

Heuristische vs. stochastische Suche - Alpha Pruning und Monte Carlo Tree Search im Spiel der Amazonen

Bachelorarbeit

Gün Yanik
401565

4. November 2021

Betreuer: Prof. Dr. Benjamin Blankertz
Prof. Dr. Markus Brill

Technische Universität Berlin
Fakultät IV – Elektrotechnik und Informatik
Institut für Softwaretechnik und Theoretische Informatik
Fachgebiet Neurotechnologie

Kurzfassung

Das Spiel der Amazonen ist ein junges Brettspiel, es vereint in sich Eigenschaften vom Schach und GO. Seine Komplexität ist die primäre Herausforderung bei der Entwicklung eines Programms zur Suche des optimalen Zugs. Ziel der vorliegenden Arbeit ist, jene Herausforderungen und mögliche Lösungsstrategien, insbesondere heuristische und stochastische Suchmethoden sowie zugehörigen Funktionen und Formeln, im Kontext der Amazonen vorzustellen, zu implementieren und vergleichen. Spezifisch wird das Alpha Beta Pruning mit verschiedenen Heuristiken unterschiedlicher Monte Carlo Tree Search Modifikationen gegenübergestellt. Dazu wurden im Rahmen verschiedener Experimente, Daten zu wichtigen Vergleichskriterien wie der Laufzeit, dem Speicherverbrauch und der Optimalität gesammelt und ausgewertet. Die Ergebnisse bestätigen zu einem großen Teil die aktuelle Vorherrschaft stochastischer Suchmethoden in diesem Feld, hierbei liefert die einfache MCTS die besten Ergebnisse.

Inhaltsverzeichnis

| | |
|--|------------|
| Abbildungsverzeichnis | vi |
| Tabellenverzeichnis | vii |
| 1. Einleitung | 1 |
| 2. Spiel der Amazonen | 2 |
| 2.1. Spielfeld | 2 |
| 2.2. Regeln | 3 |
| 2.3. Notation | 3 |
| 2.4. Spielverlauf | 4 |
| 2.5. Computer Olympiaden | 4 |
| 3. Spieltheorie | 4 |
| 3.1. Strategische Spiele | 5 |
| 3.2. Spielbäume | 6 |
| 3.3. Algorithmen auf Spielbäumen | 6 |
| 4. Heuristische Suche | 8 |
| 4.1. Heuristiken | 9 |
| 4.2. Minimax Algorithmus | 10 |
| 4.3. Alpha Beta Pruning | 11 |
| 4.4. Implementierung | 13 |
| 4.5. Heuristiken für das Spiel der Amazonen | 15 |
| 4.5.1. Mobilitäts Evaluation | 15 |
| 4.5.2. Territoriale und lokale Evaluation | 16 |
| 4.5.3. Performanz | 19 |
| 5. Stochastische Suche | 20 |
| 5.1. Monte Carlo Tree Search | 20 |
| 5.2. UCT Algorithmus | 22 |
| 5.3. Optimierter UCT Algorithmus | 24 |
| 5.3.1. Pseudo-randomisiertes Spiel | 24 |
| 5.3.2. Differenzierte Zugbetrachtung | 25 |
| 5.4. Performanz | 26 |
| 6. Alpha Beta Suche vs. Monte Carlo Tree Search | 28 |
| 6.1. Speicher | 28 |

| | |
|---|-----------|
| 6.2. Experiment | 28 |
| 6.3. Auswertung | 30 |
| Referenzen | 31 |
| A. Anhang | 35 |
| A.1. Spielbrett Konfigurationen | 35 |

Abbildungsverzeichnis

| | | |
|------|--|----|
| 2.1. | Spiel der Amazonen Startaufstellung. | 2 |
| 2.2. | Beispiel erster Spielzug, bestehend aus zwei einzelnen Aktionen: Erst wird die Amazone bewegt und von dem Endpunkt aus ein Feld „verbrannt“. | 3 |
| 2.3. | Phasen des Spiels: Anfangsphase , zwischen Mitte und Füllphase , Endstellung mit Weiß als Sieger | 4 |
| 3.1. | Schachtürken, Wolfgang Kempelen, Stich von 1789 [Rac89] | 5 |
| 3.2. | Ausschnitt einer Spielbaumrepräsentation für 2x3 Amazonen, Spielsituationen werden durch Knoten dargestellt welche durch die jeweiligen Kanten als Aktionen erreicht werden können. | 6 |
| 3.3. | Spielbaum eines imaginären 2-Spieler-Nullsummenspiels, gewonnen durch höchste Bewertung. Zu sehen, eine Kombination beispielhafter Strategien beider Spieler, Bewertung -1 aus der Sicht von Weiß. Fett markierte Pfeile stellen Strategie dar. | 7 |
| 3.4. | Beispielhafte Propagierung Blattevaluationen nach <i>Minimax</i> , demnach wäre Aktion a die optimale aus der Sicht von Weiß. | 8 |
| 4.1. | Beispiel für mögliche Beschneidungen des Spielbaumes: Ein β - <i>Cutoff</i> beendet die weitere Suche unter einem MAX Knoten, wenn der vorhergehende MIN Knoten einen höchstens gleichgroßen Wert hat. Der α - <i>Cutoff</i> findet unter MIN Knoten statt, wenn dieser unbedeutend für den maximierenden Spieler ist. . . . | 11 |
| 4.2. | <i>Mobilitäts Evaluation</i> auf ein 2x3 Spiel der Amazonen angewendet. Die erste Ebene stellt die <i>heuristischen Werte</i> der jeweiligen Züge dar. | 16 |
| 4.3. | Distanzen $D_i^j(a)$. $D_q^j(a)$ links, $D_k^j(a)$ rechts. | 17 |
| 4.4. | Laufzeit von H_M und H_{TP} auf 4x4, 6x6, 8x8 und 10x10 Initialfeldern. | 19 |
| 5.1. | Vier Phasen eines Monte Carlo Tree Search Algorithmus nach [Cha+08] | 20 |
| 5.2. | Anzahl Simulationen der MCTS Algorithmen zu diversen Zeitfenstern. | 26 |
| 5.3. | Speicherverbrauch der <i>UCT</i> Methoden bei wachsender Laufzeit. | 27 |
| 6.1. | <i>MCTS</i> Algorithmen gegen <i>Alpha Beta Pruning</i> . <i>UCT</i> und seine Optimierung gegen Heuristiken für die Amazonen. | 29 |
| A.1. | 3x3 & 4x4 Spiel der Amazonen Konfiguration. | 35 |
| A.2. | 6x6 & 8x8 Spiel der Amazonen Konfiguration. | 35 |

Tabellenverzeichnis

| | |
|--|----|
| 3.1. Matrix zum Vergleich der verschiedenen Strategien unter Betrachtung der Spielabschlussbewertung aus der Sicht von Weiß (Für Schwarz umgekehrte Vorzeichen). | 7 |
| 4.1. Gewinner diverser <i>Alpha Beta Pruning</i> Spielkonstellationen, Suchtiefe 3. . . . | 19 |
| 5.1. Gewinnraten der MCTS Algorithmen auf variablen Einstellungen. | 27 |

1. Einleitung

Die Problematik der begrenzten Berechnungskapazität moderner Maschinen lässt Algorithmen, vor allem in der Spieltheorie, schnell an ihre Grenzen stoßen. Oftmals lassen sich bereits Subprobleme nur in exponentieller Laufzeit lösen.

Spiele wie Schach oder Go, welche primär durch ihre enorm große Anzahl an möglichen Spielzügen herausfordernd sind, waren bereits jahrzehntlang Gegenstand der Forschung der theoretischen Informatik [Min+06]. Anfangs war die heuristische Suche durch *Minimax* (siehe Abschnitt 4.2) und später *Alpha beta Pruning* (siehe Abschnitt 4.3) die dominante Lösungsstrategie, doch moderne Schach- und Go-engines (AlphaChess und AlphaGO) basieren auf randomisierter Suche durch *Monte Carlo Tree Search* (siehe Abschnitt 5.1).

Diese Arbeit befasst sich mit dem weitaus jüngeren Spiel der Amazonen und der Eignung genannter Lösungsstrategien. Das in Kapitel 2 vorgestellte Amazonen ist ein zwei-Spieler Brettspiel, in welchem die Spieler zugbasiert versuchen möglichst große Territorien abzustecken und die Bewegungsfähigkeit ihrer Figuren zu sichern.

Das Spiel der Amazonen vereinigt Aspekte aus Schach und Go und stellt dabei ganz neue Herausforderungen an die Komplexität der Algorithmen, welche im Kapitel zur Spieltheorie erläutert werden (siehe Kapitel 3).

Erwähnte Algorithmen sind relevante Suchmethoden aus der Familie der heuristischen und stochastischen Algorithmen:

Die *heuristische Suche* setzt dabei auf *Heuristiken* (siehe Abschnitt 4.1) um die Suche zielgerichteter zu gestalten. In Kapitel 4 wird der *Minimax Algorithmus* und seine Optimierung, das *Alpha Beta Pruning*, sowie zwei prominente Vertreter der *Heuristiken* für die Amazonen, vorgestellt und verglichen.

Einen alternativen Ansatz bietet die *stochastische Suche*, Algorithmen dieser Klasse sammeln durch randomisiertes Spielen Erfahrung, aufgrund derer die Entscheidung über den nächsten Zug getroffen wird. Kapitel 5 thematisiert *UCT*, einen *Monte Carlo Tree Search Algorithmus* und eine Optimierung dessen.

Welche Algorithmenfamilie und welcher Algorithmus genau für das Spiel der Amazonen am geeignetsten ist, wird in einem Experiment festgestellt. In diesem spielen alle Algorithmen in diversen Spielkonstellationen gegeneinander. Die Auswertung des Experiments und weiterer, im Verlauf der Arbeit stattgefundener, Experimente erfolgt in Kapitel 6.

2. Spiel der Amazonen

Das **Spiel der Amazonen**, kurz Amazonen und ursprünglich *El Juego de las Amazonas*, ist eines vom Argentinier Walter Zamkawkas, im Jahre 1988 erfundenes, territoriales Zwei-Spieler Brettspiel [Zam99]. Es vereint Movesets aus Schach und das Abstecken von Territorien der Spieler zu Beginn. Der Name leitet sich vom griechischen *Amazónes* ab und hat seine Wurzeln in der griechischen Mythologie [RÖS84].

Amazonen ist ein noch junges Spiel und hat vor allem durch seine Komplexität, mit durchschnittlich 400 Möglichkeiten pro Zug und 2176 erlaubter Anfangszügen auf dem Standardbrett der Größe 10x10 (siehe Abbildung 2.1), wissenschaftliches Interesse gewinnen können. Schach hat vergleichsweise 35, und Go, das für seinen hohen Verzweigungsgrad bekannt ist, etwa 361 mögliche Züge pro Zug auf dem üblichen 19x19 Brett [Kat+15].

2.1. Spielfeld

Amazonen wird standardmäßig auf einem **10x10** Schach- bzw. Damenfeld mit vier **Amazonen** pro Seite gespielt, darüber hinaus verfügen beide Seiten über etwa 50 Spielsteine, auch **Pfeile** genannt. Ferner existieren Versionen mit anderer Spielfeldgröße oder Figurenanzahl, mit veränderter Komplexität und Spiellänge, geringe Werte bedeuten ein weitaus schnelleres Spiel und weniger Zugmöglichkeiten pro Zug [Zam88]. Die folgenden Kapitel beziehen sich primär auf die Standardversion. Weitere Versionen sind im Anhang A.1 zu finden.

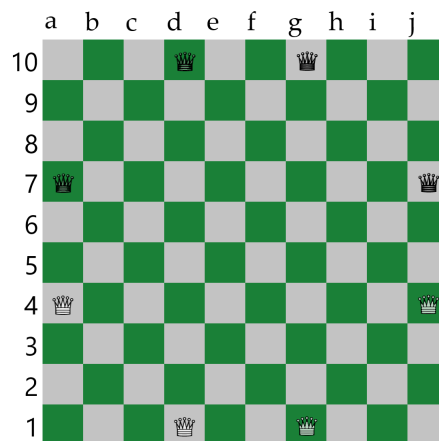


Bild 2.1.: Spiel der Amazonen Startaufstellung.

2.2. Regeln

Wie in Abschnitt 2.1 beschrieben, spielen beide Seiten, Schwarz und Weiß, mit jeweils 4 Amazonen. Diese übernehmen die Bewegungsregeln der Königin aus dem Schach: **horizontal**, **vertikal** und **diagonal** bis zu einem Hindernis (rote Linien in Abbildung 2.2). Wie im Schach wird zugbasiert gespielt und Weiß beginnt, damit enden die Gemeinsamkeiten. Zum einen können Amazonen nicht „geschlagen“ werden und zum anderen ist es dem Spieler nach dem Zug mit der Amazone erlaubt ein Feld, nach denselben Bewegungskriterien vom Zielort aus, mit einem *Pfeil* zu „verbrennen“. Dies geschieht mit den zusätzlichen Spielsteinen, darüber hinaus ist es keiner Amazone gestattet solch ein Feld oder eine andere Amazone zu überqueren [Zam88].

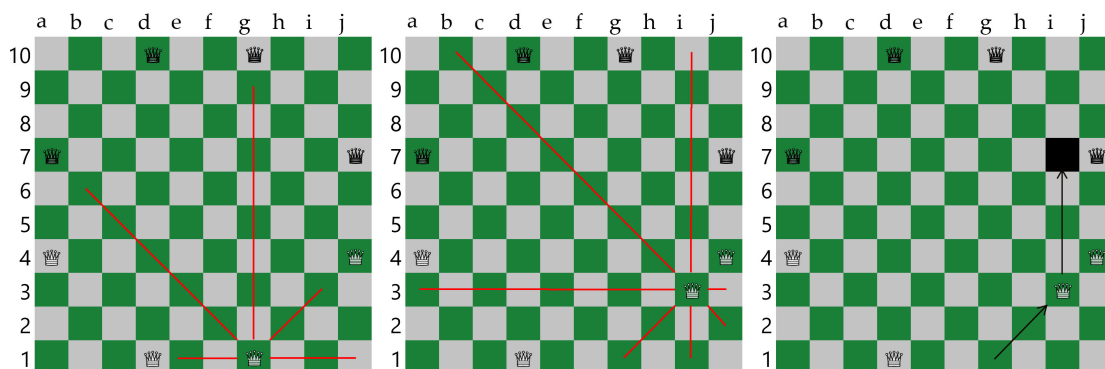


Bild 2.2.: Beispiel erster Spielzug, bestehend aus zwei einzelnen Aktionen:

Erst wird die Amazone bewegt und von dem Endpunkt aus ein Feld „verbrennt“.

Dieser kombinierte Zug (siehe Abbildung 2.2) trägt erheblich zur Komplexität des sonst einfachen Spiels bei, der Verzweigungsgrad wird erhöht [Kat+15]. Das Spielfähigkeitskriterium ist mindestens eine bewegungsfähige Amazone auf dem Feld zu haben, es gewinnt also jener Spieler, welcher zuerst alle Amazonen des Gegenspielers blockiert. Die Regeln machen zyklische Stellungen unmöglich, überdies existiert kein Unentschieden, da nach jedem Zug mindestens ein Feld unspielbar wird [Klo10]. Es verliert genau der Spieler, welcher zu Beginn des eigenen Zuges keinen Zug mehr tätigen kann, dies ist nach spätestens 92 Zügen der Fall [Lie04].

Es liegt in der Natur des Spiels, dass der Spieler mit dem ersten Zug bevorteilt ist, dies lässt sich durch verschiedene Regeln ausgleichen und muss gegebenenfalls im Rahmen von Algorithmen und Auswertung an anderer Stelle berücksichtigt werden [Lie05].

2.3. Notation

In dieser Arbeit wird eine vom Schach abgeleitete **Notation** verwendet. Während im Schach zu Beginn **e2-e4** einen Zug des Bauern von **e2** auf **e4** beschreibt, wird diese zu einem dreistelligen Format erweitert. Der im Bild 2.2 beschriebene Zug würde der Notation entsprechend **g1-i3-i7** lauten. Die Amazone von bewegt sich von **g1** auf **i3** und verbrennt von **i3** aus das Feld **i7**.

2.4. Spielverlauf

Im Verlauf des Spiels versuchen beide Spieler, den Regeln aus Abschnitt 2.2 folgend, die Mobilität der jeweils gegnerischen Amazonen einzuschränken.

Während über die ersten Züge hinweg das Spielfeld noch unübersichtlich ist und keine klaren Muster zu erkennen sind, bilden sich ab der **mittleren Phase** des Spiels Territorien, welche die Amazonen voneinander isolieren. In der *Endphase* bewegen sich diese nur noch in ihrem eigenen Gebieten und verschießen Pfeile, bis eine Seite keine Züge mehr zur Verfügung hat, daher wird diese auch als **Füllphase** bezeichnet [Lie05]. Ein solcher Spielablauf wird beispielhaft in Abbildung 2.3 gezeigt.

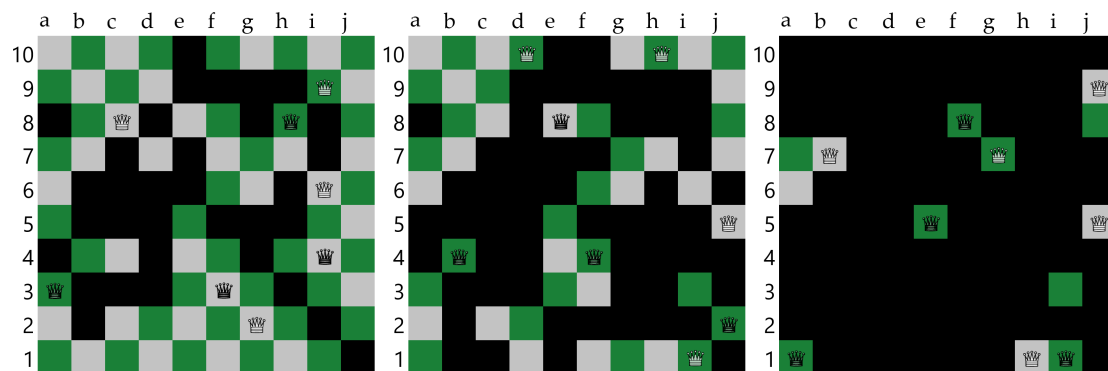


Bild 2.3.: Phasen des Spiels: **Anfangsphase**, zwischen **Mitte** und **Füllphase**, **Endstellung** mit Weiß als Sieger

Ebenfalls zu erkennen ist, dass mit dem Fortschreiten des Spiels, sich die Zugmöglichkeiten weiter beschränken, so nimmt die Komplexität mit dem Fortschreiten des Spiels stetig ab. Dennoch ist schon das Entscheiden, ob eine Amazone eine gewisse Zahl an Zügen hat, selbst im Endstadium des Spiels, ein NP-schweres Problem [Bur01].

2.5. Computer Olympiaden

Seit 2000 ist das Spiel der Amazonen ebenfalls auf der Computer Olympiade vertreten. Bei diesem Event geht es um die Meisterschaft der besten Computerprogramme zum Lösen komplexer Spiele wie chinesischem Schach, Connect6 und ursprünglich auch Schach. In den letzten Jahren konnte das momentan auf Monte Carlo Tree Search basierende Programm *Invader* [Lor18] 8 Goldmedaillen für die Amazonen gewinnen [Sat19].

3. Spieltheorie

Schon zu Zeiten des Feldherren Napoléons gab es erste Versuche, zumindest dem Anschein nach, gegen Maschinen zu spielen. Der „Schachtürke“ des Wolfgang von Kempelen siegte auf verschiedensten Höfen Europas über die begeisterten Monarchen im Schach, dieser war natürlich insgeheim von Menschenhand gesteuert [Kov84]. Seit Mitte der 1950er Jahre stellt künstliche Intelligenz ein wichtiges Feld der Informatik dar und realisiert zunehmend jene vergangenen Träume eines maschinellen Gegners [Min+06].

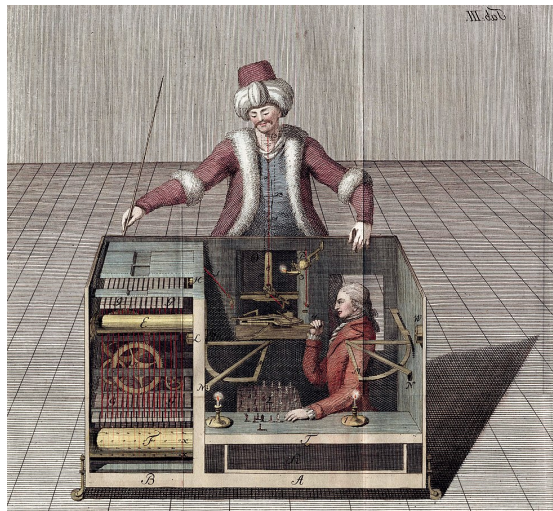


Bild 3.1.: Schachtürken, Wolfgang Kempelen, Stich von 1789 [Rac89]

Gesucht ist meist eine möglichst optimale Antwort bzw. Lösung zu einer Eingabe. In diesem Fall, in Form einer Spielsituation und dem Auffinden des besten Zuges zum Ziel. Da die Spieler in einer kompetitiven Umgebung spielen, wird diese Suche durch besondere Suchalgorithmen realisiert. Diese sind Inhalt der folgenden Kapitel, angewendet auf das eingangs eingeführte Spiel der Amazonen. Vorausgehend wird das Spiel zunächst in den spieltheoretischen Kontext überführt.

3.1. Strategische Spiele

Allgemein beschreibt ein **Suchproblem** in der theoretischen Informatik die Suche nach einer optimalen Sequenz von Aktionen, welche auf einen Startzustand angewendet werden muss, um in einen der definierten Zielzustände zu gelangen [Fri20, S. 14]. Strategische Spiele wie Schach oder das Spiel der Amazonen stellen eine besondere Form der Suchprobleme dar. Mehrere Akteure, in dem Fall zwei, spielen gegeneinander und die Aktionssequenz alterniert zwischen voneinander meist unterschiedlich gerichteten Interessen bzw. Zielen. Aufgrund dessen, müssen Aktionen, unter Berücksichtigung entgegengesetzter Interessen, gewählt werden.

3.2. Spielbäume

Das Spiel der Amazonen lässt sich rekursiv als Set möglicher Züge eines jeden Spielers definieren. Es bietet sich ein für Suchprobleme typischer gerichteter azyklischer Graph bzw. Baum als Datenstruktur zur Berechnung und Repräsentation an. In einem vollständigen Spielbaum ist die Initialstellung die Wurzel, die Finalstellungen sind Blattknoten und Züge sind Kanten.

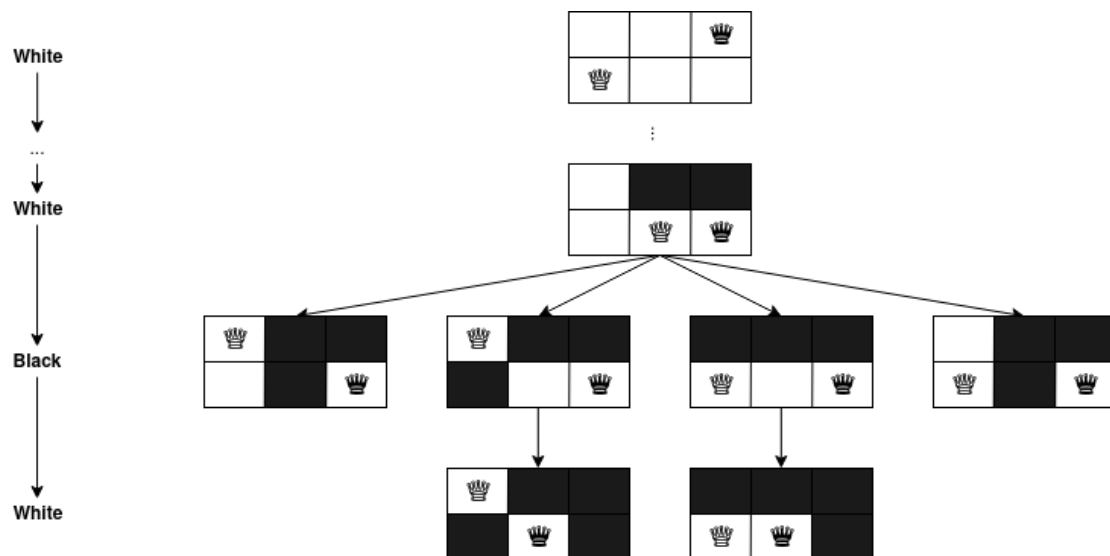


Bild 3.2.: Ausschnitt einer Spielbaumrepräsentation für 2x3 Amazonen, Spielsituationen werden durch Knoten dargestellt welche durch die jeweiligen Kanten als Aktionen erreicht werden können.

Der **Spielbaum** (siehe Abbildung 3.2) ist eine besondere Form des Suchbaums, er modelliert das Spiel und seinen Ablauf [Fri20, S. 106]. Der ziehende Akteur alterniert ebenenweise, um die Frage des gesuchten Pfades zum Sieg optimal zu beantworten zu können, wird eine Bewertung der möglichen Züge bzw. eine Abgrenzung der sich ändernden Interessen pro Zug und eine Strategie benötigt.

3.3. Algorithmen auf Spielbäumen

Das Spiel der Amazonen ist wie Schach ein *kompetitives Nullsummenspiel* [Hen01]. Ein guter Zug für den Spieler ist ein in gleichem Maße schlechter Zug für den Mitspieler und vice versa, ultimativ ist des einen Sieg des anderen Niederlage. Es liegt zudem ein *endliches und deterministisches* Spiel mit *perfekter Information* vor [KIB08], also eines in welchem beide Spieler alle Figurenpositionen kennen und damit die möglichen Züge des jeweiligen Gegenspielers. Gesucht ist eine vollständige Spezifikation über das Verhalten des Spielers, eine **Strategie**, welche zum Sieg führt.

Sind diese für beide Spieler gegeben, determiniert dies, im hier angenommenen Fall einer reinen Strategie, das Resultat des Spiels (siehe Abbildung 3.3).

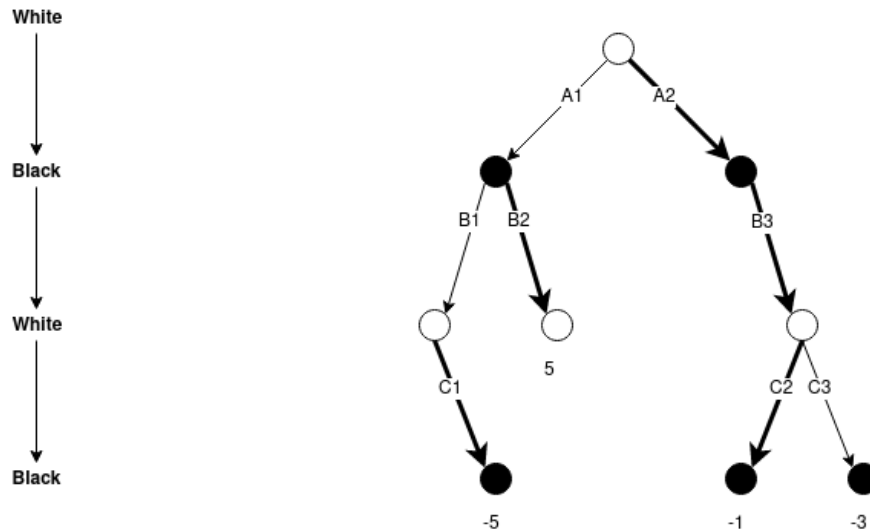


Bild 3.3.: Spielbaum eines imaginären 2-Spieler-Nullsummenspiels, gewonnen durch höchste Bewertung. Zu sehen, eine Kombination beispielhafter Strategien beider Spieler, Bewertung -1 aus der Sicht von Weiß. Fett markierte Pfeile stellen Strategie dar.

Die unterschiedlichen Kombinationen von Strategien lassen sich in einer Matrix übersichtlicher vergleichen und auswerten (siehe Tabelle 3.1): Für Weiß wäre eine gute Strategie $A1, C1, C2$ oder $A1, C1, C3$, um mit der gewählten Strategie den eigenen mindestens erzielbaren Gewinn zu maximieren auch *Maximin Strategie* bzw. den maximalen Gewinn des Gegenspielers zu minimieren (*Minimax Strategie*).

| | | Strategien Schwarz | |
|-----------------|--------------|--------------------|----------|
| | | $B1, B3$ | $B2, B3$ |
| Strategien Weiß | $A1, C1, C2$ | -5 | 5 |
| | $A1, C1, C3$ | -5 | 5 |
| | $A2, C1, C2$ | -1 | -1 |
| | $A2, C1, C3$ | -3 | -3 |

Tabelle 3.1.: Matrix zum Vergleich der verschiedenen Strategien unter Betrachtung der Spielabschlussbewertung aus der Sicht von Weiß (Für Schwarz umgekehrte Vorzeichen).

In einem *Nullsummenspiel* würden beide Spieler, der Strategie folgend, den erreichbaren Wert von einem Extremum in das jeweils Umgekehrte bewegen. Dies sowie ein substanzielles Konzept für die Lösung dieses Typs Spiel, beschreibt das erstmals 1928 durch von Neumann bewiesene **Minimax Theorem** [Neu28]. Auf jenes folgten noch weitere Varianten [BBS16].

Im Kern erwartet der Spieler maximalen Schaden durch die Aktionen des Gegenspielers, auf diese pessimistische Erwartung aufbauend versucht er diesen maximierten Schaden zu minimieren. Bei einer Betrachtung des tabellarischen Beispiels unter Berücksichtigung des *Minimax Theorems*, wählt Weiß $A2, C1, C2$ und Schwarz $B1, B3$ (oder $B2, B3$) als **Minimax Strategie**. Weiß verliert zwar knapp (-1), gleichwohl kann keiner der Spieler durch eine einseitige Änderung der Strategie den eigenen Wert verbessern bzw. den des Gegenspielers verschlechtern. Dieses Verhältnis der Strategien wird auch *Nash Equilibrium* oder *Nash Gleichgewicht* genannt [OR94].

Die eingangs genannten Eigenschaften des Spiels, würden es während der Suche potenziell erlauben, den gesamten Spielbaum zu generieren, eine Bewertung der Blattknoten vorzunehmen und diese anschließend den Baum aufwärts nach *Minimax-Prinzip* zu propagieren (siehe Abbildung 3.4). Dementsprechend für den Spieler auf jeder Spielbaumebene maximierend bzw. als Gegenspieler minimierend zu wählen (aus der Sicht des Spielers der Wurzel) und nach der sich daraus ergebene *Minimax Strategie* zu ziehen.

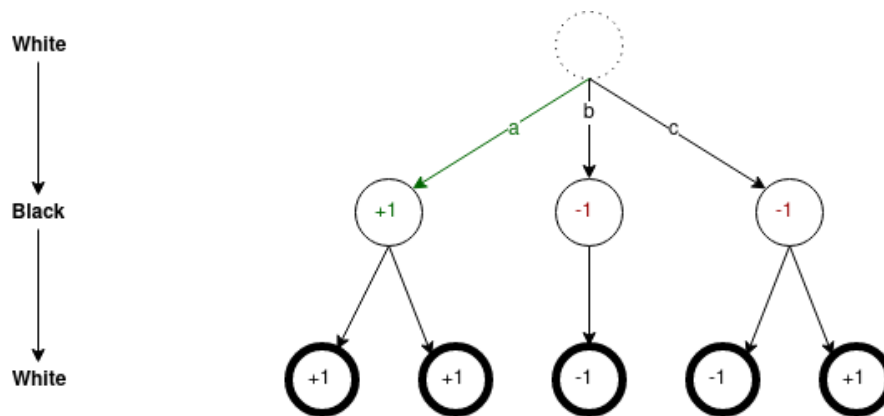


Bild 3.4.: Beispielhafte Propagation Blattevaluationen nach *Minimax*, demnach wäre Aktion **a** die optimale aus der Sicht von Weiß.

Jedoch wächst die *Spielbaum Komplexität*, die Menge an Knoten die generiert und betrachtet werden müssten, mit fortschreitender Tiefe exponentiell. Für das Spiel der Amazonen beträgt sie $\approx 10^{212}$ Knoten [Hen01], das beobachtbare Universum hat vergleichsweise $\approx 10^{80}$ Protonen (Eddington Zahl). Ein exorbitanter Zustandsraum. Dieser Ansatz wäre demnach Speicherkomplexitäts- und Laufzeittechnisch keine Option. Daher suchen gute Suchalgorithmen unbedingt zielgerichtet, um die Zahl zu untersuchenden Zustände und damit die Laufzeit zu minimieren, die Herausforderung besteht dann darin dennoch eine möglichst optimale Lösung zu finden [Fri20].

Kapitel 4 und 5 stellen zwei Ansätze und jeweilige Variationen für eine solche Suche vor. Die folgenden Algorithmen wurden in Python bzw. wesentlich beschleunigt in Cython [Wei] implementiert und veröffentlicht [Yan21]. Cython erzielt bei optimaler C-orientierter Implementierung Geschwindigkeiten ähnlich zu C, da der Code zu C Code umgewandelt und kompiliert wird. Simuliert wird unter Ubuntu 20.04 mit einer 40x2.30GHz Intel Xeon CPU und 30GB Ram.

4. Heuristische Suche

Wie bereits in Absatz 3.3 beschrieben, ist das Generieren des vollständigen Spielbaums aufgrund der Laufzeit- und Speicherkomplexität unmöglich. Es ist indes möglich über eine **informierte Suche** eine optimale Lösung bzw. den stärksten Zug zu finden. Dafür speichern Zustände zusätzliches **heuristisches Wissen**, durch welches die Suche zielorientierter erfolgen kann, abhängig von der Qualität der Heuristiken.

Die *heuristische Suche* kann durch diverse Algorithmen realisiert werden. Dieses Kapitel behandelt die für *2-Spieler-Spiele* etablierte **Minimax Suche** sowie ihre Optimierung die **Alpha-Beta-Suche**. Überdies bedarf es einer Spezifikation der *Heuristiken* für die Amazonen. Es werden zwei prominente *Heuristiken* im Abschnitt 4.5 vorgestellt und in ihrer Performanz verglichen.

4.1. Heuristiken

Heuristik stammt vom altgriechischen *heuristikein* aus welchem es zu „entdecken“ übersetzt wird und ist eine formalisierte Intuition um den Weg zu einer Lösung zu verkürzen. In der Informatik reduziert man mit ihnen primär die Komplexität von sonst informationsübersättigten und oftmals exponentiellen Problemen, um mit einer Teilmenge der Informationen und Laufzeit optimale Entscheidungen zu treffen [Pac15].

Algorithmen explorieren den Problemraum mit *Heuristiken* zielorientierter [Bla19]. Die Bestrebung für das Spiel der Amazonen ist eine gezieltere Suche, welche das Durchsuchen des gesamten Spielbaumes überflüssig macht bzw. auf eine bestimmte Spielbaumtiefe beschränkt, indem sie eine Abschätzung der voraussichtlichen Distanz zum Ziel liefert. Für die Amazonen ist das Ziel eine gewinnende Spielstellung zu erreichen, hier wird eine *Heuristik* in Form einer Bewertungsfunktion zur Evaluation der Spielsituation benötigt.

Das *heuristische Wissen* ist jedoch unsicher und garantiert keinesfalls eine optimale Abschätzung. *Heuristiken* können von unterschiedlicher Qualität sein [Bla19]. Daher beschäftigt sich dieses Kapitel mit verschiedenen Heuristiken für die Amazonen (siehe Abschnitt 4.5). Die **Bewertungsfunktion** vereint Wissen aus dem Zustand bzw. des Spielfelds und Vorwissen über die Kosten zum Ziel. Herausforderung ist die Filterung der wichtigen Information sowie eine angemessene Gewichtung, um die Spielsituation angemessen einzuschätzen [Fri20].

4.2. Minimax Algorithmus

Der **Minimax Algorithmus** basiert auf dem *Minimax Theorem* aus Abschnitt 3.3. Denn zur Berechnung des *Minimax Wert* einer Spielsituation bedarf es eines Algorithmus, dies gehört zur Zielsetzung des gleichnamigen *Minimax Algorithmus*. Dieser errechnet den *Minimax Wert*, damit den Gewinn für den aufrufenden Spieler (kurz MAX und für die folgenden Kapitel Weiß), sowie den dazugehörigen Zug mithilfe des *Minimax Theorems* [Rei89].

Betrachtet wird eine auf *Tiefensuche* basierende Version des *Minimax Algorithmus*. Diese expandiert die Knoten des Baumes vollständig von links nach rechts [Rei89]. Dabei wird unter rekursiven Aufrufen bis zu einem Blattknoten expandiert und evaluiert, der Elternknoten wählt anschließend den nach *Minimax Prinzip* optimalen Wert seiner bewerteten Nachfolger bis die Wurzel erreicht ist [Bla19].

In der Realität ist das hingegen selten praktikabel (siehe Abschnitt 3.3), da jeder Terminalknoten gefunden und evaluiert werden müsste. Eine Beschneidung des Problemraums kann durch die Beschränkung der Suchtiefe erzielt werden. Möglicherweise terminiert die Expandierung bei einer tiefenlimitierten Suche in einer nicht Finalstellung. Für die Evaluierung kommen die Heuristiken aus Abschnitt 4.1 in Form von Evaluierungsfunktionen zum Einsatz.

Algorithm 1 depthlimited *Minimax Search*

```

1: function MINIMAX(node, depth)                                ▷ Initialer Aufruf: Wurzel, Maximaltiefe
2:   if IS_TERMINAL(node) or depth = 0 then
3:     node.value ← EVALUATE(node)                                ▷ Evaluierung durch heuristische Funktion
4:     return node.value
5:
6:   if node.is_max then                                         ▷ Frage nach maximierenden Spieler
7:     best_score := ∞
8:   else
9:     best_score := -∞
10:
11:   for child in GET_CHILDREN(node) do
12:     score := MINIMAX(child, depth - 1)
13:     if node.is_max then
14:       best_score := MAX(best_score, score)
15:     else
16:       best_score := MIN(best_score, score)
17:
18:   return best_score

```

Der Algorithmus 1 zeigt eine auf [Rei89] aufbauende, umstrukturierte Version einer solchen *Minimax Suche*, die Tiefenlimitierung wird durch Zeile 2 des Algorithmus realisiert.

Die Nutzung von *Heuristiken* ermöglicht zwar eine Beschneidung des Spielbaum bezüglich der Tiefe, der Suchalgorithmus entscheidet indes gänzlich blind gegenüber allem abseits der festgelegten Tiefe liegendem. Ein Algorithmus mit höherer Suchtiefe unter Verwendung der gleichen Heuristik wird also besser spielen, da dieser dem *Horizonteffekt* weniger stark erliegt [Fri20]. Ziel sollte es daher sein, den Spielbaumalgorithmus bestmöglich zu optimieren, um das Tiefenlimit maximal setzen zu können. In dieser Form ist die *worst-case* Laufzeit $O(b^d)$ [RN99, S. 165] (Tiefe d , Verzweigungsgrad b).

4.3. Alpha Beta Pruning

Der folgende Abschnitt betrachtet die wohl wichtigste Optimierung für den *Minimax Algorithmus*, das **Alpha Beta Pruning**, mehrfach entdeckt darunter 1961 durch Edwards D. [EH61] und 1975 verbessert durch Donald Knuth and Ronald W. Moore [KM75]. Es zielt im Rahmen des *Minimax Algorithmus* (siehe Abschnitt 4.2) auf eine Reduzierung der Expansion nicht korrektkeitsbeeinflussender Knoten ab. Jene Knoten sind redundante Betrachtungen, ein Beispiel dafür zeigt Abbildung 4.1.

Auf der linken Seite des Spielbaumes hat der **MAX** Knoten den Wert 10 gefunden, da aber der minimierende Elternknoten bereits den Wert 5 hat und damit weder 10 noch einen für **MAX** möglicherweise besseren Wert wählen wird, kann die Suche an dieser Stelle eingestellt werden (β -*Cutoff*). Analog wird rechts ein α -*Cutoff* dargestellt, hierbei kann unter dem **MIN** Knoten mit dem Wert 1 abgeschnitten werden, da sich der Wert, dem Algorithmus 1 folgend, nur fallen kann und der maximierende Elternknoten mit dem Wert 5 diesen nicht wählen wird.

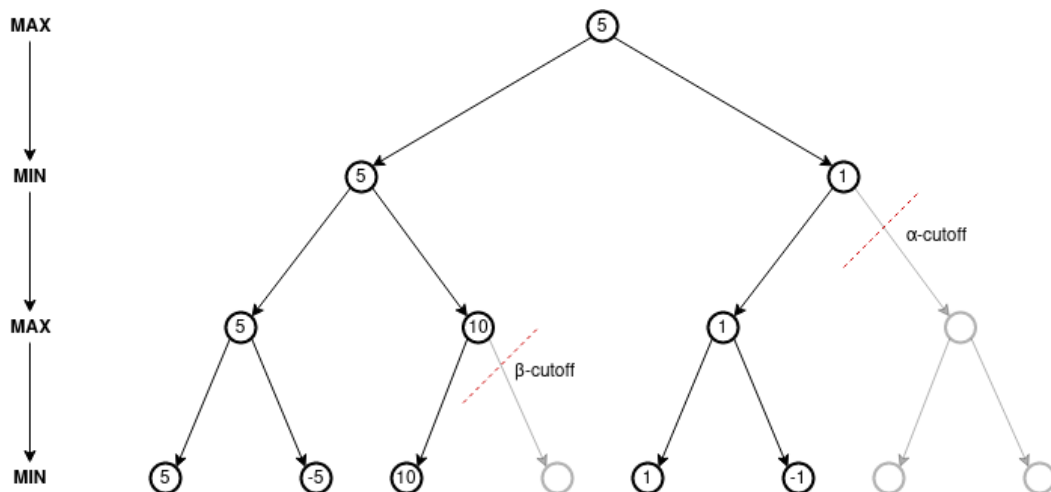


Bild 4.1.: Beispiel für mögliche Beschneidungen des Spielbaumes: Ein β -*Cutoff* beendet die weitere Suche unter einem **MAX** Knoten, wenn der vorhergehende **MIN** Knoten einen höchstens gleichgroßen Wert hat. Der α -*Cutoff* findet unter **MIN** Knoten statt, wenn dieser unbedeutend für den maximierenden Spieler ist.

Intuitiver ist eine Betrachtung der formellen Verschriftlichung der Berechnungsfunktion für die betroffenen Knoten des Beispiels (Abbildung 4.1). Denn es existiert kein $z \in \mathbb{Z}$, für welches (4.1) oder (4.2) nicht zu 5 auswerten, dies unterstreicht Nichtigkeit einer fortgesetzten Suche nach z .

$$\min(5, \max(10, z)) \quad (\beta - \text{Cutoff}) \quad (4.1)$$

$$\max(5, \min(1, z)) \quad (\alpha - \text{Cutoff}) \quad (4.2)$$

Für einen Wert x eines **MAX** Knotens und der Bewertung y des minimierenden Elternknotens ist $x \geq y$ die Bedingung für einen β -Cutoff. Analog für einen Wert x eines **MIN** Knotens und der Bewertung y des maximierenden Elternknotens ist $x \leq y$ die Bedingung für einen α -Cutoff, die weitere Expandierung an diesen Stellen kann eingestellt werden [Fri20].

Die Modifikation des *Minimax Algorithmus* durch eben diese Beschneidungen, nennt sich *Alpha Beta Pruning* oder *Alpha Beta Suche* und wird durch die Integration eines Fensters in die Suche realisiert. Dieses besteht aus einer *unteren Schranke* α und der *oberen Schranke* β , dem kleinsten bzw. größten bekannten Wert aus der Sicht des jeweiligen Elternknotens [Bla19].

Das *Alpha Beta Suchfenster* $[\alpha, \beta]$ wird im Laufe der Suche ständig aktualisiert, eine gute Einschätzung bezüglich eines Initialfensters kann die Suche weiter beschleunigen [SP96], setzt jedoch Kenntnisse und Erfahrungen über die möglichen Extremwerte im Spiel voraus, genauer, die jeweils schlechtesten Werte.

Bei absoluter Unsicherheit bezüglich des Ausgangs werden die Extrema des jeweiligen Zahlenraums genutzt $[-\infty, \infty]$. Im Verlauf der Evaluierung der Kind Knoten wird das Fenster aktualisiert. Der aufrufende Spieler versucht α zu maximieren, der Gegenspieler minimiert β und bei einer Kollision der Schranken wird die Suche an dem Knoten beendet.

Algorithmus 2 zeigt einen durch [SS19] inspirierten Code für das *Alpha Beta Pruning*. Dabei wurde die Funktion um die Werte des Fensters $[\alpha, \beta]$ als einzelne Parameter erweitert (z. 1), diese werden in den Zeilen 9 bzw. 13 aktualisiert und bestimmen in Zeile 10 und 14 ob die Suche eingestellt, der Spielbaum also beschnitten wird oder nicht.

Die Geschwindigkeit der Verkleinerung des *Alpha Beta Suchfensters* und damit zum Beschnitt hängt stark von der Struktur des Spielbaumes ab. So spielt die Reihenfolge in welcher Knoten expandiert werden eine erhebliche Rolle [MC82]. Optimierungen, welche etwa Vorwissen in die Entscheidung der Suchreihenfolge einbauen fallen dabei meist dem Speicher zur Last, da sie Tabellen zur Speicherung nutzen.

Donald E. Knuth beweist für die *Alpha Beta Suche* eine *best case* Komplexität $b^{d/2} + b^{b/2} - 1$ (in dieser Arbeit mit b als Verzweigungsgrad und d als Tiefe) oder als O-Notation $O(b^{d/2})$ [KM75]. Im *worst case* muss wie beim *Minimax Algorithmus* der gesamte Spielbaum bis zur Tiefe d durchsucht werden und teilt daher die Laufzeit $O(b^d)$ [RN99, S. 169]. Im *best case* könnte also in der selben Zeit doppelt so tief gesucht werden als es bei der *Minimax Suche* möglich ist, die der Praxis am naheliegendste Laufzeit wird sich dazwischen bewegen.

Algorithm 2 *Alpha Beta Pruning*

```

1: function ALPHABETA(node, depth,  $\alpha$ ,  $\beta$ )  $\triangleright$  Initialer Aufruf: Wurzel, Maximaltiefe,  $-\infty$ ,  $\infty$ 
2:   if IS_TERMINAL(node) or depth = 0 then
3:     node.value  $\leftarrow$  EVALUATE(node)  $\triangleright$  Evaluierung durch heuristische Funktion
4:     return node.value
5:   for child in GET_CHILDREN(node) do
6:     score := ALPHABETA(child, depth - 1,  $\alpha$ ,  $\beta$ )
7:     if node.is_max then
8:        $\alpha$  := MAX( $\alpha$ , score)
9:       if score  $\geq$   $\beta$  then  $\triangleright \beta - \text{Cutoff}$ 
10:        break
11:     else
12:        $\beta$  := MIN( $\beta$ , score)
13:       if score  $\leq$   $\alpha$  then  $\triangleright \alpha - \text{Cutoff}$ 
14:        break
15:   return score

```

4.4. Implementierung

Die *heuristische Suche* im Kontext des Spiels der Amazonen erfolgt durch die in Abschnitt 4.3 eingeführte *Alpha Beta Suche*. Im Verlauf dieses Kapitels werden weitere anwendungsspezifische Optimierungen vorgestellt und angewendet.

Zustände bzw. Knoten im zu durchsuchenden Spielbaum sind Spielstellungen der Amazonen. Dies führt bei einer planlosen Implementierung zu Laufzeit oder Speicherproblemen. Neben einer berechnungseffizienten Kodierung des Spielbrettes zum Zweck der Laufzeitoptimierung [Fri20], lässt sich der Speicher durch das Arbeiten auf einem einzelnen Brett entlasten [Bla19]. Der Zug wird vor der Suche auf das Brett angewendet und anschließend zurückgenommen, damit entfällt allerdings die Möglichkeit einer parallelisierten Suche. Im Gegenzug wird die Suchtiefe für den Speicher beinahe unerheblich, es werden ausschließlich die Züge gespeichert.

Die Suchtiefe nimmt exponentiellen Einfluss auf die Laufzeit. Von dem durchschnittlichen Verzweigungsgrad 400 ausgehend [Kat+15], sprengt eine Tiefe von 3 bereits den Rahmen mit $400^3 \approx 64000000$ zu untersuchenden Zuständen. Eine maximale Suchtiefe von 2 scheint angemessen. Jedoch nimmt der Verzweigungsgrad im Verlauf des Spiels ab [Lie05]. Daher lässt sich die Suchtiefe mit Fortschreiten des Spiels zugunsten der Optimalität anheben, ohne die Laufzeit zu erhöhen. Die ausgewogene Wahl der Parameter stellt jedoch ein gänzlich neues Optimierungsproblem dar und ist in der Implementierung daher mit experimentellen Werten gelöst worden.

Algorithm 3 Alpha Beta Pruning for the Game of the Amazons

```

1: function GET_AB_MOVE(board)                                ▷ Initialer Aufruf: Aktuelles Brett
2:   moves := GET_MOVES(board)
3:   depth := DETERMINE_DEPTH(moves)                        ▷ Legt eine angemessene Tiefe fest
4:   best_move := null
5:   best_score :=  $-\infty$ 
6:
7:   for move in moves do
8:     board  $\leftarrow$  DO_MOVE(board, move)                ▷ führe Zug aus
9:     score := ALPHABETA(board, depth, best_score,  $\infty$ , False)
10:    board  $\leftarrow$  UNDO_MOVE(board, move)                ▷ Zug zurücknehmen
11:    if score > best_score then                            ▷ updatebestmove
12:      best_move  $\leftarrow$  move
13:      best_score  $\leftarrow$  score
14:    return best_move
15:
16: function ALPHABETA(board, depth,  $\alpha$ ,  $\beta$ , maximizing)
17: if is_won(board) or depth = 0 then
18:   return EVALUATE(node)                                ▷ Evaluierung durch heuristische Funktion
19:
20: for move in GET_MOVES(board) do
21:
22:   board  $\leftarrow$  DO_MOVE(board, move)                ▷ führe Zug aus
23:   score := ALPHABETA(board, depth - 1,  $\alpha$ ,  $\beta$ , not maximizing)
24:   board  $\leftarrow$  UNDO_MOVE(board, move)                ▷ Zug zurücknehmen
25:
26:   if maximizing then
27:      $\alpha$  := MAX( $\alpha$ , score)
28:     if score >=  $\beta$  then                                ▷  $\beta$  - Cutoff
29:       break
30:   else
31:      $\beta$  := MIN( $\beta$ , score)
32:     if score <=  $\alpha$  then                                ▷  $\alpha$  - Cutoff
33:       break
34:   return score

```

Der **Algorithmus 3** stellt die *Alpha Beta Suche* für das Spiel der Amazonen dar. Eingabe ist die kodierte Spielsituation *board* und Rückgabe der beste Zug nach *minimax Strategie*. Dabei wird in Zeile 9 erstmals ALPHABETA als maximierender Spieler aufgerufen und alterniert durch die Variable *maximizing*. Die genaue Form der Funktion EVALUATE (z. 18) wird im nächsten Abschnitt thematisiert.

4.5. Heuristiken für das Spiel der Amazonen

Wie bereits John McCarthy 1955 auf der Dartmouth Conference beschrieb „We humans are not very good at identifying the heuristics we ourselves use.“ [Min+06], vermögen die meisten Menschen schnell eigene Spielintuitionen zu entwickeln, diese zu formalisieren ist indes eine Herausforderung.

Eine Heuristik für das Spiel der Amazonen sollte neben einer möglichst guten Abschätzung auch in angemessener Laufzeit berechnet werden können. Da die `EVALUATE` (Zeile 18) Funktion neben der Zuggeneration `GET_MOVES` (Zeile 20) die kritischste Operation des Algorithmus 3 ist und die Laufzeit maßgeblich beeinflusst.

4.5.1. Mobilitäts Evaluation

Das Ziel im Spiel der Amazonen ist die Immobilisierung der Amazonen des Gegenspielers unter Gewährleistung der Zugfähigkeit der eigenen Figuren. Beim Ziehen möglichst die Zugmöglichkeiten des Gegenspieler einzuschränken und der eigenen Einschränkung entgegenzuwirken ist daher eine zielführende Strategie.

Die Mobilität der Amazonen kann durch die Anzahl erlaubter Züge des Spielers im Verhältnis zum Gegenspieler eingeschätzt werden. Die *Mobilitäts Evaluation* (H_M) aus der Sicht des Spielers Z_1 errechnet sich aus der Differenz der Anzahl möglicher Züge von Z_1 und der des Gegenspielers Z_2 [Kat+15].

$$H_M = Z_1 - Z_2 \quad (4.3)$$

H_M ist eine simple *Heuristik*, welche zur Implementierung keiner weiteren Raffinessen bedarf, da sie einer Vereinfachung des Zuggenerators entspricht. Hingegen ist die sie blind gegenüber spielspezifischer Vorgänge, weshalb sie gegen erfahrene Gegenspieler nicht performant spielt [Kat+15]. Spielmechaniken wie etwa das Abstecken von Territorien in der mittleren Phase des Spiels, werden gänzlich ausgeblendet, sodass die KI sich, der im Zentrum vorhandenen erweiterten Zugmöglichkeiten geschuldet, tendenziell zentral und damit mitunter leicht angreifbar positioniert, sowie das umliegende Gebiet freigibt.

Für das folgenden Beispiel (siehe Abbildung 4.2) werden, um die Visualisierung überschaubar zu halten, die Züge in der ersten Ebene nur durch die Position der Amazonen unterschieden. Die anschließend für den Pfeil erreichbaren Felder werden durch grau unterlegte Felder angedeutet, in welchem jeweils H_M für den dazugehörigen Zug eingetragen ist.

Der Zug **c2-c1-b1** (für die Notation siehe Abschnitt 2.3) wird beispielsweise mit $6 - 6 = 0$ bewertet. Demnach sind der *Mobilitäts Evaluation* zufolge **c2-b2-b1** und **c2-b1-a2** die bestmöglichen Züge. Während **c2-b2-b1** (a), dem Verlauf auf der linken Seite des Graphen folgend, zu einem sicheren Sieg des Spieler führt, ist **c2-b1-a2** (b) eine sichere Niederlage und damit ein Beispiel für die Naivität der Heuristik.

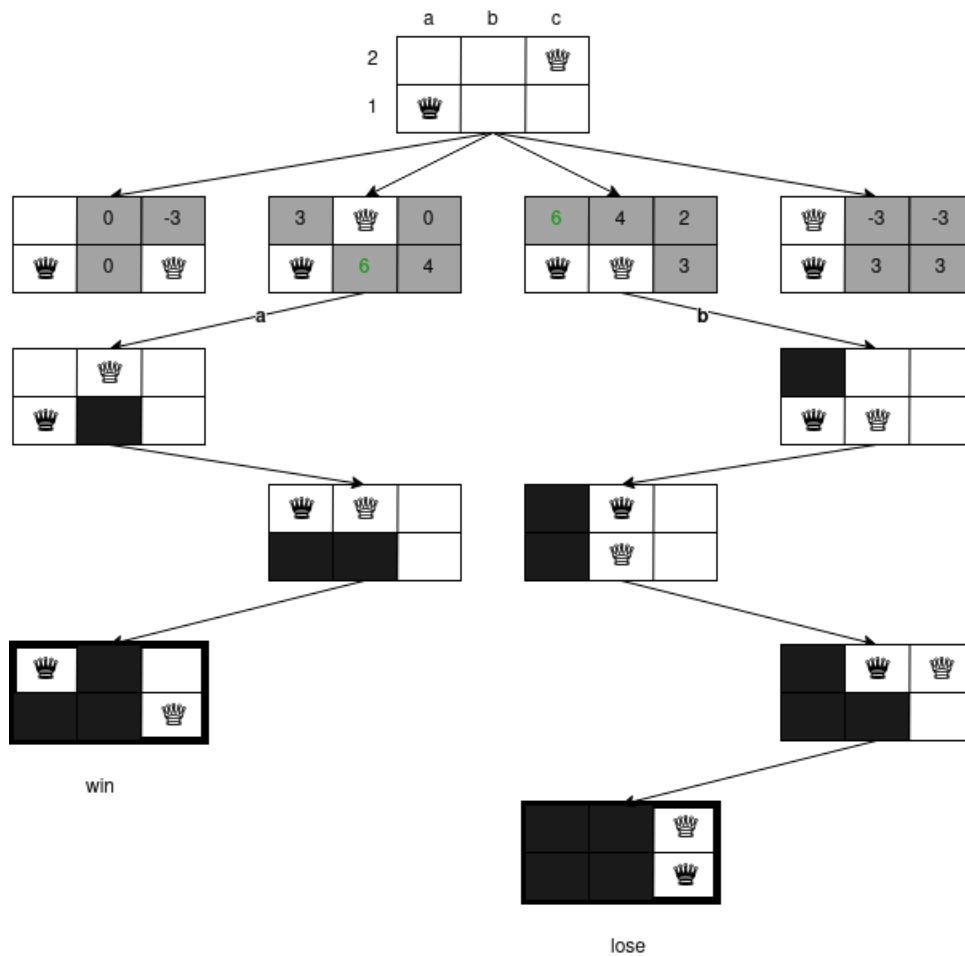


Bild 4.2.: *Mobilitäts Evaluation* auf ein 2x3 Spiel der Amazonen angewendet. Die erste Ebene stellt die *heuristischen Werte* der jeweiligen Züge dar.

4.5.2. Territoriale und lokale Evaluation

Ein weiterer Ansatz einer Bewertung, neben der Mobilität, erschließt sich durch das in das Spiel des Amazonen verankerte Konzept der *Territorien*. Die *Füllphase* (siehe Kapitel 2) bzw. das Spiel gewinnt der Spieler, welcher den exklusiven Zugriff auf die Mehrheit der freien Felder für sich beanspruchen kann. Die Erreichbarkeit einer hohen Anzahl von Feldern ist vorteilhaft um die Größe der eigenen, im Verlauf des Spiels entstehenden, *Territorien* zu maximieren [Kat+15].

Die Erreichbarkeit vollständig oder nur von einem Spieler isolierter freier Felder lässt sich eindeutig bestimmen. Für jeden erreichbare Felder bedürfen indes einer *heuristischen* Abschätzung. Ziel ist eine Bewertung des aktuellen und potentiellen Territoriums des Spieler bzw. die Differenz der beider Bewertungen. Die folgende Evaluation und ihre dazugehörigen Definitionen entspringen primär der Arbeit von Jens Lieberum zu seinem Program AMAZONG [Lie04].

Definition 4.5.1 (Distanz $d_i(a,b)$, $i \in \{\mathbf{q}, \mathbf{k}\}$). *Minimale Sequenz von Schach Königinnen (\mathbf{q}) bzw. Schach Königs (\mathbf{k}) Zügen um von Feld a zu Feld b zu gelangen, isolierte Felder haben die Distanz $d_i(a,b) = \infty$.*

Definition 4.5.2 (Distanz $D_i^j(a)$, $i \in \{\mathbf{q}, \mathbf{k}\}$, $j \in \{1, 2\}$). *Minimale Distanzen des Spielers j zu einem Feld a sind definiert wie folgt:*

$$D_i^j(a) = \text{MIN}(d_i(a,b) \mid b \text{ ist ein durch Spieler } j \text{ Amazone belegtes Feld}) \quad (4.4)$$

Die Distanz $D_i^j(a)$ (siehe Definition 4.5.2) ist die kürzeste Distanz des Spieler j zu einem Feld a durch die jeweilige Zugform i (siehe Definition 4.5.1). In anderen Arbeiten [Kat+15] wird diese Distanz ausschließlich durch $D_{\mathbf{q}}^j(a)$ bemessen. Eine geringere Distanz ermöglicht eine Erweiterung des Territoriums in potenziell weniger Zügen, daher wird dem Spieler mit der geringsten Distanz ein Vorteil angerechnet. Dieser kontrolliert das Feld.

Während $D_{\mathbf{q}}^j(a)$ in der mittleren bis Endphase gute Einschätzungen liefert, ist die Anfangsphase problematisch. Zu Beginn haben Figuren im Zentrum des Spielfeldes global die geringste Distanz, jedoch überwiegt in dem Fall die Gefahr eingebaut zu werden, der Wahrscheinlichkeit die hohe Anzahl an Feldern für sich beanspruchen zu können. Um die Werte am Spielbeginn zu stabilisieren, konzentriert sich $D_{\mathbf{k}}^j(a)$ auf die Reduzierung des positionellen Schadens, eine Sichtweise welche bessere Langzeiteinschätzungen liefert als $D_{\mathbf{q}}^j(a)$ [Lie04].

Ein Beispiel der minimalen Abstände $D_{\mathbf{q}}^j(a)$ (links) und $D_{\mathbf{k}}^j(a)$ (rechts) zeigt Abbildung 4.3. Für jedes freie Feld a ist jeweils $D_{\mathbf{q}}^1(a)$ oben links und $D_{\mathbf{k}}^2(a)$ unten rechts eingezeichnet, wobei Spieler 1 Weiß spielt und 2 Schwarz. Demnach ist $D_{\mathbf{q}}^1(\mathbf{a1}) = \infty$, $\mathbf{a1}$ ist unerreichbar für Weiß und $D_{\mathbf{q}}^2(\mathbf{a1}) = 2$ bzw. $D_{\mathbf{k}}^2(\mathbf{a1}) = 5$.

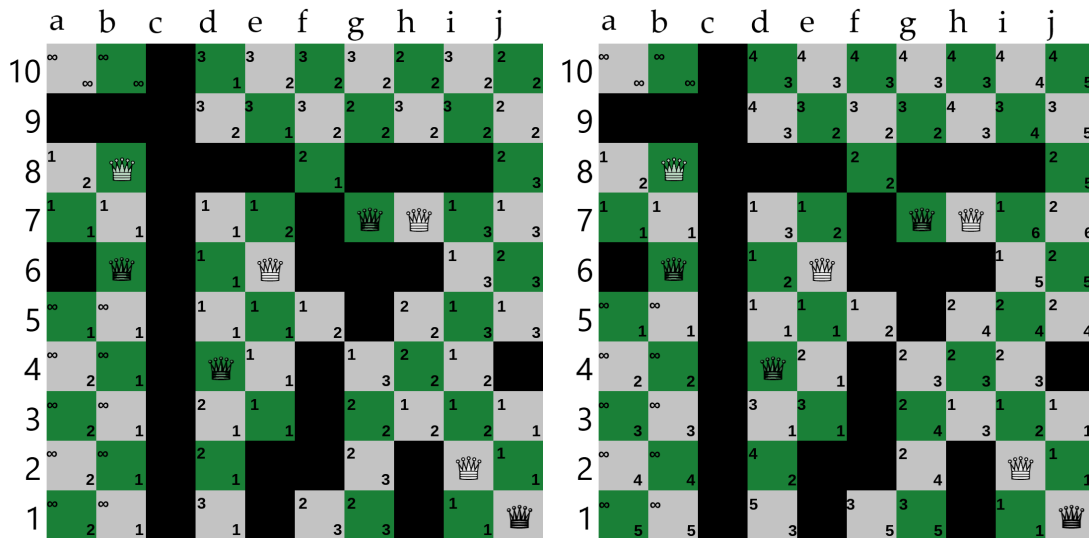


Bild 4.3.: Distanzen $D_i^j(a)$. $D_{\mathbf{q}}^j(a)$ links, $D_{\mathbf{k}}^j(a)$ rechts.

Die *territoriale Evaluationen* t_q und t_k sind wie folgt definiert:

$$t_i = \sum_{\text{freie Felder } a} \Delta(D_i^1(a), D_i^2(a)) \quad (4.5)$$

Wobei die Funktion Δ (4.6) jedem Feld einen Wert des Bereichs $[-1,1]$ zuweist. Der Wert $\frac{1}{5}$ ist ein Zugeständnis für Spieler 1, im Fall eines Gleichgewichts der Distanz Spieler [Lie04].

$$\Delta(n, m) = \begin{cases} 0 & n = m = \infty \\ \frac{1}{5} & n = m \neq \infty \\ 1 & n < m \\ -1 & n > m. \end{cases} \quad (4.6)$$

Die Kombination der *territorialen Evaluationen* (4.5) stabilisiert die Defizite der jeweils einzelnen t_i in den unterschiedlichen Spielphasen. Die Normalisierung der Distanzen über $[-1,1]$ in Δ (4.6) desensibilisiert t_i gegenüber zu Spielbeginn hoher Distanzdifferenzen. Die folgenden *lokalen Evaluationen* c_1 (4.7) und c_2 (4.8) wirken dem entgegen [Lie04]. Die *lokale Evaluation* c_1 belohnt *lokale Vorteile* und gleicht damit die Gleichgültigkeit von t_1 in den ersten Phasen aus, im Verlauf des Spiels verliert c_1 an Gewichtung. Da t_2 bereits den *positionellen Schaden* betrachtet, werden durch c_2 nur hohe Distanzdifferenzen zusätzlich gewichtet.

$$c_1 = 2 \sum_{\text{freie Felder } a} 2^{D_q^1(a)} - 2^{D_q^2(a)} \quad (4.7)$$

$$c_2 = \sum_{\text{freie Felder } a} \text{MIN}(1, \text{MAX}(-1, \frac{D_k^2(a) - D_k^1(a)}{6})) \quad (4.8)$$

Eine *Evaluationsfunktion* H_{TP} in welcher t_1 , t_2 , c_1 , c_2 eine der Spielphase entsprechende Gewichtung erhalten, könnte wie folgt aussehen:

$$H_{TP} = f_1(x)t_1 + f_2(x)c_1 + f_3(x)t_2 + f_4(x)c_2 \quad (4.9)$$

Wobei $\sum_{i=1}^4 f_i(x) = 1$ und eine dynamische Gewichtung der Summe im Spielverlauf vorgenommen wird. Die optimale Definition von f_i ist ein eigenes Optimierungsproblem [Lie04]. Im Rahmen dieser Arbeit wurde f_i mit $i \in \{1, 2, 3, 4\}$ und der Spielfortschritt x als Anteil belegter Felder in folgender experimenteller Form genutzt:

$$f_1(x) = vx \quad v \in [0, 1]; v = 0.7 \text{ in [Yan21]} \quad (4.10)$$

$$f_2(x) = (1 - v)x \quad (4.11)$$

$$f_3(x) = v(1 - x) \quad (4.12)$$

$$f_4(x) = (1 - v)(1 - x). \quad (4.13)$$

4.5.3. Performanz

Ein Vergleich der Performanz der im Abschnitt 4.5.1 und 4.5.2 eingeführten *Heuristiken* kann über Laufzeit, Speicherverbrauch sowie Optimalität bzw. spieltheoretischer Präzision erfolgen. Innerhalb des Programms [Yan21] werden für keine der *Heuristiken* größere Datenstrukturen generiert, daher liegt der Fokus auf Laufzeit und Optimalität.

Im ersten Zug hat der Spielbaum den höchsten Verzweigungsgrad (siehe Abschnitt 2.4). Eine Betrachtung der Laufzeit erfolgt in Abbildung 4.4 durch die Evaluierung der Initialstellung für Weiß. Während die Laufzeit für H_M mit wachsender Spielfeldgröße $n \times n$ linear steigt und primär durch das Wachstum der Anzahl an Spielfiguren ab $n = 8$ beeinflusst wird (siehe Anhang A.1), weist die Laufzeit von H_{TP} ein kubisches bis exponentielles Wachstum über n^2 auf.

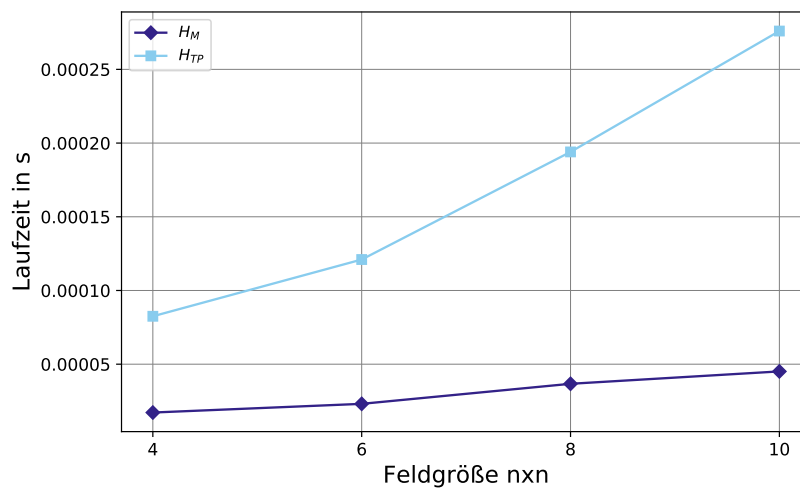


Bild 4.4.: Laufzeit von H_M und H_{TP} auf 4x4, 6x6, 8x8 und 10x10 Initialfeldern.

Die *Alpha Beta Suche* (siehe Algorithmus 3) wurde unter Verzicht randomisierter Methoden implementiert und spielt mit unbegrenztem Zeitkontingent *deterministisch*. Tabelle 4.1 stellt die siegende *Heuristik* H_i auf unterschiedlichen Spielfeldgrößen dar. H_{TP} spielt wesentlich besser als H_H und gewinnt auch als benachteiligter Spieler Schwarz (siehe Absatz 2.2). Die *Heuristik* H_{TP} ermöglicht genauere Evaluationen zu Lasten der Laufzeit. Anzumerken ist, dass die Ergebnisse nach Einführung eines Zeitkontos t mit $t \geq 1s$ unverändert bleiben. Demnach ist H_{TP} unabhängig von verfügbaren Zeitressourcen die bessere Evaluation für das Spiel der Amazonen.

| Heuristik | | Gewinner auf Spielfeldgröße $n \times n$ | | | |
|-----------|----------|--|----------|----------|----------|
| Weiß | Schwarz | 4x4 | 6x6 | 8x8 | 10x10 |
| H_M | H_{TP} | H_M | H_{TP} | H_{TP} | H_{TP} |
| H_{TP} | H_M | H_{TP} | H_{TP} | H_{TP} | H_{TP} |

Tabelle 4.1.: Gewinner diverser *Alpha Beta Pruning* Spielkonstellationen, Suchtiefe 3.

5. Stochastische Suche

Besonders für Spiele ist es schwer eine geeignete Heuristik zu finden. Zwar konnten im Rahmen des *Alpha Beta Prunings* zahlreiche Erfolge im Schach oder Dame erzielt werden, für Spiele wie Go ist es jedoch schwer eine adäquate Evaluationsfunktion zu finden [Cha+08].

Eine weitere Möglichkeit der zielgerichteten Suche auf dem Spielbaum ist die **stochastische** oder **randomisierte Suche** durch eine Kombination von *Monte Carlo (MC)* und *Upper Confidence Bound (UCB)* Methoden. *Monte Carlo Tree Search (MCTS)* hat durch den Sieg der Go KI *CrazyStone*, auf der Computer Olympiade 2006 [CC06], einen Durchbruch in der Spieltheorie erlebt. In diesem Kapitel werden zwei Algorithmen aus der Familie der *MCTS* Algorithmen eingeführt und ihre Performanz im Spiel der Amazonen verglichen.

5.1. Monte Carlo Tree Search

Ein **Monte Carlo Algorithmus** gewinnt durch wiederholte randomisierte Entscheidungen an Genauigkeit. Eine erhöhte Laufzeit verringert die Fehlerrate [MR95, S. 9]. Es werden Fehler zugunsten der Optimalität toleriert. Dies kann besonders effizient für Probleme mit exorbitanten Problemraum sein, welche durch deterministische Algorithmen nicht effektiv gelöst werden können [Kat+15].

Das Konzept der Baumsuche *TS* wird in einem *MCTS* Algorithmus durch andere Paradigmen als bei der *Minimax Suche* realisiert (siehe Abbildung 5.1). Dieses sind *Selektion*, *Expansion*, *Simulation* und *Backpropagation* [Klo10]. Zudem enthält jeder Knoten nebst Spielfeld, Informationen über die Gewinne R und Anzahl von Besuchen N während der Suche.

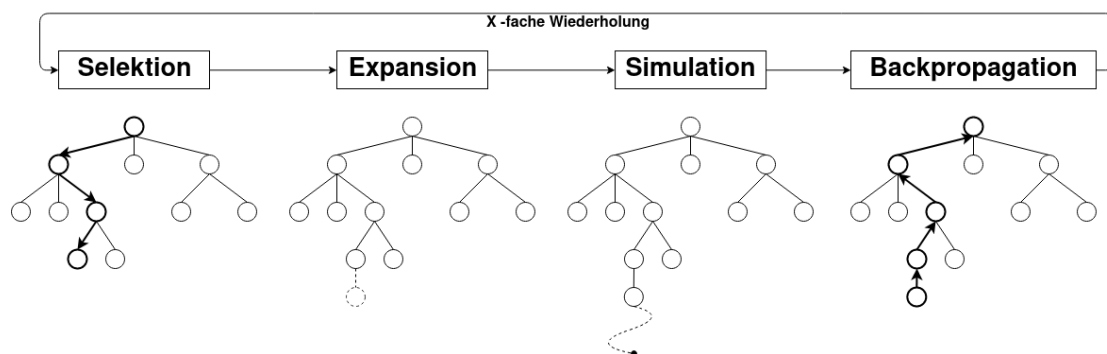


Bild 5.1.: Vier Phasen eines Monte Carlo Tree Search Algorithmus nach [Cha+08]

Selektion

Die optimale **Selektion** eines Kindknotens bestimmt die Qualität des Spielzugs [Cha+08]. Der Spielbaum wird von der Wurzel ausgehend *besten-orientiert* expandiert, der Knoten mit besten Wert wird ausgewählt und untersucht. Jedoch kann die konsequente Wahl des besten Knoten die Entdeckung potenziell besserer, auf suboptimale Knoten folgende, Alternativen verhindern. Dieses *exploration-exploitation dilemma* lösen Kocsis und Szepesvári durch die Einführung von *UCB1* (siehe Formel 5.1) für den Spielbaum bzw. *UCT (Upper Confidence Bound applied to Trees)* [KS06]. Ein *UCB* Algorithmus löst jenes Dilemma für das einfachere *k armed Bandit Problem*, die Wahl des gewinnmaximierenden Automaten aus einer Reihe von *k* Automaten. Ein Spieler kann den Automaten mit der höchsten Gewinnrate ausnutzen oder einen möglicherweise besseren Automaten ausprobieren. Es wird der Automat mit dem höchsten *upper confidence bound* gespielt [ACF02]. Die folgenden *MCTS* Algorithmen wählen auf dem Spielbaum, einer *UCT Strategie* folgend, stets den Knoten mit höchstem *UCB Wert*. Die gewählte *UCB* Methode wird jeweils im Zusammenhang mit der jeweiligen *MCTS* Variante erläutert.

Expansion

Die **Expansion** eines neuen Kindknoten geschieht, sobald die Selektion an einen Knoten mit noch nicht expandierten Kindern gelangt. Dem Spielbaum wird dabei ein neuer Blattknoten pro Simulation hinzugefügt [Cha+08].

Simulation

Während der **Simulation**, auch *rollout* [KS06] oder *playout* [Cha+08], spielt der Algorithmus bis zu einer Terminalstellung gegen sich selbst. Die gespielten Züge können randomisiert oder einer *Strategie* bzw. *policy* entsprechend gewählt werden. Diese kann die Fehlerrate reduzieren, bedarf indes zusätzlichen Wissens über das Spiel und zusätzlicher Rechenzeit [Cha+08].

Am Ende der Simulation erfolgt eine Bewertung des simulierten Knotens. Ein Sieg wird, Chaslot et al. [Cha+08] folgend, mit $r_x = +1$ und eine Niederlage mit $r_x = -1$ bewertet (r_x als Ergebnis der x -ten Simulation).

Backpropagation

Die Phase der **Backpropagation** dient der *Propagierung* des am Ende der Simulation erhaltenen Wertes r_x entlang des rekursiv traversierten Pfades bis zur Wurzel. Dabei wird r_x ebenenweise alternierend auf r des betroffenen Knoten addiert und n inkrementiert. Ziel der *Backpropagation* ist die Aktualisierung der Durchschnittsgewinnraten $\frac{r}{n}$ im Spielbaum. Diese können die *Selektion* im nächsten Zyklus verbessern. Neben der durchschnittlichen Gewinnrate gibt es weitere Strategien der *Backpropagation*, jene sind jedoch instabiler [Cha+08].

Die Wiederholung des Zyklus aktualisiert die Gewinnraten im Spielbaum, anschließend lässt sich die beste Aktion der ersten Ebene der Datenstruktur entnehmen.

5.2. UCT Algorithmus

Der UCT Algorithmus nach Kocsis und Szepesvári nutzt *UCB1* (5.1) als Strategie. Sie beweisen zudem eine Konvergenz zum *Minimax Algorithmus* (siehe Abschnitt 4.2) [Bro+12, S. 7]. Dabei konvergiert der UCT Algorithmus ähnlich wie die *Alpha Beta Suche* (siehe Abschnitt 4.3) in einer Ordnung von $b^{d/2}$ Iterationen [KS06]. Der Term \bar{x}_i mit $\bar{x}_i = \frac{r_i}{n_i}$ steht für die Gewinnrate des Kindknotens i . Der Ausdruck $\sqrt{\frac{2 \log n_I}{n_i}}$ ist die anteilige Erfahrung von i im Verhältnis zum Elternknoten I , C gewichtet damit die Explorationsbereitschaft.

$$UCT1(I) = \operatorname{argmax}_{\text{child } i \in \text{parent } I} \bar{x}_i + C * \sqrt{\frac{2 \log n_I}{n_i}} \quad (5.1)$$

Zwar dominiert UCT nach [KS06] die Familie der MCTS Algorithmen für Spiele (vgl. [Klo10]), im Rahmen dieser Arbeit wird jedoch die experimentelle *UCB1-tuned* (5.2) Auer et al. [ACF02] folgend genutzt. Diese spielt in der Praxis erfahrungsgemäß, jedoch ohne theoretische Beweise besser als *UCB1*. *UCB1-tuned* beschränkt die Varianz der Explorationsbereitschaft.

$$UCT1\text{-tuned}(I) = \operatorname{argmax}_{\text{child } i \in \text{parent } I} \bar{x}_i + \sqrt{\frac{\log n_I}{n_i} \min\left(\frac{1}{4}, \bar{x}_i - \bar{x}_i^2 + \sqrt{\frac{2 \log n_I}{n_i}}\right)} \quad (5.2)$$

Algorithmus 4 ist ein [Bro+12] und [Yan21] nachempfunder Pseudocode für einen UCT Algorithmus, angepasst auf das Spiel der Amazonen und der *UCB1-tuned* Methode (5.2). Die im Abschnitt 5.1 beschriebenen Phasen der MCTS lassen sich in Form designierter Funktionen wiederfinden. Ein Knoten v erhält bei der Initialisierung (z.2, z.21) folgende Attribute:

- | | |
|------------------------------------|---|
| 1. r (Gewinn) | 4. untried_actions (Liste ungenutzter Züge) |
| 2. n (Besuche) | 5. move (letzter Zug) |
| 3. children (Liste mit Kindknoten) | 6. parent (Elternknoten) |

Um den Speicher zu schonen wird nur ein Spielfeld dauerhaft gespeichert, dennoch wächst mit der Anzahl der Simulationen x (z.3) die Anzahl initialisierter Knoten (z.21) und damit der Speicherverbrauch. Der Algorithmus profitiert primär von der Unabhängigkeit gegenüber jeglicher Heuristiken (*aheuristisch*) und einer ständigen Terminierungsbereitschaft, da nach jedem Zug der gesamte Spielbaum aktualisiert wird (z.32) [Bro+12]. Im weiteren Verlauf der Arbeit wird diese UCT Variante als *UCT_{vanilla}* referenziert.

Algorithm 4 UCT for the Game of the Amazons

```

1: function UCB_SEARCH(board, x)  ▷ Initialer Aufruf: Ausgangsstellung, #Simulationen
2:    $v_0 := \text{INIT}(\textit{board})$ 
3:   for  $\textit{sim} \leftarrow 0$  to  $\textit{x}$  do
4:      $v_x \leftarrow \text{TREEPOLICY}(v_0, \textit{board})$ 
5:      $\Delta \leftarrow \text{ROLLOUT}(v_0, \textit{board})$ 
6:      $\text{BACKPROPAGATE}(v_l, \Delta, \textit{board})$ 
7:   return UCB1-TUNED( $v_0$ )
8:
9: function TREEPOLICY(v, board)  ▷ Phase im Zyklus: Selektion
10:  while not IS_WON(board) do
11:    if v.untried_actions = null then
12:      return EXPAND(v, board)
13:    else
14:       $v \leftarrow \text{UCB1-TUNED}(v)$   ▷ UCT Strategie
15:       $\textit{board} \leftarrow \text{DO\_MOVE}(\textit{board}, v.\textit{move})$   ▷ Dem Pfad folgend ziehen
16:    return (v)
17:
18: function EXPAND(v, board)  ▷ Phase im Zyklus: Expansion
19:   $\textit{action} := \text{POP}(v.\textit{untried\_actions})$ 
20:   $\textit{board} \leftarrow \text{DO\_MOVE}(\textit{board}, \textit{action})$ 
21:   $\textit{child} \leftarrow \text{INIT}(\textit{board})$ 
22:  APPEND(v.children, child)
23:  return (child)
24:
25: function ROLLOUT(v, board)  ▷ Phase im Zyklus: Simulation
26:   $\textit{board\_copy} := \text{COPY}(\textit{board})$ 
27:  while not IS_WON(board_copy) do
28:     $\textit{move} \leftarrow \text{SELECT\_RANDOM\_MOVE}(\textit{board\_copy})$ 
29:     $\textit{board\_copy} \leftarrow \text{DO\_MOVE}(\textit{board\_copy}, \textit{move})$ 
30:  return reward for board_copy
31:
32: function BACKPROPAGATE(v,  $\Delta$ , board)  ▷ Phase im Zyklus: Backpropagation
33:  while  $v \neq \textit{null}$  do
34:     $v.n \leftarrow v.n + 1$ 
35:     $v.r \leftarrow v.r + \Delta$ 
36:     $\Delta \leftarrow -\Delta$ 
37:     $\textit{board} \leftarrow \text{UNDO\_MOVE}(\textit{board}, v.\textit{move})$   ▷ Züge des Pfads zurücknehmen
38:     $v \leftarrow v.\textit{parent}$ 
39:

```

5.3. Optimierter UCT Algorithmus

Auf das Spiel der Amazonen zugeschnittene **Optimierungen** der *Monte Carlo Tree Search* bzw. *UCT*, stellen Kloetzer et al. in [KIB07] im Zusammenhang ihres Programms *CAMPYA* vor. In diesem Abschnitt werden speziell zwei Optimierung auf den *UCT* Algorithmus aus Abschnitt 5.2 angewendet. Zum einen wird das randomisierte Spiel durch Regeln verbessert, zum anderen die Spielbaumstruktur auf die Umstände der Amazonen angepasst.

5.3.1. Pseudo-randomisiertes Spiel

Die Abwesenheit jeglichen Vorwissens bei der Suche ist ein substantielle Charakteristik des *MCTS* Algorithmus. Andererseits erlaubt die Zugabe anwendungsspezifischen Wissens, beispielsweise in Form von Regeln, realistischeren Simulationen bzw. die Erwidern verbesserter Züge auf strategisch stärkere Gegenspieler [KIB07]. Jene Regeln basieren auf *Mobilität* und dem *Territorium*, den beiden Kernkonzepten des Spiels der Amazonen. Die eigene Implementation [Yan21] setzt diese Regeln zudem für eine verbesserte Expansion ein.

Mobilität

Die *Mobilität* ist ein Gewinn beeinflussender Faktor im Spiel der Amazonen. Eine erhöhte Anzahl möglicher Züge verhindert eine Zugunfähigkeit und ultimativ die Niederlage (siehe Abschnitt 4.5.1). *CAMPYA* setzt insbesondere zwei Regeln [KIB07] während der Simulation um:

1. Jede Amazone mit 1 oder 2 erreichbaren Feldern muss bewegt werden
2. Jede Amazone des Gegenspielers mit 1 oder 2 erreichbaren Feldern wird eingeschlossen

Unter diesen Regeln werden die eigenen schlechten Züge verhindert und die des Gegenspielers bestraft. 2 ist die kritische Anzahl freier Felder, da die Amazone unter Umständen (Adjazenz) in einem Zug vollständig immobilisiert werden kann.

Territorium

Die Bedeutsamkeit möglichst großer Territorien wurde bereits in Abschnitt 4.5.2 eingehend erläutert, jeder Zug einer isolierten Amazone verringert die Anzahl freier Felder im Territorium. Das Ziehen einer bereits isolierten Amazone ohne Zugzwang ist demnach eine Verschwendung sicherer Felder. Die folgende Regel [KIB07] verhindert das Verschenken von Zügen:

1. Eine isolierte Amazone sollte möglichst nicht gespielt werden

Isolierte Amazonen werden erst gespielt, sobald jede Amazone ein Territorium beansprucht hat. Zu diesem Zeitpunkt kann kein Einfluss mehr auf die Strategie des Gegenspielers ausgeübt werden.

5.3.2. Differenzierte Zugbetrachtung

Durch die *pseudo-randomisierung* (siehe Abschnitt 5.3.1) wächst die spielerische Performanz, doch mit ihr auch die Laufzeit. Dies hat weniger Simulationen in der gleichen Zeit zur Folge und schmälert die gewonnene Spielfähigkeit bzw. Genauigkeit. Kloetzer et al. [KIB07] suchen den Schlüssel zu einer erhöhten Simulationsrate in der Struktur des Spielbaums.

Der in Abschnitt 3.2 eingeführte Spielbaum betrachtet die Kombination aus *Amazonenbewegung+Pfeilschuss* als einen Zug, diese kann indes ebenfalls als zwei voneinander getrennte Aktionen interpretiert werden. Eine *MCTS* auf einem solchen Spielbaum spielt auf der ersten Ebene nur eine Amazone und verschießt erst in der zweiten Ebene den Pfeil des aufrufenden Spielers (analog dazu auf Ebene drei und vier beim Gegenspieler).

Da während der Simulation nicht für alle Amazonen, jede mögliche Pfeilposition gesucht werden muss, sondern nur für eine ausgewählte Amazone, wird erheblich Laufzeit gespart.

5.4. Performanz

Eine Betrachtung der Performanz des im Abschnitt 5.3 eingeführten optimierten UCT Algorithmus ($UCT_{optimiert}$) im Vergleich zur Basis ($UCT_{vanilla}$) (siehe Abschnitt 5.2), erfolgt über Simulationsrate, Speicherverbrauch und Optimalität. Die Abhängigkeit von Laufzeit und eingestellten Simulationen, macht die Simulationsrate zu einem aussagekräftigeren Leistungskriterium als die Laufzeit, da eine erhöhte Simulationsrate die Genauigkeit verbessert. Zum Zwecke eines fairen Vergleiches spielten beide Algorithmen mit begrenzter Zeit.

Der Graph 5.2 veranschaulicht die Erhöhung möglicher Simulationen, welche mit fortschreiten des Zeitrahmens einhergeht. Überdies lassen sich Raten von $591 \frac{\text{Simulationen}}{s}$ für den UCT Algorithmus und $690 \frac{\text{Simulationen}}{s}$ für die optimierte Version errechnen. Der optimierte UCT Algorithmus kann demnach annäherungsweise 17% mehr Simulationen in derselben Zeit durchführen.

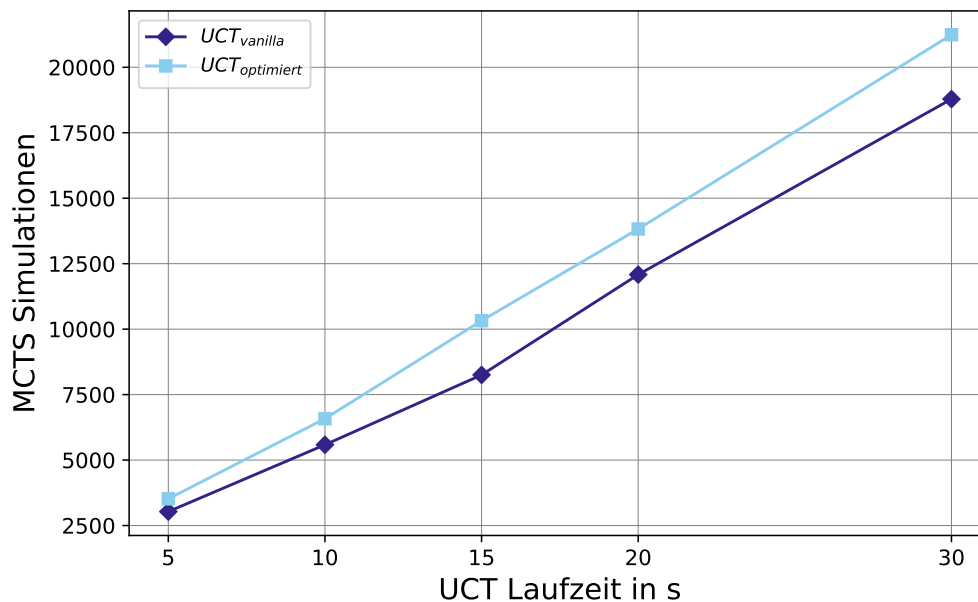
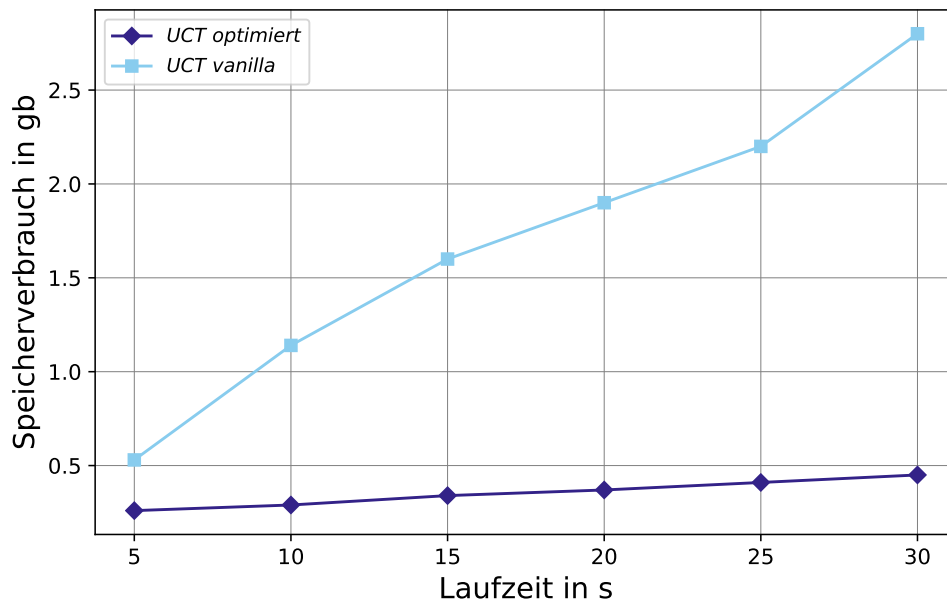


Bild 5.2.: Anzahl Simulationen der MCTS Algorithmen zu diversen Zeitfenstern.

Eine wesentliche Verbesserung des Speicherverbrauchs im optimierten UCT Algorithmus, birgt die auf Abschnitt 5.3.2 zurückzuführende Restrukturierung des Spielbaums. So belegt der normale UCT Algorithmus innerhalb von 30s circa 2,4Gb, während die optimierte Version nicht mehr als 63Mb okkupiert (siehe Abbildung 5.3). Eine Differenz dieser Dimension könnte kritisch für parallele Simulationen, sowie für die Ausführung auf leistungsschwächeren Geräten sein.

Bild 5.3.: Speicherverbrauch der *UCT* Methoden bei wachsender Laufzeit.

Ein Vergleich der spieltheoretischen Performanz beider *UCT* Algorithmen erfolgt über die in Tabelle 5.1 festgehaltenen Spieldaten. Es spielten beide Algorithmen mit diversen Einstellungen des Zeitkontos in jeweils 20 Spielen gegeneinander. Unvermutet spielt vanilla *UCT* durchschnittlich 75% genauer als die optimierte Version, dies könnte auf unterschiedliche Faktoren zurückzuführen sein:

Primär expandiert $UCT_{optimiert}$ pro Zyklus nur einen halben Zug. Auch wenn dies einen Laufzeitgewinn mit sich bringt [KS06], wird dieser Gewinn und die Optimalität durch die aufwendige Berechnung der weiteren Optimierungen geschmälert. Zudem kontert $UCT_{optimiert}$ Spieler, welche die spezifischen Mechaniken der Amazonen ausnutzen. Dies ist bei $UCT_{vanilla}$ nicht der Fall. Weiterhin können Implementierung [Yan21] und die Integration der experimentellen *UCB1 – tuned* Methode (siehe Abschnitt 5.2) ausschlaggebende Faktoren sein.

| MCTS Algorithmus | | Gewinnrate Weiß mit Zeitkonto t in s | | | |
|-------------------|-------------------|--|-----|-----|-----|
| Weiß | Schwarz | 1s | 5s | 10s | 20s |
| $UCT_{VANILLA}$ | $UCT_{OPTIMIERT}$ | 75% | 80% | 70% | 85% |
| $UCT_{OPTIMIERT}$ | $UCT_{VANILLA}$ | 20% | 25% | 35% | 30% |

Tabelle 5.1.: Gewinnraten der MCTS Algorithmen auf variablen Einstellungen.

6. Alpha Beta Suche vs. Monte Carlo Tree Search

Welche Algorithmenfamilie ist geeignet zur Lösung des Spiels der Amazonen? Das folgende Kapitel befasst sich mit der Auswertung im Rahmen dieser Fragestellung stattgefundener Experimente. Es spielten *Alpha Beta* und *UCT* in unterschiedlichen Konstellationen gegeneinander. Insbesondere die Verfeinerung der Spielqualität der *UCT* Algorithmen bei wachsender Spielzeit, sowie die individuelle Leistungsfähigkeit beider Gruppen unter beidseitiger Ressourcenbeschränkung wird verglichen.

6.1. Speicher

Der Implementierung der *Alpha Beta Suche* aus Abschnitt 4.4 folgend, wird während der Laufzeit kein zusätzlicher Speicher für das Generieren eines Spielbaums benötigt. Der *Monte Carlo Spielbaum* belegt indes mit Fortschritt der Simulation zunehmend Speicher, wobei *UCT_{vanilla}* einen wesentlich höheren Verbrauch ausweist als *UCT_{optimiert}* (siehe Abschnitt 5.4). *Alpha Beta Pruning* suchte innerhalb eines Fensters von 1 – 180 Sekunden pro Zug.

6.2. Experiment

Jede Spielspezifikation wurde jeweils 50 Runden für beide Farben simuliert. *Alpha Beta Pruning* (siehe Abschnitt 3) sucht bis Tiefe 2 bzw. 6 (bei weniger als 50 möglichen Zügen). Sowohl die *Mobilitäts Evaluation* H_M (siehe Abschnitt 4.5.1) als auch die *territoriale und lokale Evaluation* H_{TP} (siehe Abschnitt 4.5.2) wurden getestet. Die *UCT* Algorithmen *UCT_{vanilla}* und *UCT_{optimiert}* (siehe Abschnitt 5.2 und 5.3) spielten grundsätzlich mit einem Ressourcenkonto anstelle einer festgelegten Simulationszahl.

Der Graph in Abbildung 6.1 repräsentiert die Entwicklung der Gewinnrate (in %), des *UCT* Algorithmen gegen das *Alpha Beta Pruning*, unter Einfluss der wachsenden Simulationszeit (in Sekunden). Eine mit der Zeit wachsende Verbesserung der Leistung des *UCT* Algorithmus, lässt sich insbesondere gegen H_{TP} erkennen. Da die Gewinnraten gegen H_M bereits näher an der oberen Schranke angesiedelt sind, ist an der Stelle die stetige Verbesserung schwerer zu erkennen. *UCT_{vanilla}* erzielt gegen H_M eine 99% Gewinnrate von vormals 46% und gegen H_{TP} eine Verbesserung von 5% auf 56%. *UCT_{optimiert}* erzielt hingegen einen Aufstieg von 40% auf 91% für H_M und gegen H_{TP} von 5% auf 52%.

6. Alpha Beta Suche vs. Monte Carlo Tree Search

H_{TP} erweist sich somit auch gegen $MCTS$ als eindeutig stärkere Heuristik. Zwischen den UCT Algorithmen ist die Hierarchie nun nicht mehr eindeutig (vergleichsweise zu Abschnitt 5.4), wobei $UCT_{vanilla}$ gegen H_M tendenziell besser spielt. Bemerkenswert ist die Leistung von $UCT_{optimiert}$ bei geringer Simulationszeit gegen H_{TP} , es bestätigt sich die Stärke der KI gegen strategisch überlegener Gegenspieler (vgl. Abschnitt 5.4). Gleichwohl tendiert $UCT_{vanilla}$ zu besserer Leistung ab 20s Simulationszeit.

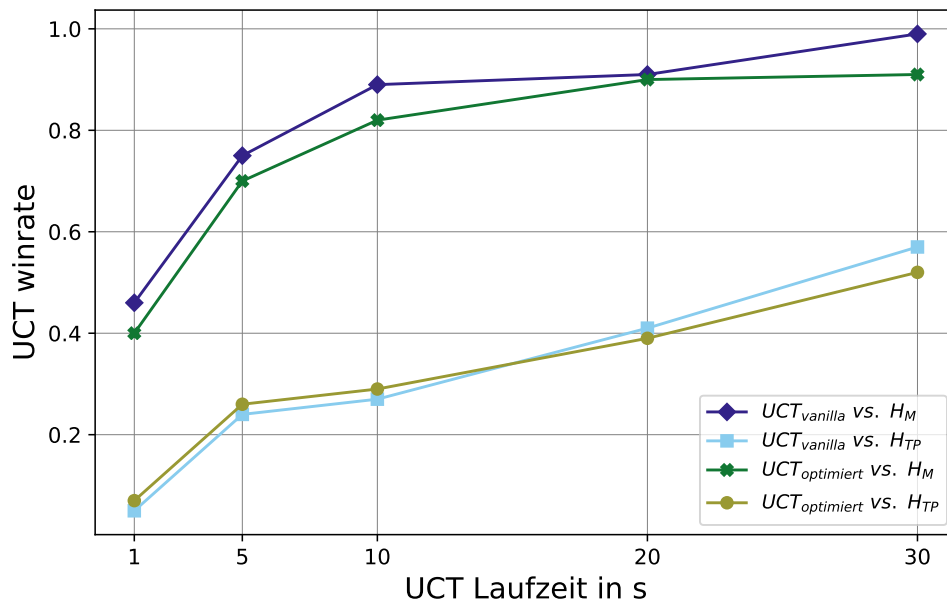


Bild 6.1.: $MCTS$ Algorithmen gegen $Alpha\ Beta\ Pruning$. UCT und seine Optimierung gegen Heuristiken für die Amazonen.

In einem weiteren Experiment spielten die UCT Algorithmen gegen H_{TP} und beidseitiger Zeitbeschränkung (30 Sekunden), um eine faire Simulationsumgebung zu schaffen. $MCTS$ dominiert in jedem Fall mit einer Gewinnrate von 81% ($UCT_{vanilla}$) und 71% ($UCT_{optimiert}$) vs. H_{TP} .

6.3. Auswertung

Die Beantwortung der Fragestellung dieser Arbeit, erfolgt durch die Auswertung der gesammelten Daten und Ergebnissen aus den Abschnitten 4.5.3, 5.4, 6.1 und 6.2.

Alpha Beta Pruning, mit der *territorialen und positionellen Evaluation* H_{TP} als Heuristik, ist stärkster Vertreter der *heuristischen Suche* im Spiel der Amazonen. Innerhalb der Klasse der stochastischen Methoden ist $UCT_{vanilla}$, durch *UCBI-tuned* erweitert (siehe Abschnitt 5.2), der stärkere *MCTS* Algorithmus. Jedoch spielt $UCT_{optimiert}$ eindeutig Speicher-effizienter. Es bleibt abzuwägen ob Speicher in Gigabyte Ordnung zur Verfügung steht oder nicht. Dennoch verwirft dieses Ergebnis die Hierarchie zwischen Algorithmus und Optimierung.

Die *stochastische Suche* ermöglicht demnach ein Spielen auf höherem Leistungsniveau als die *heuristische Suche*, für das Spiel der Amazonen. Dementsprechend scheint die weitere Forschung in diese Richtung als am vielversprechendsten. Dies beschränkt sich selbstverständlich auf die im Rahmen dieser Arbeit angewendeten Methoden und der spezifischen Implementieren jener, da die Optimalität primär von der Zeit abhängt und diese durch die individuelle Laufzeit negativen Einfluss auf das Ergebnis ausüben kann.

Die Konklusion wird indes durch die Dominanz der randomisierten Suchmethoden, all voran der *Monte Carlo Tree Search*, hinsichtlich der Computer Olympiade [Sat19] bestätigt.

Eine Weiterentwicklung der Evaluationsfunktion oder Implementation der *Alpha Beta Suche* könnte die Qualität der heuristischen Suche verbessern. Techniken wie *iterative deepening* und *Parallelisierung* [New88] sowie eine *Zug-Sortierung* [MC82], welche ultimativ Suchen in tieferen Ebenen des Spielbaums ermöglichen, sind beispielhafte Optimierungen der Implementation.

Auch besteht die Idee einer Hybridisierung stochastischer und heuristischer Methoden [KS06], dabei wird die Simulation eines Knotens innerhalb der *MCTS* auf eine festgelegte Tiefe beschränkt und bei Erreichen des Limits durch eine *heuristische* Funktion evaluiert.

Referenzen

- [Abr90] B. Abramson. „Expected-outcome: a general model of static evaluation“. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12.2 (1990), S. 182–193. DOI: [10.1109/34.44404](https://doi.org/10.1109/34.44404).
- [ACF02] Peter Auer, Nicolò Cesa-Bianchi und Paul Fischer. „Finite-time Analysis of the Multiarmed Bandit Problem“. In: *Machine Learning* 47.2 (Mai 2002), S. 235–256. ISSN: 1573-0565. DOI: [10.1023/A:1013689704352](https://doi.org/10.1023/A:1013689704352). URL: <https://doi.org/10.1023/A:1013689704352>.
- [BBS16] Markus Brill, Felix Brandt und Warut Suksompong. „An ordinal minimax theorem“. In: *Games and Economic Behavior* 95 (2016), S. 107–112. ISSN: 0899-8256. DOI: <https://doi.org/10.1016/j.geb.2015.12.010>. URL: <https://www.sciencedirect.com/science/article/pii/S0899825615001670>.
- [Bla19] Benjamin Blankertz. „Foliensatz: Algorithmen und Datenstrukturen SS19“. In: *Lehrstuhl für Neurotechnologie, Technische Universität Berlin* (2019).
- [Bro+12] Cameron B. Browne u. a. „A Survey of Monte Carlo Tree Search Methods“. In: *IEEE Transactions on Computational Intelligence and AI in Games* 4.1 (2012), S. 1–43. DOI: [10.1109/TCIAIG.2012.2186810](https://doi.org/10.1109/TCIAIG.2012.2186810).
- [Bur01] Michael Buro. „Simple Amazons Endgames and Their Connection to Hamilton Circuits in Cubic Subgrid Graphs“. In: *Computers and Games*. Hrsg. von Tony Marsland und Ian Frank. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, S. 250–261. ISBN: 978-3-540-45579-0.
- [CC06] R. Coulom und K. Che. „CrazyStone wins 9x9 Go Tournament“. In: *ICGA Journal* 29.2 (2006), S. 96–97.
- [Cha+08] Guillaume Chaslot u. a. „Progressive Strategies for Monte-Carlo Tree Search“. In: *New Mathematics and Natural Computation* 04 (Nov. 2008), S. 343–357. DOI: [10.1142/S1793005708001094](https://doi.org/10.1142/S1793005708001094).
- [Cou07] Rémi Coulom. „Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search“. In: *Computers and Games*. Hrsg. von H. Jaap van den Herik, Paolo Ciancarini und H. H. L. M. (Jeroen) Donkers. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, S. 72–83. ISBN: 978-3-540-75538-8.
- [EH61] D.J. Edwards und T.P. Hart. *The Alpha-Beta Heuristic (AIM-030)*. 1961. URL: <https://dspace.mit.edu/handle/1721.1/6098> (besucht am 14. 09. 2021).
- [Fri20] Stefan Fricke. „Symbolische Künstliche Intelligenz“. In: *Fachgebiet AOT, Technische Universität Berlin* (2020).

- [Hen01] P.P.L.M. Hensgens. „A Knowledge-based Approach of the Game of Amazons“. In: Masters Thesis. University Maastricht (2001), S. 17–18.
- [Kat+15] Hikari Kato u. a. „Comparative Study of Monte-Carlo Tree Search and Alpha-Beta Pruning in Amazons“. In: *Information and Communication Technology*. Hrsg. von Ismail Khalil u. a. Cham: Springer International Publishing, 2015, S. 139–148. ISBN: 978-3-319-24315-3.
- [KIB07] Julien Kloetzer, Hiroyuki Iida und Bruno Bouzy. *The Monte-Carlo Approach in Amazons*. 2007.
- [KIB08] Julien Kloetzer, Hiroyuki Iida und Bruno Bouzy. „A comparative study of solvers in Amazons endgames“. In: *2008 IEEE Symposium On Computational Intelligence and Games*. 2008, S. 378–384. DOI: [10.1109/CIG.2008.5035665](https://doi.org/10.1109/CIG.2008.5035665).
- [Klo10] Julien Kloetzer. *Monte-Carlo Techniques: Applications to the Game of the Amazons*. 2010. URL: <http://hdl.handle.net/10119/8867>.
- [KM75] Donald E. Knuth und Ronald W. Moore. „An analysis of alpha-beta pruning“. In: *Artificial Intelligence* 6.4 (1975), S. 293–326. ISSN: 0004-3702. DOI: [https://doi.org/10.1016/0004-3702\(75\)90019-3](https://doi.org/10.1016/0004-3702(75)90019-3). URL: <https://www.sciencedirect.com/science/article/pii/0004370275900193>.
- [Kov84] G. Kovacs. „From the first chess-automaton to the mars pathfinder“, Budapest 2016“. In: *Acta Polytechnica Hungarica* vol. 13 no. 1 (1884), S. 69–70.
- [KS06] Levente Kocsis und Csaba Szepesvári. „Bandit Based Monte-Carlo Planning“. In: *Machine Learning: ECML 2006*. Hrsg. von Johannes Fürnkranz, Tobias Scheffer und Myra Spiliopoulou. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, S. 282–293. ISBN: 978-3-540-46056-5.
- [Lie04] J. Lieberum. „An Evaluation Function for the Game of Amazons“. In: *Advances in Computer Games: Many Games, Many Challenges*. Hrsg. von H. Jaap Van Den Herik, Hiroyuki Iida und Ernst A. Heinz. Boston, MA: Springer US, 2004, S. 299–308. ISBN: 978-0-387-35706-5. DOI: [10.1007/978-0-387-35706-5_19](https://doi.org/10.1007/978-0-387-35706-5_19). URL: https://doi.org/10.1007/978-0-387-35706-5_19.
- [Lie05] Jens Lieberum. „An Evaluation Function for the Game of Amazons“. In: *Theor. Comput. Sci.* 349 (Jan. 2005), S. 230–244. DOI: [10.1007/978-0-387-35706-5_19](https://doi.org/10.1007/978-0-387-35706-5_19).
- [Lor18] Lorentz. *The Game of Amazons - Invader*. 2018. URL: <https://www.csun.edu/~lorentz/amazon.htm> (besucht am 25. 08. 2021).
- [MC82] T. A. Marsland und M. Campbell. „Parallel Search of Strongly Ordered Game Trees“. In: *ACM Comput. Surv.* 14.4 (Dez. 1982), S. 533–551. ISSN: 0360-0300. DOI: [10.1145/356893.356895](https://doi.org/10.1145/356893.356895). URL: <https://doi.org/10.1145/356893.356895>.
- [Min+06] Marvin L. Minsky u. a. „A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence, August 31, 1955“. In: 27.4 (2006), S. 12. ISSN: 2371-9621. DOI: [10.1609/aimag.v27i4.1904](https://doi.org/10.1609/aimag.v27i4.1904). URL: <https://aaai.org/ojs/index.php/aimagazine/article/view/1904>.

- [MR95] Rajeev Motwani und Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995. DOI: [10.1017/CB09780511814075](https://doi.org/10.1017/CB09780511814075).
- [Neu28] J. v. Neumann. „Zur Theorie der Gesellschaftsspiele“. In: *Mathematische Annalen* 100.1 (Dez. 1928), S. 295–320. ISSN: 1432-1807. DOI: [10.1007/BF01448847](https://doi.org/10.1007/BF01448847). URL: <https://doi.org/10.1007/BF01448847>.
- [New88] M. Newborn. „Unsynchronized iteratively deepening parallel alpha-beta search“. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 10.5 (1988), S. 687–694. DOI: [10.1109/34.6777](https://doi.org/10.1109/34.6777).
- [OR94] Martin J. Osborne und Ariel Rubinstein. *A Course in Game Theory*. The MIT Press, 1994. ISBN: 0262150417.
- [Pac15] Hertwig R. und Pachur T. „Heuristics, history of“. In: *International Encyclopedia of the Social & Behavioral Sciences* 2. Auflage (2015), S. 829–835. URL: <http://hdl.handle.net/11858/00-001M-0000-0026-B04C-7>.
- [Rac89] Joseph Friedrich zu Racknitz. *Ueber den Schachspieler des Herrn von Kempelen und dessen Nachbildung*. 1789. URL: <https://www.digi-hub.de/viewer/image/BV041097321/65/> (besucht am 14. 09. 2021).
- [Rei89] Alexander Reinefeld. *Spielbaum-Suchverfahren*. Springer Berlin Heidelberg, 1989. DOI: [10.1007/978-3-642-74413-6](https://doi.org/10.1007/978-3-642-74413-6). URL: <http://dx.doi.org/10.1007/978-3-642-74413-6>.
- [RN99] Stuart Russell und Peter Norvig. *Artificial intelligence: A modern approach: United States edition*. 2. Aufl. Pearson, 1999. ISBN: 9780137903955.
- [RÖS84] W. H. RÖSCHER. „Ausführliches Lexikon der Griechischen Und Römischen Mythologie“, Leipzig 1884-1923“. In: 1 (1884), S. 267–279.
- [Sat19] Erik Satie. *Amazons Computer Olympiads*. 2019. URL: <https://www.chessprogramming.org/Amazons> (besucht am 01. 10. 2021).
- [SM15] Jiaying Song und Martin Müller. „An Enhanced Solver for the Game of Amazons“. In: *IEEE Transactions on Computational Intelligence and AI in Games* 7.1 (2015), S. 16–27. DOI: [10.1109/TCIAIG.2014.2309077](https://doi.org/10.1109/TCIAIG.2014.2309077).
- [SP96] Jonathan Schaeffer und Aske Plaatt. „New Advances in Alpha-Beta Searching“. In: *Proceedings of the 1996 ACM 24th Annual Conference on Computer Science*. CSC '96. Philadelphia, Pennsylvania, USA: Association for Computing Machinery, 1996, S. 124–130. ISBN: 0897918282. DOI: [10.1145/228329.228344](https://doi.org/10.1145/228329.228344). URL: <https://doi.org/10.1145/228329.228344>.
- [SS19] Shubendra Pal Singhal und M. Sridevi. „Comparative study of performance of parallel alpha Beta Pruning for different architectures“. In: *2019 IEEE 9th International Conference on Advanced Computing (IACC)*. 2019, S. 115–119. DOI: [10.1109/IACC48062.2019.8971591](https://doi.org/10.1109/IACC48062.2019.8971591).
- [Wei] Stefan Behnel und Weitere. *Cython: C-Extensions for Python*. URL: <https://cython.org> (besucht am 25. 08. 2021).

- [Yan21] Gün Yanik. *Game of the amazons - AlphaBeta and MCTS AI*. 2021. URL: <https://github.com/guen-ynk/Game-of-the-Amazons> (besucht am 25.08.2021).
- [Zam88] Walter Zamkaskas. *Amazons*. 1988. URL: <https://www.mathematik.hu-berlin.de/~ploog/BSB/LG-Amazons.pdf> (besucht am 01.10.2021).
- [Zam99] Walter Zamkaskas. *Amazons*. 1999. URL: <https://www.chessvariants.com/other.dir/amazons.html> (besucht am 25.08.2021).

A. Anhang

A.1. Spielbrett Konfigurationen

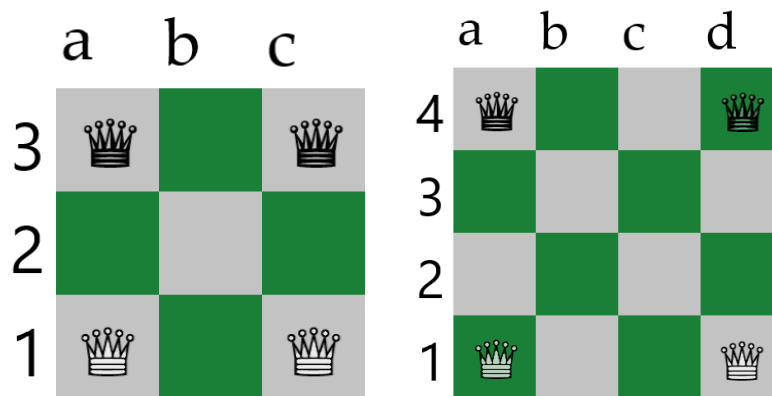


Bild A.1.: 3x3 & 4x4 Spiel der Amazonen Konfiguration.

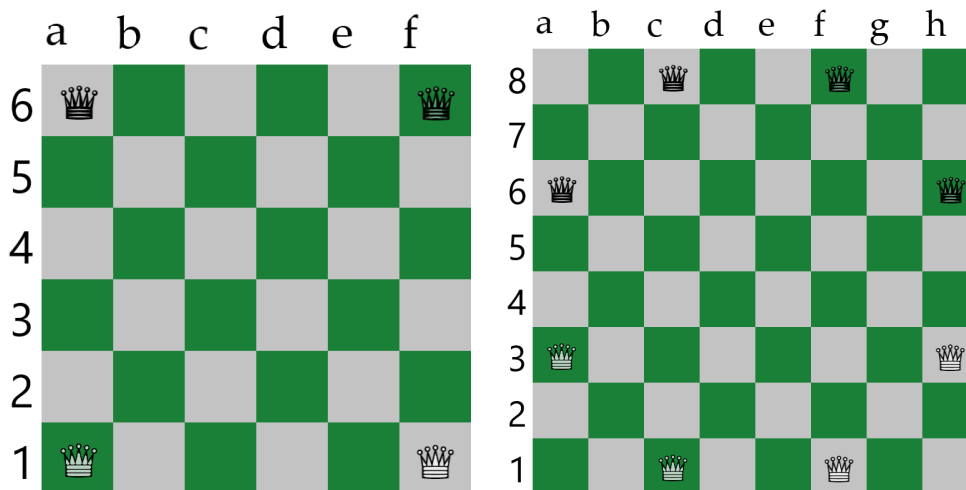


Bild A.2.: 6x6 & 8x8 Spiel der Amazonen Konfiguration.