammonspiel, weil es sich jenes Spielwissen im Spiel gegen sich selbst erschließt – und ist so spielstark, dass das Programm noch heute als einer der besten artifiziellen Backgammonspieler gilt.[347]

Soll an späterer Stelle noch ausführlich auf Tesauros *TD-Gammon* einzugehen sein, so stellen die nachfolgenden Kapitel mit den Verfahren des Lernens anhand temporaler Differenz und dem Algorithmus des Q-Lernens zwei Methoden

[343] Vgl. Tesauro 1990, S. 33.
[344] Vgl. Tesauro 1990, S. 36.
[345] Russell und Norvig 2012, S. 982.
[346] Tesauro formuliert die Herausforderungen, die an eine verbesserte Version von *Neurogammon* gestellt sind, wie folgend: „While Neurogammon can only be described as a strong program, it clearly has to improve in a number of ways before it can be claimed to perform at human expert level. One obvious improvement would be to train on a larger and more varied expert data set containing data from several experts exhibiting a variety of styles of play. The move-selection networks in particular are dangerously inbred, having been trained only on data from one individual. Further significant improvement might be obtained by adding a small amount of look-ahead, perhaps by going to a 3-ply search.“ (Tesauro 1990, S. 36).
[347] Vgl. Wiering 2010, S. 61.

verstärkenden Lernens vor, die ihre Anwendung in der Implementation spielstarker Computergegner finden.

4.3.2 Lernen durch Verstärkung

In den vorangegangenen Kapiteln wurden Lernverfahren künstlicher neuronaler Netze betrachtet, die überwacht oder nicht überwacht verfahren. Basieren überwachte Verfahren, wie sie bei dem mehrschichtigen Perzeptron ihre Anwendung finden, auf dem Abgleich zwischen Soll- und Istwert sowie der anschließenden iterativen Minderung der Differenz zwischen Erwartung und Ergebnis, wie es sich mit dem Algorithmus der Fehlerrückführung realisieren lässt, so erfolgt unüberwachtes Lernen unter Verwendung der referierten Hebbschen Lernregel einzig auf der Grundlage des Eingabemusters. Der in diesem und den folgenden Kapiteln vorgestellte Ansatz des Lernens durch Verstärkung (im Englischen als *reinforcement learning* bezeichnet) beschreitet einen anderen Weg als die betrachteten Lernverfahren – einen Methode, das Lernen zu lernen.

Im Kontext des verstärkenden Lernens steht das Konzept des Agenten und das Verhältnis des Agenten zu seiner Umwelt im Mittelpunkt. Wie in Kapitel 2.3 dargelegt, lässt sich ein Agent definieren als ein System, das in eine Umgebung eingebettet und dazu fähig ist, selbstständig und eigenverantwortlich Handlungen zu vollziehen, um individuell relevante Ziele zu verfolgen und zu erreichen. Seine Umgebung nimmt der Agent wahr über Sensoren; Handlungen in der Umgebung des Agenten vollziehen sich durch Aktuatoren.

Bei dem verstärkenden Lernen verfügt der brettspielende Agent zu Beginn seiner Spieltätigkeit einzig über Wissen um das Spiel und seine Charakteristika wie die Zugmöglichkeiten der Spielfiguren, die individuellen und überindividuellen Spielziele, die Güte von Spielereignissen – wird der Agent schachmattgesetzt, ist das für ihn ein schlechtes Spielereignis, setzt der Agent jedoch seinen Gegenspieler schachmatt, ist das für ihn ein positives Ereignis – sowie die Auszahlung am Ende des Spieles. Welche Zugmöglichkeiten im Spielverlauf sinnvoll und welche weniger sinnvoll sind, wird dem Agenten nicht durch extrinsisches Expertenfeedback gewahr, das aus der beharrlichen Überwachung des Spielverhaltens des Agenten resultiert; die Spielstärke des Agenten emergiert vielmehr intrinsisch, aus früheren Spielerfahrungen: Ein Zug, der das Spiel zu einer

hohen Auszahlung bringt, wird belohnt und verstärkt; Spielzüge jedoch, die das Spiel zu einem schlechten Ausgang bringen, werden geschwächt, werden bestraft.

Algorithmen des verstärkenden Lernens lassen einen Agenten anhand von Erfahrungen lernen, die der Agent durch die Interaktion mit seiner Umgebung macht und im früheren Interaktionsverlauf gemacht hat.[348] Am Ende jeder Iteration des Lernprozesses steht die Belohnung oder die Bestrafung durch die Agentenumwelt. Die Intention verstärkend agierender Lernmethoden besteht darin, „anhand beobachteter Belohnungen [oder Bestrafungen] eine optimale (oder fast optimale) Strategie für die Umgebung zu lernen.“[349]

4.3.2.1 Markov-Entscheidungsprozesse

Verstärkende Lernverfahren gründen in der Theorie der Markov-Entscheidungsprozesse. Als endlicher Markov-Entscheidungsprozess ist ein Verlauf von Entscheidungen signifiziert, bei dem die Belohnung des Agenten einzig von dem Zustand der Umwelt und der Aktion des Agenten abhängt[350] und ist definiert als Tupel (S, A, P, R, γ). Hierbei beizeichnet[351]

- die Menge $S = \{s^1, s^2, \dots, s^n\}$ den Zustandsraum, d.h. die möglichen Zustände der Umwelt – im vorliegenden Falle die Brettspielwelt –; $s_t \in S$ repräsentiert den Zustand der Agentenumwelt zum Zeitpunkt t.
- A signifiziert die Menge der Entscheidungen bzw. Zugmöglichkeiten, die dem Agenten zu jedem Umweltzustand $A(s)$ zur Verfügung stehen. Mit $a_t \in A(s_t)$ ist die von dem Agenten ausgeführte Aktion zum Zeitpunkt t bezeichnet.
- Die von der Umwelt determinierte Übergangsfunktion $P(s, a, s')$ führt die Umwelt in ihrem Zustand s ob der Handlung a des Agenten zu einem neuen möglichen Nachfolgezustand s'.
- Die Belohnungsfunktion $R(s, a, s')$ stellt das Feedback dar, das der Agent aufgrund seiner Handlung erhält. Abhängig ist die Belohnung vom

[348] Vgl. Wiering 2010, S. 59.
[349] Russell und Norvig 2012, S. 960.
[350] Vgl. Ertel 2013, S. 291.
[351] Die Formalisierung des Markov-Entscheidungsprozesses ist wiedergegeben nach Ertel 2013, S. 290f. und Wiering 2010, S. 59.

aktuellen Zustand der Umwelt und der ausgeführten Handlung des Agenten.

- Der konstante Diskontierungsfaktor (*discount factor*) γ mit $0 \leq \gamma \leq 1$ lässt sich dazu verwenden, um das Lernverhalten des Agenten feinzujustieren und dient dazu, künftige Belohnungen gegenüber zeitnahen Belohnungen abzuschwächen.

Verstärkendes Lernen intendiert, den Agenten aufgrund seiner Erfahrungen mit der Umwelt eine optimale Strategie $\pi^*(s) \rightarrow a^*|s$ identifizieren und lernen zu lassen. Eine Strategie – in der Literatur zumeist mit dem Begriff *policy* bezeichnet und häufig auch als *Politik* oder *Taktik* übersetzt – ist optimal, wenn sie die Belohnung des Agenten auf lange Sicht maximiert.[352] Die an Verfahren des verstärkenden Lernens gestellte Herausforderung besteht darin, eine optimale Strategie π^* für beliebige Agentenumgebungen zu lernen.

In den folgenden Kapiteln werden zuerst Methoden der dynamischen Programmierung vorgestellt, die sich der optimalen Strategie π^* anzunähern vermögen, bevor anschließend auf methodenimmanente Einschränkungen der betrachteten Verfahren eingegangen und das Lernen anhand temporaler Differenz mit den Algorithmen TD(0) und TD(λ) besprochen wird. Die überblickende Betrachtung verstärkender Lernverfahren schließt mit dem Q-Lernen, das seine Anwendung im entwickelten Framework *SpoookyJS* findet, um ein Agentengedächtnis zu realisieren.

4.3.2.2 Dynamische Programmierung und Wertiteration

Die Methode der dynamischen Programmierung,[353] die ihre Effizienz aus der Zwischenspeicherung ermittelter Lösungen schöpft, basiert auf der Unterteilung von Problemen in immer kleinere Teil- und Unterprobleme und der rekursiven Bearbeitung der Teilprobleme zur inkrementellen Lösung der Oberprobleme. Damit die dynamische Programmierung ihre Anwendung finden kann, muss das

[352] Vgl. Ertel 2013, S. 290f. sowie die Definition einer optimalen Strategie: „An optimal policy is defined as a policy that receives the highest possible cumulative discounted rewards in ist future from all states.“ (Wiering 2010, S. 59).

[353] Der Name der Methode geht zurück auf die 1957 publizierte gleichnamige Monographie von Bellman (vgl. die Neuauflage der Arbeit: Bellman 2010).

anzunähernde Optimierungsproblem die folgenden Eigenschaften erfüllen: Zum einen muss das Kriterium erfüllt sein, dass die Lösungen der Teil- und Unterprobleme die Lösungsgrundlage des Oberproblems darstellen – „jede Teillösung [ist optimal], die in der großen Lösung enthalten ist“[354]. Zum anderen müssen die Charakteristika des Entscheidungsprozesses erfahrungsunabhängig bekannt sein, d.h. sowohl die Übergangsfunktion als auch die Belohnungsfunktion und ihre jeweiligen Ergebnisse zu einem bestimmten Zustand der Umwelt und einem determinierten Zeitpunkt müssen apriori offenbar sein – und dürfen folglich nicht durch die erfahrungsabhängige Mutmaßung des Agenten bestimmt sein.

Sei s_t der Umweltzustand zum Startzeitpunkt des Lernverfahrens, r_t das von der Umwelt an den Agenten zurückgegebene Feedback zum Zeitpunkt t und π die vom Agenten verfolgte Strategie, so lässt sich der erwartete Wert eines Umweltzustandes $V^{\pi}(s_t)$ ermitteln über den folgenden Term, der die *abgeschwächte* Belohnung (*discounted reward*) bestimmt:[355]

$$V^{\pi}(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots = \sum_{i=0}^{\infty} \gamma^i r_{t+i} \tag{4.14}$$

Wie Ertel anmerkt, wird eine Strategie π^* als optimal bezeichnet, wenn für alle Umweltzustände s gilt: $V^{\pi^*}(s) \geq V^{\pi}(s)$, die Strategie π^* folglich „mindestens so gut wie alle anderen Strategien entsprechend dem definierten Wertmaßstab“[356] ist. Zugrunde gelegt das Bellmansche Optimalitätsprinzip, das besagt, dass sich jede optimale Lösungsstrategie aus optimalen Teilstrategien konstituiert,[357] lässt sich der Term der abgeschwächten Belohnung zur Bellman-Gleichung umformen, die sich wie folgend formuliert:[358]

$$V^{\pi^*}(s) = \sum_{s'} P(s, \pi^*(s), s') \left(R(s, \pi^*(s), s') + \gamma V^{\pi^*}(s') \right) \tag{4.15}$$

[354] Weicker und Weicker 2013, S. 207.
[355] Vgl. Ertel 2013, S. 290f.
[356] Ertel 2013, S. 291.
[357] Vgl. Papageorgiou et al. 2012, S. 335f.
[358] Vgl. Russell und Norvig 2012, S. 759. Die Bellman-Gleichung und die nachfolgenden Terme der Wertiteration sind wiedergegeben nach Wiering 2010, S. 59.

Um sich der optimalen Wertfunktion $V^{\pi^*}(s)$ anzunähern, lässt sich das Verfahren der Wertiteration verwenden, aus der die folgende Iterationsvorschrift resultiert:

$$V^{k+1}(s) = max_a \sum_{s'} P(s,a,s')\Big(R(s,a,s') + \gamma V^k(s')\Big) \qquad (4.16)$$

Endlich ist es dem Agenten möglich, die optimale Aktion nach den überblickend referierten Verfahren der dynamischen Programmierung unter Verwendung des folgenden Terms zu identifizieren und zu wählen:

$$\pi^*(s) = argmax_a \sum_{s'} P(s,a,s')\big(R(s,a,s') + \gamma V^*(s')\big) \qquad (4.17)$$

4.3.2.3 Lernen anhand temporaler Differenz

Wie im vorangegangenen Kapitel eingangs angemerkt, müssen die Charakteristika des Markov-Entscheidungsprozesses apriori bekannt sein, um Methoden der dynamischen Programmierung anwenden zu können. Im Alltäglichen jedoch schöpft sich menschliches Lernverhalten nicht aus der vollständigen Kenntnis aller Umwelteigenschaften und Umweltzusammenhänge, sondern formt sich aus früheren Erfahrungen, die mit und in der Umwelt gemacht wurden. Lernen allein aus Erfahrungen, folglich den individuellen „Nutzen der beobachteten Zustände anhand der beobachteten Übergänge anzupassen, so dass sie mit den Bedingungsgleichungen übereinstimmen“ [359], lässt sich durch Lernen anhand temporaler Differenz (im Folgenden mit TD-Lernen abgekürzt) erreichen.

Das Verfahren des TD-Lernens wurde zuerst im Jahre 1959 von Samuel beschrieben[360] und in seinem Dameprogramm implementiert, das die Differenz zwischen zwei aufeinanderfolgenden Spielkonfigurationen und ihren Bewertungen dazu verwendete, um früher vorgenommene Gewichtungen dynamisch anzupassen.[361] Knapp drei Jahrzehnte später, im Jahre 1988, leistete Sutton die Formalisierung des Verfahrens in seiner Arbeit *Learning to predict by the methods of temporal differences* und erweiterte das Lernverfahren anhand

[359] Russell und Norvig 2012, S. 966.
[360] Vgl. Samuel 1963.
[361] Vgl. Sutton 1988, S. 10.

temporaler Differenz um den Algorithmus $TD(\lambda)$.[362] Sutton und Barto[363] beschreiben TD-Lernen als Kombination der Ideen um Monte Carlo Verfahren und der dynamischen Programmierung: Analog den betrachteten Monte Carlo Methoden benötigen auch TD-Methoden kein Vorwissen um die Umwelt und ihrer Charakteristika. Wie die dynamische Programmierung, so nähern sich auch TD-Verfahren einer Lösung, indem sie Bewertungen aktualisieren aufgrund früher durchgeführter und gelernter Bewertungen – dieser Prozess wird auch als *Bootstrapping* bezeichnet.[364] Die Idee und das Ziel des TD-basierten Lernprozesses fasst Tesauro wie nachfolgend wiedergegeben zusammen:

> „The basic idea of TD methods is to base learning on the difference between temporally successive predictions. In other words, the goal is to make the learner‘s current prediction for the current input pattern more closely match the subsequent prediction at the next time step.“[365]

Von den sowohl quantitativ als auch qualitativ mannigfaltigen Algorithmen des verstärkenden Lernens seien im Folgenden die Algorithmen $TD(0)$ sowie $TD(\lambda)$ und das Verfahren des *Q*-Lernens betrachtet und überblickend wiedergegeben.[366]

4.3.2.4 TD(0) und TD(λ)

Der $TD(0)$ Algorithmus berechnet den Wert des Zustandes zum Zeitpunkt t unter Verwendung der Lernrate α mit $0 \leq \alpha \leq 1$ anhand der Lernregel $V(s_t)$, wie in dem folgenden Ausdruck wiedergegeben.

$$V(s_t) = V(s_t) + \alpha\big(r_t + \gamma V(s_{t+1}) - V(s_t)\big) \tag{4.18}$$

Mit Hilfe der angeeigneten Lernregel ist es dem Agenten möglich, eine Strategiewahl nach dem folgenden Term vorzunehmen:

$$\pi(s) = argmax_a \sum_{s'} P(s,a,s')\big(R(s,a,s') + \gamma V(s')\big) \tag{4.19}$$

[362] Vgl. Sutton 1988.
[363] Sutton und Barto 1998.
[364] Vgl. die Verortung des TD-Lernens neben Monte Carlo Methoden und Verfahren der dynamischen Programmierung in Sutton und Barto 1998, S. 133.
[365] Tesauro 2002, S. 182.
[366] Die Wiedergabe ausgewählter Algorithmen des TD-Lernens folgt den Ausführungen in Wiering 2010, S. 59f.

$TD(0)$ fokussiert einzig die aktuelle Beobachtung der Umgebung: Der Algorithmus blickt einen Schritt voraus und passt die Bewertungen aufgrund ihrer Abweichungen an.[367] Die außerordentliche Spielstärke des an späterer Stelle noch eingehender betrachteten Programmes *TD-Gammon* im Backgammonspiel basiert auf dem $TD(\lambda)$ Algorithmus – ein Algorithmus, der in digitalen Brettspielen häufig seine Anwendung findet, um die Gewichtungen der Evaluierungsfunktion dynamisch anzupassen.[368] $TD(\lambda)$ stellt eine Verallgemeinerung des TD-Lernens dar und arbeitet mit dem $TD(0)$ Fehler δ_t von $V(s_t)$, der sich wie folgend bestimmt:

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{4.20}$$

Der zusätzliche Parameter $\lambda \in [0,1]$ wirkt auf die Relevanz früherer Strategiewahlen in der aktuellen Entscheidungsfindung ein und beeinflusst das Verhalten des Algorithmus wesentlich: Mit $\lambda = 1$ verfährt der Algorithmus analog der Monte Carlo Methode, mit $\lambda = 0$ ist das Verhalten nach $TD(0)$ signifiziert.[369] Sei der $TD(\lambda)$-Fehler zum Zeitpunkt t mit δ_t^λ definiert als

$$\delta_t^\lambda = \sum_{i=0} (\gamma\lambda)^i \delta_{t+1}, \tag{4.21}$$

gestaltet sich die Minderung des TD-Fehlers nach $V(s_t) \leftarrow V(s_t) + \alpha\delta_t^\lambda$. Mit Einführung des Mechanismus der *Elegibility Traces*,[370] die ein Kurzzeitgedächtnis

[367] Vgl. Mańdziuk 2010, S. 65.
[368] Vgl. Mańdziuk 2010, S. 66.
[369] Vgl. Szepesvári 2013, S. 23. Wiering weist darauf hin, dass ein praktikabler Wert zwar anwendungsabhängig sei, sich jedoch häufig $0.6 \leq \lambda \leq 0.9$ bewährt habe (vgl. Wiering 2010, S. 60).
[370] Zur tiefergehenden Erläuterung der *Elegibility Traces* sei an dieser Stelle ausführlich Sutton und Barto zitiert: „There are two ways to view eligibility traces. The more theoretical view, which we emphasize here, is that they are a bridge from TD to Monte Carlo methods. When TD methods are augmented with eligibility traces, they produce a family of methods spanning a spectrum that has Monte Carlo methods at one end and one-step TD methods at the other. In between are intermediate methods that are often better than either extreme method. In this sense eligibility traces unify TD and Monte Carlo methods in a valuable and revealing way. The other way to view eligibility traces is more mechanistic. From this perspective, an eligibility trace is a temporary record of the occurrence of an event, such as the visiting of a state or the taking of an action. The trace marks the memory parameters associated with the event as eligible for undergoing learning changes. When a TD error occurs, only the eligible states or actions are assigned credit or blame for the error. Thus, eligibility traces help bridge the gap between events and training information. Like TD methods themselves, eligibility traces are a basic mechanism for temporal credit assignment. For reasons that will become apparent shortly, the more theoretical view of eligibility traces is called the forward view, and the more mechanistic view is called the backward view. The forward view is most useful for understanding what is computed by methods using eligibility

über bestimmte eingetretene Ereignisse implementieren, lässt sich die Aktualisierungsregel endlich formulieren als

$$V(s) = V(s) + \alpha\delta_t e_t(s) \quad \text{mit } e_t(s) = \gamma\lambda e_{t-1}(s) + \eta_t(s) \tag{4.22}$$

Die Indikatorfunktion $\eta_t(s)$ liefert den Wert 1 zurück, wenn ein Zustand s zum Zeitpunkt t erreicht wurde und gibt im anderen Falle den Wert 0 zurück.[371]

4.3.2.5 Q-Lernen

Oft als wohlerforschtes,[372] leicht zu implementierendes und ebendarum häufig eingesetztes Verfahren referiert,[373] bietet sich mit der Methode des Q-Lernens ein Ansatz verstärkenden Lernens, bei dem der lernende Agent über kein vollständiges Wissen verfügen muss, wie seine Umgebung funktioniert, um effizientes Lernverhalten zu generieren und sinnvolle Entscheidungen zu treffen. Die Kernidee des im Jahre 1989 von Watkins in seiner Arbeit *Learning from delayed rewards*[374] vorgeschlagenen Algorithmus' besteht darin, den Agenten schrittweise die Aktion-Wertfunktion $Q(s, a)$ lernen und optimieren zu lassen. Anhand der Q-Funktion bestimmt der Agent den zu erwartenden Wert einer Handlung a, die dem Agenten zu einem bestimmten Umweltzustand s möglich ist. Jene Q-Werte, die über die Q-Funktion bestimmt werden, hält der Agent in einer zugriffsperformanten Datenstruktur fest und aktualisiert sie nach jeder Erfahrung, die er anhand seiner Aktionen und anhand des Feedbacks macht, das ihm nach ausgeführter Handlung von seiner Umwelt zurückgeliefert wird.

Die Erfahrungsbildung des Agenten vollzieht sich in mehreren Schritten: Zuallererst startet der Agent als Tabula rasa: Alle Q-Werte werden mit Null initialisiert oder mit Zufallswerten belegt. Im nächsten Schritt wählt der Agent

traces, whereas the backward view is more appropriate for developing intuition about the algorithms themselves." (Sutton und Barto 1998, S. 163).

[371] Vgl. Wiering 2010, S. 60.

[372] Vgl. Pfeifer und Scheier 2001, S. 494.

[373] Vgl. Wiering und van Otterlo 2012, S. 31.

[374] Watkins 1989. Vgl. auch Watkins 1992.

aufgrund seiner Strategie (*policy*) $\pi(s)$ eine Aktion aus der Menge von möglichen Aktionen A aus, die im darauffolgenden Schritt den Q-Wert maximiert:[375]

$$\pi(s) = a \text{ mit } Q(s,a) = max_{b \in A} Q(s,b) \tag{4.23}$$

Anschließend führt der Agent die ermittelte Aktion aus und erhält eine Rückmeldung r von seiner Umwelt. Anhand des Umweltfeedbacks, im Folgenden als *Belohnung* oder *Auszahlung* bezeichnet, aktualisiert der Agent seine Aktionserinnerungen – d.h. die gespeicherten Q-Werte – und nähert seine Q-Werte somit durch stetige Reaktion auf das Umweltfeedback den optimalen Q-Werten an. Der Lernprozess des Agenten gestaltet sich nach der im Folgenden wiedergegebenen Q-Lernregel:[376]

$$Q^{neu}(s,a) \leftarrow (1-\alpha)Q^{alt}(s,a) + \alpha\big(r + \gamma U(s)\big)$$
$$\text{mit} \tag{4.24}$$
$$U(s') = max_{b \in A} Q^{alt}(s',b)$$

Im Term ist mit $U(s')$ die maximale Auszahlung bezeichnet, die der Agent aufgrund seines Wissens um seine aktuelle Umgebung erwartet. Jene augenblickliche Agentenumgebung ist mit $Q^{alt}(s',b)$ repräsentiert, wobei s' den Umweltzustand bezeichnet, der aus der Handlung a im Zustand s resultiert. Die mit $U(s')$ bestimmte maximale Auszahlung, die für alle möglichen Agentenaktionen ausgehend vom aktuellen Umweltzustand berechnet werden konnte, wird mit dem Diskontierungsfaktor γ im Intervall $0 \leq \gamma \leq 1$ multipliziert, der dem Zweck dient, spätere Umweltbelohnungen gegenüber früheren Belohnungen abzuschwächen. Die Lernrate des Algorithmus‘ ist mit α im reellzahligen Intervall $[0,1]$ angegeben und bestimmt, wie stark ältere Q-Werte in den Aktualisierungsprozess einbezogen werden: Ist die Lernrate sehr hoch gewählt, wird $Q^{alt}(s,a)$ ignoriert; ist a sehr klein dimensioniert, erhalten vergangene Agentenerfahrungen eine große Bedeutung beigemessen.[377]

[375] Der Term zur Verdeutlichung der Aktionswahl ist wiedergegeben nach Pfeifer und Scheier 2001, S. 494.
[376] Die *Q*-Lernregel und ihre Beschreibung ist wiedergegeben nach Pfeifer und Scheier 2001, S. 495.
[377] Vgl. Pfeifer und Scheier 2001, S. 495.

Bevor im nächsten Kapitel die praktische Anwendung des Q-Lernverfahrens geleistet wird, um der Formelhaftigkeit des vorgestellten Verfahrens zu opponieren, sei abschließend der Algorithmus des Q-Lernverfahrens mit dem folgenden Pseudocode wiedergegeben:[378]

Algorithmus 4.5 *Pseudocode des Q-Lernalgorithmus'*

- Parameter:
  - *problem*: Instanz der Klasse *ReinforcementProblem*. Spezifikation des zu bearbeitetenden Problems mitsamt Zugriffsmöglichkeiten auf Problem-/Umweltzustand, mögliche Aktionen, etc.
  - *episodes*: Maximale Anzahl der Verfahrensläufe.
  - *timeLimit*: Maximale Laufzeit des Verfahrens in Millisekunden.
  - *alpha:* Lernrate im reellzahligen Intervall $[0,1]$.
  - *gamma:* Diskontierungsfaktor im reellzahligen Intervall $[0,1]$.
  - *rho:* Parameter im reellzahligen Intervall $[0,1]$; vermittelt zwischen Exploration- und Eploitation, indem der Parameter steuert, wie häufig eine zufällige oder die beste Aktion ausgeführt wird. Mit *rho* = 0 verfährt der Algorithmus in Gänze explorierend, mit *rho* = 1 probiert der Algorithmus immer neue Aktionen aus. Für offline-Lernen schlagen Millington und Funge einen Wert von $0{,}2$ vor.
  - *nu:* Der Parameter im reellzahligen Intervall $[0,1]$ dient dazu, die Anzahl der Iterationen festzulegen, die in einer Aktionsfolge durchgeführt werden. Ist *nu* = 0, verwendet der Algorithmus den Umweltzustand als Ausgangszustand für die nächste Iteration, den der Algorithmus in der vergangenen Iteration erreicht hat. Mit *nu* = 1 beginnt jede Iteration des Verfahrens mit einem zufällig gewählten Umweltzustand.

```
// Globale Instanz der Klasse QValueStore zur
// Speicherung der Q-Werte
qValues ← new QValueStore()
```

[378] Der Pseudocode des *Q*-Lernverfahrens ist wiedergegeben nach Millington und Funge 2009, S. 635f.

```
function QLearner(problem, episodes, timeLimit, alpha, gamma,
rho, nu)
    // Wähle zufallsbedingt einen Startzustand aus
    state ← problem.getRandomState()
    while timeLimit > 0 and episodes > 0 do
      startTime ← Date.now()
      // Auswahl eines neuen Zustandes in Abhängigkeit von nu
      // random() generiert eine reelle Zahl im Intervall 0..1
      if random() < nu then
          state ← problem.getRandomState()
      // Liste aller möglichen Aktion generieren
      actions ← problem.getAvailableActions(state)
      // Entweder: zufallsbasierte Wahl einer Aktion
      if random() < rho then
          action ← oneOf(actions)    // oneOf wählt zufällig
                                     // ein Element aus
      else  // Oder: Wahl der besten Aktion anhand der Q-Werte
          action ← store.getBestAction(state)
      // Aktion ausführen und Belohnung sowie aus Aktion
      // resultierenden neuen Umweltzustand speichern
      reward, newState ← problem.takeAction(state, action)
      // Q-Wert für state aus dem Speicher holen
      Q ← store.getQValue(state, action)
      // Q-Wert für die beste Aktion von newState holen
      maxQ ← store.getQValue(newState,
                   store.getBestAction(newState))
      // Anwendung der Q-Lernregel
      Q ← (1 −alpha) × Q + alpha × (reward + gamma × maxQ)
      // Speicherung des neuen Q-Wertes
```

```
store.storeQValue(state, action, Q)
// Aktuellen Umweltzustand aktualisieren
state ← newState
 // Aktualisierung der Laufzeitbedingungen
timeLimit ← timeLimit - (Date.now() — startTime)
episodes ← episodes - 1
```

Der vorgetragene Algorithmus nach Millington und Funge benötigt in seiner Arbeit die Klasse *ReinforcementProblem*, die das zu bearbeitende Problem mitsamt Zugriffsmöglichkeiten auf Umweltzustände definiert sowie die Klasse *QValueStore*, deren Intention und Aufgabe die Speicherung von Q-Werten und das Abfragen zusammengetragener Q-Werte leistet. Beide Klassen seien mit dem folgenden Pseudocode wiedergegeben.[379]

Klassendefinition *ReinforcementProblem*

```
class ReinforcementProblem
    // Liefere einen zufällig ausgewählten
    // Zustand des Problems zurück
    function getRandomState() return randomState
    // Hole alle möglichen Aktionen
    // für den Umweltzustand state
    function getAvailableActions(state)
       return possibleActions
    // Führe eine Aktion action im Umweltzustand state aus
    function takeAction(state, action)
       return reward, newState
```

Klassendefinition *QValueStore*

```
class QValueStore
```

[379] Die Klassendefinitionen sind wiedergegen nach Millington und Funge 2009, S. 636f.

```
// Hole den mit einer Aktion und einem Zustand
// verbundenen Q-Wert
function getQValue(state, action) return associatedQValue
// Frage die beste Aktion für einen Zustand ab
function getBestAction(state) return max QValues (state)
// Führe eine Aktion action im Umweltzustand state aus
function storeQValue(state, action, value)
  QValues[state][action] ← value
```

Die Performanz des Algorithmus‘ skaliert mit der Anzahl der Iterationen i und der Zustands- und Aktionsmenge. In seiner Zeitkomplexität gestaltet sich der Algorithmus als $\mathrm{O}(i)$; signifiziert a die Anzahl der Aktionen und s die Zustandsmenge, ist die Speicherkomplexität mit $\mathrm{O}(as)$ beschrieben.[380]

4.3.2.6 Q-Lernen – Ein Anwendungsbeispiel

Der Formelhaftigkeit der vorangegangenen Kapitel, die sich mit *TD*- und Q-Lernverfahren beschäftigten, sei mit den folgenden Ausführungen ein Anwendungsbeispiel beigestellt, das die Methode des Q-Lernens anhand einer Praxisanwendung veranschaulicht.[381] Im Anwendungsbeispiel, wie es in der folgenden Darstellung visualisiert ist, befindet sich der Agent in einem Gebäude und muss verschiedene Räume durchqueren, um einen Ausweg aus dem Bauwerk zu finden. Jeder Raum lässt sich im Kontext des Q-Lernens als Zustand und jeder Schritt, der den Agenten vom einen in den anderen Raum führt, als Aktion bezeichnen.

[380] Vgl. Millington und Funge 2009, S. 637f.

[381] Das Beispiel orientiert sich in seinen Darstellungen an dem *Q-Learning Step-By-Step Tutorial*, wie es sich findet unter http://mnemstudio.org/path-finding-q-learning-tutorial.htm (zuletzt aufgerufen am 03.04.2014).

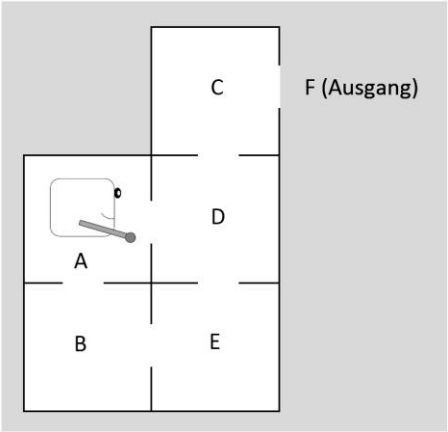


Abbildung 37: Veranschaulichung des Q-Lernverfahrens: Der Agent befindet sich in Raum A und muss den Ausgang aus dem Gebäude finden.

In Abbildung 37 befindet sich der Agent in Raum A, von dem aus er die Räume B und D erreichen kann. Das Ziel des Agenten besteht darin, möglichst schnell den Ausgang (Raum F) zu erreichen. Wüsste der Agent erfahrungsunabhängig von der Anordnung und Verknüpfung der Räume miteinander, könnte er mit der Aktionsfolge D → C → F das Problem optimal lösen. Der Agent weiß zu Beginn des Verfahrens jedoch nichts von seiner Umwelt und muss den Weg zum Ausgang durch Ausprobieren der verschiedenen Aktionsmöglichkeiten zuallererst lernen, um in späteren Erfahrungsepisoden von seinem gespeicherten Umweltwissen zu profitieren. Das Q-Lernverfahren, wie es im Folgenden beispielhaft vorgestellt wird, ermöglicht es dem Agenten, aus vergangenen Erfahrungen zu lernen und implementiert das verhaltenssteuernde Gedächtnis des Agenten.

Umweltzustände – d.h. im vorliegenden Falle der Raum, in dem sich der Agent befindet – und Aktionen, die Zustandstransformationen bewirken, lassen sich darstellen in einer Matrix, die über die ganzzahligen Belohnungen (positive Werte) und Bestrafungen (negative Werte) informiert, die der Agent bei jeder seiner Aktionen von der Umwelt erhält:

$$
\begin{array}{cc}
 & \text{Zustände / Räume} \\
 & \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\
R = & \begin{bmatrix} - & 0 & - & 0 & - & - \\ 0 & - & - & - & 0 & - \\ - & - & - & 0 & - & 0 \\ 0 & - & 0 & - & 0 & - \\ - & 0 & - & 0 & - & - \\ - & - & 100 & - & - & 100 \end{bmatrix} \begin{array}{c} A \\ B \\ C \\ D \\ E \\ F \end{array} \quad \text{Aktionen / Zustandsübergänge}
\end{array}
$$

Abbildung 38: Belohnungsmatrix für unterschiedliche Umweltzustände und Zustandsübergänge.

Wie am Rande der Belohnungsmatrix angemerkt, verzeichnet die Matrix das Umweltfeedback, das der Agent erhält, wenn er sich durch seine ausgeführte Handlung von einem Umweltzustand (im vorliegenden Beispiel ein Raum) in einen anderen Zustand seiner Umwelt bewegt. Querstriche in der Matrix signifizieren hierbei, dass keine Zustandsübergänge möglich sind – so ist es im obigen Beispiel nicht möglich, auf direktem Wege von Raum B nach Raum D, wohl aber nach Raum A zu gelangen. Ausgehend von dem oben dargestellten Umweltfeedback lässt sich nun das mehrschrittige *Q*-Lernverfahren des Agenten implementieren und betrachten. Zu Beginn des Verfahrens verzeichnet das Gedächtnis des Agenten keine verarbeiteten Erfahrungen – die Q-Wertetabelle des Agenten ist folglich leer bzw. alle Q-Werte werden mit Zufallswerten belegt oder mit Null initialisiert, wie mit der folgenden Q-Wertematrix veranschaulicht:

$$
\begin{array}{cc}
 & \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\
Q = & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{array}{c} A \\ B \\ C \\ D \\ E \\ F \end{array}
\end{array}
$$

Abbildung 39: Initiale Q-Wertematrix des Agenten.

Im ersten Schritt des Algorithmus‘ wird ein zufälliger Initialzustand ausgewählt, der Agent im vorliegenden Beispiel somit in Raum C platziert. Ausgehend von

Raum C bieten sich dem Agenten zwei Aktionsmöglichkeiten: Entweder er wechselt in den Raum D oder er erreicht den Zielzustand F. Das unmittelbare Umweltfeedback beträgt im einen Falle 0, im anderen Falle 100, wie in der folgenden Matrix veranschaulicht:

$$
\begin{array}{cc}
 & \text{Zustände / Räume} \\
 & \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\
R = & \begin{bmatrix} - & 0 & - & 0 & - & - \\ 0 & - & - & - & 0 & - \\ - & - & - & 0 & - & 0 \\ 0 & - & \boxed{0} & - & 0 & - \\ - & 0 & - & 0 & - & - \\ - & - & \boxed{100} & - & - & 100 \end{bmatrix} \begin{array}{c} A \\ B \\ C \\ D \\ E \\ F \end{array} \quad \text{Aktionen / Zustandsübergänge}
\end{array}
$$

Abbildung 40: Belohnungsmatrix mit hervorgehobenem Umweltfeedback für den Wechsel des Agenten von Raum C nach Raum D oder F.

Sei die Lernrate $\alpha = 1$ und der Diskontierungsfaktor $\gamma = 0.5$ und sei jede Aktionswahl des Agenten in den folgenden Erfahrungsepisoden vollständig dem Zufall anheimgestellt, wählt der Agent zufallsbasiert den Zielzustand F als seinen nächsten Schritt. Im Zielzustand F angelangt, bietet sich dem Agenten die Möglichkeit, zu Raum C zurückzukehren oder im Zielraum F zu bleiben. Basierend auf dem Umweltfeedback aktualisiert der Agent den Q-Wert der von Raum C nach Raum F führenden Aktion basierend auf der Q-Lernregel:

$$
\begin{aligned}
Q(C,F) = R(C,F) + 0.5 \times Max[Q(F,C), Q(F,F)] &= 100 + 0.5 \times Max[0{,}0] \\
&= 100 + 0.5 \times 0 = 100.
\end{aligned}
$$

Anschließend speichert der Agent den ermittelten Q-Wert in seinem Gedächtnis und das Verfahren nimmt mit einer weiteren Erfahrungsepisode einen neuen Lauf.

$$Q = \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 100 & 0 & 0 & 0 \end{bmatrix} \end{array} \begin{array}{c} A \\ B \\ C \\ D \\ E \\ F \end{array}$$

Abbildung 41: Q-Wertematrix des Agenten nach einer Erfahrungsepisode.

In der neuen Episode startet der Agent in Raum A und entscheidet sich zwischen den beiden Räumen B und D. Die Aktionswahl fällt auf Raum D, so dass sich der Agent in den Raum bewegt und anschließend den entsprechenden Q-Wert bestimmt: $Q(A,D) = R(A,D) + 0.5 \times Max[Q(D,A), Q(D,C), Q(D,E)] = 0 + 0.5 \times 0 = 0$. Anschließend entscheidet sich der Agent dafür, in den benachbarten Raum C zu wechseln. Der Q-Wert für den Zustandsübergang bestimmt sich nun als $Q(D,C) = R(D,C) + 0.5 \times Max[Q(C,F), Q(C,D)] = 0 + 0.5 \times 100 = 50$ und die Q-Wertematrix des Agenten findet sich wie folgend dargestellt ergänzt:

$$Q = \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 50 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 100 & 0 & 0 & 0 \end{bmatrix} \end{array} \begin{array}{c} A \\ B \\ C \\ D \\ E \\ F \end{array}$$

42: Q-Wertematrix des Agenten nach zwei Erfahrungsepisoden.

Im nächsten und letzten Schritt wählt der Agent die Aktion F und findet sich auf seine Aktionswahl folgend im finalen Zustand wieder. Der Q-Wert bestimmt sich schließlich erneut über den oben wiedergegebenen Term mit $Q(C,F) = 100$.[382] In der nächsten Erfahrungsepisode startet der Agent in Raum B und bewegt sich

[382] Ist die Lernrate $\alpha \neq 1$ und > 0, gestaltet sich der Algorithmus akkumulierend.

mit zufälligen Aktionswahlen durch die Räume in der Reihenfolge B → A → D → C → F. Die jeweiligen Q-Werte aktualisieren sich somit in den folgenden Schritten:

I. $Q(B,A) = R(B,A) + 0.5 \times Max[Q(A,B), Q(A,D)] = 0 + 0.5 \times 0 = 0$

II. $Q(A,D) = R(A,D) + 0.5 \times Max[Q(D,A), Q(D,C)] = 0 + 0.5 \times 50 = 25$

III. $Q(D,C) = R(D,C) + 0.5 \times Max[Q(C,D), Q(C,F)] = 0 + 0.5 \times 100 = 50$

Nach der dritten Erfahrungsepisode verzeichnet die Q-Wertematrix des Agenten drei gespeicherte Werte:

$$Q = \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 50 & 0 & 0 \\ 25 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 100 & 0 & 0 & 0 \end{bmatrix} \end{array} \begin{array}{c} A \\ B \\ C \\ D \\ E \\ F \end{array}$$

Abbildung 43: Q-Wertematrix des Agenten nach drei Erfahrungsepisoden.

Schritt für Schritt, Erfahrungsepisode für Erfahrungsepisode generiert der Agent ein Gedächtnis, das ihn zu jedem Zeitpunkt über die Gegebenheiten der Umgebung zu informieren vermag. Weist jenes Gedächtnis eine ausreichend hohe Anzahl von Q-Werten auf, wie es der Fall ist nach zahlreichen Iterationen des Verfahrens, so ist es dem Agenten möglich, die zu einem bestimmten Zeitpunkt bestmögliche Handlung auszuwählen, indem der Agent diejenige Aktion auswählt und ausführt, die – für den aktuellen Umweltzustand – mit dem höchsten Q-Wert assoziiert ist.

Analog des referierten Beispieles der Wegfindung lässt sich auch in digitalen Brettspielen Lernverhalten realisieren, wenngleich der äußerst große Zustandsraum, der hohe Verzweigungsfaktor der betrachteten Brettspiele und die endliche Speicherkapazität das Q-Lernverfahren einschränken und erweiterte Mechanismen erfordern, um zeitnahe Entscheidungsfindung zu ermöglichen. Wie sich das Q-Lernverfahren in spielübergreifender künstlicher Intelligenz auf Grundlage von Monte Carlo Verfahren einsetzen lässt, darauf sei an späterer Stelle eingegangen, wenn das implementierte Framework *SpoookyJS* besprochen wird.

Zunächst jedoch sei das Kapitel zu verstärkenden Lernverfahren abgeschlossen mit einem Blick auf das System *TD-Gammon*, das sich mithilfe des *TD*-Lernverfahrens meisterhaft im Backgammonspiel gerierte.

4.3.2.7 State of the Art: Meisterhaftes Backgammonspiel auf Grundlage verstärkenden Lernens

Als Tesauro in den frühen 1990er Jahren mit seinem selbstlernenden backgammonspielenden System *TD-Gammon* den Nachfolger von *Neurogammon* vorstellt, [383] ist die Resonanz auf *TD-Gammon* ob der Spielstärke des Computerprogrammes allzumeist durch Adjektive höchst eindrücklichen Wohlgefallens bestimmt. So berichten Russell und Norvig, das Programm *TD-Gammon* demonstriere „das Potenzial verstärkender Lerntechniken auf beeindruckende Weise“[384] und so beschreibt Nilsson Tesauros *TD-Gammon* als „[o]ne of the most impressive demonstrations of the power of machine-learning methods.“[385] In seiner 2002 publizierten Arbeit[386] weist Tesauro gar auf einen Paradigmenwechsel im menschlichen Backgammonspiel hin: Einige Zugentscheidungen, die das Programm hervorbrachte, offenbarten sich dem Autor zufolge als innovative Spielentscheidungen, die von menschlichen Spielern zunächst nicht in Betracht gezogen und nun übernommen wurden – nicht mehr lernt das Computerprogramm von menschlicher Expertise, sondern orientieren sich menschliche Spieler an den Entscheidungen eines Computerprogramms.[387]

Der begeisterte Zuspruch, den *TD-Gammon* erfährt, schöpft sich aus dem Ansatz, den das Programm verfolgt und umsetzt. Realisiert als dreischichtiges Perzeptron mit 198 Eingabeneuronen, 40 versteckten Neuronen und vier Ausgabeeinheiten, trainierte sich das künstliche neuronale Netz in mehreren Hunderttausend Spielen gegen sich selbst und erreichte nach einer weiteren Iteration in der Entwicklung des Programmes meisterhafte Spielstärke, die aus der Integration spielspezifischer

[383] Vgl. u.a. Tesauro 1994, Tesauro 1995 und Tesauro 2002.
[384] Russell und Norvig 2012, S. 982.
[385] Nilsson 2010, S. 420.
[386] Vgl. Tesauro 2002.
[387] Vgl. Tesauro 2002, S. 182f.

Merkmale, aus vorausschauendem Spiel [388] und aus rund 1.5 Millionen Trainingsspielen resultierte.[389] Bemerkenswert ist dabei die Ausgangssituation des Programmes, die kein Expertenwissen um das Backgammonspiel vorsah, wie Tesauro näher ausführt:

> „TD-Gammon achieves a surprisingly strong level of play. With zero knowledge built in at the start of learning (i.e., given only a 'raw' description of the board state), the network learns to play at a strong intermediate level. Furthermore, when a set of handcrafted features is added to the network's input representation, the result is a truly staggering level of performance: the latest version of TD-Gammon is now estimated to play at a strong master level that is extremely close to the world's best human players."[390]

Wie bei den zuvor betrachteten KI-Implementierungen *Chinook*, *Deep Blue* und *Stockfish*, basiert die Entscheidungsfindung auch bei *TD-Gammon* auf einer Evaluierungsfunktion, die Spielkonfigurationen hinsichtlich ihres Nutzens bewertet. Anders jedoch als die vorgestellten Implementationen lernt *TD-Gammon* die Evaluierungsfunktion durch wiederholtes Spielen gegen sich selbst, indem die Positionen der Spielfiguren auf dem Spielbrett als Eingabevektoren in das mehrschichtige Perzeptron eingespeist und durch das neuronale Netz propagiert werden: Jedem Eingabemuster w_t ist ein Ausgabevektor Y_t zugeordnet, der die erwartete Auszahlung für das Eingabemuster w_t repräsentiert und die vier finiten Spielzustände (1) *Weiß gewinnt einfach* (2) *oder mit einem Gammon* und (3) *Schwarz gewinnt einfach* (4) *oder mit einem Gammon* neuronal abbildet. Mit jedem Zeitschritt t werden die Gewichtungen des mehrschichtigen Perzeptrons nach dem $TD(\lambda)$ Algorithmus angepasst:[391]

$$w_{t+1} - w_t = \alpha(Y_{t+1} - Y_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w Y_k \tag{4.25}$$

Hierbei bezeichnet die Konstante α die Lernrate, w den Gewichtungsvektor, $w_{t+1} - w_t$ signifiziert die Gewichtungsänderung von w_t nach w_{t+1} und $Y_{t+1} - Y_t$

[388] Verwendet wurden drei Halbzüge (*ply*), vgl. Szita 2012, S. 543.
[389] Vgl. Tesauro 2002, S. 182.
[390] Tesauro 1994, S. 215.
[391] Vgl. Tesauro 2002, S. 186.

die Auszahlungsdifferenz zwischen aktueller und vergangener Spielkonfiguration. Der mit dem Nabla-Operator eingeleitete Ausdruck $\nabla_w Y_k$ bezeichnet den Gradienten des TD-Fehlers unter Berücksichtigung der Gewichtungen. Wie im vorangegangenen Kapitel ausgeführt, beeinflusst die Konstante λ mit $\lambda \in [0,1]$ den temporalen Aspekt der Fehlerrückführung.[392] Am Ende jeder Spielpartie dient das Belohnungssignal z dem Zweck, die Gewichtungen erneut anzupassen. So wird in den oben wiedergegebenen Ausdruck anstelle der Differenz $Y_{t+1} - Y_t$ die Differenz $z - Y_f$ eingesetzt, um die Gewichtungsänderungen zu bestimmen. Das Signal z setzt sich hierbei zusammen aus den oben wiedergegebenen vier finiten Spielzuständen.[393]

Der Trainingsprozess beginnt mit zufällig initialisierten Gewichtungen zwischen den Elementen des neuronalen Netzes und nimmt zu jedem Zeitpunkt der Spielpartie eine Bewertung aller möglichen Spielzüge vor. Ausgewählt wird anschließend der vielversprechendste Zug, der wiederum die Gewichtungen nach oben wiedergegebener Logik beeinflusst. Nach mehr als sechs Millionen absolvierter Trainingsspiele gelang es der optimierten Version 3.1 von *TD-Gammon*, gute Spielentscheidungen unter Verwendung eines mit 400 MHz getakteten *Pentium II* in weniger als 12 Sekunden zu treffen.[394]

4.3.2.8 Verstärkendes Lernen in unterschiedlichen Brettspieldomänen

Mit den kommerziellen Backgammonprogrammen *eXtreme Gammon*, [395] *Snowie*, [396] *Bgblitz* [397] oder dem kostenfreien und quelloffenen *GNU Backgammon* [398] existieren zahlreiche, in Wettbewerben äußerst erfolgreiche, Umsetzungen der Mechanismen, die in den vorangegangenen Kapiteln referiert wurden. Wie Szita berichtet, lässt sich der Erfolg von *TD-Gammon* und nachfolgenden Implementationen als Katalysator in der Beschäftigung mit

[392] Tesauro weist darauf hin, dass der Wert für λ empirisch-experimentell bestimmt werden müsse. Wurde in früheren *TD-Gammon* Versionen ein Wert von $\lambda = 0.7$ verwendet, arbeiteten spätere Modifikationen des Programmes häufig mit $\lambda = 0$ (vgl. Tesauro 2002, S. 186).
[393] Vgl. Tesauro 2002, S. 186.
[394] Vgl. Tesauro 2002, S. 189.
[395] Vgl. http://www.extremegammon.com (zuletzt erfolgreich aufgerufen am 13.12.2013).
[396] Vgl. http://www.bgsnowie.com (zuletzt erfolgreich aufgerufen am 13.12.2013).
[397] Vgl. http://www.bgblitz.com (zuletzt erfolgreich aufgerufen am 13.12.2013).
[398] Vgl. http://www.gnubg.org (zuletzt erfolgreich aufgerufen am 13.12.2013).

Verfahren des verstärkenden Lernens begreifen[399] – doch warum die Anwendung des $TD(\lambda)$ Algorithmus im Rahmen des Backgammonspieles besonders gut funktionierte, offenbart sich nicht unmittelbar. So trägt Szita einige Belege zusammen, die den Erfolg von *TD-Gammon* fundierten:[400] Neben der adäquaten Repräsentation und Modellierung der Spielcharakteristika und der angemessenen Wahl des Parameters λ sei das Zufallsmoment des Spieles den Verfahren dienlich, weil es das Dilemma zwischen Exploration und Exploitation mindere.[401] Erschöpft sich das Interesse an Computerbackgammon ob der Spielstärke der Programme in partikulären Optimierungen der Verfahren, so eröffnet die Beschäftigung mit Backgammon-Varianten wie *Plakoto* oder *Fevga* und der damit verbundenen Frage, wie sich spielübergreifende schlaue Computergegner generieren lassen, neue Aspekte in der Beschäftigung mit Methoden des verstärkenden Lernens.[402]

Was sehr gut für Backgammon funktioniert, trägt bei anderen klassischen Brettspielen mitunter jedoch wenig Früchte. Zwar lassen sich mit *KnightCap*,[403] *NeuroChess*[404] oder *NeuroDraughts*[405] praktikable Umsetzungen des TD-Lernens für das Schach- und Damespiel[406] identifizieren, jedoch sind die Eigenschaften und Eigenheiten der unterschiedlichen Spiele zu vielfältig, als dass sich Verfahren des verstärkenden Lernens direkt und unverändert übertragen ließen – und existiere kein Allgemeinrezept, lasse sich keine *magische Kugel* ausfindig machen, so weist Szita auf die unterschiedlichen Probleme in der Übertragung des verstärkenden Lernens auf differierende Spieldomänen hin:

---

[399] Vgl. Szita 2012, S. 544.
[400] In seiner Vorstellung der Gründe, warum *TD-Gammon* funktionierte und funktioniere, bemüht Szita die Arbeiten von Tesauro 1995, Tesauro 1998, Pollack und Blair 1997, Ghory 2004 sowie Wiering 2010 (vgl. Szita 2012, S. 544f.).
[401] Vgl. Szita 2012, S. 544.
[402] Hingewiesen sei an dieser Stelle auf die Arbeiten von Papahristou und Refanidis, die mit ihrer Software *AnyGammon* (http://ai.uom.gr/nikpapa/AnyGammon, zuletzt erfolgreich aufgerufen am 13.12.2013) das Spiel alter Varianten und Variationen des Backgammonspieles ermöglichen (vgl. die entsprechenden Aufsätze: Papahristou und Refanidis 2012 sowie Papahristou und Refanidis: AnyGammon: Playing Backgammon Variants Using Any Board Size 2013. Online verfügbar unter http://www.fdg2013.org/program/festival/anygammon.pdf, zuletzt geprüft am 06.10.2013).
[403] Baxter et al. 1998 und Baxter et al. 2000.
[404] Thrun 1995.
[405] Lynch und Griffith 1997. Vgl. auch die Erweiterung von *NeuroDraughts* um ein verfeinertes Modul zur Suche im Spielbaum: Caixeta und Silva Julia, Rita Maria 2008.
[406] Ein Dame-spielendes System, das mit TD-Lernen verfährt und sich Spielwissen in Form einer Datenbank zu Nutze macht beschreiben Patist und Wiering 2004.

> „In any application of RL [Reinforcement Learning], the choice of algorithm is just one among many factors that determine success or failure. Oftentimes the choice of algorithm is not even the most significant factor: the choice of representation, formalization, the encoding of domain knowledge, additional heuristics and variations, proper setting of parameters can all have great influence. For each of these issues, we can find exciting ideas that have been developed for conquering specific games. Sadly (but not surprisingly), there is no 'magic bullett': all approaches are more-or-less game- or genre-specific."[407]

Besonders im Schachspiel, in dem ein geschickt agierender artifizieller Computerspieler Materialwerte berücksichtigen, Muster erkennen und gegnerische Strategien antizipieren muss, gelangen Ansätze des verstärkenden Lernens an ihre Grenzen, offenbaren sich jedoch als dienlich, um die Materialwerte der Spielfiguren und die Relevanz der verschiedenen Spielfelder für die einzelnen Spielfiguren zu lernen.[408]

[407] Szita 2012, S. 540.
[408] Vgl. Droste 2008, Droste und Fürnkranz 2008 sowie Szita 2012, S. 549f.

4.3.3 Zusammenfassung und Ausblick

Die vorangegangenen Kapitel referierten eine Auswahl von Methoden, die spielstarke artifizielle Gegner in digitalen Brettspielen hervorzubringen vermögen und sich in den als *Computational Intelligence* überschriebenen Forschungs- und Methodenkomplex einfügen. Allen betrachteten Verfahren gemein ist ihr einendes zentrales Konzept: das Moment des Lernens. Schöpfen künstliche neuronale Netze aus unüberwacht oder überwacht verfahrenden Lernmethoden ihre Fähigkeit, adaptiv auf unterschiedliche Anforderungen einzugehen, so liegt den referierten verstärkenden Lernmechanismen das elaborierte Konzept des Agenten zugrunde, der die individuelle Güte seiner Handlungen durch Erfahrungen mit und in seiner Umwelt iterativ – Erfahrungsepisode für Erfahrungsepisode – identifiziert. Im Unterschied zum Minimax-Algorithmus und seinen Erweiterungen, lernen die vorgetragenen Ansätze Spielmuster und auszahlungsoptimierende Spielmerkmale – und identifizieren erfolgsversprechende Zugmöglichkeiten expertenunabhängig durch Training auf Grundlage von Umweltfeedback.

Ob *TD-Gammon*, *Chinook* oder *Deep Blue* – eines ist den Implementationen gemein, die in den vorangegangenen Kapiteln vorgestellt wurden: Zwar vermögen die Systeme, meisterhaft zu spielen, dies jedoch einzig und allein in ihrer spezifischen Spieldomäne, dem Backgammon-, Dame oder Schachspiel. Die Herausforderung, die sich mit der vorliegenden Arbeit formuliert, ist die der spielübergreifenden künstlichen Intelligenz: Ist es möglich, ein Spieleframework zu entwickeln und bereitzustellen, das sowohl die Erstellung unterschiedlichster (Brett)spiele – Spiele mit perfekter Information, Spiele mit Zufallsmomenten, Spiele mit versteckter Information – mit einfachen Mitteln ermöglicht als auch künstliche Intelligenzen hervorzubringen vermag, die des Spielens in unterschiedlichen Spieldomänen ohne menschliche Expertise mächtig sind.

Mit seinem plattformunabhängigen multiagentenbasierten Ansatz, der verschiedene Agenten im Agentenkollektiv individuelle Aspekte des jeweiligen Spieles und der jeweiligen Spielsituation auf Grundlage der Monte Carlo Spielbaumsuche und des Q-Lernalgorithmus‘ analysieren und lernen lässt, nimmt

sich das im Rahmen der vorliegenden Ausarbeitung entwickelte Framework *SpoookyJS* jener Herausforderung an, spielübergreifende künstliche Intelligenz zu realisieren. Wie sich webbasierte Spiele mit elementaren Kenntnissen der Programmiersprache JavaScript erstellen lassen und wie sich die Entscheidungsfindung im Agentenensemble gestaltet, darauf sei in Kapitel 6 ausführlich eingegangen, wenn das Framework *SpoookyJS* vorgestellt wird. Im nächsten Kapitel werden mit dem Forschungskomplex des *General Game Playing* zunächst Methoden und Implementationen spielübergreifender künstlicher Intelligenzen vorgetragen und diskutiert, um *SpoookyJS* im Forschungskomplex zu verorten und den eigenen Ansatz gegenüber dem Ansatz des *General Game Playing* herauszustellen.

5 Spielübergreifende künstliche Intelligenz(en) – General Game Playing

> „The successes of many AI systems are due not to their own intelligence but to the intelligence of their designers. He or she gives them a method to solve the given problem or instructs them how to perform an efficient combinatorial search for this problem: he or she spoonfeeds them. As these systems are unable to find themselves a good method for other problems, they lack generality, an essential part of intelligence. Is it possible to call such systems "intelligent," [sic] for instance a chess program that defeats the world champion but is unable to play any other game?"[409]

Schachcomputer wie *Deep Blue* oder das *Dame*-spielende Computerprogramm *Chinook* gerieren sich als hochspezialisierte Systeme, die einzig in ihrer dedizierten Spieldomäne zu agieren und zu bestehen vermögen. Mit seiner Grundfrage, ob sich Softwaresysteme entwickeln lassen, die des Spielens unterschiedlichster Spiele fähig sind – und wie jene Softwaresysteme beschaffen sein sollten, um hocheffizientes spielübergreifendes Spielverhalten zu realisieren –, verschiebt der Forschungskomplex des *General Game Playing* den Beschäftigungsfokus von äußerst ressourcenfordernden Systemen auf Softwarelösungen, die Algorithmen wie die Monte Carlo Spielbaumsuche einsetzen, um spielübergreifende artifizielle Computergegner zu realisieren.

Die nachfolgenden Kapitel stellen Möglichkeiten und Herausforderungen in der Umsetzung universeller Spielsysteme vor. Einleitend mit dem 1998 veröffentlichten kommerziellen System *Zillions of Games* wird in den Forschungskomplex des General Game Playing eingeführt, in dem das Framework *SpoookyJS* abschließend verortet wird.

[409] Pitrat 1995, S. 349

5.1 Zillions of Games

Auf ihrer Homepage zillions-of-games.com beschreiben die Entwickler der 1998 veröffentlichten und bis 2003 weiterentwickelten kommerziellen Software *Zillions of Games*, im Folgenden mit *ZoG* abgekürzt, ihre Softwarelösung begeistert als

> „Board gamers rejoice! Zillions Development is changing the rules... Zillions technology incorporates a **universal gaming engine** that can intelligently play any board game read in. Advanced AI techniques borrowed from the field of computer chess allow Zillions to look far ahead, calculating a best move. You can think up a new game and Zillions will immediately battle wits with you. With Zillions of Games, you can create your own games! […] The ultimate opponent. Plays **over 350** board games and puzzles! Match wits against the computer or play & chat over the Internet. Expandable -- download new games from the Internet or invent your own.! [sic]“[410]

Präsentiert die Anwendung[411] auf ihrem Startbildschirm, wie in der nachfolgenden Abbildung dargestellt, noch eine überschaubare Vielfalt von Spielen, so offenbaren sich nach einem Klick auf die jeweiligen Spielsymbole mehr als 300 Spiele[412] und Spielvarianten wie Puzzlespiele, Brettspiele (u.a. Schach, Dame) und m,n,k-Spiele wie Tic Tac Toe oder Gomoku.

[410] www.zillions-of-games.com/aboutus.html (zuletzt erfolgreich aufgerufen am 16.08.2014, die Hervorhebungen wurden von der Webseite übernommen).

[411] Die zum Zeitpunkt der vorliegenden Ausarbeitung aktuelle Version der Software trägt die Versionsnummer 2.0.1p und ist datiert auf das Jahr 2003.

[412] Auf ihrer Website verzeichnen die Entwickler 375 Spiele und ihre Variationen (http://www.zillions-of-games.com/games.html, zuletzt erfolgreich aufgerufen am 18.08.2014); die Moduldatenbank unter http://www.zillions-of-games.com/cgi-bin/zilligames/submissions.cgi, zuletzt erfolgreich aufgerufen am 18.08.2014, verzeichnet mehr als 2000 Spiele und Spielvarianten, die von Benutzerinnen und Benutzern der Anwendung erstellt wurden.

Abbildung 44: Screenshot des *Zillions of Games* Startbildschirms.

Die Flexibilität in der Umsetzung von Spielen und ihren Varianten resultiert aus der strikten Separation von Spiel-Engine[413] und Spielbeschreibung: Spiele werden in einer *Lisp*-ähnlichen proprietären Sprache in ASCII-Dateien mit der Dateiendung *zrf*[414] beschrieben und von der Spiel-Engine interpretiert.[415] Die Spiel-Engine leistet zum einen die Darstellung der Spielwelt sowie der Spielfiguren und realisiert zum anderen die Interaktionsmöglichkeiten mit den Spielbstandteilen, wie sie sich in der Regeldatei formuliert finden. Eine Regeldatei besteht aus vier Hauptbestandteilen: Der Beschreibung des Spielbrettes und der Spielfiguren, der Deklaration der Spielregeln sowie einem Informationsteil, durch den sich Spiel- und Strategiehinweise mitteilen lassen. Veranschaulicht sei der Aufbau der

[413] Mit Gregory sei eine Spiel-Engine (*Game Engine*) verstanden als „software that is extensible and can be used as the foundation for many different games without major modification“ (Gregory 2009, S. 11).

[414] Mit der Dateiendung *zrf* ist die Bezeichnung der Regeldatei als *Zillions Rule File* abgekürzt.

[415] Die 1958 von John McCarthy spezifizierte Programmiersprache *LISP* (für *LISt Processor*) und ihre Sprachderivate waren besonders in den 1970er und 1980er Gegenstand und Arbeitsmittel der Forschung um künstliche Intelligenz; das abnehmende Interesse an der Sprache in der Mitte der 1980er Jahre sieht Kurzweil in der Nichterfüllung der hohen Erwartungen begründet, die sich mit *LISP* verbanden: „LISP became the rage in the artificial intelligence community in the 1970s and early 1980s. The conceit of the LISP enthusiasts of the earlier decade was that the language mirrored the way the human brain worked – that any intelligent process could most easily and efficiently be coded in LISP. There followed a mini-boomlet in ‘artificial intelligence‘ companies that offered LISP interpreters and related LISP products, but when it became apparent in the mid-1980s that LISP itself was not a shortcut to creating intelligent processes, the investment balloon collapsed“ (Kurzweil 2013, S. 154).

Regeldatei durch den folgenden Quelltext, mit dem das Tic Tac Toe Spiel umgesetzt ist:[416]

```
1     (game
2       (title "Tic-Tac-Toe")
3       (description "One side takes X's and the other side
            takes O's. Players alternate placing their marks
            on open spots. [...]")
4       (history "Tic-Tac-Toe was an old adaptation of
            Three Men's Morris to situations where there were
            no available pieces. [...]")
5       (strategy "With perfect play, Tic-Tac-Toe
            is a draw. [...]")
6       (players X O)
7       (turn-order X O)
8       (board
9         (image "images\TicTacToe\TTTbrd.bmp")
10        (grid
11          (start-rectangle 16 16 112 112) ; top-left position
12          (dimensions ;3x3
13            ("top-/middle-/bottom-" (0 112)) ; rows
14            ("left/middle/right" (112 0))) ; columns
15          (directions (n -1 0) (e 0 1) (nw -1 -1) (ne -1 1))
16        )
17      )
18      (piece
19        (name man)
20        (help "Man: drops on any empty square")
21        (image X "images\TicTacToe\TTTX.bmp"
22         O "images\TicTacToe\TTTO.bmp")
23        (drops ((verify empty?) add))
24      )
25      (board-setup
26        (X (man off 5))
27        (O (man off 5))
28      )
29      (draw-condition (X O) stalemated)
30      (win-condition (X O)
31        (or (relative-config man n man n man)
32            (relative-config man e man e man)
33            (relative-config man ne man ne man)
34            (relative-config man nw man nw man)
35        )
36      )
37    )
```

Auf die Angabe des Kopfbereiches der Spielbeschreibung, in dem sich der Titel, die Beschreibung, Geschichtliches und Hinweise zur Spielstrategie finden (vgl. die Zeilen 2 bis 5 des Quelltextes) folgt die Definition der beiden Spieler *X* und *O*. Nach Festlegung der Zugreihenfolge wird das Spielbrett und seine Darstellung

[416] Der Quelltext zur Implementation des Tic Tac Toe Spieles ist ausschnittweise wiedergegeben nach der Anwendungseigenen *Rules File Language Reference*, die über das Hilfemenu der Applikation zugänglich ist.

initialisiert (vgl. die Zeilen 9 bis 17). Der Bereich *piece* (Zeilen 18 bis 24) bestimmt das visuelle Erscheinungsbild der Spielfiguren beider Spieler und verfügt mit Zeile 23, dass Spielfiguren einzig auf leere Spielfelder abgelegt werden dürfen. Bevor die Spielbeschreibung mit der Angabe der Unentschieden- und Gewinnbedingungen (vgl. die Zeilen 29 bis 36) abgeschlossen wird, ist mit dem *board-setup* angegeben, über wie viele Spielsteine die jeweiligen Spieler zu Beginn des Spieles verfügen (Zeilen 25 bis 28). Interpretiert durch die Spiel-Engine präsentiert sich das Tic Tac Toe Spiel nach obiger Spieldefinition wie mit dem nachfolgenden Bildschirmausschnitt veranschaulicht:

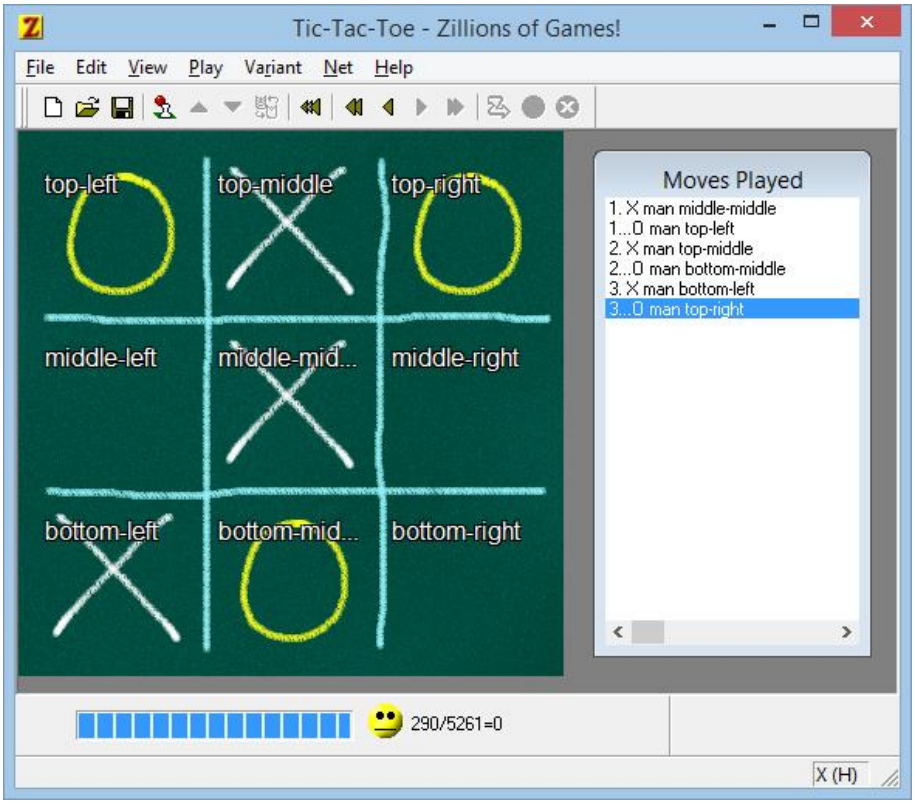


Abbildung 45: Screenshot einer Tic Tac Toe Partie in *Zillions of Games*.

Als General Game Playing System beschränkt sich *ZoG* nicht einzig darauf, Spiele zu beschreiben und darzustellen, sondern bietet spielübergreifende artifizielle Computergegner. Wie genau sich die artifizielle Entscheidungsfindung in *ZoG* gestaltet und welche Algorithmen der Zugabwägung dienen, darüber informieren die Entwickler der Software zwar begeistert, jedoch spärlich aussagekräftig. So berichtet die Startseite der Anwendungshomepage, dass *ZoG* einen „absolut einzigartigen, universellen Spielalgorithmus [verwende], mit dem fast jedes

Brettspiel oder Puzzle der Welt gespielt werden kann.“[417] In der *Häufig gestellte Fragen* Rubrik offenbart sich jener „einzigartige“ Algorithmus als klassisches Spielbaumsuchverfahren – und wird auf die Möglichkeiten und Grenzen der Software hingewiesen:

> „Zillions‘ internal AI is basically a classical, brute-force, tree-search engine. For simple games like TicTacToe, Zillions can figure out every possibility in a fraction of a second and play perfectly! Zillions plays well in games like Checkers where it can find good tactical possibilities by looking ahead. Zillions doesn't play as well in games with huge ‘branching factors‘ (large number of moves available to play) such as ‘Shogi‘ with pieces in hand. This is a well-known syndrome of game search engines that AI researchers are currently grappling with.“[418]

Mit seiner formalen Spielbeschreibungssprache und seiner Spiel-Engine bietet *ZoG* ausschließlich die Möglichkeit der Umsetzung von Spielen mit perfekter Information. Spiele mit versteckter Information wie Poker und Verbindungsspiele wie *Hex* sind mit der Software nicht realisierbar. Wenngleich keine native Unterstützung für einen oder mehrere Würfel existiert, lässt sich das Moment des Zufalls in *ZoG* durch einen Zufallsspieler abbilden, der mögliche Züge randomisiert auswählt.[419] Die Forschungsintention um universelle Spielsysteme schnürt *ZoG* in ein allzu starres Korsett: So ist es mit der kommerziellen Software nur eingeschränkt möglich, das mitgelieferte KI-Modul durch eigene Entwicklungen zu ergänzen oder das Modul zu ersetzen.[420]

---

[417] http://zillions.de, zuletzt erfolgreich aufgerufen am 18.08.2014.

[418] http://www.zillions-of-games.com/supportedFAQ.html (zuletzt erfolgreich aufgerufen am 18.08.2014). Eine Rezension der Software weist auf den Minimax-Algorithmus und das Verfahren der Alpha-Beta Kürzung hin: „Natürlich kommen Minimax und Alpha-Beta zum Einsatz, wie auch weitere Techniken, die wir von der Schachprogrammierung her kennen. Sogar Hashtabellen werden angelegt und benutzt.“ (http://zillions.de/css01.htm, zuletzt erfolgreich aufgerufen am 18.08.2014). Auf der Website wiedergegeben ist eine Rezension aus der Zeitschrift *Computerschach und Spiele*. Über das Datum der Rezensionsveröffentlichung lässt die Website ihre Besucherinnen und Besucher jedoch im Unklaren.

[419] Vgl. http://www.zillions-of-games.com/supportedFAQ.html (zuletzt erfolgreich aufgerufen am 18.08.2014).

[420] Diese Kritik üben auch Kulick et al. 2009, S. 2.

5.2 General Game Playing

General Game Playing, im Folgenden mit *GGP* abgekürzt, beschäftigt sich mit der Entwicklung von Systemen, die dazu fähig sind, die Spielregeln ihnen unbekannter Spiele zu interpretieren und jene Spiele ohne menschlichen Eingriff und ohne menschliche Expertise gut zu spielen vermögen.[421] Thielscher zufolge gestaltet sich die Herausforderung in der Entwicklung universeller Spielsysteme als Bestrebung, Systeme zu implementieren, die das Moment nichtspezialisierter umfassender (artifizieller) Intelligenz realisieren:

> „Unlike specialised systems like the chess program Deep Blue, a general game player cannot rely on algorithms designed in advance for specific games. Such a system rather requires a form of general intelligence that enables it to autonomously adapt to new and possibly radically different environments.“[422]

Die Beschäftigung mit universellen Spielsystemen nahm ihren frühen Lauf mit dem 1968 veröffentlichten Aufsatz *Realization of a general game playing program*[423] von Jacques Pitrat. In seiner Arbeit gelang es Pitrat, ein Computerprogramm zu entwickeln, das beliebige schachähnliche Spiele anhand ihrer Regeln zu lernen vermochte.[424] Mehr als zwei Jahrzehnte später beschäftigte sich Barney Pell in den frühen 1990er Jahren mit dem Forschungskomplex um universelle Spielsysteme, den Pell mit dem Begriff *Metagame* signifiziert und die Herausforderung um *Metagame* wie folgend beschreibt:

> „The idea is to write programs which take as input the rules of a set of new games within a pre-specified class. The programs compete against each other in many matches on each new game, and the programs can then be evaluated based on their overall performance and improvement through experience.“[425]

[421] Die Definition des General Game Playing ist wiedergegeben nach Schiffel und Thielscher 2014, S. 171.
[422] Thielscher 2011a, S. 26.
[423] Pitrat 1968.
[424] Vgl. Thielscher 2011a, S. 27.
[425] Barney Pell 1992, S. 2. Vgl. auch die Dissertationsschrift mit dem Titel *Strategy generation and evaluation for meta-game playing* (Pell 1993).

Pells Softwareimplementation *Metagamer* vermochte eine Vielzahl vereinfachter Schachvarianten zu spielen[426] und verwendete den iterativ vertiefenden Minimax-Algorithmus mit Alpha-Beta Kürzung, um das Spielverhalten der artifiziellen Spielgegner zu realisieren.[427] Gemein ist den Ansätzen von Pitrat und Pell die Wahl einer eigenen formalen Sprache, um bestimmte Klassen von Spielen mit perfekter Information zu beschreiben. So intendierte Pell, mit seiner Sprache *Metagame* symmetrische schachähnliche Spiele – Zweispieler-Spiele mit vollständiger Information ohne Zufallsereignisse[428] – zu beschreiben. Weitergeführt wird der Ansatz einer Spielbeschreibungssprache von Koller und Pfeffer, die in ihrer Arbeit *Representations and solutions for game-theoretic problems*[429] das System *Gala* vorstellen, das sich als Forschungsinfrastruktur verstanden wissen möchte und mit dessen systemeigener formaler Beschreibungssprache *GAme LAnguage* (*Gala*) sich Spiele den Autoren zufolge auch von Laien beschrieben ließen:

> „The Gala system can provide the infrastructure for experimental game-theoretic research. Different abstractions and variants of a game can be generated easily and solved efficiently. Gala therefore provides a convenient tool for testing and evaluating different approaches for solving games. Perhaps more importantly, Gala can play a crucial role in making game-theoretic reasoning more accessible to people and computer systems. The system allows complex games to be described simply, in a language even a layperson (one who is not a game-theory expert) can use.“[430]

Verhielt sich das Forschungsinteresse im referierten Zeitraum von rund 30 Jahren mit wenigen themenrelevanten Publikationen zunächst sehr zögerlich,[431] so findet sich die Beschäftigung mit spielübergreifender künstlicher Intelligenz besonders seit dem Jahr 2005 stark belebt: Um die Arbeit in jenem Forschungskomplex

[426] Vgl. Finnsson und Björnsson 2008, S. 260.
[427] Vgl. Pell 1996, S. 150.
[428] Vgl. Kulick et al. 2009, S. 2.
[429] Koller und Pfeffer 1997.
[430] Koller und Pfeffer 1997, S. 213.
[431] Thielscher weist darauf hin, dass die Beschreibungssprache *Gala* keine Verwendung außerhalb des von entwickelten Systems fand (vgl. Thielscher 2011b, S. 1). Einen ergänzenden Überblick über weitere frühe universelle Spielsysteme wie das *Search and Learning* System *SAL* (Gherrity 1993), *Hoyle* (u.a. Epstein 1994), *Morph* (Levinson und Snyder 1991) und *Morph II* (Levinson 1994) bieten Mańdziuk et al. 2013, S. 17–19.

anzuregen, der nun unter dem Begriff *General Game Playing* firmiert, lobt die *American Association for Artificial Intelligence* (*AAAI*) einen mit 10.000 US-Dollar dotierten Preis aus, der seit 2005 jährlich an das bestspielende *GGP*-System vergeben wird.[432] Mit den nachfolgenden Kapiteln sei zuerst über die Wettbewerbs- und *GGP*-eigene *Game Description Language* informiert, die als formale Spielbeschreibungssprache die Grundlage des Wettbewerbes darstellt; daran anschließend werden mit den Softwaresystemen *Cluneplayer*, *Fluxplayer* und *Cadiaplayer* eine Auswahl der Gewinner der in den vergangenen Jahren ausgetragenen Wettbewerben vorgestellt, die unterschiedliche Ansätze in der Realisierung spielübergreifender künstlicher Intelligenz verfolgen.

5.2.1 Game Description Language

Die *Game Description Language* (*GDL*), zuweilen auch als *Game Definition Language* signifiziert,[433] ist eine formale logikbasierte, an *Datalog* angelehnte Sprache, mit der sich diskrete Spiele mit vollständiger Information definieren lassen.[434] Zustände der Spielwelt werden in der GDL mit Fakten und die Zustandsübergänge des Spieles mit logischen Regeln beschrieben;[435] die Syntax der GDL basiert auf dem *Knowledge Interchange Format* (*KIF*). Um Spiele zu beschreiben, stehen die Grundbegriffe und Relationen `role`, `true`, `init`, `next`, `legal`, `does`, `goal` und `terminal` zur Verfügung, die im Folgenden anhand der GDL-Beschreibung des Tic Tac Toe Spieles erläutert werden.[436]

5.2.1.1 Praxisbeispiel – Tic Tac Toe in der Game Description Language

Tic Tac Toe wird von den Spielern `x` und `o` gespielt, die in der GDL durch die `role` Beziehung nach folgender Syntax ausgedrückt und repräsentiert werden:[437]

```
(role x)
```

[432] Vgl. Genesereth et al. 2005, S. 63.
[433] Vgl. Genesereth et al. 2005, S. 65.
[434] Vgl. Genesereth et al. 2005, S. 65.
[435] Vgl. Love et al. 2008, S. 2.
[436] Die Beschreibung und Definition des Tic Tac Toe Spieles ist wiedergegeben nach Love et al. 2008, S. 8–11. Für den vollständigen Quelltext einer Tic Tac Toe Beschreibung sei an dieser Stelle auf Anhang TODO verwiesen.
[437] Um die Übersichtlichkeit zu erhöhen und den nonlinearen Aufbau der Spielbeschreibung zu verdeutlichen, wurde von der Zeilennummerierung abgesehen.

```
(role o)
```

Anschließend wird der initiale Zustand des Spieles und der Spielwelt durch die `init` Relation ausgedrückt. Mit dem folgenden Quelltextausschnitt wird zuerst Spieler `x` anhand der Funktionskonstante `control` als Startspieler gesetzt und anschließend jedes Feld der zweidimensionalen Spielwelt geleert (das Zeichen `b` steht hierbei für `blank`):

```
(init (control x))
(init (cell 1 1 b))
(init (cell 1 2 b))
(init (cell 1 3 b))
[...]
(init (cell 3 3 b))
```

Mit der `next` Beziehung wird auf einen Sachverhalt Bezug genommen, der im nächsten Spielzustand, d.h. in der nächsten Spielrunde, wahr ist. Die nachfolgenden Ausdrücke bestimmen, dass Spieler `x` in der nächsten Spielrunde an der Reihe ist, wenn Spieler *o* im aktuellen Spielzustand am Zuge ist et vice versa:

```
(<= (next (control x))
    (true (control o)))
(<= (next (control o))
    (true (control x)))
```

Über welche Zugmöglichkeiten die Spieler verfügen, ist über die `legal` Relation repräsentiert. So wird mit `legal(spieler, zug)` eine Zugmöglichkeit für einen Spieler bestimmt. Im Tic Tac Toe Spiel dürfen die Spieler einen Spielstein auf ein leeres Spielfeld ablegen, wenn sie im aktuellen Spielzustand an der Reihe sind:

```
(<= (legal ?player (mark ?x ?y))
    (true (cell ?x ?y b))
    (true (control ?player)))
```

Spieler, die im aktuellen Spielzustand nicht am Zuge sind, führen die Pseudoaktion `noop` aus:

```
(<= (legal x noop)
```

```
    (true (control o)))
(<= (legal o noop)
    (true (control x)))
```

Mit der `does` Relation in der Form `does(spieler, zug)` wird die aktivierte Zugmöglichkeit eines Spielers abgebildet. Vollführt ein Spieler einen Zug auf ein leeres Spielfeld der zweidimensionalen Spielwelt, wird das Spielsymbol des Spielers auf dem entsprechenden Spielfeld abgelegt:

```
(<= (next (cell ?x ?y ?player))
    (does ?player (mark ?x ?y)))
```

Mit den folgenden Rahmenkriterien wird definiert, dass die aktivierte Zugmöglichkeit keinen Einfluss auf andere Spielweltbereiche hat:

```
(<= (next (cell ?x ?y b))
    (does ?player (mark ?m ?n))
    (true (cell ?x ?y b))
    (distinctCell ?x ?y ?m ?n))
(<= (distinctCell ?x ?y ?m ?n)
    (distinct ?x ?m))
(<= (distinctCell ?x ?y ?m ?n)
        (distinct ?y ?n))
```

Ist der Initialzustand des Spieles und sind Spieler und ihre Zugmöglichkeiten repräsentiert, werden abschließend die Spielziele anhand der `goal` Relation abgebildet. In der Formulierung von Spielzielen bietet die GDL die Möglichkeit der Gewichtung von Spielzielen im ganzzahligen Intervall von 0 bis 100. So wird mit dem folgenden Ausdruck der Spielzustand als hochgewichtetes Spielziel definiert, in dem ein Spieler alle Spielfelder einer horizontalen Reihe besetzt hat:

```
(<= (goal ?player 100)
    (line ?player))
```

Um den Ausdruck `line` verwenden zu können, muss jener zuerst definiert werden:

```
(<= (row ?x ?player)
      (true (cell ?x 1 ?player))
```

```
      (true (cell ?x 2 ?player))
      (true (cell ?x 3 ?player)))
(<= (line ?player) (row ?x ?player))
```

Schließlich wird der Endzustand des Spieles über die `terminal` Relation abgebildet: Das Tic Tac Toe Spiel ist beendet, wenn ein Spieler es vermochte, drei Felder der Spielwelt in horizontaler Folge zu belegen oder wenn keine Zugmöglichkeit mehr besteht, wie mit dem nachfolgenden Quelltext abgebildet.

```
(<= terminal
    (role ?player)
    (line ?player))
(<= terminal
    (not open))
```

5.2.1.2 GDL und das Moment des Zufalls – GDL-II

In ihrer ursprünglichen Version, wie sie im vorangegangenen Kapitel am Beispiel des Tic Tac Toe Spieles vorgestellt wurde, ermöglicht die *Game Description Language* (*GDL*) die Beschreibung rundenbasierter deterministischer Spiele mit vollständiger Information ohne stochastische Ereignisse.[438] Um die GDL um das Moment des Zufalls zu erweitern, schlagen Kulick et al. in ihrer Arbeit *General Game Playing mit stochastischen Spielen*[439] vor, die Sprache um die Relation `random` zu ergänzen. Jene Relation folgt der Syntax `random <name> <value> <expression>`, bei der `name` als Bezeichner der Zufallsvariable fungiert. Als Parameter `value` wird eine ganze Zahl angegeben, aus der sich die Wahrscheinlichkeit für das Zufallsereignis konstituiert. Der dritte Parameter `expression` beschreibt das Ereignis, das „in Abhängigkeit eines Zufallsexperiments auftreten kann.“[440] Als Beispiel führen die Autoren den Wurf eines sechsseitigen Würfels an:

```
(random dice 1 1)
(random dice 1 2)
```

[438] Vgl. Kulick et al. 2009, S. 1.
[439] Kulick et al. 2009.
[440] Kulick et al. 2009, S. 4.

```
(random dice 1 3)
(random dice 1 4)
(random dice 1 5)
(random dice 1 6)
(<= (legal ?player (move dice))
      (control ?player))
```

Einen weiteren Vorschlag der Erweiterung der *GDL* um stochastische Ereignisse macht Thielscher mit seiner Arbeit *A General Game Description Language for Incomplete Information Games*[441], in der er zudem die Problematik in der formalsprachlichen Beschreibung von Spielen mit unvollkommener Information herausstellt:

> „A fundamental limitation of the existing Game Description Language (GDL), and therefore of contemporary GGP systems, is the restriction to deterministic games with complete information about the game state. While a game may involve simultaneous moves, players are immediately informed about the moves by their opponents and have complete knowledge of the game model and the current position throughout the game […]. This applies to a variety of classical games such as Chess, Go, Chinese Checkers, etc. On the other hand, the existing limitation excludes games with elements of chance like Backgammon, games with information asymmetry such as Bridge or Poker, and games which involve private communication among cooperating players like in Bughouse Chess, or in the form of negotiations like in Diplomacy. Moreover, envisaged applications for General Game Playing systems, like automated trading agents, are usually characterised by imperfect information.“[442]

Unter dem Namen *GDL-II* schlägt Thielscher die Erweiterung der GDL um die zwei Sprachelemente `sees` und `random` vor. Mit seiner Syntax `sees(player, percept)` determiniert der Ausdruck, unter welchen Umständen der Spieler `player` Informationen über das Spielereignis `percept` erhält. Das Zufallsmoment wird in GDL-II repräsentiert durch den Ausdruck `random`, mit dem sich

[441] Thielscher 2010.
[442] Thielscher 2010, S. 994.

Zufallselemente wie Würfel oder das Mischen von Spielkarten umsetzen lassen.[443] Als Praxisbeispiel für die Verwendung der Sprachergänzungen sei im Folgenden die Beschreibung des Spieles *Krieg-Tictactoe* wiedergegeben, bei dem die beiden Spieler `xplayer` und `oplayer` nicht sehen, welche Züge ihre Gegenspieler ausgeführt haben:[444]

```
1    role(xplayer).
2    role(oplayer).
3    init(control(xplayer)).
4    init(cell(1,1,b)).
5    [...]
6    init(cell(3,3,b)).
7
8    legal(R,mark(M,N)) :- true(control(R)),
9    true(cell(M,N,Z)),
10   not true(tried(M,N)).
11   legal(xplayer,noop) :- true(control(oplayer)).
12   legal(oplayer,noop) :- true(control(xplayer)).
13
14   validmove :- does(R,mark(M,N)), true(cell(M,N,b)).
15
16   next(F) :- not validmove, true(F).
17   next(tried(M,N)) :- not validmove, does(P,mark(M,N)).
18   next(cell(M,N,x)):- validmove, does(xplayer,mark(M,N)).
19   next(cell(M,N,o)):- validmove, does(oplayer,mark(M,N)).
20   next(cell(M,N,Z)):- validmove, true(cell(M,N,Z)),
21   does(P,mark(I,J)), distinct(M,I).
22   next(cell(M,N,Z)):- validmove, true(cell(M,N,Z)),
23   does(P,mark(I,J)), distinct(N,J).
```

[443] Vgl. Thielscher 2010, S. 995.
[444] Das Spiel *Krieg-Tictactoe* ist wiedergegeben nach Schiffel und Thielscher 2011, S. 848.

```
24   next(control(oplayer)):- validmove,
25   true(control(xplayer)).
26   next(control(xplayer)):- validmove,
27   true(control(oplayer)).
28
29   sees(R,yourmove) :- not validmove,
30   true(control(R)).
31   sees(xplayer,yourmove) :- validmove,
32   true(control(oplayer)).
33   sees(oplayer,yourmove) :- validmove,
34   true(control(xplayer)).
```

In der Praxis – beispielsweise in dem ausgetragenen Wettbewerb der *General Game Playing Competition* - erfordert die GDL-II wenige Änderungen im Kommunikationsprotokoll, das dem Austausch der am Wettbewerb beteiligten Softwarekomponenten dient. In den nachfolgenden Kapiteln wird jener Wettbewerb vorgestellt, der sich als Katalysator für die Beschäftigung mit dem Forschungskomplex des *General Game Playing* signifizieren lässt – und werden mit *Cadiaplayer*, *TurboTurle* und *Sancho* die Wettbewerbsgewinner der Jahre 2010 bis 2013 vorgestellt.

5.2.2 General Game Playing Competition

Die *General Game Playing Competition* (*GGPC*), wie sie jährlich von der *American Association for Artificial Intelligence* (*AAAI*) ausgetragen wird, dient dem Vergleich der Fortschritte auf dem Gebiet universeller Spielsysteme und verläuft in zwei Runden: der Qualifikations- und der eigentlichen Wettbewerbsrunde. Stellen die teilnehmenden Softwarelösungen in der Qualifikationsrunde ihr Können in zahlreichen unterschiedlichen Spielen unter Beweis – gespielt werden in jener Runde sowohl Einzelspieler-Spiele wie die Suche nach dem Ausweg aus einem Labyrinth als auch Zweispieler-Nullsummenspiele wie Tic Tac Toe oder Schach –, so treten in der Wettbewerbsrunde die bestqualifizierten Systeme gegeneinander

an.[445] Ausgeschlossen vom Wettbewerb sind sowohl menschliche Spielerinnen und Spieler als auch Angehörige der *Stanford University*.

Die Spielbeschreibung auf Grundlage der Game Description Language stellt in der GGPC zwar ein wesentliches, nicht jedoch zentrales Moment dar. Wie in der nachfolgenden Darstellung visualisiert, ist das Softwaremodul des Spiel-Managers (*Game Manager*) das zentrale Element des Wettbewerbes:

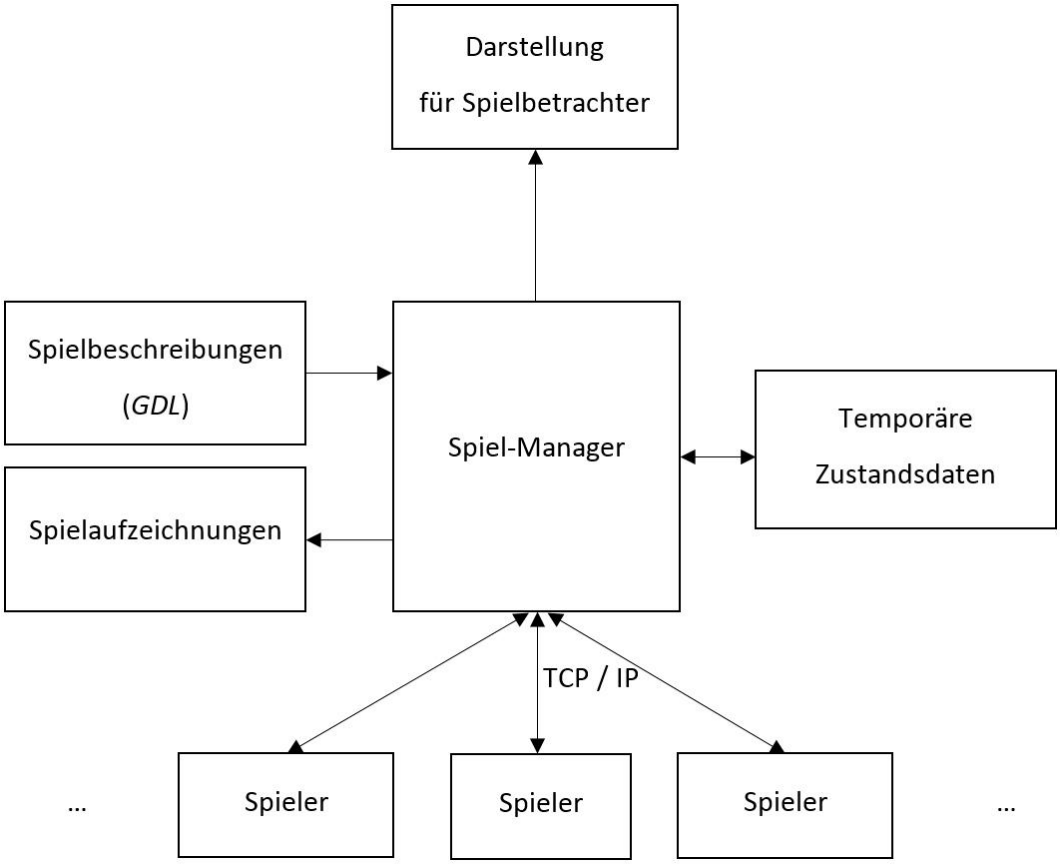


Abbildung 46: Spiel-Manager Infrastruktur im Rahmen der *General Game Playing Competition*, wiedergegeben nach Genesereth et al. 2005, S. 70.

Der Spiel-Manager verteilt zu Spielbeginn die Spielbeschreibungen an die Spieler, verwaltet den aktuellen Spielzustand, reagiert auf legale Züge der Spieler, indem er den Spielzustand aktualisiert und bestimmt den Gewinner am Ende der Spielpartie.[446] Die individuellen Spielsysteme (d.h. die Spieler des Spieles) kommunizieren mit dem Spiel-Manager durch den Austausch von Nachrichten, die

[445] Vgl. Genesereth et al. 2005, S. 71.
[446] Vgl. Genesereth et al. 2005, S. 69.

dem Hypertext Transfer Protocol (*HTTP*) folgen.[447] Weil die Spielsysteme ihre Spielbeschreibungen erst unmittelbar vor Beginn der Spielpartie erhalten, sind die erfolgreichen Softwarelösungen wie *Cluneplayer*, *Fluxplayer* oder *Cadiaplayer* darauf angewiesen, adaptiv und effizient auf die Spielgegebenheiten zu reagieren. In seinem Verlauf über mehrere Jahre verfolgten die bestspielenden Systeme, wie sie in den nachfolgenden Kapiteln gesondert vorgestellt werden, zwei unterschiedliche Ansätze: Intendierten die ersten Wettbewerbsgewinner *Cluneplayer* (2005) und *Fluxplayer* (2006), spielspezifische Eigenschaften zu extrahieren und für die eigene Evaluierungsfunktion zu nutzen, arbeiteten die Gewinner der anschließenden Wettbewerbsjahre 2007 bis 2014 mit Ansätzen, die auf Monte Carlo Simulationen basieren.[448]

5.2.2.1 Cluneplayer

Die erste General Game Playing Competition, ausgetragen im Jahr 2005, entschied die Software *Cluneplayer* für sich. *Cluneplayer* analysierte eigenständig die Spielregeln, um auszahlungsoptimierende Spieleigenschaften zu extrahieren:

> „The core aspects of the game that we model are the expected payoff, the control (or relative mobility), and the expected game termination (or game longevity). These core game aspects are modeled as functions of game-state features. The features must be automatically selected, and the key idea guiding this selection is stability […]. Our approach strives to evaluate states based on only the most stable features.“[449]

Die identifizierten Eigenschaften (*Features*) verwendete *Cluneplayer* dazu, um eine spielspezifische Evaluierungsfunktion automatisch zu generieren. Gemeinsam mit einer auf dem Minimax-Algorithmus basierenden Spielbaumsuche bildete die dergestalt erarbeitete Evaluierungsfunktion den Kern des Systems. Wenngleich es *Cluneplayer* nicht vermochte, für jedes Spiel eine ausreichende Zahl relevanter Eigenschaften zu identifizieren – und *Cluneplayer* seinen Wettbewerbserfolg in den darauffolgenden Jahren nicht wiederholen

[447] Für das Beispiel eines vollständigen Kommunikationsprotokolles sei an dieser Stelle verwiesen auf Anhang TODO.
[448] Vgl. Mańdziuk et al. 2013, S. 20f.
[449] Clune 2007, S. 1135.

konnte –, lässt sich *Cluneplayer* ob seines adaptiven Ansatzes als inspirierende Ausgangsarbeit signifizieren.[450]

5.2.2.2 Fluxplayer

Im Unterschied zu *Cluneplayer* wurde die Software *Fluxplayer* für jede Iteration der General Game Playing Competition beständig weiterentwickelt – und entschied den Wettbewerb im Jahr 2006 für sich.[451] *Fluxplayer* analysiert semantische Eigenschaften der Aussagen in den Spielbeschreibungen, um spielzustandsunabhängige Strukturen zu identifizieren, die in der Evaluierungsfunktion zur Abwägung der Güte einer Zugmöglichkeit eingesetzt werden:

> „The idea is to detect structures in the game description which can be used for non-binary evaluations like successor relations, order relations, quantities or game boards. [...] we exploit semantical properties of the predicates to detect static structures, which are independent of the state of the game.“[452]

Die Spielbaumsuche von *Fluxplayer* gestaltet sich über die Alpha-Beta Suche und verwendet Zugumstellungstabellen sowie die *History Heuristic*, um den Auszahlungswert eines Spielzustandes zu bestimmen.[453] In Spielen mit hohem Zustandsraum bestimmt *Fluxplayer* anhand unscharfer Logik (*Fuzzylogik*), inwieweit ein Spielzustand der Beschreibung einer Gewinnposition entspricht.[454]

5.2.2.3 Cadiaplayer

Mit *Cadiaplayer* finden Monte Carlo Verfahren ihren – äußerst erfolgreichen – Eingang in die General Game Playing Competition.[455] Entwickelt an der Universität von Reykjavík, vermochte *Cadiaplayer*, den Wettbewerb in den Jahren 2007, 2008

[450] Vgl. Mańdziuk et al. 2013, S. 20.
[451] Vgl. Mańdziuk et al. 2013, S. 20.
[452] Schiffel und Thielscher 2007, S. 1194.
[453] Vgl. Schiffel und Thielscher 2007, S. 1192.
[454] Vgl. Schiffel und Thielscher 2007, S. 1191 und Schiffel und Thielscher 2007, S. 1193.
[455] Sowohl die Software *Ary* (vgl. Méhat und Cazenave 2010), die den Wettbewerb in den Jahren 2009 und 2010 für sich entscheiden konnte als auch die jenseits universitärer Strukturen entwickelten Wettbewerbsgewinner *TurboTurtle* (Gewinner in den Jahren 2011 und 2013) sowie *Sancho* (Sieger im 2014 ausgetragenen Wettbewerb) arbeiten mit Monte Carlo Verfahren.

und 2012 für sich zu entscheiden.[456] Entscheidungsfindender Kernbestandteil der Softwarearchitektur von *Cadiaplayaer*, wie sie mit der nachfolgend wiedergegebenen Darstellung veranschaulicht wird, ist der *UCT Player*:

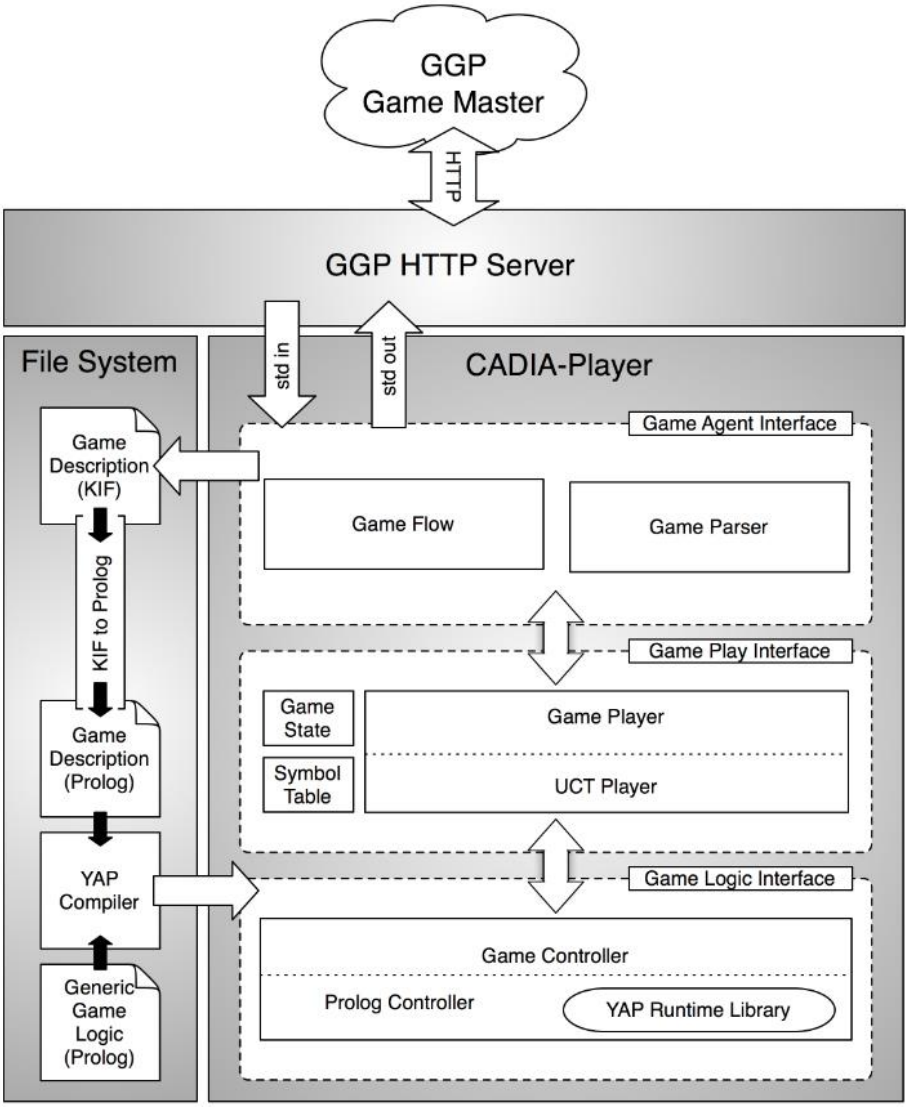


Abbildung 47: Softwarearchitektur von *Cadiaplayer*. Wiedergeben nach Björnsson und Finnsson 2009, S. 3.

Anders als die zuvor referierten erfolgreichen Implementationen trifft *Cadiaplayer* seine Zugentscheidungen nicht auf Grundlage des *Minimax*-Algorithmus mit Alpha-Beta Kürzung, sondern verwendet die Monte Carlo Spielbaumsuche mit dem *Upper Confidence Bounds applied to Trees* (*UCT*) Algorithmus: Anstatt den Spielbaum möglichst tief vorauszubestimmen, geben simulierte Zufallspartien Aufschluss über die Güte einer Zugmöglichkeit.[457] Verfuhr die *Cadiaplayer*-Version

[456] *Cadiaplayer* ging hervor aus der Masterarbeit Finnsson 2007.
[457] Vgl. Björnsson und Finnsson 2009, S. 4.

des Jahres 2007 noch zufallsbasiert in der Ausspielphase der Monte Carlo Spielbaumsuche, so verwendete die im Jahr 2008 aktive Version die *Move-Average Sampling Technique* (*MAST*).[458] Die darauffolgende Modifikation der Software, die den 2009 ausgetragenen Wettbewerb bestritt, fand sich ergänzt um die *RAVE*-Methode.[459] Forschungsbedarf sehen die Entwickler in der Frage formuliert, wie sich Spielspezifika in die Simulationskontrolle der Monte Carlo Spielbaumsuche aufnehmen lassen, um weitere Spiele und Spielkonzepte verstehen zu können.[460]

5.3 Zusammenfassung und Ausblick

Aus den vorangegangenen Betrachtungen generieren sich Möglichkeiten für die praktische Umsetzung spielübergreifender künstlicher Intelligenz und Anforderungen an die Modellierung und Implementation einer universellen Spiel-Engine, die sich dem Forschungskomplex des General Game Playing (GGP) anzugliedern intendiert. Ausgehend von der kommerziellen Software *Zillions of Games*, die sich ob ihres proprietären Charakters und mangelnder Erweiterbarkeit als frühes, dem Forschungsgebiet jedoch fernes Anwendungsexempel signifizieren lässt, wurde in den vorangegangenen Kapiteln eingeführt in die Spielbeschreibungssprache GDL und den Wettbewerb der General Game Playing Competition (GGPC).

Analog den Softwarelösungen *Clune*-, *Flux*- und *Cadiaplayer*, realisiert auch das im Rahmen der vorliegenden Ausarbeitung entwickelte JavaScript-Framework *SpoookyJS* künstliche Intelligenz in unterschiedlichen Spieldomänen ohne menschliche Expertise. Im Unterschied zu den Teilnehmern und Wettbewerbsgewinnern der GGPC versteht sich das vorgelegte Framework *SpoookyJS* als Machbarkeitsstudie jenseits des GGP und intendiert zum einen die leicht verständliche webbasierte Umsetzung von Spielen, die im Webbrowser – und somit plattformunabhängig – ausführbar sind. Zum anderen ist mit *SpoookyJS*

[458] Vgl. Finnsson und Björnsson 2008.
[459] Vgl. Finnsson und Björnsson 2011, S. 14.
[460] Vgl. Finnsson und Björnsson 2011, S. 15.

beabsichtigt, ein JavaScript-Framework bereitzustellen, das sich durch eine große Wiedervendbarkeit und Erweiterbarkeit auszeichnet, die differierenden Anwendungsfällen und Einsatzzwecken wie der universitären oder außeruniversitären Lehre dienlich ist.

Die folgenden Kapitel referieren zunächst die Systemarchitektur von *SpoookyJS* und leiten anhand des Schachspiels ein in die praktische Arbeit mit dem Framework. Anschließend wird auf die Umsetzung spielübergreifender künstlicher Intelligenz mit *SpoookyJS* eingegangen, in dessen Rahmen ein multiagentenbasierter evolutionärer Ansatz verfolgt und die Monte Carlo Spielbaumsuche mit dem *Upper Confidence Bounds applied to Trees* Algorithmus verwendet wurde. Abschließend werden Möglichkeiten – wie die Erweiterung des Frameworks um ein Modul zum Verständnis der GDL – und Grenzen des Frameworks besprochen.

6 SpoookyJS – Ein webbasiertes universelles Spielsystem

Als Machbarkeitsstudie nimmt sich das JavaScript-Framework *SpoookyJS* drei Herausforderungen in der Umsetzung eines Spieleframeworks an: Zum einen der Herausforderung, eine universelle Spiel-Engine bereitzustellen, auf deren Grundlage sich unterschiedlichste (Brett)Spiele wie Dame, Schach, Backgammon, Gomoku oder das Spiel der Amazonen anhand elementarer Kenntnisse der leicht erlernbaren Programmiersprache JavaScript umsetzen lassen. Zum anderen nimmt sich *SpoookyJS* der Herausforderung an, eine spielübergreifende künstliche Intelligenz zu modellieren und umzusetzen, die in differierenden Spieldomänen zu agieren vermag und sich durch menschliche Expertise erweitern lässt. Realisiert als Multiagentensystem, schöpfen die artifiziellen Spielgegner von *SpoookyJS* ihr Spielvermögen aus dem Zusammenspiel verschiedener Agenten, die unterschiedliche Aspekte der Spielsituation unter Verwendung von Mechanismen wie der Monte Carlo Spielbaumsuche auf individuelle Art analysieren. Mit der Wahl von Monte Carlo Methoden zur Abwägung der Güte von Zugmöglichkeiten reagiert *SpoookyJS* auf den Erfolg von Monte Carlo Methoden im Kontext des General Game Playing. Anders als die in den vergangenen Kapitel referierten Ansätze verfolgt *SpoookyJS* einen evolutionären multiagentenbasierten Ansatz, in dem die Entscheidung über die Zugwahlen einem Agentenensemble obliegt.[461]

Die dritte Herausforderung in der Umsetzung eines spielübergreifenden Spieleframeworks formuliert sich in der Plattformunabhängigkeit, der Wiederverwendbarkeit, der Modularität und Erweiterbarkeit des JavaScript-Rahmenwerkes: Die zur Implementation von Spielen angebotenen Bestandteile von *SpoookyJS* lassen sich flexibel kombinieren und leicht durch eigens entwickelte Module ergänzen. Als vollständig JavaScript- und browserbasierte Softwarelösung ermöglicht *SpoookyJS* die Implementation von Spielen, die auf unterschiedlichsten Plattformen, Geräten und Webbrowsern ausführbar sind.

[461] Einen multiagentenbasierten Ansatz im General Game Playing verfolgen auch Kobti und Sharma 2007.

Durch Verwendung des CSS-Frameworks *Bootstrap*[462] präsentieren sich die mit *SpoookyJS* umgesetzten Spiele auch auf mobilen Endgeräten unterschiedlicher Auflösungen wie Tablets oder Smartphones ansprechend.

Der gewählte browserbasierte Ansatz bietet den großen Vorteil der Plattformunabhängigkeit, ist jedoch ob des browserimmanenten *Sandbox*-Konzeptes mit einigen Schwierigkeiten verbunden: Gegenüber maschinennahen Programmiersprachen wie *C* oder *C++*, die effiziente Manipulationen des Arbeitsspeichers sowie das Laden und Speichern großer Dateien – beispielsweise für Eröffnungs- oder Endspieldatenbanken – ermöglichen, geriert sich die Webbrowserumgebung stark eingeschränkt. So wird die Programmiersprache JavaScript in einem gesonderten Bereich des Webbrowsers interpretiert und ausgeführt – der *Sandbox* –, innerhalb jener es nur eingeschränkt möglich ist, auf das Dateisystem zuzugreifen oder prozessübergreifende Kommunikation zu leisten.

Ob der eingeschränkten Anwendungsumgebung klingt die Intention von *SpoookyJS* ressourcenfokussiert an: *SpoookyJS* realisiert künstliche Intelligenzen, die des Agierens in verschiedenartigsten Spieldomänen fähig sind und ihr Spielvermögen nicht aus höchst umfangreichen Eröffnungs- und Endspieldatenbanken oder dedizierter Computerhardware vom Umfang eines Rechenzentrums schöpfen, sondern in der eingeschränkten Browserumgebung zu bestehen vermögen. Weil der zur Verfügung stehende Arbeitsspeicher rationalisiert, der Dateizugriff beschränkt und die verfügbare Rechenzeit endlich ist, geht es dem System *SpoookyJS* und seinen Bestandteilen, wie sie in den nachfolgenden Kapiteln vorgestellt werden, nicht darum, künstliche Computerspieler zu entwerfen und zu entwickeln, die sich in ihren Spielentscheidungen perfekt und fehlerfrei gerieren – jenem perfekten Spielvermögen haben sich einige Forscherinnen und Forscher mit ihren Arbeiten bereits auf eindrückliche Art und Weise angenähert, wie in den vergangenen Kapiteln mit *TD-Gammon* im Backgammonspiel, *Deep Blue* im Schachspiel oder

[462] Das Cascading Stylesheet (CSS) Framework *Bootstrap* ist frei verfügbar auf seiner Homepage http://getbootstrap.com (zuletzt erfolgreich aufgerufen am 12.08.2014).

universellen Spielsystemen wie *Clune-*, *Flux-* oder *Cadiaplayer* angeklungen. Vielmehr skizziert *SpoookyJS* einen Weg, künstliche Intelligenz zu implementieren, die in unterschiedlichsten Spieldomänen zu spielen vermag, auf so manche Entscheidungsfrage jedoch keine Antwort bietet – eine künstliche Intelligenz, die aus mehreren künstlichen Intelligenzen komponiert ist: ein plattformunabhängiges JavaScript-basiertes brettspielendes Multiagentensystem.

Bevor in Kapitel 6.2 die Kernarchitektur des Frameworks besprochen wird, führt das nachfolgende Kapitel in die Spielumgebungen ein, die mit dem Framework *SpoookyJS* bereitgestellt werden. Anhand des Schachspieles informieren die daran anschließenden Kapitel über die praktische Erstellung von Spielen mit dem JavaScript-Framework, bevor im letzten Teil der vorliegenden Arbeit die Implementation der spielunabhängigen Spiel-Engine und spielübergreifender künstlicher Intelligenz sowie Möglichkeiten und Grenzen des Rahmenwerkes diskutiert werden.

6.1 Spiele spielen mit SpoookyJS – Ein Überblick über das Framework

Die folgenden Kapitel geben einen kurzen Überblick über Projekthomepage, Bezugsquellen und Systemanforderungen des JavaScript-Frameworks *SpoookyJS* und erläutern die verschiedenen Interface-Elemente der Spieleseiten, die der Anpassung der artifiziellen Gegner dienen und die über Spielverlauf und Details der artifiziellen Entscheidungsfindung informieren.

6.1.1 Projekthomepage und Bezugsquellen

Bereitgestellt wird das JavaScript-Framework über seine Projekthomepage http://www.spoookyjs.de, auf der sich an zentraler Stelle das aktuelle Downloadpaket von *SpoookyJS* findet. Neben dem Downloadpaket, das die lokale Erstellung eigener Spiele mit dem Rahmenwerk ermöglicht, informiert die Homepage über das Projektrepositorium, das die Webanwendung über das Versionsverwaltungssystem *Git* bereitstellt und finden sich die einzelnen Bestandteile des Frameworks auf der Website dokumentiert. Unterschiedliche

Brettspiele wie Backgammon, Schach, Dame, Gomoku, Tic Tac Toe oder das *Spiel der Amazonen*, die mit dem Framework erstellt wurden, sind auf der Homepage referenziert.

6.1.2 Systemanforderungen

In seinem Anforderungsprofil gestaltet sich *SpoookyJS* schlicht: Das Framework erfordert keine dedizierte Client-Server-Architektur, keine relationale *MySQL*-Datenbank, um digitale Brettspiele einschließlich spielübergreifender künstlicher Intelligenz zu realisieren – benötigt wird einzig ein aktueller Webbrowser, in dem die Spiele gespielt werden. Gerierte sich die Leistungsfähigkeit der browsereigenen JavaScript-Engines bis vor wenigen Jahren noch äußerst bescheiden, so präsentieren sich die getesteten Webbrowser *Safari,*[463] *Firefox*,[464] Chrome[465] und die aktuelle Version des *Internet Explorer*s[466] zum Zeitpunkt der vorliegenden Ausarbeitung mit einer guten bis sehr guten Performanz in der Verarbeitung von JavaScript-Anweisungen.

Zwar reagieren die individuellen Spieleseiten responsiv auf unterschiedliche Darstellungs- und Ausgabegeräte – um die Übersichtlichkeit der Spieldarstellung zu gewährleisten, sei an dieser Stelle die Verwendung eines Rechners oder Tablets mit einer Auflösung größer gleich 1024×768, besser: 1280×768 empfohlen, wie sie zeitgenössische Tablets (z.B. *iPad* / *iPad Air*), Netbooks, Laptops und Desktop-Rechner bieten.

6.1.3 Seitenaufbau und Interfaces

Jede Spieleseite, wie sie auf dem folgenden Screenshot beispielhaft anhand des Tic Tac Toe Spieles wiedergegeben ist, verfügt neben der oberen horizontalen Navigationsleiste mindestens über sechs Interaktionselemente, die über Spielcharakteristika informieren und die Veränderung von Spieleigenschaften ermöglichen: Spielverlauf, *Agent Log*, *Ensemble*, Spielwelt, Symbolnavigationsleiste und durch die Symbolnavigationsleiste aktivierte

[463] Entwickelt von *Apple*, geteste Version: 8.0, JavaScript-Engine *Nitro* / *JavaScriptCore*.
[464] *Mozilla*, geteste Version: 33.0 (14. Oktober 2014), JS-Engine *IonMonkey*.
[465] *Google*, geteste Version: 38.0.2125.104 m, JS-Engine *V8*.
[466] *Microsoft*, geteste Version: 11.0.9600.17351, JS-Engine *Chakra*.

Interaktionselemente wie die Darstellung der Spielbäume, die im Rahmen der Monte Carlo Spielbaumsuchverfahren der einzelnen Agenten erstellt werden.

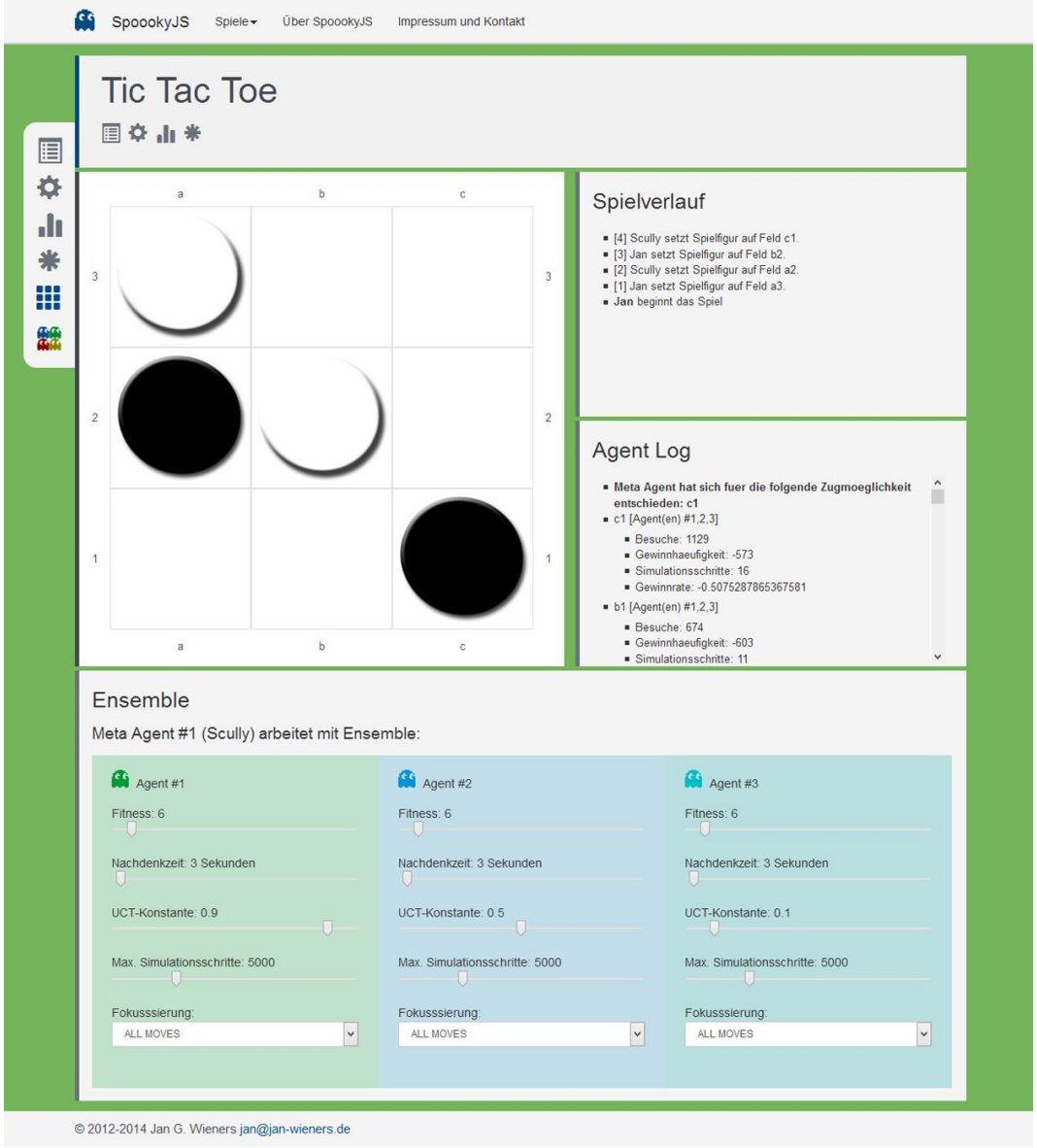


Abbildung 48: Screenshot des Tic Tac Toe Spieles, umgesetzt mit *SpoookyJS*.

Am rechten Rand jeder Spieleseite dokumentieren Spielverlauf und *Agent Log* den Lauf der Spielpartie und die Entscheidungen des Meta Agenten sowie seiner assoziierten Agenten. Unterhalb des Spielfeldes informiert das Steuerelement *Ensemble* über die einzelnen Agenten des Agentenensembles und ihre individuellen Eigenschaften:

Abbildung 49: Screenshot des Agentenensembles für das Tic Tac Toe Spiel.

Wie in den nachfolgenden Kapiteln noch ausführlich besprochen wird, gestaltet sich die spielübergreifende Entscheidungsfindung in den mit *SpoookyJS* implementierten Spielen anhand von Meta Agenten. Jeder artifizielle Spieler – d.h. jeder Meta Agent – verfügt über ein Team von Agenten, das die jeweilige Spielumgebung in ihrem aktuellen Zustand analysiert und dem assoziierten Meta Agenten Zugvorschläge unterbreitet. Die Art und Weise, wie die einzelnen Agenten ihre Spielumgebung untersuchen, ist bestimmt durch verschiedene Faktoren: Die Nachdenkzeit signifiziert die Zeit, die den einzelnen Agenten zur Entscheidungsfindung eingeräumt wird; anhand der Fokussierungsweise ist es den verschiedenen Agenten des Ensembles möglich, unterschiedliche Aspekte des Spieles eingehender zu betrachten. Eigenschaften der Monte Carlo Spielbaumsuche wie der reellzahlige Wert der UCT-Konstante und die maximale Anzahl der Simulationsschritte in den Rollouts lassen sich anhand der entsprechenden Schieberegler verändern.

Der Symbolnavigationsleiste am linken Seitenrand kommt eine wesentliche Funktion zu: Mit ihr lässt sich zwischen den verschiedenen Seitenbereichen und Interaktionselementen navigieren und lassen sich per Mouseklick oder Touch-Geste Steuerelemente einblenden, die beim initialen Aufruf der Seite zunächst verborgen werden, um die Übersichtlichkeit zu wahren. So verweisen die beiden Symbole und auf die Spielbeschreibung und Steuerelemente zur Einstellung von Spieleigenschaften. Anhand des Navigationselementes lassen sich im

oberen Bereich der Spieleseiten Statistiken zur Entscheidungsfindung des Meta Agenten einblenden:

Abbildung 50: Screenshot der Spieleseite mit eingeblendeten Statistiken.

Die zur Laufzeit der Spielpartie aktualisierten Diagramme lassen sich als Indikator für die Performanz der Entscheidungsfindung im verwendeten Webbrowser verstehen. Im zuvor wiedergegebenen Screenshot sind anhand der Diagramme zwei Spielrunden des Meta Agenten im Tic Tac Toe Spiel dargestellt. In der ersten Spielrunde hat eine menschliche Spielerin ihr weißes Spielsymbol auf einem Spielfeld abgelegt. Um eine sinnvolle Spielentscheidung zu treffen, befragt der Meta Agent sein Agentensemble nach der besten Zugmöglichkeit. Die einzelnen Agenten analysieren die Spielwelt in ihrem aktuellen Zustand mit der Monte Carlo Spielbaumsuche und simulieren eine Vielzahl von Spielen, um den bestmöglichen Zug zu identifizieren. Das Liniendiagramm, wie es oben dargestellt ist, gibt die Anzahl der simulierten Spiele (Rollouts) wieder, die in der zweiten und vierten Spielrunde von den Agenten ausgeführt wurden. Die Summe der Simulationsschritte, die virtuell durchgeführt wurden, um die simulierten Spiele zu ihrem Ende zu bringen, ist im Balkendiagramm dargestellt – so haben die drei Agenten in der vierten Runde der Tic Tac Toe Partie 4322 Spiele simuliert mit einer Gesamtzahl von 4845 Simulationsschritten.

Dienen die Symbole und der vereinfachten Navigation zu Spielwelt und Agentenensemble, so lassen sich anhand des Navigationselementes laufzeitgenerierte Spielbäume betrachten, die - bei aktiviertem Interaktionsbereich – in jeder Spielrunde dynamisch erstellt werden, in der die Entscheidungsfindung einem Meta Agenten und seinem Ensemble obliegt:

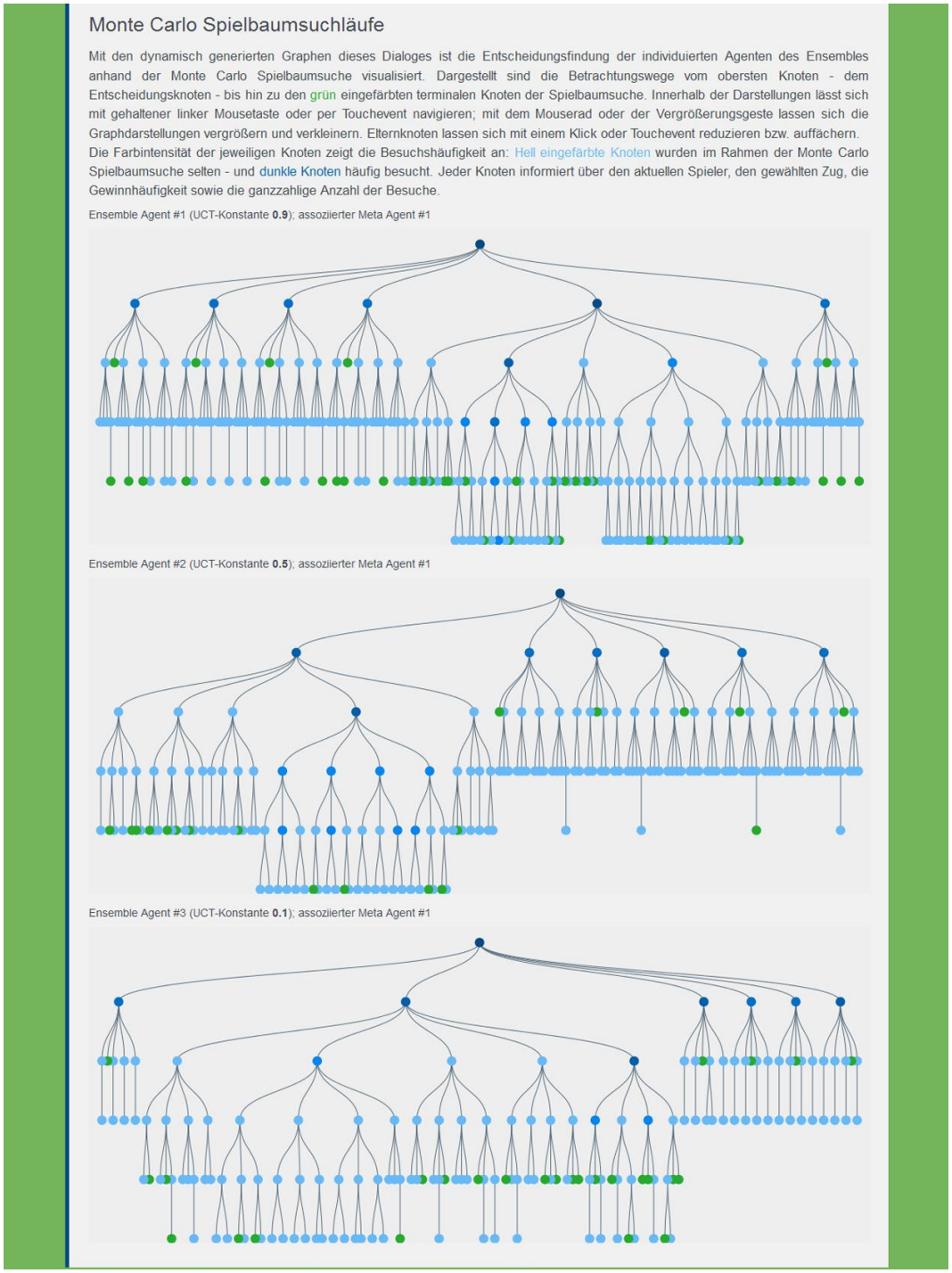


Abbildung 51: Screenshot der laufzeitgenerierten Monte Carlo Spielbaumsuchläufe der einzelnen Agenten des Agentenensembles.

In dem zuvor wiedergegebenen Bildschirmausschnitt sind die Spielbäume dargestellt, die von den drei Agenten des zum Meta Agenten mit der ID 1 zugehörigen Ensembles zur Entscheidungsfindung erstellt und für die Entscheidungsfindung verwendet wurden. Ausgehend vom Wurzelknoten sind die unterschiedlichen Zugmöglichkeiten als Knoten dargestellt, die wiederum in Folgezugmöglichkeiten verzweigen. Verweisen hellblau eingefärbte Knoten auf wenig besuchte und folglich weniger interessante Zugmöglichkeiten, sind eingehend betrachtete Knoten in der Spielbaumdarstellung dunkelblau hervorgehoben. Grün eingefärbte Knoten des Grpahen stellen Terminalknoten dar, an denen ein Spieler oder seine Gegenspielerin die Spielpartie gewonnen hat oder die Partie unentschieden ausgegangen ist. Weil jeder Agent des Agententeams mit einem anderen Wert der UCT-Konstante arbeitet (von oben nach unten: 0.9, 0.5 und 0.1), unterscheiden sich die laufzeitgenerierten Spielbäume – teils marginal, wie in den dargestellten Spielbäumen in der Tic Tac Toe Partie, teils stark differierend in Spielen mit hoher Entscheidungskomplexität und hohem Verzweigungsfaktor.

Informationen wie die Besuchshäufigkeit und die Gewinnrate jeder im Rahmen der Monte Carlo Spielbaumsuche analysierten Zugmöglichkeit lassen sich in der Graphenvisualisierung per Touch-Event oder Mouseover abrufen. Mit dem Mouserad oder der Vergrößerungsgeste lässt sich die Darstellung vergrößern und verkleinern – und lässt sich mit gehaltener Mousetaste oder unter Verwendung des Touch-Events innerhalb der Spielbäume navigieren.

6.2 Systemarchitektur – Kernkomponenten von SpoookyJS

Das Framework *SpoookyJS* besteht aus verschiedenen miteinander verknüpften Modulen, die die Funktionalität des Systems implementieren. Anhand der folgenden informellen Systemskizze sei einleitend eine Übersicht über zentrale Teile der Systemarchitektur von *SpoookyJS* gegeben:[467]

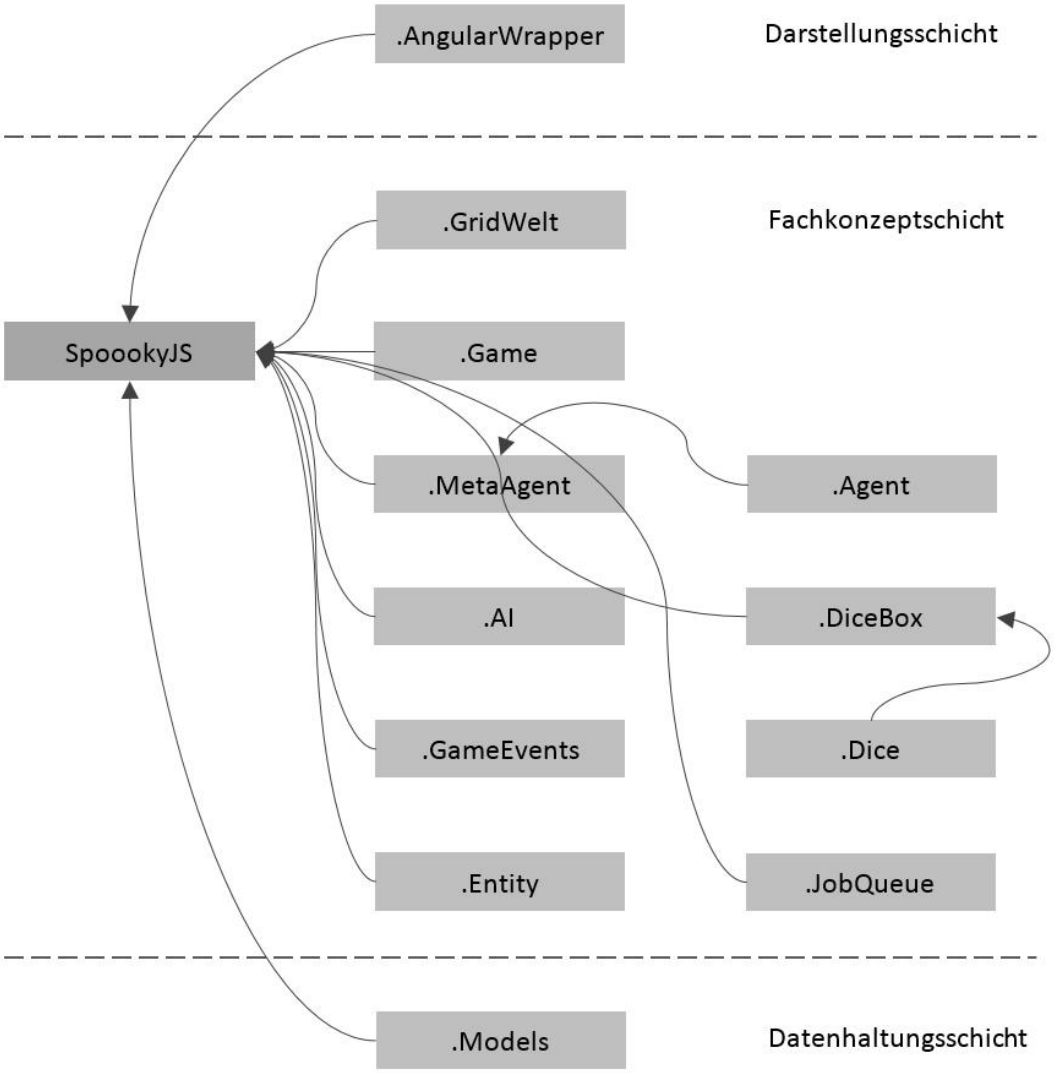


Abbildung 52: Systemskizze des JavaScript-Frameworks *SpoookyJS*.

[467] In der skizzenhaften Darstellung der Systemarchitektur wurde eine Auswahl relevanter Submodule von *SpoookyJS* getroffen. So wurde davon abgesehen, Module wiederzugeben, die einzig der Darstellung von Spieleigenschaften dienen (u.a.: *Spoooky.GameProcess* zur Ausgabe des Spielverlaufes und *Spoooky.AgentLog* zur Ausgabe der Agentenentscheidungen) und wurden der Übersichtlichkeit halber die Modulbezeichnungen von *Spoooky.Game* verkürzt auf .Game.

Wie in obiger Systemskizze veranschaulicht, ist das JavaScript-Framework *SpoookyJS* in einer flexiblen Drei-Schichten-Architektur[468] konzipiert: Auf der untersten Ebene, der Datenhaltungsschicht, erfasst die Klasse *Spoooky.Models* vollständig alle wesentlichen Spielcharakteristika: Die Spielregeln, die sich aus Spielregelatomen konstituieren, die Spielfiguren – im Folgenden auch als *Spielentitäten* oder *entities* bezeichnet –, die sich auf dem Spielbrett befinden, die Spielerobjekte sowie die Gesamtheit der zu einem bestimmten Spielzeitpunkt ausführbaren Zugmöglichkeiten. Um die vollständige Speicherung und den erschöpfenden Export aller Spielcharakteristika unter Verwendung der JavaScript Object Notation (JSON) zu realisieren, wie es das Verfahren der Monte Carlo Spielbaumsuche erfordert, verfügt die Klasse *Spoooky.Models* einzig über Attribute und keine Operationen, wie in dem folgenden Klassenausschnitt veranschaulicht; Klassen der Fachkonzeptschicht wie *Spoooky.JobQueue* oder *Spoooky.Entity* fungieren als Controller[469] und realisieren den Zugriff auf die in der Datenhaltungsschicht gespeicherten Spieleigenschaften.

Spoooky.Models
MetaAgents MovableEntities GameRules GameRuleAtoms GameRuleConsequences GameGrid DiceBox gameName gameState gameMode winnerID …

Abbildung 53: Klassenausschnitt *Spoooky.Models*.

[468] Vgl. Balzert 2011, S. 407.
[469] Vgl. auch das Model View Controller (MVC) Gestaltungsmuster bei Gamma 1995, S. 4–6.

In der Darstellungsschicht vermittelt die Klasse *Spoooky.AngularWrapper* zwischen Interna des Rahmenwerkes und *AngularJS*[470]. Das externe JavaScript-Framework *AngularJS* leistet u.a. die Darstellung des Spielfeldes sowie der Spielentitäten (d.h. der Spielsteine und Spielfiguren), realisiert unterschiedliche Spielfeldbereiche wie den überblickenden Spielverlauf oder spieldedizierte Bereiche und sieht Interaktionsmöglichkeiten mit den Spielelementen vor. *SpoookyJS* realisiert alle Spielbestandteile als Elemente des *Document Object Models*[471] (*DOM*) und sieht aufgrund der Schlichtheit und der Mächtigkeit des *DOM*-Ansatzes von der Verwendung von *WebGL* oder *HTML5 Canvas*[472] ab. Die Schnittstellen zur Darstellungsschicht sind klar definiert und adaptiv umgesetzt, so dass sich an Stelle von *AngularJS* auch eine alternative JavaScript-Bibliothek wie *KnockoutJS*[473] verwenden ließe, die das Entwurfsmuster *Model View ViewModel* implementiert.

Die Kernbestandteile von *SpoookyJS* konstituieren sich aus den verschiedenen Modulen der Fachkonzeptschicht, von denen in den nachfolgenden Subkapiteln ausgewählte zentrale Komponenten erläutert werden.

6.2.1 Spoooky.Game

Als Kernmodul fungiert die Klasse *Spoooky.Game*, die die Erstellung von Spielen mit basalen Kenntnissen der Programmiersprache JavaScript ermöglicht und Zugriffsmethoden auf die unterschiedlichen Spielcharakteristika bietet. *Spoooky.Game* fungiert als Mittler zwischen den einzelnen Modulen der Fachkonzept- und der Datenhaltungsschicht und bietet mit seiner

[470] Das JavaScript-Framework *AngularJS* ist frei verfügbar unter dem URL http://www.angularjs.org (zuletzt erfolgreich aufgerufen am 15.08.2014).
[471] Das *Document Object Model* (*DOM*) repräsentiert alle Objekte einer HTML-Seite intern in einer Baumstruktur, ist sprach- und plattformneutral und lässt sich durch die Programmiersprache JavaScript manipulieren (vgl. Fulton und Fulton 2011, S. 5).
[472] Auf Grundlage der *Open Graphics Library for Embedded Systems* (*OpenGL ES*) bietet sich mit *WebGL* die Möglichkeit, ohne zusätzliche Browsererweiterungen hardwarebeschleunigte 3D-Grafik zu nutzen. Das im *HTML 5* Standard spezifizierte *Canvas* Element ermöglicht das JavaScript-basierte Zeichnen graphischer Primitiven wie Linien, Kreise oder Rechtecke auf einer virtuellen Leinwand (vgl. Shankar 2012, S. 2).
[473] http://knockoutjs.com (zuletzt erfolgreich aufgerufen am 15.08.2014).

Memberfunktion[474] `clone` die Möglichkeit, das aktuelle Spiel zu replizieren – Funktionalität, die besonders bei der Monte Carlo Spielbaumsuche unabkömmlich ist.

In den individuellen und spieldedizierten Skriptdateien *game.js*, auf die in Kapitel 6.2.4 noch ausführlich einzugehen sein wird, nimmt jede Spielimplementation zuallererst ihren Lauf, indem eine Instanz[475] der Klasse *Spoooky.Game* erzeugt wird. Anschließend wird die gewünschte Spielfunktionalität unter Verwendung entsprechender Memberfunktionen realisiert. Der im Folgenden wiedergegebene Klassenausschnitt veranschaulicht die zentrale Funktionalität zur Erstellung von Spielen mit *Spoooky.Game*:

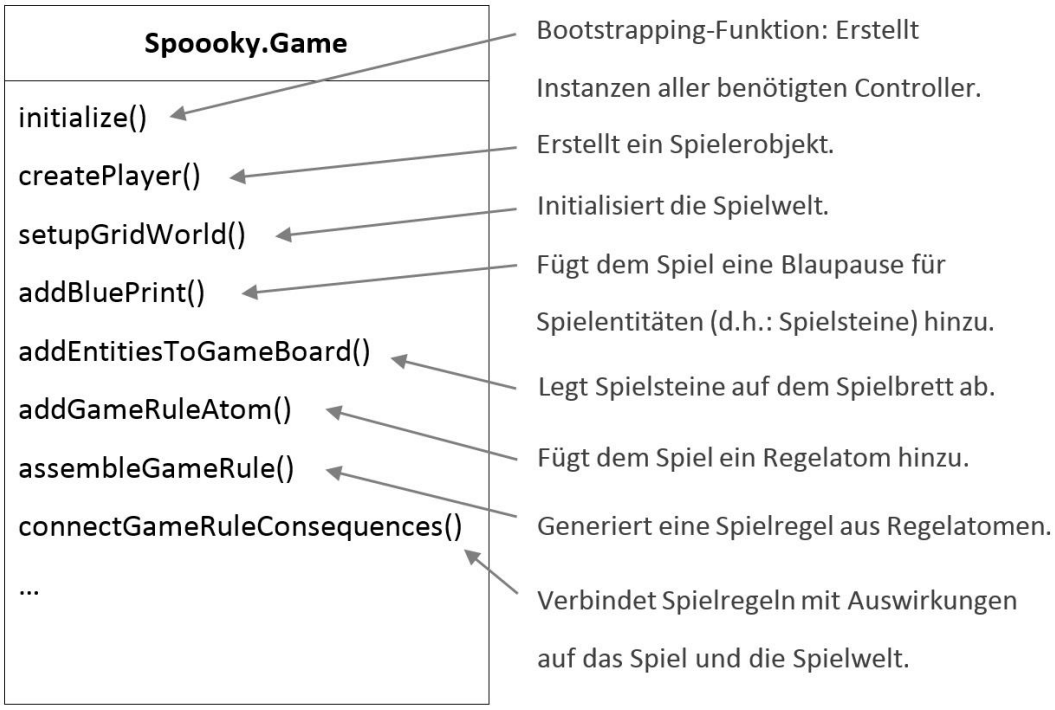


Abbildung 54: Klassenausschnitt *Spoooky.Game*.

Ein Spiel wird gespielt von mindestens einem Spieler – in *SpookyJS* als *Meta Agent* bezeichnet –, spielt sich ab in genau einer Spielwelt und verfügt über eine Datenstruktur, die Zugmöglichkeiten mit kausalen Folgen für das Spiel und die

[474] Als Memberfunktionen werden Funktionen bezeichnet, die sich innerhalb einer bestimmten Klasse deklariert finden. Der Zugriff auf die Memberfunktion einer Klasse gestaltet sich in JavaScript durch den Punktoperator „.“.

[475] In der objektorientierten Entwicklung wird zwischen der Klasse und der Instanz einer Klasse, dem Objekt unterschieden. Definiert eine Klasse für „eine Kollektion von Objekten deren Struktur (Attribute), Verhalten (Operationen) und Beziehungen“ (Balzert 2011, S. 23), ist das Objekt ein konkretes Exemplar einer solchen Blaupause, der Klasse.

Spielwelt verknüpft sowie weiteren Datenstrukturen, die den Spielzustand repräsentieren. Wie zuvor besprochen, finden sich alle zentralen Spielcharakteristika in der Klasse *Spoooky.Models* vereinigt, um das vollständige Kopieren von Spielen zu realisieren. Der Zugriff auf Bestandteile der Datenhaltungsschicht gestaltet sich durch entsprechende Controllerklassen, die in *Spoooky.Game* anhand der Member- und Bootstrappingfunktion `initialize` instanziiert werden und dem Spielobjekt anschließend zur Verfügung stehen. Mit dem folgenden Klassendiagrammauszug veranschaulicht sich ausschnitthaft die Beziehung zwischen *Spoooky.Game* und den angesprochenen weiteren Grundbausteinen des Frameworks.

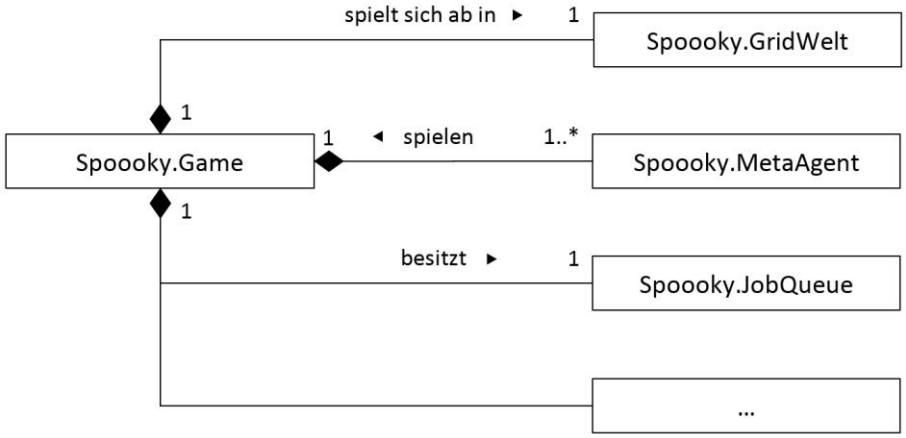


Abbildung 55: Klassendiagrammausschnitt *Spoooky.Game* und assoziierte Klassen.

Die Art der Assoziation zwischen den dargestellten Klassen ist die Komposition, visualisiert durch die ausgefüllte schwarze Raute an der Klasse *Spoooky.Game*: „Teile und Ganzes bilden eine Einheit, deren Auflösung […] die Zerstörung des Ganzen zur Folge haben kann“[476] – wird das Spiel beendet, finden auch Spielwelt, Spielsteine und weitere Spielbestandteile ihr Ende.

6.2.2 Spoooky.GridWelt

Unterschiedliche Spiele erfordern verschiedenartige Spielumgebungen. Der physische Aufbau der Spielwelten – die Umgebung, in der die Spielpartie stattfindet – lässt sich mit Björk und Holopainen sinnvoll differenzieren und aufschlüsseln. So unterscheiden die Autoren in ihrer Arbeit *Patterns in Game*

[476] Rupp et al. 2007, S. 147.

Design zwischen linearen, netzartigen und zwei- oder dreidimensionalen Spielwelten.[477] Jene Charakterisierung verschiedenartiger Spielwelten basiert auf den Zugmöglichkeiten, über welche die Spielsteine in den jeweiligen Spielumgebungen verfügen: Ist es Spielsteinen in linearen Spielwelten wie der des Backgammon-Spieles einzig möglich, sich in die eine oder in die andere Richtung zu bewegen, so können sich Agenten in netzartigen Umgebungen (*reticular Game Worlds*) zwischen miteinander verknüpften Spielfeldern bewegen, wie in der Mitte der im Folgenden wiedergegebenen Darstellung veranschaulicht. Die Spielwelt des Dame- und des Schachspieles jedoch ist eine zweidimensionale: Jede Spielfigur bewegt sich mit den ihr eigenen zur Verfügung stehenden Zugmöglichkeiten durch den zweidimensionalen Raum des Spieles.

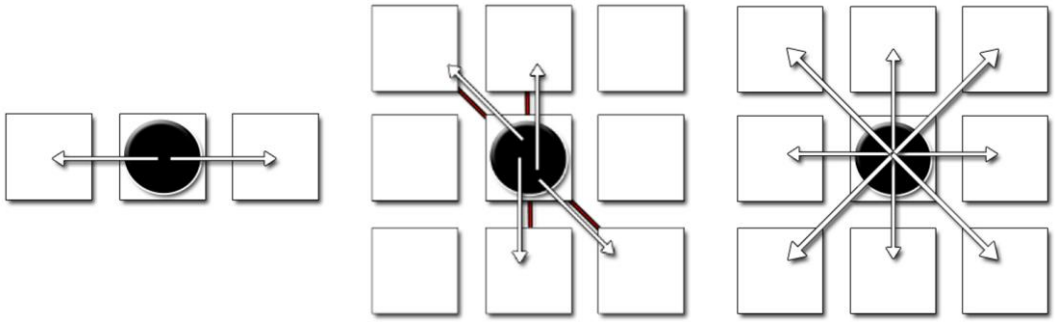


Abbildung 56: Zugmöglichkeiten in linearen (links), netzartigen (mitte) und zweidimensionalen (rechts) Spielwelten.

Um die Implementation mannigfaltigster Spielwelten mit *SpoookyJS* zu ermöglichen, wurde von der Verwendung und Implementation spielspezifischer Datenstrukturen wie *Bitboards* [478] abgesehen. Intern ist die Spielwelt als zweidimensionales Array[479] umgesetzt, in dem jede Speicherzelle des Arrays als

[477] Vgl. das Gestaltungsmuster *Game World* in Björk und Holopainen 2006, S. 55–58.

[478] Im Rahmen der Spielweltimplementation als *Bitboard* wird ein Spielbrett „durch mehrere 64-Bitzahlen im Speicher dargestellt.“ (Kantschick 2006, S. 112), deren Vorteile sich Kantschick zufolge wie nachfolgend zitiert gestalten: „[Die Darstellung eines Spielbrettes durch mehrere 64-Bitzahlen im Speicher] ermöglicht es sehr schnell, Züge durchzuführen und wieder zurückzunehmen, da diese durch xor-Verknüpfungen erfolgen. Da man durch eine 64-Bitzahl nur die Positionen feststellen kann, an denen sich die Figuren auf dem Schachbrett befinden, benötigt man mehrere 64-Bitzahlen um ein ganzes Schachbrett abzubilden […]. Insgeamt [sic] benötigt man neun 64-Bitzahlen, eine mit allen Figuren, eine nur mit den schwarzen Figuren, eine nur mit den weißen Figuren und eine für jeden Figurtyp. Durch und-Verknüpfungen der verschiedenen Zahlen kann man die Position einer Figur ermitteln. Diese Implementierung ist sehr schnell und ermöglicht es viele Stellungen pro Sekunde zu betrachten, besonders vorteilhaft ist das Verfahren bei der Verwendung von 64-Bit Prozessoren.“ (Kantschick 2006, S. 112).

[479] Ist eine Variable eine benannte Speicherstelle, in der sich Werte und Objekte ablegen lassen, so lässt sich ein Array vorstellen wie ein Schubladenelement, das über einen eindeutigen Namen

Kellerspeicher[480] realisiert ist. Über die Zugriffsmethode `pushToCell` ist es möglich, Spielsteine auf Spielfeldern abzulegen – und mit den Methoden `popFromCell` sowie `peekCell` lässt sich der Speicherzelleninhalt betrachten und zurückliefern. Geriert sich die Spielwelt standardmäßig zweidimensional, so ist es mit der Memberfunktion `setFieldIDs` möglich, jeder Zelle des Spielfeldarrays einen eindeutigen Identifikator zuzuweisen, so dass sich mit dem Framework über zweidimensionale Spielwelten hinaus auch lineare Umgebungen wie die des Backgammonspieles oder netzartige Spielwelten generieren lassen.

Weitere Funktionalität der Controllerklasse *Spoooky.GridWelt*, die neben der referierten Zugriffsfunktionalität auch die Erstellung von Spielbrettsignaturen anbietet, sei mit dem folgenden Klassendiagrammausschnitt angemerkt und veranschaulicht:

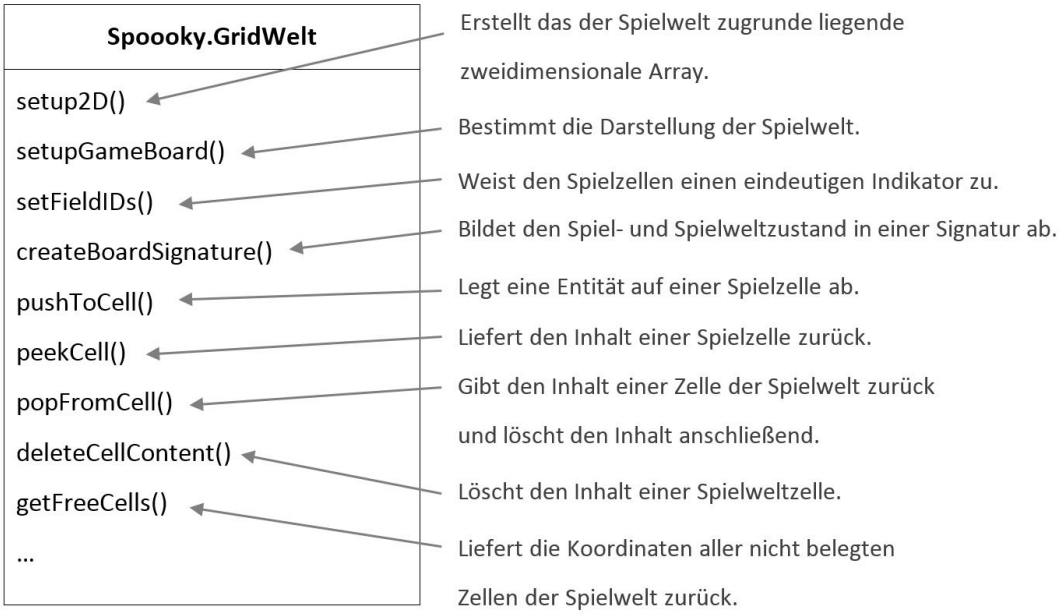


Abbildung 57: Klassenausschnitt *Spoooky.GridWelt*.

Ihren Lauf nimmt die Spielpartie anhand der Spielschleife, die im nachfolgenden Kapitel erläutert wird.

---

verfügt und mehrere Schubladen (Speicherstellen / -zellen) beinhaltet, auf die sich über Indices zugreifen lässt.

[480] Die Datenstruktur des Kellerspeichers folgt dem *Last-In-First-Out* (*LIFO*) Prinzip, demzufolge Elemente auf dem Kellerspeicher obenauf abgelegt, d.h. gestapelt werden. Zugreifbar ist das oberste Element, d.h. das Element, welches zuletzt in den Kellerspeicher abgelegt (d.h. gestapelt) wurde.

6.2.3 Spoooky.Game.loop

Ob Schach, Dame, Backgammon oder Tic Tac Toe – die Spielschleife ist für alle Spiele dieselbe und manifestiert sich in der Memberfunktion `loop` der Klasse *Spoooky.Game*. In jeder Iteration der Spielschleife wird zuallererst der Spielzustand geprüft, der als konstante Zeichenkette formuliert ist: Befindet sich das Spiel im Zustand `"END"`, so sind die Interaktionsmöglichkeiten mit der Spielwelt deaktiviert und keine weitere Iteration der Spielschleife ist möglich. Weilt das Spiel in einem anderen – auch eigens definierbaren – Zustand (z.B. `"INGAME"` oder `"WAITINGFORDICEROLL"`), ist die Interaktion mit der Spielwelt freigeschaltet. Klickt eine menschliche Spielerin mit der linken Mousetaste auf ein Element der Spielwelt – oder löst ein artifizieller Spieler ein virtuelles Klickereignis aus –, unterscheidet die Spielschleife zunächst den Spielmodus, der die Art und Weise bestimmt, wie mit den Spielentitäten interagiert werden kann: Werden Spielsteine auf dem Spielbrett bewegt (Spielmodus `"MOVING"`), wie im Schach- oder Damespiel, verzweigt die Spielschleife in den entsprechenden Verarbeitungszweig. Handelt es sich bei dem Spiel in seinem aktuellen Zustand um ein Legespiel (Spielmodus `"PLACING"`, greift die zugehörige Verarbeitungskette. Wie mit dem implementierten *Spiel der Amazonen* veranschaulicht, ist es möglich, den Spielmodus während der Spielpartie zu wechseln: Wurde eine Amazone auf ein leeres Feld bewegt (Spielmodus `"MOVING"`), wechselt der Spielmodus, so dass anschließend ein Pfeil auf ein leeres Feld abgelegt werden kann (Spielmodus `"PLACING"`).

In jeder Verzweigung prüft die Spielschleife, ob mit einem Spielweltbereich interagiert, ob eine Spielfigur ausgewählt oder ein Zielfeld selektiert wurde und reagiert mit einem definierten Verhalten auf das Interaktions- und Spielereignis. Auswirkungen auf das Spiel und die Spielwelt sind hierbei durch Aufgaben (*Jobs*) charakterisiert, die jeder zu einem Spielzustand ausführbaren Zugmöglichkeit zugeordnet und mit jeder aktivierten Zugmöglichkeit ausgeführt werden. Welche Spielereignisse zuallererst möglich sind – z.B. das Rollen von Würfeln, das Platzieren von Spielsteinen oder der Wechsel des Spielmodus‘ während der Laufzeit –, ist bestimmt durch die Klasse *Spoooky.GameEvents*; die für einen Zug

auszuführenden Spielereignisse werden durch die Klasse *Spoooky.JobQueue* koordiniert.

Der im Folgenden wiedergegebene kommentierte JavaScript-Quelltext bietet einen Auszug aus der Spielschleife und veranschaulicht die Erläuterungen:

```
1    self_Game.loop = function(areaName, xPos, yPos){
2
3        var isMove = false;
4
5        // Ermögliche die Interaktion mit dem Spiel,
            so lange der Endzustand nicht erreicht ist
6        if (self_Game.getGameState() !== "END") {
7
8            // Unterscheide unterschiedliche Spiel-
                 und Interaktionsmodi
9            switch (self_Game.getGameMode()) {
10
11               // Spiele wie Gomoko, Tic Tac Toe oder Go,
                       in denen Spielentitäten auf dem
                       Spielbrett abgelegt werden.
12               case "PLACING":
13
14                   // Zeige Interaktionsmöglichkeiten.
15                   self_Game.showFieldsToPlaceEntity();
16
17                   // Verarbeite das Klickevent.
18                   isMove = self_Game.isMove(xPos, yPos);
19
20                   // Wurde eine legitime
                        Zugmöglichkeit ausgewählt?
21                   if (isMove) {
22
23                       // Führe Jobs aus, die mit der
                             aktivierten Zugmöglichkeit
                             verbunden sind.
24                       self_Game.
                             executeJobsForThisMove(xPos, yPos);
25
26                       // Prüfe globale Spielziele und
                             führe verknüpfte Konsequenzen aus.
27                       if (self_Game.executeGameGoals()) {
28                           // Wenn globale Spielziele
                                 erfüllt sind, dann führe
                                 assoziierte Jobs aus.
29                           self_Game.executeJobsForRecentMove();
30                       } else {
31                           // Stoße eine neue Spielrunde an
32                           self_Game.proceed();
33                           // Zeige freie Spielfelder
                                 nach Spielerwechsel.
34                           self_Game.showFieldsToPlaceEntity();
35                       }
36                   }
37                   break;
38
39               // Spiele wie Schach oder Dame,
                       in denen Spielsteine bewegt werden
```

```
40                  case "MOVING":
41                      // [...]
42              }
43          }
44
45          // Steuerimpuls für den artifiziellen Gegner
46          self_Game.playArtificial();
47      };
```

6.2.4 Spoooky.MetaAgent

Realisiert das zuvor betrachtete Modul *Spoooky.Game* die Kernfunktionalität zur Implementation von Spielen und deren Spielverlauf, finden sich die Spieler des Spiels repräsentiert durch das Modul *Spoooky.MetaAgent*, das zum einen die Funktionalität zur menschlichen oder artifiziell gestalteten Interaktion mit dem Spiel sowie der Spielwelt vorsieht und zum anderen künstliche Opponenten kreiert. Um dem Spiel einen Spieler hinzuzufügen, wird die Memberfunktion `createPlayer` von *Spoooky.Game* aufgerufen, die eine neue Instanz von *Spoooky.MetaAgent* generiert und zurückliefert. Als Übergabewert wird der Funktion `createPlayer` ein Literal übergeben, das über den Namen sowie den Typ des Spielers informiert. Die Typangabe determiniert hierbei das Spielerobjekt: Wird der Typ auf `"HUMAN"` gesetzt, so steuert ein menschlicher Spieler bzw. eine menschliche Spielerin den entsprechenden Meta Agenten – die Darstellungsschicht ermöglicht nun die Interaktion mit der Spielwelt per Mouseklick. Wird der Typ mit der Zeichenkette `"ARTIFICIAL"` belegt, obliegt die Entscheidungsfindung einem artifiziellen Spieler und die Darstellungsschicht sperrt den Zugriff auf Spielsteine des künstlichen Kontrahenten. Veranschaulicht sei die Erstellung eines artifiziellen Spielers mit dem folgenden Quelltextausschnitt aus der spielindividuellen Skriptdatei *game.js*:

```
1    var player1 = game.createPlayer({
2        name: "Scully", type: "ARTIFICIAL"
3    });
```

Anders als bei der Erstellung eines menschlich gesteuerten Spielerobjektes, bei dem die Interaktion mit der Spielwelt durch Klickereignisse bestimmt ist, werden bei einem Spielerobjekt, das mit dem Typ `"ARTIFICIAL"` generiert wird, weitere Prozesse angestoßen, die artifizielle Entscheidungsfindung in unterschiedlichen Spielen ermöglicht: Jedes artifizielle Spielerobjekt – d.h. jeder Meta Agent – verfügt über ein Team dedizierter Agenten, die unterschiedliche Aspekte des aktuellen Spielzustandes untersuchen und dem Meta Agenten Handlungsvorschläge unterbreiten. Wie sich die Entscheidungsfindung im Agentenkollektiv gestaltet, darauf sei an in Kapitel 6.4 ausführlich eingegangen.

6.3 Spielerstellung mit SpoookyJS

Die nachfolgenden Kapitel leiten ein in die Arbeit mit dem JavaScript-Framework *SpoookyJS*. Werden zunächst Projekthomepage, Bezugsquellen und die Ordnerstruktur des Frameworks vorgestellt, wird anschließend die Implementation von digitalen Brettspielen mit der spieldedizierten Skriptdatei *game.js* anhand des Schachspiels besprochen.

6.3.1 Download und Ordnerstruktur

Um ein Spiel mit *SpoookyJS* zu erstellen, wird das Framework in seiner aktuellen Version von der Webadresse http://www.spoookyjs.de heruntergeladen oder eine lokale Kopie des Repositoriums mittels *Git*-Client erstellt.[481] Nach Extraktion des komprimierten Paketes präsentiert sich *SpoookyJS* in der folgenden Ordner- und Dateistruktur, deren Bestandteile sich in der nachfolgenden Übersicht erläutert finden:

[481] Für den URL des *SpoookyJS*-Repositoriums sei an dieser Stelle verwiesen auf die Projekthomepage.

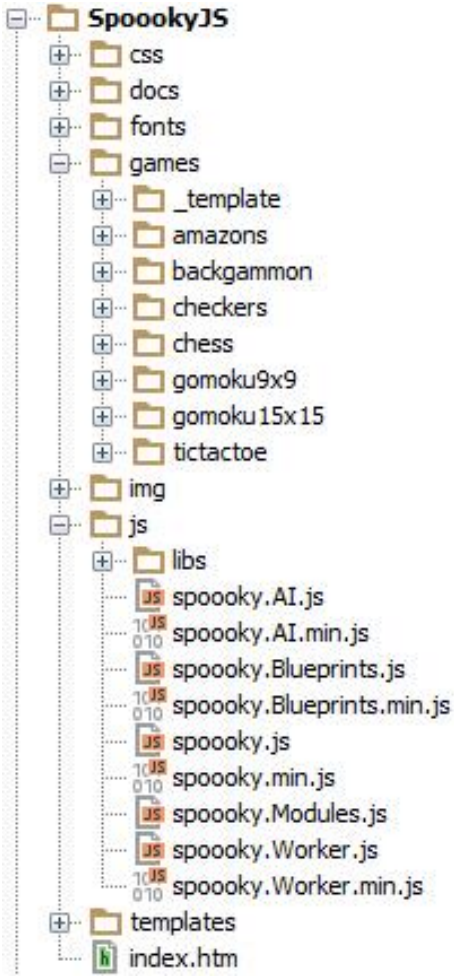


Abbildung 58: Ordnerstruktur von *SpoookyJS* (Screenshot aus der Entwicklungsumgebung *PHPStorm*).

Ordnername	Inhalt und Funktion
css	Cascading Style Sheets (CSS) zur Anpassung der Darstellung.
docs	Dokumentation der Programmierschnittstellen.
fonts	Von dem CSS-Framework *Bootstrap* verwendete Icons und Symbole.
games	Mit dem Framework erstellte und lauffähige Spiele wie Dame (*checkers*), Schach (*chess*) oder Backgammon. Im Unterordner *_template* finden sich Startvorlagen für die eigene Erstellung von Spielen.
img	Rastergrafiken für die Darstellung der Homepage.

js	Der Kern des Frameworks: JavaScript-Dateien, die die Funktionalität des Frameworks realisieren.
js/libs	JavaScript-Dateien der verwendeten Bibliotheken und Frameworks.
template	HTML-Vorlagen für häufig genutzte Bereiche wie die Spielkonfiguration, der Spielverlauf oder zweidimensionale Spielwelten.[482]

Zentral für die Erstellung von digitalen Brettspielen mit dem Framework ist der Ordner *games*, in dem sich bereits erstellte Spiele finden – und der mit der Startvorlage im Unterordner *_template* als Ausgangspunkt für die eigene Spielerstellung dienen mag. Besprochen und implementiert wird im Folgenden das Schachspiel, dessen Aufbau, Spielfiguren und Spielregeln sich mit dem anschließenden Kapitel zunächst erläutert und mit den nachfolgenden Kapiteln unter Verwendung des Frameworks realisiert findet.

[482] Die Vorlagen (*templates*) liegen in Form von HTML-Dateien vor und werden von *AngularJS* dynamisch geladen und in das Zieldokument eingefügt. Weil das Sicherheitskonzept der *Same-Origin-Policy* im lokalen Browser das Laden von Vorlagedateien aus überliegenden Ordnern verwehrt, sind im Downloadpaket des Frameworks die jeweiligen Indexdateien der Spiele (z.B. *games/chess/index.htm*) bereits vollständig ergänzt.

6.3.2 Das Regelwerk des Schachspieles

Eine Partie des Schachspiels wird gespielt von den zwei Spielern *Schwarz* und *Weiß*, die nacheinander ihre Spielfiguren Zug für Zug auf dem Schachbrett bewegen.[483] Jenes Schachbrett manifestiert sich als zweidimensionale Matrix von $8 \times 8 = 64$ quadratischen Spielfeldern, die abwechselnd hell und dunkel eingefärbt sind. Vor die Spielerinnen und Spieler wird das Schachbrett dergestalt abgelegt, dass das rechte untere Spielfeld ein weiß eingefärbtes Feld ist. Zu Beginn der Spielpartie verfügt jede Spielerin und jeder Spieler über 16 Spielsteine unterschiedlicher Charakteristika, über deren Symbole sowie Quantität zu Beginn des Spieles die folgende Übersicht informiert:

Spielfiguren des Spielers *Schwarz*			Spielfiguren des Spielers *Weiß*		
Name	Symbol	Initiale Anzahl	Name	Symbol	Initiale Anzahl
Bauer	♟	8	Bauer	♙	8
Turm	♜	2	Turm	♖	2
Springer	♞	2	Springer	♘	2
Läufer	♝	2	Läufer	♗	2
Dame	♛	1	Dame	♕	1
König	♚	1	König	♔	1

Tabelle 6: Die Spielfiguren des Schachspieles

[483] Die Regeln des Schachspieles werden im Folgenden auszugsweise wiedergegeben nach dem offiziellen Regelwerk des Weltschachbundes *FIDE* (Féderation Internationale des Échecs). Den Erläuterungen grundlegend sind hierbei die Artikel eins bis fünf des Kapitels *Grundspielregeln* der offiziellen deutschen Übersetzung der FIDE-Schachregeln, wie sie sich finden unter http://www.schachbund-bayern.de/uploads/media/FIDE_Regeln-09.pdf (zuletzt erfolgreich aufgerufen am 27.06.2014).

Zu Beginn der Schachpartie werden die Spielfiguren der beiden Spieler nach der im Folgenden wiedergegebenen Anordnung auf das Spielbrett gesetzt.

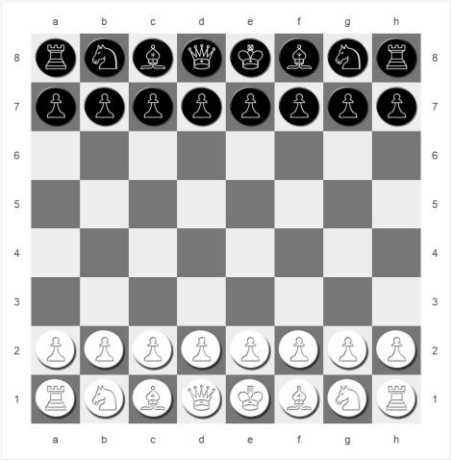


Abbildung 59: Initialkonfiguration des Schachbrettes

Ausgehend von der dargestellten Initialkonfiguration beginnt die Spielerin *Weiß* das Spiel und nimmt das Schachspiel seinen Lauf.

6.3.2.1 Spielfiguren und ihre Bewegungsmöglichkeiten

Jede der sechs unterschiedlichen Arten von Spielfiguren verfügt über individuelle Möglichkeiten, sich auf dem Spielbrett zu bewegen. So vermag es der Springer, Spielfelder zu überspringen und so ist es dem Turm möglich, sich horizontal und vertikal um eine beliebige Anzahl von freien Feldern fortzubewegen. Einer Spielfigur ist es nicht möglich, auf ein Feld des Spielbrettes zu ziehen, das bereits von einer eigenen Spielfigur besetzt ist; zieht eine Spielfigur auf ein Feld, auf dem sich eine gegnerische Figur findet, so wird die gegnerische Spielfigur geschlagen und vom Spielbrett entfernt. Der folgende Überblick gibt die Zugmöglichkeiten der Spielfiguren wieder;[484] gestalten sich die Zugregeln zumeist trivial – ein Zug ist gültig und ausführbar, WENN sich auf dem Zielfeld kein eigener Spielstein befindet –, so bieten die Zugregeln der kurzen und langen Rochade sowie Möglichkeit des Bauern, *en passant* zu schlagen, dienliche Erläuterungsfälle für die Aufgliederung der Zugregeln in Zugregelatome.

Zugmöglichkeiten und Zugregeln des Königs

[484] Vgl. auch den entsprechenden Überblick über die Zugmöglichkeiten der einzelnen Spielfiguren des Schachspieles in Glonnegger 2009, S. 181 ff.

Als zentraler Figur des Schachspieles ist es dem König ausschließlich möglich, sich – wie in nachfolgender Darstellung visualisiert – auf unmittelbar an ihn angrenzende Felder des Spielbrettes zu bewegen, die nicht von einer gegnerischen Spielfigur zu erreichen, d.h. bedroht sind.

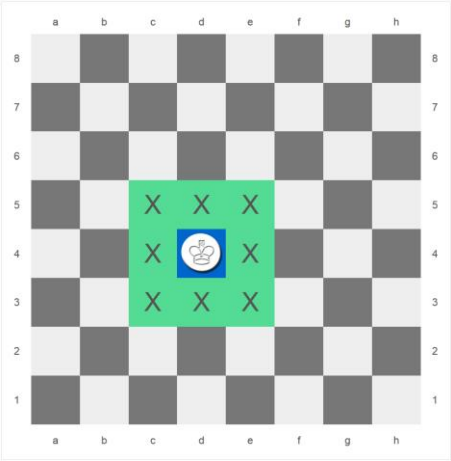


Abbildung 60: Zugmöglichkeiten des Königs im Schachspiel.

Eine Besonderheit in der Bewegungsmöglichkeit der Spielfiguren stellt die Fähigkeit des Königs dar, zu *rochieren*. Bei der Rochade bewegt sich der König um zwei Felder auf einen eigenen Turm hin; der Turm wird anschließend auf das vom König überquerte Feld gesetzt. Unterschieden wird zwischen der langen und der kurzen Rochade: Agiert der König bei der kurzen Rochade mit dem Turm zu seiner Rechten, so wechseln König und der linke Turm ihre Position bei der langen Rochade. Veranschaulicht seien die beiden Rochademöglichkeiten mit den folgenden Darstellungen:

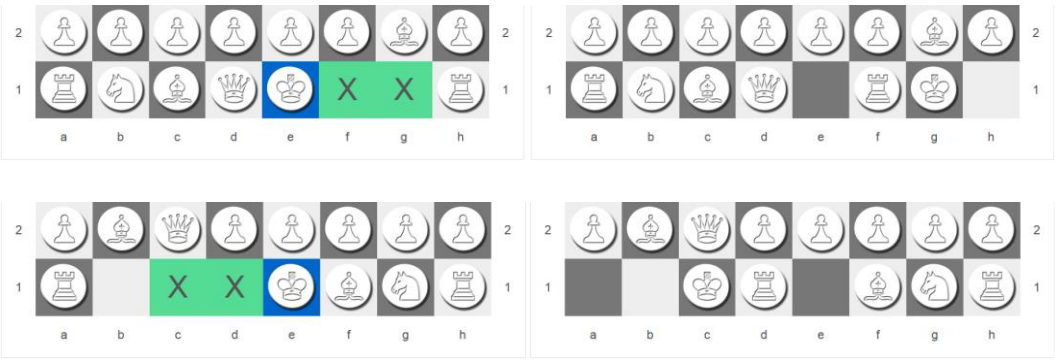


Abbildung 61: Zugmöglichkeiten der kurzen Rochade (oben) und langen Rochade (unten) im Schachspiel. Ausschnitthaft dargestellt sind die Spielkonfiguration vor (linke Seite) und nach Ausführung der Rochaden.

Die Zugregeln der kurzen und der langen Rochade konstituieren sich aus jeweils fünf Zugregelatomen ($Atom_1$ bis $Atom_5$), wie nachfolgend wiedergegeben.

Zugregel *Kurze Rochade (Spielerin weiß)*

Wenn der weiße König nicht von einem gegnerischen Spielstein bedroht wird ($Atom_1$) **und** der weiße König noch nicht bewegt wurde ($Atom_2$) **und** der weiße Turm auf Position h2 noch nicht bewegt wurde ($Atom_3$) **und** sich zwischen dem weißen König auf e1 und dem weißen Turm keine eigenen oder gegnerischen Spielsteine befinden ($Atom_4$) **und** sich zwischen dem weißen König und dem weißen Turm kein Feld findet, das von einem gegnerischen Spielstein bedroht wird ($Atom_5$), **dann** bewege den weißen König nach g1 **und** bewege den weißen Turm nach f1.

Zugregel *Lange Rochade (Spielerin weiß)*

Wenn der weiße König nicht von einem gegnerischen Spielstein bedroht wird ($Atom_1$) **und** der weiße König noch nicht bewegt wurde ($Atom_2$) **und** der weiße Turm auf Position a2 noch nicht bewegt wurde ($Atom_3$) **und** sich zwischen dem weißen König auf e1 und dem weißen Turm keine eigenen oder gegnerischen Spielsteine befinden ($Atom_4$) **und** sich zwischen dem weißen König und dem weißen Turm kein Feld findet, das von einem gegnerischen Spielstein bedroht wird ($Atom_5$), **dann** bewege den weißen König nach c1 **und** bewege den weißen Turm nach d1.

Zugmöglichkeiten und Zugregeln des Bauern

Der Bauer verfügt über unterschiedliche Möglichkeiten, sich vorwärts auf die gegnerischen Reihen zu bewegen. Erreicht der Bauer die letzte gegenüberliegende Reihe des Spielbrettes, so wird er eingetauscht entweder in einen Turm, einen Springer, einen Läufer oder eine Dame gleicher Farbe. Hat sich der Bauer noch nicht bewegt, so kann der Bauer in seinem ersten Zug horizontal um ein oder zwei Felder (*Doppelschritt*) bewegt werden:

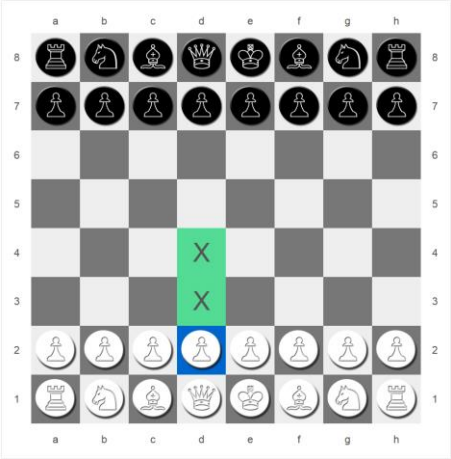


Abbildung 62: Einfach- und Doppelschritt des Bauern.

Schlagende Züge des Bauern finden ausschließlich diagonal statt: Befindet sich ein gegnerischer Spielstein auf einem vorwärts diagonal unmittelbar angrenzenden Feld (in der nachfolgenden Grafik zur Linken rot bzw. verdunkelt dargestellt), so kann der Bauer den gegnerischen Spielstein schlagen.

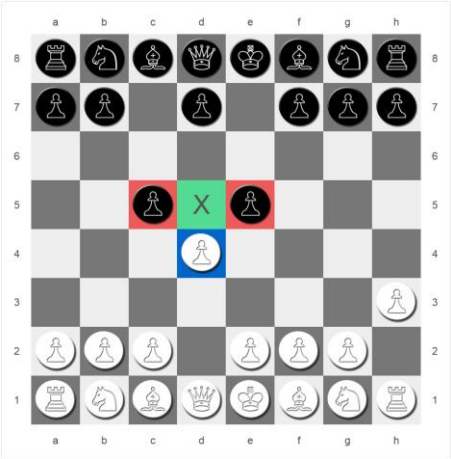


Abbildung 63: Schlagende Züge des Bauern.

Schlagen en passant: Überspringt ein gegnerischer Bauer mit einem Doppelschritt den Schlagbereich des eigenen Bauern, so ermöglicht es die Zugregel *en passant* (frz. *im Vorbeigehen*), den gegnerischen Zug als Einzelzug zu werten und den gegnerischen Bauern zu schlagen.

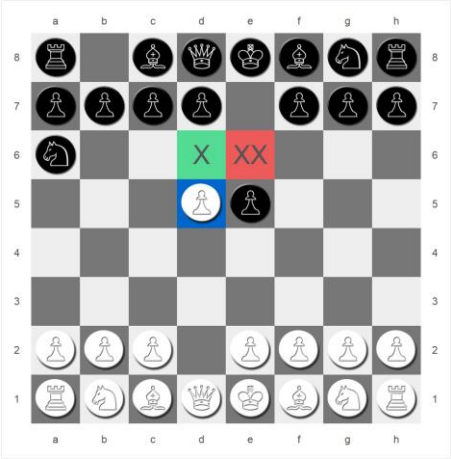


Abbildung 64: Bauernzug *Schlagen en passant*.

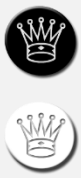
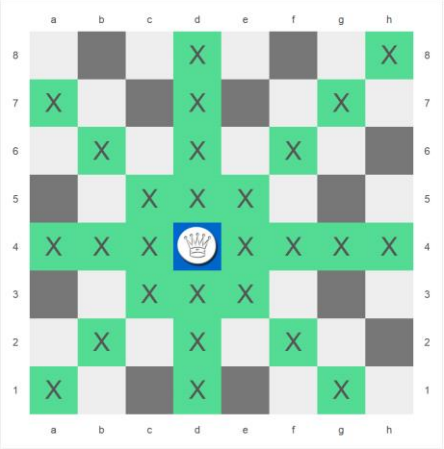
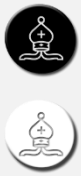
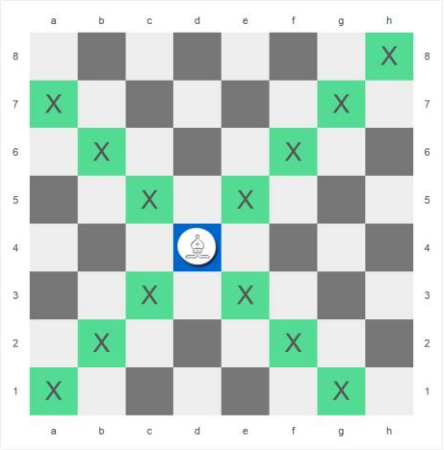
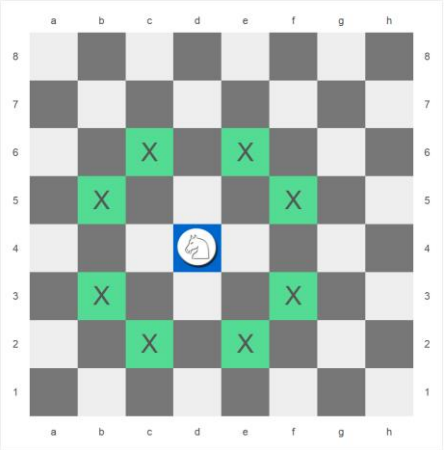
Die Zugregel des Schlagens im Vorbeigehen konstituiert sich aus vier Zugregelatomen, wie nachfolgend wiedergegeben.

Zugregel *Schlagen en passant (Spielerin weiß)*

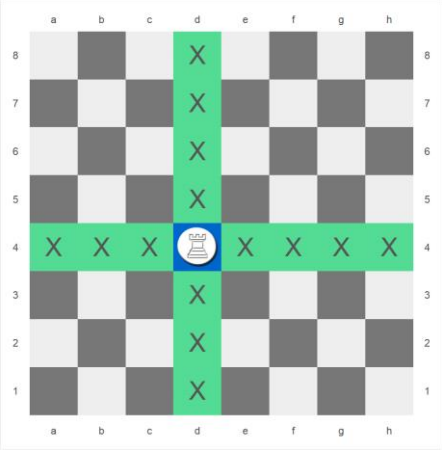
Wenn sich ein weißer Bauer auf einem Feld in Reihe fünf befindet ($Atom_1$) **und** sich ein gegnerischer Bauer B_G auf einem horizontal unmittelbar angrenzenden Feld befindet ($Atom_2$) **und** B_G nur ein einziges Mal bewegt wurde ($Atom_3$) **und** B_G der letzte vom Gegner bewegte Spielstein ist ($Atom_4$), **dann** setze den weißen Bauern auf das von B_G übersprungene Feld **und** nehme B_G vom Spielbrett.

Zugmöglichkeiten und Zugregeln der übrigen Spielfiguren

Die Bewegungsmöglichkeiten der Dame, des Läufers, des Springers sowie des Turms sind in der folgenden Übersicht zusammengefasst:

Name	Zugmöglichkeiten	Erläuterung
Dame 		Die Dame kann sich um eine beliebige Anzahl von Feldern sowohl horizontal als auch vertikal als auch diagonal bewegen. Bei ihren Zügen darf die Dame keine eigenen oder gegnerischen Spielsteine überspringen.
Läufer 		Der Läufer kann sich auf ein diagonal angrenzendes Feld in beliebiger Entfernung bewegen, ohne einen gegnerischen oder eigenen Spielstein zu überspringen.
Springer		Der Springer verfügt über die Möglichkeit, sich auf Spielfelder zu bewegen, die nicht unmittelbar an ihn angrenzen, wie in der Darstellung zur Linken visualisiert.

Turm



Dem Turm ist es möglich, sich horizontal und vertikal um eine beliebige Anzahl freier Felder zu bewegen.

6.3.2.2 Spielverlauf, Spielregeln und Spielregelatome

Der Verlauf der Schachpartie gestaltet sich sequenziell: Hat der aktuelle Spieler einen legitimen Zug ausgeführt, so ist die Runde beendet und der andere Spieler ist an der Reihe. Das individuelle Ziel der beiden Spieler besteht darin, den gegnerischen König zugunfähig zu setzen. Wird ein König von einem gegnerischen Spielstein bedroht, so steht er im *Schach*; verfügt der bedrohte König über keine legitimen Zugmöglichkeiten, so ist er *schachmatt*. Die Spielpartie gewinnt, wer den gegnerischen König schachmatt setzt; als unentschieden[485] wird die Spielpartie gewertet, wenn alle Spielfiguren beider Spieler geschlagen, die Spieler somit einzig im Besitz ihres Königs sind. Wenngleich eine Schachpartie durch hochkomplexe Entscheidungssituationen charakterisiert ist, so sind die dem Spiel zugrunde liegenden Regeln, die den Lauf der Partie bestimmen und determinieren, in ihrer Quantität ausgesprochen überschaubar, wie die folgende Aufgliederung in Regelpartikel veranschaulicht. So verursacht der Spielzug eines Spielers, der den gegnerischen König Schach setzt, einen Wechsel des Spielzustandes in den Zustand *SCHACH*, der vom Gegenspieler in der nächsten Spielrunde unbedingt aufgelöst werden muss:

Spielregel *Ein König steht im Schach*

Wenn der zuletzt ausgeführte Zug eine Bedrohung für den gegnerischen König verursacht ($Atom_1$), **dann** ändere den Zustand des Spieles von *IM SPIEL* auf *SCHACH*.

Der Spielzustand *SCHACH* schränkt die Zugmöglichkeiten des aktuellen Spielers in der folgenden Spielrunde ein auf Figuren und ihre Zugmöglichkeiten, die dazu dienen, die Bedrohung des Königs aufzuheben. Ist es dem bedrohten König selbst jedoch nicht möglich, sich aus seiner misslichen Lage zu befreien – und kann auch keine andere Spielfigur des aktuellen Spielers die Bedrohung auflösen, so ist der

[485] Vgl. weiterführend das Kapitel *Die Beendigung der Partie* in der offiziellen deutschen Übersetzung der FIDE-Schachregeln (http://www.schachbund-bayern.de/uploads/media/FIDE_Regeln-09.pdf, zuletzt aufgerufen am 28.04.2014).

bedrohte König matt gesetzt und der gegnerische Spieler gewinnt das Spiel, wie durch die folgende Spielregel und ihre –atome repräsentiert:

Spielregel *Schachmatt*

Wenn sich das Spiel im Zustand *SCHACH* befindet ($Atom_1$) **und** der bedrohte König über keine Zugmöglichkeit verfügt, sich aus seiner Bedrohung zu befreien ($Atom_2$) **und** kein Spielstein des aktuellen Spielers die Bedrohung des Königs aufheben kann ($Atom_3$), **dann** beende das Spiel, d.h. setze den Zustand des Spieles auf *ENDE* **und** erkläre den schachmatt gesetzten Spieler zum Verlierer und seinen Gegenspieler zum Gewinner.

6.3.3 Entitätenblaupausen – Implementation von Spielsteinen und Zugmöglichkeiten

Spielsteine wie der Turm, der Springer oder der Läufer des Schachspiels werden im Framework in Form von Entitätenblaupausen repräsentiert, aus denen konkrete Spielinstanzen generiert werden.[486] Eine Entitätenblaupause formuliert sich als JavaScript-Objektliteral[487] und besteht aus verschiedenen Komponenten wie dem Typnamen, dem Verweis auf eine Rastergrafik, die der visuellen Darstellung des Spielsteines dient, sowie der Definition basaler und schlagender Zugmöglichkeiten, die in *SpoookyJS* als individuelle Ziele des Spielsteines verstanden werden. Vordefinierte Entitätenblaupausen für die Spielfiguren der bereits implementierten Spiele finden sich in dem Vorlageskript *js/spoooky.Blueprints.js*, aus der im Folgenden die Umsetzung der Spielfigur des schwarzen Bauern besprochen wird. Im Schachspiel verfügt der schwarze Bauer über die Fähigkeit, gegnerische Spielsteine *en passant* zu schlagen und darf sich nur bewegen, wenn er den eigenen König mit seinem Zug nicht gefährdet.

Zu Beginn jeder Entitätenblaupause steht die Angabe des eindeutigen Bezeichners `typeID`, der zur Erstellung von Spielbrettsignaturen dient. Darauf folgt die Typangabe `entityType`, die es ermöglicht, mehrere Spielsteine gleichen Verhaltens aus einer Blaupause zu erstellen. Vervollständigt wird der Kopfbereich der Entitätenschablone von der Angabe des assoziierten Meta Agenten und Informationen zur Darstellung der Spielfigur, wie mit dem nachfolgenden Quelltext wiedergegeben, der die Blaupause des schwarzen Bauern einleitet:

```
1    black_pawn : {
2
3        typeID : "AA",
4        entityType : "Black Pawn",
5        associatedWithMetaAgent : null,
6        representation : { type : "image",
7            texture : "assets/black_pawn.png" },
```

[486] Die Erstellung der Spielsteine aus ihren Blaupausen leistet die Methode `entityFactory` des Meta Agenten.

[487] Ein JavaScript-Objektliteral verzeichnet null oder mehr Schlüssel / Wert-Paare nach der folgenden Syntax: `var spielFigur = {name: "Dame", ID : 0}` (vgl. Crockford 2008, S. 20).

Die individuellen Bewegungsmöglichkeiten der Spielfigur sind durch den Zugbereich *moves* des Objektliterals bestimmt. Jede Zugmöglichkeit innerhalb des Zugbereiches besteht aus den folgenden Komponenten:

- `name` vergibt einen Bezeichner für die Zugmöglichkeit der Entität.
- `type` gibt an, ob der Zug in einer zweidimensionalen (`type : "Default"`) oder in einer linearen oder netzartigen Spielwelt stattfindet (`type : "By Field ID"`). Bei linearen Spielwelten wie der des Backgammon-Spieles ist jeder Zelle der Spielwelt eine eindeutige ID zugewiesen, anhand derer sich die Spielfiguren in der Spielwelt bewegen.
- `direction` bezeichnet die Angabe, in welche Richtung der Zug erfolgt. Die Richtungsangabe lässt sich zum einen anhand der vordefinierten Zeichenkettenkonstanten `"north"`, `"northeast"`, `"east"`, `"southeast"`, `"south"`, `"southwest"`, `"west"`, `"northwest"` ausdrücken, um Bewegungen an unmittelbar angrenzende Felder der Spielwelt zu ermöglichen.[488] Zum anderen lässt sich die Richtungsangabe anführen als zweidimensionale Richtungsanweisung in der Form `[+1, +2]` für eine Positionsänderung um ein Feld nach rechts (x-Achse) und zwei Felder nach unten (y-Achse).
- Mit `frequency` lässt sich angeben, wie häufig der Zug ausgeführt wird, d.h. wie weit sich der Spielstein in die angegebene Richtung bewegen kann. Möglich sind ganzzahlige Werte oder die Zeichenkettenkonstante `"ANY"`, die den Spielstein auf jedes freie Feld in Zugrichtung ziehen lässt.

Jede Zugmöglichkeit lässt sich an verschiedene Bedingungen (*conditions*) binden, die allesamt erfüllt sein müssen, bevor die Zugmöglichkeit ausgeführt werden kann. Definiert sind die unterschiedlichen Zugauflagen in dem Objekt `Spoooky.Game.moveConditions`, anhand dessen sich Bedingungen wie *Ein bestimmtes Spielfeld ist unbesetzt* (Bedingung `"Is Empty"`) oder *Der Spielstein befindet sich nicht in der letzten Reihe des Spielbrettes* (Bedingung `"Is Not The`

[488] Jede Entität übersetzt die in der Entitätenblaupause angegebene Zeichenkettenkonstante anhand der Funktion `translateDirection` in zweidimensionale Richtungsanweisungen in der Form `[+1, 0]`. Für die vollständige Liste der Zeichenkettenkonstanten sei verwiesen auf die Funktion `translateDirection` in der Datei *spoooky.js*.

Last Row") formulieren lässt. Jeder Zugmöglichkeit lässt sich eine Prüfbedingung zuordnen, die nach Ausführung des Zuges erfüllt sein muss, damit die Zugmöglichkeit eine gültige ist. Der folgende Quelltext ergänzt den einleitend wiedergegebenen Quelltext, realisiert die Zugmöglichkeiten des schwarzen Bauern und knüpft die möglichen Züge an die Bedingung, dass der schwarze König nach Ausführung des Zuges nicht von einem gegnerischen Spielstein bedroht wird – und somit schachmatt gesetzt wäre:

```
8         // Zugbedingung: Prüfen, ob der eigene König nach
9         // ausgeführter Zugmöglichkeit des Bauern angreifbar ist
10        postMoveCheck : [{
11            condition : "Entity Is Attackable After Move",
12            state : false, entity : "Black King" }],
13        // Zugmöglichkeiten der Spielfigur
14        moves : [{
15            name : "Zug in Richtung des unteren Spielfeldrandes",
16            type : "Default",
17            direction : "south",
18            frequency : 1,
19            conditions : [
20                { condition : "Is Empty",
21                    state : true },
22                { condition : "Is Not The Last Row",
23                    state : true }]
24        }, {
25            name : "Startzug: Zwei Felder nach unten",
26            type : "Default",
27            direction : [ 0, +2 ],
28            frequency : 1,
29            conditions : [
30                { condition : "Is Empty",
31                    state : true },
32                // Andere Spielfiguren dürfen nicht
33                // übersprungen werden
34                { condition : "Is Empty At",
35                    relativeCoordinate : [ 0, +1 ],
36                    state : true },
37                { condition : "yPosition",
38                    value : 1, state : true }
39            ]
40        }],
```

Mit der Intention, den agentenbasierten Ansatz des Frameworks auch auf die unterste Ebene, die der einzelnen Spielsteine, zu übertragen, verfügt jede Spielfigur in *SpoookyJS* fakultativ über die Eigenschaft, eigene Ziele vorzusehen – Ziele wie das Schlagen gegnerischer Spielsteine oder im Schachspiel das Bestreben, die gegenüberliegende Seite des Spielbrettes zu erreichen. Ziele setzen

sich zusammen aus Subzielen, wie mit der folgenden Fortsetzung der Entitätenblaupause des schwarzen Bauern veranschaulicht:

```
41        // Definition von Unterzielen, die anschließend
42        // zu Zielen der Spielfigur zusammengesetzt werden
43        goalAtoms : [{
44            atomName : "Gegner auf suedoestlich
                              angrenzendem Feld",
45            atomFunction : "Is Opponent",
46            atomArguments : "southeast"
47        },{
48            atomName : "Gegner auf suedwestlich
                              angrenzendem Feld",
49            atomFunction : "Is Opponent",
50            atomArguments : "southwest"
51        }, {
52            atomName : "Leeres Feld Suedlich",
53            atomFunction : "Is Empty Cell",
54            atomArguments : "south"
55        }, {
56            atomName : "Figur steht auf einem Spielfeld
                              in Zeile vier",
57            atomFunction : "Current Y Position Is",
58            atomArguments : 4
59        },{
60            atomName : "Weißer Bauer westlich",
61            atomFunction : "Entity At Cell Is Of Type",
62            atomArguments : [ -1, 0, "White Pawn" ]
63        },{
64            atomName : "Weißer Bauer oestlich",
65            atomFunction : "Entity At Cell Is Of Type",
66            atomArguments : [ +1, 0, "White Pawn" ]
67        },{
68            atomName : "Weißer Bauer westlich
                              wurde nur einmal bewegt",
69            atomFunction : "Entity At Cell Has Been
                              Moved n Times",
70            atomArguments : [ -1, 0, 1 ]
71        },{
72            atomName : "Weißer Bauer oestlich wurde
                              nur einmal bewegt",
73            atomFunction : "Entity At Cell Has Been
                              Moved n Times",
74            atomArguments : [ +1, 0, 1 ]
75        },{
76            atomName : "Weißer Bauer westlich wurde
                              zuletzt bewegt",
77            atomFunction : "Entity At Cell Has Been
                              Moved In Last Game Round",
78            atomArguments : [ -1, 0 ]
79        },{
80            atomName : "Weißer Bauer oestlich wurde
                              zuletzt bewegt",
81            atomFunction : "Entity At Cell Has Been
                              Moved In Last Game Round",
82            atomArguments : [ +1, 0 ]
83        }, {
84            atomName : "Spielfigur kann die unterste
                              Reihe erreichen",
85            atomFunction : "Entity Is Able To Reach
                              A Specific Row",
```

```
86          atomArguments : [ "last", "south" ]
87      }],
```

Im letzten Teil der Blaupause des schwarzen Bauern werden die obigen Zielatome zu individuellen Zielen der Spielfigur zusammengesetzt:

```
88      // Zielatome zu Spielsteinzielen zusammensetzen
89      goals : [{
90          type     : "CAPTURE",
91          name     : "Schlage Spielfigur auf Feld suedost",
92          atoms    : ["Gegner auf suedoestlich
                            angrenzendem Feld"],
93          move     : "southeast"
94      },{
95          type     : "CAPTURE",
96          name     : "Schlage Spielfigur auf Feld suedwest",
97          atoms    : ["Gegner auf suedwestlich
                            angrenzendem Feld"],
98          move     : [ -1, +1 ]
99      },{
100         type     : "CAPTURE",
101         name     : "Schlage Gegner en passant suedwestlich",
102         atoms    : ["Figur steht einem Spielfeld
                            in Zeile vier",
103             "Weißer Bauer westlich",
104             "Weißer Bauer westlich wurde nur einmal bewegt",
105             "Weißer Bauer westlich wurde zuletzt bewegt"],
106         move     : "southwest"
107     },{
108         type     : "CAPTURE",
109         name     : "Schlage Gegner en passant suedoestlich",
110         atoms    : ["Figur steht einem Spielfeld
                            in Zeile vier",
111             "Weißer Bauer oestlich",
112             "Weißer Bauer oestlich wurde nur einmal bewegt",
113             "Weißer Bauer oestlich wurde zuletzt bewegt"],
114         move     : "southeast"
115     }, {
116         type     : "GOALMOVE",
117         name     : "Erreiche die letzte Reihe
                            des Spielbrettes",
118         atoms    : ["Leeres Feld Suedlich",
119             "Spielfigur kann die unterste Reihe erreichen"],
120         move     : "south"
121     }]
122 }
```

Auf die Definition der einzelnen Entitätenblaupausen mitsamt individueller Zugmöglichkeiten und Entitätenzielen folgt schließlich die Verknüpfung erfüllter Ziele mit ihren Auswirkungen (*consequences*) auf die Spielwelt und den Spielverlauf. Beispielhaft für die Repräsentation der Konsequenzen erfüllter Spielsteinziele im Framework seien mit dem folgenden Quelltext die

Konsequenzen repräsentiert, die sich mit Erreichen der letzten Reihe des Spielbrettes (Zeilen 2 bis 21) und dem Schlagen einer gegnerischen Spielfigur en passant (Zeilen 22 bis 45) verbinden.

```
1    blackPlayer : {
2        black_pawn_reach_last_row_south : {
3            goalName     : "Erreiche die letzte Reihe
                                des Spielbrettes",
4            entityType : "Black Pawn",
5            consequences : [
6                {
7                    jobName: "Markiere das Zielfeld",
8                    jobFunction: "Highlight Cell",
9                    jobArguments: [ 0, +1, "move_goal",
10                       "RELATIVE", "Black Pawn" ],
11                   execute: "immediately"
12               }, {
13                   jobName: "Entferne gegnerischen Spielstein",
14                   jobFunction: "Move Entity Relative To",
15                   jobArguments: [ 0, +1, "Black Pawn",
                                "captureMove" ]
16               }, {
17                   jobName: "Transformiere Bauer in Koenigin",
18                   jobFunction: "Transform Entity",
19                   jobArguments: "Black Queen"
20               }]
21       },
22       black_pawn_capture_opponent_southwest_enpassant : {
23           goalName     : "Schlage Gegner en passant
                                suedwestlich",
24           entityType : "Black Pawn",
25           consequences : [
26               {
27                   jobName: "Markiere das Zielfeld",
28                   jobFunction: "Highlight Cell",
29                   jobArguments: [ -1, +1, "move_goal",
30                       "RELATIVE", "Black Pawn" ],
31                   execute: "immediately"
32               },
33               {
34                   jobName: "Entferne gegnerischen Spielstein",
35                   jobFunction: "Capture Opponent At",
36                   jobArguments: [ -1, 0, "RELATIVE",
37                       "Black Pawn" ]
38               },
39               {
40                   jobName: "Bewege Spielfigur",
41                   jobFunction: "Move Entity Relative To",
42                   jobArguments: [ -1, +1, "Black Pawn",
                                "captureMove" ]
43               }]
44       }, // [...]
45   }
```

Konsequenzen bestehen aus mindestens einem Job, der bei erfülltem Spielfigurziel abgearbeitet wird. Für die vollständige Übersicht über alle

ausführbaren Jobs sei an dieser Stelle verwiesen auf die Dokumentation der Klasse *Spoooky.GameEvents*, die verschiedene Spielereignisse wie das Schlagen gegnerischer Spielsteine (`"Capture Opponent At"`), die Hervorhebung von Spielfeldzellen realisiert (`"Highlight Cell"`) oder die Transformation eines Spielsteines in einen Spielstein eines anderen Typs (`"Transform Entity"`) durchführt.[489]

6.3.4 Spielwelt- und Spielregeldefinition

Ein Spiel und seine Regeln wird in der JavaScript-Datei *game.js* im Ordner *games/[spielname]* beschrieben. Eingeleitet wird die Spieldefinition mit der Erstellung eines neuen Spielobjektes, der Generierung von Spielerobjekten und Angaben zum Aufbau und zur Darstellung der Spielwelt, wie mit dem folgenden Quelltext für das Schachspiel veranschaulicht:

```
1    // Ein neues SpoookyJS Spiel erstellen.
2    var game = new Spoooky.Game;
3
4    // Bootstrapping-Funktion: Controller generieren und
           initialisieren.
5    game.initialize("Schach");
6
7    // Menschlichen Spieler erstellen.
8    var player1 = game.createPlayer({
9        name: "Jan", type: "HUMAN"
10   });
11
12   // Artifizielle Gegenspielerin erstellen.
13   var player2 = game.createPlayer({
14       name: "Scully", type: "ARTIFICIAL"
15   });
16
17   // CSS-Klassennamen (vordefiniert in css/spoookystyle.css)
           zur späteren Verwendung in Variablen speichern.
18   var b = "gridCellBlack",
19       w = "gridCellWhite";
20
21   // Spielbrett mit 8x8 Spielfeldzellen erstellen.
22   // Den einzelnen Spielzellen werden mit "w" und "b"
           die entsprechenden CSS-Klassen zugeordnet.
23   game.setupGridWorld(8, 8, [
24       w, b, w, b, w, b, w, b,
25       b, w, b, w, b, w, b, w,
26       w, b, w, b, w, b, w, b,
27       b, w, b, w, b, w, b, w,
28       w, b, w, b, w, b, w, b,
29       b, w, b, w, b, w, b, w,
30       w, b, w, b, w, b, w, b,
31       b, w, b, w, b, w, b, w
```

[489] Vgl. die Memberfunktion `fireEvent` der Klasse *Spoooky.GameEvents*.

```
32    ]);
```

Anschließend werden die im vorherigen Kapitel eingehend betrachteten Spielfigurenblaupausen dem Spiel hinzugefügt, um an späterer Stelle konkrete Instanzen der Spielfiguren zu erstellen. Um das Rochieren der Könige und Türme im Schachspiel zu implementieren, muss jeder der beiden Türme beider Spieler eindeutig identifizierbar sein. So wird jedem Turm mit den Zeilen 45 bis 52 des folgenden JavaScript-Quelltextes ein Name zugewiesen, über den sich der jeweilige Turm adressieren lässt. Im letzten Schritt werden die Konsequenzen für das Spiel und die Spielwelt vorgesehen, die aus erfolgreichen Zielzügen der Spielfiguren resultieren (vgl. die Zeilen 55 bis 58):

```
33    // Entitätenblaupausen dem Spiel hinzufügen
34    var black_bishop = game.addBlueprint(player2,
35            Spoooky.Blueprints.CHESS.entities.black_bishop),
36        black_king = game.addBlueprint(player2,
37            Spoooky.Blueprints.CHESS.entities.black_king),
38        // [...] Weitere benötigte Blaupausen [...]
39        white_rook_left = game.addBlueprint(player1,
40            Spoooky.Blueprints.CHESS.entities.white_rook),
41        white_rook_right = game.addBlueprint(player1,
42            Spoooky.Blueprints.CHESS.entities.white_rook);
43
44    // Erweiterung der Entitätenblaupause,
          um Rochaden zu ermöglichen
45    game.extendBlueprint(black_rook_left,
46        { entityName : "Black Rook Left" });
47    game.extendBlueprint(black_rook_right,
48        { entityName : "Black Rook Right" });
49    game.extendBlueprint(white_rook_left,
50        { entityName : "White Rook Left" });
51    game.extendBlueprint(white_rook_right,
52        { entityName : "White Rook Right" });
53
54    // Entitätenziele mit Folgen für das Spiel
          und die Spielwelt verknüpfen
55    game.connectConsequences(Spoooky.Blueprints.CHESS.
56        consequences.blackPlayer);
57    game.connectConsequences(Spoooky.Blueprints.CHESS.
58        consequences.whitePlayer);
```

Nachdem die Spielfiguren dem Spiel zur späteren Verwendung hinzugefügt und die Auswirkungen erfolgreich aktivierter Spielfigurenziele anhand der Funktion `connectConsequences` vorgesehen wurden, generiert die Methode `addEntitiesToGameBoard` den Initialzustand der Spielwelt, indem die Spielfiguren als manipulierbare Objekte auf dem Spielbrett abgelegt werden.

Zellen des Spielfeldes, auf denen sich zu Beginn des Spieles keine Spielfiguren befinden, werden hierbei mit der Ziffer 0 markiert. Abgeschlossen ist die Erstellung der Spielerobjekte, der Spielfiguren und der Spielwelt, wenn der Spieler bestimmt wird, der das Spiel startet, wie in der letzten Zeile des nachfolgend wiedergegebenen Quelltextes vorgenommen:

```
59   // Spielfiguren auf dem Spielbrett ablegen
60   game.addEntitiesToGameBoard([
61       black_rook_left, black_knight, black_bishop,
            black_queen, black_king, black_bishop,
            black_knight, black_rook_right,
62       black_pawn, black_pawn, black_pawn,
            black_pawn, black_pawn, black_pawn,
            black_pawn, black_pawn,
63       0, 0, 0, 0, 0, 0, 0, 0,
64       0, 0, 0, 0, 0, 0, 0, 0,
65       0, 0, 0, 0, 0, 0, 0, 0,
66       0, 0, 0, 0, 0, 0, 0, 0,
67       white_pawn, white_pawn, white_pawn,
            white_pawn, white_pawn, white_pawn,
            white_pawn, white_pawn,
68       white_rook_left, white_knight, white_bishop,
            white_queen, white_king, white_bishop,
            white_knight, white_rook_right
69   ]);
70   // Spieler 1 (weiß) startet das Spiel
71   game.setPlayer(player1);
```

Vervollständigt und abgeschlossen wird die Spieldefinition durch ihre Spielregeln. Jede Spielregel konstituiert sich in *SpoookyJS* aus einem oder mehreren Spielregelatomen, wie in den nachfolgenden Subkapiteln anhand ausgewählter Regeln des Schachspieles vorgetragen.[490]

[490] Um die Übersichtlichkeit zu wahren, sind im Folgenden die Spielregeln und ihre Umsetzung in *SpoookyJS* einzig für Spieler 1 wiedergegeben. Die Spielregeln für Spielerin 2 gestalten sich in ihrer Umsetzung analog der betrachteten Spielregeldefinition.

6.3.4.1 Schachspielregel Unentschieden

Eine Schachpartie endet unentschieden, wenn sowohl Spieler 1 als auch Spielerin 2 einzig über ihre Könige verfügen. Um die Spielregel zu implementieren, werden dem Spiel zunächst die einzelnen Spielregelatome `"Spieler 1 hat nur noch eine Spielfigur"` und `"Spieler 2 hat nur noch eine Spielfigur"` hinzugefügt.[491]

```
72   game.addGameRuleAtom({
73       atomName : "Spieler 1 hat nur noch eine Spielfigur",
74       atomFunction : "Player Has Number Of Entities",
75       atomArguments : [ player1.getID(), 1 ]
76   });
77
78   game.addGameRuleAtom({
79       atomName : "Spielerin 2 hat nur noch eine Spielfigur",
80       atomFunction : "Player Has Number Of Entities",
81       atomArguments : [ player2.getID(), 1 ]
82   });
```

Anschließend werden die einzelnen Spielregelatome mit der Klassenmethode `assembleGameRule` anhand der Regelatomnamen zu einer Spielregel verknüpft und die Folgen für das Spiel und die Spielwelt mit der Funktion `connectGameRuleConsequences` vorgesehen:

```
83   game.assembleGameRule({
84       name     : "Unentschieden: König gegen König",
85       atoms    : ["Spieler 1 hat nur noch eine Spielfigur",
86           "Spielerin 2 hat nur noch eine Spielfigur"]
87   });
88   game.connectGameRuleConsequences({
89       ruleName     : "Unentschieden: König gegen König",
90       consequences : [{
91           jobName: "Spiel anhalten",
92           jobFunction: "Stop Game"
93       }, {
94           jobName: "Unentschieden-Nachricht ausgeben",
95           jobFunction: "Print Game Process",
96           jobArguments: "Unentschieden."
97   }]});
```

Welche Konsequenzen eine erfüllte Spielregel hervorbringt, ist bestimmt durch eine oder mehrere Aufgaben (*Jobs*), die mit der Spielregel verknüpft sind. Alle

[491] Alle ausführbaren Spielregelatome wie die Abfrage der Spielfigurenanzahl eines Spielers oder die Prüfung, ob eine Spielfigur von einer gegnerischen Figur bedroht wird, sind definiert in der zugriffsperformanten Hashtabelle *Spoooky.Game.gameRuleAtoms*.

verknüpfbaren Aufgaben, die sich als Zeichenkettenkonstante in `jobFunction` (vgl. die Zeilen 92 und 95 des zuvor wiedergegebenen Quelltextes) angeben lassen, sind in der Klasse *Spoooky.GameEvents* in einer zugriffsperformanten Hashtabelle definiert.

6.3.4.2 Schachspielregel König steht im Schach

Um die Spielregel *König steht im Schach* zu implementieren, wird einzig ein Spielregelatom benötigt, das prüft, ob der König eines Spielers im aktuellen Zustand der Spielwelt von einer gegnerischen Spielfigur bedroht wird:

```
98   game.addGameRuleAtom({
99       atomName : "König von Spieler 1 steht im Schach",
100      atomFunction : "Entity Is Under Attack",
101      atomArguments : white_king
102  });
103
104  game.assembleGameRule({
105      name     : "Schach (Spieler 1)",
106      atoms    : ["König von Spieler 1 steht im Schach"]
107  });
```

Ist das Spielregelatom erfüllt, so erfolgt zunächst eine entsprechende Meldung im Spielverlaufsdialog. Anschließend wird der Spielerwechsel durchgeführt und der Spielzustand von `"INGAME"` auf `"CHECK"` gesetzt, um die Zugmöglichkeiten der Spielfiguren des bedrohten Spielers einzuschränken und die Spielregel *Schachmatt* vorzubereiten:

```
108  game.connectGameRuleConsequences({
109      ruleName     : "Schach (Spieler 1)",
110      consequences : [{
111          jobName: "Ausgabe, dass der König von Spieler 1
                       im Schach steht",
112          jobFunction: "Print Game Process",
113        jobArguments: "Der König von Spieler 1 steht im Schach."
114      },{
115          jobName: "Spielerwechsel",
116          jobFunction: "Next Player"
117      },{
118          jobName: "Spielzustand auf CHECK setzen",
119          jobFunction: "Set Game State",
120          jobArguments: "CHECK"
121  }]});
```

6.3.4.3 Schachspielregel Schachmatt

Ein Spieler ist schachmatt gesetzt, wenn kein eigener Spielstein den eigenen König zu schützen vermag, wie mit dem nachfolgenden JavaScript-Quelltext zur Implementation der entsprechenden Spielregel und ihren Auswirkungen auf das Spiel veranschaulicht:

```
122  game.addGameRuleAtom({
123      atomName : "Keine Spielfigur von Spieler 1
                        kann dem eigenen König helfen.",
124      atomFunction : "Got No Protecting Entities",
125      atomArguments : white_king
126  });
127  game.addGameRuleAtom({
128      atomName : "Das Spiel ist im Zustand CHECK",
129      atomFunction : "Game State Is",
130      atomArguments : "CHECK"
131  });
132  game.assembleGameRule({
133      name     : "Schachmatt (Spieler 1)",
134      atoms    : ["König von Spieler 1 steht im Schach",
135          "Keine Spielfigur von Spieler 1
                        kann dem eigenen König helfen.",
136          "Das Spiel ist im Zustand CHECK"]
137  });
138
139  game.connectGameRuleConsequences({
140      ruleName     : "Schachmatt (Spieler 1)",
141      consequences : [{
142          jobName: "Spiel beenden",
143          jobFunction: "Stop Game"
144      }, {
145          jobName: "Ausgabe im Spielverlaufsdialog, dass
                        Spieler 1 das Spiel verloren hat.",
146          jobFunction: "Print Game Process",
147          jobArguments: "Spieler 1 ist schachmatt."
148      },{
149          jobName: "Ausgabe im Spielverlaufsdialog, dass
                        Spielerin 2 das Spiel gewonnen hat.",
150          jobFunction: "Print Game Process",
151          jobArguments: "Spielerin 2 gewinnt das Spiel."
152      },{
153          jobName: "ID der Spielerin speichern,
                        die das Spiel gewonnen hat.",
154          jobFunction: "Set Winner",
155          jobArguments: player2.getID()
156  }]});
```

6.3.4.4 Schachspielregel Patt

Als unentschieden wird das Spiel gewertet, wenn ein Spieler über keine gültige Zugmöglichkeit verfügt und sich sein König nicht im Schach befindet:

```
157  game.addGameRuleAtom({
158      atomName : "Keine Spielfigur von Spieler 1
                        kann sich bewegen.",
159      atomFunction : "Player Has No Movable Entities",
160      atomArguments : player1.getID()
161  });
162
163  game.addGameRuleAtom({
164      atomName : "Das Spiel ist im Zustand INGAME",
165      atomFunction : "Game State Is",
166      atomArguments : "INGAME"
167  });
168
169  game.assembleGameRule({
170      name     : "Patt (Spieler 1)",
171      atoms    : ["Keine Spielfigur von Spieler 1
                        kann sich bewegen.",
172          "Das Spiel ist im Zustand INGAME"]
173  });
174
175  game.connectGameRuleConsequences({
176      ruleName     : "Patt (Spieler 1)",
177      consequences : [{
178          jobName: "Spiel beenden",
179          jobFunction: "Stop Game"
180      }, {
181          jobName: "Ausgabe im Spielverlaufsdialog, dass
                        Spieler 1 ueber keine gueltige
                        Zugmoeglichkeit verfuegt.",
182          jobFunction: "Print Game Process",
183          jobArguments: "Patt: Spieler 1 verfuegt ueber
                        keine gueltige Zugmoeglichkeit."
184  }]});
```

6.3.5 Spielebaukasten – Zur Definition unterschiedlicher Spielarten

Die Erstellung des Schachspieles, wie sie in den vorangegangenen Kapiteln vorgetragen und demonstriert wurde, gestaltet sich mit dem Framework *SpoookyJS* in wenigen Arbeitsschritten: Zuerst werden die Spielfiguren einschließlich ihrer Zugmöglichkeiten und individueller Aktionsziele in Form wiederverwendbarer Blaupausen vorbereitet. Anschließend werden die unterschiedlichen Spieleigenschaften in der spieldedizierten Skriptdatei *game.js* vorgesehen – Spieleigenschaften wie die Spielerobjekte, der Aufbau und die Beschaffenheit der Spielwelt sowie die Spielregeln, die sich aus Spielregelatomen konstituieren. Vollständig beschrieben, präsentiert sich das Schachspiel im Webbrowser wie auf dem folgenden Screenshot dargestellt:

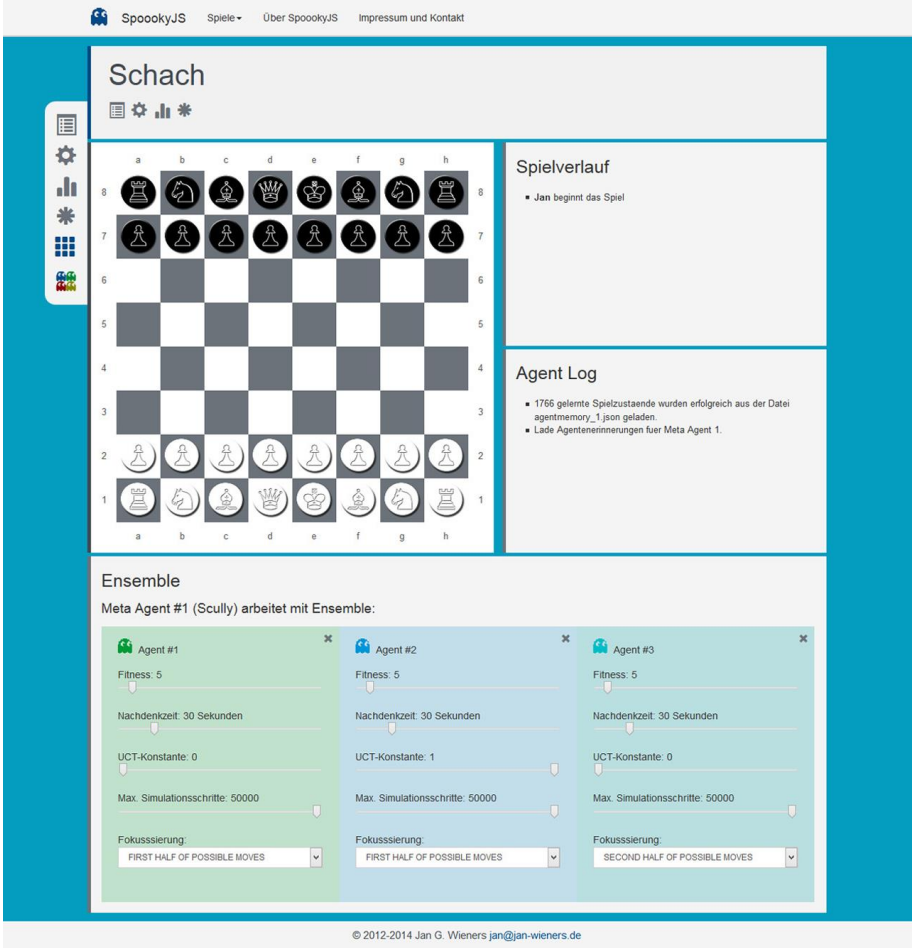


Abbildung 65: Bildschirmausschnitt des mit *SpoookyJS* implementierten Schachspieles im Webbrowser *Firefox*.

Wurde in den vorangegangenen Kapiteln ausführlich eingegangen auf die Erstellung zugbasierter Spiele mit zweidimensionalen Spielwelten, erläutern die

nachfolgenden Subkapitel, wie sich Spiele mit dem Framework umsetzen lassen, in denen Spielfiguren auf dem Spielbrett abgelegt werden, wie sich lineare und netzartige Spielwelten erstellen lassen – und erörtern die Einbindung des Zufallsmomentes, wie es im Backgammonspiel in Form von Spielwürfeln benötigt wird.

6.3.5.1 Interaktionsmodi

In der Entwicklung digitaler webbasierter Brettspiele fungiert *SpoookyJS* als JavaScript-basierter Spielebaukasten, mit dem sich rundenbasierte Nullsummenspiele mit perfekter Information und zweidimensionaler Spielwelt wie die des Schach- oder Damespieles implementieren lassen. Wie im vorangegangenen Kapitel 6.2.3 besprochen, differenziert die Spielschleife `game.loop` den Spielmodus, der als Zeichenkettenkonstante repräsentiert ist und bestimmt, wie mit den Spielfiguren interagiert werden kann: Werden Spielfiguren auf dem Spielbrett bewegt, wie im Schachspiel, wird das Spiel im Modus `"MOVING"` betrieben; werden Spielfiguren auf Spielfeldzellen abgelegt, nimmt das Spiel in der Betriebsart `"PLACING"` seinen Lauf. Ändern lässt sich der voreingestellte Spielmodus `"MOVING"` durch die Funktion `setGameMode`, wie mit dem folgenden Quelltextausschnitt aus der Skriptdatei *gomoku15x15/game.js* veranschaulicht, die der Definition des Gomoku-Spieles dient:

```
1    // Eine neue SpoookyJS Instanz erstellen
2    var game = new Spoooky.Game;
3    // Controller und Spielinterna initialisieren
4    game.initialize("Gomoku 15x15");
5    // Spielmodus setzen, um Legespiel zu realisieren
6    game.setGameMode("PLACING");
```

Durch das Spielereignis `"Change Game Mode"`, wie es in der Klasse *Spoooky.GameEvents* definiert ist, lässt sich der Spielmodus dynamisch während des laufenden Spieles wechseln, so dass die Implementation von Spielen wie Amazons (*Spiel der Amazonen*) ermöglicht wird.

6.3.5.2 Lineare und netzartige Spielwelten

Ob seiner andersartigen visuellen Darstellung und des Zufallsmomentes in Form sechsseitiger Würfel ist das Backgammonspiel ein dienlicher Prüfstein für die

Umsetzung unterschiedlichster Arten von Spielen mit *SpoookyJS*. Anders als die Brettspiele Schach, Dame, Amazons oder Gomoku bewegen sich die Spielfiguren im Backgammonspiel in einer linearen Spielwelt, wie in der folgenden Darstellung veranschaulicht:

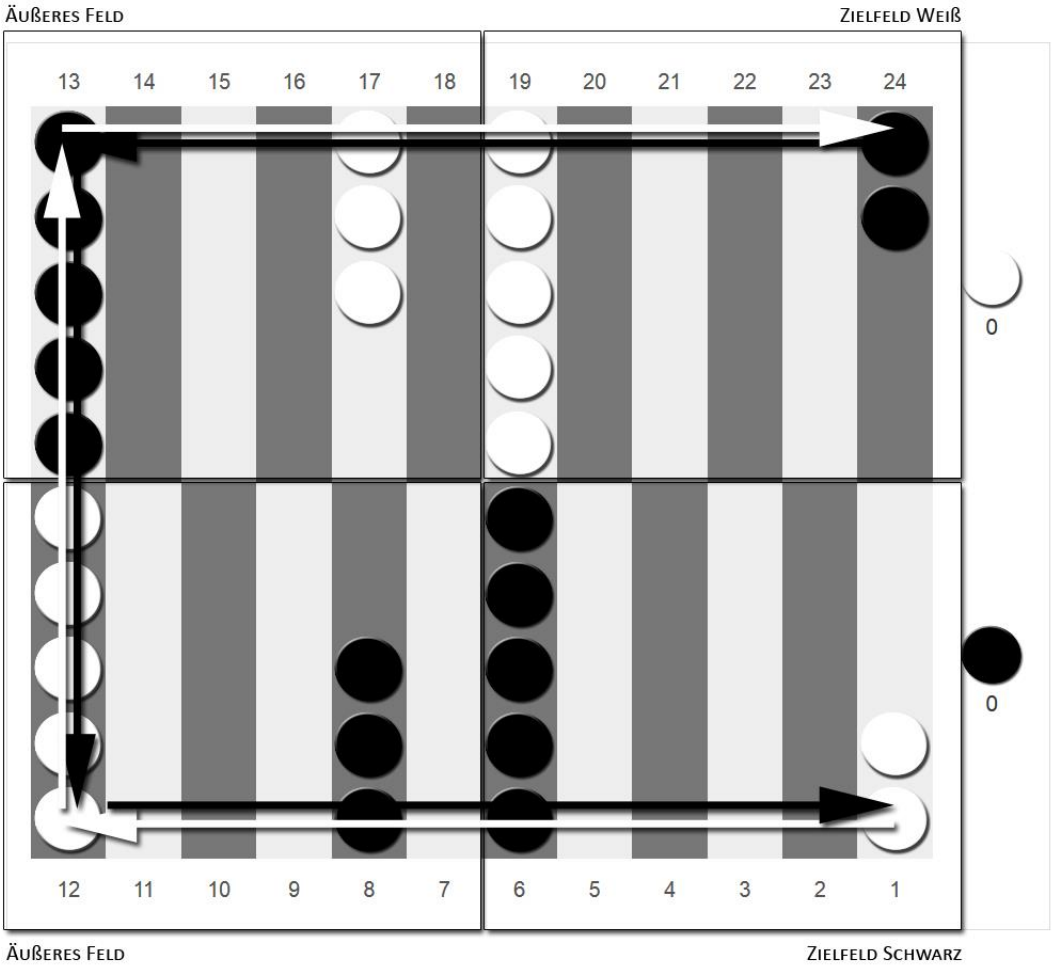


Abbildung 66: Darstellung der Backgammonspielwelt mit eingezeichneten Zugrichtungen der beiden Spieler Weiß und Schwarz.

Die Spielwelt des Backgammonspieles besteht aus 24 Teilen, den *Zungen*. Jede der 24 Zungen nimmt eine beliebige Anzahl von Spielfiguren der beiden Spieler Schwarz und Weiß auf. Eine Zunge, auf der sich zwei oder mehr Steine eines Spielers befinden, wird als *Band* bezeichnet; als *Block* sind zwei oder mehr Bänder signifiziert, die unmittelbar nebeneinander liegen. Die Bewegung der Spielsteine auf dem Spielbrett erfolgt entgegengesetzt: Weiß zieht seine Spielsteine im Uhrzeigersinn und Schwarz bewegt seine Steine gegen den Uhrzeigersinn nach den Augen der Würfel in Richtung seines Zielfeldes.

Um die lineare Spielwelt mit *SpoookyJS* abzubilden, wird das Spielbrett in der Skriptdatei *game.js* des Backgammonspieles zuerst initialisiert. Durch die Angabe der CSS-Klassen `"gridCellBlack"` und `"gridCellWhite"` generiert sich zuerst die Darstellung der Spielwelt:

```
1     var b = "gridCellBlack",
2         w = "gridCellWhite";
3
4     game.setupGridWorld(12, 10, [
5         w, b, w, b, w, b, w, b, w, b, w, b,
6         w, b, w, b, w, b, w, b, w, b, w, b,
7         w, b, w, b, w, b, w, b, w, b, w, b,
8         w, b, w, b, w, b, w, b, w, b, w, b,
9         w, b, w, b, w, b, w, b, w, b, w, b,
10        b, w, b, w, b, w, b, w, b, w, b, w,
11        b, w, b, w, b, w, b, w, b, w, b, w,
12        b, w, b, w, b, w, b, w, b, w, b, w,
13        b, w, b, w, b, w, b, w, b, w, b, w,
14        b, w, b, w, b, w, b, w, b, w, b, w
15    ]);
```

Anschließend werden mit der Funktion `setFieldIDs` eindeutige Identifizierer vergeben, um die Bewegungsmöglichkeiten der Spielfiguren auf der x-Achse der linearen Spielwelt zu ermöglichen. Den einzelnen Feldern einer Zunge wird hierfür ein ganzzahliger Identifizierer vergeben. So erhalten die fünf Felder der weißen Zunge am rechten unteren Spielbrettrand eine 1 und die fünf Felder der schwarzen Zunge am oberen rechten Spielbrettrand die Zahl 24 zugewiesen:

```
16    game.getGameWorld().setFieldIDs([
17        13,14,15,16,17,18,19,20,21,22,23,24,
18        13,14,15,16,17,18,19,20,21,22,23,24,
19        13,14,15,16,17,18,19,20,21,22,23,24,
20        13,14,15,16,17,18,19,20,21,22,23,24,
21        13,14,15,16,17,18,19,20,21,22,23,24,
22        12,11,10,9,8,7,6,5,4,3,2,1,
23        12,11,10,9,8,7,6,5,4,3,2,1,
24        12,11,10,9,8,7,6,5,4,3,2,1,
25        12,11,10,9,8,7,6,5,4,3,2,1,
26        12,11,10,9,8,7,6,5,4,3,2,1
27    ]);
```

Um die Interaktionsmöglichkeit einzuschränken, so dass ausschließlich die zuoberst auf den Zungen liegenden Spielfiguren selektierbar sind, findet sich der Kopfbereich der Entitätenblaupausen ergänzt um die Auswahlbedingung `"selectCondition"`, mit der geprüft wird, auf welchem Feld der y-Achse sich die

jeweilige Spielfigur befindet – und ob die ausgewählte Spielfigur in nördlicher und südlicher Richtung von anderen Spielfiguren umgeben ist:

```
1    player1StandardEntity : {
2
3        typeID : "A",
4        entityType : "Spielfigur von Spielerin 1",
5        associatedWithMetaAgent : null,
6        representation : { type : "image",
            texture : "assets/white_01.png" },
7        selectCondition : {
8            neighboursY : [{
9                count: 0, condition: "EQUAL",
                 direction: "NORTH", appliesTo: {
10                   axis: "y",
11                   operator: ">",
12                   value: 4 }
13           },{ count: 0, condition: "EQUAL",
                 direction: "SOUTH", appliesTo: {
14               axis: "y",
15               operator: "<",
16               value: 5 }
17           }]
18       },
```

An die Auswahlbedingung schließt sich der Zugbereich der Entitätenblaupause an, mit dem das Vermögen der Spielfigur realisiert wird, sich anhand der Würfelaugen in der linearen Spielwelt zu bewegen. Der nachfolgend wiedergegebene Quelltextauszug gibt den Zugbereich der weißen Spielfiguren wieder, die sich in positiver Richtung – von den Zungen eins bis 24 – auf leere Spielfelder zu bewegen vermögen:

```
19       moves : [{
20           name : "Zug nach Wuerfelaugen",
21           type : "By Field ID",
22           direction : ["X-AXIS", "POSITIVE"],
23           frequency : "DICE",
24           conditions : [{ condition : "Is Empty",
                                state : true }]
25       }],
```

Die individuellen Ziele der Spielfiguren bestehen darin, isolierte gegnerische Spielsteine zu schlagen, das eigene Zielfeld zu erreichen sowie den Auswürfelbereich zu erreichen, wie mit den folgenden Regelatomen und zusammengesetzten Regeln formuliert:

```
26        goalAtoms : [{
27            atomName : "Eine gegnerische Spielfigur
                              befindet sich auf dem Zielfeld",
28            atomFunction : "Number Of Opponents
                              At Destination Field Is",
29            atomArguments : ["MOVE POSITIVE", 1]
30        },{
31            atomName : "Jede Spielfigur befindet sich im
                              letzten Drittel des Spielbrettes",
32            atomFunction : "No Own Entitys On Fields With ID
                              Less Than",
33            atomArguments : 19
34        },{
35            atomName : "Spielfigur kann den Auswuerfelbereich
                              erreichen",
36            atomFunction : "Destination Field ID Is
                              Greater Than",
37            atomArguments : ["MOVE POSITIVE", 24]
38        }],
39        goals : [{
40            type     : "CAPTURE",
41            name     : "Schlage gegnerische Zielfigur
                              auf dem Spielfeld",
42            atoms    : ["Eine gegnerische Spielfigur
                              befindet sich auf dem Zielfeld"],
43            move     : "MOVE POSITIVE"
44        },{
45            type     : "BEAROFF",
46            name     : "Bear off",
47            weight   : 200,
48            atoms    : ["Jede Spielfigur befindet sich im letzten
                              Drittel des Spielbrettes",
49                "Spielfigur kann den Auswuerfelbereich
                              erreichen"],
50            move     : "BEAR OFF MOVE",
51            area : "player1BearOffArea"
52        }]
53    }
```

6.3.5.3 Spielbereiche jenseits der Spielwelt mit Spoooky.Areas

Gemeinhin generiert *SpoookyJS* eine zweidimensionale Spielwelt, die sich durch die zuvor betrachtete Methode `setFieldIDs` in lineare und netzartige Spielwelten transformieren lässt. Spielbereiche jenseits der Spielwelt wie die beiden Auswürfelbereiche im Backgammonspiel lassen sich mit der Klasse *Spoooky.Areas* realisieren. Eine Instanz der Klasse kann eine unbeschränkte Anzahl von Spielbereichen verwalten und realisiert die Zugriffsmöglichkeiten auf die Bereiche. Dem Spiel hinzugefügt wird ein separierter Spielbereich in der Skriptdatei *game.js* über die Methode `addArea` der Klasse *Spoooky.Game*. Für das Backgammonspiel wurden die beiden Bereiche mit den folgenden Anweisungen hinzugefügt:

```
game.addArea("player1BearOffArea");

game.addArea("player2BearOffArea");
```

Sowohl die Funktionalität als auch das Erscheinungsbild der mit der Methode `addArea` vorgesehenen Spielbereiche werden individuell im Spielskript *game.js* formuliert und in der spieldedizierten HTML-Datei *index.htm* im entsprechenden Ordner bestimmt.[492]

6.3.5.4 Spiele mit Zufallselementen – Spielwürfel mit Spoooky.Dice

Um das Moment des Zufalls in Spielen zu realisieren, bietet das Framework mit seiner Klasse *Spoooky.DiceBox* Funktionalität, um eine beliebige Menge von Würfeln mit selbstdefinierter Augenanzahl zu generieren und zu verwalten. Erstellt werden die beiden sechsäugigen Würfel des Backgammonspieles im Spielskript *game.js* mit der Funktion `addDice` der Klasse *Spoooky.Game*, wie mit den beiden folgenden Quelltextzeilen veranschaulicht:[493]

```
game.addDice(1, 6);

game.addDice(1, 6);
```

Der rundenbasierte Spielverlauf wird anschließend über die Spielregelatome und zusammengesetzte Spielregeln realisiert: Verzeichnet die Klasseninstanz von *Spoooky.DiceBox* keinen ungespielten Würfel mehr, so wird im Backgammonspiel die nächste Spielrunde begonnen. Veranschaulicht sei der Rundenwechsel mit dem folgenden Ausschnitt aus der Skriptdatei *game.js*:

```
1    game.addGameRuleAtom({
2        atomName : "Jeder Wuerfel wurde gespielt",
3        atomFunction : "Dice Box Is Empty"
4    });
5    game.assembleGameRule({
6        name     : "Naechste Spielrunde, wenn jeder Wuerfel
                        gespielt wurde",
7        atoms    : ["Game State is INGAME",
8            "Jeder Wuerfel wurde gespielt"]
9    });
10   game.connectGameRuleConsequences({
```

[492] Für das Backgammonspiel vgl. das Spielskript *game.js* sowie die HTML-Datei *index.htm* im Ordner *games/backgammon*.

[493] Der erste Parameter der Funktion `addDice` bestimmt den Start- und der zweite Parameter den Endbereich des Würfels. Ein Würfel mit 42 Augen ließe sich folglich generieren mit dem Funktionsaufruf `game.addDice(1, 42)`.

```
11        ruleName     : "Naechste Spielrunde, wenn jeder Wuerfel
                         gespielt wurde",
12        consequences : [{
13            jobName: "Change current player",
14            jobFunction: "Next Player"
15        },{
16            jobName: "Set Game State",
17            jobFunction: "Set Game State",
18            jobArguments: "WAITINGFORDICEROLL"
19        },{
20            jobName: "Restrict selectable entities to
                         off board entities",
21            jobFunction: "Restrict Selectable Entities",
22            jobArguments: null
23        }]});
```

Analog der Gestaltung der mit *Spoooky.Areas* erstellten Spielbereiche, erfolgt auch die visuelle Umsetzung der Würfel manuell und individuell. Vorlagerastergrafiken zur Darstellung der Würfelaugen sind im Ordner *games/_template/assets* bereitgestellt.

6.4 Spielübergreifende künstliche Intelligenz in SpoookJS

SpoookyJS verfolgt einen multiagentenbasierten evolutionären Ansatz zur Umsetzung spielübergreifender künstlicher Intelligenz. Für jedes Spiel, das mit dem JavaScript-Framework umgesetzt wird, stellt *SpoookyJS* artifizielle Spielgegner bereit, die *out of the box*, d.h. ohne menschliches Expertenwissen, ohne externe Anpassungen in dem jeweiligen Spiel zu agieren vermögen. Im Mittelpunkt der Entscheidungsfindung steht das Modul *Spoooky.MetaAgent*, über das sich artifizielle Agenten erstellen lassen, die ein Team assoziierter Agenten beauftragen, um den aktuellen Spielzustand auf auszahlungsmaximierende Zugmöglichkeiten zu analysieren. Jeder Agent des Agententeams ist eine Instanz der Klasse *Spoooky.Agent* und verfährt als eigener Thread im Webbrowser mit den Routinen der KI-Bibliothek *Spoooky.AI*.

In den nachfolgenden Kapiteln wird zunächst auf den Meta Agenten und seine assoziierten Agenten eingegangen. Anschließend wird der Aufbau und die Verfahrensweise des Agentenkollektivs in der Entscheidungsfindung besprochen und wird mit der Agentenfitness sowie der Tradierung assoziierter Agenten und ihrer Eigenschaften das evolutionäre Moment des Multiagentensystems angemerkt. Abgeschlossen wird das Kapitel mit der Diskussion über Möglichkeiten und Grenzen des spielübergreifenden Multiagentenansatzes, der in seinem Kern mit der Monte Carlo Spielbaumsuche und dem *Upper Confidence Bounds applied to Trees* Algorithmus arbeitet.

6.4.1 Spoooky.MetaAgent und Agentenensemble

Wie in Kapitel 6.2.4 ausgeführt, finden sich die Spieler repräsentiert durch die Klasse *Spoooky.MetaAgent*, mit der sich artifizielle Opponenten instanziieren lassen. Seinen Namen *Meta Agent* trägt jedes künstliche Spielerobjekt ob seiner herausgehobenen Stellung in der Hierarchie entscheidungsrelevanter Agenten: Jeder Meta Agent verfügt über ein Team dedizierter Agenten, im Folgenden als *Ensemble* bezeichnet, die unterschiedliche Aspekte des aktuellen Spielzustandes untersuchen und dem Meta Agenten Handlungsvorschläge unterbreiten. [494] Visualisiert sei der Aufbau der Klassen *Spoooky.MetaAgent* und *Spoooky.Agent* sowie die Art ihrer Assoziation mit dem folgenden UML-Klassendiagramm:[495]

[494] Die Bezeichnung des Agententeams als Agentenensemble folgt den Ausführungen von Fern und Lewis 2011, die mit ihrem *Ensemble UCT* Ansatz verschiedene parallel arbeitende Instanzen der Monte Carlo Spielbaumsuche dazu verwenden, um die Güte von Zugmöglichkeiten zu bewerten (vgl. auch Browne et al. 2012, S. 12).

[495] Mit der Abkürzung *UML* ist die *Unified Modeling Language* signifiziert, die der „Modellierung, Dokumentation, Spezifizierung und Visualisierung komplexer Softwaresysteme, unabhängig von deren Fach- und Realisierungsgebiet [dient]. Sie liefert die Notationselemente gleichermaßen für die statischen und dynamischen Modelle von Analyse, Design und Architektur und unterstützt insbesondere objektorientierte Vorgehensweisen.“ (Rupp et al. 2007, S. 12).

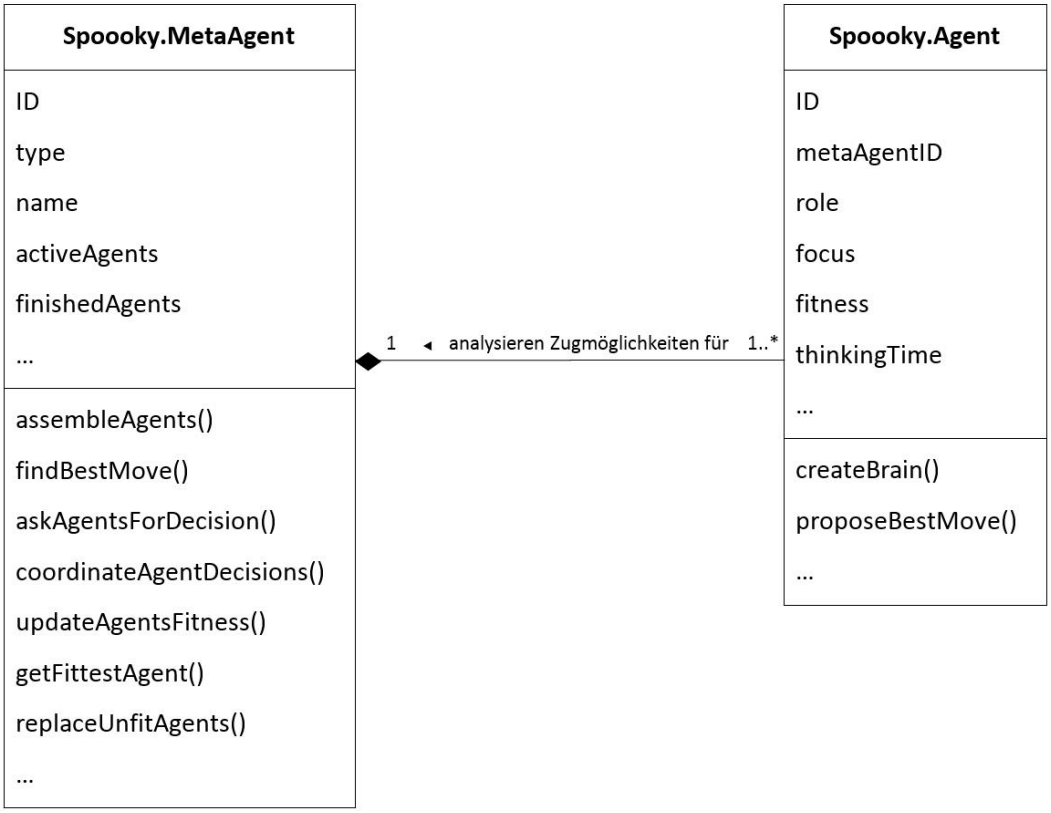


Abbildung 67: Klassendiagramm *Spoooky.MetaAgent* und *Spoooky.Agent*: Eine beliebige Anzahl von mehr als einem Agenten (*Spoooky.Agent*, Häufigkeit 1..*) analysieren Zugmöglichkeiten für einen Meta Agenten (*Spoooky.MetaAgent*).

Wie in dem UML-Diagramm skizziert, verfügt ein Meta Agent über die Fabrikmethode [496] `assembleAgents` zur Erstellung von Agenten, die der Entscheidungsfindung des Meta Agenten dienen. Über die Methode `askAgentsForDecision` fragt der Meta Agent seine assoziierten Agenten, welche der zum aktuellen Spielzustand ausführbaren Aktionen die vermutlich beste Handlung ist. Alle dem Meta Agenten zugeordneten Agenten beschäftigen sich auf unterschiedliche und individuelle Art mit der Spielwelt in ihrem aktuellen Zustand und schlagen dem Meta Agenten nach separater Analyse des Spielzustandes eine Zugmöglichkeit vor, wie in der folgenden Darstellung veranschaulicht.

[496] Vgl. Gamma 1995, S. 107–116.

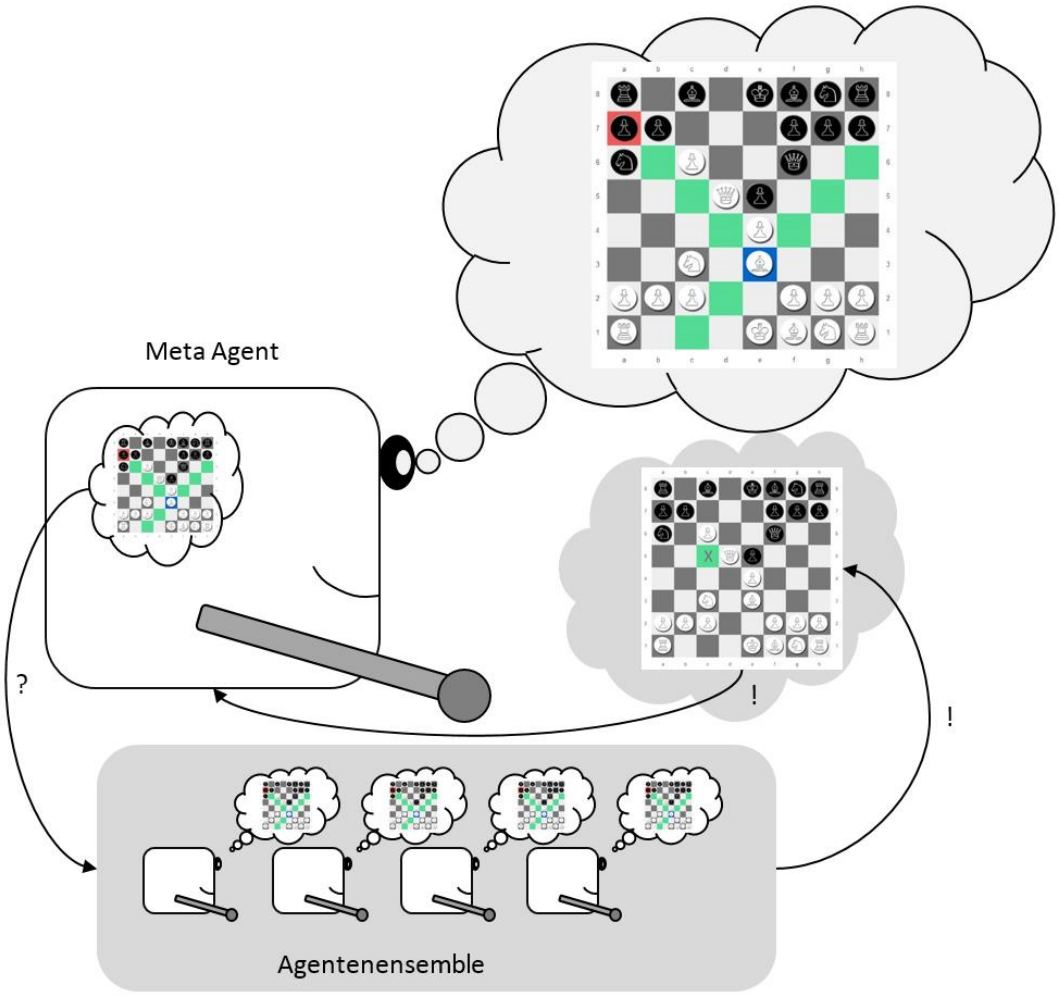


Abbildung 68: Prozess der Entscheidungsfindung auf Grundlage des Agentenensembles. Der Meta Agent nimmt eine Problemstellung wahr – im vorliegenden Falle einen Zustand des Schachspieles – und beauftragt sein Agentenensemble damit, Lösungen für das an ihn gestellte Entscheidungsproblem zu erarbeiten. Nach Analyse des Problems schlägt das Agentenensemble dem Meta Agenten Lösungsmöglichkeiten für das gestellte Problem vor.

Ein jeder Agent des Agentenensembles bildet eine Instanz der Klasse *Spoooky.Agent* und verfügt über eine individuelle Strategie, sich dem zu lösenden Entscheidungsproblem zu nähern. Wie sich die Entscheidungsfindung im Agentenensemble vollzieht, darauf sei in den nachfolgenden Kapiteln ausführlich eingegangen.

6.4.1.1 Agentenfokus

Jeder mit dem Meta Agenten assoziierte Agent verfügt über das Attribut `focus`, das bestimmt, welche Aspekte des Spieles vom Agenten eingehend betrachtet werden. Zeichenkettenkonstanten wie die im Folgenden wiedergegebenen bestimmen hierbei die Gerichtetheit des Agenten auf die Spielwelt, den individuellen Analysefokus des Agenten:

- `"ALL"`: Ein Agent mit dem Fokus `"ALL"` intendiert, alle zum Zeitpunkt der Betrachtung ausführbaren Zugmöglichkeiten zu analysieren.
- `"FIRST HALF OF POSSIBLE MOVES"` oder `"SECOND HALF OF POSSIBLE MOVES"`: Ein Agent mit einer dieser beiden Fokussierungsarten teilt den Raum aller im aktuellen[497] Spielzustand möglichen Aktionen auf in zwei Hälften und analysiert einzig Zugmöglichkeiten, die sich im ersten bzw. zweiten Aktionsbereich befinden.
- `"MOVES NEAR OPPONENT FIELDS"`: Ein Agent mit diesem Agentenfokus untersucht alle Zugmöglichkeiten, die auf Felder der Spielwelt führen, die an gegnerische Spielsteine angrenzen.
- `"MOVES NEAR OPPONENT OR OWN FIELDS"`: Ein Agent mit jenem Agentenfokus wertet alle Zugmöglichkeiten aus, die auf Spielfelder führen, die an gegnerische Spielsteine oder an eigene Spielsteine angrenzen.

Die Implementation der unterschiedlichen Möglichkeiten eines Agenten, sich mit seiner Umwelt auseinanderzusetzen, lässt sich im Framework leicht um weitere Kriterien und Betrachtungsweisen erweitern[498] und resultiert im vorliegenden Falle aus der Empirie, d.h. der Betrachtung verschiedenartiger Spiele und ihrer Charakteristika: Im Gomoku-Spiel besteht das Ziel jeder Spielerin und jedes Spielers darin, fünf eigene Spielsteine in horizontaler, diagonaler oder vertikaler Folge auf dem Spielbrett abzulegen. Anstatt jeden möglichen Zug in der großen Spielwelt zu betrachten, ist es im Gomoku-Spiel zumeist sinnvoll, Züge eingehender zu betrachten, die auf Spielfelder führen, die an gegnerische oder eigene Spielsteine angrenzen. Im Schach- oder Damespiel ist dieser Ansatz mitunter weniger erfolgreich und andere Fokussierungsmöglichkeiten erachten sich als sinnvoll.

Veranschaulicht seien die unterschiedlichen Fokussierungsweisen der Agenten mit der folgenden Tic Tac Toe Partie, die in einer Spielwelt von 5×5 Spielfeldern ausgetragen wird. Nach drei Spielrunden, in denen die Spieler abwechselnd ihre

[497] Mit dem *aktuellen* Spielzustand ist der Spielzustand zum Zeitpunkt der Entscheidungsfindung bezeichnet.
[498] Um den Agentenfokus auf eigendefinierte Spielaspekte zu richten, ist die Methode `getExecutableMovesByAgentFocus` der Klasse *Spoooky.MetaAgent* zu ergänzen.

weißen und schwarzen Spielsteine auf dem Spielbrett abgelegt haben, stellt sich der Spielzustand wie folgend dar:

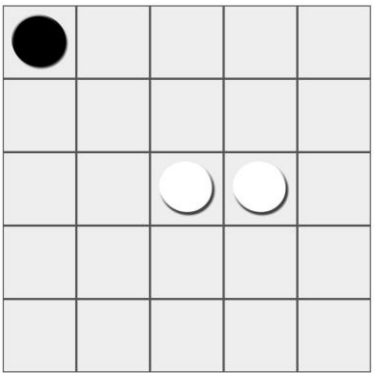


Abbildung 69: Initialzustand des 5x5 *Tic Tac Toe* Spieles zur Verdeutlichung der Agentenfokussierung.

Agenten, die den Analysefokus `"ALL"` verfolgen, intendieren, im gegebenen Zeitrahmen alle Zugmöglichkeiten zu analysieren. Wie aus nachfolgender Grafik ersichtlich, obliegt es einem Agenten mit der Fokussierungsweise `"ALL"` folglich, im aktuellen Zustand des Spieles, 22 Zugmöglichkeiten zu untersuchen:

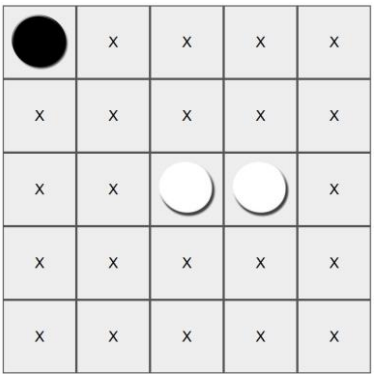


Abbildung 70: Zu analysierende Zugmöglichkeiten eines Agenten mit der Fokussierungsweise „ALL“.

Konzentriert sich der Agent auf die erste oder zweite Hälfte der Zugmöglichkeiten des aktuellen Spielzustandes, fokussiert er den halbierten Aktionsraum von jeweils elf Zugmöglichkeiten:

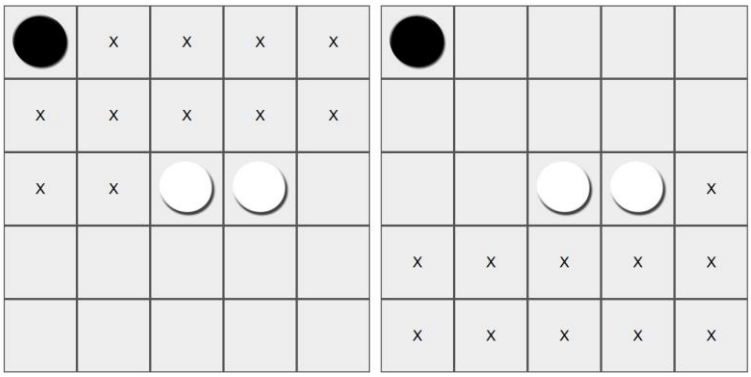


Abbildung 71: Analysefokus „SECOND HALF OF POSSIBLE MOVES“ (links) und „FIRST HALF OF POSSIBLE MOVES“.

Richtet der Agent seine Aufmerksamkeit auf Zugmöglichkeiten, die an gegnerische und / oder eigene Spielfelder angrenzen, reduzieren sich die zu untersuchenden Zugmöglichkeiten auf zehn oder zwölf ausführbare Züge:

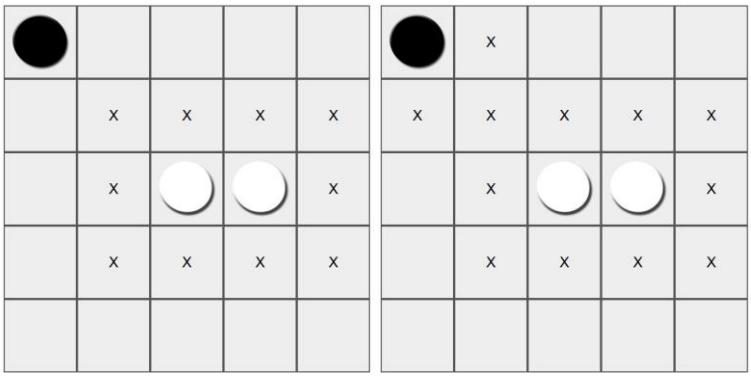


Abbildung 72: Analysefokus „MOVES NEAR OPPONENT FIELDS“ (links) und „MOVES NEAR OPPONENT OR OWN FIELDS“.

Wie im nachfolgenden Kapitel vorgestellt, vollzieht sich die Analysetätigkeit jedes Agenten durch das Modul des Agentengehirns, das als leistungsfähiger *Web Worker* umgesetzt ist und die Monte Carlo Spielbaumsuche als Kernalgorithmus zur Identifikation sinnvoller Zugmöglichkeiten verwendet.

6.4.1.2 Agentengehirn

Clientseitiges JavaScript wird im Webbrowser für gewöhnlich in einem singulären Thread ausgeführt – ist eine JavaScript-Funktion durch eine lange Laufzeit charakterisiert, so verhindert sie folglich die Ausführung weiterer Funktionen und so blockiert sie die Interaktionsmöglichkeit mit dem Webbrowser und der

Webseite.[499] Um die bestmöglichste Ressourcennutzung zu garantieren, die Arbeitslast auf mehrere Prozesse und physische Prozessoren zu verteilen und das Browserinterface während der agentenbasierten Entscheidungsfindung nicht zu blockieren, ist die Analysephase jedes Agenten anhand eines agentendedizierten *Web Workers* implementiert.

Web Worker fanden ihren Eingang in die Webentwicklung mit der fünften HTML-Version[500] und laufen in einem eigenen, separierten Thread im Browser ohne direkten Zugriff auf das *Document Object Model* (*DOM*), das die Schnittstelle für Interaktions- und Manipulationsmöglichkeiten von Elementen des HTML-Dokumentes definiert. Formuliert werden Web Worker in individuellen JavaScript-Dateien, die durch die JavaScript-Anweisung

```
var webWorker = new Worker("workerscript.js");
```

verarbeitet werden und den Web Worker ins Leben rufen. Weil ein jeder Web Worker in seiner eigenen Umgebung arbeitet, in der sich die Mittel der Programmiersprache JavaScript eingeschränkt finden,[501] erfolgt die Kommunikation mit dem Hauptthread durch den Austausch von Nachrichten. Um die Nachrichtenübertragung zwischen dem Web Worker – ob seiner entscheidungsbestimmenden Ausprägung im Folgenden auch als das *Gehirn* des Agenten bezeichnet – und dem Meta Agenten zu ermöglichen, wird die Funktion `createBrain` aufgerufen, die einen agentendedizierten Web Worker erstellt und Mitteilungen des Web Workers mit Handlungsbitten für den Meta Agenten verknüpft. Im folgenden Quelltextauszug aus der Funktion `createBrain` der Klasse *Spoooky.Agent* veranschaulicht sich das Verhalten des Agenten: Hat der Agent eine Lösung für das an ihn gestellte Problem identifizieren können (vgl. die Zeilen 12 bis 15) oder mutmaßt der Agent, ein Zufallszug sei die beste Zugwahl

[499] Vgl. Flanagan 2011, S. 680.
[500] Web Worker waren zuerst Bestandteil der HTML5-Spezifikation, wurden im Laufe der Entwicklung am HTML5-Standard jedoch in eine eigene Spezifikation ausgelagert, die sich unter http://dev.w3.org/html5/workers (zuletzt erfolgreich aufgerufen am 12.08.2014) erläutert findet (vgl. Flanagan 2011, S. 680). Neben Web Workern, wie sie im Folgenden erläutert werden, definiert die Spezifikation gemeinsame (*shared*) Worker, die unterschiedlichen Skripten zugänglich sind (und somit z.B. die Möglichkeit der Kommunikation zwischen verschiedenen Browsertabs oder Browserfenstern ermöglichen), in der Praxis jedoch selten ihre Anwendung finden, wie Firtman anmerkt (vgl. Firtman 2013, S. 464f.).
[501] Vgl. Zakas 2010, S. 120.

(vgl. die Zeilen 18 bis 21 des Quellcodes), schlägt der Agent seinem assoziierten Meta Agenten die von ihm als sinnvoll erachtete Zugwahl vor.

```
1    self_Agent.createBrain = function() {
2
3        var agentWorker = new Worker("[…]/spoooky.Worker.js"),
4            player = Spoooky.AI.game.
                        getPlayerWithID(self_Agent.metaAgentID);
5
6       // Eventlistener für Nachrichten des Web Workers erstellen
7        agentWorker.addEventListener("message", function(event) {
8
9            switch(event.data.type){
10
11               // Der Web Worker konnte
                        einen sinnvollen Zug identifizieren
12               case "decision":
13                   // Schlage dem assoziierten Meta Agenten
                        den besten Zug vor
14                   player.coordinateAgentDecisions(event.data);
15                   break;
16
17               // Der Web Worker konnte keinen
                        besten Zug identifizieren
18               case "random":
19                   // Schlage dem assoziierten Meta Agenten
                        einen Zufallszug vor. Die Bitte um den
                        Zufallszug ist formuliert als Interna
                        des Parameters event.data).
20                   player.coordinateAgentDecisions(event.data);
21                   break;
22
23               // […]
24           }
25       });
26
27       return agentWorker;
28   };
```

Die Fähigkeiten des Agentengehirns sind in dem Skript *spoooky.Worker.js* beschrieben, in dem anhand der Funktion `importScripts` zuerst die Bibliotheken *underscore.js*, *spoooky.js* und *spoooky.AI.js* geladen werden, um Algorithmen bereitzustellen, die der Entscheidungsfindung essentiell sind:[502]

```
1    // Import necessary scripts to build virtual SpoookyJS games
2    importScripts("libs/underscore-min.js",
                        "spoooky.js", "spoooky.AI.js");
```

[502] Für den vollständigen Quelltext des Worker-Skripts vgl. die Datei *spoooky.Worker.js* im Ordner js/libs.

Die Kommunikation des Agenten mit seinem assoziierten Meta Agenten gestaltet sich über eigendefinierte Nachrichtenereignisse: Erhält der Agent die Nachricht "`message`" von seinem Meta Agenten, wie in Zeile sieben des vorangegangenen Quelltextes bestimmt, startet der Agent seine Arbeit und differenziert Arbeitsanweisungen wie "`monte carlo tree search with uct`" oder "`alpha beta negamax`", die von dem Meta Agenten gestellt werden:

```
3    self.addEventListener("message", function(e) {
4
5        var data = e.data, bestMove;
6
7        switch (data.command) {
8
9            // Driver for Monte Carlo Methods
10           case "monte carlo tree search with uct":
```

Innerhalb des Falles "`monte carlo tree search with uct`" der *switch case* Anweisung benachrichtigt der Agent seinen assoziierten Meta Agenten über die Aufnahme seiner Analysetätigkeit (Zeilen 11 bis 13 des folgenden Quelltextausschnittes), erstellt eine Arbeitskopie des Spieles in seinem aktuellen Zustand (Zeilen 15 und 16) und analysiert das – somit virtuell ablaufende – Spiel mit der Monte Carlo Spielbaumsuche und dem UCT-Algorithmus:

```
11               self.postMessage({
12                   type: "Starting Monte Carlo Tree Search
                             With UCT"
13               });
14
15               var gameTmp = new Spoooky.Game,
16                   gameModel = gameTmp.clone(data.gameModel),
17                   uctResults = Spoooky.AI.UCT(gameModel,
                             data.agentFocus, data.maxSteps,
                             data.maxTime, data.learn);
```

Nach Abschluss seiner Analysetätigkeit sendet der Agent eine Nachricht des Typs "`decision`"an seinen Meta Agenten und übermittelt Informationen über die Ergebnisse seiner Arbeit:

```
18               // Return the best moves found by monte carlo uct
19               self.postMessage({
20                   type : "decision",
21                   agentID : data.agentID,
22                   agentRole : data.agentRole,
23                   agentFocus : data.agentFocus,
24                   results : uctResults.results,
25                   QLearner:JSON.stringify(uctResults.QLearner)
```

```
26                  });
27
28                  // Terminate the worker
29                  self.close();
30                  break;
```

Der vorliegende Agent untersucht das an ihn weitergeleitete Entscheidungsproblem anhand der Funktion `UCT`, wie sie in der KI-Bibliothek *Spoooky.AI* definiert ist.[503] Implementiert ist mit der Funktion `UCT` die Monte Carlo Spielbaumsuche mit dem Selektionsalgorithmus *Upper Confidence Bounds applied to Trees*. Seine Erweiterung erfährt der vierschrittige Algorithmus (Selektionsphase, Expansionsphase, Simulationsphase und Rückpropagation der Ergebnisse) mit dem verstärkenden Q-Lernverfahren, über das der Meta Agent aus simulierten Erfahrungen zu lernen vermag.

6.4.1.3 Memorieren und agieren – Das Lernvermögen des Meta Agenten

Jeder Meta Agent verfügt in *SpoookyJS* über sein eigenes Lernmodul, dessen Aufbau und Eigenschaften durch den Q-Lernalgorithmus bestimmt sind.[504] Anhand seines Lernmodules vermag der Meta Agent, auf Spielsituationen zu reagieren, die ihm aus früheren Spielpartien bekannt sind. Um innerhalb kurzer Zeit eine große Anzahl von Spielerfahrungen zu sammeln, wurde das Q-Lernverfahren in der vorliegenden Arbeit nicht einzig auf die Spielpraxis des Meta Agenten beschränkt, sondern auf die Agenten des Ensembles erweitert: Jeder einzelne Agent des Ensembles verfügt über sein eigenes Lernmodul, das er in der Simulationsphase der Monte Carlo Spielbaumsuche mit simulierten Erfahrungsepisoden füllt. Hat der einzelne Agent seine Analyse des aktuellen Spielzustandes auf Grundlage der Monte Carlo Spielbaumsuche abgeschlossen, reicht er seine Erfahrungen weiter an den Meta Agenten, der sein Lernmodul anschließend durch die Erfahrungsepisoden der Agenten des Agentenensembles aktualisiert.

[503] Vgl. die JavaScript-Datei *spoooky.AI.js* im Ordner *js/libs*.

[504] Für die Erläuterung des Q-Lernalgorithmus‘ vgl. die Kapitel 4.3.2.5 und 4.3.2.6. Die Implementation des Q-Lernalgorithmus‘ folgt der Implementation, wie sie sich unter https://github.com/nrox/q-learning.js findet (zuletzt aufgerufen am xxx).

Damit der Meta Agent nicht jedes Spiel von neuem lernen muss, werden die Erfahrungen des Meta Agenten langfristig in der Datei *agentmemory_[ID des Meta Agenten].json* gespeichert, die sich im entsprechenden Spielverzeichnis befindet – und die – sofern vorhanden – zum Start der Spielpartie geladen wird.[505] Neben Angaben zur Beschaffenheit des Agentenensembles, wie sie im übernächsten Kapitel erläutert werden, finden sich in der Gedächtnisdatei des Meta Agenten die Signaturen von Spielzuständen und Belohnungen sowie Bestrafungen, die mit unterschiedlichen Zugmöglichkeiten verbunden sind. Die folgenden Quelltextausschnitte aus der Datei *agentmemory_1.json* veranschaulichen den Aufbau und den Inhalt der Gedächtnisdatei des Meta Agenten mit der ID 1 im Tic Tac Toe Spiel. Eingeleitet wird das Lernmodul mit Angaben zu den Parametern des Lernalgorithmus':

```
1    // [...]
2    "learnModule" : {
3        "alpha": 0,
4        "gamma": 0.8,
5        "rho": 0.2,
6        "nu": 0,
```

Jeder Spielzustand, der im Rahmen des Lernverfahrens generiert und aufgefunden wurde, wird in der JSON-Datei in Form einer Zeichenkette mitsamt der Zugmöglichkeiten gespeichert, die im aktuellen Zustand bestehen. Die Abbildung des Spielzustandes durch eine Zeichenkette leistet die Methode `createBoardSignature` der Klasse *Spoooky.GridWelt*, die jede Zelle der Spielwelt traversiert und den in der Spielsteinblaupause gesetzten Typidentifizierer (`typeID`) für ein besetztes und das Symbol 0 für ein nicht besetztes Spielfeld notiert – aus einem fingierten Tic Tac Toe Spielzustand, bei dem drei weiße Spielsteine in horizontaler Folge in der obersten Reihe des Spielbrettes liegen, generiert die Funktion `createBoardSignature` somit die Zustandssignatur `"WWW|000|000"`. Im Belohnungsteil der Gedächtnisdatei des Meta Agenten sind die besuchten Spielzustände in Form ihrer Signaturen sowie mögliche Züge und ihre Belohnungen nach der folgenden Syntax wiedergegeben:

[505] Ist keine Datei *agentmemory_[ID des Meta Agenten].json* im Spielverzeichnis auffindbar, wird ein neues Lernmodul für den Meta Agenten zum Start der Spielpartie erstellt.

```
7         "rewards": {
8             // [...]
9             "W00|0B0|00W": {
10                "b1": 0,
11                "a2": 0,
12                "c3": 0,
13                "c2": 0,
14                "b3": 0,
15                "a1": 0
16            },
17            "WB0|00W|0BW": {
18                "b2": 1
19            }
20            // [...]
21        },
```

Anhand seines Lernmodules ist es dem Agenten möglich, den Prozess der Entscheidungsfindung deutlich zu verkürzen: Bevor der Meta Agent sein Agentenensemble nach dem bestmöglichen Zug befragt, aktiviert er sein Spielgedächtnis. Befindet sich in seinem Gedächtnis bereits ein Eintrag, der auf eine sinnvolle Zugmöglichkeit verweist, führt der Meta Agent jene Zugmöglichkeit aus. Fehlt ein Gedächtniseintrag über die Spielwelt in ihrem aktuellen Zustand, beauftragt der Meta Agent sein Ensemble, um Erfahrungsepisoden zu generieren und den aktuellen Spielzustand zu analysieren.

6.4.1.4 Agentenfitness

Jeder einzelne Agent des Agentenensembles verfügt neben seinem Untersuchungsfokus über eine Fitness, die seinen Erfolg, gar sein Dasein in der Spielwelt bestimmt. Entlehnt der genetischen Programmierung bzw. evolutionären Algorithmen, ist mit der individuellen Agentenfitness die Fähigkeit eines Agenten bezeichnet, in einer bestimmten Umgebung zu überleben und sich reproduzieren zu können.[506] In *SpoookyJS* wird die Fitness der einzelnen Agenten von dem Meta Agenten beeinflusst: Wirkt ein Agent des Ensembles häufig an der Entscheidungsfindung mit, so wird er bestärkt und seine Fitness erhöht. Schließt sich ein Agent von der Entscheidungsfindung aus, weil er keine sinnvolle Zugmöglichkeit finden konnte – d.h. weil die an ihn gestellte Analysefrage ob seiner Fokussierungsweise nicht in einem bestimmten Zeitrahmen erschöpfend zu

[506] Vgl. Fogel 2006, S. 35.

betrachten, gar zu lösen ist –, so wird seine individuelle Fitness von dem übergeordneten Meta Agenten verringert.

Nachdem alle Agenten des Ensembles ihre Zugvorschläge dem Meta Agenten unterbreitet haben, betrachtet der Meta Agent die gesammelten Zugempfehlungen und aktualisiert anschließend die Fitnesswerte der Agenten nach oben vorgetragener Verfahrensweise. Ein Agent, dessen Fitness erschöpft, d.h. auf einen Wert unter null gesunken ist, wird von dem Meta Agenten aus dem Ensemble entfernt und durch einen Agenten ersetzt, der die Fokussierungsweise des erfolgreichsten Agenten erhält. Jener Agent gliedert sich anschließend in das Ensemble ein mit einem reellzahligen Fitnesswert von 1.0. Vollzogen wird die Aktualisierung der Agentenfitness durch die Memberfunktion `coordinateAgentDecisions` der Klasse *Spoooky.MetaAgent*, wie mit dem folgenden kommentierten Quelltextauszug veranschaulicht:

```
1    self_MetaAgent.coordinateAgentDecisions = function(results) {
2
3        // [...]
4
5        // Verarbeite die Vorschläge des Agentenensembles,
            wenn alle Agenten ihre Analysearbeit
            abgeschlossen haben.
6        if (self_MetaAgent.finishedAgents ===
                 self_MetaAgent.activeAgents) {
7
8            // [...]
9
10           // Verarbeite den besten Zug
11           if (bestMove === "Random") {
12
13               // Führe einen Zufallszug aus, wenn kein
                       bester Zug vom Ensemble
                       identifiziert werden konnte.
14               self_MetaAgent.getGame().executeRandomMove();
15           } else {
16
17               // Erhöhe die Fitness aller an der
                       Zugfindung beteiligten Agenten.
18               self_MetaAgent.
                       updateAgentsFitness(bestMove.agentID);
19
20               // Ersetze unfitte Agenten.
21               self_MetaAgent.replaceUnfitAgents();
22
23               // Führe den besten Zug aus.
24               self_MetaAgent.getGame().
                       executeArtificialMove(bestMove);
25           }
26       }
27   };
```

6.4.1.5 Agentenensembles und ihre Tradierung

Jeder Meta Agent verfügt über die Möglichkeit, seine Spielerfahrungen am Ende der Spielpartie zu speichern und zu aktualisieren. Spielerfahrungen, die sich aus dem besten Agentenensemble sowie den Erfahrungsepisoden des Q-Lernalgorithmus‘ konstituieren. Der folgende Auszug aus der spiel- und metaagenteneigenen Skriptdatei *agentmemory_1.json* gibt beispielhaft das als beste identifizierte Ensemble von vier Agenten wieder, das der Meta Agent mit der ID 1 nach wenigen Spielpartien des Tic Tac Toe Spieles identifizieren konnte:

```
1    {
2        "game" : "Tic Tac Toe",
3        "metaAgentID" : 1,
4        "bestAgentEnsemble" : [
5        {
6            "ID": 1,
7            "role": "ANALYZE POSSIBLE MOVES",
8            "focus": "ALL MOVES",
9            "fitness": 2.5,
10           "thinkingTime": 3000,
11           "maximumSteps": 15000
12       },
13       {
14           "ID": 2,
15           "role": "ANALYZE POSSIBLE MOVES",
16           "focus": "MOVES NEAR OPPONENT OR OWN FIELDS",
17           "fitness": 2.5,
18           "thinkingTime": 3000,
19           "maximumSteps": 15000
20       },
21       {
22           "ID": 3,
23           "role": "ANALYZE POSSIBLE MOVES",
24           "focus": "MOVES NEAR OPPONENT FIELDS",
25           "fitness": 2.5,
26           "thinkingTime": 3000,
27           "maximumSteps": 15000
28       },
29       {
30           "ID": 4,
31           "role": "ANALYZE POSSIBLE MOVES",
32           "focus": "ALL MOVES",
33           "fitness": 2.5,
34           "thinkingTime": 3000,
35           "maximumSteps": 15000
36       }
37   ],
38       "learnModule" : { // [...]
```

Wie mit dem oben wiedergegebenen Quelltext veranschaulicht, werden individuelle Agenteneigenschaften wie Fokussierungsweise und die Agentenfitness am Ende der Spielpartie gespeichert. Darüber hinaus wird die maximale Denkzeit (*thinkingTime*) in Millisekunden des Agenten sowie die maximale Anzahl der Schritte der Monte Carlo Spielbaumsuche gespeichert. Um das Lernverhalten des Meta Agenten zu realisieren, verfügt die Klasse *Spoooky.MetaAgent* über das Attribut `learningEnabled`, das bestimmt, ob der Meta Agent die simulierten Erfahrungsepisoden seiner assoziierten Agenten berücksichtigt und das die Speicherung der Erfahrungen des Meta Agenten am Ende der Spielpartie umsetzt. Aktiviert wird das Lernmodul einer Instanz der Klasse *Spoooky.MetaAgent* durch Aufruf der Memberfunktion `enableLearning` im Spielskript *game.js*.

6.4.2 Evaluation und Diskussion der spielübergreifenden KI

Der in den vorangegangenen Kapiteln vorgetragene Ansatz zur Implementation spielübergreifender künstlicher Intelligenz beruht auf der hierarchischen Differenzierung entscheidungsfindender Momente in Meta Agent, Agentenensemble und den individuierten Agenten des Agentenkollektivs: Um eine Zugentscheidung im aktuellen Zustand der Spielwelt herbeizuführen, aktiviert der Meta Agent seine memorierten Spielerinnerungen oder beauftragt sein assoziiertes Agentenensemble mit der Suche nach erfolgversprechenden Spielzügen. Jeder Agent des Ensembles analysiert den aktuellen Zustand der Spielwelt anhand der Monte Carlo Spielbaumsuche auf Grundlage des UCT-Selektionsalgorithmus'. Die im Rahmen des Monte Carlo Verfahrens simulierten Spielverläufe lassen sich vom Meta Agenten dazu verwenden, um sein Spielgedächtnis zu aktualisieren. Das evolutionäre Moment des JavaScript-basierten Multiagentensystems manifestiert sich in der Tradierung und Entwicklung des Agentenensembles.

In seiner Leistungsfähigkeit skaliert die spielübergreifende künstliche Intelligenz positiv proportional mit der zur Verfügung stehenden physischen Arbeitsspeicherkapazität (*RAM*), der Rechenleistung der physischen Prozessoren (*CPU*) und der eingeräumten Laufzeitdauer der Entscheidungsfindungsprozesse. Exponentiell negativ skaliert der implementierte Ansatz mit erhöhter Spielbaumkomplexität der Spiele – und auch die Wahl des Webbrowsers offenbart sich ob der differierenden Leistungsfähigkeit unterschiedlicher JavaScript-Engines als relevantes Faktum. Um die Performanz der Analysetätigkeit, die Ressourcennutzung und Entscheidungskraft des Ansatzes zu evaluieren, sei mit den nachfolgenden Kapiteln gesondert auf die verschiedenen, mit dem JavaScript-Framework implementierten, Spiele eingegangen und Besonderheiten der Spiele in der Umsetzung spielübergreifender artifizieller Gegner vorgetragen. Das zur Evaluation verwendete Testsystem besteht aus den folgenden Hard- und Softwarebestandteilen: Intel Core i5-2500 CPU @ 3.30 GHz, 4 logische (physische) Prozessoren; 8 GB RAM; Microsoft Windows 8.1 Pro (64-Bit). Durchgeführt

wurden die Spielevaluationen mit der compilierten[507] und minimierten Version des Frameworks im Webbrowser *Mozilla Firefox* in der Version 33.0 (32 Bit, 14.10.2014).[508] Aufgrund des spielübergreifenden Charakters der Grundversion des Frameworks ist allen durchgeführten Evaluationen gemein, dass sie kein Expertenwissen verwenden, um den Verlauf der Monte Carlo Simulationen positiv zu beeinflussen – die hiermit verbundenen Probleme und Herausforderungen finden sich in den Evaluationen diskutiert.

[507] Zur Compilierung, Minimierung und Optimierung des JavaScript-Codes wurde der *Google Closure Compiler* verwendet (vgl. https://github.com/google/closure-compiler, zuletzt aufgerufen am 08.10.2014).

[508] Im Browservergleich zeichnet sich Mozillas *Firefox* in der Version 33.0 (32 Bit, veröffentlicht am 14.10.2014) durch die höchste Anzahl von durchgeführten Simulationen und simulierten Spielrunden in determiniertem Zeitrahmen aus – und dient infolgedessen als Werkzeug der Evaluation. Einzig die Rendering-Engine des Browsers trübt das positive Bild des Browsers aufgrund ihrer schlechten Performanz im Zeichnen der Monte Carlo Spielbäume, die mit der JavaScript-Bibliothek *D3.js* (http://d3js.org, zuletzt aufgerufen am 11.10.2014) als skalierbare Vektorgrafiken (SVG) umgesetzt sind.

6.4.2.1 Tic Tac Toe

Aufgrund seiner geringen Spielbaumkomplexität stellt das Tic Tac Toe Spiel für die artifizielle Entscheidungsfindung des Meta Agenten kein Problem dar. Bereits bei einer Entscheidungszeit von fünf Sekunden und einem Ensemble, das mit nur einem Agenten arbeitet, werden Bedrohungssituationen und terminale Spielzustände auf Grundlage der Monte Carlo Spielbaumsuche zuverlässig erkannt und in der Entscheidungsfindung berücksichtigt. Wenngleich bereits ein isolierter Agent in der Tic Tac Toe Partie gute Arbeit leistet, sei die Verwendung von zwei oder mehr Agenten empfohlen, um die zur Verfügung stehenden Ressourcen – physische Prozessoren und Arbeitsspeicher – optimal zu nutzen. Zwar vermag es der Meta Agent, aufgrund seines Lernmodules zahlreiche Spielzustände und ihre Belohnungswerte zu memorieren, aufgrund seiner geringen Komplexität wird im Tic Tac Toe Spiel jedoch von der Verwendung des Q-Lernmodules der Agenten abgesehen.

Zur Veranschaulichung der unterschiedlichen Analysetätigkeiten der Agenten dient das folgende Agentenensemble des Meta Agenten, der die Spielpartie beginnt:

```
1    {
2        "game" : "Tic Tac Toe",
3        "metaAgentID" : 1,
4        "bestAgentEnsemble" : [
5        {
6            "ID": 1,
7            "role": "ANALYZE POSSIBLE MOVES",
8            "focus": "ALL MOVES",
9            "fitness": 5,
10           "uctConstant" : 1.0,
11           "thinkingTime": 5000,
12           "maximumSteps": 5000
13       },
14       {
15           "ID": 2,
16           "role": "ANALYZE POSSIBLE MOVES",
17           "focus": "ALL MOVES",
18           "fitness": 5,
19           "uctConstant" : 0.5,
20           "thinkingTime": 5000,
21           "maximumSteps": 5000
22       },
23       {
24           "ID": 3,
25           "role": "ANALYZE POSSIBLE MOVES",
26           "focus": "ALL MOVES",
27           "fitness": 5,
```

```
28            "uctConstant" : 0,
29            "thinkingTime": 5000,
30            "maximumSteps": 5000
31        }
32    ],
33        "learnModule" : {"alpha":0,"gamma":0.8,"rho":0.2,"nu":0,
                                "rewards":{},"gameStates":{}}
34    }
```

In der Entscheidungssituation des Meta Agenten, dem Initialzustand des Spieles, simulierten die drei Agenten innerhalb des determinierten Zeitrahmens von fünf Sekunden auf dem Testsystem 6536 Spiele mit insgesamt 21914 Simulationsschritten (rund 1307 simulierte Spiele pro Sekunde mit rund drei Schritten/Simulation). Anhand der laufzeitgenerierten Spielbaumdarstellungen veranschaulicht sich die unterschiedliche Arbeitsweise der Agenten unter Verwendung differierender UCT-Konstantenwerte. So arbeitet der erste Agent mit einer UCT-Konstante von 1.0 und generiert einen Spielbaum, der einschließlich des Wurzelknotens eine Tiefe von sechs erreicht und terminale Spielzustände erreicht, die durch grün eingefärbte Knoten visualisiert sind:

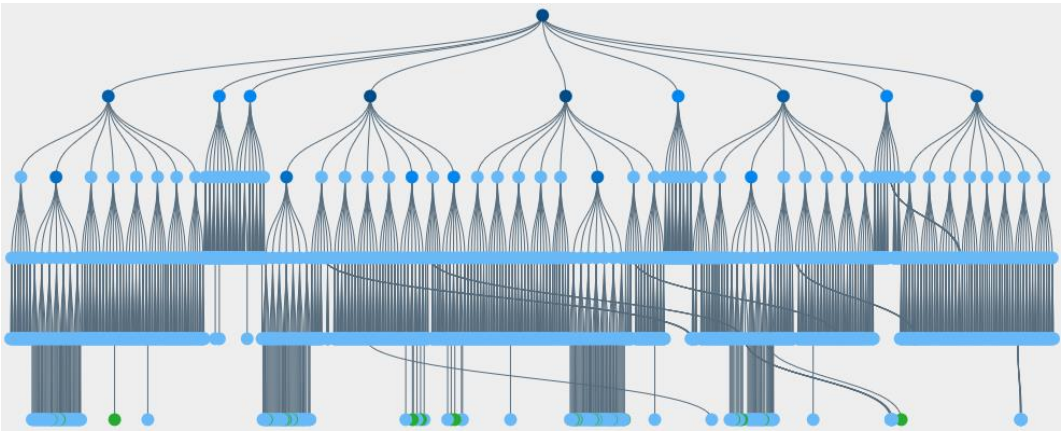


Abbildung 73: Dynamisch generierter Spielbaum des Agenten mit einem UCT-Konstantenwert von 1.0 zur Untersuchung der Zugmöglichkeiten im Initialzustand des Tic Tac Toe Spieles.

Lässt sich die Arbeitsweise des ersten Agenten mit dem Konstantenwert von 1.0 als Breitensuche signifizieren, verfährt der zweite Agent mit dem UCT-Wert von 0.5 auswählend: Erfolgversprechende Zugmöglichkeiten werden häufiger besucht, so dass der Spielbaum eine Untersuchungstiefe von zehn Ebenen erreicht, wie in nachfolgender Darstellung veranschaulicht.

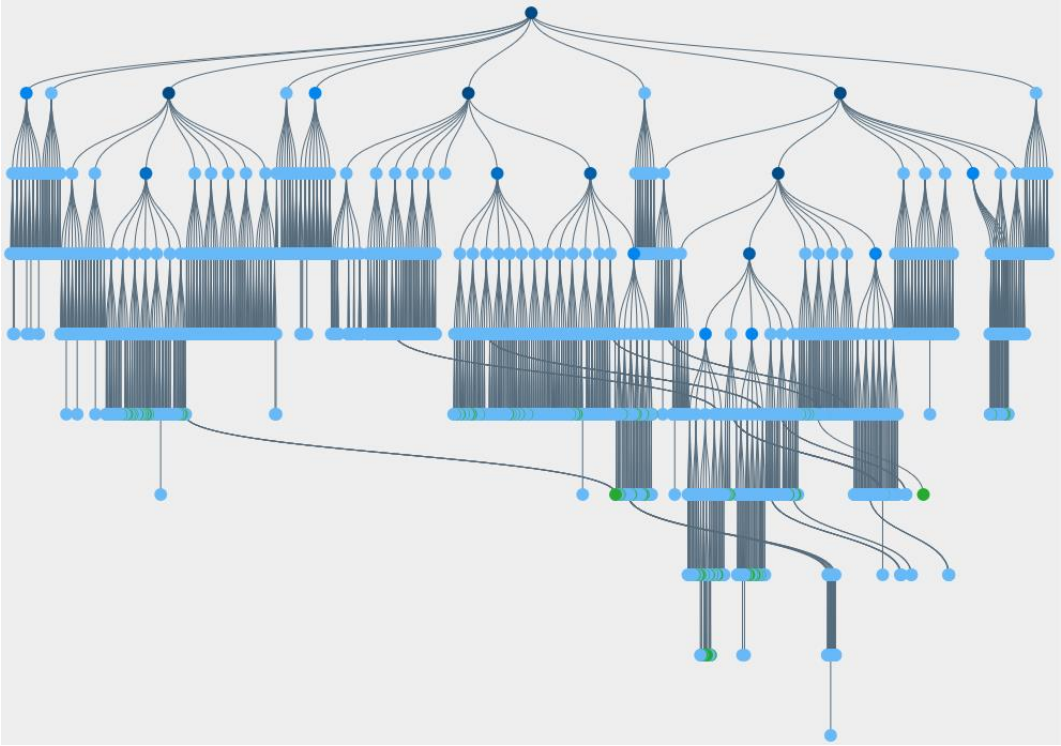


Abbildung 74: Dynamisch generierter Spielbaum des Agenten mit einem UCT-Konstantenwert von 0.5 zur Untersuchung der Zugmöglichkeiten im Initialzustand des Tic Tac Toe Spieles.

Besonders selektiv verhält sich der dritte Agent, der mit einem Konstantenwert von 0.0 arbeitet: Scheinbar wenig gute Zugmöglichkeiten werden sehr kurz betrachtet, um die Analysetätigkeit auf erfolgversprechende Spielzüge zu konzentrieren. Im durchgeführten fünfsekündigen Evaluationslauf vermag es der Agent, eine höhere Anzahl von Terminalzuständen der Spielpartie zu erreichen als die anderen beiden Agenten:

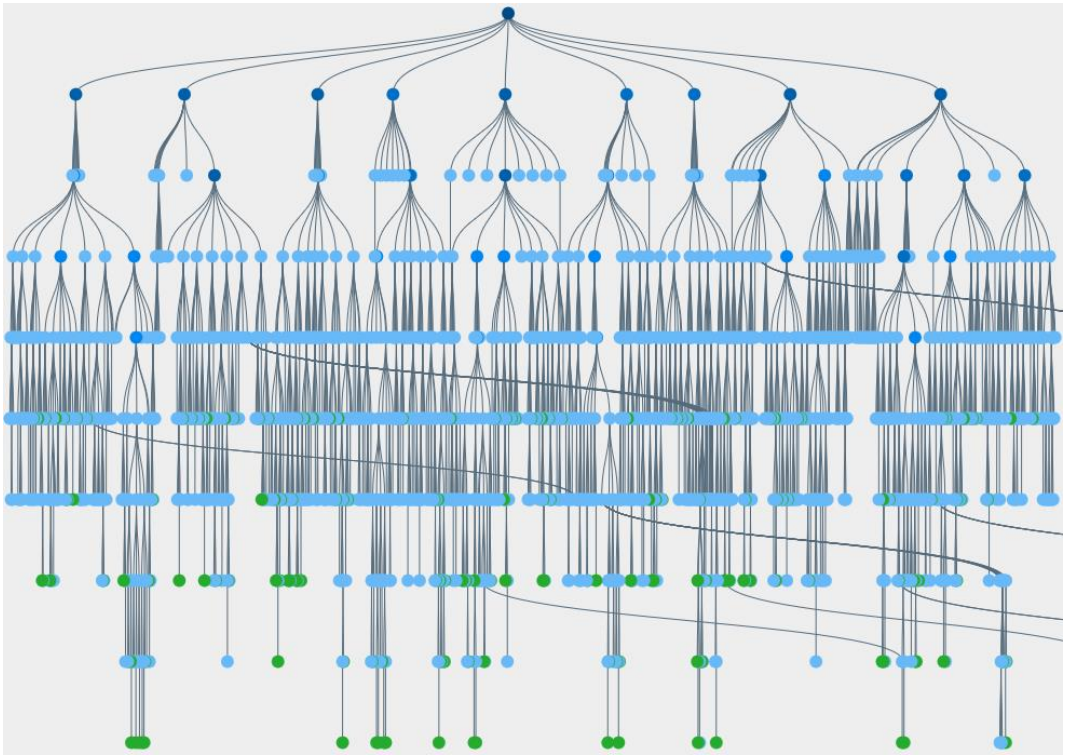


Abbildung 75: Dynamisch generierter Spielbaum des Agenten mit einem UCT-Konstantenwert von 0.0 zur Untersuchung der Zugmöglichkeiten im Initialzustand des Tic Tac Toe Spieles (verkürzte Darstellung).

Aufgrund der geringen Komplexität des Tic Tac Toe Spieles ist keine Feinjustierung der UCT-Konstantenwerte vonnöten – beginnt der artifizielle Spieler die Spielpartie, lässt sich – ausreichende Rechenleistung und Laufzeit vorausgesetzt – die Partie gegen den Meta Agenten nicht gewinnen.

6.4.2.2 Schach und Dame (Checkers)

Die rundenbasierten Zugspiele Schach und Dame stellen Meta Agenten und Agentenensemble vor eine mehrdimensionale Herausforderung: Zum einen gestaltet sich die Spielbaumkomplexität der beiden Spiele außerordentlich größer als im Tic Tac Toe Spiel[509] – die einzelnen Agenten haben sich folglich mit ihrem Arbeitswerkzeug der Monte Carlo Spielbaumsuche in der Vielzahl aller möglichen Spielzustände mit einer Unzahl von Zugmöglichkeiten zu beschäftigen. Zum anderen wirkt sich in den Rollouts des Monte Carlo Verfahrens besonders ein Charakteristikum der Spiele stark auf die Performanz des Verfahrens aus: Weil sowohl im Schach- als auch im Damespiel Spielfiguren agieren, die sich – mehr oder weniger beliebig – in der Spielwelt vor- und zurückbewegen können, gestaltet sich die Anzahl der Spielrunden in den simulierten Spielen ohne eingebrachte Expertise zumeist äußerst hoch – und führen die Rollouts somit selten zu Terminalzuständen der simulierten Spielpartien.

Grundlage der im Folgenden durchgeführten Evaluation der Leistungsfähigkeit des realisierten Ansatzes im Schach- und Damespiel dient das folgende Agentenensemble, das im Rahmen einer Verfahrenslaufzeit von 30 Sekunden den Suchraum der Zugmöglichkeiten im Initialzustand der Spiele halbiert (Agentenfoki `"FIRST HALF OF POSSIBLE MOVES"` und `"SECOND HALF OF POSSIBLE MOVES"`) und mit exemplarischen UCT-Konstantenwerten von 0.0 und 1.0 bei 50000 maximalen Runden in den simulierten Spielen arbeitet:[510]

```
1    "bestAgentEnsemble" : [
2        {
3            "ID": 1,
4            "role": "ANALYZE POSSIBLE MOVES",
5            "focus": "FIRST HALF OF POSSIBLE MOVES",
6            "fitness": 5,
7            "uctConstant" : 0.0,
8            "thinkingTime": 30000,
9            "maximumSteps": 50000
10       },
11       {
12           "ID": 2,
13           "role": "ANALYZE POSSIBLE MOVES",
14           "focus": "FIRST HALF OF POSSIBLE MOVES",
15           "fitness": 5,
```

[509] Für das Damespiel merkt Allis eine Spielbaumkomplexität von 10^{31} an (vgl. Allis 1994, S. 168) und für das Schachspiel eine Spielbaumkomplexität von 10^{123} (vgl. Allis 1994, S. 171).
[510] Vgl. die Dateien *agentmemory_1.json* in den Ordnern *games/chess* und *games/checkers*.

```
16          "uctConstant" : 1.0,
17          "thinkingTime": 30000,
18          "maximumSteps": 50000
19      },
20      {
21          "ID": 3,
22          "role": "ANALYZE POSSIBLE MOVES",
23          "focus": "SECOND HALF OF POSSIBLE MOVES",
24          "fitness": 5,
25          "uctConstant" : 0.0,
26          "thinkingTime": 30000,
27          "maximumSteps": 50000
28      },
29      {
30          "ID": 4,
31          "role": "ANALYZE POSSIBLE MOVES",
32          "focus": "SECOND HALF OF POSSIBLE MOVES",
33          "fitness": 5,
34          "uctConstant" : 1.0,
35          "thinkingTime": 30000,
36          "maximumSteps": 50000
37      }
38  ],
```

Wie sich im Rahmen der Evaluation offenbart, stellen sich die zur Verfügung stehenden Ressourcen als zu knapp bemessen und Implementationsinterna wie die zeitintensive Verarbeitung von Regelatomen, Spiel- und Zugregeln als Hindernis für die Bereitstellung artifizieller Gegner im Schachspiel dar. So ließen sich auf dem Testsystem im gegebenen Zeitrahmen einer halben Minute einzig 27 Spiele mit insgesamt 8206 Schritten simulieren – weniger als ein simuliertes Spiel/Sekunde bei einer Anzahl von rund 304 Simulationsschritten pro Spiel.

Weil sich selbst bei hohen Laufzeiten der Verfahren zur Entscheidungsfindung keine oder nur wenige terminale Zustände des Spieles auffinden lassen, wurde im Schachspiel von der Verwendung des Lernmodules abgesehen. In den Erinnerungsfundus des Meta Agenten ließe sich zwar menschliche Expertise um sinnvolle Züge in verschiedenen Zuständen des Spieles einbinden – ob der großen Zustands- und Spielbaumkomplexität und der mangelnden Analysefähigkeit der Agenten des Ensembles ein nur wenig erfolgreiches Unterfangen. Um die Leistungsfähigkeit der artifiziellen Gegner im Schachspiel zu erhöhen – die Analysetätigkeit gar zu ermöglichen –, muss die Performanz der frameworkinternen Regelverarbeitung gesteigert und die Einbeziehung menschlicher Expertise in den Rollouts ermöglicht werden. Module zur Nullzug-

oder Ruhesuche, zur Berücksichtigung von Killerzügen oder zur Vorsortierung von Zugmöglichkeiten böten sinnvolle ergänzende Erweiterungen des Frameworks.

Offenbart sich das Schachspiel in seiner umgesetzten Form als zu komplex für die expertisefreie Entscheidungsfindung im Agententeam, so lassen sich im Damespiel befriedigende Ergebnisse erzielen.[511] Zwar profitierten die Spielsimulationen auch im Damespiel stark von menschlicher Kenntnis um Charakteristika des Spieles und Wissen über sinnvolle Spielverläufe – wie die nachfolgenden Screenshots der durchgeführten Verfahren veranschaulichen, vermögen die einzelnen Agenten des Ensembles jedoch, Spielbäume zu generieren, die der Entscheidungsfindung dienlich sind und sich im Initialzustand des Spieles über eine Tiefe von sechs (Agent mit Konstantenwert 1.0) oder zehn (Agent mit Konstantenwert 0.0) Ebenen erstrecken:

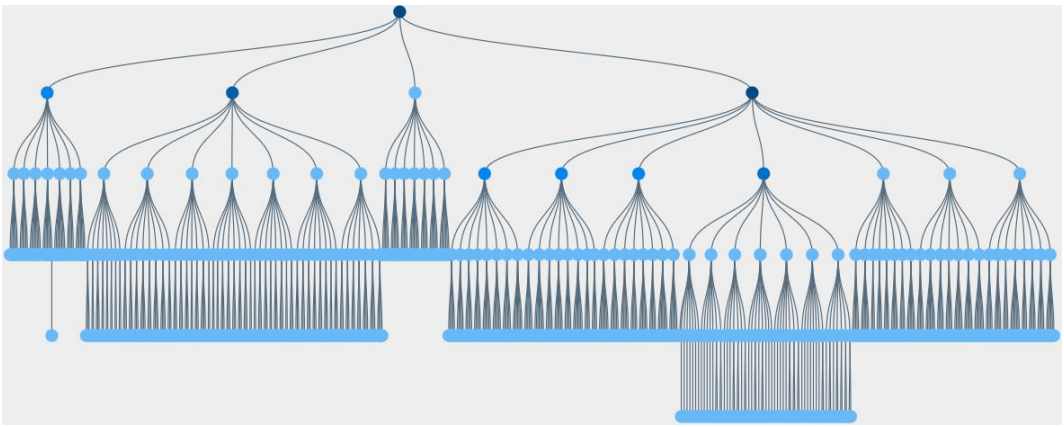


Abbildung 76: Laufzeitgenerierter Spielbaum des Agenten mit dem Analysefokus "FIRST HALF OF POSSIBLE MOVES" und einem UCT-Konstantenwert von 0.0 zur Untersuchung der Zugmöglichkeiten im Initialzustand des Dame-Spieles.

[511] Die Spielregeln und ihre Implementation im Framework *SpoookyJS* sind in Anhang 13.3.1 beschrieben.

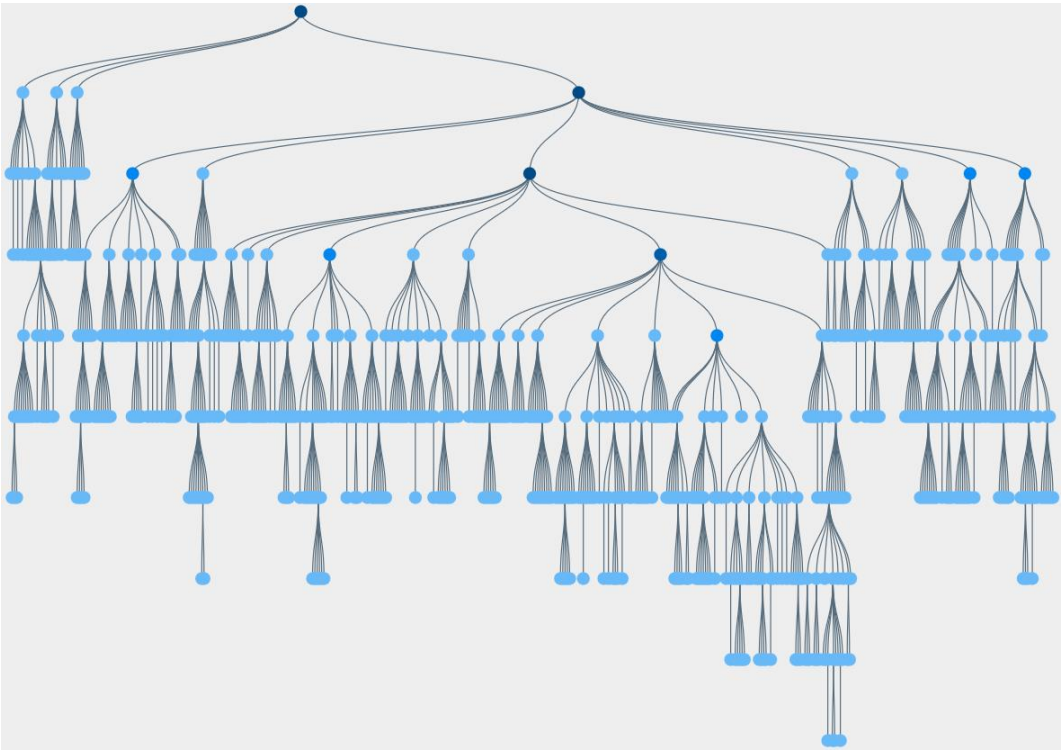


Abbildung 77: Laufzeitgenerierter Spielbaum des Agenten mit dem Analysefokus "FIRST HALF OF POSSIBLE MOVES" und einem UCT-Konstantenwert von 1.0 zur Untersuchung der Zugmöglichkeiten im Initialzustand des Dame-Spieles.

Als besonders wesentliches und qualitätsbestimmendes Moment der multiagentenbasierten Entscheidungsfindung stellt sich sowohl im Schach- und Damespiel als auch den weiteren implementierten Spielen der Verlauf und die Laufzeit der Rollouts dar: Werden die simulierten Spielpartien früher – d.h. mit einer geringeren Anzahl simulierter Spielzüge – zu einem Terminalzustand gebracht, lässt sich im gegebenen Zeitrahmen eine größere Anzahl von Spielen simulieren – und generiert sich folglich ein tiefer verzweigter Spielbaum im Rahmen der Monte Carlo Spielbaumsuche. Wie zuvor dargelegt, ist die Einbindung externen Wissens der Entscheidungsfindung im Schach- und Damespiel essentiell, um die Spielsimulationen früher zu ihren terminalen Zuständen auszuspielen und ein Herumirren der Spielfiguren in den Simulationen durch zielgerichtetes Agieren zu verbessern.

6.4.2.3 Gomoku

Bei dem rundenbasierten Gomoku-Spiel legen die Spielerinnen und Spieler ihre Spielsymbole auf dem Spielfeld ab, um mit einer horizontalen, vertikalen oder diagonalen Folge von fünf Spielsymbolen das Spiel für sich zu entscheiden. Eine besondere Herausforderung des Gomoku-Spieles manifestiert sich in Bedrohungssituationen, die einen Sieg in wenigen Zügen ermöglichen. Anhand des folgenden Spielzustandes sei ein einfaches Beispiel einer Bedrohungssituation veranschaulicht und erläutert:[512]

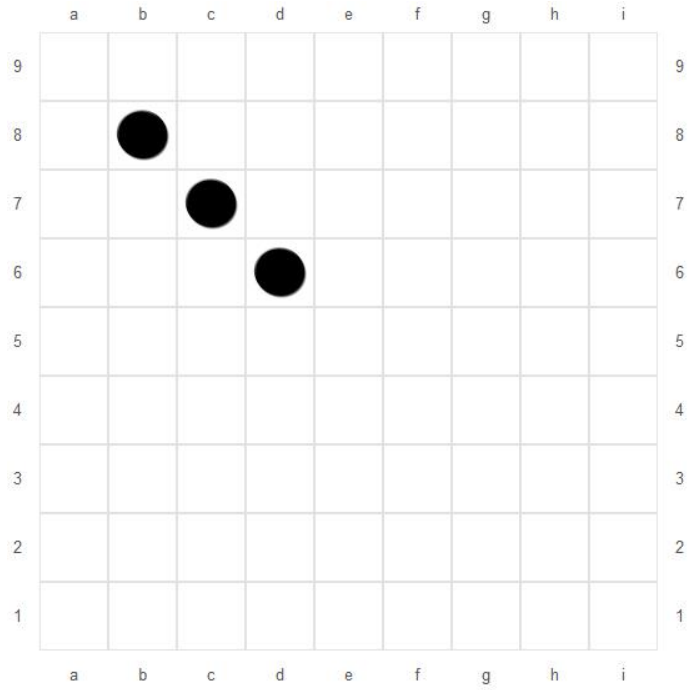


Abbildung 78: Bedrohungssituation im Gomoku-Spiel.

Weiß ist am Zug und muss die Spielerin mit den schwarzen Spielsteinen daran hindern, ihre Folge von drei diagonalen Steinen auf fünf zu erweitern. Aufgrund der Bedrohungssituation reduzieren sich die 78 Zugmöglichkeiten von Weiß auf zwei Zugmöglichkeiten: Die Ablage eines weißen Spielsteines auf das Feld a9 oder das Feld e5, um den gegnerischen Sieg zu verhindern. Jeder Zug auf ein anderes Feld führt zum Sieg des Gegenspielers, wenn die Spielerin mit den schwarzen Spielsteinen das Feld e5 belegt. Bedrohungssituationen wie die oben wiedergegebene in komplexen Spielzuständen zu erkennen, bedarf der

[512] Einen Überblick über verschiedene Arten von Bedrohungen im Gomoku-Spiel bieten Allis et al. unter der Überschrift *Expert knowledge – Definitions and threat sequences* (Allis et al. 1993, S. 2f.).

Erweiterung des Frameworks um unterschiedliche Intentionen des Meta Agenten: Zum einen die Intention, offensiv zu verfahren, zum anderen das Bestreben, Bedrohungssituationen zu erkennen und den Gegner am Sieg zu hindern.

Der grundsätzlichen Verfahrensweise der Agenten lässt sich ein Moment menschlicher Expertise beigeben, indem der individuelle Agentenfokus auf die Betrachtung von Zugmöglichkeiten neben gegnerische Spielsteine (`"MOVES NEAR OPPONENT FIELDS"`) oder Zugmöglichkeiten neben gegnerische oder eigene Spielsteine (`"MOVES NEAR OPPONENT OR OWN FIELDS"`) gerichtet wird. In der Evaluation offenbart sich der Agentenfokus als einziges Analysekriterium jedoch als mäßig erfolgreich. Zwar vermochte das aus sechs Agenten bestehende Ensemble mit dem Betrachtungsfokus auf Zugmöglichkeiten neben gegnerische Spielsteine innerhalb von 30 Sekunden knapp 4000 Spiele zu simulieren, wie der folgende Screenshot des Monte Carlo Spielbaumes eines Agenten mit UCT-Konstante 0.0 veranschaulicht, ist der Agent jedoch überwältigt von den zahlreichen Zugmöglichkeiten in jedem Zustand der Spielwelt – und benötigt ein geringes Maß an Expertenwissen, das den zufallsbasierten Charakter der Simulationen in einen angeleiteten Verlauf transformiert.

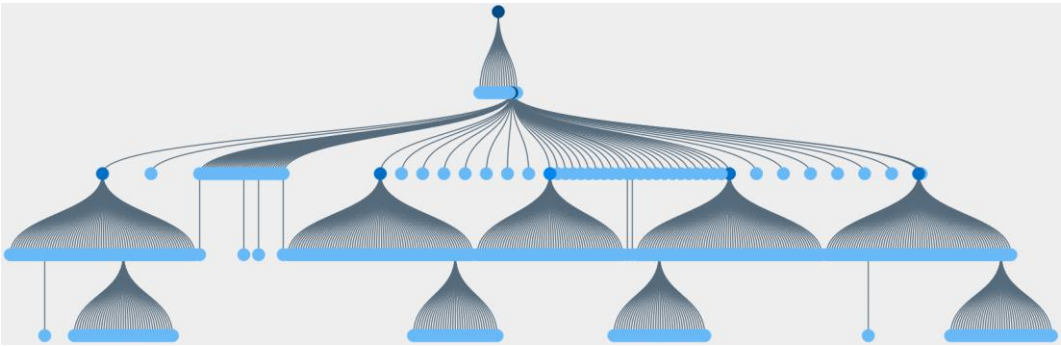


Abbildung 79: Laufzeitgenerierter Spielbaum des Agenten mit dem Analysefokus "MOVES NEAR OPPONENT FIELDS" und einem UCT-Konstantenwert von 0.0 zur Untersuchung einer Bedrohungssituation im Gomoku-Spiel.

Um eine entscheidungskräftige Verfahrensweise der Agenten zu bieten und zu gewährleisten, sieht das Framework in den Erinnerungsdateien der Meta Agenten die Ergänzung offensiver und defensiver Aktionsmöglichkeiten in Form von Expertenwissen vor – Expertenwissen wie die Intention, die Gegenspielerin daran zu hindern, vier oder fünf Steine in Folge auf dem Spielbrett abzulegen sowie das Bestreben, die Anzahl eigener Spielsteine in horizontaler, vertikaler oder

diagonaler Folge zu maximieren. Eine sinnvolle Weiterentwicklung des Frameworks stellt die Ergänzung des Rahmenwerkes um das Modul eines künstlichen neuronalen Netzes dar, das auszahlungsmaximierende Spielmuster und Bedrohungssituationen zu lernen vermag und dem Meta Agenten und seinem Ensemble die Entscheidungsfindung in *k-in-einer-Reihe* Spielen auf einem Spielbrett von $m \times n$ Feldern erleichtert.

6.4.2.4 Spiel der Amazonen (Amazons)

Das rundenbasierte *Spiel der Amazonen*, im Folgenden als *Amazons* abgekürzt, kombiniert die Herausforderungen von Zug- und Legespielen und wird von zwei Spielern auf einem 10×10 Felder großen Spielbrett gespielt. Ausgehend von dem im Folgenden dargestellten initialen Spielzustand beginnt die Spielerin Weiß das Spiel, indem Sie eine ihrer Spielfiguren – im Spiel als *Amazone* bezeichnet – analog der Dame im Schachspiel auf ein leeres benachbartes Feld auf horizontaler, vertikaler oder diagonaler Achse in der Spielwelt bewegt.

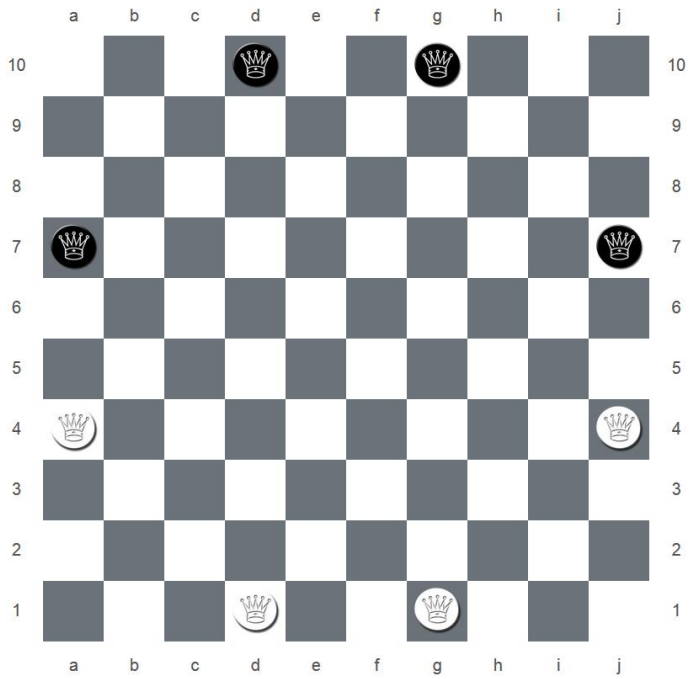


Abbildung 80: Startzustand des *Amazons*-Spiels.

Auf die Bewegung der Spielfigur folgt ein weiterer Zug: Ausgehend vom Zielfeld der ausgeführten Bewegung verschießt die Amazone einen Pfeil auf ein leeres Feld, das von der Amazone erreichbar wäre. Sowohl in der Zugbewegung als auch dem Verschießen des Pfeiles darf kein Spielfeld übersprungen werden, das von einem gegnerischen oder eigenen Pfeil oder einer gegnerischen oder eigenen Amazone besetzt ist. Das Spiel gewinnt, wer die gegnerischen Amazonen zugunfähig setzt.

Umgesetzt sind die Zugmöglichkeiten der weißen und schwarzen Dame und das Abschießen eines Pfeiles nach ausgeführter Bewegung durch den Wechsel des

Spielzustandes und des Spielmodus von "MOVING" auf "PLACING", wie der folgende Ausschnitt aus der Spielfigurenblaupause veranschaulicht:[513]

```
1    white_queen: {
2        entityType : "White Queen",
3        representation : { type : "image",
                 texture : "assets/white_queen.png" },
4        // ...
5        moves : [{
6            name : "north",
7            type : "Default",
8            direction : "north",
9            frequency : "ANY",
10           conditions : [{ condition : "Is Empty",
                 state : true }],
11           postMove : [{
12               jobName: "Enable Second Move",
13               jobFunction: "Prevent Player Change"
14           }, {
15               jobName: "Enable Firing Of One Arrow",
16               jobFunction: "Change Game Mode",
17               jobArguments: { mode: "PLACING" }
18           }]
19       },
20       // ...
21   },
22   white_arrow: {
23       entityType : "Arrow",
24       typeID : "AX",
25       associatedWithMetaAgent : null,
26       representation : { type : "image",
                 texture : "assets/white_block.png" },
27       mode : "PLACE",
28       placeTo : "REACHABLE BY RECENTLY MOVED ENTITY"
29   }
30   // ...
```

Mit seiner Kombination aus Zug- und Legespiel stellt sich das 1988 von Walter Zamkauskas vorgestellte Amazons-Spiel[514] als außerordentliche Herausforderung für die Implementation schlau agierender artifizieller Gegner dar. Wie Hearn ausführt, eröffnet sich im Amazons-Spiel eine größere Anzahl an Zugmöglichkeiten als im Go-Spiel, so dass klassische Suchverfahren an der Komplexität des Spieles scheitern. Benötigt würde dem Autor zufolge die umfangreiche Einbettung menschlicher Expertise:

> „Amazons has a very large number of moves available from a typical position, even more than in Go. This makes straightforward search algorithms impractical

[513] Vgl. den entsprechenden *Amazons*-Part in der Blaupausendatei *spoooky.Blueprints.js*.
[514] Vgl. Hearn 2005, S. 1.

for computer play. As a result, computer programs need to incorporate more high-level knowledge ofAmazons strategy.“

Zwar reduziert sich der Entscheidungsraum stetig mit jedem ausgeführten Zug und jedem abgelegten Pfeil, wie die nachfolgend wiedergegebenen Statistiken zweier gegeneinander agierender Meta Agenten anhand erhöhter Anzahl von Rollouts und Simulationsschritten dokumentieren, sinnvolle Entscheidungen im Amazons-Spiel zu treffen erfordert jedoch die Erweiterung des spielübergreifenden Frameworks um Expertenwissen. Aufgrund der außerordentlichen Komplexität des Spieles fungiert *SpoookyJS* somit als modulare Testumgebung, die des Experimentierens intendiert und als Grundlage von Erweiterungen fungiert.

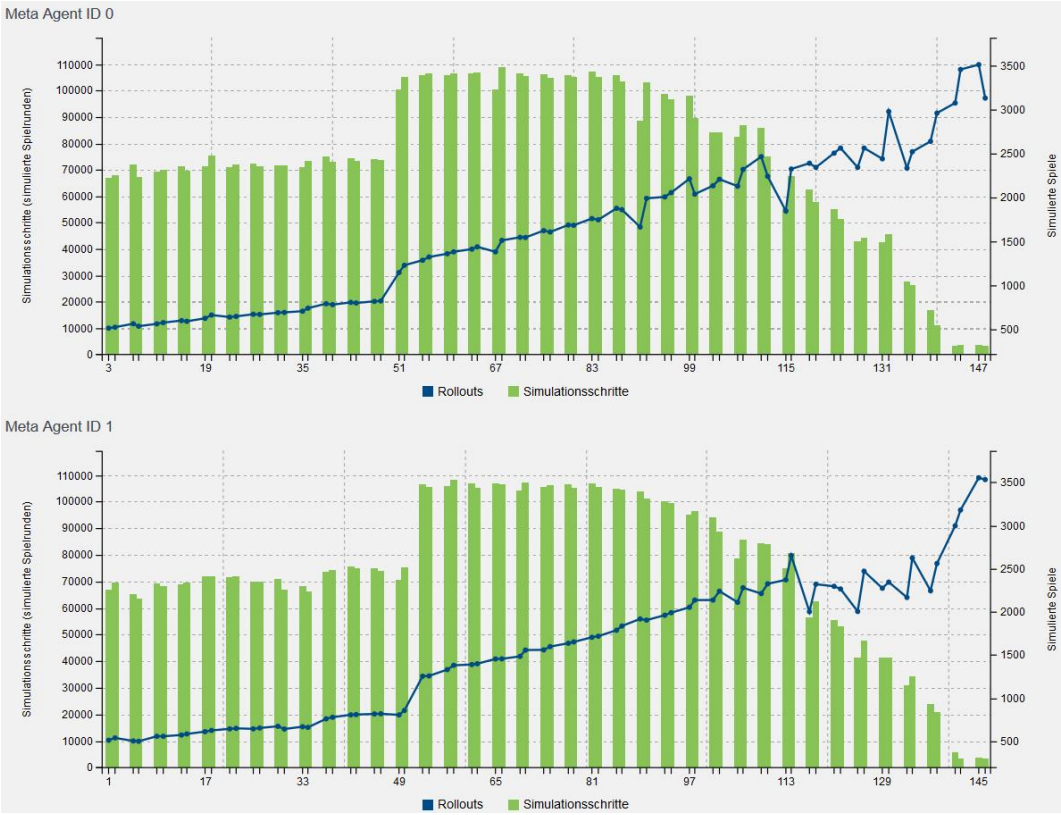


Abbildung 81: Statistiken der Entscheidungsfindung zweier konkurrierender Meta Agenten im Amazons-Spiel.

6.4.2.5 Backgammon

Das Backgammonspiel zählt zu den ältesten Brettspielen, die sich der Mensch im Laufe seines schöpferischen und kreativen Werdens ausgedacht hat.[515] Besonders interessant für die Beschäftigung im Kontext der Implementation künstlicher Intelligenz in Brettspielen ist das Zufallsmoment, das sich im Spiel in Form zweier sechsseitiger Würfel manifestiert.[516] Um die Performanz der Monte Carlo Spielbaumsuchläufe zu evaluieren, wurde das Backgammonspiel in seinem intialen Zustand betrachtet und die Zugmöglichkeiten analysiert, die aus dem jeweils ersten Würfelwurf resultieren. Jeder Agent des vierköpfigen Ensembles arbeitet mit einem exemplarischen UCT-Konstantenwert von 1.0, einer maximalen Simulationsschrittzahl von 50000 und mit einer flachen Monte Carlo Spielbaumsuche, anhand derer die umittelbaren Zugmöglichkeiten fokussiert werden. Durchgeführt wurden zehn Testläufe mit einer jeweiligen Laufzeitdauer von zehn Sekunden. Die Ergebnisse der Versuchsläufe finden sich in folgender Tabelle wiedergegeben.

Würfelwert	Rollouts	Simulationsschritte	Schritte pro Rollout (gerundet)
5	118	154811	1312
3	146	191446	1311
5	108	155078	1436
4	157	192563	1227
1	54	297981	5518
6	136	198153	1383
3	99	140247	1417
6	118	147540	1250
3	71	135377	1907
2	120	257551	2146

[515] Vgl. Glonnegger 2009, S. 34. Einen Überblick über zahlreiche Variationen und Regelmodifikationen des Backgammonspieles wie *Plakoto* oder *Fevga* findet sich unter http://www.bkgm.com/variants (zuletzt aufgerufen am 28.04.2014).

[516] Für einen Überblick über die Backgammon-Regeln sei auf Anhang 13.3.2 verwiesen, in dem die Regeln erläutert und die Umsetzung mit dem Framework besprochen werden.

Im Durchschnitt wurden rund 113 Spiele im Rahmen einer Laufzeit von zehn Sekunden simuliert bei einer durchschnittlichen Anzahl von knapp 1871 Simulationsschritten pro simuliertem Spiel. Aufgrund des spielübergreifenden browserbasierten Ansatzes und der Vorgehensweise der Agenten, die sich frei von Expertenwissen gestaltet, liegt die Leistungsfähigkeit der vorliegenden JavaScript-Implementation jenseits spieldedizierter Umsetzungen wie die Anwendung *McGammon*, deren Erfolg bereits im Kapitel 4.2.6 zu State of the Art Umsetzungen der Monte Carlo Spielbaumsuche in digitalen Brettspielen vorgestellt wurde. In seiner 2007 veröffentlichten Version simulierte *McGammon* seinen Entwicklern zufolge rund 6500 Spiele pro Sekunde auf einem mit 2,6 GHz getakteten Quad-Opteron-System.[517]

Die Performanz von *McGammon* beruht zum einen auf der Verwendung der maschinennahen Programmiersprache *C++*, die der Umsetzung des Programmes dient, zum anderen bringen expertisebasierte Anpassungen die simulierten Spielpartien früher zu ihren terminalen Zuständen – Anpassungen wie die Vereinfachung der Spielregeln in den Rollouts: In *McGammon* gewinnt der Spieler die simulierte Partie, der alle seine Spielfiguren in sein eigenes Zielfeld gebracht hat, so dass das zeitintensive Auswürfeln der Spielfiguren entfällt.[518] Die Einbindung ebensolcher externer Kriterien zur Verkürzung der Simulationsdauer ist der Arbeitsweise und Leistungsfähigkeit des Frameworks *SpoookyJS* dienlich und aufgrund des modularen Aufbaus des Rahmenwerkes für das Backgammon-Spiel umsetzbar.

[517] Vgl. van Lishout et al. 2007, S. 182.
[518] Vgl. van Lishout et al. 2007, S. 182.

7 Zusammenfassung und Ausblick

In ihren ersten Kapiteln beschäftigte sich die vorliegende Arbeit mit dem Begriff der künstlichen Intelligenz, dem Vokabular in der Beschreibung von Spielen und dem Konzept intelligenter Agenten. Anschließend wurden unterschiedliche klassische Verfahren der künstlichen Intelligenz zur suchbasierten Entscheidungsfindung in digitalen Brettspielen besprochen – grundlegende Verfahren wie der Minimax-Algorithmus und die Alpha-Beta Kürzung sowie spielspezifische Optimierungen wie die Zugvorsortierung, die Ruhesuche oder Eröffnungs- und Endspieldatenbanken. Offenbarte sich der Stand der Kunst mit den Systemen *Chinook* und *Deep Blue* im Dame- und Schachspiel als das Spielvermögen höchst spezialisierter Soft- und dedizierter Hardwarelösungen, wurde mit dem Verfahren der Monte Carlo Spielbaumsuche und ihrem UCT-Selektionsalgorithmus eine Methode zur Entscheidungsfindung referiert, die spielübergreifendes artifizielles Spielvermögen zu realisieren vermag und sich mit den Anwendungen *Fuego* und *MoGo* im hochkomplexen Go-Spiel als besonders erfolgreiche Methode zur Entscheidungsfindung gerierte. Abgeschlossen wurde der Methoden- und State of the Art Komplex mit der Erläuterung künstlicher neuronaler Netze und Verfahren des verstärkenden Lernens (reinforcement learning), anhand derer sich Spielmuster und auszahlungsoptimierende Spielmerkmale expertenunabhängig auf Grundlage von Umweltfeedback lernen lassen. Als besonders erfolgreiche Anwendung von Methoden des reinforcement learnings im Backgammon-Spiel präsentierte sich die Anwendung *TD-Gammon*.

Eine besondere Herausforderung in der Entwicklung artifizieller Gegner in digitalen Spielen wurde mit dem Anforderungskomplex des General Game Playing vorgetragen: Ist es möglich, Softwaresysteme zu entwickeln, die des Spielens in unterschiedlichsten Spieldomänen fähig sind? Softwaresysteme, die artifizielles Spielverhalten auf Grundlage von Spielbeschreibungen realisieren, die in der Game Description Language und ihren Erweiterungen vorliegen? Positiv beantwortet wurden die gestellten Fragen mit den Anwendungen *Cluneplayer*, *Fluxplayer* und *Cadiaplayer*, die im jährlich ausgetragenen Wettbewerb der

General Game Playing Competition zu bestehen vermochten, indem sie klassische Methoden der künstlichen Intelligenz oder die Monte Carlo Spielbaumsuche zur Identifizierung sinnvoller Zugmöglichkeiten verwendeten.

Im Anschluss an die Methoden- und State of the Art Kapitel wurde das browserbasierte JavaScript-Framework *SpoookyJS* vorgestellt, das im Rahmen der vorliegenden Arbeit entwickelt wurde und drei Beiträge zur Forschung um spielübergreifende künstliche Intelligenz in digitalen Brettspielen leistet:

- *SpoookyJS* stellt ein Framework bereit, mit dem sich browserbasierte Spiele mit basalen Kenntnissen der Programmiersprache JavaScript leicht erstellen lassen.
- *SpoookyJS* stellt artifizielle Gegner für alle Spiele bereit, die mit dem Framework umgesetzt wurden.
- *SpoookyJS* bietet ob seines modularen Aufbaus und der intendierten Weiterentwicklungsmöglichkeiten eine Test- und Entwicklungsumgebung für Experimente, die sich mit der Forschung um artifizielle Spielentscheidungen in digitalen Brettspielen beschäftigen.

In den folgenden Subkapiteln seien die Beiträge gesondert herausgearbeitet und sei auf die Erweiterung der implementierten Ansätze eingegangen.

7.1 Vereinfachte Spielerstellung und General Game Playing

Um browserbasierte Spiele mit dem Framework zu erstellen, stehen Vorlageskripten sowie Rastergrafiken für Spielfiguren bereit. Einzig ist die Bearbeitung der Skriptdatei *game.js* im entsprechenden Spieleordner notwendig. Ist die Spielwelt standardmäßig eine zweidimensionale, so stellt das Framework Methoden bereit, um lineare oder netzartige Spielwelten zu generieren. Spiele mit verschiedenen Interaktionsbereichen wie das Backgammon-Spiel lassen sich unter Einsatz der bereitgestellten Klassen und Methoden des Frameworks umsetzen. Spielfiguren sind in Form von Entitätenblaupausen als JavaScript-Objektliteral umgesetzt und lassen sich mit beliebigen Zugmöglichkeiten und Entitätenzielen ausstatten, die mit kausalen Folgen für Spielwelt und den Spielverlauf verbunden

sind. Zur Umsetzung von Spiel- und Zugregeln wird zwischen Regelatomen und den finalen zusammengesetzten Regeln differenziert: Zunächst werden mögliche Zustände der Spielwelt anhand der Funktion `addGameRuleAtom` bestimmt, die anschließend mit der Funktion `assembleGameRule` zu einer Spielregel zusammengesetzt werden. Welche Folgen sich mit der im Spielverlauf erfüllten Spielregel verbinden, wird mit der Funktion `connectGameRuleConsequences` definiert.[519] Aufgrund des browserbasierten Ansatzes, bei dem sich Änderungen am Quelltext der Spieldefinitionen unmittelbar betrachten und nachvollziehen lassen, stellt sich die Einarbeitung in das Funktionalitätsangebot des Frameworks als wenig zeitintensiv dar, so dass sich die Verwendung von *SpoookyJS* in universitärer und außeruniversitärer Lehrangeboten vorstellen ließe.[520]

Unmittelbare Erweiterungsmöglichkeiten, die der Erstellung von Spielen mit dem Framework dienlich sind, bieten sich zum einen mit der Ergänzung anderweitiger Spielzustände in *Spoooky.Game.gameRuleAtoms* und weiterer Spielereignisse, die in der Klasse *Spoooky.GameEvents* definiert werden. Aufgrund der modularen Architektur des Rahmenwerkes und seiner zentralisierten Funktionalität in Form des Spielskriptes *game.js*, ließe sich die Erstellung von Spielen anhand eines Spielgenerators vorstellen, der Eigenschaften des zu implementierenden Spieles abfragt – Eigenschaften wie die Art der Spielwelt, die Anzahl der Spielfelder, die Beschaffenheit der Spielfiguren sowie die Zusammensetzung von Spielregeln – und eine neue Skriptdatei *game.js* mitsamt zugehöriger Ordnerstruktur, HTML-Dateien und Rastergrafiken generiert.

Eine sinnvolle Erweiterung von *SpoookyJS* stellt die Anbindung an die Game Description Language (GDL) dar. So ist es vorstellbar, Methoden zum Einlesen von

---

[519] Die drei Funktionen `addGameRuleAtom`, `assembleGameRule` und `connectGameRuleConsequences` gehören der Klasse *Spoooky.Game* an.

[520] Vgl. die Intention, Schülerinnen und Schülern informatisches Denken (*Computational Thinking*) anhand von Brettspielen zu vermitteln: „[Die Adaption von Brettspielen] beinhaltet folgende Vorteile. Erstens kann man diese Spiele spielen, um sich in den Kontext einzuarbeiten. Zweitens hat man es in diesen Spielen mit einer überschaubaren Anzahl von Objekten zu tun, deren Eigenschaften und Aufgaben objektorientiert beschreibbar sind. Drittens scheint dieser Kontext Schüler zu motivieren […]. Viertens können ‘Computerspieler‘ die angesprochenen Fragen zum Unterschied menschlicher und maschineller Denkweisen zumindest ansatzweise einbezogen werden. Fünftens ergeben sich durch die Virtualisierung des Brettspiels neue Möglichkeiten das Spiel zu spielen und es zu skalieren.“ (Engbring 2014, S. 17).

Spielbeschreibungen zu implementieren, die in der GDL vorliegen. Die Beschreibungen im GDL-Format ließen sich anschließend parsen und verarbeiten, so dass aus einer GDL-Datei ein browserbasiertes Spiel auf Grundlage von *SpoookyJS* resultierte.

Neue Aspekte und Herausforderungen in der Umsetzung spielübergreifender artifizieller Intelligenz eröffnen sich mit der Implementation weiterer Spiele und Spielvariationen: Abwandlungen des Schachspieles wie *DarkChess* transformieren das Schachspiel in ein Spiel unvollständiger Information oder integrieren mit der Spielvariante *Schach960* das Moment des Zufalls in die Initialisierungsphase der Spielpartie, so dass Eröffnungsdatenbanken an Wert verlieren, gar hinfällig werden. Spiele wie *EinStein würfelt nicht*, *Arimaa*, *Hex*, *Lines of Action* oder das Go-Spiel erfordern neue Interaktionselemente, die sich in das Framework einhängen lassen sowie Spielregeln, die sich aus neuen, zu formulierenden Regelatomen konstituieren. Neben erweiterten Interaktionselementen dehnen Kartenspiele wie Poker das Beschäftigungsfeld auf Spiele mit versteckter Information aus.

7.2 Artifizielles Spielvermögen und Erweiterungen

Die Leistungsfähigkeit und Spielstärke der spielübergreifenden künstlichen Intelligenz, die von *SpoookyJS* bereitgestellt wird, gestaltet sich der durchgeführten Evaluation zufolge noch zögerlich. Zwar leistet der multiagentenbasierte Ansatz auf Basis der Monte Carlo Spielbaumsuche mit dem UCT-Selektionsalgorithmus in Spielen schlichter Komplexität gute Arbeit, in Spielen mit hoher Zustandsraum- und Spielbaumkomplexität gelangt der browserbasierte Ansatz aufgrund der stark eingeschränkt zur Verfügung stehenden Ressourcen jedoch schnell an seine Grenzen.

Erweiterungs- und Verbesserungsmöglichkeiten der Leistungsfähigkeit bieten sich an zahlreichen Stellen: Setzt sich die Entwicklung von Webbrowsern und ihren JavaScript-Engines auch in Zukunft so rasant fort wie in den vergangenen Jahren, so ist künftig von einer beachtlichen Leistungssteigerung in der Verarbeitung von

JavaScript-Anweisungen auszugehen. Jenseits externer Leistungssteigerungen[521] gestalten sich Frameworkinterna wie die effektive Verarbeitung von Regelatomen und zusammengesetzten Regeln, das performante Klonen von Spielzuständen oder die Aufbereitung der in einem Spielzustand ausführbaren Zugmöglichkeiten als sinnvolle Anknüpfungspunkte für die Verbesserung der Gesamtleistung von *SpoookyJS*.

Als besonders kritisch für die Qualität der Entscheidungsfindung offenbarte sich in den Evaluationsläufen die Rollout-Phase der Monte Carlo Spielbaumsuche: Je höher die Zahl der simulierten Spiele, desto aussagekräftiger und sinnvoller die Zugentscheidungen des Agententeams. Um die Zahl der simulierten Spiele zu erhöhen, ließe sich menschliche Expertise durch externe Module einbringen – Expertise wie das in der Software *McGammon* erfolgreich angewandte Verfahren, die Auswürfelphase im Backgammon-Spiel vorzeitig zu beenden, wenn ein Spieler alle seine Spielsteine in sein Zielfeld gebracht hat. Fungierte das Lernmodul des Meta Agenten in der vorliegenden Implementation als Proof of Concept, ist eine intensivere An- und Einbindung des Moduls vorstellbar, um Spielzustände und ihre individuelle Auszahlungsgüte zu memorieren. Neben der Einbindung des Q-Lernmodules erachtet sich die Implementation und Verwendung eines künstlichen neuronalen Netzes als sinnvoll, um Spielmuster zu lernen, die sich beispielsweise im Gomoku-Spiel dazu verwenden ließen, um Bedrohungs- und Belohnungssituationen zu memorieren und erfolgreich abzurufen.

7.3 Entwicklungsumgebung und multiagentenbasierter Ansatz

In einem frühen Entwicklungsstadium intendierte die vorliegende Arbeit, den multiagentenbasierten Ansatz auf die unterste Ebene zu übertragen, die Ebene der einzelnen Spielfiguren. So wurde jede Spielfigur als separater Agent vorgesehen, der über die Möglichkeit verfügte, seine Umgebung wahrzunehmen, mit seiner Umwelt zu interagieren und individuelle Ziele zu verfolgen. Durch die Kooperation der Agenten miteinander und der Abstimmung über sinnvolle

[521] Vgl. auch das Vorhaben, die Monte Carlo Spielbaumsuche auf Grafikprozessoren (GPU) durchzuführen: Barriga et al. 2014.

Zugmöglichkeiten emergiere schlaues artifizielles Agieren in unterschiedlichen Brettspielen, so die Idee des basalen multiagentenbasierten Ansatzes. In der Spiel- und Entscheidungspraxis stellte sich die Interpretation und Umsetzung der Spielfiguren als individuelle Agenten jedoch als wenig dienlich dar: Wie lassen sich artifizielle Entscheidungen in Spielen treffen, die als Ausgangszustand eine leere Spielwelt haben? Wie lassen sich Spielfiguren in den Ansatz integrieren, die auf dem Spielbrett abgelegt werden und folglich erst im Anschluss an die Entscheidungsfindung zu agieren vermögen?[522]

Aufgrund der mangelnden Flexibilität des Spielfigurenansatzes in seiner Übertragung auf ein spielübergreifendes System, wurde der Ansatz in die Methodik des Agentenensembles transformiert. So obliegt die Entscheidungsfindung in *SpoookyJS* keiner zentralen artifiziellen Intelligenz, sondern einem Team individueller Agenten, die unterschiedliche Aspekte des Spieles analysieren und ihre Zugvorschläge an den übergeordneten Meta Agenten weiterleiten. Jene Genese des Frameworks, die den Spielfigurenansatz in den Ansatz des Agentenensembles überführte, ist der Intention der vorliegenden Arbeit charakteristisch: Mit seinem hohen Maß an Erweiterbarkeit stellt *SpoookyJS* ein *Testbed* für Experimente bereit, die sich mit artifiziellen multiagentenbasierten Spielentscheidungen in digitalen Brettspielen beschäftigen.

Mit den Fragen nach der Beschaffenheit individueller und kollektiver Intentionalität, dem Agentendesign, der Agentenkommunikation und der Kooperation von Agenten bietet der Multiagentenansatz, wie er dem Framework grundlegend ist, ein breites Feld für die theoretische und praktische Auseinandersetzung mit multiagentenbasierter künstlicher Intelligenz in digitalen Brettspielen: (Wann) lässt sich den Agenten des Ensembles und dem Meta Agenten eine Absicht in ihren Spielhandlungen beimessen?[523] Ist das Handeln der

[522] Vgl. auch die verwandten Arbeiten *AgentChess – An Agent Chess Approach – Can Agents Play Chess?* (Fransson 2003) sowie *When Ants Play Chess – Or Can Strategies Emerge From Tactical Behaviors?* (Drogoul 1995).

[523] Vgl. die annähernde Differenzierung zwischen absichtlichen und nicht absichtlichen Handlungen: „Was unterscheidet absichtliche Handlungen von solchen, die nicht absichtlich sind?

Agenten zielgerichtet oder haben die Agenten einen intrinsischen Grund, sich auf ihre bestimmte Art und Weise in der Spielwelt zu verhalten?[524] Vermögen die Agenten, Handlungen in ihren Spielumgebungen zu intendieren und wie konstituiert sich ein gemeinsames kollektiv-intendiertes Handeln der Agenten – ein Handeln, das „dadurch gekennzeichnet ist, dass es auf besondere Art und Weise beabsichtigt ist“[525]?

Aufbauend auf dem grundlegenden Charakter des in der vorliegenden Arbeit umgesetzten Agentenmodelles ist zu untersuchen, inwieweit sich ein elaboriertes Agentendesign wie das von Bratman vorgeschlagene *belief-desire-intention* (*BDI*) Modell [526] auf die spielübergreifende Entscheidungsfindung zum einen im reduzierten Kontext isolierter Agenten, zum anderen im Agentenkollektiv übertragen und sinnvoll anwenden lässt. Im Rahmen des BDI-Modells ist ein Agent anhand von Ansichten und Überzeugungen (*beliefs*) informiert über sich, über seine Umwelt und weitere Agenten, die sich in der Agentenumwelt befinden; die Ansichten des Agenten über seine Umwelt bilden das Weltwissen des Agenten. Als Ziele (*desires*)[527] werden alle möglichen Umwelt- und Selbstzustände bezeichnet, die der Agent zu Stande bringen könnte. Der Agent wählt aus einer Menge von Handlungsoptionen eine Handlung aus, die ihm in einem bestimmten Zustand seiner Umwelt zur Verfügung stehen, um sich seinem Ziel anzunähern. In der Absicht (*intention*), sein Ziel zu erreichen, wählt der Agent aus seinen Handlungsmöglichkeiten aus:

> „Intentions are the states of affairs that the agent has decided to work towards. Intentions may be goals that are delegated to the agent, or may result from

Die Antwort, die ich vorschlagen werde, lautet: Das sind jene Handlungen, bei denen die in einem bestimmten Sinn gestellte Frage ‘Warum?‘ Anwendung findet“ (Anscombe, G. E. M. 2011, S. 23).

[524] Vgl. die Kausalismus-Teleologie-Debatte in Horn 2010, S. 8–25.

[525] Schmid und Schweikard 2009, S. 14.

[526] Vgl. Bratman 1999.

[527] Zur Entsprechung der Bezeichner *desire* und *goal* vgl.: „An agent’s *desires* represent states of affairs (options) that it would choose. We usually use the term *goal* for this concept, but for historical reasons we use the abbreviation BDI.“ (Dunin-Kęplicz und Verbrugge 2010, S. 4, Kursivschrift nach den Autoren).

> considering options: we think of an agent looking at ist options and choosing between them. Options that are selected in this way become intentions.“[528]

Bereits im Hinblick auf das Spielverhalten der Agenten im Gomoku-Spiel scheint die differenzierende Umsetzung von Agentenzielen und Agentenabsichten sinnvoll: Da sich das Hauptproblem der agentenbasierten Entscheidungsfindung im Gomoku-Spiel in der Erkennung von bedrohlichen Situationen und der mangelnden Zielgerichtetheit der Agenten zeigte, ist die Ergänzung des Frameworks um individuelle Agentenziele („Ich sollte versuchen, die Anzahl meiner in Folge liegenden Spielsteine zu maximieren“) sowie Intentionen und Pläne („Wenn ich in dieser Runde meinen Spielstein auf jenes Feld ablege, stelle ich dem Gegner eine Falle, die mich womöglich zum Sieg führt“) wünschenswert.

In der Zielsetzung der vorliegenden Arbeit, ein hohes Maß an Wiederverwendbarkeit und Erweiterbarkeit der Klassen, Methoden und Algorithmen des Frameworks zu bieten, gestaltet sich das Konzept eines Multiagentensystems als Fixpunkt, das mit seinen Problem- und Fragekomplexen um die Kommunikation der Agenten, der elaborierten Abstimmung über sinnvolle Züge im Agentenensemble und der automatisierten Bildung von aufgabenspezifischen Agententeams[529] ein breites Feld für Erweiterungen des JavaScript-Frameworks bieten.

[528] Bordini et al. 2007, S. 16.
[529] Vgl. u.a. Weiss 2013 für Herausforderungen in der Umsetzung von Multiagentensystemen.

8 Literaturverzeichnis

Abramson, Bruce (1990): Expected-outcome: A general model of static evaluation. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 12 (2), S. 182–193.

Akl, Selim; Newborn, Monroe (1977): The principal continuation and the killer heuristic. In: Proceedings of the ACM Annual Conference, S. 466–473.

Allis, L. V. (1994): Searching for solutions in games and artificial intelligence. Dissertationsschrift. University of Limburg, Maastricht. Online verfügbar unter http://fragrieu.free.fr/SearchingForSolutions.pdf, zuletzt geprüft am 09.10.2013.

Allis, Victor; Herik, H.J. van den; Huntjens, M.P.H. (1993): Go-Moku and threat-space search. Maastricht: University of Limburg, Department of Computer Science (Technical reports in computer science, CS 93-02).

Anscombe, G. E. M. (2011): Absicht. 1. Aufl. Berlin: Suhrkamp (Suhrkamp-Taschenbuch Wissenschaft, 1978).

Auer, Peter; Cesa-Bianchi, Nicolo; Freund, Yoav; Schapire, Robert E. (1995): Gambling in a rigged casino: The adversarial multi-armed bandit problem. In: Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on. IEEE, S. 322–331.

Auer, Peter; Cesa-Bianchi, Nicolò; Fischer, Paul (2002): Finite-time analysis of the multiarmed bandit problem. In: *Machine learning* 47 (2-3), S. 235–256.

Axelrod, Robert (2009): Die Evolution der Kooperation. 7. Aufl. München: Oldenbourg (Scientia Nova).

Balzert, Heide (2011): Lehrbuch der Objektmodellierung. Analyse und Entwurf mit der UML 2 ; mit e-learning-Online-Kurs. 2. Aufl., Nachdr. 2011. Hg. v. Helmut Balzert. Heidelberg: Spektrum, Akad. Verl (Lehrbücher der Informatik).

Barney Pell (1992): METAGAME: A new challenge for games and learning. In: Programming in Artificial Intellegence: The Third Computer Olympiad. Ellis

Horwood: Ellis Horwood Limited, S. 237–251. Online verfügbar unter http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.88.1331, zuletzt geprüft am 19.08.2014.

Barriga, Nicolas A.; Stanescu, Marius; Buro, Michael (2014): Parallel UCT search on GPUs. In: IEEE Conference on Computational Intelligence and Games (CIG). Online verfügbar unter http://kghost.de/cig_proc/full/paper_50.pdf, zuletzt geprüft am 20.10.2014.

Baxter, Jonathan; Tridgell, Andrew; Weaver, Lex (1998): KnightCap: A Chess Programm That Learns by Combining TD (lambda) with Game-Tree Search. In: Proceedings of the Fifteenth International Conference on Machine Learning. Morgan Kaufmann Publishers Inc, S. 28–36.

Baxter, Jonathan; Tridgell, Andrew; Weaver, Lex (2000): Learning to play chess using temporal differences. In: *Machine learning* 40 (3), S. 243–263.

Beckermann, Ansgar (2008): Analytische Einführung in die Philosophie des Geistes. 3., aktualisierte und erw. Aufl. Berlin [u.a.]: de Gruyter (De-Gruyter-Studienbuch).

Beil, Benjamin (2013): Game Studies. Eine Einführung. Berlin [u.a.]: LIT (Red Guide).

Bellman, Richard (2010): Dynamic programming. 1st Princeton landmarks in mathematics ed. Princeton: Princeton University Press (Princeton landmarks in mathematics).

Bennett, Maxwell; Hacker, Peter (2012): Die philosophischen Grundlagen der Neurowissenschaften. 2. Aufl. Darmstadt: WBG (Wissenschaftliche Buchgesellschaft).

Berninghaus, Siegfried; Ehrhart, Karl-Martin; Güth, Werner (2010): Strategische Spiele. Eine Einführung in die Spieltheorie. 3. Aufl. Berlin, Heidelberg: Springer Verlag.

Bewersdorff, Jörg (2012): Glück, Logik und Bluff. Mathematik im Spiel - Methoden, Ergebnisse und Grenzen. 6., akt. Aufl. 2012. Wiesbaden: Vieweg+Teubner Verlag.

Björk, Staffan; Holopainen, Jussi (2006): Patterns in game design. Boston, Massachusetts: Charles River Media (Game development series).

Björnsson, Yngvi; Finnsson, H. (2009): CadiaPlayer: A Simulation-Based General Game Player. In: *Computational Intelligence and AI in Games, IEEE Transactions on* 1 (1), S. 4–15. DOI: 10.1109/TCIAIG.2009.2018702.

Bordini, Rafael H.; Hübner, Jomi Fred; Wooldridge, Michael J. (2007): Programming multi-agent systems in AgentSpeak using Jason. Chichester, England, Hoboken, NJ: J. Wiley.

Bratman, Michael (1999): Intention, plans, and practical reason. Stanford, Calif: Center for the Study of Language and Information (David Hume series).

Browne, Cameron B.; Powley, Edward; Whitehouse, Daniel; Lucas, Simon M.; Cowling, Peter I.; Rohlfshagen, Philipp et al. (2012): A survey of monte carlo tree search methods. In: *Computational Intelligence and AI in Games, IEEE Transactions on* 4 (1), S. 1–43.

Brügmann, Bernd (1993): Monte carlo go. Technical Report. Max Planck Institute for Physics. Online verfügbar unter http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.10.449&rep=rep1&type=pdf, zuletzt geprüft am 06.03.2014.

Busch, Gabriel Christoph Benjamin (1821): Handbuch der Erfindungen: Elfter Theil, die Buchstaben R und S enthaltend: Wittekindt (Handbuch der Erfindungen: Elfter Theil, die Buchstaben R und S enthaltend). Online verfügbar unter http://books.google.de/books?id=7ghCAAAAcAAJ.

Cai, Xindi; Wunsch, DonaldC., II (2007): Computer Go: A Grand Challenge to AI. In: Włodzisław Duch und Jacek Mańdziuk (Hg.): Challenges for Computational Intelligence, Bd. 63: Springer Berlin Heidelberg (Studies in Computational Intelligence), S. 443-465. Online verfügbar unter http://dx.doi.org/10.1007/978-3-540-71984-7_16.

Caixeta, Gutierrez Soares; Silva Julia, Rita Maria (2008): A Draughts Learning System Based on Neural Networks and Temporal Differences: The Impact of an Efficient Tree-Search Algorithm. In: Proceedings of the 19th Brazilian

Symposium on Artificial Intelligence: Advances in Artificial Intelligence. Berlin, Heidelberg: Springer-Verlag (SBIA ’08), S. 73–82. Online verfügbar unter http://dx.doi.org/10.1007/978-3-540-88190-2_13.

Campbell, Murray; Hoane Jr., A. Joseph; Hsu, Feng-hsiung (2002): Deep blue. In: *Artificial Intelligence* 134 (1), S. 57–83.

Carion, Ulysse (2013): Creating a Chess-Playing Computer Program. Online verfügbar unter http://www.ulyssecarion.com/ComputerChess.pdf, zuletzt geprüft am 05.11.2013.

Carroll, Lewis (2000): Durch den Spiegel und was Alice dort fand. Stuttgart: Reclam (Universal-Bibliothek, 9747).

Cazenave, Tristan; Jouandeau, Nicolas (2007): On the parallelization of UCT. In: Proceedings of the Computer Games Workshop, S. 93–101.

Chaslot, Guillaume (2010): Monte-Carlo Tree Search. Ph. Universiteit Maastricht, Maastricht. Online verfügbar unter https://project.dke.maastrichtuniversity.nl/games/files/phd/Chaslot_thesis.pdf, zuletzt geprüft am 06.03.2014.

Chaslot, Guillaume; Bakkes, Sander; Szita, Istvan; Spronck, Pieter (2008a): Monte-Carlo Tree Search: A New Framework for Game AI. In: AIIDE.

Chaslot, Guillaume; Winands, Mark; Uiterwijk, JWHM; van den Herik, H; Bouzy, Bruno (2008b): Progressive strategies for monte-carlo tree search. In: *New Math. and Nat. Computation* 04 (03), S. 343–357. DOI: 10.1142/S1793005708001094.

Chaslot, Guillaume MJ-B; Winands, Mark H. M.; van Den Herik, H Jaap (2008c): Parallel monte-carlo tree search. In: Computers and Games: Springer, S. 60–71.

Chess Programming Wiki: Evaluation. Online verfügbar unter http://chessprogramming.wikispaces.com/Evaluation, zuletzt geprüft am 30.09.2013.

Chess Programming Wiki: Point Value. Online verfügbar unter http://chessprogramming.wikispaces.com/Point+Value, zuletzt geprüft am 30.09.2013.

Clune, James (2007): Heuristic Evaluation Functions for General Game Playing. In: Proceedings of the 22nd National Conference on Artificial Intelligence - Volume 2: AAAI Press (AAAI'07), S. 1134–1139. Online verfügbar unter http://www.aaai.org/Papers/AAAI/2007/AAAI07-180.pdf, zuletzt geprüft am 21.08.2014.

computerchess.org.uk: Computer Chess Rating Lists Website. Online verfügbar unter http://computerchess.org.uk/, zuletzt geprüft am 06.11.2013.

Coquelin, Pierre-Arnaud; Munos, Rémi (2007): Bandit Algorithms for Tree Search. In: Ronald Parr und van der Gaag, Linda C. (Hg.): UAI 2007, Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence, Vancouver, BC, Canada, July 19-22, 2007: AUAI Press, S. 67–74.

Crockford, Douglas (2008): JavaScript: The Good Parts. Working With the Shallow Grain of JavaScript. Farnham: O'Reilly.

Davis, Morton D. (2005): Spieltheorie für Nichtmathematiker. 4. Aufl. München: Oldenbourg Wissenschaftsverlag.

Dennett, D. C. (1998): Brainchildren. Essays on designing minds. Cambridge, Mass: MIT Press (Representation and mind).

Dickfeld, Gunnar (2003): Stein für Stein. Eine Einführung in das Brettspiel Go. 1. Aufl. Frankfurt/Main [Scheffelstr. 23]: G. Dickfeld.

Diekmann, Andreas (2009): Spieltheorie: Einführung, Beispiele, Experimente. 2. Aufl. Reinbek bei Hamburg: Rowohlt.

Dresler, Martin (2009): Künstliche Intelligenz, Bewusstsein und Sprache. Das Gedankenexperiment des "chinesischen Zimmers". Würzburg: Königshausen & Neumann.

Drogoul, Alexis (1995): When Ants Play Chess (Or Can Strategies Emerge From Tactical Behaviors?). In: In Proceedings of Fifth European Workshop on

Modelling Autonomous Agents in a Multi-Agent World (MAAMAW ’93): Springer, S. 13–27.

Droste, Sacha (2008): Lernen von Evaluierungsfunktionen für Schachvarianten. Diplomarbeit. Technische Universität Darmstadt, Darmstadt. Fachbereich Informatik, Fachgebiet Knowledge Engineering. Online verfügbar unter http://www.ke.tu-darmstadt.de/lehre/arbeiten/diplom/2008/Droste_Sacha.pdf, zuletzt geprüft am 17.12.2013.

Droste, Sacha; Fürnkranz, Johannes (2008): Learning the Piece Values for Three Chess Variants. In: *ICGA Journal* 31 (4), S. 209–233.

Dunin-Kęplicz, Barbara; Verbrugge, Rineke (2010): Teamwork in multiagent systems. Hoboken, N.J, Chichester: Wiley; John Wiley [distributor].

Engbring, Dieter (2014): Zum Verhältnis von Informatik und Naturwissenschaften Ein Vorschlag zur MINT-Förderung. In: Informatik und Natur: 6. Münsteraner Workshop zur Schulinformatik. BoD-Books on Demand, S. 9–18.

Ensmenger, N. (2012): Is chess the drosophila of artificial intelligence? A social history of an algorithm. In: *Social Studies of Science* 42 (1), S. 5–30. DOI: 10.1177/0306312711424596.

Enzenberger, Markus; Müller, Martin (2010): A lock-free multithreaded Monte-Carlo tree search algorithm. In: Advances in Computer Games: Springer, S. 14–20. Online verfügbar unter http://webdocs.cs.ualberta.ca/~emarkus/publications/enzenberger-mueller-acg12.pdf, zuletzt geprüft am 13.02.2014.

Epstein, Susan (1994): Identifying the right reasons: Learning to filter decision makers. In: Proceedings of the AAAI 1994 Fall Symposium on Relevance. New Orleans: AAAI Press, S. 68–71.

Ertel, Wolfgang (2013): Grundkurs Künstliche Intelligenz. Eine praxisorientierte Einführung. 3. Aufl. Wiesbaden: Springer Fachmedien (Lehrbuch).

Fahlman, Scott E. (1988): An empirical study of learning speed in back-propagation networks. Pittsburgh, Pa: School of Computer Science Carnegie Mellon Univ (CMU-CS, 88,162).

Favre-Bulle, Bernard (2000): Information und Zusammenhang. Informationsfluss in Prozessen der Wahrnehmung, des Denkens und der Kommunikation. Wien: Springer.

Feist, M. (1999): The 9th World Computer-Chess Championship: Report on the tournament. In: *ICCA JOURNAL* 22 (3), S. 155–164.

Fern, Alan; Lewis, Paul (2011): Ensemble Monte-Carlo Planning: An Empirical Study. In: Fahiem Bacchus, Carmel Domshlak, Stefan Edelkamp und Malte Helmert (Hg.): Proceedings of the 21st International Conference on Automated Planning and Scheduling, ICAPS 2011, Freiburg, Germany June 11-16, 2011: AAAI.

Finley, Klint (2012): Did a Computer Bug Help Deep Blue Beat Kasparov? In: *Wired*, 28.09.2012. Online verfügbar unter http://www.wired.com/playbook/2012/09/deep-blue-computer-bug/, zuletzt geprüft am 15.10.2013.

Finnsson, Hilmar (2007): Cadia-Player: A general game playing agent. Masterthesis. Reykjavík University, Reykjavík. School of Computer Science. Online verfügbar unter http://skemman.is/stream/get/1946/7478/19933/1/MSc_Hilmar-Finnsson.pdf, zuletzt geprüft am 22.08.2014.

Finnsson, Hilmar (2012): Generalized Monte-Carlo Tree Search Extensions for General Game Playing. In: AAAI. Online verfügbar unter http://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/viewFile/4935/5300, zuletzt geprüft am 06.11.2013.

Finnsson, Hilmar; Björnsson, Yngvi (2008): Simulation-Based Approach to General Game Playing. In: AAAI, Bd. 8, S. 259–264. Online verfügbar unter http://www.ru.is/starfsfolk/yngvi/pdf/FinnssonB08a.pdf, zuletzt geprüft am 19.08.2014.

Finnsson, Hilmar; Björnsson, Yngvi (2011): CadiaPlayer: Search-Control Techniques. In: *KI - Künstliche Intelligenz* 25 (1), S. 9-16. DOI: 10.1007/s13218-010-0080-9.

Firtman, Maximiliano R. (2013): Programming the mobile web. 2nd ed. Sebastopol, CA: O'Reilly Media.

Flanagan, David (2011): JavaScript. The definitive guide. 6th ed. Beijing, Sebastopol, CA: O'Reilly.

Fogel, David B. (2001): Blondie24. Playing at the Edge of AI. San Francisco, Calif, London: Morgan Kaufmann; International Thomson.

Fogel, David B. (2006): Evolutionary computation. Toward a new philosophy of machine intelligence. 3rd ed. Hoboken, N.J: John Wiley & Sons (IEEE Press series on computational intelligence).

Fransson, Henric (2003): AgentChess - An Agent Chess Approach. Masterthesis. Blekinge Institute of Technology, Ronneby, Schweden. Department of Software Engineering and Computer Science. Online verfügbar unter http://www.bth.se/fou/cuppsats.nsf/all/20ff8a1d9a6f7d09c1256cb9003f1ab2/$file/Master%20Thesis%20-%20AgentChess.pdf, zuletzt geprüft am 17.10.2014.

French, Robert M. (2012): Moving beyond the Turing test. In: *Commun. ACM* 55 (12), S. 74–77. DOI: 10.1145/2380656.2380674.

Fulton, Steve; Fulton, Jeff (2011): HTML5 Canvas. Native interactivity and animation for the Web. 1st ed. Sebastopol, CA: O'Reilly.

GamesCoop (2012): Theorien des Computerspiels zur Einführung. Hamburg: Junius.

Gamma, Erich (1995): Design patterns. Elements of reusable object-oriented software. Reading, Mass: Addison-Wesley (Addison-Wesley professional computing series).

Geller, Tom (2007): Checkmate for checkers. Computer program is unbeatable at English draughts, 19.07.2007. Online verfügbar unter

http://www.nature.com/news/2007/070716/full/news070716-13.html, zuletzt geprüft am 09.10.2013.

Gelly, Sylvain; Kocsis, Levente; Schoenauer, Marc; Sebag, Michèle; Silver, David; Szepesvári, Csaba; Teytaud, Olivier (2012): The grand challenge of computer Go: Monte Carlo tree search and extensions. In: *Communications of the ACM* 55 (3), S. 106–113.

Gelly, Sylvain; Silver, David (2007): Combining online and offline knowledge in UCT. In: Proceedings of the 24th international conference on Machine learning. ACM, S. 273–280. Online verfügbar unter http://machinelearning.wustl.edu/mlpapers/paper_files/icml2007_GellyS07.pdf, zuletzt geprüft am 14.02.2014.

Gelly, Sylvain; Silver, David (2011): Monte-Carlo tree search and rapid action value estimation in computer Go. In: *Artificial Intelligence* 175 (11), S. 1856–1875. DOI: 10.1016/j.artint.2011.03.007.

Gelly, Sylvain; Wang, Yizao; others (2006): Exploration exploitation in Go: UCT for Monte-Carlo Go. In: NIPS: Neural Information Processing Systems Conference On-line trading of Exploration and Exploitation Workshop.

Genesereth, Michael; Love, Nathaniel; Pell, Barney (2005): General game playing: Overview of the AAAI competition. In: *AI magazine* 26 (2), S. 62–72.

Gherrity, Michael (1993): A game-learning machine. Dissertationsschrift. University of California, San Diego, CA.

Ghory, Imran (2004): Reinforcement learning in board games. In: *Department of Computer Science, University of Bristol, Tech. Rep*.

Glonnegger, Erwin (2009): Das Spiele-Buch. Brett- und Legespiele aus aller Welt. Herkunft, Regeln und Geschichte. 2. Aufl. Ravensburg: Ravensburger Buchverlag Otto Maier GmbH.

Goetsch, G.; Campbell, M. S. (1990): Experiments with the Null-Move Heuristic. In: T. Anthony Marsland und Jonathan Schaeffer (Hg.): Computers, Chess, and Cognition. New York, NY: Springer New York, S. 159–168.

Gregory, Jason (2009): Game engine architecture. Wellesley, Mass: A K Peters.

Günzel, Stephan (2010): Der reine Raum des Spiels. Zur Kritik des Magic Circle. In: Mathias Fuchs und Ernst Strouhal (Hg.): Das Spiel und seine Grenzen. Passagen des Spiels 2. Wien: Springer, S. 187–200. Online verfügbar unter http://www.stephan-guenzel.de/Texte/Guenzel_Spielraum.pdf, zuletzt geprüft am 14.06.2013.

Güth, Werner (1999): Spieltheorie und ökonomische (Bei)Spiele. 2., völlig neubearb. Aufl. Berlin [u.a.]: Springer (Springer-Lehrbuch).

Haugeland, John (1989): Artificial Intelligence. The Very Idea. 1. Aufl. Cambridge, Mass: MIT Press.

He, Yulin; Wang, Xizhao; Fu, Tingting (2013): A Combined Position Evaluation Function in Chinese Chess Computer Game. In: Transactions on Computational Science XVII: Springer, S. 31–50.

Hearn, Robert A. (2005): Amazons is PSPACE-complete. In: *CoRR* abs/cs/0502013.

Hebb, D. O. (2002): The organization of behavior. A neuropsychological theory. Mahwah, N.J: L. Erlbaum Associates.

Heinz, Ernst A. (1999): Adaptive null-move pruning. In: *ICCA JOURNAL* 22 (3), S. 123–132.

Heinz Nixdorf MuseumsForum: Frühe Automaten - Wunder der Technik. Schachtürke wieder zum Leben erweckt. Online verfügbar unter http://www.hnf.de/museum/die-mechanisierung-der-informationstechnik/fruehe-automaten-wunder-der-technik/schachtuerke-wieder-zum-leben-erweckt.html, zuletzt geprüft am 22.02.2013.

Helmbold, David P.; Parker-Wood, Aleatha (2009): All-Moves-As-First Heuristics in Monte-Carlo Go. In: IC-AI. Citeseer, S. 605–610.

Hindenburg, Carl Friedrich (1784): Ueber den Schachspieler des Herrn von Kempelen: Nebst einer Abbildung und Beschreibung seiner Sprachmaschine: J. G. Müller. Online verfügbar unter http://vd18.de/de-slub-vd18/content/titleinfo/4048456.

Hobbes, Thomas (1984): Leviathan, oder Stoff, Form und Gewalt eines kirchlichen und bürgerlichen Staates. 1. Aufl. Frankfurt am Main: Suhrkamp (Suhrkamp Taschenbuch Wissenschaft, 462).

Holler, Manfred J.; Illing, Gerhard (2009): Einführung in die Spieltheorie. 7. Aufl. Berlin, Heidelberg: Springer Verlag.

Horn, Christoph (Hg.) (2010): Gründe und Zwecke. Texte zur aktuellen Handlungstheorie. Orig.-Ausg., 1. Aufl. Berlin: Suhrkamp (Suhrkamp-Taschenbuch Wissenschaft, 1950).

Huizinga, Johan (2009): Homo ludens - Vom Ursprung der Kultur im Spiel. 21. Aufl. Reinbek bei Hamburg: Rowohlt.

IBM: Frequently Asked Questions: Deep Blue. Online verfügbar unter https://www.research.ibm.com/deepblue/meet/html/d.3.3a.html, zuletzt geprüft am 03.11.2013.

Igel, Christian; Hüsken, Michael (2000): Improving the Rprop learning algorithm. In: Proceedings of the second international ICSC symposium on neural computation (NC 2000). Citeseer, S. 115–121.

Juul, Jesper (2003): 255,168 ways of playing Tic Tac Toe. Online verfügbar unter http://www.jesperjuul.net/ludologist/255168-ways-of-playing-tic-tac-toe, zuletzt geprüft am 24.09.2013.

Juul, Jesper (2005): Half-real: Video Games Between Real Rules and Fictional Worlds. Cambridge, Massachusetts: MIT Press.

Juul, Jesper (2010): A casual revolution. Reinventing video games and their players. Cambridge, Mass: MIT Press.

Kantschick, Wolfgang (2006): Genetische Programmierung und Schach. Dissertationsschrift. Universität Dortmund, Dortmund. Fachbereich Informatik. Online verfügbar unter http://webdoc.sub.gwdg.de/ebook/dissts/Dortmund/Kantschik2006.pdf, zuletzt geprüft am 17.06.2014.

Kloetzer, Julien (2011): Monte-Carlo Opening Books for Amazons. In: H. Jaap Herik, Hiroyuki Iida und Aske Plaat (Hg.): Computers and Games, Bd. 6515: Springer Berlin Heidelberg (Lecture Notes in Computer Science), S. 124-135. Online verfügbar unter http://dx.doi.org/10.1007/978-3-642-17928-0_12.

Kloetzer, Julien; Iida, Hiroyuki; Bouzy, Bruno (2007): The Monte-Carlo Approach in Amazons. In: Proceedings of the Computer Games Workshop, S. 185–192.

Kloetzer, Julien; Iida, Hiroyuki; Bouzy, Bruno (2008): A comparative study of solvers in Amazons endgames. In: Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On. IEEE, S. 378–384.

Klüver, Christina; Klüver, Jürgen; Schmidt, Jörn (2012): Modellierung komplexer Prozesse durch naturanaloge Verfahren. Soft Computing und verwandte Techniken; mit 11 Tabellen und 25 QR-Codes. 2., erw. und aktualisierte Aufl. Wiesbaden: Springer Vieweg (Lehrbuch).

Knuth, Donald E.; Moore, Ronald W. (1975): An analysis of alpha-beta pruning. In: *Artificial Intelligence* 6 (4), S. 293–326. DOI: 10.1016/0004-3702(75)90019-3.

Kobti, Ziad; Sharma, Shiven (2007): A Multi-Agent Architecture for Game Playing. In: IEEE Symposium on Computational Intelligence and Games, 2007. CIG 2007, S. 276–281.

Kocsis, Levente; Szepesvári, Csaba (2006): Bandit based monte-carlo planning. In: Machine Learning: ECML 2006: Springer, S. 282–293.

Koller, Daphne; Pfeffer, Avi (1997): Representations and solutions for game-theoretic problems. In: *Artificial Intelligence* 94 (1), S. 167–215.

Korf, Richard E. (2010): Artificial intelligence search algorithms. Algorithms and theory of computation handbook. In: Mikhail Atallah und Marina Blanton (Hg.): Algorithms and theory of computation handbook: special topics and techniques. 2. Aufl.: Chapman & Hall/CRC, S. 22. Online verfügbar unter http://dl.acm.org/citation.cfm?id=1882723.1882745.

Kroll, Andreas (2013): Computational Intelligence. Eine Einführung in Probleme, Methoden und technische Anwendungen. München: Oldenbourg.

Kruse, Rudolf; Borgelt, Christian; Klawonn, Frank; Moewes, Christian; Ruß, Georg; Steinbrecher, Matthias (2011): Computational Intelligence: Eine methodische Einführung in Künstliche Neuronale Netze, Evolutionäre Algorithmen, Fuzzy-Systeme und Bayes-Netze. Wiesbaden: Vieweg +Teubner Verlag.

Kulick, Johannes; Block, Marco; Rojas, Raúl (2009): General Game Playing mit stochastischen Spielen. Online verfügbar unter http://gameai.mi.fu-berlin.de/scripts/TR_09_GDL.pdf, zuletzt geprüft am 20.08.2014.

Kurzweil, Ray (2013): How to create a mind. The secret of human thought revealed. New York, N.Y.: Penguin Books.

Laux, Helmut; Gillenkirch, Robert M.; Schenk-Mathes, Heike Y. (2012): Entscheidungstheorie. 8. Aufl. Berlin, Heidelberg: Springer Gabler (Springer-Lehrbuch).

Levinson, Robert; Snyder, Richard (1991): Adaptive pattern-oriented chess. In: Proceedings of the ninth National conference on Artificial intelligence-Volume 2. AAAI Press, S. 601–605.

Levinson, Robert A. (1994): MORPH II: A universal agent: Progress report and proposal: Citeseer.

Lorentz, Richard J. (2008): Amazons discover monte-carlo. In: Computers and Games: Springer, S. 13–24.

Love, Nathaniel; Hinrichs, Timothy; Haley, David; Schkufza, Eric; Genesereth, Michael (2008): General Game Playing: General Game Playing: Game Description Language Specification. Online verfügbar unter http://logic.stanford.edu/classes/cs227/2013/readings/gdl_spec.pdf, zuletzt geprüft am 20.08.2014.

Luce, R. Duncan; Raiffa, Howard (1989): Games and decisions. Introduction and critical survey. Mineola: Dover Publications.

Lynch, M.; Griffith, N. (1997): Neurodraughts: the role of representation, search, training regime and architecture in a TD draughts player. In: Eighth Ireland Conference on Artificial Intelligence. Citeseer, S. 64–72.

Macal, Charles M.; North, Michael J. (2009): Agent-based Modeling and Simulation. In: Ann Dunkin, Ricki G. Ingalls, Enver Yücesan, Manuel D. Rossetti, Ray Hill und Björn Johansson (Hg.): Proceedings of the 2009 Winter Simulation Conference, WSC 2009, Hilton Austin Hotel, Austin, TX, USA, December 13-16, 2009: WSC, S. 86–98.

Mańdziuk, Jacek (2010): Knowledge-free and Learning-based Methods in Intelligent Game Playing. Berlin, Heidelberg: Springer Verlag, zuletzt geprüft am 03.09.2012.

Mańdziuk, Jacek; Ong, Yew Soon; Waledzik, Karol (2013): Multi-game playing—A challenge for computational intelligence. In: Computational Intelligence for Human-like Intelligence (CIHLI), 2013 IEEE Symposium on. IEEE, S. 17–24.

Marcolino, Leandro Soriano (2011): Multi-Agent Monte Carlo Go. School of Systems Information Science at Future University Hakodate, Hakodate. Online verfügbar unter http://teamcore.usc.edu/people/sorianom/thesis.pdf, zuletzt geprüft am 05.03.2014.

Marcolino, Leandro Soriano; Matsubara, Hitoshi (2011): Multi-Agent Monte Carlo Go. In: The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 1. International Foundation for Autonomous Agents and Multiagent Systems, S. 21–28. Online verfügbar unter http://teamcore.usc.edu/people/sorianom/AAMAS2011.pdf, zuletzt geprüft am 05.03.2014.

Méhat, Jean; Cazenave, Tristan (2010): Ary, a general game playing program. In: Board Games Studies Colloquium. Online verfügbar unter http://www.lamsade.dauphine.fr/~Cazenave/papers/ggp-bga.pdf, zuletzt geprüft am 22.08.2014.

Metropolis, Nicholas; Ulam, Stanislaw (1949): The monte carlo method. In: *Journal of the American statistical association* 44 (247), S. 335–341.

Millington, Ian; Funge, John David (2009): Artificial Intelligence for Games. 2. Aufl. Burlington, MA: Morgan Kaufmann, zuletzt geprüft am 03.09.2012.

Mitchell, Melanie (2009): Complexity. A guided tour. Oxford [England], New York: Oxford University Press.

Murakami, Haruki (2007): Hard Boiled Wonderland und das Ende der Welt. München: btb Verlag.

Nash, John F. (1950): Equilibrium Points in n-Person Games. In: *Proceedings of the National Academy of Sciences of the United States of America* 36 (1), S. 48–49.

Neitzel, Britta: Gespielte Geschichten. Struktur- und prozessanalytische Untersuchungen der Narrativität von Videospielen. Dissertationsschrift. Bauhaus-Universität Weimar, Weimar. Fakultät Medien. Online verfügbar unter http://e-pub.uni-weimar.de/volltexte/2004/72/pdf/Neitzel.pdf, zuletzt geprüft am 09.10.2013.

Neumann, John von; Morgenstern, Oskar (2004): Theory of Games and Economic Behavior. Sixtieth-Anniversary Edition. Princeton, New Jersey: Princeton University Press.

Newell, Allen; Shaw, John Calman; Simon, Herbert Alexander (1958): Chess-playing programs and the problem of complexity. In: *IBM Journal of Research and Development* 2 (4), S. 320–335.

Nilsson, Nils J. (2010): The quest for artificial intelligence. A history of ideas and achievements. Cambridge,, New York: Cambridge University Press.

Omid, David-Tabibi; Netanyahu, Nathan S. (2002): Verified Null-Move Pruning. In: *ICGA Journal* 25 (3), S. 153–161.

Padgham, Lin; Winikoff, Michael (2004): Developing autonomous agent systems. A practical guide to designing, building, implementing and testing agent systems. Chichester: Wiley.

Papageorgiou, Markos; Leibold, Marion; Buss, Martin (2012): Optimierung. Statische, dynamische, stochastische Verfahren. 3., neu bearb. und erg. Aufl. Berlin: Springer (Lehrbuch).

Papahristou, Nikolaos; Refanidis, Ioannis (2012): Improving Temporal Difference Learning Performance in Backgammon Variants. In: H.Jaap Herik und Aske Plaat (Hg.): Advances in Computer Games, Bd. 7168: Springer Berlin Heidelberg (Lecture Notes in Computer Science), S. 134-145. Online verfügbar unter http://dx.doi.org/10.1007/978-3-642-31866-5_12.

Papahristou, Nikolaos; Refanidis, Ioannis (2013): AnyGammon: Playing Backgammon Variants Using Any Board Size. Online verfügbar unter http://www.fdg2013.org/program/festival/anygammon.pdf, zuletzt geprüft am 06.10.2013.

Patist, Jan Peter; Wiering, M. A. (2004): Learning to play draughts using temporal difference learning with neural networks and databases. In: *Proceedings of the 13th Annual Machine Learning Conference of Belgium and the Netherlands (BeneLearn'04)*.

Pearl, Judea (1980): SCOUT: A Simple Game-Searching Algorithm with Proven Optimal Properties. In: AAAI, S. 143–145.

Pell, Barney (1993): Strategy generation and evaluation for meta-game playing. Dissertationsschrift. University of Cambridge, Cambridge.

Pell, Barney (1996): A strategic metagame player for general chess-like games. In: *Computational Intelligence* 12 (1), S. 177–198.

Pfeifer, Rolf; Bongard, Josh; Grand, Simon (2007): How the body shapes the way we think. A new view of intelligence. Cambridge, Mass: MIT Press.

Pfeifer, Rolf; Scheier, Christian (2001): Understanding intelligence. 1st MIT Press pbk. ed. Cambridge, Mass: MIT Press.

Pitrat, Jacques (1968): Realization of a general game-playing program. In: IFIP congress (2), S. 1570–1574.

Pitrat, Jacques (1995): AI systems are dumb because AI researchers are too clever. In: *ACM Comput. Surv.* 27 (3), S. 349–350. DOI: 10.1145/212094.212124.

Plaat, Aske; Schaeffer, Jonathan; Pijls, Wim; Bruin, Arie de (1995): Best-first fixed-depth game-tree search in practice. In: Proceedings of the 14th international joint conference on Artificial intelligence-Volume 1. Morgan Kaufmann Publishers Inc, S. 273–279.

Pollack, Jordan B.; Blair, Alan D. (1997): Why did TD-Gammon Work? In: *Advances in neural information processing systems*, S. 10–16.

Prechtl, Peter (2008): Metzler Lexikon Philosophie. Begriffe und Definitionen. Hg. v. Franz P. Burkard. Stuttgart: Verlag J.B. Metzler.

Ramanujan, Raghuram; Sabharwal, Ashish; Selman, Bart (2012): Understanding Sampling Style Adversarial Search Methods. Online verfügbar unter http://arxiv.org/pdf/1203.4011.

Rapaport, William J. (1988): To think or not to think. In: *Noûs* 22 (4), S. 585–609.

Rapaport, William J. (2000): How to Pass a Turing Test: Syntactic Semantics, Natural-Language Understanding, and First-Person Cognition. In: *Journal of Logic, Language, and Information*, S. 467–490.

Reinefeld, Alexander (1983): An improvement of the Scout tree-search algorithm. In: *ICCA JOURNAL* 6 (4), S. 4–14.

Rheinwald, Rosemarie (1991): Koennen Maschinen eine Sprache sprechen? Sind Computerprogramme syntaktisch oder semantisch? In: *Kognitionswissenschaft* 2 (1), S. 37–49.

Rich, Elaine; Knight, Kevin (1991): Artificial intelligence. 2. Aufl. New York: McGraw-Hill.

Riedmiller, Martin; Braun, Heinrich (1992): RPROP-A fast adaptive learning algorithm. In: Proc. of ISCIS VII), Universitat. Citeseer.

Riedmiller, Martin; Braun, Heinrich (1993): A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In: Neural Networks, 1993., IEEE International Conference on. IEEE, S. 586–591.

Rimmel, Arpad; Teytaud, Olivier; Lee, Chang-Shing; Yen, Shi-Jim; Wang, Mei-Hui; Tsai, Shang-Rong (2010): Current Frontiers in Computer Go. In: *IEEE Trans.*

Comput. Intell. AI Games 2 (4), S. 229–238. DOI: 10.1109/TCIAIG.2010.2098876.

Robbins, Herbert (1952): Some aspects of the sequential design of experiments. In: *Bulletin of the American Mathematical Society* 58 (5), S. 527–535.

Roth, Gerhard (2009): Aus Sicht des Gehirns. Orig.-Ausg., vollst. überarb. Neuaufl. Frankfurt, M: Suhrkamp (Suhrkamp-Taschenbuch Wissenschaft, 1915).

Rumelhart, David E.; Hinton, Geoffrey E.; Williams, Ronald J. (1985): Learning internal representations by error propagation. La Jolla, Calif: Institute for Cognitive Science, University of California, San Diego (ICS report, 8506).

Rupp, Chris; Queins, Stefan; Zengler, Barbara (2007): UML 2 glasklar. Praxiswissen für die UML-Modellierung. 3., aktualisierte Aufl. München, Wien: Hanser.

Russell, Stuart J.; Norvig, Peter (2012): Künstliche Intelligenz. Ein moderner Ansatz. 3. Aufl. München: Pearson.

Šalamon, Tomáš (2011): Design of agent-based models. Developing computer simulations for a better understanding of social processes. Řepín-Živonín: Tomáš Bruckner.

Salen, Katie; Zimmerman, Eric (2004): Rules of Play. Game Design Fundamentals. Cambridge, Massachusetts: MIT Press.

Salen, Katie; Zimmerman, Eric (2006): The Game Design Reader. A Rules of Play Anthology. Cambridge, Massachusetts: MIT Press.

Samuel, Arthur L. (1963): Some Studies in Machine Learning Using the Game of Checkers. In: Edward A. Feigenbaum und Julian Feldman (Hg.): Computers and Thought. New York: McGraw-Hill, S. 71–105.

Schädler, Ulrich (Hg.) (2007): Spiele der Menschheit. 5000 Jahre Kulturgeschichte der Gesellschaftsspiele ; [aus Anlass des 20jährigen Bestehens des Schweizerischen Spielmuseums (1987 - 2007)]. Darmstadt: Wiss. Buchges (Veröffentlichungen des Schweizerischen Spielmuseums, La Tour-de-Peilz).

Schaeffer, Jonathan (2009): One Jump Ahead. Computer Perfection at Checkers. 2. Aufl. New York: Springer.

Schaeffer, Jonathan; Björnsson, Yngvi; Burch, Neil; Kishimoto, Akihiro; Lake, R.; Lu, P. et al. (2005): Solving checkers. In: Proceedings of the 19th international joint conference on Artificial intelligence. Morgan Kaufmann Publishers Inc, S. 292–297.

Schaeffer, Jonathan; Björnsson, Yngvi; Burch, Neil; Lake, Robert; Lu, Paul; Sutphen, Steve (2004): Building the checkers 10-piece endgame databases. In: Advances in Computer Games: Springer, S. 193–210.

Schaeffer, Jonathan; Burch N; Björnsson Y; Kishimoto A; Müller M; Lake R et al. (2007): Checkers is solved. In: *Science* 317 (5844), S. 1518–1522.

Scheuerl, Hans (1990): Das Spiel. Band 1: Untersuchungen über sein Wesen, seine pädagogischen Möglichkeiten und Grenzen. 11. Aufl. Weinheim und Basel: Beltz Verlag.

Scheuerl, Hans (1991): Das Spiel. Band 2: Theorien des Spiels. 11. Aufl. Weinheim und Basel: Beltz Verlag.

Schiffel, Stephan; Thielscher, Michael (2007): Fluxplayer: A successful general game player. In: Proceedings of the National Conference on Artificial Intelligence, Bd. 22. Menlo Park, CA Cambridge, MA London AAAI Press MIT Press 1999, S. 1191–1196. Online verfügbar unter http://aaaipress.org/Papers/AAAI/2007/AAAI07-189.pdf, zuletzt geprüft am 21.08.2014.

Schiffel, Stephan; Thielscher, Michael (2011): Reasoning About General Games Described in GDL-II. Online verfügbar unter http://www.aaai.org/ocs/index.php/AAAI/AAAI11/paper/view/3557, zuletzt geprüft am 20.08.2014.

Schiffel, Stephan; Thielscher, Michael (2014): Representing and Reasoning About the Rules of General Games With Imperfect Information. In: *Journal of Artificial Intelligence Research* 49, S. 171–206. Online verfügbar unter

http://cgi.cse.unsw.edu.au/~mit/Papers/JAIR14.pdf, zuletzt geprüft am 19.08.2014.

Schmid, Hans Bernhard; Schweikard, David P. (2009): Kollektive Intentionalität. Eine Debatte über die Grundlagen des Sozialen. Frankfurt am Main: Suhrkamp (Suhrkamp-Taschenbuch Wissenschaft, 1898).

Searle, John R. (1990): Ist der menschliche Geist ein Computerprogramm. In: *Spektrum der Wissenschaft* 3 (1990), S. 40–47.

Segal, Richard B. (2011): On the scalability of parallel UCT. In: H. Jaap Herik, Hiroyuki Iida und Aske Plaat (Hg.): Computers and Games: Springer Berlin Heidelberg (Lecture Notes in Computer Science), S. 36–47.

Shankar, Aditya Ravi (2012): Pro HTML5 Games. [New York]: Apress; Distributed to the book trade worldwide by Springer Science+Business Media New York (The expert's voice in web development).

Shannon, Claude Elwood (1950): XXII. Programming a computer for playing chess. In: *Philosophical magazine* 41 (314), S. 256–275.

Shoham, Yoav; Leyton-Brown, Kevin (2009): Multiagent systems. Algorithmic, game-theoretic, and logical foundations. Cambridge, New York: Cambridge University Press.

Shredder Computer Schach: UCI-Protokoll. Online verfügbar unter http://www.shredderchess.de/schach-info/features/uci-universal-chess-interface.html, zuletzt geprüft am 07.11.2013.

Sieg, Gernot (2010): Spieltheorie. 3. Aufl. München: Oldenbourg Wissenschaftsverlag.

Silver, Nate (2012): The Signal and the Noise. The Art and Science of Prediction. London: Penguin Group.

Stampfl, Nora S. (2012): Die verspielte Gesellschaft. Gamification oder Leben im Zeitalter des Computerspiels. 1. Aufl. Hannover: Heise (Telepolis).

Standage, Tom (2005): Der Türke. Die Geschichte des ersten Schachautomaten und seiner abenteuerlichen Reise um die Welt. Berlin: Berliner Taschenbuch-Verlag.

Strupp, Christoph (2000): Johan Huizinga: Geschichtswissenschaft als Kulturgeschichte. Göttingen: Vandenhoeck & Ruprecht.

Suits, Bernard (2005): The grasshopper: Games, Life and Utopia. Peterborough, Ontario, Canada: Broadview Press.

Sutton, Richard S. (1988): Learning to predict by the methods of temporal differences. In: *Machine learning* 3 (1), S. 9–44.

Sutton, Richard S.; Barto, Andrew G. (1998): Reinforcement learning. An introduction. Cambridge, Mass. [u.a.]: MIT Press (Adaptive computation and machine learning).

Szepesvári, Csaba (2013): Algorithms for Reinforcement Learning. Online verfügbar unter http://www.ualberta.ca/~szepesva/papers/RLAlgsInMDPs.pdf, zuletzt geprüft am 06.12.2013.

Szita, István (2012): Reinforcement Learning in Games. In: Marco Wiering und Martijn van Otterlo (Hg.): Reinforcement Learning, Bd. 12: Springer Berlin Heidelberg (Adaptation, Learning, and Optimization), S. 539-577. Online verfügbar unter http://dx.doi.org/10.1007/978-3-642-27645-3_17.

Tesauro, Gerald (1989): Neurogammon wins computer Olympiad. In: *Neural Computation* 1 (3), S. 321–323.

Tesauro, Gerald (1990): Neurogammon: A neural-network backgammon program. In: Neural Networks, 1990., 1990 IJCNN International Joint Conference on. IEEE, S. 33–39.

Tesauro, Gerald (1994): TD-Gammon, a self-teaching backgammon program, achieves master-level play. In: *Neural Computation* 6 (2), S. 215–219.

Tesauro, Gerald (1995): Temporal difference learning and TD-Gammon. In: *Communications of the ACM* 38 (3), S. 58–68.

Tesauro, Gerald (1998): Comments on “co-evolution in the successful learning of backgammon strategy”. In: *Machine learning* 32 (3), S. 241–243.

Tesauro, Gerald (2002): Programming backgammon using self-teaching neural nets. In: *Artificial Intelligence* 134 (1), S. 181–199.

Thielscher, Michael (2010): A General Game Description Language for Incomplete Information Games. In: AAAI, Bd. 10, S. 994–999. Online verfügbar unter http://www.cs.huji.ac.il/~jeff/aaai10/02/AAAI10-175.pdf, zuletzt geprüft am 20.08.2014.

Thielscher, Michael (2011a): General game playing in AI research and education. In: Proceedings of the 34th Annual German conference on Advances in artificial intelligence. Springer-Verlag, S. 26–37. Online verfügbar unter http://www.cse.unsw.edu.au/~mit/Papers/KI11.pdf, zuletzt geprüft am 19.08.2014.

Thielscher, Michael (2011b): The general game playing description language is universal. In: IJCAI Proceedings-International Joint Conference on Artificial Intelligence, Bd. 22, S. 1107. Online verfügbar unter http://www.cse.unsw.edu.au/~mit/Papers/IJCAI11.pdf, zuletzt geprüft am 19.08.2014.

Thimm, Caja; Wosnitza, Lukas (2010): Das Spiel – analog und digital. In: Caja Thimm (Hg.): Das Spiel: Muster und Metapher der Mediengesellschaft. Wiesbaden: VS Verlag für Sozialwissenschaften, S. 33–54.

Thompson, Clive (2007): Death of Checkers, The. In: *The New York Times* 2007, 09.12.2007. Online verfügbar unter http://www.nytimes.com/2007/12/09/magazine/09_15_checkers.html, zuletzt geprüft am 09.10.2013.

Thompson, Richard F. (2010): Das Gehirn. Von der Nervenzelle zur Verhaltenssteuerung. Nachdr. der 3. Aufl. 2001. Heidelberg: Spektrum Akad. Verl.

Thrun, Sebastian (1995): Learning to play the game of chess. In: *Advances in neural information processing systems* 7.

Turing, Alan M. (1950): Computing machinery and intelligence. In: *Mind* 59 (236), S. 433–460.

van den Herik, H. Jaap; Uiterwijk, Jos W.H.M.; van Rijswijck, Jack (2002): Games solved: Now and in the future. In: *Artificial Intelligence* 134 (1), S. 277–311.

van Lishout, François; Chaslot, Guillaume Maurice Jean-Bernard; Uiterwijk, Jos W. H. M. (2007): Monte-Carlo Tree Search in Backgammon. In: Proc. Comput. Games Workshop. Amsterdam, Netherlands, S. 175–184. Online verfügbar unter http://orbi.ulg.ac.be/bitstream/2268/28469/1/vanlishout_backgammon.pdf, zuletzt geprüft am 02.04.2014.

Wang, Yizao; Gelly, S. (2007): Modifications of UCT and sequence-like simulations for Monte-Carlo Go. In: Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on. IEEE, S. 175–182.

Watkins, Christopher (1989): Learning from delayed rewards. University of Cambridge. Online verfügbar unter http://www.cs.rhul.ac.uk/home/chrisw/new_thesis.pdf, zuletzt geprüft am 11.02.2014.

Watkins, Christopher (1992): Q-learning. In: *Machine learning* 8 (3-4), S. 279–292.

Waugh, Sam; Adams, Anthony (1997): A practical comparison between quickprop and backpropagation. In: The Eighth Australian Conference on Neural Networks. Citeseer, S. 143–147.

Weicker, Karsten; Weicker, Nicole (2013): Algorithmen und Datenstrukturen. Wiesbaden: Springer Vieweg (SpringerLink : Bücher).

Weiss, Gerhard (Hg.) (2013): Multiagent Systems: MIT Press.

Wessler, Markus (2012): Entscheidungstheorie. Von der klassischen Spieltheorie zur Anwendung kooperativer Konzepte. Wiesbaden: Gabler Verlag; Imprint: Gabler Verlag (SpringerLink : Bücher).

Widrow, B.; Hoff, M. E. (1960): Adaptive Switching Circuits. In: IRE WESCON Convention Record, S. 96–104.

Wiering, Marco; van Otterlo, Martijn (2012): Reinforcement learning and markov decision processes. In: Marco Wiering und Martijn van Otterlo (Hg.): Reinforcement Learning: Springer Berlin Heidelberg (Adaptation, Learning, and Optimization), S. 3–42.

Wiering, Marco A. (2010): Self-Play and Using an Expert to Learn to Play Backgammon with Temporal Difference Learning. In: *JILSA* 2 (2), S. 57–68.

Winands, Mark H. M.; Björnsson, Yngvi; Saito, Jahn-Takeshi (2008): Monte-carlo tree search solver. In: Computers and Games: Springer, S. 25–36. Online verfügbar unter https://dke.maastrichtuniversity.nl/m.winands/documents/uctloa.pdf, zuletzt geprüft am 08.11.2013.

Windisch, Karl Gottlieb von (1783): Briefe über den Schachspieler des Herrn von Kempelen: A. Löwe. Online verfügbar unter http://vd18.de/de-bsb-vd18/content/titleinfo/26298828.

Wittgenstein, Ludwig (2006): Werkausgabe Band 1. Tractatus logico-philosophicus; Tagebücher 1914-1916; Philosophische Untersuchungen. Frankfurt am Main: Suhrkamp.

Wooldridge, Michael J. (2009): An introduction to multiagent systems. 2. Aufl. Chichester, U.K: John Wiley & Sons.

Zakas, Nicholas C. (2010): High performance JavaScript. Description based on print version record. - "Build faster web application interfaces"--Cover. - Includes index. 1st ed. Sebastopol, Calif: O'Reilly.

Zimmer, Dieter E. (1990): Deus in machina. oder: Wieviel Geist steckt im Computer? In: *Die Zeit* 1990, 25.05.1990 (Nr. 22). Online verfügbar unter http://www.zeit.de/1990/22/deus-in-machina/seite-1, zuletzt geprüft am 13.11.2013.

Zobrist, Albert L. (1969): A model of visual organization for the game of Go. In: Proceedings of the May 14-16, 1969, spring joint computer conference. ACM, S. 103–112.

9 Abbildungsverzeichnis

Abbildung 1: Screenshot eines laufzeitgenerierten Monte Carlo Spielbaumes, wie er zur Entscheidungsfindung eines artifiziellen Spielers im Tic Tac Toe Spiel aufgebaut und verwendet wurde. 16

Abbildung 2: Im Jahr 2004 der Öffentlichkeit vorgestellter Nachbau des Automaten Wolfgang von Kempelens. Auf der linken Seite in geschlossenem, auf der rechten Seite in geöffnetem Zustand. Bildverwendung mit freundlicher Genehmigung des Heinz Nixdorf MuseumsForum. 23

Abbildung 3: Versuchsaufbau des Turing-Tests: Der Fragesteller C befragt die Spielpartner A und B – und versucht herauszufinden, welchen Geschlechts A ist. 30

Abbildung 4: Modell eines basalen Agenten: Über seine Sensoren (beispielsweise eine Kamera) nimmt der Agent seine Umwelt wahr und wirkt auf sie ein mit Hilfe seiner Aktuatoren, die auch als *Effektoren* bezeichnet werden. Das Fragezeichen im Inneren des Agenten symbolisiert interne Prozesse, die Agentenaktionen aus sensorisch gewonnenen Daten generieren. 40

Abbildung 5: Skizze eines allgemeinen lernenden Agenten nach Russell und Norvig. 49

Abbildung 6: Schachmatt in zwei Zügen. 51

Abbildung 7: Forschungsgebiete der Computational Intelligence nach Kroll. 56

Abbildung 8: Zugregel *kurze Rochade* im Schachspiel auf der linken Seite vor und auf der rechten Seite nach Ausführung des Zuges. 69

Abbildung 9: Informelle Darstellung des Spielbaumes.....88

Abbildung 10: Spielbaum eines abstrakten Spieles mit unvollkommener Information.....90

Abbildung 11: Spielbaum des auf eine Spielwelt von 4x4 Felder reduzierten Schachspieles. Um die unterschiedlichen Zugwahlen zu veranschaulichen, finden sich die Knoten des Spielbaumes ergänzt um Bilder der Spielkonfigurationen. 91

Abbildung 12: Spielpartie *Schachmatt in zwei Zügen* mit eingezeichneten Zugmöglichkeiten von Weiß. 97

Abbildung 13: Binärbaum zur Illustration der Breiten- und Tiefensuche. 98
Abbildung 14: Binärbaum zur Verdeutlichung der iterativ vertiefenden Tiefensuche. 100
Abbildung 15: Partieller Spielbaum des Tic Tac Toe Spieles, wiedergegeben nach Russell und Norvig 2012, S. 208. 102
Abbildung 16: Spielbaum mit zwei Entscheidungsschichten. Die als Dreieck visualisierten Knoten bezeichnen Knoten, an denen der maximierende Spieler MAX am Zuge ist; an rechteckigen Knoten ist der Spieler MIN am Zuge. Die an den Blättern $x4$ bis $x12$ notierten ganzzahligen Werte repräsentieren die Auszahlungen am Ende des Spieles. 103
Abbildung 17: Visualisierung des Minimax-Verfahrens.....104
Abbildung 18: Lösung einer Partie des Tic Tac Toe Spieles anhand des Minimax-Algorithmus‘. 105
Abbildung 19: Spielbaum zur Verdeutlichung der Alpha-Beta Kürzung.....109
Abbildung 20: Erster Schritt des Verfahrens der Alpha-Beta Kürzung. 109
Abbildung 21: Zweiter Schritt des Verfahrens der Alpha-Beta Kürzung. 110
Abbildung 22: Alpha-Beta Kürzung. Grau hinterlegte Knoten werden in der Suche nicht berücksichtigt. 111
Abbildung 23: Vollständig analysierter Spielbaum im Rahmen des Verfahrens der Alpha-Beta Kürzung. Von der Betrachtung ausgeschlossene Knoten sind grau hinterlegt. 112
Abbildung 24: Effizienz der Alpha-Beta Kürzung nach Vertauschung der Knoten $x5$ und $x6$ und ihrer Kindknoten. 118
Abbildung 25: Beispielvisualisierung des flachen Monte Carlo Verfahrens zur Identifikation guter Spielzüge. Die Knotenbezeichnungen in der Form n/m informieren über die Anzahl der gewonnenen Spiele (n) aus der Menge der simulierten Spiele (m). 132
Abbildung 26: Annäherung der Knotenwerte eines Minimax Spielbaumes unter Verwendung der Monte Carlo Spielbaumsuche. Quelle: Gelly et al. 2012, S. 109. 136

Abbildung 27: Schematische Darstellung der Monte Carlo Spielbaumsuche mit ihren vier Phasen der Selektion, Expansion, Simulation und Rückpropagation. 137
Abbildung 28: Beispielgrafik eines durch das Verfahren der Monte Carlo Spielbaumsuche generierten asymmetrischen Spielbaumes. Quelle: Coquelin und Munos 2007, S. 74. 145
Abbildung 29: Anwendungsbeispiel des RAVE-Algorithmus'. Quelle: Gelly et al. 2012, S. 111. 147
Abbildung 30: Zwei Verfahren der Parallelisierung der Monte Carlo Spielbaumsuche. Die gestrichelten Linien mit Pfeilabschluss signifizieren individuelle Threads für die Simulationsphase des Verfahrens. Quelle: Chaslot et al. 2008c, S. 63.....149
Abbildung 31: Spielzustand des Go-Spieles. Eigene Aufnahme. 151
Abbildung 32: Einfaches Modell eines künstlichen Neurons i. 159
Abbildung 33: Darstellung der Schwellenwertfunktion (links) und Sigmoidfunktion (rechts) mit $T \rightarrow 0$. 160
Abbildung 34: Ein vorwärtsgerichtetes künstliches neuronales Netz mit einer Eingabe-, einer versteckten- und einer Ausgabeschicht. 161
Abbildung 35: Verknüpfungsmatrix eines künstlichen neuronalen Netzes. Zur Veranschaulichung der Matrix sind die Gewichtungen $w42$ und $w64$ hervorgehoben. 162
Abbildung 36: Arten der Rückkopplung in einem Künstlichen neuronalen Netz. 163
Abbildung 37: Veranschaulichung des Q-Lernverfahrens: Der Agent befindet sich in Raum A und muss den Ausgang aus dem Gebäude finden.....184
Abbildung 38: Belohnungsmatrix für unterschiedliche Umweltzustände und Zustandsübergänge. 185
Abbildung 39: Initiale Q-Wertematrix des Agenten. 185
Abbildung 40: Belohnungsmatrix mit hervorgehobenem Umweltfeedback für den Wechsel des Agenten von Raum C nach Raum D oder F. 186
Abbildung 41: Q-Wertematrix des Agenten nach einer Erfahrungsepisode.....187
42: Q-Wertematrix des Agenten nach zwei Erfahrungsepisoden. 187

Abbildung 43: Q-Wertematrix des Agenten nach drei Erfahrungsepisoden.....188
Abbildung 44: Screenshot des *Zillions of Games* Startbildschirms.....198
Abbildung 45: Screenshot einer Tic Tac Toe Partie in *Zillions of Games*.....200
Abbildung 46: Spiel-Manager Infrastruktur im Rahmen der *General Game Playing Competition*, wiedergegeben nach Genesereth et al. 2005, S. 70.....211
Abbildung 47: Softwarearchitektur von *Cadiaplayer*. Wiedergeben nach Björnsson und Finnsson 2009, S. 3.....214
Abbildung 48: Screenshot des Tic Tac Toe Spieles, umgesetzt mit *SpoookyJS*. ..222
Abbildung 49: Screenshot des Agentenensembles für das Tic Tac Toe Spiel.....223
Abbildung 50: Screenshot der Spieleseite mit eingeblendeten Statistiken. 224
Abbildung 51: Screenshot der laufzeitgenerierten Monte Carlo Spielbaumsuchläufe der einzelnen Agenten des Agentenensembles. 225
Abbildung 52: Systemskizze des JavaScript-Frameworks *SpoookyJS*. 227
Abbildung 53: Klassenausschnitt *Spoooky.Models*.....228
Abbildung 54: Klassenausschnitt *Spoooky.Game*.....230
Abbildung 55: Klassendiagrammausschnitt *Spoooky.Game* und assoziierte Klassen. 231
Abbildung 56: Zugmöglichkeiten in linearen (links), netzartigen (mitte) und zweidimensionalen (rechts) Spielwelten. 232
Abbildung 57: Klassenausschnitt *Spoooky.GridWelt*.....233
Abbildung 58: Ordnerstruktur von *SpoookyJS* (Screenshot aus der Entwicklungsumgebung *PHPStorm*).....239
Abbildung 59: Initialkonfiguration des Schachbrettes 242
Abbildung 60: Zugmöglichkeiten des Königs im Schachspiel. 243
Abbildung 61: Zugmöglichkeiten der kurzen Rochade (oben) und langen Rochade (unten) im Schachspiel. Ausschnitthaft dargestellt sind die Spielkonfiguration vor (linke Seite) und nach Ausführung der Rochaden. 243
Abbildung 62: Einfach- und Doppelschritt des Bauern.....245
Abbildung 63: Schlagende Züge des Bauern.....245
Abbildung 64: Bauernzug *Schlagen en passant*.....246
Abbildung 65: Bildschirmausschnitt des mit *SpoookyJS* implementierten Schachspieles im Webbrowser *Firefox*.....265

Abbildung 66: Darstellung der Backgammonspielwelt mit eingezeichneten Zugrichtungen der beiden Spieler Weiß und Schwarz. 267
Abbildung 67: Klassendiagramm *Spoooky.MetaAgent* und *Spoooky.Agent*: Eine beliebige Anzahl von mehr als einem Agenten (*Spoooky.Agent*, Häufigkeit 1..*) analysieren Zugmöglichkeiten für einen Meta Agenten (*Spoooky.MetaAgent*).....275
Abbildung 68: Prozess der Entscheidungsfindung auf Grundlage des Agentenensembles. Der Meta Agent nimmt eine Problemstellung wahr – im vorliegenden Falle einen Zustand des Schachspieles – und beauftragt sein Agentenensemble damit, Lösungen für das an ihn gestellte Entscheidungsproblem zu erarbeiten. Nach Analyse des Problems schlägt das Agentenensemble dem Meta Agenten Lösungsmöglichkeiten für das gestellte Problem vor. 276
Abbildung 69: Initialzustand des 5x5 *Tic Tac Toe* Spieles zur Verdeutlichung der Agentenfokussierung. 278
Abbildung 70: Zu analysierende Zugmöglichkeiten eines Agenten mit der Fokussierungsweise „ALL“. 278
Abbildung 71: Analysefokus „SECOND HALF OF POSSIBLE MOVES“ (links) und „FIRST HALF OF POSSIBLE MOVES“. 279
Abbildung 72: Analysefokus „MOVES NEAR OPPONENT FIELDS“ (links) und „MOVES NEAR OPPONENT OR OWN FIELDS“. 279
Abbildung 73: Dynamisch generierter Spielbaum des Agenten mit einem UCT-Konstantenwert von 1.0 zur Untersuchung der Zugmöglichkeiten im Initialzustand des Tic Tac Toe Spieles. 292
Abbildung 74: Dynamisch generierter Spielbaum des Agenten mit einem UCT-Konstantenwert von 0.5 zur Untersuchung der Zugmöglichkeiten im Initialzustand des Tic Tac Toe Spieles. 293
Abbildung 75: Dynamisch generierter Spielbaum des Agenten mit einem UCT-Konstantenwert von 0.0 zur Untersuchung der Zugmöglichkeiten im Initialzustand des Tic Tac Toe Spieles (verkürzte Darstellung). 294
Abbildung 76: Laufzeitgenerierter Spielbaum des Agenten mit dem Analysefokus "FIRST HALF OF POSSIBLE MOVES" und einem UCT-Konstantenwert von 0.0

zur Untersuchung der Zugmöglichkeiten im Initialzustand des Dame-Spieles. 297
Abbildung 77: Laufzeitgenerierter Spielbaum des Agenten mit dem Analysefokus "FIRST HALF OF POSSIBLE MOVES" und einem UCT-Konstantenwert von 1.0 zur Untersuchung der Zugmöglichkeiten im Initialzustand des Dame-Spieles. 298
Abbildung 78: Bedrohungssituation im Gomoku-Spiel. 299
Abbildung 79: Laufzeitgenerierter Spielbaum des Agenten mit dem Analysefokus "MOVES NEAR OPPONENT FIELDS" und einem UCT-Konstantenwert von 0.0 zur Untersuchung einer Bedrohungssituation im Gomoku-Spiel. 300
Abbildung 80: Startzustand des *Amazons*-Spiels.....302
Abbildung 81: Statistiken der Entscheidungsfindung zweier konkurrierender Meta Agenten im Amazons-Spiel.....304
Abbildung 82: Initialkonfiguration des Damespieles (Checkers).....358
Abbildung 83: Die Initialkonfiguration des Backgammonspieles mit eingezeichneten Zugrichtungen der beiden Spieler Weiß und Schwarz. ..362
Abbildung 84: Zugmöglichkeiten in einer Runde des Backgammonspieles.....363

10 Tabellenverzeichnis

Tabelle 1: Informelle Auszahlungsmatrix des Kochspieles. 77

Tabelle 2: Auszahlungsmatrix des Gefangenendilemmas 79

Tabelle 3: Auszahlungsmatrix des Gefangenendilemmas mit hervorgehobenem Nash-Gleichgewicht. 81

Tabelle 4: Auszahlungsmatrix des Matching-Pennies Spieles mit reduzierten Auszahlungen 87

Tabelle 5: Übersicht über die Spielkonfiguration *Schachmatt in zwei Zügen* 115

Tabelle 6: Die Spielfiguren des Schachspieles 241

Tabelle 7: Die Spielfiguren des Damespieles 357

Tabelle 8: Die Spielfiguren des Backgammonspieles 361

11 Abkürzungsverzeichnis

AAAI	American Association for Artificial Intelligence
AI	Artificial Intelligence
AMAF	All Moves as First [Heuristik]
ASCII	American Standard Code for Information Interchange
CI	Computational Intelligence
CSS	Cascading Style Sheets
CPU	Central Processing Unit
DOM	Document Object Model
Gala	Game Language
GDL	Game Description [auch: Definition] Language
GGP	General Game Playing
GGPC	General Game Playing Competition
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
KI	Künstliche Intelligenz
KIF	Knowledge Interchange Format
KNN	Künstliches neuronales Netz
MAST	Move Average Sampling Technique
MBP	Problem des mehrarmigen Banditen
MCSS	Monte Carlo Spielbaumsuche
MCTS	Monte Carlo Tree Search
MVC	Model View Controller
RAVE	Rapid Action Value Estimation
RAM	Random Access Memory, physischer Arbeitsspeicher
TD-Lernen	Temporal Difference Lernen [Lernen anhand zeitlicher Differenz]
UCB	Upper Confidence Bounds
UCT	Upper Confidence Bounds applied to Trees
UML	Unified Modeling Language

WebGL	Web Graphics Library
ZoG	Zillions of Games

12 Verzeichnis der Online-Ressourcen

In der Arbeit und den *SpoookyJS*-Quellen erwähnte JavaScript-Bibliotheken und -Frameworks:

- Projekthomepage des Frameworks *SpoookyJS*: http://www.spoookyjs.de
- AngularJS: https://angularjs.org
- Angular UI Bootstrap: http://angular-ui.github.io/bootstrap
- Bootstrap: http://getbootstrap.com
- D3.js: http://d3js.org
- C3.js – D3-based reusable chart library: http://c3js.org
- jQuery: http://jquery.com
- Underscore.js: http://underscorejs.org
- Beispielimplementation des Q-Lernalgorithmus‘: https://github.com/nrox/q-learning.js
- ConvNetJS – Bibliothek für JavaScript-basierte künstliche neuronale Netze: https://github.com/karpathy/convnetjs bzw. http://convnetjs.com

13 Anhang

13.1 Listing – Vollständige Beschreibung des Tic Tac Toe Spieles in der GDL

Der folgende Quelltext beschreibt das *Tic Tac Toe* Spiel vollständig in der Game Description Language:[530]

```
1    ;;; Tictactoe
2
3    ;;; Komponenten
4
5        (role white)
6        (role black)
7
8        (<= (base (cell ?m ?n x)) (index ?m) (index ?n))
9        (<= (base (cell ?m ?n o)) (index ?m) (index ?n))
10       (<= (base (cell ?m ?n b)) (index ?m) (index ?n))
11       (base (control white))
12       (base (control black))
13
14       (<= (input ?r (mark ?m ?n))
                 (role ?r) (index ?m) (index ?n))
15       (<= (input ?r noop) (role ?r))
16
17       (index 1)
18       (index 2)
19       (index 3)
20
21   ;;; init
22
23       (init (cell 1 1 b))
24       (init (cell 1 2 b))
25       (init (cell 1 3 b))
26       (init (cell 2 1 b))
27       (init (cell 2 2 b))
28       (init (cell 2 3 b))
29       (init (cell 3 1 b))
30       (init (cell 3 2 b))
31       (init (cell 3 3 b))
32       (init (control white))
33
34   ;;; legal
35
36       (<= (legal ?w (mark ?x ?y))
37           (true (cell ?x ?y b))
38           (true (control ?w)))
39
40       (<= (legal white noop)
41           (true (control black)))
```

[530] Die Spielbeschreibung folgt dem *Knowledge Interchange Format* (*KIF*) und ist wiedergegeben nach http://gamemaster.stanford.edu/gamemaster/games/tictactoe/tictactoe.kif (zuletzt aufgerufen am 20.08.2014).

```
42
43        (<= (legal black noop)
44            (true (control white)))
45
46    ;;; next
47
48        (<= (next (cell ?m ?n x))
49            (does white (mark ?m ?n))
50            (true (cell ?m ?n b)))
51
52        (<= (next (cell ?m ?n o))
53            (does black (mark ?m ?n))
54            (true (cell ?m ?n b)))
55
56        (<= (next (cell ?m ?n ?w))
57            (true (cell ?m ?n ?w))
58            (distinct ?w b))
59
60        (<= (next (cell ?m ?n b))
61            (does ?w (mark ?j ?k))
62            (true (cell ?m ?n b))
63            (distinct ?m ?j))
64
65        (<= (next (cell ?m ?n b))
66            (does ?w (mark ?j ?k))
67            (true (cell ?m ?n b))
68            (distinct ?n ?k))
69
70        (<= (next (control white))
71            (true (control black)))
72
73        (<= (next (control black))
74            (true (control white)))
75
76
77        (<= (row ?m ?x)
78            (true (cell ?m 1 ?x))
79            (true (cell ?m 2 ?x))
80            (true (cell ?m 3 ?x)))
81
82        (<= (column ?n ?x)
83            (true (cell 1 ?n ?x))
84            (true (cell 2 ?n ?x))
85            (true (cell 3 ?n ?x)))
86
87        (<= (diagonal ?x)
88            (true (cell 1 1 ?x))
89            (true (cell 2 2 ?x))
90            (true (cell 3 3 ?x)))
91
92        (<= (diagonal ?x)
93            (true (cell 1 3 ?x))
94            (true (cell 2 2 ?x))
95            (true (cell 3 1 ?x)))
96
97
98        (<= (line ?x) (row ?m ?x))
99        (<= (line ?x) (column ?m ?x))
100       (<= (line ?x) (diagonal ?x))
101
102
103       (<= open (true (cell ?m ?n b)))
```

```
104
105  ;;; goal
106
107      (<= (goal white 100)
108          (line x)
109          (not (line o)))
110
111      (<= (goal white 50)
112          (not (line x))
113          (not (line o)))
114
115      (<= (goal white 0)
116          (not (line x))
117          (line o))
118
119      (<= (goal black 100)
120          (not (line x))
121          (line o))
122
123      (<= (goal black 50)
124          (not (line x))
125          (not (line o)))
126
127      (<= (goal black 0)
128          (line x)
129          (not (line o)))
130
131  ;;; terminal
132
133      (<= terminal
134          (line x))
135
136      (<= terminal
137          (line o))
138
139      (<= terminal
140          (not open))
```

13.2 Kommunikationsprotokoll einer vollständigen GGP-Partie des Tic Tac Toe Spieles

Die nachfolgend wiedergegebenen *HTTP*-Nachrichten dokumentieren die Kommunikation zwischen dem *Gamemaster* und dem spielenden System. Das Kommunikationsprotokoll ist vollständig wiedergegeben nach Love et al. 2008, S. 19–22.

```
POST / HTTP/1.0

Accept: text/delim

Sender: GAMEMASTER

Receiver: GAMEPLAYER

Content-type: text/acl

Content-length: 1554

(START MATCH.3316980891 X (ROLE X) (ROLE O) (INIT (CELL 1 1 B))
(INIT (CELL 1 2 B)) (INIT (CELL 1 3 B))(INIT (CELL 2 1 B)) (INIT
(CELL 2 2 B)) (INIT (CELL 2 3 B)) (INIT (CELL 3 1 B))(INIT (CELL 3
2 B)) (INIT (CELL 3 3 B)) (INIT (CONTROL X))(<= (NEXT (CELL ?X ?Y
?PLAYER)) (DOES ?PLAYER (MARK ?X ?Y)))(<= (NEXT (CELL ?X ?Y ?MARK))
(TRUE (CELL ?X ?Y ?MARK)) (DOES ?PLAYER (MARK ?M ?N))(DISTINCTCELL
?X ?Y ?M ?N)) (<= (NEXT (CONTROL X)) (TRUE (CONTROL O)))(<= (NEXT
(CONTROL O)) (TRUE (CONTROL X))) (<= (ROW ?X ?PLAYER) (TRUE (CELL
?X 1 ?PLAYER))(TRUE (CELL ?X 2 ?PLAYER)) (TRUE (CELL ?X 3 ?PLAYER)))
(<= (COLUMN ?Y ?PLAYER)(TRUE (CELL 1 ?Y ?PLAYER)) (TRUE (CELL 2 ?Y
?PLAYER)) (TRUE (CELL 3 ?Y ?PLAYER)))(<= (DIAGONAL ?PLAYER) (TRUE
(CELL 1 1 ?PLAYER)) (TRUE (CELL 2 2 ?PLAYER))(TRUE (CELL 3 3
?PLAYER))) (<= (DIAGONAL ?PLAYER) (TRUE (CELL 1 3 ?PLAYER))(TRUE
(CELL 2 2 ?PLAYER)) (TRUE (CELL 3 1 ?PLAYER))) (<= (LINE ?PLAYER)
(ROW ?X ?PLAYER))(<= (LINE ?PLAYER) (COLUMN ?Y ?PLAYER)) (<= (LINE
?PLAYER) (DIAGONAL ?PLAYER))(<= OPEN (TRUE (CELL ?X ?Y B))) (<=
(DISTINCTCELL ?X ?Y ?M ?N) (DISTINCT ?X ?M))(<= (DISTINCTCELL ?X ?Y
?M ?N) (DISTINCT ?Y ?N)) (<= (LEGAL ?PLAYER (MARK ?X ?Y))(TRUE (CELL
?X ?Y B)) (TRUE (CONTROL ?PLAYER))) (<= (LEGAL ?PLAYER NOOP)(NOT
(TRUE (CONTROL ?PLAYER)))) (<= (GOAL ?PLAYER 100) (LINE ?PLAYER))(<=
(GOAL ?PLAYER 50) (NOT (LINE X)) (NOT (LINE O)) (NOT OPEN)) (<=
(GOAL ?PLAYER1 0)(LINE ?PLAYER2) (DISTINCT ?PLAYER1 ?PLAYER2)) (<=
(GOAL ?PLAYER 0) (NOT (LINE X))(NOT (LINE O)) OPEN) (<= TERMINAL
(LINE ?PLAYER)) (<= TERMINAL (NOT OPEN))30 30)
```

```
HTTP/1.0 200 OK
Content-type: text/acl
Content-length: 5

READY

POST / HTTP/1.0
Accept: text/delim
Sender: GAMEMASTER
Receiver: GAMEPLAYER
Content-type: text/acl
Content-length: 27

(PLAY MATCH.3316980891 NIL)

HTTP/1.0 200 OK
Content-type: text/acl
Content-length: 10

(MARK 3 3)

POST / HTTP/1.0
Accept: text/delim
Sender: GAMEMASTER
Receiver: GAMEPLAYER
Content-type: text/acl
Content-length: 40

(PLAY MATCH.3316980891 ((MARK 3 3) NOOP))
```

```
HTTP/1.0 200 OK
Content-type: text/acl
Content-length: 4

NOOP

POST / HTTP/1.0
Accept: text/delim
Sender: GAMEMASTER
Receiver: GAMEPLAYER
Content-type: text/acl

Content-length: 40
(PLAY MATCH.3316980891 (NOOP (MARK 1 3)))
HTTP/1.0 200 OK
Content-type: text/acl
Content-length: 10

(MARK 2 2)

POST / HTTP/1.0
Accept: text/delim
Sender: GAMEMASTER
Receiver: GAMEPLAYER
Content-type: text/acl
Content-length: 40

(PLAY MATCH.3316980891 ((MARK 2 2) NOOP))

HTTP/1.0 200 OK
Content-type: text/acl
```

```
Content-length: 4

NOOP

POST / HTTP/1.0
Accept: text/delim
Sender: GAMEMASTER
Receiver: GAMEPLAYER
Content-type: text/acl
Content-length: 40

(PLAY MATCH.3316980891 (NOOP (MARK 1 2)))

HTTP/1.0 200 OK
Content-type: text/acl
Content-length: 10
(MARK 1 1)
POST / HTTP/1.0
Accept: text/delim

Sender: GAMEMASTER
Receiver: GAMEPLAYER
Content-type: text/acl
Content-length: 40

(STOP MATCH.3316980891 ((MARK 1 1) NOOP))

HTTP/1.0 200 OK
Content-type: text/acl
Content-length: 4
DONE
```

13.3 Regelwerke

13.3.1 Dame International, Checkers, Draughts

Ob Dame nach deutscher oder internationaler Grundregel, ob *Schaschki*, *Le Jeu des Dames*, das italienische *Dama*, *Contract-Checkers* oder weitere Varianten wie *Eck-Dame*, *Grashüpfer*, *Wolf und Schafe* oder *Blockade* – das Damespiel ist sich in seinen mehr als 150 Varianten und Variationen[531] ob seiner konstituierenden Bestandteile zumeist gleich und doch allzu verschieden: In der Spielwelt des Dame-Spieles und seinen Varianten stehen sich die Spielfiguren zweier Spielerinnen und Spieler gegenüber, um mit- und gegeneinander zu agieren, sich gegenseitig zu blockieren und zu dezimieren, um Sieg oder Niederlage einer Spielerin und eines Spielers zu fordern.

In seiner internationalen Erscheinungs- und Regelform, die den Namen *Draughts* (u.a.: England, Schottland) oder *Checkers* (USA) trägt – und gemeint ist, wenn auf den folgenden Seiten vom Damespiel die Rede ist –, spielt sich das Damespiel auf einem Spielbrett mit $8 \times 8 = 64$ alternierend weiß und schwarz eingefärbten Spielfeldern ab. Zu Beginn des Spieles verfügt jeder Spieler über ein Dutzend Spielsteine, die auch als *stone*, *piece* oder *men* bezeichnet werden. Erreicht ein solcher Spielstein im Laufe der Spielpartie die letzte der gegenüberliegenden Reihen des Spielfeldes, so wird der Stein transformiert in eine *Dame*.

Spielfiguren des Spielers *Schwarz*			Spielfiguren des Spielers *Weiß*		
Name	Symbol	Initiale Anzahl	Name	Symbol	Initiale Anzahl
Stein	●	12	Stein	◯	12
Dame	●	0	Dame	◯	0

Tabelle 7: Die Spielfiguren des Damespieles

Im Initialzustand des Spieles, wie er in der folgenden Abbildung 82 dargestellt ist, belegen die zwölf Steine des Spielers Schwarz die dunklen Spielfelder der oberen

[531] Vgl. Schaeffer 2009, S. 25 sowie den Überblick über zahlreiche Dame-Varianten in Glonnegger 2009, S. 154–161.

zwei Reihen des Spielbrettes und die Steine des Spielers Weiß die dunklen Felder der beiden unteren Reihen des Spielfeldes. In seinem Verlauf gestaltet sich die Spielpartie rundenbasiert: Auf den Zug des einen Spielers folgt der Zug des anderen Spielers. Schwarz beginnt die Spielpartie und bewegt sich mit einem seiner Steine auf die gegnerischen Reihen zu. Anschließend beantwortet Weiß den Zug des Gegenspielers mit einem seiner Züge.

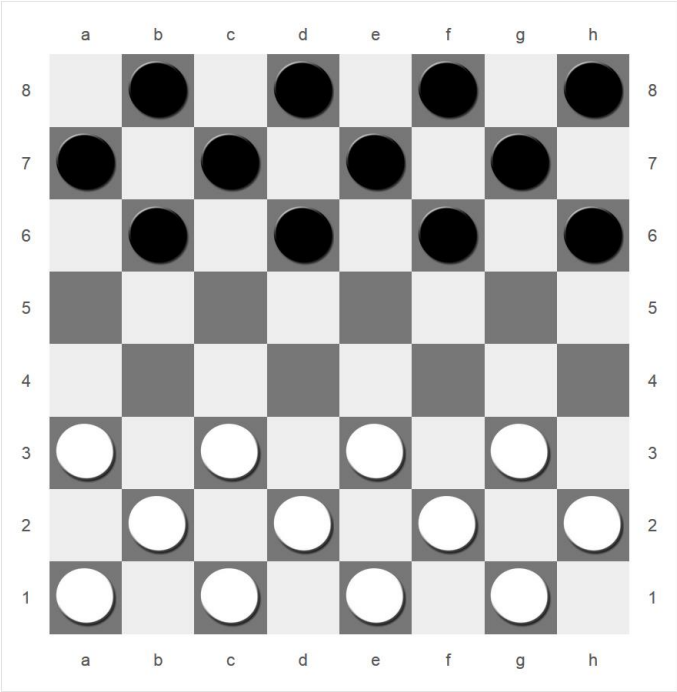


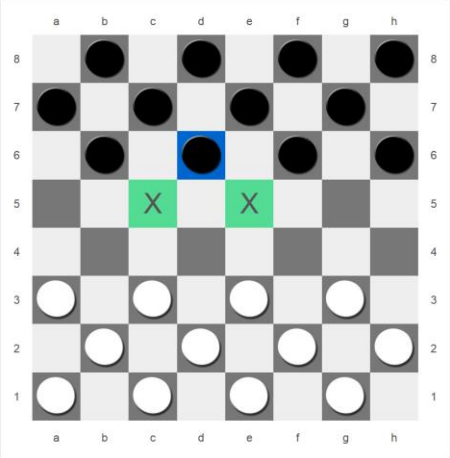
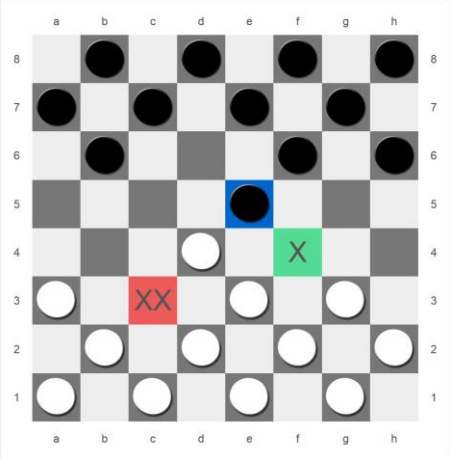
Abbildung 82: Initialkonfiguration des Damespieles (Checkers)

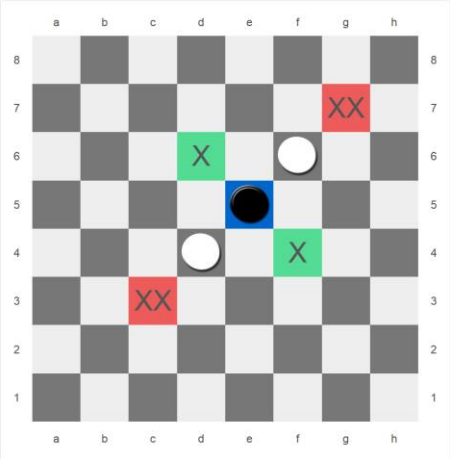
In der betrachteten Variante des Damespieles besteht Schlagzwang: Kann ein Spieler einen gegnerischen Spielstein schlagen, so muss er den schlagenden Zug auch auszuführen. Ist es einem Spieler möglich, in einer Runde mehrere gegnerische Spielsteine nacheinander zu schlagen, so muss er jene mehreren schlagenden Züge vollziehen, auch wenn sich für ihn durch den Zugzwang eine ungünstige Spielsituation ergibt. Das Spiel gewinnt, wer alle Spielfiguren seines Gegenspielers geschlagen oder wer den Kontrahenten zugunfähig gemacht hat.

Differenzieren lässt sich im Damespiel zwischen den individuellen Zielen der Spielsteine und übergeordneten Zielen, den globalen Spielzielen der Spieler Schwarz und Weiß: Die individuellen Ziele der Agierenden des Spieles, d.h. der Spielsteine, bestehen zum einen darin, die letzte gegenüberliegende Reihe zu erreichen, um sich in einen Damestein zu transformieren, der über elaborierte

Möglichkeiten verfügt, sich auf dem Spielbrett zu bewegen. Zum anderen verfolgen die Spielsteine das individuelle Ziel, gegnerische Spielsteine zu überspringen, um den Spielfigurenvorrat des Gegenspielers zu dezimieren. Globale Spielziele, die alle Agierenden des Spieles zu erreichen bestrebt sind, stellen sich als die Gewinnzustände, die Endzustände des Spieles dar: Das Spiel gewinnt, wer seinen Gegner zugunfähig gesetzt hat oder jeden Spielstein des Gegners schlägt, sich folglich kein gegnerischer Spielstein mehr auf dem Spielbrett befindet.

Die folgende Darstellung bietet abschließend eine kurze Übersicht über die einzelnen Spielfiguren und informiert sowohl über Zug- als auch die Schlagmöglichkeiten von Stein und Dame.

Name	Zugmöglichkeiten	Erläuterung
Stein		Dem Spielstein des Damespieles ist es möglich, sich in Richtung der gegnerischen Reihen auf ein diagonal unmittelbar angrenzendes Feld – in der Darstellung zur Linken mit einem Kreuz markiert – zu bewegen, das von keinem Spielstein besetzt ist.
		Ein gegnerischer Spielstein wird geschlagen, wenn sich auf dem diagonal unmittelbar angrenzenden Feld in Zugrichtung ein gegnerischer Spielstein befindet und das Feld hinter dem gegnerischen Stein frei ist (mit „XX“ markiert“). Der schlagende Stein überspringt den gegnerischen und wird auf dem

		Feld hinter dem geschlagenen Stein abgelegt.
Dame		Ein Damespielstein darf diagonal in alle Richtungen ziehen und schlagende Züge ausführen.

13.3.2 Backgammon

In der digitalen Implementation ist Backgammon das aufwändigste Brettspiel der betrachteten Spiele. So erfordert die lineare Spielwelt, in der Spielsteine beliebig aufeinandergestapelt werden können, besondere Mechanismen und so arbeitet Backgammon mit unterschiedlichen Spielbereichen jenseits des eigentlichen Spielbrettes. Seinen Lauf nimmt eine Partie Backgammon auf einem besonderen Spielfeld, das aus 24 Teilen besteht, den *Zungen*. Jene Zungen nehmen eine beliebige Anzahl von Spielsteinen der Spieler auf. Eine Zunge, auf der sich zwei oder mehr Steine eines Spielers befinden, wird als *Band* bezeichnet, mehrere Bänder nebeneinander bilden einen *Block*.

Unterteilt ist das Spielbrett in vier unterschiedliche Bereiche: Jeder Spieler verfügt jeweils über ein *inneres* Feld, das als Heim- oder Zielfeld bezeichnet wird und bewegt seine Spielsteine über die *äußeren* Felder in Richtung seines eigenen Zielfeldes. Zu Beginn des Spieles besitzt jeder Spieler 15 Spielsteine, die in festgelegter Konfiguration auf dem Spielbrett abgelegt werden.

Spielfiguren des Spielers *Schwarz*			Spielfiguren des Spielers *Weiß*		
Name	Symbol	Initiale Anzahl	Name	Symbol	Initiale Anzahl
Stein	●	15	Stein	◯	15

Tabelle 8: Die Spielfiguren des Backgammonspieles

Die Bewegung der Spielsteine auf dem Spielbrett erfolgt entgegengesetzt: Weiß zieht seine Spielsteine im Uhrzeigersinn und Schwarz bewegt seine Steine gegen den Uhrzeigersinn nach den Augen der Würfel in Richtung seines Zielfeldes. Die folgende Darstellung visualisiert die initiale Anordnung der jeweils 15 Spielsteine der beiden Spieler Schwarz und Weiß, veranschaulicht die Zugrichtungen der beiden Spieler und skizziert die Unterteilung des Spielbrettes in vier unterschiedliche Bereiche.

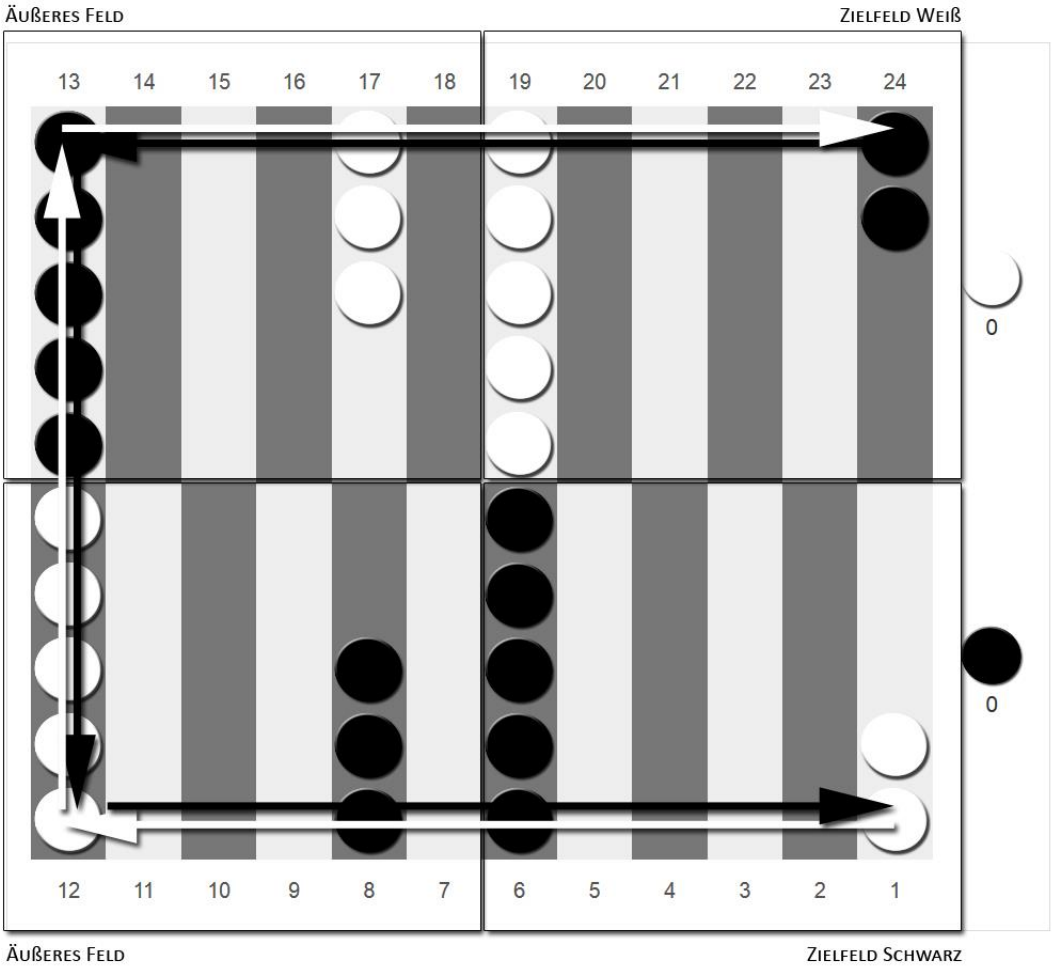


Abbildung 83: Die Initialkonfiguration des Backgammonspieles mit eingezeichneten Zugrichtungen der beiden Spieler Weiß und Schwarz.

Zu Beginn der Backgammonpartie entscheidet der Zufall, welcher Spieler in der ersten Runde am Zug ist: Beide Spieler werfen jeweils einen Würfel und derjenige beginnt das Spiel, wer die höchste Augenzahl gewürfelt hat. Anschließend nimmt die Spielpartie ihren Lauf: Am Anfang einer jeden Spielrunde werden die beiden sechsseitigen Würfel geworfen und jene Spielsteine nach den Augen der Würfel bewegt, die sich an oberster Position auf einer Zunge des Spielbrettes befinden. Zeigen beide Sechsflächner die gleiche Augenzahl, so werden die Würfel verdoppelt und der aktuelle Spieler kann in der Spielrunde vier Züge ausführen. Die Reihenfolge, in der die Würfelwerte eingesetzt werden, ist beliebig, jedoch besteht Zugzwang, so dass alle Zugmöglichkeiten ausgeführt werden müssen. Ist es einem Spieler nicht möglich, einen seiner Spielsteine zu bewegen, ist die Runde beendet und der Gegenspieler ist an der Reihe.

In der folgenden Darstellung hat Weiß eine Eins und eine Drei gewürfelt. Weil Züge auf gegnerisch besetzte Bänder nicht gestattet sind, kann Weiß einen seiner vier Steine auf eine der sieben dargestellten Arten bewegen:

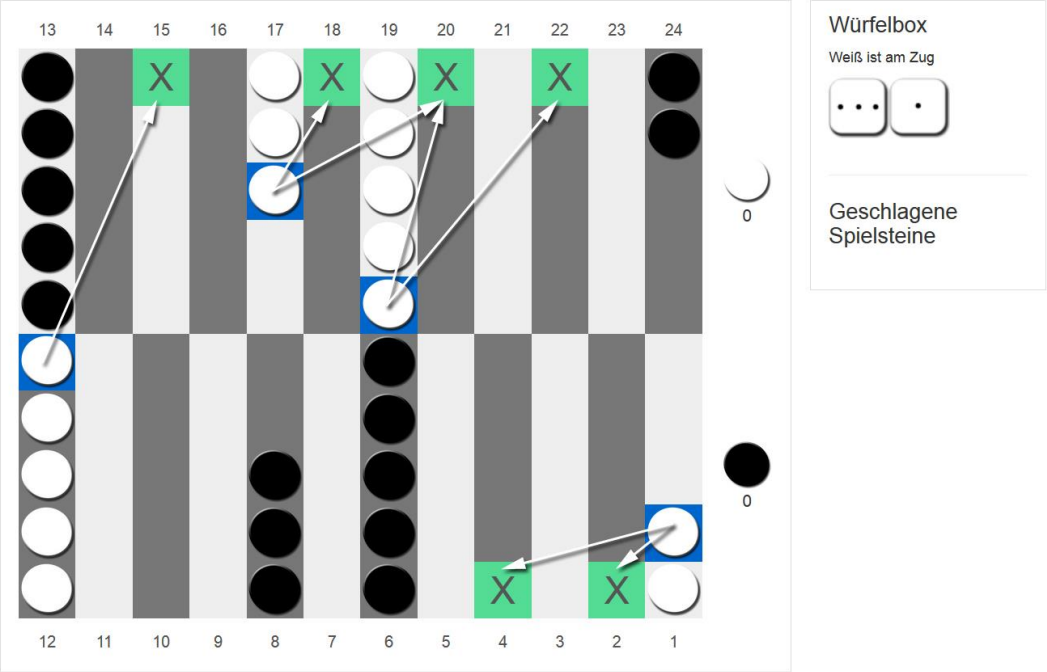


Abbildung 84: Zugmöglichkeiten in einer Runde des Backgammonspieles

Erreicht ein Spieler mit einer seiner Spielfiguren eine Zunge, auf der sich einzig ein gegnerischer Stein befindet, so wird der gegnerische Stein vom Spielbrett genommen und in einem gesonderten Bereich der Spielwelt abgelegt. Anschließend nimmt der schlagende Spielstein die Position des geschlagenen Steines ein. Auf eine solche Art geschlagene Steine müssen schnellstmöglich wieder in das Spiel gebracht werden: Verfügt ein Spieler über ausgewürfelte Steine, so muss er die entsprechenden Steine einwürfeln, bevor erneut andere Spielsteine bewegt werden dürfen. Ausgangspunkt für das Einwürfeln der geschlagenen Steine ist das gegnerische Zielfeld. Ist es einem Spieler nicht möglich, seinen geschlagenen Stein in das gegnerische Zielfeld zu bringen, so dürfen keine anderen Steine bewegt werden – die Runde wird beendet und der Gegenspieler ist am Zug. Aufgegliedert in ihre Bestandteile gestaltet sich die *Einwürfelregel* kompakt:

Spielregel *Einwürfeln geschlagener Spielsteine*

Wenn die Menge M der geschlagenen Steine des aktuellen Spielers nicht leer ist ($Atom_2$), **dann** beschränke die Zugmöglichkeiten des aktuellen Spielers einzig auf Spielsteine aus der Menge M der geschlagenen Steine des aktuellen Spielers.

Im Laufe der Spielpartie intendieren beide Spieler zum einen, die gegnerischen Spielsteine durch geschickte Anordnung der eigenen Steine zu blockieren, mitunter gar zugunfähig zu machen und zum anderen, die eigenen Spielsteine auszuwürfeln und somit das Spiel für sich zu entscheiden. Befinden sich alle Steine eines Spielers im eigenen Zielfeld – für Weiß sind das folglich die Zungen 19 bis 24 und für Schwarz die Zungen eins bis sechs –, so werden die gewürfelten Augen dafür verwendet, die entsprechenden Steine vom Spielbrett zu nehmen. Einfach ist das Spiel gewonnen, wenn ein Spieler alle seine Spielsteine ausgewürfelt und der Gegenspieler bereits mit dem Auswürfeln seiner Steine begonnen hat. Als *Gammon* wird ein Sieg bezeichnet, bei dem der Gegenspieler noch keinen seiner Steine ausgewürfelt hat. Mit *Backgammon* ist ein Spielsieg signifiziert, bei dem sich gegnerische Steine auf dem Zielfeld des Siegers befinden. Bei einem *Gammon* erhält der Gewinner die doppelte Auszahlung und bei einem *Backgammon* wird die Auszahlung verdreifacht.

Neben der Intention, das Spiel mit einer möglichst hohen Auszahlung für sich zu bestreiten, besteht das individuelle Spielziel der Spielerinnen und Spieler des Backgammonspieles darin, alle eigenen Steine möglichst schnell in das Zielfeld zu bewegen und sich auf dem Weg dahin tunlichst selten angreifbar zu machen, indem Bänder und Blöcke gebildet werden, die eigene Steine schützen und dem Gegenspieler das Fortkommen auf dem Spielbrett erschweren.

14 Tabellarischer Lebenslauf

Persönliche Daten

Name:	Jan Gerrit Wieners
Wohnhaft:	Sülzburgstr. 200, 50937 Köln
Geburtsdatum,	15. März 1981
Geburtsort:	in Bergisch-Gladbach
Staatsangehörigkeit:	Deutsch

Studium / Schulbildung

2009-2015	Promotionsstudium an der Universität zu Köln
2002-2009	Magisterstudium an der Universität zu Köln: Historisch-Kulturwissenschaftliche Informationsverarbeitung, Germanistik und Philosophie Magisterarbeit: „Zur Erweiterungsfähigkeit bestehender OCR Verfahren auf den Bereich extrem früher Drucke“
2001-2002	Zivildienst am Zentrum für Frühbehandlung und Frühförderung gGmbH, Köln-Mülheim
1997-2000	Werner-von-Siemens-Berufs-Kolleg, Köln-Deutz Abschluss: Elektrotechnischer Assistent und allgemeine Hochschulreife
1991-1997	Städt. Realschule Köln-Dellbrück Abschluss: Fachoberschulreife mit Qualifikation
1987-1991	Städt. Katholische Grundschule Neufelder Straße, Köln-Holweide

Berufliche Erfahrung

10/2011-12/2011	Wissenschaftlicher Mitarbeiter im Projekt „Digitales Archiv NRW“
01/2009-12/2011	Wissenschaftlicher Mitarbeiter im Projekt „VD18 - Verzeichnis der im deutschen Sprachraum erschienenen Drucke des 18. Jahrhunderts“.
03/2010-05/2010	Wissenschaftlicher Mitarbeiter im Projekt „PLANETS Preservation and Long-term access through Networked Services – Digital Preservation Research and Technology“.
09/2004-12/2008	Studentische Hilfskraft am Regionalen Rechenzentrum Köln (RRZK)

05/2005-12/2008	Studentische Hilfskraft am Institut für Historisch-Kulturwissenschaftliche Informationsverarbeitung, Universität zu Köln
2004-2015	Freiberuflicher Webentwickler
2000-2001	Syseca, Gesellschaft für Unternehmensberatung mbH. Betreuung des Intranets, User Helpdesk

Veröffentlichungen

11/2010	M. Guttenbrunner, A. Rauber, M. Thaller, J. Wieners: Same Same But Different – Comparing Rendering Environments for Interactive Digital Objects. In Ioannides, M. et. al.: Digital Heritage - Third International Conference, EuroMed 2010. Lemessos, Cyprus, November 8-13, 2010 Proceedings. Berlin 2010: Springer Verlag.
05/2010	PLANETS Deliverable PC4/D13: „Extend the capabilities of the extraction tools to extract layout characteristics“ (http://planetarium.hki.uni-koeln.de/planets_cms/sites/default/files/PC4D13%20final.pdf)

Mitgliedschaften, Ämter

	CCeH - Cologne Center for eHumanities, University of Cologne
Seit 2013	Mittelbauvertreter in der Engeren Fakultät der Phil.-Fak. der Universität zu Köln

Universitäre Lehre

Seit 10/2010

Lehrkraft für besondere Aufgaben („Lecturer“) im Rahmen des IT-Zertifikates der Philosophischen Fakultät der Universität zu Köln mit den Lehrveranstaltungen:

- Advanced Web Basics
  Inhalte: Client- und serverseitige Webentwicklung mit HTML5, CSS, JavaScript, jQuery, PHP und MySQL
- Tools & Methods in Digital Humanities
  Inhalte: Web- und Appentwicklung für mobile Devices mit Bootstrap, jQuery Mobile und AngularJS

- Daten- und Metadatenstandards (bis 2014) Inhalte: X-Technologien (XML, DTD, XML Schema, XPath und XSLT, Standards), Metadatenstandards (TEI, Standards im BAM-Sektor, etc.)

Seit 10/2009 Lehraufträge für die Veranstaltungen

- Basisinformationstechnologie I & II
- Programmierpropädeutikum
- Programmierpraktikum C++

Durchgeführte Lehrveranstaltungen

WiSem 2014/2015

- Programmierpraktikum C++
- Basisinformationstechnologie / HK – Medien (Teil 1)
- IT-Zertifikat der Phil.-Fak.: Advanced Web Basics / Kurs A und B
- IT-Zertifikat der Phil.-Fak.: Tools & Methods in Digital Humanities - Mobile Web Developing

SoSem 2014

- Programmierpropädeutikum
- Basisinformationstechnologie / HK – Medien (Teil 2)
- IT-Zertifikat der Phil.-Fak.: Advanced Web Basics / Kurs A und B
- IT-Zertifikat der Phil.-Fak.: Tools & Methods in Digital Humanities - Mobile Web Developing

WiSem 2013/2014

- Digital Humanities (gemeinsam mit Simone Kronenwett)
- Programmierpraktikum C++
- Basisinformationstechnologie / HK – Medien (Teil 1)
- IT-Zertifikat der Phil.-Fak.: Advanced Web Basics / Kurs A und B
- IT-Zertifikat der Phil.-Fak.: Daten- und Metadatenstandards

SoSem 2013

- Programmierpropädeutikum
- Basisinformationstechnologie / HK – Medien (Teil 2)
- IT-Zertifikat der Phil.-Fak.: Advanced Web Basics / Kurs A und B
- IT-Zertifikat der Phil.-Fak.: Daten- und Metadatenstandards

WiSem 2012/2013

- Programmierpraktikum C++

	▪ Basisinformationstechnologie / HK – Medien (Teil 1) ▪ IT-Zertifikat der Phil.-Fak.: Advanced Web Basics / Kurs A und B ▪ IT-Zertifikat der Phil.-Fak.: Daten- und Metadatenstandards
SoSem 2012	▪ Basisinformationstechnologie / HK – Medien (Teil 2) ▪ IT-Zertifikat der Phil.-Fak.: Advanced Web Basics / Kurs A und B ▪ IT-Zertifikat der Phil.-Fak.: Daten- und Metadatenstandards
WiSem 2011/2012	▪ Basisinformationstechnologie / HK – Medien (Teil 1) ▪ IT-Zertifikat der Phil.-Fak.: Advanced Web Basics / Kurs A und B ▪ IT-Zertifikat der Phil.-Fak.: Daten- und Metadatenstandards
SoSem 2011	▪ Basisinformationstechnologie / HK – Medien (Teil 2) ▪ IT-Zertifikat der Phil.-Fak.: Advanced Web Basics / Kurs A und B ▪ IT-Zertifikat der Phil.-Fak.: Daten- und Metadatenstandards
WiSem 2010/2011	▪ Programmierpraktikum C++ ▪ Basisinformationstechnologie / HK – Medien (Teil 1) ▪ IT-Zertifikat der Phil.-Fak.: Advanced Web Basics / Kurs A und B
SoSem 2010	▪ Basisinformationstechnologie / HK – Medien (Teil 2)
WiSem 2009 / 2010	▪ Basisinformationstechnologie / HK – Medien (Teil 1)

Weiterbildungen

02/2003-05/2003	Ausbildung zum Übungsleiter-C Breitensport
05/1996-01/1997	Kurs in Maschinenschreiben, Privatschule Sienang-Zaier, Köln

15 Register

Agent
  Agentenfunktion 40, 45, 46
  Agentenprogramm 40, 45
  Definition 38
  Deskriptive Beschreibung 37
Chinesisches Zimmer 33
Kempelen, Wolfgang von 22
Künstliche Intelligenz 27, 35, 53, 94
Schachtürke 22, 23, 24, 26, 38
Turing, Alan M. 29
Turing-Test 29, 32, 33, 34