Dynamische $\hat{f}$ Suche in General Video Game Playing

Dynamic $\hat{f}$ Search in General Video Game Playing
Bachelor-Thesis von Miriam Ulrike Moneke aus Darmstadt
Tag der Einreichung:

1. Gutachten: Prof. Dr. Johannes Fürnkranz
2. Gutachten: Christian Wirth

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Fachbereich Informatik
Knowledge Engineering Group

Dynamische $\hat{f}$ Suche in General Video Game Playing
Dynamic $\hat{f}$ Search in General Video Game Playing

Vorgelegte Bachelor-Thesis von Miriam Ulrike Moneke aus Darmstadt

1. Gutachten: Prof. Dr. Johannes Fürnkranz
2. Gutachten: Christian Wirth

Tag der Einreichung:

Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 20. Juni 2017

_____

(Miriam Ulrike Moneke)

Zusammenfassung

Für Suchalgorithmen wie A* kann es schwierig sein geeignete Heuristiken zu finden. In dieser Arbeit wird die dynamische $\hat{f}$ Suche vorgestellt, welche in der Lage ist, Fehler die durch die Heuristik entstehen zu beheben. Diese Variante wird auf Spiele des General Video Game AI Wettbewerbes angewendet. Es stellt sich als schwierig heraus, für verschiedene Spiele eine Heuristik zu finden, die zulässig und konsistent ist. Auf ausgewählten Spielen wird die dynamische $\hat{f}$ Suche mit unzulässiger und inkonsistenter Heuristik getestet. Hierfür muss ein initialer ϵ Wert für die Suche gefunden werden. Ein Vergleichsexperiment wird auf den selben Spielen ohne Fehlerbehebung gestartet und anschließend analysiert. Die Auswertung zeigt, dass der positive Effekt der dynamischen $\hat{f}$ Suche bei geeigneter Wahl des initialen ϵ Wertes, auch für unzulässige und inkonsistente Heuristiken zu trifft.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Diese Arbeit befasst sich mit Suchverfahren, welche für unterschiedliche Problemstellungen Lösungen finden können. Das Lösen von Problemen gehöhrt seit jeher zum Menschen. Viele Problemstellungen sind komplex und groß, sodass auf technische Hilfsmittel zurückgegriffen wird. Es gibt zahlreiche Varianten und Einsatzgebiete. Heuristische Suchverfahren können bei geeigneter Wahl der Heuristik optimale Lösungen erzielen. Jedoch ist es für einige Problemstellungen schwierig eine geeignete Heuristik zu finden. Die dynamisch $\hat{f}$ Suche [1] bietet eine Möglichkeit, Ungenauigkeiten von Heuristiken auszugleichen.
Ein Suchverfahren dient dazu in einem Suchraum ein Objekt zu finden, welches vorher definierten Eigenschaften genügt. Ein Suchraum kann ein Baum oder ein Graph sein, der aus Knoten besteht, die über Kanten verbunden sind. Ein gesuchtes Objekt könnte ein Knoten oder ein Blatt des Baumes oder Graphen sein. Eine Baumsuche ist ein Suchverfahren, das in einem Baum einen speziellen Knoten sucht. Die Suche kann nach verschiedenen Strategien erfolgen. Viele Suchverfahren verfolgen die Idee, den Baum systematisch zu durchsuchen. Klassische Algorithmen die diesen Ansatz verfolgen sind Breitensuche und Tiefensuche [10].

Abbildung 1: Ein Spiel aus dem Framework des GVG-AI Wettbewerbes mit dem Namen BAIT. Zusätzliche Beschreibungen der Elemente auf dem Spielfeld.

Bei diesen Verfahren kann es passieren, dass alle Knoten des Baumes betrachtet werden müssen, um einen speziellen Knoten zu finden. Sollte ein Baum eine endliche Anzahl an Knoten haben, so kann garantiert werden, dass der gesuchte Knoten gefunden wird. Jedoch kann dies im schlechtesten Fall dazu führen, dass alle Knoten betrachtet werden müssen. Die jeweiligen Suchverfahren benötigen jedoch viel Zeit und Speicherressourcen. Um Ressourcen zu sparen

und früher den speziellen Knoten zu finden, gibt es heuristische Suchverfahren.

Die A* Suche [6] und die Bestensuche sind sehr verbreitete Suchalgorithmen, die mit Schätzwerten arbeiten. Hierbei wird versucht nur Knoten des Baumes zu betrachten, die vielversprechend sind, um zu dem gesuchten Knoten zu gelangen. Die Herausforderung ist hierbei eine Bewertung zu finden, um die vielversprechenden Knoten zu ermitteln. Diese Bewertungen werden durch die Heuristik repräsentiert. Da die benötigten Werte abgeschätzt werden, können Fehler entstehen.
Wenn diese Heuristik gewisse Eigenschaften hat, kann man zeigen, dass diese Varianten effizientere bis hin zu idealere Lösungen finden, als die Suchen, die alle Knoten nach einer festgelegten Reihenfolge durchsuchen. Eine Lösung ist das Auffinden eines Knoten, der gesuchte Eigenschaften aufweist. Dadurch sind die Schätzwerte für die Suchverfahren von großer Bedeutung. In einigen Bäumen ist es schwierig, eine gute Abschätzung zu finden, um die Knoten zu bewerten. Des Weiteren erfüllen die gefundenen Schätzwerte nicht unbedingt die Bedingungen, so dass das entsprechende Suchverfahren nicht garantieren kann, eine optimale Lösung zu finden. In diesem Fall muss weiterer Aufwand betrieben werden, um dennoch eine optimale Lösung zu erhalten.
Für einige Bäume gibt es geeignete Methoden, um auf diesen hervorragende Ergebnisse zu erzielen. Somit können die einzelnen Suchverfahren auf den unterschiedlichen Bäumen optimiert werden. Das bedeutet, dass pro Baum eine Optimierung stattfinden müsste. Bei Problemstellungen, bei denen zeitnah viele verschiedene Bäume zu betrachten sind, kann man versuchen diese Bäume mit dem selben Verfahren zu betrachten. Die Schwierigkeit liegt darin, auf die Besonderheiten der Bäume einzugehen. Dazu ist es notwendig, dass während des Suchverfahrens die Besonderheiten mit Hilfe von Ansätzen aus der künstlichen Intelligenz ermittelt werden. Mit Hilfe dieser Erkenntnisse kann direkt während der Suche auf die Eigenschaften des Baumes eingegangen werden.

Ausgangspunkt dieser Arbeit ist das Framework des GVG-AI Wettbewerbes, an dem in den vergangenen Jahren einige Teams der TU Darmstadt teilgenommen haben. Jedes Team reicht einen Agenten ein, der versucht unbekannte Spiele zu lösen. Um einen guten Agenten zu entwickeln ist es hilfreich diese mit guten Heuristiken auszustatten. Jedoch besteht auch hier die Schwierigkeit eine allgemeingültige Heuristik zu finden, die auf unterschiedlichen Spielen großen Erfolg erzielt. Daher muss eine Methode gefunden werden, sodass der Algorithmus trotz schlechter Heuristik eine gute Lösung liefert. Mit Hilfe der Dynamische $\hat{f}$ Suche können Fehler, die durch die Heuristik entstehen ausgeglichen werden. Diese Möglichkeit wird genutzt und auf die Problemstellung des Wettbewerbes angewendet. Die Kernfrage, mit der sich diese Arbeit befasst, lautet:

Bewährt sich diese Methode des dynamischen $\hat{f}$ Algorithmus auf Heuristiken, die weder konsistent noch zulässig sind?

1.1 Aufbau der Arbeit

In Kapitel 2 wird auf die Bedeutung von Spielen im Bereich künstliche Intelligenz eingegangen. Es werden die zwei Themengebiete GGP und GVGP vorgestellt. Auch auf die dazugehörigen Wettbewerbe, für die Agenten erstellt werden müssen, wird eingegangen. Auf die Nutzung von Baumsuchalgorithmen wird in Kapitel 3 eingegangen. Hier werden Probleme und Fehlermöglichkeiten, die in heuristischen Suchverfahren auftreten können aufgezeigt. In Kapitel 4 wird die dynamische $\hat{f}$ Suche vorgestellt. Mit dieser Suche wird ein Experiment, wie in Kapitel 5 erläutert, gestartet, um mögliche Verbesserungen zu einer herkömmlichen Variante zu untersuchen. Die Untersuchungsergebnisse werden in Kapitel 6 beschrieben. Abschließend werden die Ergebnisse in Kapitel 7 ausgewertet. Hier wird die Kernfrage aufgenommen und beantwortet.

2 Spiele im Bereich GVGP

Die künstliche Intelligenz erlangt immer mehr Einsatzgebiete und Anwendungsmöglichkeiten. Einige Strategien werden bereits in der Medizin, in der Spielentwicklung, von der Autoindustrie oder für Such Algorithmen verwendet. Um diese Technologien zu bewerten können Spiele verwendet werden. Sie sind gut als Benchmark für künstliche Intelligenz.
Spiele sind nicht nur hervorragend geeignet, um künstliche Intelligenz zu bewerten und zu vergleichen, sie bieten den Entwicklern auch die Möglichkeit sich zu messen. Denn Spiele können einfache bis sehr komplexe Problemstellungen wiederspiegeln. Zudem lassen sich die Spielergebnisse häufig in Rankings aufgliedern, aus denen man Rückschlüsse auf die Güte des Spielers führen kann.
In Kapitel 2.1 werden Agenten vorgestellt, die zur Teilnahme an den Wettbewerben 2.2 wie GGP 2.3 oder GVGAI 2.5, 2.6 genutzt werden können. Dazu wird General Video Game Playing in Kapitel 2.4 näher beschrieben.

2.1 Agent

Ein Agent [10] kann seine Umgebung wahrnehmen und Handlungen durchführen. Er versucht in der Umgebung in der er sich befindet, selbständig zu handeln, die Umgebung zu erkunden, auf Veränderungen einzugehen oder Ziele zu erstellen und zu verfolgen. Um seine Umgebung wahrnehmen zu können benötigt ein Agent Sensoren. Für die Durchführung von Handlungen werden Aktionsmöglichkeinten verlangt.
Agenten im Bereich Robotik nutzen häufig Kameras, Entfernungsmessgeräten oder Berührungsseinsoren als Sensoren. Das gezielte Ansteuern von Motoren bietet einem Roboteragent Aktionsmöglichkeiten. Der Input der Sensoren wird von dem Roboteragenten genutzt um Informationen zu sammeln. Mittels dieser stellt er Berechnungen an, um die Aktionen zu bestimmen. Die Berechnungen liefern einen Output, um die Motoren entsprechend zu betreiben. Mit diesem Konzept ist es Roboteragenten möglich verschieden komplexe Aufgaben zu bewerkstelligen.
Das Ziel eines Spieleagenten ist es, ein Spiel so gut wie möglich zu lösen. Vom Spiel erhält der Agent Informationen über die Umgebung. Die Spielfigur kann der Agent über Befehle steuern, und somit Aktionen ausführen. Damit der Agent das Spiel gewinnen kann, muss er der Spielfigur passende Aufträge erteilen. Dazu ist es notwendig, mit Hilfe des Inputs und den gesammelten Informationen Entscheidungen zu treffen. Dieser Schritt der Entscheidungsfindung kann nach einfachen Bedingungen erfolgen, oder der Agent verwendet dazu Lernprozesse der künstlichen Intelligenz.
In Anwendungen, in denen dem Agenten die Regeln des Spieles nicht bekannt sind, ist es hilfreich Lernprozesse zu verwenden, damit er die Regeln und Spielabläufe erlernt. Ohne dieses Wissen ist es für einen Agenten schwierig Spiele zu lösen, da er nicht beurteilen kann ob Aktionen die er ausführen möchte zum Ziel führen. Den Input seiner Sensoren kann der Agent dazu nutzen, um auf folgende Fragen eine Antwort zu erhalten:

- Wo befindet sich die Spielfigur des Spielers?
- Wo sind Gegner und Spielobjekete?
- Was sind die aktuellen Möglichkeiten (Aktionen)?
- Welche Möglichkeit wird die Spielfigur zu einem bestimmten Ziel bringen?

Wenn der Agent die Position der Spielfigur und die Position eines anderen Objektes kennt, kann er die Spielfigur zu diesem Objekt steuern. Die Information, welche Eigenschaften das Objekt

hat ist nötig, um eine Entscheidung zu treffen. Denn wenn das Einsammeln des Objektes Punkte bringt, kann der Agent das Objekt einsammeln. Das Objekt kann auch eine Falle sein, durch die das Spiel verloren geht. In diesem Fall sollte der Agent die Spielfigur um das Objekt herum bewegen. Um einen erfolgreichen Agenten zu entwickeln, muss der Prozess der Informationsgewinnung und Entscheidungsfindung optimiert werden.

2.2 Wettbewerbe

In Wettbewerben kann man sich messen und vergleichen. Im Bereich künstliche Intelligenz gibt es verschiedene Wettbewerbe. Für viele Wettbewerbe ist es nötig einen Agenten zu konzipiert, der Algorithmen und Techniken der künstlichen Intelligenz verwendet. Ein Agent nimmt seine Umgebung über Sensoren wahr und handelt in dieser Umgebung durch Akteure. Es sind diverse Varianten möglich, über die die Agenten bewertet werden können. Entweder treten die Agenten gegen Menschen, andere Agenten oder das Spiel an. Darüber versucht man zu bestimmen, wie erfolgreich die Agenten im Vergleich zu den anderen Agenten oder zur Menschheit sind.

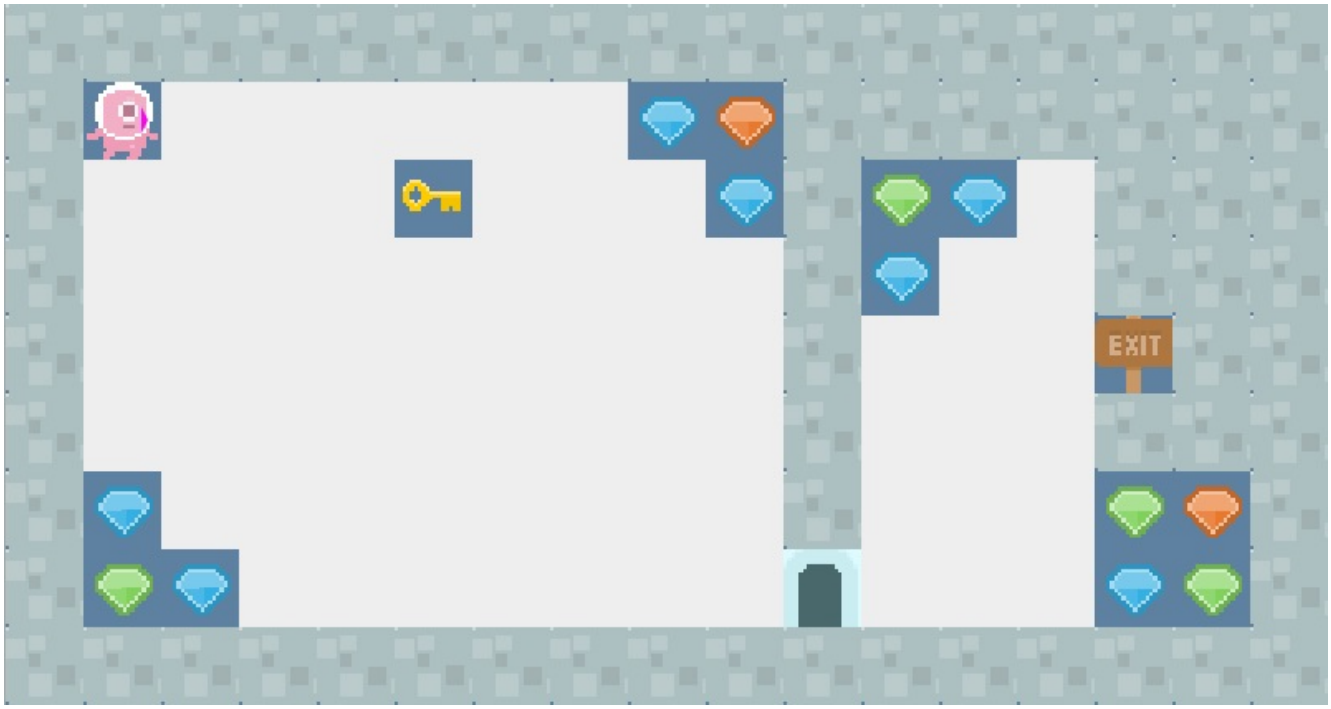

Abbildung 2: Ein Spiel aus dem Framework des GVG-AI Wettbewerbes mit dem Namen *Brainman*.

Die zum Teil grundverschiedenen Anwendungsgebiete der künstlichen Intelligenz fordern spezifische Lösungsansätze. Die Technologien der künstlichen Intelligenz werden entsprechend der jeweiligen Problemstellung angepasst und kombiniert, um die bestmögliche Lösung zu erzeugen.
Das Übertragen einer Lösung auf eine andere Problemstellung gestaltet sich als schwer, da die Lösungsansätze auf das eine Problem spezialisiert sind. Es gibt Lösungsansätze, die in vielen Anwendungen sehr erfolgreich sind, sich aber verbessern lassen, wenn man sie auf ein Problem spezialisiert. Dennoch ist es interessant, zu erfahren, wie verschiedenen Lösungsansätze mit unterschiedlichen Problemstellungen zurecht kommen.
Mit Hilfe von Wettbewerben werden auch künstliche Intelligenz Agenten auf verschiedene Pro-

blemstellungen angewendet und verglichen. Für die Wettbewerbe eignen sich Spiele als Problemstellungen, da es möglich ist ein Framework zu erstellen, dass verschiedenen Problemestellungen in Art von Spielen zur Verfügung stellt.

2.3 General Game Playing

General Game Playing (GGP) ist ein Forschungsgebiet der künstlichen Intelligenz. General Game Player [4] sind Computersysteme, die in der Lage sind ein Strategiespiel während der Laufzeit zu spielen. Das General Game Playing System arbeitet auf einer formalen Beschreibung des Spieles. Während der Laufzeit findet keine Interaktion mit einem Menschen statt. Der Erfolg eines Gerneral Game Players ist abhängig von seiner Intelligenz, und somit nur indirekt abhängig von der Intelligenz des Programmierers.
General Game Player werden nicht für ein spezielles Spiel entwickelt, sondern dafür, verschiedene Spiele zu spielen. Hierfür können die General Game Player verschiedene Komponenten der künstlichen Intelligenz, wie Lernen, rationale Entscheidungsfindung oder Wissensverarbeitung kombiniert nutzen. Dadurch sollte ein solches System in der Lage sein simple Spiele (wie Tic-Tac-Toe) und komplexe Spiele (wie Schach) zu spielen. Auch mit den verschiedenen Eigenschaften von Spielen sollte ein General Game Player umgehen können.
Um Forschung in diesem Bereich attraktiver zu gestalten, wird ein gleichnamiger Wettbewerb von der AAAI Konferenz veranstaltet. Der General Game Playing (GGP) Wettbewerb [3] wird von der Logik Gruppe der Stanfort Universität organisiert. Die Teilnahme erfolgt durch Abgabe eines Agenten. Der Agent übernimmt die Rolle des Game Players. Dieser hat die Aufgabe, ein unbekanntes Spiel zu lösen, wobei der Agent Zugang auf die Regeln und die Spielumgebung hat.

2.4 General Video Game Playing

General Video Game Playing (GVGP) [7] ist ein Tool um künstliche Intelligenz Algorithmen zu testen. GVGP hat zwei Einschränkungen. Zum einen ist die Zeit, die einem General Video Game Player zur Verfügung steht um eine Aktion durchzuführen, auf Millisekunden beschränkt. Zum anderen sind die Spielregeln der Spiele unbekannt. Diese Bedingungen können in einem Framework verankert werden.
Ein General Video Game Player kann durch einen Agenten repräsentiert werden, der verschiedene Spiele spielt. Hierfür ist das Lernen der Spielregeln wichtig.
In einigen Spielen gibt es Positionen, die beim Betreten der Spielfigur zum Sieg führen oder auf denen die Spielfigur stirbt und somit das Spiel verloren wird. Solche Beobachtungen kann ein Agent leicht interpretieren, da die Effekte direkt eintreffen. Es gibt andere Szenarien, in denen die Effekte nicht direkt eintreten und dem Agenten dadurch die Interpretation schwer fällt. Dies kann der Fall sein, wenn ein Sieg aus mehreren Aufgaben besteht, wie das ablegen verschiedener Objekte auf bestimmten Spielfeldern. Wenn die Spielfigur ein Spielobjekt an der richtigen Position abgelegt hat, und das Ablegen zu keiner besonderen Beobachtung führt, kann der Agent nur schwer interpretieren, ob diese Aktion sinnvoll ist. Mit dieser Problemstellung befasst sich der Wettbewerb GVG-AI (General Video Game Artificial Intelligence).

2.5 GVG-AI Wettbewerb

Auf Grundlage des GVGP gründete sich der Wettbewerb General Video Game Artificial Intelligenz Competition (GVG-AI Wettbewerb). AI ist die englische Übersetztung von künstliche In-

telligenz. In dem Wettbewerb treten verschiedene Agenten gegeneinander an. Dadurch kann festgestellt werden, welche Strategien der Graphensuche und künstlichen Intelligenz am besten abschneiden. Um die Auswertung so einfach und genau wie möglich zu machen wird auf zweidimensionalen Arkade-Spielen getestet. Der GVG-AI Wettbewerb stellt einem das Problem, einen Agenten zu kreieren, der es schafft, viele verschiedene Spiele zu lösen. Es handelt sich um Spiele, deren Regeln dem Agenten nicht bekannt sind. Um die Spiele zu lösen befinden sich die Agenten in einer völlig beobachtbaren Umgebung. Die Schwierigkeit liegt darin, mithilfe geeigneter Suchverfahren die Spiele zu lösen. Das lernen von Ereignissen während des Suchlaufes ermöglicht es dem Agenten auf verschiedene Arten von Spielen zu reagieren. Es ist nicht möglich allgemeingültige Regeln dem Agenten vor zu setzten, da die verschiedenen Spiele gegensätzliche Regeln beinhalten können. Daher ist das Lernen der Spielregeln von großer Bedeutung. Die zweidimensionalen Spiele sind in der Video Game Description Language implementiert.

2.6 Ablauf des GVG-AI Wettbewerbes

An dem Wettbewerb können Einzelpersonen oder Teams aus der ganzen Welt teilnehmen. Dazu legt man einen Account auf der GVG-AI Webseite an, kreiert einen Agenten und läd diesen hoch. Ein passendes Framework wird von den Veranstaltern zur Verfügung gestellt. Das Framework liefert zweidimensionale Spiele zum testen, die in der Video Game Description Language [8] verfasst sind.
Die durch die Teilnehmer eingereichten Agenten spielen zehn Spiele, die den Teilnehmern und somit auch den Agenten unbekannt sind. Jedes Spiel hat fünf einzelne Level, somit werden insgesamt 50 verschiedene Level gespielt. Um die Auswertung fair zu halten, spielen die Agenten jedes der 50 Level (zehn Spiele mit jeweils fünf Level) zehn mal, denn einige Spiele enthalten stochastische Effekte, die je nach Situation das Spiel vereinfachen oder auch erschweren können.
Pro Spiel wird erstmal ein eigenes Ranking über alle Level des Spieles erstellt. Dieses Spieleranking wird nach Anzahl an gewonnenen Levels des Spiels, dann nach erreichtem Punktestand und dann nach benötigten Schritten ermittelt. Die besten zehn Teams des Spiels erhalten einen Punktestand entsprechend ihrer Platzierung { 25, 18,15,12,10,8,6,4,2,1}. Somit erhält das Team mit der besten Platzierung 25 Punkte. Die Teams die hintere Plätze erzielt haben erhalten keine Punkte.
Diese Werte werden für jedes Team über alle Spiele aufsummiert. Daraus ergibt sich das endgültige Ranking. Bei Punktegleichstand herrscht die gleiche Reihenfolge an Präferemzen wie für die einzelnen Spiele. Um den gesamte Wettbewerb zu gewinnen, muss man über alle Runden hinweg die meisten Punkte sammeln.

3 Suchverfahren

In diesem Kapitel werden verschiedene Suchverfahren vorgestellt. Diese basieren auf der grundlegenden Baumsuche 3.1. Die Bestensuche 3.6 verwendet eine Heuristik 3.11 und kann zu einer A* Suche 3.7 erweitert werden. In einigen Problemstellungen ist der Speicherplatz beschränkt oder die Suche soll zur Laufzeit passieren. Mit solchen Einschränkungen können die Varianten IDA* 3.8, LRTA* 3.9 und LSS-LRTA* 3.10 umgehen. Das verwenden von Heuristiken kann Fehler und Probleme verursachen 3.12.

3.1 Baumsuche

Baumsuchalgorithmen [10] arbeiten auf Knoten und Kanten. Alle Knoten eines gerichteten Baumes repräsentieren die Zustände. Die dazugehörigen Aktionen werden über die Kanten, die von einem Knoten weg führen dargestellt. Ein Baum hat einen Wurzelknoten, bei dem die Suche beginnt, den Startzustand. Alle Knoten, die keine herausführende Kante haben heißen Blätter. Ein Blatt ist ein Endzustand. Ausgehend von dem Wurzelknoten führen Kanten über Knoten zu Blättern. Ein Pfad ist eine solche Abfolge von Kanten und Knoten, die sowohl den Wurzelknoten als auch ein Blatt enthält.
Der Suchalgorithmus ist auf der Suche nach Pfaden zu bestimmten Blättern. Diese Blätter unterliegen gewissen Ansprüchen, wie das Erreichen eines Fahrziels oder die Lösung eines Spieles. Um ein solches Blatt zu finden werden verschiedene Pfade betrachtet. Ein Endzustand, der den Ansprüchen genügt, heißt Zielzustand.
Um Pfade zu erzeugen, ist es notwendig zu wissen, welcher Knoten n' von einem Knoten n erreicht werden kann. Dies ist der Fall, wenn eine Kante a von n nach n' führt.
Das Erzeugen und Betrachten von Pfaden kann auf verschiedene Arten erfolgen. Für einige Problemstellungen ist es nicht notwendig einen optimalen Pfad zu finden, oder den Suchaufwand einzuschränken. Jedoch ist das optimale Auffinden eines Pfades für manche Anwendungen, wie Echtzeitsysteme, essentiell. Daher gibt es verschiedene Ansätze, um einen Baum zu durchsuchen.

3.2 Zielzustände in dieser Arbeit

Manche Problemstellungen haben Eigenschaften, die den Zuständen zugeordnet werden können. Diese Arbeit behandelt eine Problemstellung, bei der gefundene Blätter Endzustände sind, denen die Begriffe SIEG und NIEDERLAGE zugeordnet werden können. Zielzustände sind die Blätter, deren besondere Eigenschaft SIEG als Ausgabe ist. Durch die Problemstellung der Spiele ist es gegeben, dass einige Zustände Punkte generieren. Zum Beispiel erhält man für das Einsammeln von DIAMANTEN einen Punktewert.

3.3 Kosten

Der durch Aktionen ausgelöste Schritt zwischen Zuständen kann Kosten [10] verursachen. Die Kosten für einen Schritt können mit Hilfe einer Kostenfunktion $c(n, a, n')$ dargestellt werden. Die Kosten können der Kante a des Baumes angeheftet werden, die die Knoten n und n' verbindet. Um die Kosten für einen (Teil-)Pfad zu ermitteln, werden die Kosten der einzelnen Kanten aufsummiert.
Die Summe der Kosten vom Startknoten s bis zu einem Knoten n wird mit $g(n)$ bezeichnet.

Wenn der Algorithmus einen Zielzustand gefunden hat war die Suche erfolgreich. Die Suche ist optimal verlaufen, wenn die Kosten über den Pfad vom Startknoten bis zum Zielknoten minimal sind. Das bedeutet, dass $g(z)$ für den gefundenen Pfad kleiner ist, als alle $g(z)$ für möglichen anderen Pfade.

3.4 Breiten- und Tiefensuche

Zwei sehr verbreitete Ansätze der Baumsuche sind Breitensuche [10] und Tiefensuche [10]. Die Strategie von Breitensuche ist, nach der Reihenfolge in der die Knoten gesehen werden, die Knoten zu betrachten und dabei alle Nachfolgeknoten zu erzeugen. Sowohl Breitensuche als auch Tiefensuche läuft in Iterationen ab. Pro Iteration wird der aktuelle Knoten betrachtet, das ist in der ersten Iteration die Wurzel. Um die anderen Knoten des Baumes zu verwalten, werden weitere Datenstrukturen benötigt.

Um Breitensuche zu realisieren bietet es sich an, zwei Listen, eine Open-Liste und eine Closed-Liste, zu verwenden. Zu Beginn der Suche enthält die Open-Liste den Startknoten und die Closed-Liste ist leer.
Während der Iteration wird ein Knoten der Open-Liste entnommen. Dieser Knoten ist für die Dauer der Iteration der aktuelle Knoten. Die Open-Liste kann durch eine First in First Out (FIFO) Liste repräsentiert werden. Alle Nachfolgerknoten des aktuellen Knotens werden der Open-Liste hinzugefügt. Diese Knoten können als gesehen bezeichnet werden. Der aktuelle Knoten wird der Closed-Liste hinzugefügt und gilt als beendet.
Dieser Ablauf wiederholt sich, bis die Suche einen Pfad gefunden hat, der ein gesuchtes Blatt enthält oder die Open-Liste keine Konten mehr enthält.
Das Speichern der gesehenen und beendeten Knoten dient dazu, dass jeder Knoten nur ein mal der aktuelle Knoten ist. Für endliche Bäume kann die Suche dadurch terminieren.

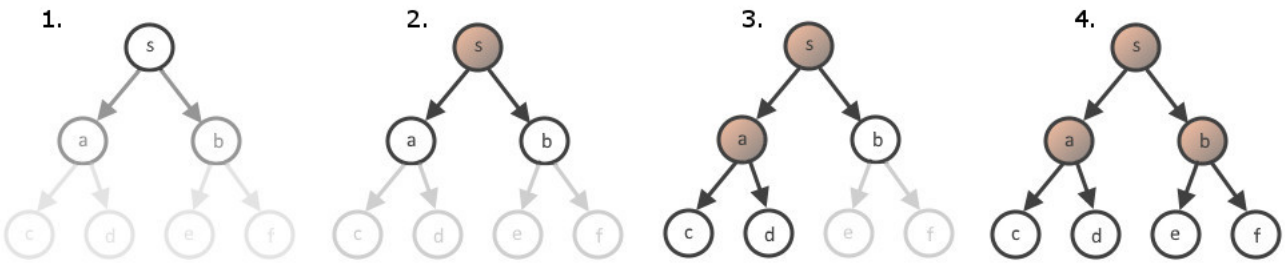

Abbildung 3: Suchreihenfolge einer Breitensuche auf einem einfachen Baum mit drei Iterationen (2., 3., 4.), und dem initialen Baum (1.).

In Abbildung 3 sind die ersten drei Iterationen der Breitensuche dargestellt: Der erste Baum zeigt den initialen Zustand des Algorithmus. Der Knoten *s* befindet sich in der Open-Liste (Knoten mit schwarzer Umrandung, ohne Füllfarbe) und kein Knoten ist in der Closed-Liste (farbiger Knoten).
In der ersten Iteration wird der Knoten *s* aus der Open-Liste entfernt. Er ist der aktuelle Knoten und wird am Ende der Iteration in die Closed-Liste aufgenommen, nachdem seine Nachfolgerknoten *a* und *b* der Open-Liste hinzugefügt wurden.

In der nächsten Iteration wird der Open-Liste zuerst a entnommen, da dieser Knoten zuerst aufgenommen wurde. Somit ist Knoten a der aktuelle Knoten. Die Nachfolgerknoten c und d von a werden an die Open-Liste angehängt. Der Knoten a ist beendet indem er in die Closed-Liste aufgenommen wird.
Der vierte Baum zeigt die Veränderungen der dritten Iteration. Der Knoten b wird aus der Open-Liste in die Closed-Liste verschoben. Seine Nachfolgerknoten e und f werden in der Open-Liste aufgenommen.

Tiefensuche betrachtet zuerst die Knoten, die als letztes gesehen wurden. Ein Knoten x wird vor einem Knoten y beendet, wenn der Knoten y vor dem Knoten x gesehen wurde. Durch den Austausch der FIFO-Liste durch einen Stack kann aus der Breitensuche eine Tiefensuche erzeugt werden. Die erste Iteration (2.) der in Abbildung 3 dargestellten Breitensuche entspricht auch der Tiefensuche. Bereits in der zweiten Iteration die durch den dritten Baum dargestellt wird, verändert sich die Suchreihenfolge. Der Konten b wird für diese Iteration als aktueller Knoten gewählt. Dadurch gelangen nicht die Knoten c und d in die Open-Liste, sondern die Knoten e und f. Der Knoten b wird danach der Closed-Liste hinzugefügt. Da der Knoten e vor dem Knoten f in die Open-Liste, die durch einen Stack realisiert ist, aufgenommen wurde, wird in der dritten Iteration der Knoten f als aktueller Knoten gewählt.
Der Knoten f wurde erst in der zweiten Iteration gesehen, aber dennoch werden alle Knoten, die an diesem Knoten hängen, betrachtet, bevor ein Knoten betrachtet wird, der an dem Knoten a hängt. Da der Knoten a vor dem Knoten f gesehen wurde, wird dieser erst nach dem Knoten f beendet.
Beide Strategien dienen dazu eine bestimmtes Blatt zu finden.
Wenn der Baum einen endliche Anzahl an Knoten hat, wird das gesuchte Blatt garantiert gefunden, da alle existierenden Knoten betrachten werden, bis das entsprechende Blatt gefunden wurde. Diese Suchverfahren können lange dauern und viel Speicherplatz erfordern.
Für Problemstellungen, bei denen eine kurze Lösung, also ein Pfad mit wenig Knoten, gesucht wird, eignet sich Breitensuche. Breitensuche arbeitet sich Schichtweise durch die Knoten. Daher haben die Teilpfade, die zu den Knoten in der Open-Liste führen alle eine Länge von k oder $k+1$. Da Bäume bereits bei mindestens zwei Nachfolgerknoten pro Knoten breit werden, also sich die Anzahl an Knoten pro Schicht jeweils mindestens verdoppelt, wird die Open-Liste schnell groß. Die Verwaltung dieser Knoten kann viel Speicherplatz erfordern.
Die Anzahl an Knoten die bei Tiefensuche gespeichert werden, ist kleiner, da bis zu dem aktuellen Knoten nur die Knoten gespeichert werden, die an dem Teilpfad über eine Kante verbunden sind.

3.5 Heuristische Suche

Von Suchverfahren wie Breiten- und Tiefensuche werden Bereiche eines Baumes durchsucht, von denen man annehmen kann, dass sie den gesuchten Knoten nicht enthalten. Das Betrachten der unwahrscheinlichen Bereiche eines Baumes verbraucht die Ressource Zeit. Des Weiteren werden bei Breitensuche und Tiefensuche alle Knoten in einer geeigneten Datenstruktur gespeichert. Dadurch steigt der benötigte Speicherplatz. Das Ziel von heuristischen Suchen [10], ist die Reduzierung des Zeitaufwandes sowie des Speicherplatzes.
Heuristische Suchen verwenden Heuristiken, die in Kapitel 3.11 beschrieben sind, während der Suche.
Um in den Iterationen der Suche einen aktuellen Knoten auszuwählen, wird eine Bewertungs-

funktion $f(n)$ verwendet. In diese Bewertung fließt die Heuristik $h(n)$ mit ein. In den folgenden Kapiteln wird $f(n)$ näher spezifiziert. Es bietet sich an, eine Prioritätswarteschlange (PWS) zu verwenden. Alle bereits gesehenen, aber noch nicht beendeten Knoten sind in der PWS enthalten und anhand ihrer Bewertungsfunktion sortiert. Die Bewertungsfunktion steuert, welche Bereiche des Baumes früher betrachtet werden.

3.6 Bestensuche

Eine Bestensuche [10] ist eine heuristische Suche, die versucht, so schnell wie möglich eine gute Lösung zu finden. Die Bewertungsfunktion $f(n)$ entspricht der Heuristik $h(n)$:

$$f(n) = h(n) \tag{1}$$

Die Suche beginnt bei dem Wurzelknoten, der in der PWS, die in Kapitel 3.5 vorgestellt wird, enthalten ist. Gesucht ist ein Blatt, das einen Zielknoten repräsentiert. In jeder Iteration wird der erste Knoten aus der PWS entfernt. Dieser Knoten ist für die jeweilige Iteration der aktuelle Knoten. Die Nachfolgerknoten des aktuellen Knoten werden anhand ihrer Bewertungsfunktion in die PWS einsortiert.
Der Algorithmus versucht über den Schätzwert dem Ziel in jedem Schritt so nahe wie möglich zu kommen, deshalb erzielt er nicht immer eine komplette oder optimale Lösung, aber der Suchaufwand verringert sich in vielen Anwendungsfällen.

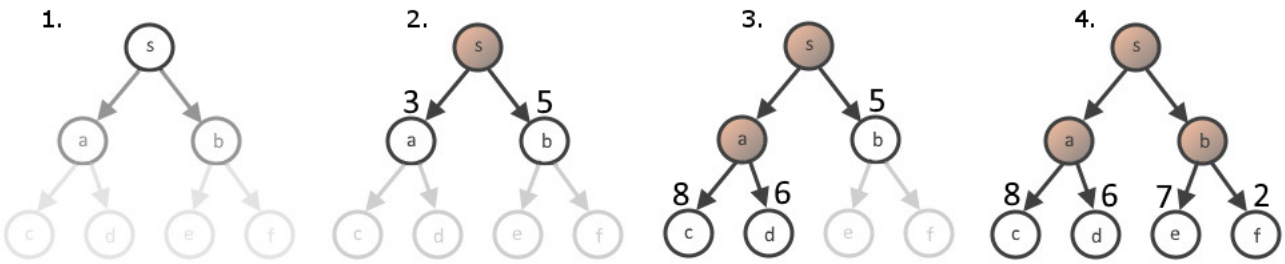

Abbildung 4: Suchreihenfolge einer Bestensuche auf einem einfachen Baum mit drei Iterationen (2., 3., 4.), und dem initialen Baum (1.). Angabe der Heuristik in Zahlen oberhalb der Knoten.

In Abbildung 4 1. ist der initiale Baum einer Bestensuche zu sehen. Die Suche startet bei dem Knoten s, der sich in einer PWS (schwarze Umrandung) befindet. In 2. ist die erste Iteration der Suche abgebildet. Der kleinste Knoten wird der PWS entnommen. Da nur der Knoten s enthalten ist, ist dieser der aktuelle Knoten. Die Nachfolgerknoten a und b werden in die PWS einsortiert. Der Knoten a wird vor dem Knoten b in die PWS einsortiert, da die Heuristik mit drei kleiner als die von Knoten b mit fünf ist.
In Iteration zwei, der Baum 3., wird der vordere Knoten der PWS entnommen. Dieser Knoten ist a, dessen Nachfolger entsprechend ihrer Heuristik in die PWS aufgenommen werden. Die PWS hat nun die folgende Form:

{b; d; c}

In 4. ist die Veränderung der dritten Iteration zu sehen. Der erste Knoten wird der PWS entnommen. Der aktuelle Knoten ist nun b. Dieser Knoten wird expandiert und die Knoten e und f werden entsprechend ihrer Heuristik in die PWS eingefügt:

{f; d; e; c}

In der nächsten Iteration, würde die Suche den Knoten f als aktuellen Knoten wählen. Die Suche wird abgebrochen, wenn ein Zielknoten gefunden wurde.

3.7 A* Suche

Die verbreitetste Form der heuristischen Baumsuchalgorithmen ist die A* Suche [10]. Wegfindungsprobleme, wie sie in Navigationssystemen auftreten, lassen sich mit Hilfe der A* Suche beheben. Dieser Standardlösungsansatz findet stets eine optimale Lösung. Die Bestensuche ist anfällig für Sackgassen, da sie die Schritte die bereits durchgeführt wurden nicht beachtet. Die Suche nach einer optimalen Lösung mit geringerem Suchaufwand als bei Breiten- oder Tiefensuche wird mittels der A* Suche gelöst. Die Bewertung $f(n)$ der Knoten wird aus der Addition von $g(n)$, den tatsächlichen Kosten um einen Knoten zu erreichen und der Heuristik $h(n)$ berechnet:

$$f(n) = g(n) + h(n) \tag{2}$$

Wenn $g(n)$ die Pfadkosten vom Startknoten zum Knoten n angibt und $h(n)$ die geschätzten Kosten des günstigsten Pfades von n zum Zielknoten entspricht, dann ist:

$$f(n) = \text{geschätzte Kosten der günstigsten Lösung, die } n \text{ enthält}$$

Die Suche nach einem optimalen Pfad von einem Knoten s zu einem anderen Knoten t kann mit dem Algoritmus A* [2] durchgeführt werden. Der Algorithmus wird von der Kostenfunktion geleitet. Hierbei sind g(n) die Kosten des aktuellen Pfades vom Startknoten s bis zu einem Knoten n. Die Heuristische Funktion $h(n)$ schätzt die Kosten von n bis zum Zielknoten t. Um die günstigste Lösung zu finden muss der kleinste $f(n)$ Wert für die Suche ausgewählt werden.

Ablauf des Algorithmus:

1. Starte bei Knoten s
2. Expandiere alle Kindknoten s' von s
3. Bestimme die f Werte aller Kindknoten
4. Füge die Kindknoten der Open-Liste hinzu, solange sie nicht in der Closed-Liste enthalten sind
5. Füge s der Closed-Liste hinzu
6. Wähle Knoten mit kleinstem f Wert aus der Open-Liste
7. Starte bei Schritt 1. mit $s = neuerKnoten$

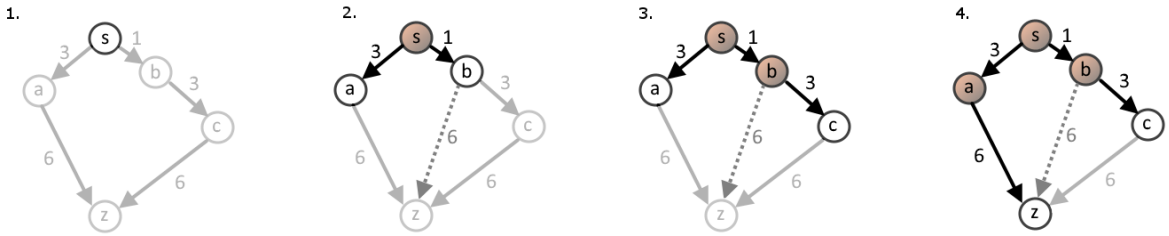

Abbildung 5: Suchreihenfolge einer A* Suche auf einem einfachen Baum mit drei Iterationen (2., 3., 4.) und der initiale Baum (1.). Die Suche erfolgt auf den Knoten s, a, b, c und z. Angabe der Heuristik und der tatsächlichen Kosten als Zahlenwert neben den Pfeilen.

Der Algorithmus findet eine komplette und optimale Lösung, wenn die verwendete Heuristik zwei Bedingungen, die im weiteren Verlauf mit zulässig und konsistent betitelt werden, erfüllt.

In Abbildung 5 ist eine A* Suche auf einem einfachen Baum zu sehen. Die Suche startet bei dem Knoten s des Baumes. Der initiale Baum mit Knoten s in der Open-Liste ist in (1.) zu sehen. Das Ziel der Suche ist mit dem Buchstaben z markiert. Die tatsächlichen Kosten stimmen mit den geschätzten Werten überein, und stehen in Zahlen neben den Kanten. Der Schätzwert der Luftliniendistanz zwischen den Knoten b und z ist über eine gepunktete Kante dargestellt, da keine tatsächliche Kante existiert.
In der ersten Iteration (2.) werden die f Werte der Nachfolgerknoten von s bestimmt. Die Knoten a und b werden in die Open-Liste aufgenommen. Der Knoten s wandert in die Closed-Liste.

$$f(a) = g(a) + h(a) = 3 + 6 = 9$$
$$f(b) = g(b) + h(b) = 1 + 6 = 7$$

Für die nächste Iteration wird der Knoten mit dem kleineren f Wert gewählt, also b. Dieser wird aus der Open-Liste entfernt und in der Closed-Liste aufgenommen. Dieser Schritt ist in Schaubild (3.) zu sehen. Der f Wert des Nachfolgerknoten c wird bestimmt:

$$f(c) = g(c) + h(c) = 4 + 6 = 10$$

Der Knoten c wird der Open-Liste hinzugefügt.
In Schaubild vier wird der Knoten a gewählt, da $f(a) = 9 < f(c) = 10$. Der Nachfolgerknoten von a ist der gesuchte Zielknoten z. Somit findet der Algorithmus die optimale Lösung von s nach z über a.

3.8 Iterativer-Tiefen A* (IDA*)

In vielen Anwendungen ist die Größe des Speichermediums beschränkt. Viele Suchalgorithmen sind aber so konzipiert, dass ihr Speicheraufwand mit Anzahl der Knoten einer Problemstellung wächst. Der iterative-Tiefen A* (IDA*) [2] beachtet diese Problematik.
Iterative Tiefensuche vereint die Vorteile von Breitensuche (Optimalität) und von Tiefensuche (geringerer Speicheraufwand), die in Kapitel 3.4 beschrieben sind. Die Suchstrategie ist Tiefensuche, jedoch wird die Suchtiefe auf ein Level beschränkt. Dieses Level startet bei 0 und wird

schrittweise erhöht. Darüber wird sichergestellt, dass die Tiefensuche vollständig ausgeführt wird und sich nicht auf endlos langen Pfaden verliert. Des Weiteren wird so die kürzeste Lösung gefunden. Iterative Tiefensuche kann den Eindruck erwecken, sehr aufwendig zu sein, da einige Knoten häufig durchlaufen werden. Der zusätzliche Aufwand hält sich allerdings in Grenzen, da sich bei typischen Bäumen die meiste Knoten unten im Baum befinden. Die wenigen Knoten an der Spitze des Baumes, die meistens betrachtet werden, haben keinen großen Einfluss auf die Gesamtanzahl betrachteter Knoten. Geht man von einem binären Baum aus, in dem jeder Knoten zwei Nachfolgerknoten hat, ist die Anzahl an Knoten in einer Tiefe des Baumes mindestens so groß, wie die Anzahl aller vorherigen Knoten.
Diese Idee der iterativen Tiefensuche wird dem A* und seinem heuristischen Suchkontext hinzugefügt. Der Unterschied liegt darin, dass die Tiefe nicht über einen Wert erhöht wird, sondern über den f Wert. Pro Iteration wird das Level über den kleinsten f Wert aller Knoten ermittelt, der größer ist, als der f Wert der vorigen Iteration.
IDA* ist anwendbar auf viele Probleme mit einheitlichen Schrittkosten. Mit diesem Verfahren lassen sich Probleme mit großem Suchraum lösen, bei denen A* den verfügbaren Speicherplatz bereits verbrauch hat, und somit keine Lösung findet.

3.9 Lernender Echtzeit A* (LRTA*)

Einige Problemstellungen haben eine Zeitschranke für das Ausführen von Aktionen. Der Algorithmus LRTA* [1] beachtet solche Einschränkungen der Zeit zur Laufzeit. Der Algorithmus führt einen A* Algorithmus aus, bis die Zeitgrenze erreicht wird. Die nächsten Knoten für die Expansion wird über $f = g + h$ ermittelt. Der Knoten mit dem kleinsten f Wert wird gewählt und der Pfad zu diesem Knoten wird ausgeführt. Dieser Knoten ist der neue Startknoten für den Algorithmus.

3.10 Lokaler Suchraum-Lernender Echtzeit A* (LSS-LRTA*)

LSS-LRTA* [6] ist eine Version des LRTA* der die A* Suche nutzt, um seinen lokalen Suchraum zu ermitteln und schnell zu lernen. Echtzeit Algorithmen [9] werden angewendet, wenn eine reale Zeiteinschränkung pro Aktion vorliegt. LSS-LRTA* besteht aus zwei Prozessen. Der erste Prozess führt eine A*-ähnliche Suche vom aktuellen Knoten des Agenten bis zu einem Zielknoten aus, solange das Expansionslimit noch nicht erreicht wurde. Im zweiten Prozess wählt der Algorithmus den besten Zustand auf der Suchgrenze aus und fügt diesen Zustand dem Pfad, der in einer Datenstruktur gespeichert wird, hinzu. Die Datenstruktur wird mit einem Backup Prozess geupdated. Der neue Wert ermittelt sich aus der Summe des h Wertes des besten Kindes und den Kantenkosten zwischen den beiden. Durch diesen intensiven Lernschritt erkennt der Algorithmus lokale Minima schneller als bisherige Echtzeit Suchalgorithmen.

3.11 Heuristik

Heuristische Suchalgorithmen verwenden Schätzwerte, um die Güte von Knoten zu bewerten. Diese Schätzwerte heißen Heuristik [10]. Typische Heuristiken sind die Luftliniendistanz oder die Manhattandistanz. Hier entspricht $h(n)$ einer Distanz zwischen dem Knoten n und einem Zielknoten. Heuristiken können die beiden Bedingungen:

- zulässig

- konsistent

erfüllen.

Die Bewertungsfunktion eines Algorithmus ist zulässig [2], wenn sie die tatsächlichen Kosten eines Pfades niemals überschreitet. Wenn $h(n)$ zulässig ist, überschreitet auch $f(n)$ niemals die tatsächlichen Kosten eines Pfades. Das liegt daran, dass $g(n)$ den Kosten vom Startknoten bis zum Knoten n des Pfades sind. Die Kosten von n bis zum Zielknoten werden mit $h(n)$ unterschätzt. Somit ist auch die Summe der Kosten von:

$$f(n) = g(n) + h(n) \tag{3}$$

eine Unterschätzung der tatsächlichen Kosten von s nach t über n. Eine weit verbreitete Heuristik ist die Luftliniendistanz. Bei dieser Wahl von $h(n)$ gibt es keinen Weg, der kürzer ist als die Luftlinie zwischen zwei Punkten. Dadurch entsteht niemals eine Überschätzung.

Eine Heuristik kann zudem konsistent sein. Wenn eine Aktion a von Knoten n zu dessen Nachfolgeknoten n' führt, dann sind die Kosten von n nach n':

$$c(n, a, n') \tag{4}$$

Die Heuristik ist konsistent, wenn sie die folgende Ungleichung erfüllt:

$$h(n) \leq c(n, a, n') + h(n') \tag{5}$$

Das bedeutet, dass die Schätzung der Kosten von n bis zum Zielknoten niemals größer als die Kosten von n zu n' addiert mit der Schätzung von n' bis zum Zielknoten sind.

3.12 Probleme und Fehlermöglichkeiten

Bei der A* basierten Suche werden die zu expandierenden Knoten anhand ihres f Wertes ausgewählt. Diese Auswahl kann zu Problemen [1] führen, da diese Suchverfahren nicht auf Fehler der Heuristik reagieren. Die Fehler entstehen, da die Heuristik die tatsächlichen Kosten zum Zielknoten unterschätzt.

Es kann passieren, dass mehrere Knoten mit dem gleichen f Wert in der Open-Liste vorkommen. Die Knoten mit demselben f Wert bilden eine Schicht um den Startknoten. Diese Schicht wird f Layer genannt. Ein f Layer ist in Abbildung 6 (a) durch gepunktete Kreise dargestellt. Die Heuristik unterschätzt die tatsächlichen Kosten, enthält aber Fehler. Sie ist oberhalb pro Spalte angegeben. Es ist aber nicht gewährleistet, dass alle Knoten auf dem selben f Layer dem Zielknoten gleich nahe sind. Manche Knoten des f Layer führen sogar in die entgegengesetzte Richtung von dem Zielknoten (links von der Figur).
Wenn der Algorithmus nun einen solchen Knoten auswählt, verlässt der Algorithmus den optimalen Weg vom Startknoten zum Zielknoten. Dadurch kann es passieren, dass mehr Zeit oder Schritte benötigt werden, um die Lösung zu finden. Eine Möglichkeit der Problematik entgegenzusteuern besteht darin, aus den Knoten mit den kleinsten f Werten einen auszuwählen, der zusätzlich den kleinsten h Wert hat. Für den A* Algorithmus ist diese Methode eine gute Option, da ein kleineres $h(n)$ schneller zu der gesuchten Lösung führen sollte.
Wenn das Suchverfahren ein Zeitlimit für jeden Schritt hat, reicht diese Methode nicht aus, da

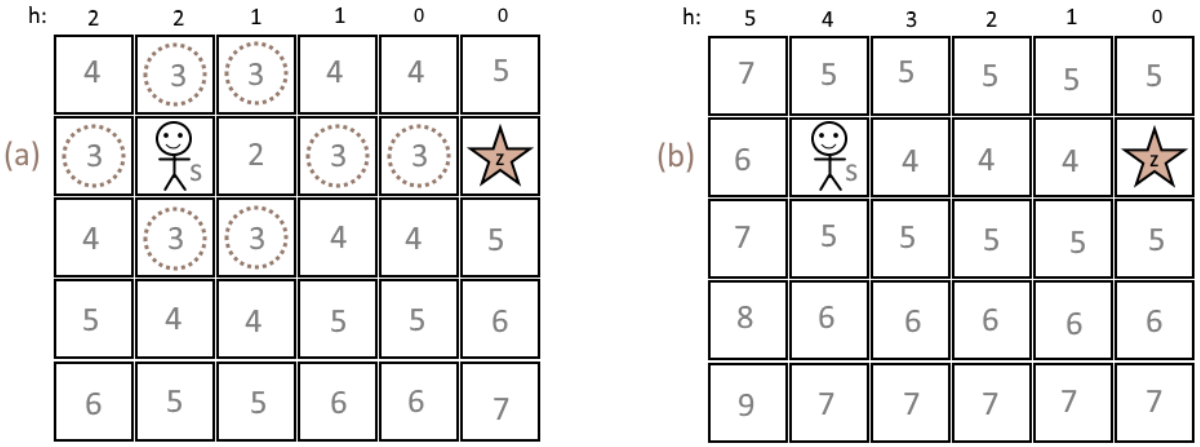

Abbildung 6: (a) Beispiel der Heuristik mit Fehler (b) Beispiel der Heuristik ohne Fehler. Das Feld mit der Figur ist mit s markiert. Dieses Feld ist der Startknoten der Suche. Das Feld z, mit einem Stern markiert, ist der Zielknoten. In den Feldern ist die Bewertung f eingetragen. Oberhalb des Spielfeldes ist die Heuristik spaltenweise angegeben.

der Fall, dass der f Layer noch nicht vollständig generiert wurde, die Zeit aber bereits abgelaufen ist, eintreffen kann. Dadurch kann es sein, dass der Knoten mit dem kleinsten f und h Wert noch gar nicht betrachtet wurde. In diesem Fall würde sich der Algorithmus auf jeden Fall für einen nicht optimalen Knoten entscheiden.
Eine weitere Problematik tritt in unvollständigen f Layern auf, wenn das Zeitlimit erreicht wird und ein f Layer noch nicht vollständig ist, befinden sich alle Knoten des Layers die bereits betrachtet wurden in der Closed-Liste. Die Knoten, die noch nicht betrachtet wurden, sind weiterhin in der Open-Liste. Zusätzlich befinden sich alle Kindknoten, der bereits gesehenen Knoten dieses Layers ebenfalls in der Open-Liste. Deren f Wert ist größer oder gleich dem f Wert des Layers. Diese Knoten bilden den f' Layer. Der Algorithmus muss nun einen Knoten aus der Open-Liste wählen zu dem die Spielfigur sich bewegt. Er wird einen Knoten aus dem f Layer wählen der noch in der Open-Liste ist. Jedoch sind die Konten des f' Layers vielversprechender.

4 Dynamische $\hat{f}$ Suche

Der Algorithmus der dynamischen $\hat{f}$ Suche [1] basiert auf einer Baumsuche. Das Ziel der Suche ist es, einen Zielknoten von dem Startknoten aus zu erreichen. Diese Variante der Baumsuche versucht den durch die Heuristik entstandenen Fehler für jeden Knoten zu korrigieren. Dadurch soll die Zeit zur Lösungsfindung reduziert werden.
Abbildung 7 zeigt in (a) einen Startknoten s und einen Zielknoten z. Die Heuristik, um von s nach z zu gelangen, ist mit $h(s)$ eingezeichnet. Diese Heuristik kann einen Fehler beinhalten. In (b) ist die Fehlermöglichkeit der Heuristik $h(n)$ eingezeichnet, diese versucht die Strecke zwischen den Knoten n und z zu schätzen. Die tatsächlichen Kosten $g(n)$, um von Knoten s nach n zu gelangen, sind bekannt.

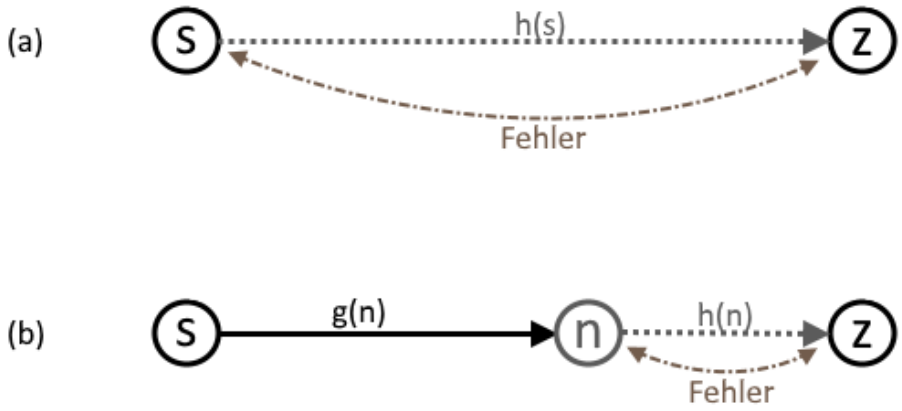

Abbildung 7: (a) Eine Heuristik $h(s)$ und ihr Fehler zwischen den Knoten s und z. (b) Ein Iterationsschritt und der Fehler zwischen den Knoten n und z. Die tatsächlichen Kosten $g(n)$ um von Knoten s zu Knoten n zu gelangen und die Heuristik $h(n)$ sind abgebildet.

Die dynamische $\hat{f}$ Suche [9] steuert der durch die Heuristik erzeugten Ungenauigkeit entgegen. Der Algorithmus basiert auf LSS-LRTA* mit zwei Modifikationen. Anstelle eine feste Zeitgrenze zu nutzen, wird diese von dem dynamischen $\hat{f}$ nach jeder Iteration gesetzt, basierend auf den Aktionen, die während der Ausführung für die jeweiligen Aktionen gesetzt wurden.
Die zweite Modifikation besteht darin, eine nicht zulässige Heuristik, die im weiteren Verlauf der Arbeit mit $\hat{h}$ bezeichnet wird, zu verwenden. Daduch verändert sich die in A* beschriebene Heuristik $f(n)$ zu:

$$f(n) = g(n) + \hat{h}(n) \tag{6}$$

Da die Heuristik ein Schätzwert ist, kann $\hat{h}(n)$ von den tatsächlichen Kosten abweichen. Die Bewertungsfunktion $f(n)$ ist keine untere Schranke mehr. Diese Abweichung wird durch einen

Fehlerterm dargestellt. Der Fehlerterm ermittelt sich aus dem Durchschnitt der bisherigen ϵ Werte multipliziert mit d(n). Das ϵ gibt den Fehler der Heuristik an, der pro Iteration gemacht wird.
In jeder Iteration wird ein neues ϵ berechnet. Der Fehler der während der Iteration Auftritt wird mit $\bar{\epsilon}$ dargestellt. Dieser Wert ergibt sich aus der Differenz des f Wertes des aktuellen Knoten n und dem besten f Wert eines Nachfolgerknoten n', der über eine Kante vom aktuellen Knoten aus erreicht werden kann. Wobei die Menge N alle Nachfolgerknoten von n enthält.

$$\bar{\epsilon} = f(n) \text{ - } min\{f(n')\},\ n' \in N$$

Zur Ermittlung des ϵ wird die Summe aller bisherigen $\bar{\epsilon}$ durch die Anzahl aller $\bar{\epsilon}$ Werte geteilt. Die Distanz $d(n)$ gibt an, wie viele Aktionen bis zum Ziel- oder Endzustand noch durchgeführt werden können. Der Term

$$d(n) * \epsilon \tag{7}$$

korrigiert den Fehler, der durch die Heuristik entsteht.

Die Auswahl für den Algorithmus, welcher Knoten als nächstes expandiert werden soll, wird über $\hat{f}$ ermittelt:

$$\hat{f}(n) = g(n) + \hat{h}(n) + d(n) * \epsilon \tag{8}$$

Für den Fall, dass die Heuristik ideal ist, gilt:

- f(n) = f(n')
- $\hat{h}(n) = \hat{h}(n')$+c(n, n')
- g(n) = g(n') - c(n, n')

Dadurch lässt sich der entstandene Fehler in der Heuristik durch die Differenz ermittelt.

Der Algorithmus kann durch folgenden Pseudocode dargestellt werden:

1. Solange, bis ein Zielknoten gefunden wurde:
2. Füre Iterationen einer Bestensuche auf $\hat{f}$ aus, bis die Zeitgrenze erreicht wird.
3. Update die Heuristik der Knoten in der Closed-Liste.
4. Der Knoten n, aus der Open-Liste, ist der mit dem niedrigsten $\hat{f}$ Wert.
5. Führe den Pfad nach n aus.
6. Die Zeitgrenze entspricht der Zeit, um n zu erreichen.
7. Die Open-Liste enthält nur den Knoten n, entferne alle Inhalte der Closed-Liste.
8. Starte bei Schritt 1.

In Abbildung 4 sind sieben Teilschritte einer Iteration des 2. Schrittes des Pseudocodes der dynamischen $\hat{f}$ Suche abgebildet. Der Startknoten und somit aktuelle Knoten der Iteration ist a. Die Werte $f(a)$ und $\hat{f}(a)$ existieren bereits. Im ersten Teilschritt (1.) werden die f Werte, $f(b)$ und $f(c)$, der Nachfolgerknoten von a berechnet.
Es wird der kleinste f Wert ($f\,min$) der Nachfolgerknoten gewählt (2.). Der dritte Baum stellt die Bestimmung des Betrags der Differenz: $f(a) - f\,min$ dar. Der Differenzwert im Betrag erzeugt den neuen ϵ Wert.
In Teilschritt (4.) wird aus dem neuen ϵ Wert und dem alten der Durchschnitt aller berechnet. Es wird $\bar{\epsilon}$ erzeugt, mit dem in Schritt (5.) die $\hat{f}$ Werte von b und c berechnet werden können.
Für die nächste Iteration wird ein neuer aktueller Knoten benötigt. Dieser wird in Schritt (6.) ermittelt, indem von allen Knoten, die bereits einen $\hat{f}$ Wert haben, aber noch nicht als aktueller Knoten gewählt wurden der Knoten mit dem kleinsten $\hat{f}$ Wert ausgewählt wird. In Teilschritt (7.) wird dieser Knoten, hier Knoten c als neuer aktueller Knoten gesetzt und die nächste Iteration kann beginnen.

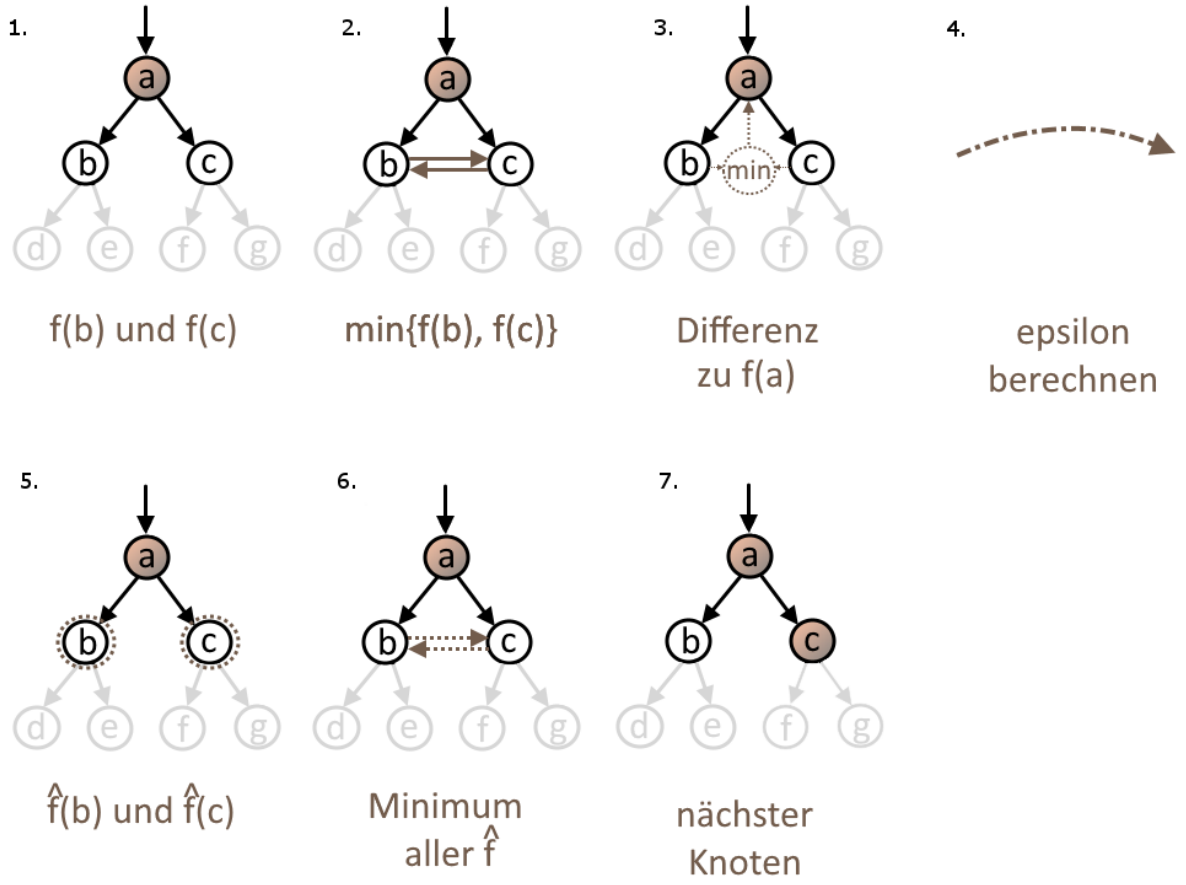

Abbildung 8: Iterationsschritt einer $\hat{f}$ Suche mit sieben Teilschritten auf einem einfachen Baum. Abgebildete Knoten sind: a, b, c, d, e, f und g.

5 Experiment

Die Anpassungen der dynamischen $\hat{f}$ Suche, die in Kapitel 4 zu sehen ist, werden extrahiert und auf ein Szenario übertragen, in dem die Heuristik 3.11 weder konsistent noch zulässig ist. Die Versuchsumgebung entstammt dem GVG-AI Wettbewerb, der in Kapitel 2.5 beschrieben ist. Der Wettbewerb stellt ein Framework zur Verfügung, welches zahlreiche zweidimensionale Spiele mit verschiedenen Eigenschaften enthält. Auf einigen dieser Spiele wird die auf das Framework angepasste dynamische $\hat{f}$ Suche, die in Kapitel 5.2 erläutert wird, angewendet. Um die Anpassungen der dynamischen $\hat{f}$ Suche bewerten zu können, wird auf den selben Spielen mit gleichen Voraussetzungen eine Variante mit f betrachtet.

5.1 Auswahl der Spiele

Für die Auswertung der Anpassungen ist es notwendig, die Variante der dynamischen $\hat{f}$ Suche mittels Testdaten zu ermitteln. Das in Kapitel 2.6 vorgestellte GVGAI Framework stellt über 100 verschiedene Spiele zum Testen bereit. Einige dieser Spiele sind komplett deterministisch und andere haben stochastische Effekte. Da die Auswertung klare Daten liefern soll und keinen Spielraum für Spekulationen bereit halten darf, wurden ausschließlich Spiele ohne stochastische Effekte gewählt. Dadurch basieren die Ergebnisse auf der gleichen Ausgangssituation. Suchalgorithmen wie A*, in Kapitel 3.7 beschrieben, betrachten nur einen möglichen Spielzustand, können aber nicht davon ausgehen, wenn der Agent einen Pfad zum Zielknoten gefunden hat, dass dieser bei der Ausführung der Schritte ebenso durchgeführt werden kann. Für Spiele mit stochastischen Effekten gibt es umfangreiche Forschung die Monte Carlo Baumsuche [5] behandelt.
Die Auswahl an repräsentativen deterministischen Spielen hat sich über die verschiedenen Spielmechaniken, die Komplexität und Kombinationen aus beiden ergeben. Die Anzahl an Aufgaben, die zu lösen sind, um das Spiel zu gewinnen, hat nur bedingt Einfluss auf die Schwierigkeit des Spieles. Die ausgewählten Spiele sind:

Spiel	**begehbare Felder**	**Aufgaben**
Bait	unterschiedlich (9-99)	schieben, sammeln, Ziel finden, kombinieren
Brainman	mittelgroß (36-108)	schieben, sammeln, Ziel finden, rutschen
Catapults	mittel (44-75)	Ziel finden
Chainreaction	groß (>100)	schieben, kombinieren, rutschen, indirekt
Colourescape	groß (>100)	schieben, sammeln, Ziel finden, kombinieren
Modality	klein (15-36 {99})	schieben
Sokoban	mittel(19-64)	schieben

Durch die Vorgaben des GVGAI Wettbewerbes sind die Agenten in ihren Möglichkeiten beschränkt. Der Agenten kann zwei GB Speicherplatz nutzen. Vor dem ersten Schritt steht dem Agenten eine Sekunde zur Verfügung, um Informationen zu sammeln und mit der Suche zu starten. Für jeden weiteren Schritt sind es 40 ms. Das Speichern von Informationen über Spiele und Level hinweg ist nicht gestattet. Das bedeutet, der Agent muss die benötigten Informationen zum Lösen der Level jedes Mal neu sammeln.
Der Agent eines Suchalgorithmus steuert eine Spielfigur des jeweiligen Spiels. Das Spielfeld und die Position der Spielfigur wird von dem Algorithmus über die Zustände ermittelt. Die Spielfigur kann sich einen Schritt auf dem Spielfeld bewegen, dafür stehen dem Agenten die folgenden Aktionen zur Verfügung:

- nach oben bewegen
- nach unten bewegen
- nach links bewegen
- nach rechts bewegen
- stehen bleiben
- Inventar nutzen

Es kann passieren, dass Aktionen den gleichen Effekt haben wie „stehen bleiben“, da es Spiele gibt, in denen dem Agenten kein Inventar zur Verfügung steht, das er benutzen könnte. Das Inventar kann sich in manchen Spielen verbrauchen, entweder beim Benutzen oder durch andere Effekte. Wenn der Agent durch eine Bewegungsaktion in eine WAND laufen würde, befindet sich der Agent nach Ausführung des Schrittes weiterhin auf dem vorigen Spielfeld. Des Weiteren befinden sich auf manchen Spielfeldern Objekte, mit denen die Spielfigur direkt, indirekt oder gar nicht interagieren kann.
Die verschiedenen Objekte sind:

Objekt	**Eigenschaft**
Wand	Spielfeld, auf dem sich die Spielfigur niemals befindet
Stein	lässt sich durch dagegenbewegen verrutschen
indirekter Stein	lässt sich verrutschen, in dem ein anderes Objekt gegen ihn rutscht
Kiste	lässt sich durch dagegenbewegen verschieben
Diamant, Pilz	kann eingesammelt werden und erhöht dabei den Punktestand
Fahne	verändert das Inventar der Spielfigur, kann bestimmte Objekte verschieben
Schlüssel	verändert das Inventar der Spielfigur, kann bestimmte Objekte vernichten
Ziel	Spielfeld, das durch betreten der Spielfigur zum Sieg des Spieles führt
Tor	kann von Objekten vernichtet werden
Höhle	vernichtet Objekte, kann den Punktestand erhöhen
Katapult	befördert die Spielfigur zu einem anderen Spielfeld

Die unterschiedlichen Aufgaben der Spiele lassen sich noch weiter spezifizieren. Die Spielfigur hat im Spiel die Möglichkeit gegen Objekte zu laufen. Bei dem Objekt WAND befindet er sich danach auf dem Spielfeld, auf dem er sich zuvor befand. Diese Aktion entspricht der Aktion STEHEN BLEIBEN. Wenn er geben Objekte läuft, die sich SCHIEBEN oder RUTSCHEN lassen, befindet sich die Spielfigur auf dem Spielfeld, auf dem sich zuvor das Objekt befunden hat. Objekte die sich SCHIEBEN lassen, bewegen sich ein Spielfeld weiter in die Richtung in die sich die Spielfigur bewegte, aber nur wenn dieses Spielfeld frei ist. Sollte das Feld durch ein anderes Objekt blockiert sein, dann endet die Bewegung der Spielfigur auf dem vorigen Feld und entspricht ebenfalls der Aktion STEHEN BLEIBEN. Die Objekte die sich RUTSCHEN lassen, gleiten so lange in die Richtung in die sich die Spielfigur bewegt, bis sie gegen ein Objekt treffen. Um die Aufgabe ZIEL FINDEN zu lösen, muss die Spielfigur zu dem entsprechenden Spielfeld laufen, auf dem sich das Objekt ZIEL befindet.
Um die Komplexität eines Spieles zu ermitteln müssen einige Punkte beachtet werden. Beispielsweise sind im Spiel BAIT besondere Anforderungen von Bedeutung. In dem Spiel müssen KISTEN verschoben werden, TORE vernichtet werden, SCHLÜSSEL gesammelt werden und dann das ZIEL erreicht werden. Zusätzlich kann der Punktestand erhöht werden, indem die Spielfigur PILZE einsammelt. Durch die Rahmenbedingungen des Frameworks, der Reihenfolge, in der die Aufgaben erfüllt werden müssen und der räumlichen Einschränkung für die Spielfigur ergibt sich ein komplexes Spiel.

5.2 Anpassung des Suchverfahren

Die in Kapitel 4 beschriebene dynamische $\hat{f}$ Suche wird an das Framework des GVG-AI Wettbewerbes, der in Kapitel 2.5 vorgestellt wird, angepasst. Die Anpassung beinhaltet eine Suche, die zur Laufzeit statt findet und erst abbricht, wenn ein Zielzustand gefunden wurde. Dieses Verhalten wird realisiert, in dem der Agent die Spielfigur solange nicht bewegt, bis er eine Lösung gefunden hat. Der Agent nutzt die 40 ms, die er hat, um einen Schritt durchzuführen, um seinen Suchbaum weiter aufzuspannen. Am Ende der Zeit, gibt er der Spielfigur den Auftrag die Aktion STEHEN BLEIBEN auszuführen. Durch das Ausführen der Aktion STEHEN BLEIBEN agiert der Agent zur Laufzeit, kann die Suche aber von dem Startzustand weiterführen, da dieser sich nicht ändert.
Wenn die Grenze des Speichers von zwei GB erreicht wird, wird der Teilpfad ausgeführt, der bisher den höchsten Punktestand hat.
In den ausgewählten Spielen können Punkte gesammelt werden, zum Teil unabhängig davon ob im späteren Verlauf der Suche ein Zielzustand gefunden wird. Die erreichten Punktestände sind für die potentielle Auswertung des GVG-AI Wettbewerbes entscheidend. Daher fließt der Punktestand in den Term für die tatsächlichen Kosten mit ein. Um das Vorantreiben des Punktesammelns zu fördern, wird der Punktestand mehr bis genauso viel gewertet, wie die verbrauchte Anzahl an Schritten. Die Erhöhung des Schrittzählers beträgt pro Zeiteinheit Eins. Dieser Schrittzähler wird durch die Anzahl der zu Beginn maximalmöglichen Schritte geteilt. Dadurch wird gewährleistet, dass die erreichten Punkte mehr Gewicht haben, als die verbrauchten Schritteinheiten. Durch die potentielle Erhöhung des Punktestandes wird das Endergebnis positiv beeinflusst. Da ein kleinerer $g(n)$ Wert besser ist, wird der Punktestand p negiert. Die Anzahl der maximalen Schritte, die in dem jeweiligen Spiel zur Verfügung stehen, um dieses zu lösen wird mit $\#maxS$ bezeichnet. Die Anzahl an Schritten, die der Agent bereits verwendet hat, wird mit $\#S$ angegeben. Daraus ergibt sich folgende Formel:

$$g(n) = -p + ((1/\#maxS) * \#S) \tag{9}$$

Die Luftliniendistanz lld zum nächten Ziel dient als Heuristik. Das nächste Ziel wird über einen Zielsucher [5] ermittelt. Da die Heuristik weder konsistent noch zulässig ist, wird sie mit $h'(n)$ bezeichnet.

Um $d(n)$ zu ermitteln, bestimmt der Agent in der ersten Sekunde der Suche wie viele Schritte er zur Verfügung hat, um das Spiel zu lösen. Meistens sind Spiele nach 2000 Schritten verloren, wenn vorher keine Lösung gefunden wurde. Dieser Wert ist nicht über alle Spiele hinweg gleich. Außerdem ist die zur Verfügung stehende Anzahl an Schritten, um das Spiel zu lösen, dem Agenten unbekannt und muss daher zu Beginn bestimmt werden. Aus diesem Wert ergibt sich:

$$d(n) = \#maxS - \#S \tag{10}$$

Wenn der Zielzustand gefunden wurde, werden die Aktionen die zu diesem geführt haben durchgeführt. Der Agent schickt der Spielfigur die entsprechenden Aktionen, um das Spiel zu gewinnen.
Um diese Anpassung der dynamischen $\hat{f}$ Suche zu starten muss ein initialer ϵ Wert gesetzt werden. Die Wahl des initialen ϵ Wertes wird im folgenden Kapitel 5.3 behandelt.

5.3 Initialisierung des ϵ Wertes

Während der dynamischen $\hat{f}$ Suche wird für jeden Schritt das ϵ dynamisch angepasst. Um die $\hat{f}$ Werte der Nachfolgeknoten des Startknotens zu berechnen, muss mit einem initialen ϵ Wert gestartet werden. Hierzu wird ein eigenes Experiment gestartet, in dem ϵ Werte zwischen null und dem maximal erreichbaren Punktestand von 2000 auf ihre Auswirkung auf den Algorithmus getestet werden. Alle Level der sieben Spiele werden auf 45 ϵ Werten getestet. Jeder Startwert wird auf 50 Testlevel des Level getestet. Der erreichte Punktestand, die Anzahl verbrauchter Schritte sowie die Angabe, ob das jeweilige Testlevel gewonnen (Zahlenwert 1) oder verloren (Zahlenwert 0)wird, werden für die Auswertung gespeichert.

5.4 Datenerzeugung für Vergleich

Um die mögliche Verbesserung des A* Algorithmus durch die $\hat{f}$ Variante zu belegen, wird sie mit einer A* Variante verglichen. Die A* Variante läuft unter den gleichen Bedingungen wie die $\hat{f}$ Variante, jedoch ohne Beachtung des Fehlerterms. Um an der Laufzeit der einzelnen Schritte keine Verzerrung zu erhalten, werden die Berechnungen von ϵ und $\hat{f}$ auch in diesem Experiment durchgeführt, aber nicht in der Auswahl des nächsten Knoten berücksichtigt:

$$f(n) = g(n) + h'(n) \tag{11}$$

$$g(n) = -p + ((1/\#maxS) * \#S) \tag{12}$$

$$h'(n) = lld \tag{13}$$

Die Werte von $g(n)$ und $h'(n)$ entsprechen somit den Werten der $\hat{f}$ Variante, die in Kapitel 5.2 beschrieben sind. Pro Level wurden 50 Durchläufe gestartet, um aussagekräftige Vergleichsdaten zu erzeugen. Für jedes Level wird aufgezeichnet, wie viele Schritte benötigt werden, welcher Punktestand erreicht wird und ob das Spiel gewonnen oder verloren wird. Diese Daten dienen als Vergleich zu den Daten, die von der dynamischen $\hat{f}$ Variante erzeugt werden.

Level	#Spielfelder	#Objekte	Dichte
0	9	4	0,44
1	29	6	0,2
2	70	26	0,37
3	99	77	0,7
4	28	18	0,64

Tabelle 1: Daten zu dem Spiel BAIT

6 Auswertung

In diesem Kapitel werden die erzeugten Daten graphisch dargestellt und ausgewertet. In den folgenden Grafiken sind die erzeugten Daten abgebildet. Die Ergebnisse der getesteten initialen ϵ Werte auf der dynamischen $\hat{f}$ Suche sind in schwarz zu sehen. Zur Darstellung wurde ein Box-Wiskers-Plot gewählt, der Ausreißer mit schwarzen Punkten markiert. In den Abbildungen ist der Sonderfall mit initialem ϵ Wert null (eps=0) jeweils mit einem violetten Plus markiert. Zusätzlich wird der Vergleichswert ohne Fehlerterm (ohne FT) durch grüne Kreuze gekennzeichnet. In der Variante wurde $f(n) = g(n) + h'(n)$ zur Auswahl des nächsten Knoten verwendet.
Auf der y-Achse befindet sich auf einem Intervall zwischen 0 und 1 die Siegrate der Testlevel. Diese gibt an, wieviele der getesteten Testlevel gewonnen beziehungsweise verloren werden. Die Siegrate 0 bedeutet, dass alle Testlevel verloren wurden. Wenn 25 Testlevel gewonnen und 25 Testlevel verloren wurden, wird eine Siegrate von 0.5 erreicht. Wenn die Siegrate 1 beträgt, wurden alle Testlevel gewonnen.
In manchen Spielen fließt der Punktestand mit in die Auswertung ein. Für diese Spiele wurde ein y-Achsen Intervall zwischen 0 bis 2 gewählt. Die Siegrate $siegR$ und der normalisierte maximal erreichte Punktestand P, werden als Güte G bezeichnet.

$$G = siegR + P \tag{14}$$

In diesen Fällen ist die Güte auf der y-Achse aufgetragen.

6.1 Spiel BAIT

BAIT ist ein Spiel, das gelöst werden kann, indem KISTEN in HÖHLEN, oder auf andere Felder geschoben werden. Dadurch ergeben sich Wege zu einem Objekt SCHLÜSSEL, das eingesammelt werden kann. Nur mit dem SCHLÜSSEL kann das Spielfeld betreten werden, wodurch das Spiel gewonnen wird. Das Erreichen dieses Feldes erhöht den Punktestand um fünf Punkte. Jede KISTE, die in eine HÖHLE geschoben wird, bringt einen Punkt.

In Abbildung 9 sind die Ergebnisse levelweise zu sehen. Die Werte zwischen null und zwei kommen durch G zustande. Der Abbildung ist zu entnehmen, dass die Level 0 und 1 auf allen getesteten Varianten alle Testlevel mit maximalem Punktestand gewinnen. Bei Level 2 ist zu sehen, dass 25 % der initialen ϵ Werte ein besseres Ergebnis erzielen als die getesteten Level ohne Fehlerterm.
Innerhalb der 25 % befindet sich der mit null initialisierte ϵ Wert. Die Variante ohne Fehlerterm hat in keinem Level besser abgeschnitten. In Level 4 ist die Rate des initialen ϵ Wertes von null höher als der obere Wisker.

Die Siegrate scheint anhängig von der Anzahl an Objekten und Spielfeldern zu sein. Als Dichte wird die Anzahl an Objekten pro Anzahl an Spielfeldern bezeichnet. Diese Werte sind der

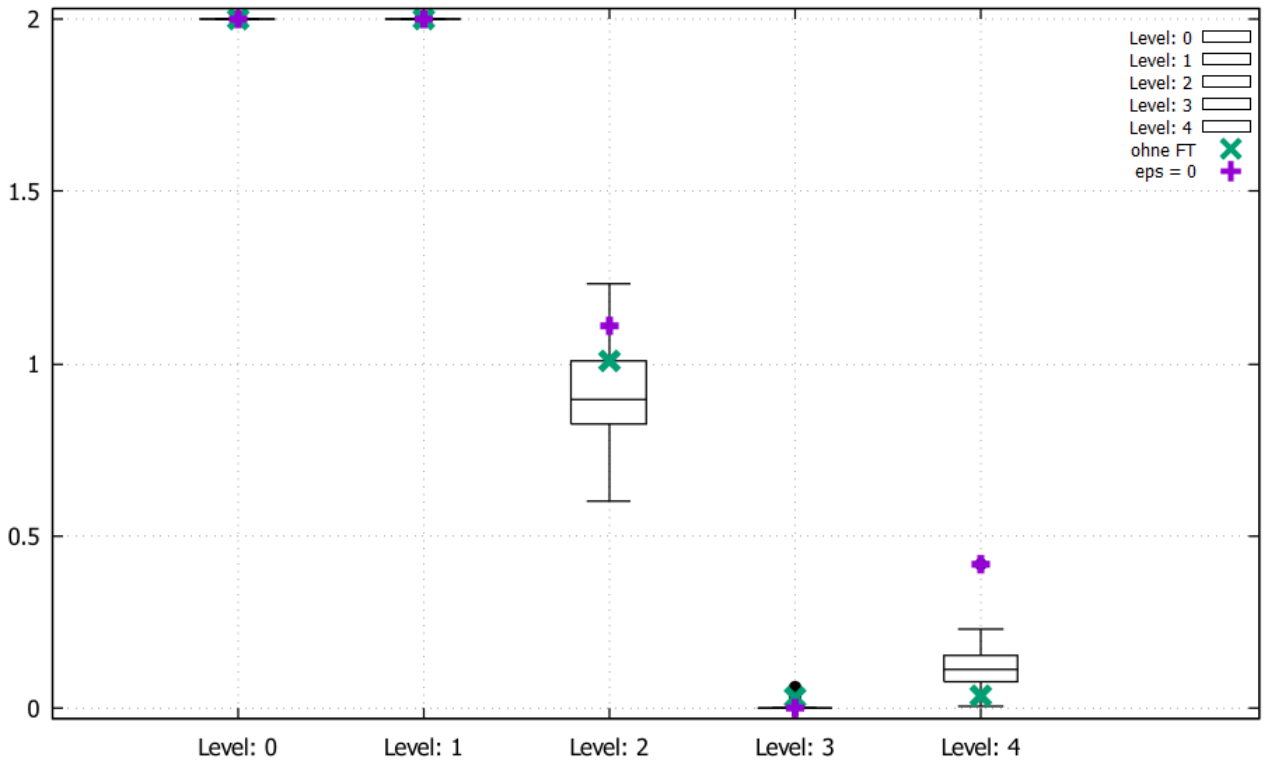

Abbildung 9: Levelweise Darstellung der Siegchancen für das Spiel BAIT, im Vergleich beider Varianten. Zusätzliche Information für initialen ϵ Wert von 0.

Tabelle 1 zu entnehmen. Je höher die Dichte, desto seltener werden die Testlevel gewonnen. Eine Ausnahme ist in Level 0 zu sehen, hier ist die Dichte mit 0,44 höher als die des Levels 2. Die Güte hingegen liegt bei einem Wert von 2,0. Zu beachten ist, dass die Anzahl an Spielfeldern mit 9 niedriger ist, als die des Levels 2 mit 70.

6.2 Spiel CATAPULTS

Das Spiel CATAPULTS wird durch Erreichen des Feldes mit dem Objekt ZIEL gewonnen. Es gibt wenige Spielfelder, auf denen sich die Spielfigur unbeschadet bewegen kann. Angrenzendes Felder können durch Betreten zum Verlieren des Spieles führen. Auf manchen Feldern befindet sich ein Objekt KATAPULT, welches die Spielfigur auf ein anderes Feld in einer bestimmten Richtung befördert. Das nutzen von KATAPULTEN erhöht den Punktestand um eins. Einige dieser KATAPULTE befördern die Spielfigur Felder, durch die das Spiel verloren geht. Das ZIEL kann nur durch Nutzen der richtigen KATAPULTE erreicht werden.

In Abbildung 10 sind die Güten des Spiels CATAPULTS levelweise dargestellt. Die Vergleichsvariante ohne Fehlerterm hat in allen Leveln schlecht abgeschnitten. Diese Version gewinnt kein Spiel auf den Leveln 0,1,2 und 4. In Level 3 befindet sich die Güte unterhalb der oberen Quantilline.
In den Leveln 0 und 4 liegen nicht ausreichend viele Daten zur Auswertung der Variante mit Fehlerterm vor. In den anderen Leveln gibt es mindestens 25% der getesteten initialen ϵ Werte, die eine bessere Güte erzielen. Die Initialisierung von ϵ mit null hat in den Leveln 1 und 2 zu der besten Güte geführt.

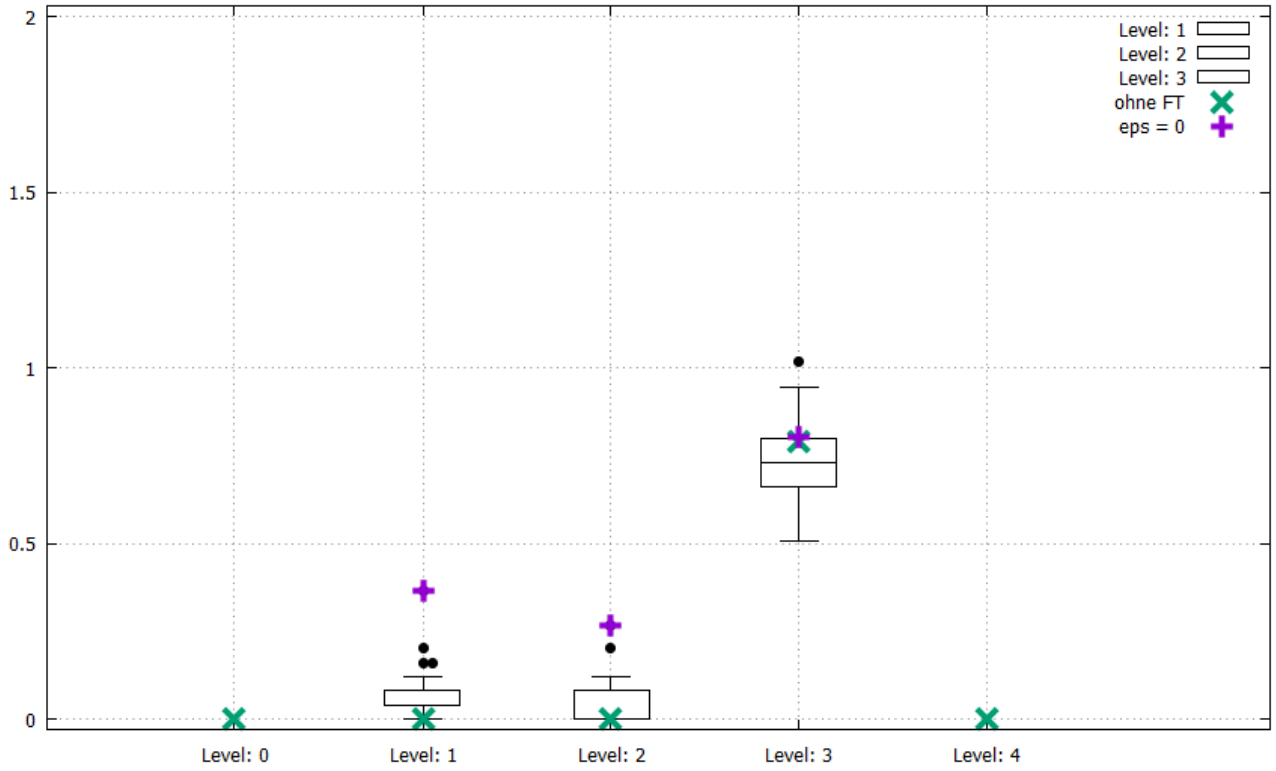

Abbildung 10: Levelweise Darstellung der Siegchancen für das Spiel CATAPULTS, im Vergleich beider Varianten. Zusätzliche Information für den initialen ϵ Wert von 0.

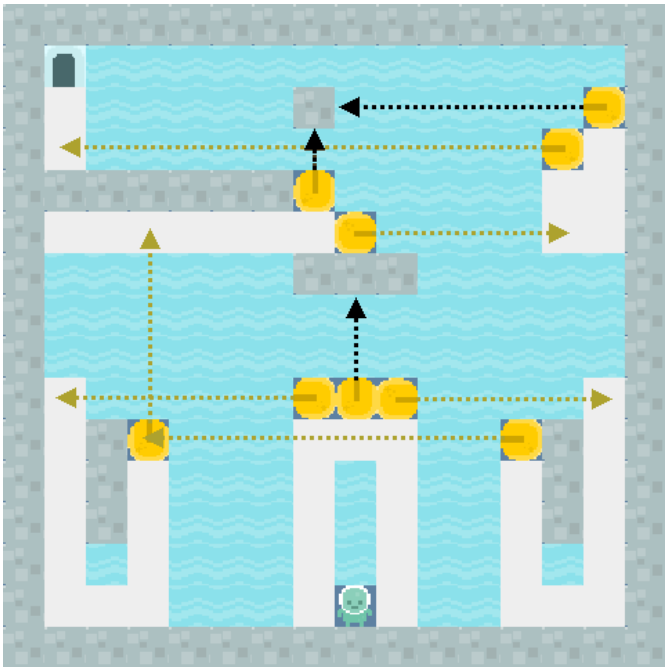

Abbildung 11: Level 3 des Spieles CATAPULTS. Die gelben Katapulte befördern die Spielfigur auf das Spielfeld, auf dem sich die Pfeilspitze befindet. Die schwarzen Pfeile zeigen die Flugbahn der KATAPULTE an, die mit einer Niederlage des Spieles enden.

Die Spielfeldgröße ist über alle Level hinweg 196. In Level 2 und 3 sind jeweils drei KATAPULTE vorhanden, die dafür verantwortlich sind, dass das Spiel nach dem Nutzen nicht mehr gewonnen

werden kann. Die Flugbahnen der KATAPULTE für das Level 3 sind in Abbildung 11 eingezeichnet. In schwarz sind die Flugbahnen markiert, durch die das Spiel verloren geht. In Level 1 gibt es 4 solche KATAPULTE. Die Anzahl an nutzbaren Spielfeldern, um zum ZIEL zu gelangen berträgt für Level 1 48, für Level 2 41 und für Level 3 55. Die Güte scheint abhängig von der Anzahl an nutzbaren Spielfeldern und den KATAPULTEN, die zu nutzen sind um zum ZIEL zu kommen, zu sein.

6.3 Spiel CHAINREACTION

Um das Spiel CHAINREACTION zu gewinnen, müssen alle Objekte vom Typ INDIREKTER STEIN durch rutschen in ein bestimmtes Feld vernichtet werden. Jedes Vernichten erhöht den Punktestand um zwei. Die Objekte INDIREKTER STEIN lassen sich nur verrutschen, indem ein Objekt STEIN gegen sie gerutscht wird. Die Objekte rutschen so lange in eine Richtung, bis sie gegen ein anderes Objekt treffen. Es gibt Felder, die beim Betreten oder indem Objekte auf sie rutschen, zum Verlieren des Spieles führen. Das Spiel endet bereist nach 1500 Schritten, wenn zuvor noch kein Sieg errungen wurde.

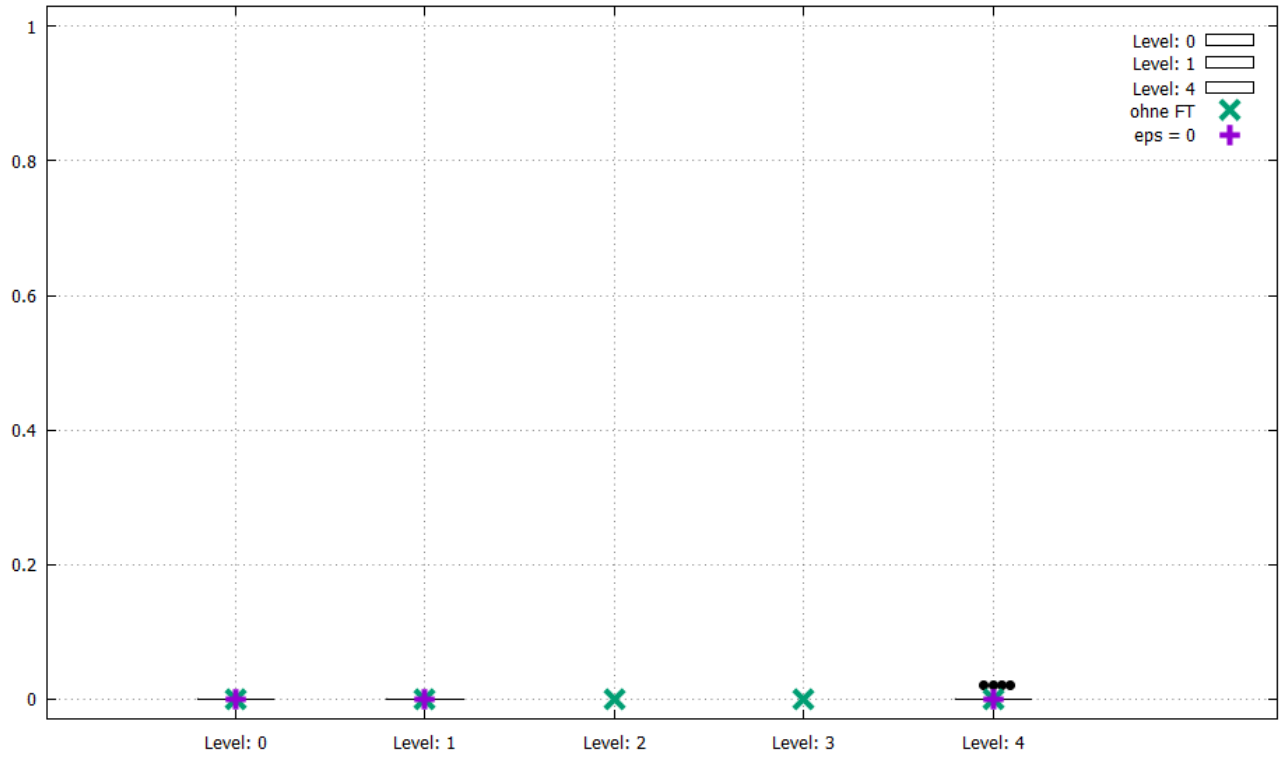

Abbildung 12: Levelweise Darstellung der Siegchancen für das Spiel CHAINREACTION, im Vergleich beider Varianten mit verkürzter y-Achse. Zusätzliche Information für initialen ϵ Wert von 0.

Für dieses Spiel liegen wie in Abbildung 12 zu sehen, nicht alle Daten vor. In den Level 2 und 3 existieren nicht ausreichend viele Daten, die Aussagen zur Siegrate der dynamischen $\hat{f}$ Variante zulassen. In den Level 0 und 1 gibt es jeweils fünf dieser Objekte. Da kein Testlevel gewonnen wurde, und auch keine Punkte gesammelt wurden entspricht die Siegrate der Güte. Das Spiel CHAINREACTION kann in den Level 0 bis 3 Punkte erreichen ohne zu gewinnen. Jedoch wurde auf die Darstellung von Punkten und Siegrate in der y-Achse verzichtet, da nur Level 4

Objekte vernichtet wurden. Das Vernichten erzeugt zwei Punkte und führt in Level 4 direkt zum Sieg.
Die Variante ohne Fehlerterm verliert alle Testlevel. Ebenso wie die Variante mit Fehlerterm in den Leveln 0 und 1. In Level 4 gibt es vier initiale ϵ Werte, durch die die Siegrate gesteigert werden kann. Der initiale ϵ Wert von null ist keiner dieser vier Werte. Für alle Level gilt, dass die Siegrate der Variante mit einem initialen ϵ Wert mindestens genauso gut ist, wie die Variante ohne Fehlerterm. Die Anzahl an Spielfeldern ist bei jedem Level in der gleichen Größenordnung. Die Anzahl an den Objekten INDIREKTER STEIN, die vernichtet werden müssen, variiert. In Level 4 ist ein solches Objekt vorhanden, in den anderen Level sind es mindestens drei, bis maximal fünf.

6.4 Spiel SOKOBAN

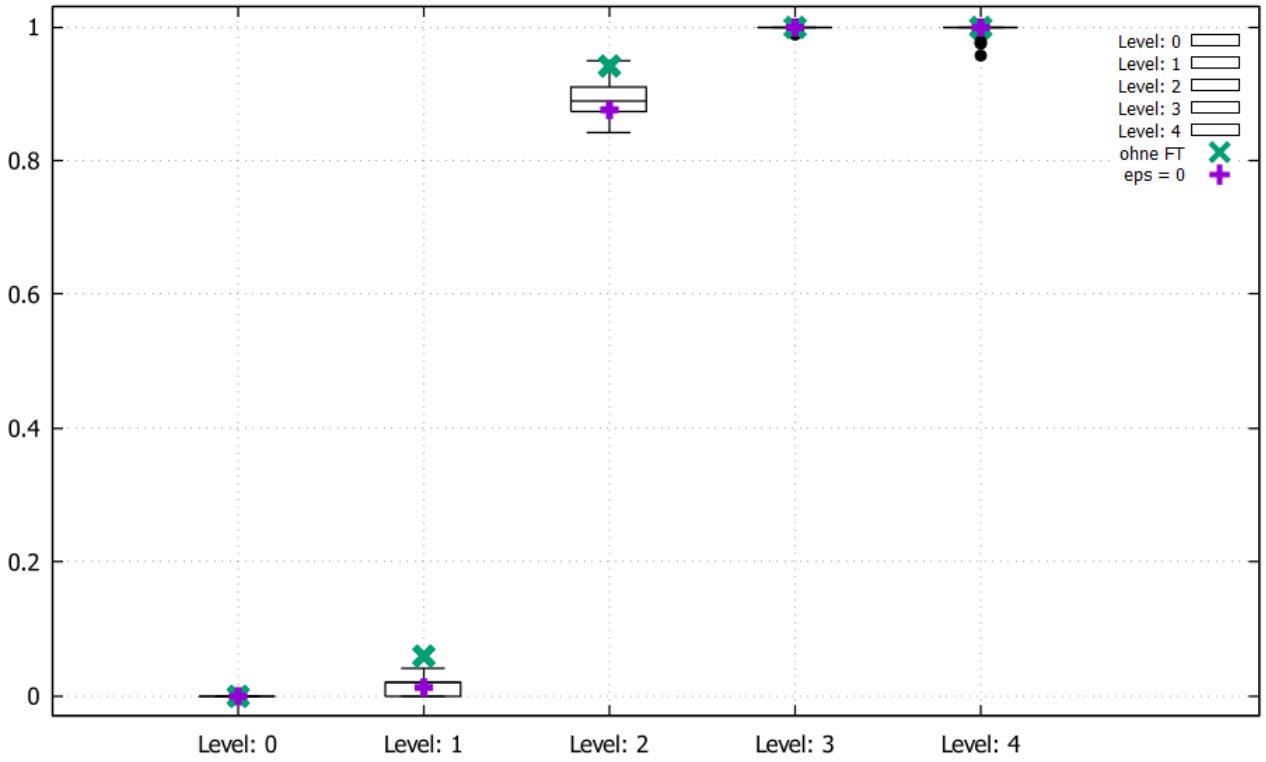

Abbildung 13: Levelweise Darstellung der Siegchancen für das Spiel SOKOBAN, im Vergleich beider Varianten. Zusätzliche Information für initialen ϵ Wert von 0.

SOKOBAN ist ein Spiel, in dem KISTEN in HÖHLEN geschoben werden müssen. Wenn alle KISTEN in eine beliebige HÖHLE geschoben wurden, ist das Spiel gewonnen. Es stehen 2000 Zeitschritte zur Verfügung, um das Spiel zu gewinnen. Jede KISTE, die in eine HÖHLE geschoben wurde, erhöht den Punktestand um eins.

Die Abbildung 13 zeigt die Ergebnisse des Spieles SOKOBAN. Da durch das Erreichen des maximalen Punktestandes ein Sieg herbeigeführt wird und somit Sieg und Punktestand voneinander abhängen, wurde nur die Siegrate betrachtet. Level 0, 3 und 4 haben auf den unterschiedlichen ϵ Werten und Varianten jeweils die selben Werte erzielt. In Level 0 gingen alle Spiele verloren. Die Testlevel der Level 3 und 4 wurden alle gewonnen. Die Siegrate für ein initiales ϵ von null

liegt bei den Leveln 1 und 2 im hinteren Mittelfeld. Die Siegrate für die Variante ohne Fehlerterm liegt in Level 1 etwas höher als die des besten initialen ϵ Wertes. In Level 2 liegt die Siegrate der Variante ohne Fehlerterm knapp unterhalb der Siegrate des besten initialen ϵ Wertes.
Die Siegrate steigt mit Höhe des Levels. Eine Abhängigkeit der Spielfeldgröße, die mit steigendem Level abnimmt von 64 über 61, 48, 19 zu 31, zu der steigenden Siegrate ist zu erkennen. Level 4 hat mit 31 Spielfeldern mehr Felder als Level 3, was die Ausreißer nach unten erklären kann. Zudem ist die Anzahl von Testlevel mit 50 relativ gering.
Auf den Spielfeldern sind nur zwei Arten von Objekten, KISTE und HÖHLE, zu finden. In den ersten vier Leveln sind jeweils insgesamt sechs Objekte vorhanden. Vier Objekte gibt es in Level 4.

6.5 Spiel COLOURESCAPE

COLOURESCAPE ist ein Spiel, das beim Betreten des Feldes, auf dem sich das Objekt ZIEL befindet, gewonnen wird. Der Sieg bringt einen Punkt. Um zu diesem ZIEL zu gelangen, müssen verschieden farbige KISTEN verschoben werden. Es gibt Felder, auf denen sich farbige FAHNEN befinden, die der Spielfigur einen farbigen Marker auferlegt. Farbige KISTEN lassen sich nur verschieben, wenn die Spielfigur den richtigen Marker erhalten hat.

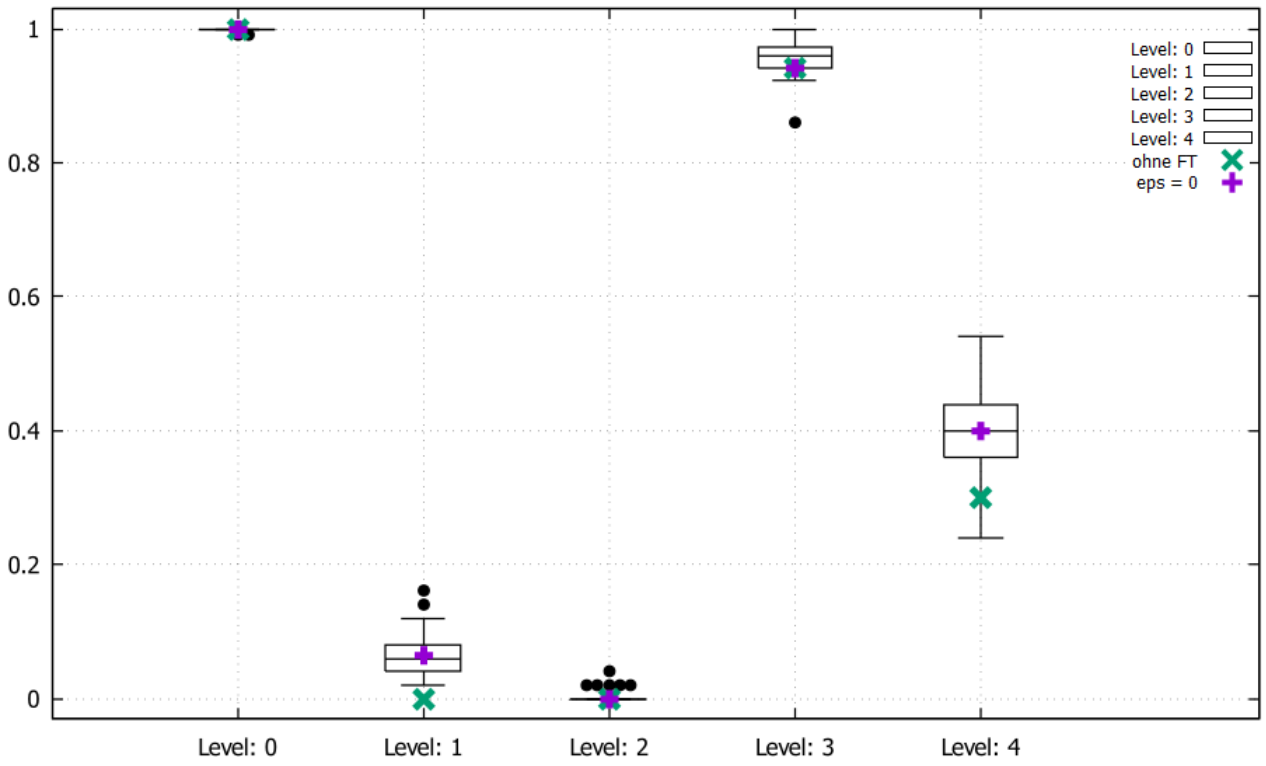

Abbildung 14: Levelweise Darstellung der Siegchancen für das Spiel COLOURESCAPE, im Vergleich beider Varianten. Zusätzliche Information für initialen ϵ Wert von 0.

Der Abbildung 14 ist zu entnehmen, dass die Variante ohne Fehlerterm schlechter abschneidet. In Level 1 liegt die Siegrate 0,02 Einheiten unterhalb des schlechtesten Wertes. Die Siegrate des initialen ϵ Wert von 0 liegt in der Nähe des Medians. Die Level 0 und 3 werden meistens gewonnen, Level 1 und 2 werden eher verloren. Das Level 4 hat einen Median bei 0,4. Dieses Level wird geringfügig weniger gewonnen als verloren.

Level	#Spielfelder	#Objekte	Farben	#Höhlen	#Fahnen
0	120	21	B,R,N	2	3
1	123	40	B,R,N,G	2	4
2	139	56	B,R,N,G	5	8
3	139	57	B,R, G	6	7
4	138	59	B,R,N,G	6	10

Tabelle 2: Daten zu dem Spiel COLOURESCAPE

In dem Spiel gibt es vier mögliche Farben an KISTEN und FAHNEN. Die Farben blau B, rot R neutral N und grün G werden verwendet. Der Tabelle 2 ist zu entnehmen, dass in den Level 0 und 3 jeweils nur drei der möglichen vier Farben verwendet werden. Diese Tatsache kann mit der hohen Siegrate in Verbindung gebracht werden. Die Testlevel von Level 0 werden häufiger gewonnen, da sowohl die Anzahl an Spielfeldern und die Anzahl an Objekten geringer ist.
In den Spielen mit vier Farben scheint die Positionierung der Objekte Einfluss auf die Siegrate zu nehmen, da die Anzahl an Spielfeldern ähnlich ist, auch die Verteilung der Objekte ist relativ gleich. Die Daten des Spieles lassen keine Schlussfolgerung zu, warum Testlevel des Level 4 eher gewonnen werden als die der Level 1 und 2. Im Spiel COLOURESCAPE erreichen mindestens 75% der Initialisierungen eine bessere oder gleich gute Siegrate wie die Variante ohne Fehlerterm.

6.6 Spiel BRAINMAN

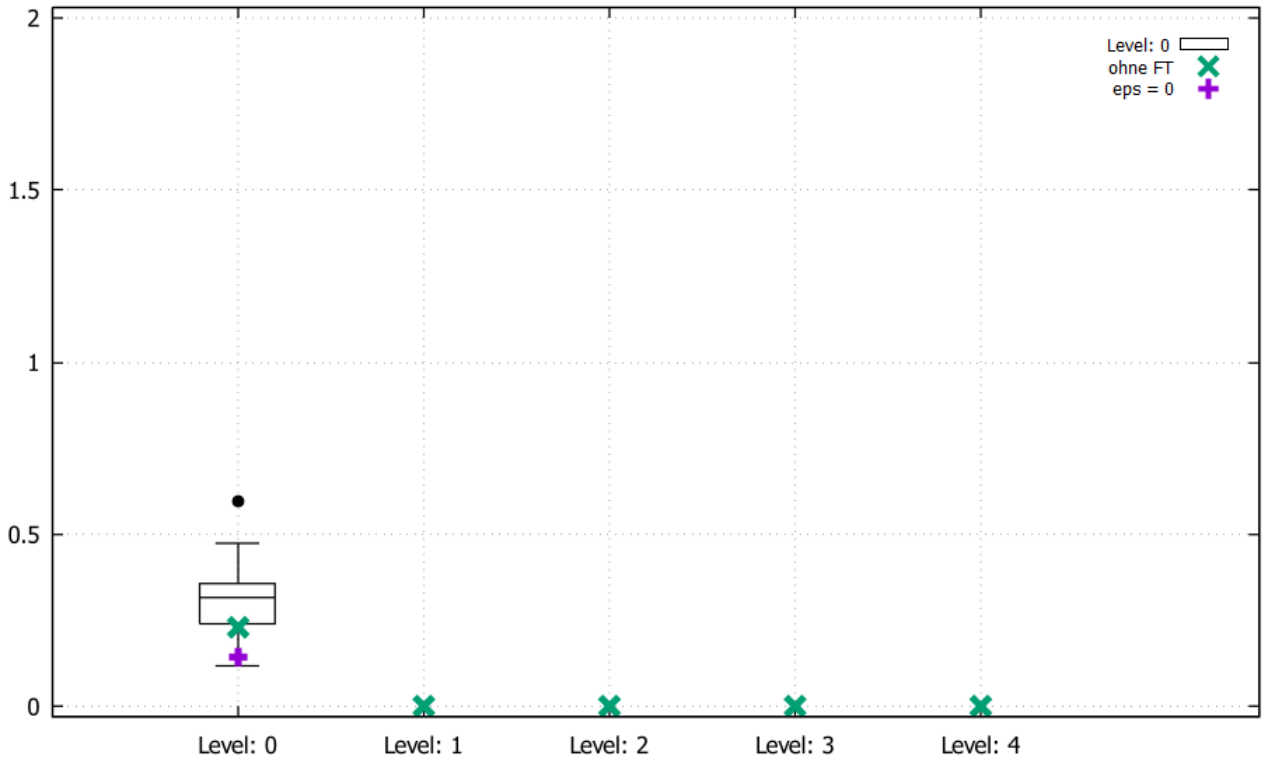

Abbildung 15: Levelweise Darstellung der Siegchancen für das Spiel BRAINMAN, im Vergleich beider Varianten. Zusätzliche Information für initialen ϵ Wert von 0.

BRAINMAN kann als mehrschichtiges Spiel betrachtet werden. Es gibt ein Objekt SCHLÜSSEL, welches durch dagegenlaufen verrutscht werden kann. Dieser SCHLÜSSEL muss auf ein Feld mit einem TOR gerutscht werden. Dadurch wird der Weg zu dem Bereich hinter dem TOR frei und man erhält 4 Punkte. Von dieser Aufgabe kann es hintereinander mehrere Bereiche geben, daher kann das Spiel als mehrschichtig bezeichnet werden. Wenn der letzte Bereich erreicht wurde, muss das Feld mit dem Objekt ZIEL betreten werden, wodurch das Spiel gewonnen wird und der Punktestand um 10 erhöht wird. Zusätzlich können sich in den einzelnen Bereichen weitere Objekte befinden mit denen interagiert werden kann. Zum Teil befinden sich STEINE oder DIAMANTEN in den Bereichen, die dazu genutzt werden können, um die SCHLÜSSEL entsprechend zu verrutschen. Das Einsammeln der DIAMANTEN erhöht den Punktestand entsprechend der Farbe $(blau + 1, grün + 2, orange + 5)$.

Für das Spiel BRAINMAN sind die Güte Werte der einzelnen Experimente in Abbildung 15 dargestellt. Es wurde die Darstellung mit Verwendung von Punkten gewählt.
Für die Level 1 bis 4 liegen nicht ausreichend viele Ergebnisse der Testlevel mit initialen ϵ Werten vor.
Die Güte der Variante ohne Fehlerterm ist in Level 0 unterhalb der unteren Quantillinie. Die Güte der Testlevel mit ϵ Wert von null ist 0,0129 Einheiten besser als das Minimum. Die anderen vier Level haben auf allen Testleveln der Variante ohne Fehlerterm eine Güte von null erzielt. Das bedeutet, dass keines dieser Testlevel Punkte erzielt hat oder gewonnen wurde. Somit kann die Vergleichsvariante gleich oder besser abschneiden.

6.7 Spiel MODALITY

Das Ziel des Spieles MODALITY ist es, ein Objekt KISTE auf ein Feld mit einem Objekt HÖHLE zu schieben. Die KISTE lässt sich feldweise verschieben. Das Spiel gilt als gewonnen, wenn sich die KISTE und die HÖHLE auf dem selben Feld befinden. Dafür wird ein Punkt vergeben. Somit entspricht der Punktestand dem Siegverhalten.

Die Level 0, 1, 2 und 4 haben, wie in Abbildung 16 zu sehen ist, in allen Testleveln eine Siegrate von eins erreicht. Level 3 hat ein Maximum bei 0,967 und ein Minimum bei 0,72. Die Siegrate der Version ohne Fehlerterm liegt bei 0,96, also knapp unterhalb des oberen Wisker. Mit 0,919 liegt die Siegrate der Testlevel für den ϵ Wert null knapp unterhalb der oberen Quantillinie. Das Level 3 ist mit 99 Spielfeldern größer als die übrigen Level, die zwischen 15 und 36 Spielfelder haben. Die Größe der Spiele kann eine Ursache dafür sein, dass Level 3 nicht so häufig gewonnen wird wie die anderen Level. Die Anzahl an Objekten liegt in jedem Level bei zwei. Die Komplexität des Spieles steigt daher mit der Anzahl der Spielfelder. Für das Spiel MODALITY erzielt die dynamische $\hat{f}$ Suche ähnliche Werte, wie die Suche ohne Fehlerterm.

6.8 Wahl des initialen ϵ Wertes

In den vorigen Abschnitten des Kapitels 5 wurden die Siegraten und Güten der getesteten Spiele vorgestellt. In diesem Kapitel wird auf die Siegrate pro initialem ϵ Wert eingegangen.

In Abbildung 17 sind die durchschnittlichen Siegraten pro initialem ϵ Wert für sechs Level zu sehen. Die Level 1. (BRAINMAN Level 0) und 2. (CATAPULTS Level 3) haben schwankende Siegraten. Das 1. Level hat eine maximale Siegrate bei einer Initialisierung mit 1,5 erreicht. Der initiale ϵ Wert von 200 hat in Level 2. zu einem Maximum geführt. In diesem beiden Leveln ist zu sehen, wie verschieden die Siegraten sind. In Level 1. erreicht 0 eine mittelmäßige Siegrate

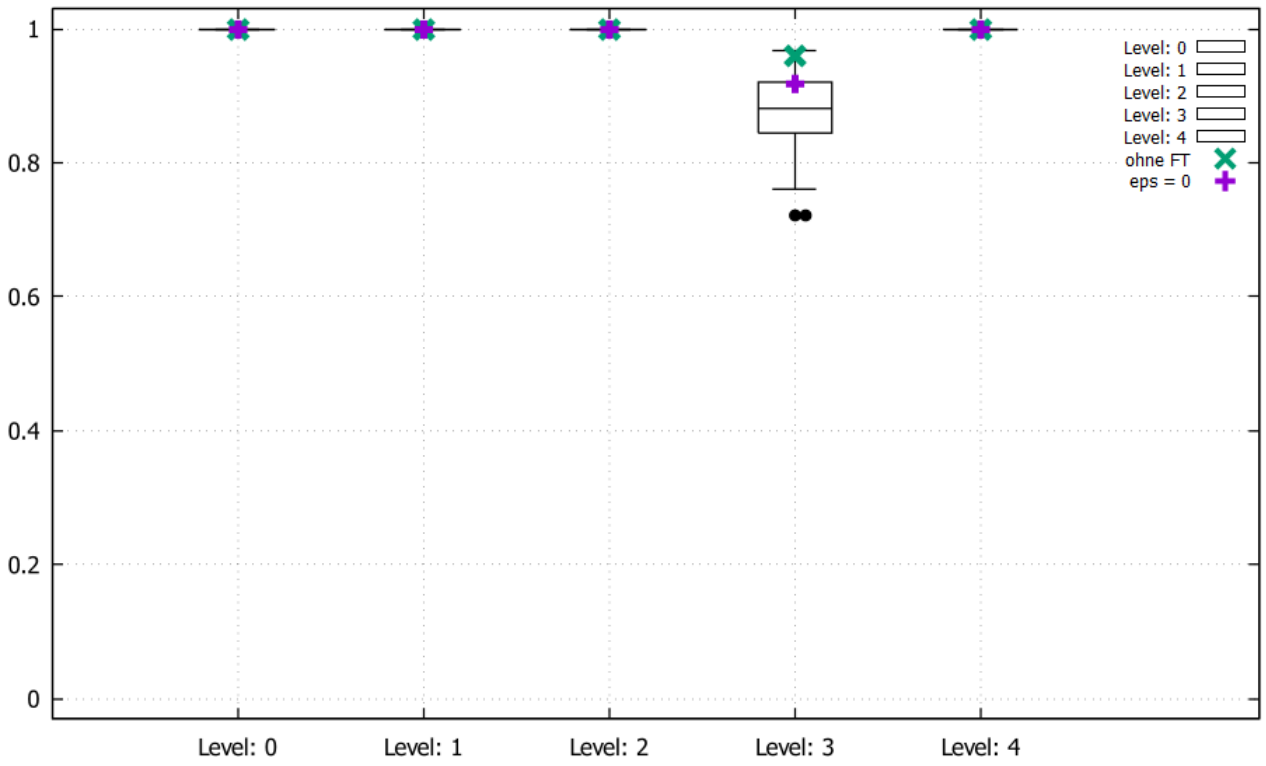

Abbildung 16: Levelweise Darstellung der Siegchancen für das Spiel MODALITY, im Vergleich beider Varianten. Zusätzliche Information für initialen ϵ Wert von 0.

und 2000 eine eher schlechte. Umgekehrtes Verhalten ist in Level 2. zu beobachten. Für die Werte 1,0 bis 2,0 erreicht 1. überwiegend hohe Siegraten. In Level 2. sind auf diesem Intervall zum Teil niedrige Siegraten zu finden.
Die Level 3. (SOKOBAN Level 1) und 4. (COLOURESCAPE Level 3) haben für die unterschiedlichen Initialisierungen jeweils ähnliche Ergebnisse erzielt. Es gibt einige Abweichungen in Level 3. nach oben. Hier werden 24 der 45 getesteten Initialisierungen mit einer besseren Siegrate getestet. Diese 24 initialen ϵ Werte sind relativ gleichmäßig verteilt. Auch in Level 4. sind die Siegraten nah beieinander und sowohl die schlechteren als auch die besseren Initialisierungen sind relativ durcheinander verteilt.
Vier Siegraten, die etwas höher sind als die der anderen Initialisierungen, sind in Level 5. (CHAINREACTION Level 4) zu finden. Diese Siegraten wurden für die initialen ϵ Werte von 0,3 1,9 650 und 2000 erreicht. In Level 6. (SOKOBAN Level 4) wurden wenige Initialisierungen etwas seltener gewonnen. Die schlechteren beziehungsweise besseren Siegraten sind relativ gleichmäßig verteilt.

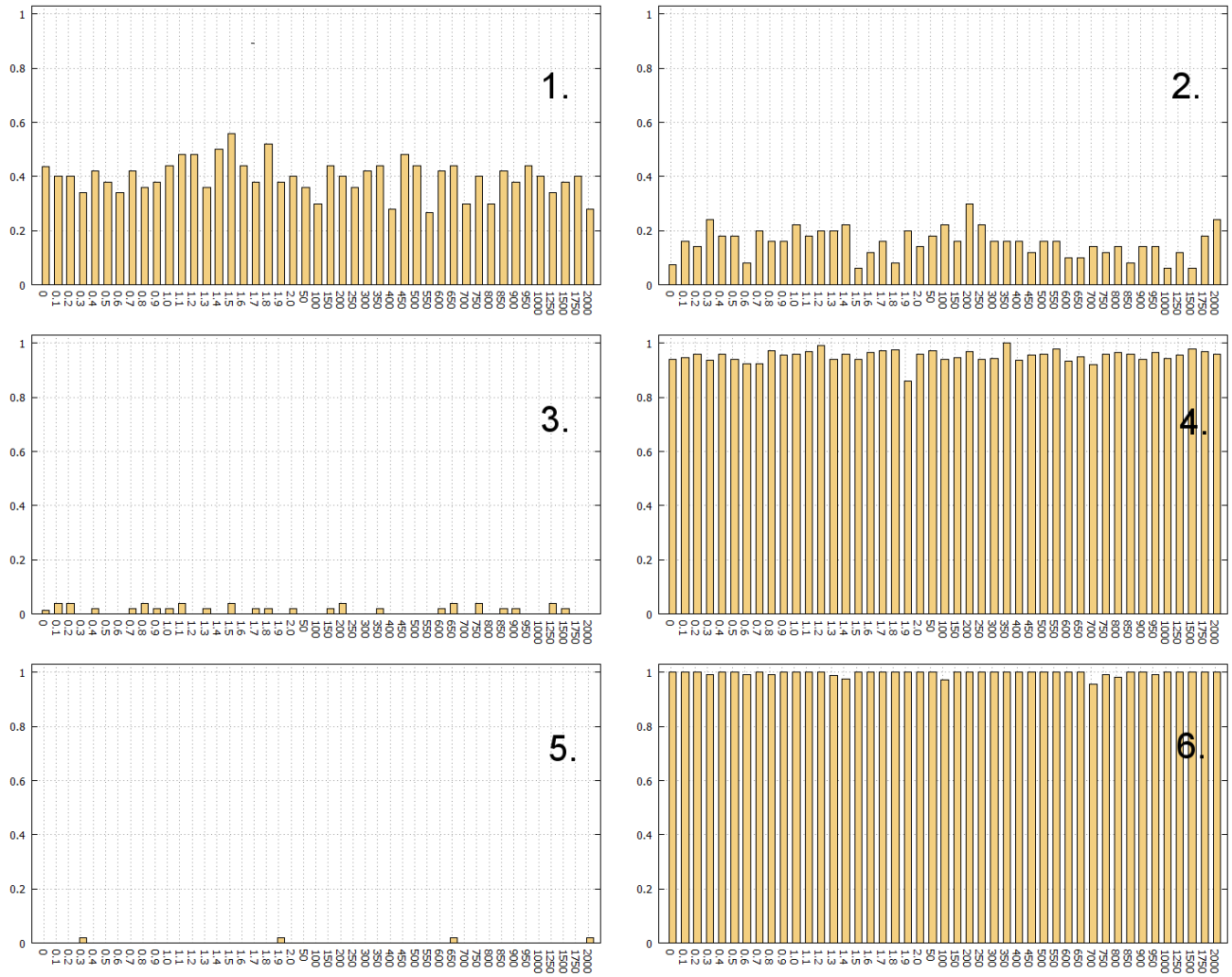

Abbildung 17: Durchschnittliche Siegrate für die getesteten initialen ϵ Werte auf sechs repräsentativen Leveln. Die Siegrate ist auf der y-Achse aufgetragen und die Initialisierungen können der x-Achse entnommen werden. Die Siegrate der jeweils 50 Testlevel wird als Balken dargestellt.

7 Fazit

In Kapitel 4 wurde die dynamische $\hat{f}$ Suche vorgestellt. Diese Suche bietet die Möglichkeit heuristische Ungenauigkeiten auszugleichen. Dieses Mittel wurde extrahiert und auf eine Problemstellung angewendet, für die die Heuristik die Eigenschaften zulässig und konsistent nicht erfüllen kann. Um die in Kapitel 1 gestellte Kernfrage

> Bewährt sich diese Methode des dynamischen $\hat{f}$ Algorithmus auf Heuristiken, die weder konsistent noch zulässig sind?

beantworten zu können wurde ein Experiment durchgeführt.
Das Experiment beinhaltet sieben Spiele des GVG-AI Wettbewerbes, welches in Kapitel 2.5 beschrieben ist. Jedes dieser Spiele hat fünf Level.
Innerhalb des Experimentes wurden zwei verschiedene Algorithmen verwendet. Der eine Algorithmus beinhaltet die Mechanik der dynamischen $\hat{f}$ Suche, der andere nutzt den Fehlerterm

nicht.
Um den Algorithmus mit Verwendung des Fehlerterms zu starten, war es notwendig einen initialen ϵ Wert zu wählen. Insgesamt wurden 45 verschiedene initiale ϵ Werte zwischen 0 und 2000 getestet.
Für den Algorithmus ohne Fehlerterm liegen die Ergebnisse aller Level der sieben Spiele vor. Um eine Aussage über die Wahl des initialen ϵ Wertes treffen zu können, war für den Algorithmus, der die Mechanik der dynamischen $\hat{f}$ Suche verwendet, ein 45-facher Aufwand nötig.
Das Testen aller Testlevel war zeitbedingt nicht möglich. Insgesamt sind drei Spiele unvollständig. Dem Spiel BRAINMAN fehlen vier Level. Jeweils zwei Level sind in den Spielen CATAPULTS und CHAINREACTION nicht vorhanden. Alle Testlevel dieser acht Level erreichten auf der Variante ohne Fehlerterm eine Siegrate von null. Somit hat die dynamische $\hat{f}$ Variante die Möglichkeit auf den Testleveln eine höhere Siegrate zu erzielen. In jedem Fall ist diese Variante mindestens gleich gut.
Somit bleiben von den insgesamt 35 Leveln die für das Experiment, das in Kapitel 5 behandelt wird, ausgewählt wurden, 27 Level mit vollstnndig vorhandenen Daten übrig.
Von 12 Leveln sind die Ergebnisse so, dass beide Varianten dieselbe Siegrate haben. Die jeweilige Siegrate dieser Level liegt entweder bei null oder eins. Das Bedeutet, dass der Agent mit allen verwendeten initialen ϵ Werten dieselbe Siegrate erspielt habt. In drei dieser Level sind wenige initiale ϵ Werte etwas schlechter als die anderen initialen ϵ Werte.
In einem der übrigen 15 Level ist die Siegrate für den Algorithmus ohne Fehlerterm 0,02 Einheiten besser als die größte Siegrate der getesteten initialen ϵ Werte. Dieses Level ist das Level 1 des Spieles SOKOBAN.
Die Testlevel mit einem initialen ϵ Wert von null, die auf den 15 Level angewendet wurden, erreichen in drei Leveln die beste Siegrate beziehungsweise die beste Güte. Die Level 1 und 2 des Spieles CATAPULTS und das Level 4 des Spieles BAIT sind diese drei Level. Auffällig ist in diesen Leveln, dass die Siegrate beziehungsweise Güte der Variante ohne Fehlerterm auf oder unterhalb der unteren Quantillinie liegt.
In fünf Leveln erreicht mindestens ein initialer ϵ Wert eine höhere Siegrate als die Testlevel der Vergleichsvariante.
Für neun Level erreichen mindestens 25% der initialen ϵ Werte eine höhere Siegrate als die Variante ohne Fehlerterm. In sieben der neun Level ist die Siegrate der Variante ohne Fehlerterm auf oder unterhalb der unteren Quantillinie, also sind 75% der initialen ϵ Werte besser.
Das Level 1 des Spieles COLOURESCAPE ist das einzige Spiel, in dem jeder getestete initiale ϵ Wert eine höhere Siegrate erzielt als die Variante ohne Fehlerterm. Die niedrigste Siegrate der initialen ϵ Werte liegt 0,02 Einheiten oberhalb der Siegrate der anderen Variante.
Für die Wahl des initialen ϵ Wertes konnte in Kapitel 6.8 kein Zusammenhang festgestellt werden. In einigen Leveln werden Siegraten für bestimmte Initialisierungen erreicht, die besser sind als die übrigen. Jedoch konnte kein Trend festgestellt werden, welche initialen ϵ Werte geeignet sind, um hohe Siegraten zu erzeugen. Die Ausreißer nach oben und unten der Siegrate scheinen relativ gleichmäßig über die getesteten Initialisierungen vorzukommen. Für einzelne Level lassen sich die initialen ϵ Werte finden, die zu einer hohen Siegrate führen. Jedoch für ein Spiel eine Initialisierung zu finden, die auf allen Leveln eine bessere Siegrate erspielt, als die Variante ohne Fehlerterm, ist schwierig. Die Wahl einer Initialisierung sollte daher für jede Problemstellung gesondert betrachtet werden.

Von diesen 35 Leveln werden 25 Level ähnlich gut gewonnen und in neun Leveln erzielen mindestens 25% der initialen ϵ Werte bessere Ergebnisse. Daher bewährt sich die Methode des dynamischen $\hat{f}$ Algorithmus auf Heuristiken, die weder konsistent noch zulässig sind. Es ist nicht auszuschließen, dass für das Level 1 des Spieles SOKOBAN ein geeigneter initialer ϵ Wert gefunden werden kann, der eine Siegrate erreicht, die mindestens der Siegrate der Variante ohne Fehlerterm entspricht.
Abhänig von der Wahl des initialen ϵ Wertes kann die herkömmliche Variante ohne Fehlerterm eine um bis zu 0,24 Einheiten höhere Siegrate erreichen.
Um bessere Ergebnisse zu erzielen ist die Wahl des initialen ϵ Wertes relevant. Es konnte kein Zusammenhang zwischen den Spielen festgestellt werden, aus denen erkenntlich ist welcher initiale ϵ Wert dazu führt, eine höhere Siegrate zu erlangen.
Die Ergebnisse zeigen, dass sich geeignete Startwerte finden lassen, um zu einer bessere Lösung zu gelangen.
Die Wahl des initialen ϵ Wertes von null hat in drei Leveln zwar bessere Ergebnisse erzielt als alle anderen getesteten Testlevel, erreicht aber auf sechs anderen Leveln eine Siegrate beziehungsweise Güte die schlechter ist als die der Variante ohne Fehlerterm. Für die drei Level in denen dieser Startwert am Besten abschneidet gibt es mindesten 75% aller getesteten initialen ϵ Werte, die besser als der Vergleichswert sind.

In Problemstellungen, die es zu lassen, vorab einen geeigneten initialen ϵ Wert zu bestimmen, erzielt die Methode des dynamischen $\hat{f}$ Algorithmus, der Heuristiken verwendet, die weder konsistent noch zulässig sind, bessere Ergebnisse.

8 Ausblick

Da sich die Verwendung der dynamischen $\hat{f}$ Suche auf Heuristiken, die weder zulässig noch konsistent sind, im Allgemeinen bessere Ergebnisse erzielte, ist es möglich diesen Ansatz weiter zu verfolgen. Der Austausch der Heuristik durch Rollouts ist dadurch denkbar, denn Rollouts erfüllen nicht die Anforderungen an die Heuristik. Zudem kann durch Rollouts das Problem, eine geeignete Heuristik zu finden, umgangen werden.
Es ist denkbar, weitere Experimente zu starten, um Zusammenhänge von der Wahl der initialen ϵ Werte und der erreichten Siegrate beziehungsweise Güte festzustellen. Hierfür müssten mehr ϵ Werte getestet werden. Die Abhängigkeit dieser Werte müsste in Bezug zu Spiel-spezifischen Parametern getroffen werden. Ob es eine geeignete Wahl eines initialen ϵ Wertes gibt, der auf verschiedenen Problemstellungen bessere Lösungen erzielt, muss untersucht werden.
Ob sich die Methodik der dynamischen $\hat{f}$ Suche auf den verschiedensten Problemen bewährt, kann nicht beurteilt werden. Hierfür muss abgeklärt werden, ob es für die jeweilige Problemstellung eine Initialisierung der ϵ Werte gibt, sodass der Algorithmus bessere Ergebnisse erzielt. Diese Fragen sind offen, da in einem der 35 getesteten Level die Variante ohne Fehlerterm eine höhere Siegrate erreicht hat. Von den 45 getesteten initialen Werte für ϵ konnte keine eine gleiche oder bessere Siegrate erspielen. Da nur 45 stichprobenartige Werte für die Initialisierung der ϵ Werte gewählt wurden, ist es denkbar, dass es eine Initialisierung gibt, die besser abschneidet.

Literatur

[1] Ethan Burns, Scott Kiesel, and Wheeler Ruml. Experimental real-time heuristic search results in a video game. In *Proceedings of the Sixth Annual Symposium on Combinatorial Search, SOCS 2013, Leavenworth, Washington, USA, July 11-13, 2013.*, 2013.

[2] Ariel Felner, Uzi Zahavi, Robert Holte, Jonathan Schaeffer, Nathan R. Sturtevant, and Zhifu Zhang. Inconsistent heuristics in theory and practice. *Artif. Intell.*, 175(9-10):1570–1603, 2011.

[3] Michael R. Genesereth, Nathaniel Love, and Barney Pell. General game playing: Overview of the AAAI competition. *AI Magazine*, 26(2):62–72, 2005.

[4] Michael R. Genesereth and Michael Thielscher. *General Game Playing*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2014.

[5] Tobias Joppen, Miriam Moneke, Nils Schröder, Christian Wirth, and Johannes Fürnkranz. Informed hybrid game tree search. Technical Report TUD-KE-2016-01, Knowledge Engineering Group, Technische Universität Darmstadt, December 2016. under review.

[6] Sven Koenig and Xiaoxun Sun. Comparing real-time and incremental heuristic search for real-time situated agents. *Autonomous Agents and Multi-Agent Systems*, 18(3):313–341, 2009.

[7] John Levine, Clare Bates Congdon, Marc Ebner, Graham Kendall, Simon M. Lucas, Risto Miikkulainen, Tom Schaul, and Tommy Thompson. General video game playing. In *Artificial and Computational Intelligence in Games*, pages 77–83. 2013.

[8] Diego Perez Liebana, Spyridon Samothrakis, Julian Togelius, Tom Schaul, Simon M. Lucas, Adrien Couëtoux, Jerry Lee, Chong-U Lim, and Tommy Thompson. The 2014 general video game playing competition. *IEEE Trans. Comput. Intellig. and AI in Games*, 8(3):229–243, 2016.

[9] Dylan O'Ceallaigh and Wheeler Ruml. Metareasoning in real-time heuristic search. In *Proceedings of the Eighth Annual Symposium on Combinatorial Search, SOCS 2015, 11-13 June 2015, Ein Gedi, the Dead Sea, Israel.*, pages 87–95, 2015.

[10] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 3 edition, 2014.

Anhang

Siegrate	Punktewert	#Schritte	Spiel	Level
1	5	178,24	Bait	0
1	7	633,96	Bait	1
0,6	4,54	1568,22	Bait	2
0,02	0,44	1992,5	Bait	3
0,02	0,22	1993,18	Bait	4
0	0,1	2000	Catapults	0
0	0	2000	Catapults	1
0	0	2000	Catapults	2
0,44	1,76	1525,66	Catapults	3
0	0	2000	Catapults	4
0	0	1500	Chainreaction	0
0	0	1500	Chainreaction	1
0	0	1500	Chainreaction	2
0	0	1500	Chainreaction	3
0	0	1500	Chainreaction	4
0	0	2000	Sokoban	0
0,06	0,18	1923,24	Sokoban	1
0,94	2,82	664,78	Sokoban	2
1	3	927,06	Sokoban	3
1	2	1121,4	Sokoban	4
1	1	223,74	Colourescape	0
0	0	1500	Colourescape	1
0	0	1500	Colourescape	2
0,94	0,94	545	Colourescape	3
0,3	0,3	1396,42	Colourescape	4
0,12	4,54	1798,58	Brainman	0
0	0	2000	Brainman	1
0	0	2000	Brainman	2
0	0	2000	Brainman	3
0	0	2000	Brainman	4
1	1	142,54	Modality	0
1	1	257,54	Modality	1
1	1	562,76	Modality	2
0,96	0,96	1654,16	Modality	3
1	1	305,52	Modality	4

Tabelle 3: Daten zu der Variante ohne Fehlerterm

avgSiegrate	minSiegrate	maxSiegrate	Spiel	Level
1	1	1	Bait	0
1	1	1	Bait	1
0,55036444	0,3585	0,7439	Bait	2
0,00133333	0	0,04	Bait	3
0,06349778	0	0,2174	Bait	4
0,02689556	0	0,1176	Catapults	0
0,03695111	0	0,1828	Catapults	1
0,01897111	0	0,1337	Catapults	2
0,39788	0,2667	0,56	Catapults	3
0,01912222	0	0,25	Catapults	4
0	0	0	Chainreaction	0
0	0	0	Chainreaction	1
0	0	0	Chainreaction	2
0	0	0	Chainreaction	3
0,00177778	0	0,02	Chainreaction	4
0	0	0	Sokoban	0
0,01449111	0	0,04	Sokoban	1
0,89190444	0,8421	0,95	Sokoban	2
0,99898222	0,9888	1	Sokoban	3
0,99628222	0,9565	1	Sokoban	4
0,99955556	0,99	1	Colourescape	0
0,06096889	0,02	0,16	Colourescape	1
0,00311111	0	0,04	Colourescape	2
0,95342444	0,86	1	Colourescape	3
0,40190667	0,24	0,54	Colourescape	4
0,15495333	0,06	0,3	Brainman	0
0	0	0	Brainman	1
0	0	0	Brainman	2
0	0	0	Brainman	3
0	0	0	Brainman	4
1	1	1	Modality	0
1	1	1	Modality	1
1	1	1	Modality	2
0,87558222	0,72	0,9667	Modality	3
1	1	1	Modality	4

Tabelle 4: Siegraten zu der Variante mit Fehlerterm

avgPunktewert	minPunktewert	maxPunktewert	Spiel	Level
5	5	5	Bait	0
7	7	7	Bait	1
3,9809525991	2,6603773585	5,3658536585	Bait	2
0,0302222222	0	0,9	Bait	3
0,7233015873	0,08	2,4285714286	Bait	4
0,1935522198	0	0,8235294118	Catapults	0
0,2222700119	0	1,1021505376	Catapults	1
0,1138253119	0	0,8021390374	Catapults	2
1,6564996713	1,16	2,28	Catapults	3
0,0764790765	0	1	Catapults	4
0	0	0	Chainreaction	0
0	0	0	Chainreaction	1
0	0	0	Chainreaction	2
0	0	0	Chainreaction	3
0,0035555556	0	0,04	Chainreaction	4
0	0	0	Sokoban	0
0,0434747475	0	0,12	Sokoban	1
2,6756963425	2,5263157895	2,85	Sokoban	2
2,9973057215	2,9662921348	3	Sokoban	3
1,9925635459	1,9130434783	2	Sokoban	4
0,9995555556	0,99	1	Colourescape	0
0,0609685292	0,02	0,16	Colourescape	1
0,0031111111	0	0,04	Colourescape	2
0,9534256276	0,86	1	Colourescape	3
0,4019048506	0,24	0,54	Colourescape	4
5,8271574074	2,24	11,48	Brainman	0
0	0	0	Brainman	1
0	0	0	Brainman	2
0	0	0	Brainman	3
0	0	0	Brainman	4
1	1	1	Modality	0
1	1	1	Modality	1
1	1	1	Modality	2
0,8755848814	0,72	0,9666666667	Modality	3
1	1	1	Modality	4

Tabelle 5: Punktewerte zu der Variante mit Fehlerterm

ϵ	Level 0	Level 1	Level 2	Level 3	Level 4
0	1	1	0,6783	0	0,2174
0.1	1	1	0,7439	0	0,06
0.2	1	1	0,6494	0	0,06
0.3	1	1	0,6154	0	0,12
0.4	1	1	0,6053	0	0,02
0.5	1	1	0,5952	0	0,08
0.6	1	1	0,625	0	0,06
0.7	1	1	0,6024	0	0,02
0.8	1	1	0,5301	0	0,02
0.9	1	1	0,5357	0	0,1
1.0	1	1	0,5294	0	0,1
1.1	1	1	0,5	0	0,06
1.2	1	1	0,4706	0	0,08
1.3	1	1	0,5217	0	0,12
1.4	1	1	0,46	0	0,06
1.5	1	1	0,5849	0	0,06
1.6	1	1	0,3585	0	0,04
1.7	1	1	0,46	0	0,08
1.8	1	1	0,54	0	0,02
1.9	1	1	0,6	0	0,02
2.0	1	1	0,4848	0,04	0
50	1	1	0,5	0	0,06
100	1	1	0,4746	0	0,08
150	1	1	0,5094	0	0,08
200	1	1	0,5	0	0,06
250	1	1	0,56	0	0,1
300	1	1	0,54	0	0,04
350	1	1	0,5	0	0,02
400	1	1	0,42	0	0,02
450	1	1	0,62	0	0,12
500	1	1	0,5714	0	0,02
550	1	1	0,6	0	0,02
600	1	1	0,5932	0	0,04
650	1	1	0,4462	0	0,02
700	1	1	0,64	0	0,08
750	1	1	0,5385	0	0,1
800	1	1	0,66	0	0,06
850	1	1	0,48	0	0,04
900	1	1	0,6429	0	0,1
950	1	1	0,5789	0,02	0,08
1000	1	1	0,6207	0	0,06
1250	1	1	0,62	0	0,04
1500	1	1	0,48	0	0,08
1750	1	1	0,44	0	0,06
2000	1	1	0,54	0	0,08

Tabelle 6: Durchschnittliche Siegraten des Spiels BAIT zu den initialen ϵ Werten

ϵ	**Spiel 1, 1**	**Spiel 1, 2**	**Spiel 1, 3**	**Spiel 2, 0**	**Spiel 2, 1**	**Spiel 2, 4**	**Spiel 5, 0**
0	0,1828	0,1337	0,4379	0	0	0	0,0729
0.1	0,02	0	0,4	0	0	0	0,16
0.2	0,06	0	0,4	0	0	0	0,14
0.3	0,02	0,02	0,34	0	0	0,02	0,24
0.4	0,04	0,02	0,42	0	0	0	0,18
0.5	0,02	0,04	0,38	0	0	0	0,18
0.6	0,04	0,04	0,34	0	0	0	0,08
0.7	0,04	0	0,42	0	0	0	0,2
0.8	0,04	0,04	0,36	0	0	0	0,16
0.9	0,1	0	0,38	0	0	0	0,16
1.0	0,08	0	0,44	0	0	0	0,22
1.1	0,06	0,02	0,48	0	0	0	0,18
1.2	0,06	0	0,48	0	0	0	0,2
1.3	0,04	0,02	0,36	0	0	0	0,2
1.4	0,04	0	0,5	0	0	0	0,22
1.5	0,04	0,1	0,56	0	0	0	0,06
1.6	0,06	0,06	0,44	0	0	0	0,12
1.7	0	0	0,38	0	0	0	0,16
1.8	0,04	0,04	0,52	0	0	0	0,08
1.9	0,02	0	0,38	0	0	0,02	0,2
2.0	0,02	0	0,4	0	0	0	0,14
50	0,02	0	0,36	0	0	0	0,18
100	0,06	0,02	0,3	0	0	0	0,22
150	0,02	0	0,44	0	0	0	0,16
200	0,08	0	0,4	0	0	0	0,3
250	0,04	0	0,36	0	0	0	0,22
300	0,02	0	0,42	0	0	0	0,16
350	0	0,06	0,44	0	0	0	0,16
400	0,02	0	0,28	0	0	0	0,16
450	0	0,02	0,48	0	0	0	0,12
500	0,02	0,04	0,44	0	0	0	0,16
550	0,02	0	0,2667	0	0	0	0,16
600	0,04	0	0,42	0	0	0	0,1
650	0,04	0,02	0,44	0	0	0,02	0,1
700	0	0,04	0,3	0	0	0	0,14
750	0,02	0	0,4	0	0	0	0,12
800	0,02	0	0,3	0	0	0	0,14
850	0	0,04	0,42	0	0	0	0,08
900	0	0	0,38	0	0	0	0,14
950	0,04	0	0,44	0	0	0	0,14
1000	0,02	0,04	0,4	0	0	0	0,06
1250	0,06	0	0,34	0	0	0	0,12
1500	0,04	0	0,38	0	0	0	0,06
1750	0,04	0,02	0,4	0	0	0	0,18
2000	0,02	0,02	0,28	0	0	0,02	0,24

Tabelle 7: Durchschnittliche Siegraten der Spiele CATAPULTS (1), CHAINREACTION (2) und BRAINMAN (5) zu den initialen ϵ Werten. In der Form: Spiel (1; 2; 5), Levelnummer.

ϵ	Level 0	Level 1	Level 2	Level 3	Level 4
0	0	0,0121	0,876	1	1
0.1	0	0,04	0,9292	1	1
0.2	0	0,04	0,8819	0,9917	1
0.3	0	0	0,881	1	0,9919
0.4	0	0,02	0,8649	1	1
0.5	0	0	0,9211	1	1
0.6	0	0	0,9084	1	0,9921
0.7	0	0,02	0,8462	1	1
0.8	0	0,04	0,93	1	0,9909
0.9	0	0,02	0,8819	1	1
1.0	0	0,02	0,87	1	1
1.1	0	0,04	0,8862	1	1
1.2	0	0	0,8594	0,992	1
1.3	0	0,02	0,9182	1	0,9896
1.4	0	0	0,8952	0,9919	0,9747
1.5	0	0,04	0,95	1	1
1.6	0	0	0,8819	0,9888	1
1.7	0	0,02	0,9247	1	1
1.8	0	0,02	0,8889	1	1
1.9	0	0	0,92	0,9898	1
2.0	0	0,02	0,91	1	1
50	0	0	0,8444	1	1
100	0	0	0,9	1	0,974
150	0	0,02	0,8816	1	1
200	0	0,04	0,87	1	1
250	0	0	0,92	1	1
300	0	0	0,9091	1	1
350	0	0,02	0,92	1	1
400	0	0	0,88	1	1
450	0	0	0,872	1	1
500	0	0	0,93	1	1
550	0	0	0,8455	1	1
600	0	0,02	0,869	1	1
650	0	0,04	0,8986	1	1
700	0	0	0,8739	1	0,9565
750	0	0,04	0,9057	1	0,9904
800	0	0	0,93	1	0,9808
850	0	0,02	0,8889	1	1
900	0	0,02	0,8421	1	1
950	0	0	0,8921	1	0,9918
1000	0	0	0,877	1	1
1250	0	0,04	0,9011	1	1
1500	0	0,02	0,8506	1	1
1750	0	0	0,9032	1	1
2000	0	0	0,9058	1	1

Tabelle 8: Durchschnittliche Siegraten des Spiels SOKOBAN zu den initialen ϵ Werten

ϵ	**Level 0**	**Level 1**	**Level 2**	**Level 3**	**Level 4**
0	1	0,0636	0	0,94	0,4
0.1	1	0,06	0	0,9483	0,34
0.2	1	0,1	0	0,96	0,42
0.3	1	0,06	0	0,936	0,4074
0.4	1	0,14	0	0,958	0,3774
0.5	1	0,04	0	0,94	0,46
0.6	1	0,1	0	0,9238	0,38
0.7	1	0,02	0	0,9238	0,5167
0.8	1	0,06	0	0,9709	0,36
0.9	0,99	0,02	0	0,9571	0,4
1.0	1	0,16	0	0,96	0,38
1.1	1	0,1	0,04	0,97	0,4
1.2	1	0,08	0	0,99	0,4
1.3	1	0,08	0	0,9417	0,5
1.4	1	0,1	0	0,96	0,36
1.5	1	0,12	0	0,94	0,4643
1.6	1	0,04	0,02	0,9661	0,44
1.7	1	0,06	0	0,9722	0,36
1.8	1	0,08	0,02	0,9748	0,54
1.9	1	0,04	0	0,86	0,42
2.0	1	0,02	0	0,9595	0,34
50	1	0,04	0	0,9735	0,44
100	1	0,04	0	0,94	0,4
150	1	0,08	0	0,9479	0,38
200	1	0,08	0	0,97	0,38
250	1	0,08	0	0,94	0,3
300	1	0,02	0	0,9439	0,4
350	1	0,06	0	1	0,5
400	1	0,04	0	0,9367	0,34
450	1	0,06	0	0,956	0,34
500	1	0,04	0,02	0,9587	0,32
550	1	0,04	0	0,98	0,36
600	1	0,08	0	0,9341	0,32
650	1	0,02	0	0,95	0,52
700	0,99	0,08	0	0,9223	0,32
750	1	0,08	0,02	0,958	0,48
800	1	0,06	0	0,9662	0,46
850	1	0,02	0	0,96	0,4
900	1	0,04	0	0,9407	0,36
950	1	0,06	0	0,965	0,4
1000	1	0,02	0	0,9435	0,38
1250	1	0,08	0,02	0,9554	0,54
1500	1	0,04	0	0,98	0,4
1750	1	0,02	0	0,97	0,24
2000	1	0,02	0	0,96	0,44

Tabelle 9: Durchschnittliche Siegraten des Spiels COLOURESCAPE zu den initialen ϵ Werten

ϵ	Level 0	Level 1	Level 2	Level 3	Level 4
0	1	1	1	0,9187	1
0.1	1	1	1	0,9091	1
0.2	1	1	1	0,9667	1
0.3	1	1	1	0,8077	1
0.4	1	1	1	0,9286	1
0.5	1	1	1	0,875	1
0.6	1	1	1	0,9091	1
0.7	1	1	1	0,9412	1
0.8	1	1	1	0,8772	1
0.9	1	1	1	0,8113	1
1.0	1	1	1	0,9038	1
1.1	1	1	1	0,8644	1
1.2	1	1	1	0,9412	1
1.3	1	1	1	0,78	1
1.4	1	1	1	0,76	1
1.5	1	1	1	0,72	1
1.6	1	1	1	0,8305	1
1.7	1	1	1	0,72	1
1.8	1	1	1	0,82	1
1.9	1	1	1	0,9057	1
2.0	1	1	1	0,84	1
50	1	1	1	0,9	1
100	1	1	1	0,8431	1
150	1	1	1	0,9	1
200	1	1	1	0,96	1
250	1	1	1	0,9615	1
300	1	1	1	0,88	1
350	1	1	1	0,86	1
400	1	1	1	0,92	1
450	1	1	1	0,8852	1
500	1	1	1	0,8431	1
550	1	1	1	0,92	1
600	1	1	1	0,86	1
650	1	1	1	0,88	1
700	1	1	1	0,94	1
750	1	1	1	0,8333	1
800	1	1	1	0,9	1
850	1	1	1	0,8448	1
900	1	1	1	0,86	1
950	1	1	1	0,9	1
1000	1	1	1	0,84	1
1250	1	1	1	0,94	1
1500	1	1	1	0,86	1
1750	1	1	1	0,92	1
2000	1	1	1	0,92	1

Tabelle 10: Durchschnittliche Siegraten des Spiels MODALITY zu den initialen ϵ Werten