

B A C H E L O R A R B E I T

Verbesserung der Analyse von  
Rubiks-Cube-Algorithmen:

Ein visueller Ansatz zum Vergleich von Reinforcement Learning  
und dem Kociemba Zwei-Phasen-Algorithmus

Vorgelegt an der TH Köln  
Campus Gummersbach  
im Studiengang  
Medieninformatik

ausgearbeitet von:  
SIMON MARKO WÖHLER  
(Matrikelnummer: 11132806)

**Erster Prüfer:** Prof. Dr. Wolfgang Konen  
**Zweiter Prüfer:** Prof. Dr. Christian Kohls

Gummersbach, im April

## Zusammenfassung

Diese Bachelorarbeit befasst sich mit der Verbesserung der Analyse von Lösungsalgorithmen für Rubik's Cube Puzzles im General Board Game (GBG) Framework. Das Framework ermöglicht das Training von künstlicher Intelligenz (KI) mittels Reinforcement Learning für verschiedene Spiele, weist jedoch Limitationen bei der Visualisierung und Vergleichbarkeit von Rubik's Cube Lösungen auf. Die Arbeit verfolgt drei zentrale Ziele:

1. Die Entwicklung und Integration einer interaktiven 3D-Visualisierung für eine intuitivere Darstellung
2. Die Implementation eines deterministischen Referenzalgorithmus
3. Die vergleichende Analyse zwischen KI-basierten und deterministischen Lösungsansätzen.

Nach einer umfassenden Evaluation verschiedener Visualisierungswerkzeuge wurde CubeTwister als geeignete Lösung identifiziert und teilweise in das GBG-Framework integriert. Die technischen Herausforderungen bei der vollständigen Integration führten zur Entwicklung eines alternativen Workflows, der die Stärken des GBG-Frameworks, des CubeTwister-Solvers und der modernen Twizzle-Visualisierung kombiniert.

Durch systematische Tests mit unterschiedlichen Scramble-Längen (5-13 Züge) wurden die Lösungsqualitäten der verschiedenen Ansätze verglichen. Die Ergebnisse zeigen, dass die KI-Agenten bei moderater Problemkomplexität (bis etwa 10 Züge) konkurrenzfähig sind und ähnlich effiziente Lösungen wie deterministische Algorithmen finden können. Bei steigender Komplexität weisen sie jedoch eine signifikant sinkende Lösungsrate auf, während die Lösungswege für die noch lösbaren Fälle weiterhin effizient bleiben. Besonders auffällig ist der kritische Schwellenwert von 10-11 Zügen beim  $3 \times 3 \times 3$  Würfel, ab dem die Erfolgsrate der KI-Agenten dramatisch abfällt. Die verbesserte Visualisierung bietet eine intuitivere Darstellung der Würfelzustände, während die systematische Protokollierung der Lösungssequenzen tiefere Einblicke in die qualitativ unterschiedlichen Lösungsstrategien ermöglicht. Dies bildet eine solide Grundlage für zukünftige Weiterentwicklungen im Bereich des Reinforcement Learning für komplexe Entscheidungsprobleme.

## Abstract

This bachelor thesis addresses the improvement of analysis methods for Rubik's Cube solving algorithms within the General Board Game (GBG) Framework. This framework enables artificial intelligence (AI) training through reinforcement learning for various games but has limitations regarding visualization and comparability of Rubik's Cube solutions. The work pursues three main objectives:

1. Developing and integrating an interactive 3D visualization for more intuitive representation
2. Implementing a deterministic reference algorithm
3. Conducting a comparative analysis between AI-based and deterministic solution approaches.

Following a comprehensive evaluation of various visualization tools, CubeTwister was identified as a suitable solution and partially integrated into the GBG Framework. The technical challenges encountered during the complete integration led to the development of an alternative workflow that combines the strengths of the GBG Framework, the CubeTwister solver, and the modern Twizzle visualization.

Through systematic tests with varying scramble lengths (5-13 moves), the solution qualities of the different approaches were compared. The results show that AI agents are competitive at moderate problem complexity (up to about 10 moves) and can find solutions as efficient as deterministic algorithms. With increasing complexity, however, they exhibit a significantly decreasing success rate, while maintaining solution efficiency for those cases they can still solve. The improved visualization offers a more intuitive representation of cube states, while the systematic recording of solution sequences enables deeper insights into the qualitatively different solution strategies. This provides a solid foundation for future developments in reinforcement learning for complex decision problems.

# Inhaltsverzeichnis

Abbildungsverzeichnis	5
<b>1 Einleitung</b>	<b>6</b>
1.1 Reinforcement Learning und Deep Learning in der KI-Forschung . . . . .	6
1.2 Die Bedeutung von Puzzle-Spielen für die KI-Forschung . . . . .	6
1.3 Das General Board Game (GBG) Framework . . . . .	7
1.3.1 Zusammenhang mit Reinforcement Learning . . . . .	7
1.3.2 TD-n-tuple-Learning und ergänzende Verfahren . . . . .	7
1.3.3 MCTS-Wrapper . . . . .	9
1.4 Problemstellung . . . . .	9
1.5 Zielsetzung der Arbeit . . . . .	9
1.6 Forschungsfragen . . . . .	10
1.7 Aufbau der Arbeit . . . . .	10
<b>2 Analyse potenzieller Visualisierungswerkzeuge</b>	<b>12</b>
2.1 Visuelle Darstellung im GBG-Framework . . . . .	12
2.2 Anforderungen an das Visualisierungstool . . . . .	13
2.3 Kategorisierung der untersuchten Werkzeuge . . . . .	14
2.3.1 Webbasierte JavaScript-Lösungen . . . . .	14
2.3.2 Plattformunabhängige Lösungen . . . . .	14
2.3.3 Java-basierte Werkzeuge . . . . .	14
2.4 Vergleichende Analyse der Visualisierungstools . . . . .	15
<b>3 Implementation des Visualisierungswerkzeugs</b>	<b>16</b>
3.1 Technische Herausforderungen bei der Integration von CubeTwister . . . . .	16
3.2 Extraktion und Integration der CubeTwister-Bibliothek . . . . .	16
3.3 Reverse-Engineering-Ansätze zur Identifikation der Integrationsschnittstelle	17
3.4 Implementationsansatz über den ScriptPlayer . . . . .	18
3.5 Ergebnisse der Implementation . . . . .	18
3.6 Vergleich der 2D- und 3D-Visualisierung . . . . .	19
3.7 Prozessablauf und Protokollierung der Spiele . . . . .	20
3.8 Alternative Lösung mit CubeTwister-Standalone und Twizzle . . . . .	21
<b>4 Testaufbau und Durchführung</b>	<b>24</b>
4.1 Testmethodik . . . . .	24
4.2 Ergebnisse . . . . .	25
4.2.1 Lösungsrate . . . . .	26

4.2.2	Lösungseffizienz . . . . .	27
4.2.3	Vergleichsanalyse der erfolgreich gelösten Fälle . . . . .	28
4.3	Analyse und Interpretation der Ergebnisse . . . . .	29
<b>5</b>	<b>Zusammenfassung und Ausblick</b>	<b>30</b>
5.1	Zusammenfassung der Arbeit . . . . .	30
5.2	Beantwortung der Forschungsfragen . . . . .	31
5.3	Kritische Bewertung der Ergebnisse . . . . .	32
5.4	Offene Probleme und zukünftige Forschungsrichtungen . . . . .	33
5.5	Fazit . . . . .	33
<b>6</b>	<b>Quellenverzeichnis</b>	<b>35</b>
	<b>Erklärung über die selbständige Abfassung der Arbeit</b>	<b>37</b>

# Abbildungsverzeichnis

1	2D-Darstellung eines Rubik's Cube im GBG-Framework mit abgewickelter Oberflächenansicht . . . . .	12
2	Die implementierte 3D-Darstellung mit CubeTwister-Integration im GBG-Framework mit interaktiver Würfelansicht . . . . .	19
3	Eigenständige Anwendung von CubeTwister mit Visualisierung einer Würfelsequenz . . . . .	21
4	Twizzle-Weboberfläche mit 3D-Darstellung eines Rubik's Cube und Bewegungssteuerung . . . . .	22
5	Lösungsrate für Rubik's Cube ( $3 \times 3 \times 3$ ) . . . . .	26
6	Durchschnittliche Lösungslänge für Pocket Cube ( $2 \times 2 \times 2$ ) . . . . .	27
7	Durchschnittliche Lösungslänge für Rubik's Cube ( $3 \times 3 \times 3$ ) . . . . .	27
8	Vergleich der Lösungslängen für die Schnittmenge erfolgreich gelöster Fälle ( $3 \times 3 \times 3$ ) . . . . .	28

# 1 Einleitung

In den letzten Jahrzehnten hat die Entwicklung künstlicher Intelligenz (KI) signifikante Fortschritte erzielt, so auch im Bereich des maschinellen Lernens und seiner Anwendung auf komplexe Entscheidungsprobleme. Eine vielversprechende Methode ist das Reinforcement Learning (RL), bei dem Agenten durch Interaktion mit ihrer Umgebung eigenständig Strategien entwickeln [SB18]. Dieses Paradigma hat in den vergangenen Jahren bemerkenswerte Erfolge erzielt, wie die Überlegenheit gegenüber menschlicher Fähigkeiten in Spielen wie Go, Schach und Shogi [McA+].

## 1.1 Reinforcement Learning und Deep Learning in der KI-Forschung

Reinforcement Learning basiert auf dem Prinzip des Lernens durch Belohnung und Bestrafung. Ein Agent interagiert mit einer Umgebung, führt Aktionen aus und erhält Feedback in Form von Belohnungen oder Strafen. Mit dieser Methode sollen langfristig optimale Strategien entwickelt werden [SB18]. In Kombination mit Deep Learning-Techniken, insbesondere tiefen neuronalen Netzen, steigert sich die Leistungsfähigkeit von RL-Algorithmen erheblich, was schließlich als Deep Reinforcement Learning (DRL) bezeichnet wird [McA+].

Wie Sutton und Barto in ihrem grundlegenden Werk “Reinforcement Learning: An Introduction” darlegen, eignet sich RL besonders gut für Probleme, bei denen eine langfristige Planung und strategische Entscheidungsfindung erforderlich sind – Eigenschaften, die in vielen Spielumgebungen häufig vorkommen [SB18]. Die Anwendung von RL auf Spiele hat sich als fruchtbares Forschungsfeld erwiesen, da Spiele eine klar definierte Umgebung mit expliziten Regeln und Zielen bieten. Das macht sie zu idealen Testumgebungen für die Entwicklung und Bewertung von KI-Algorithmen.

## 1.2 Die Bedeutung von Puzzle-Spielen für die KI-Forschung

Puzzle-Spiele wie der Rubik’s Cube stellen eine besondere Herausforderung für KI-Systeme dar. Der klassische  $3 \times 3 \times 3$  Rubik’s Cube verfügt über etwa  $4,3 \cdot 10^{19}$  verschiedene Zustände, von denen nur einer die Lösung ist. Im Gegensatz zu Spielen wie Go oder Schach ist der Rubik’s Cube ein Einzelspieler-Spiel. Eine zufällige Abfolge von Zügen wird, unabhängig von ihrer Länge, mit hoher Wahrscheinlichkeit nicht zum gelösten Zustand führen.

Diese Eigenschaften machen den Rubik’s Cube zu einer stark relevanten Testumgebung für Reinforcement Learning-Algorithmen. Wie McAleer et al. [McA+] betonen, stellen Umgebungen mit einer hohen Anzahl von Zuständen und nur wenigen Belohnungszuständen eine besondere Herausforderung für Deep Reinforcement Learning-Methoden dar. Algorithmen, die in der Lage sind, den Rubik’s Cube zu lösen, können wertvolle Er-

kenntnisse für weitere Anwendungsgebiete liefern – insbesondere dort, wo Belohnungen selten und Entscheidungsräume groß sind.

Ein bemerkenswerter Fortschritt im Bereich des maschinellen Lernens für den Rubik’s Cube wurde 2019 von Agostinelli et al. mit DeepCubeA erzielt. Dieser Ansatz kombiniert tiefes Reinforcement Learning mit klassischen Suchalgorithmen und kann nicht nur den Rubik’s Cube, sondern auch andere kombinatorische Puzzle wie das 15-Puzzle, 24-Puzzle und Sokoban lösen. Im Gegensatz zum in dieser Arbeit untersuchten GBG-Framework, das auf  $n$ -tuple-basiertem TD-Learning aufbaut, zeigt DeepCubeA einen alternativen Ansatz zur Lösung komplexer kombinatorischer Probleme [Ago+19].

Des Weiteren lassen sich die Zustände des Rubik’s Cube und seiner Varianten (wie der  $2 \times 2 \times 2$  Pocket Cube) mithilfe der Gruppentheorie, der mathematischen Disziplin zur Untersuchung von algebraischen Strukturen von Gruppen, beschreiben. Das wirft interessante und weitreichende Fragen zur Anwendbarkeit von maschinellen Lernmethoden auf komplexe symbolische Systeme auf [McA+].

### 1.3 Das General Board Game (GBG) Framework

Das General Board Game (GBG) Learning and Playing Framework wurde für Bildung und Forschung im Bereich der KI entwickelt [Kon23]. Es handelt sich um ein Open-Source-Framework, das die Anwendung von Algorithmen auf eine Vielzahl von Spielen ermöglicht [Wol25].<sup>1</sup>

#### 1.3.1 Zusammenhang mit Reinforcement Learning

Das GBG-Framework implementiert den zuvor beschriebenen Reinforcement-Learning-Ansatz. Im Gegensatz zu komplexen Deep-Learning-Architekturen, wie sie bei DeepCube [McA+], [Ago+19] zum Einsatz kommen, setzt das GBG-Framework auf einen alternativen Ansatz mit geringerem Rechenaufwand: *TD-n-tuple-Learning*.

#### 1.3.2 TD-n-tuple-Learning und ergänzende Verfahren

Das sogenannte *TD-n-tuple-Learning* kombiniert zwei bewährte Ansätze aus dem Bereich des maschinellen Lernens, um spielbasierte Probleme effizient zu lösen:

- **Temporal Difference Learning (TD-Learning):** Hierbei handelt es sich um eine Methode aus dem Reinforcement Learning, bei der ein Agent lernt, wie gut ein bestimmter Zustand ist – nicht durch direkte Belohnung, sondern durch den Vergleich aufeinanderfolgender Zustände. Der Agent passt seine Bewertung also immer dann leicht an, wenn sich seine Vorhersage über den erwarteten Belohnungswert

---

<sup>1</sup><https://github.com/wolfgangkonen/GBG>

als ungenau herausstellt. Über viele Iterationen hinweg entwickelt er dadurch eine Strategie, die langfristig zu besseren Entscheidungen führt.

- **N-tuple-Systeme:** Um die Bewertung von Spielzuständen effizient darzustellen, kommen sogenannte n-tuple-Systeme zum Einsatz. Dabei werden gezielt kleine, relevante Ausschnitte (n-Tupel) des Spielfeldes betrachtet. Jeder dieser Ausschnitte wird mit einem Gewicht versehen, das im Laufe des Lernprozesses angepasst wird. So entsteht eine kompakte, aber dennoch leistungsfähige Näherung an eine komplexe Bewertungsfunktion – ohne dass das gesamte Spielfeld explizit analysiert werden muss.

Um die Lernleistung weiter zu verbessern, wurde das Framework um zwei zusätzliche Verfahren ergänzt:

- **Temporale Kohärenz (TCL):** Diese Technik erweitert das TD-Learning um die Fähigkeit, für jedes Gewicht im Modell eine eigene Lernrate zu verwenden. Das bedeutet: Bereiche, in denen sich die Vorhersagen über die Zeit als stabil erweisen, können schneller angepasst werden. Gleichzeitig werden instabile oder unsichere Bereiche vorsichtiger behandelt. Dadurch wird das Lernen nicht nur robuster, sondern auch effizienter, weil der Agent schneller zu einer stabilen Bewertungsfunktion konvergiert.
- **Monte Carlo Tree Search (MCTS):** In der Evaluierungsphase lässt sich die Entscheidungsqualität weiter steigern, indem der trainierte Agent durch eine Monte Carlo Tree Search ergänzt wird. Dabei handelt es sich um eine Suchstrategie, die mithilfe zufälliger Simulationen verschiedene mögliche Spielzüge durchspielt und statistisch auswertet. So kann der Agent nicht nur auf sein erlerntes Wissen zurückgreifen, sondern auch gezielt nach dem aktuell vielversprechendsten Zug suchen. Das führt zu einer deutlich höheren Spielstärke – insbesondere in komplexen Spielsituationen.

Konen [Kon23] zeigt, wie das GBG-Framework diese TD-n-tuple-Learning-Ansätze erfolgreich auf verschiedene Spiele wie Othello und ConnectFour anwendet. Basierend auf den im vorherigen Abschnitt diskutierten Erfolgen von Reinforcement Learning bei komplexen Puzzles und der besonderen Herausforderung des Rubik's Cube erscheint die Anwendung und Anpassung dieser Methoden auf den Rubik's Cube und seine Varianten als vielversprechender Forschungsansatz.

Das Framework ermöglicht es Agenten, durch Selbstspiel zu lernen, ohne dass spezifisches Domänenwissen oder menschliche Anleitung erforderlich ist [Kon23]. Allerdings besitzt die Integration des Rubik's Cube in das GBG-Framework noch Schwachstellen, insbesondere bei der visuellen Darstellung und der Vergleichbarkeit mit etablierten Lösungsalgorithmen.

### 1.3.3 MCTS-Wrapper

Im GBG-Framework werden die auf TD-n-tuple-Learning basierenden KI-Agenten optional mit einem MCTS-Wrapper ergänzt. Dieser Wrapper fungiert als zusätzliche Entscheidungsschicht, die den bereits trainierten Agenten umgibt und dessen Bewertungen als Basis für eine gezielte Baumsuche nutzt. Anstatt direkt die vom Agenten vorgeschlagene Aktion auszuführen, führt der MCTS-Wrapper zunächst mehrere simulierte Spielverläufe durch, um die vielversprechendsten Zugfolgen zu identifizieren. Die Anzahl der MCTS-Iterationen ist dabei ein wichtiger Parameter, der die Suchmächtigkeit und damit die Spielstärke beeinflusst.

In der vorliegenden Arbeit bezieht sich der Begriff “KI” oder “KI-Agent” durchgängig auf die im GBG-Framework trainierten TD-n-tuple-basierten Agenten, erweitert durch den MCTS-Wrapper.

## 1.4 Problemstellung

Im GBG-Framework bestehen bezüglich Rubik’s Cube Puzzlen noch Limitationen bei der aktuellen Implementierung: Zum einen ist die visuelle Darstellung der Würfel im Framework auf eine 2D-Ansicht beschränkt, was für das menschliche Verständnis unintuitiv ist und gespielte Sequenzen schwer nachvollziehbar macht.

Zum anderen fehlt es im aktuellen GBG-Framework an einem Vergleichsmittel zur Messung der Leistung der implementierten KI-Lösungen. es ist unklar wie die KI-Agenten im Vergleich zu etablierten deterministischen Lösungsalgorithmen wie Kociembas Zweiphasenalgorithmus abschneiden.

Diese Lücken limitieren sowohl die pädagogische Wirksamkeit des Frameworks für Bildungszwecke als auch dessen Nützlichkeit für die wissenschaftliche Forschung im Bereich des Reinforcement Learning für komplexe Entscheidungsprobleme.

## 1.5 Zielsetzung der Arbeit

Die vorliegende Bachelorarbeit verfolgt drei Hauptziele:

1. Die Entwicklung und Integration einer interaktiven 3D-Visualisierung für Rubik’s Cubes und Pocket Cubes in das GBG-Framework. Diese Visualisierung soll eine intuitive Darstellung der Würfelzustände und der ausgeführten Operationen ermöglichen, um die Nachvollziehbarkeit der Lösungswege zu verbessern. Nach einer umfassenden Evaluation existierender Open-Source-Lösungen soll die am besten geeignete Visualisierungsbibliothek ausgewählt und an die spezifischen Anforderungen des GBG-Frameworks angepasst werden.
2. Die Implementation und Integration eines etablierten deterministischen Lösungsalgorithmus in das GBG-Framework. Hierzu soll der Kociemba-Algorithmus für den

$3 \times 3 \times 3$  Cube als wichtigster Bestandteil, sowie entsprechende Algorithmen für den  $2 \times 2 \times 2$  Pocket Cube angebunden werden. Diese sollen als Referenzlösungen und Benchmarks für die Bewertung der im Framework trainierten KI-Agenten dienen.

3. Die Durchführung einer systematischen vergleichenden Analyse zwischen den im GBG-Framework trainierten KI-Agenten und den implementierten deterministischen Lösungsalgorithmen. Als Bewertungskriterien sollen dabei die Lösungslänge (gemessen in der Anzahl der benötigten Züge), sowie die maximale Menge an Scrambling-Zügen (zufällige Verdrehungen), die gelöst werden können, gelten. Die Ergebnisse dieser Analyse sollen mithilfe der implementierten 3D-Visualisierung anschaulich dargestellt werden, um Stärken und Schwächen der verschiedenen Ansätze zu identifizieren.

Durch die Realisierung dieser Ziele soll ein wesentlicher Beitrag zur Verbesserung des GBG-Frameworks geleistet und gleichzeitig tiefere Einblicke in die Leistungsfähigkeit von Reinforcement-Learning-Methoden für komplexe kombinatorische Optimierungsprobleme gewonnen werden.

## 1.6 Forschungsfragen

Die vorliegende Arbeit orientiert sich an folgenden zentralen Forschungsfragen:

1. Wie kann eine interaktive 3D-Visualisierung für Rubik's Cubes effektiv in das GBG-Framework integriert werden, um die Nachvollziehbarkeit der Lösungswege zu verbessern?
2. Inwiefern unterscheiden sich die im GBG-Framework trainierten KI-Agenten hinsichtlich ihrer Lösungsqualität von etablierten deterministischen Algorithmen für den Rubik's Cube und den Pocket Cube?
3. Welche spezifischen Stärken und Schwächen zeigen die verschiedenen Lösungsansätze?
4. Welchen Beitrag leistet die 3D-Visualisierung zum besseren Verständnis der von KI-Agenten generierten Lösungsstrategien?

Die Beantwortung dieser Forschungsfragen soll nicht nur einen direkten Beitrag zur Verbesserung des GBG-Frameworks leisten, sondern auch zeigen, wie man RL-Methoden praxisnah bewerten und verbessern kann.

## 1.7 Aufbau der Arbeit

Im Anschluss an diese Einleitung gliedert sich die Arbeit wie folgt:

In Kapitel 2 werden die Anforderungen an ein Visualisierungswerkzeug für den Rubik's Cube definiert und verschiedene potenzielle Tools anhand dieser Kriterien systematisch analysiert und verglichen. Das Kapitel schließt mit der Auswahl von CubeTwister als geeignetstem Werkzeug für die Integration in das GBG-Framework.

Kapitel 3 beschreibt den technischen Prozess der Integration von CubeTwister in das GBG-Framework. Es werden die aufgetretenen Herausforderungen, der gewählte Implementationsansatz über den ScriptPlayer sowie die schließlich realisierte Lösung mit einer Kombination aus GBG-Framework, CubeTwister-Standalone und Twizzle dargestellt.

In Kapitel 4 folgt eine detaillierte Beschreibung der Testmethodik und der durchgeführten Vergleichstests zwischen den KI-basierten und deterministischen Lösungsansätzen. Die Ergebnisse werden hinsichtlich Lösungsrate und Lösungseffizienz analysiert und interpretiert.

Die Arbeit schließt mit einer Zusammenfassung der gewonnenen Erkenntnisse und einem Ausblick auf mögliche weiterführende Forschungsansätze und Verbesserungen.

## 2 Analyse potenzieller Visualisierungswerkzeuge

### 2.1 Visuelle Darstellung im GBG-Framework

Das GBG-Framework bietet für alle implementierten Spiele eine grafische Benutzeroberfläche, die es ermöglicht, die Interaktionen der KI-Agenten mit dem Spielumfeld zu visualisieren und nachzuvollziehen. Im Fall des Rubik's Cube wurde ursprünglich eine einfache 2D-Darstellung implementiert, die zwar funktional ausreichend, aber für das menschliche Verständnis wenig intuitiv ist.

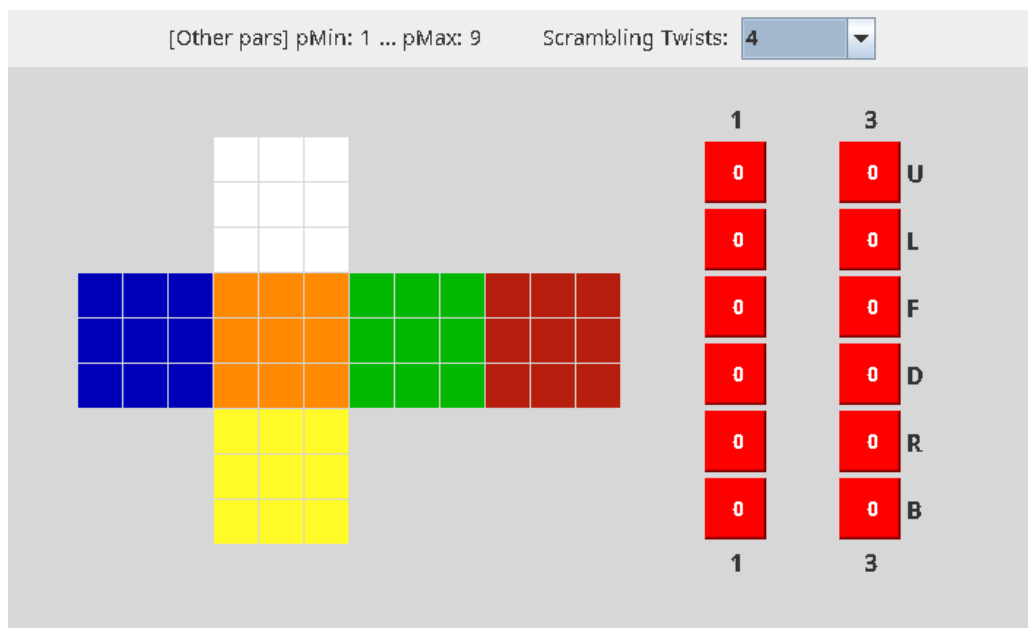


Abbildung 1: 2D-Darstellung eines Rubik's Cube im GBG-Framework mit abgewickelter Oberflächenansicht

Wie in Abbildung 1 zu sehen ist, verwendet die ursprüngliche Implementierung eine abgewinkelte 2D-Darstellung des Würfels. Die sechs Würfelflächen werden als zweidimensionales Netz angeordnet, wobei die UP-Fläche oben, die DOWN-Fläche unten und die vier Seitenflächen (LEFT, FRONT, RIGHT, BACK) horizontal in der Mitte angeordnet sind. Jede Fläche wird durch ein  $3 \times 3$  (bzw.  $2 \times 2$  für den Pocket Cube) Raster von farbigen Quadraten dargestellt, die den aktuellen Zustand des Würfels repräsentieren.

Diese Darstellung hat mehrere Nachteile:

- **Fehlende räumliche Intuition:** Die Abwicklung des dreidimensionalen Würfels in eine flache 2D-Ansicht erschwert es, die räumlichen Beziehungen zwischen den Würfelflächen zu erfassen.
- **Schwierige Nachvollziehbarkeit von Rotationen:** Wenn eine Würfelfläche gedreht wird, ist die resultierende Änderung in der 2D-Ansicht oft schwer nachzuvollziehen, da der räumliche Kontext fehlt.

- **Geringe Anschaulichkeit für Nicht-Experten:** Für Personen ohne Erfahrung mit Rubik's Cube ist die Abwicklung besonders schwer zu interpretieren, da sie ein mentales Zusammenfallen der Flächen erfordert.

Diese Limitationen der visuellen Darstellung erschweren sowohl die Nachvollziehbarkeit der von KI-Agenten gespielten Lösungswege als auch die didaktische Nutzung des Frameworks. Eine intuitivere, dreidimensionale Darstellung würde das Verständnis der komplexen Operationen am Rubik's Cube erheblich verbessern.

Der Versuch einer Implementation einer 3D-Visualisierung, wie sie in den folgenden Kapiteln beschrieben wird, zielt daher darauf ab, diese Limitationen zu überwinden und eine anschaulichere, interaktivere Darstellung des Rubik's Cube im GBG-Framework zu ermöglichen.

## 2.2 Anforderungen an das Visualisierungstool

Für die Entwicklung einer 3D-Visualisierung von Rubik's Cubes und Pocket Cubes im Rahmen des GBG-Frameworks wurde zunächst eine umfassende Recherche zu geeigneten Visualisierungstools durchgeführt. Ziel war es, ein Werkzeug zu identifizieren, das sich nahtlos in das bestehende Java-Framework integrieren lässt und gleichzeitig eine intuitive, ansprechende Darstellung der Würfel ermöglicht.

Die Auswahlkriterien für geeignete Werkzeuge wurden nach Priorität wie folgt festgelegt:

- **Lizenzierung:** Open-Source-Software mit freier Nutzung war Voraussetzung.
- **Native Java-Kompatibilität:** Das GBG-Framework ist vollständig in Java implementiert und besitzt keine API. Die Integration eines externen, nicht-Java-basierten Tools würde daher zunächst die Entwicklung einer API-Schnittstelle erfordern, was den Arbeitsaufwand erheblich erhöhen würde. Eine native Java-Implementierung ist daher ein wichtiges Auswahlkriterium.
- **Integrierter Solver:** Da ein Ziel dieser Arbeit der Vergleich zwischen KI-basierten und deterministischen Lösungsalgorithmen ist, wurde ein Visualisierungstool mit integriertem Solver bevorzugt. Ein Solver ist ein Algorithmus, der für einen gegebenen (verdrehten) Zustand des Würfels eine Sequenz von Zügen berechnet, die zur Lösung führt. Ohne einen integrierten Solver müsste eine separate Implementierung eines deterministischen Algorithmus erfolgen, was zusätzlichen Entwicklungsaufwand bedeuten würde.
- **Unterstützung der GBG Würfeltypen:** Darstellung des klassischen  $3 \times 3 \times 3$  Rubik's Cube sowie des  $2 \times 2 \times 2$  Pocket Cube.
- **Visuelle Qualität:** Die Darstellung sollte intuitiv, modern und ansprechend sein.

## 2.3 Kategorisierung der untersuchten Werkzeuge

Die recherchierten Tools lassen sich in drei Hauptkategorien einteilen:

### 2.3.1 Webbasierte JavaScript-Lösungen

Diese Tools setzen auf Webtechnologien wie JavaScript und WebGL:

- **AnimCubeJS** [Ani]: Leichtgewichtiger Simulator für verschiedene Würfeltypen.
- **Vasu-gondaliya's rubiks-cube** [Gon24]: Unterstützt Drehungen, Rotationen und Scramble-Generierung.
- **Freecube** [Wan24]: Kompakte WebGL-Lösung mit visuellen Schwächen.
- **Elm-Cubik** [Kuz24]: Realisiert in der funktionalen Sprache Elm.
- **cubing.js** [Twi; Cubb]: Umfassende Bibliothek mit Solver und aktiver Entwicklung.

Viele der Tools bieten hohe visuelle Qualität, besonders cubing.js [Twi; Cubb] erfüllt nahezu alle Anforderungen des Projekts. Allerdings würde die Integration dieser Tools eine Webschnittstelle benötigen, die erst entwickelt werden müsste, da das GBG-Framework keine API bereitstellt.

### 2.3.2 Plattformunabhängige Lösungen

Diese Tools sind plattformunabhängig, allerdings nicht direkt mit Java integrierbar.

- **Hyperspeedcube** [Far25]: Sehr mächtiges Rust-basiertes Tool mit Unterstützung für 3D- und 4D-Würfel. Favorit unter Speedcubern.
- **Cubik** [Gao24]: C++-basierte Visualisierung, beschränkt auf  $3 \times 3 \times 3$  und mit reduzierter visueller Qualität.

Beide Lösungen wären nur über aufwändige Schnittstellen durch language bridging nutzbar, wo die Laufzeitverhalten beider Werkzeuge miteinander kartiert werden, um die Benutzung gemeinsamer Ressourcen zu ermöglichen. Dennoch fällt Hyperspeedcube [Far25] als hoch modernes, vielversprechendes Werkzeug auf.

### 2.3.3 Java-basierte Werkzeuge

Tools mit nativer Java-Implementierung.

- **CubeTwister** [Cuba]: Umfangreiche Software mit Solver auf Basis des Kociemba-Algorithmus.

- **Cubez** [Zha24]: Entwickelt mit LWJGL library, beschränkt auf 3x3x3.
- **CuboRubik** [Ara23]: JavaFX-basierte Visualisierung beschränkt auf 3x3x3.

Durch die gemeinsame technologische Basis ist eine direkte Integration möglich. Besonders CubeTwister [Cuba] sticht heraus, da es alle gegebenen Anforderungen erfüllt.

## 2.4 Vergleichende Analyse der Visualisierungstools

Um eine fundierte Entscheidung zu treffen, wurden die recherchierten Tools anhand der definierten Auswahlkriterien systematisch verglichen. Tabelle 1 stellt die Ergebnisse dieser Analyse dar.

Tabelle 1: Vergleich der untersuchten Visualisierungstools

<b>Tool</b>	<b>Native Java</b>	<b>Integrierter Solver</b>	<b>Pocket &amp; Rubiks Support</b>	<b>Visuelle Qualität</b>
AnimCubeJS	Nein	Nein	Ja	Mittel
Vasu's rubiks-cube	Nein	Nein	Nein	Gut
Freecube	Nein	Nein	Nein	Niedrig
Elm-Cubik	Nein	Nein	Nein	Mittel
cubing.js/Twizzle	Nein	Ja	Ja	Sehr gut
Hyperspeedcube	Nein	Ja	Ja	Sehr gut
Cubik	Nein	Nein	Nein	Niedrig
CubeTwister	Ja	Ja	Ja	Gut
Cubez	Ja	Nein	Nein	Mittel
CuboRubik	Ja	Nein	Nein	Mittel

Wie aus der Tabelle ersichtlich wird, ist CubeTwister das eindeutig attraktivste Werkzeug, da es alle gestellten Anforderungen erfüllt: native Java-Kompatibilität, einen integrierten Solver und Unterstützung für sowohl 2×2×2 als auch 3×3×3 Würfel. Obwohl einige andere Tools, insbesondere Hyperspeedcube und cubing.js, in Bezug auf visuelle Qualität und Funktionalität überlegen sind, würde ihre Integration aufgrund der fehlenden nativen Java-Unterstützung einen erheblichen Mehraufwand bedeuten.

Es ist jedoch anzumerken, dass bei dieser ersten Evaluationsphase nicht alle Herausforderungen erkannt wurden, die mit der Verwendung von CubeTwister verbunden sind. Obwohl das Tool alle ursprünglich gestellten Anforderungen erfüllt, wurden potenzielle Schwierigkeiten bezüglich der Dokumentation und des Quellcode-Zugriffs zunächst unterschätzt. Diese Aspekte erwiesen sich später als signifikante Herausforderungen im Integrationsprozess, wie im folgenden Kapitel ausführlicher dargestellt wird.

## 3 Implementation des Visualisierungswerkzeugs

Nachdem CubeTwister als das am besten geeignete Visualisierungswerkzeug identifiziert wurde (vgl. Kapitel 2), folgte die technische Umsetzung der Integration in das GBG-Framework. In diesem Kapitel werden die technischen Herausforderungen, der Implementationsansatz sowie die schließlich realisierte Lösung beschrieben.

### 3.1 Technische Herausforderungen bei der Integration von CubeTwister

Bei der Integration von CubeTwister in das GBG-Framework traten mehrere technische Herausforderungen auf, die den Implementationsprozess erheblich beeinflussten:

- **Veraltete Distributionsform:** Der letzte Release von CubeTwister stammt aus dem Jahr 2012 und wird in Form einer JNLP-Datei (Java Network Launch Protocol) bereitgestellt. JNLP ist ein mittlerweile abgekündigtes Format, das ursprünglich für Java Web Start entwickelt wurde zur Verbreitung und Ausführung von Java-Anwendungen über das Internet, ohne einen Browser zum Ablauf zu benötigen [Jav].
- **Fehlende Dokumentation:** Es liegt keine umfassende Dokumentation zu CubeTwister vor, was die Identifikation und Nutzung relevanter Klassen und Methoden erschwert.
- **Kein Quellcode-Zugriff:** Zwar existiert ein öffentliches Repository zu CubeTwister allerdings lässt sich dieses nicht kompilieren, hat ein fehlerhaft aufgesetztes Build-System und unterscheidet sich in seiner Struktur stark von der letzten Release Version.

### 3.2 Extraktion und Integration der CubeTwister-Bibliothek

Trotz der genannten Herausforderungen wurde der Versuch, CubeTwister in das GBG-Framework zu integrieren, gestartet:

1. **Extraktion der JAR-Datei:** Durch Untersuchung der JNLP-Datei, die im Wesentlichen ein XML-Dokument ist, konnte der Pfad zur eigentlichen JAR-Datei (Java Archive) von CubeTwister identifiziert werden. Diese JAR-Datei enthält die kompilierten Klassen der Anwendung.
2. **Integration als Bibliothek:** Da das GBG-Framework kein Build-System verwendet, wurde die extrahierte JAR-Datei direkt als Bibliothek in das Projekt eingebunden. Dies entspricht dem etablierten Vorgehen innerhalb des Frameworks.

3. **Analyse der Klassenstruktur:** Ohne direkten Zugriff auf den Quellcode und bei fehlender Dokumentation musste die Struktur der Anwendung durch Analyse der dekompierten Klassen rekonstruiert werden. Dies erwies sich als aufwändig und fehleranfällig.

Durch das Fehlen eines funktionalen Repositories oder einer Dokumentation war kein systematisches Debugging der CubeTwister-Klassen möglich, um deren Struktur, Hierarchien und Funktionalitäten vollständig zu verstehen. Dies erschwerte die Integration erheblich, da die Interaktion zwischen dem GBG-Framework und CubeTwister weitgehend durch empirische Versuche und begrenzte Reverse-Engineering-Ansätze entwickelt werden musste.

### 3.3 Reverse-Engineering-Ansätze zur Identifikation der Integrationsschnittstelle

Der Reverse-Engineering-Prozess zur Analyse der CubeTwister-Bibliothek umfasste mehrere Strategien:

- **Dekompilierung der JAR-Datei:** Mithilfe der in der IDE integrierten Dekompilierungswerkzeuge wurden die kompilierten Java-Klassen in lesbaren Quellcode zurückverwandelt. Dies ermöglichte einen grundlegenden Einblick in die Struktur und Funktionsweise der Anwendung.
- **Analyse der Klassenbeziehungen:** Durch systematische Untersuchung der Abhängigkeiten und Hierarchien zwischen den Klassen wurden zentrale Komponenten der Software identifiziert.
- **Identifikation von Kernkomponenten:** Besondere Aufmerksamkeit galt den Klassen, die für die Würfeldarstellung und -manipulation zuständig sind, insbesondere den Geometrie-Klassen und den Steuerungsklassen.

Bei der Analyse wurden fundamentale Unterschiede zwischen der Würfelrepräsentation im GBG-Framework und in CubeTwister festgestellt:

- **GBG-Framework:** Verwendet eine abstrakte, zustandsbasierte Darstellung des Würfels, die primär auf der logischen Struktur und den Zustandsübergängen basiert. Der Fokus liegt auf der effizienten Darstellung für KI-Algorithmen.
- **CubeTwister:** Nutzt eine komplexe 3D-Geometriedarstellung mit visuellen Komponenten, die auf Rendering-Primitiven und Transformationsmatrizen aufbaut. Der Fokus liegt auf der visuellen Darstellung und Benutzerinteraktion.

Diese grundlegend verschiedenen Implementationsansätze machten eine direkte Verbindung der beiden Würfelrepräsentationen unmöglich. Eine vielversprechendere Alternative wurde in der Verwendung des ScriptPlayers als Brücke zwischen den beiden Systemen identifiziert.

### 3.4 Implementationsansatz über den ScriptPlayer

Nach Analyse der verfügbaren Funktionalitäten wurde der ScriptPlayer von CubeTwister als zentraler Integrationsansatz identifiziert. Der ScriptPlayer bietet folgende relevante Funktionen:

- **Sequenz-Wiedergabe:** Möglichkeit, Sequenzen von Würfelbewegungen in verschiedenen Notationen zu übergeben und abzuspielen.
- **Zeitstrahl-Navigation:** Ein interaktiver Zeitstrahl erlaubt die Navigation durch die Bewegungssequenzen.
- **Schrittweise Bewegungen:** Bedienelemente ermöglichen es, Bewegungen einzeln und schrittweise zu betrachten.
- **Interaktive 3D-Ansicht:** Der dargestellte Würfel kann mit der Maus gedreht werden, um verschiedene Perspektiven zu ermöglichen.

Der konzeptionelle Ansatz war es, die 3D-Geometrie-Klassen von CubeTwister in Verbindung mit dem ScriptPlayer zu nutzen, ohne eine direkte funktionale Verbindung zum CubeTwister-Cube herzustellen. Dies erschien als pragmatischer Kompromiss, da der ScriptPlayer primär auf vordefinierten Bewegungssequenzen operiert, die vom GBG-Framework geliefert werden könnten, ohne eine bidirektionale Synchronisation der Würfelzustände zu erfordern.

Das konzeptionelle Ziel der Implementation war es, die bestehende 2D-Darstellung des GBG-Frameworks durch die 3D-Visualisierung des CubeTwister ScriptPlayers zu ersetzen. Bei durchgeführten Spielen im GBG-Framework sollten die Scramble-Sequenzen (zufällige Verdrehungen des Würfels) und die Spielsequenzen der KI-Agenten an den ScriptPlayer zur 3D-Darstellung übergeben werden.

### 3.5 Ergebnisse der Implementation

Aufgrund der zuvor beschriebenen technischen Herausforderungen konnte das Integrationsziel im zeitlichen Rahmen dieser Arbeit nur teilweise erreicht werden:

- **Erfolgreiche Komponenten:**
  - Einbindung eines interaktiven 3D-Würfels im GBG-Anwendungsfenster

- Integration des Zeitstrahls zur Navigation
  - Implementierung der Bewegungssteuerungselemente
  - Erfassung und Protokollierung von Scramble- und Spielsequenzen im GBG-Framework
- **Nicht realisierte Komponenten:**
    - Übertragung der Sequenzen vom GBG-Framework zum ScriptPlayer
    - Abspielbarkeit der Sequenzen innerhalb des GBG-Framework

Die protokollierten Sequenzen werden in einer Logdatei gespeichert, um dennoch eine nachträgliche Analyse zu ermöglichen. Diese dient als Grundlage für eine alternative Lösungsstrategie, die im Abschnitt 3.8 beschrieben wird.

### 3.6 Vergleich der 2D- und 3D-Visualisierung

Der Übergang von der ursprünglichen 2D-Darstellung zur 3D-Visualisierung stellt eine signifikante Verbesserung in der Nachvollziehbarkeit und intuitiven Verständlichkeit dar:

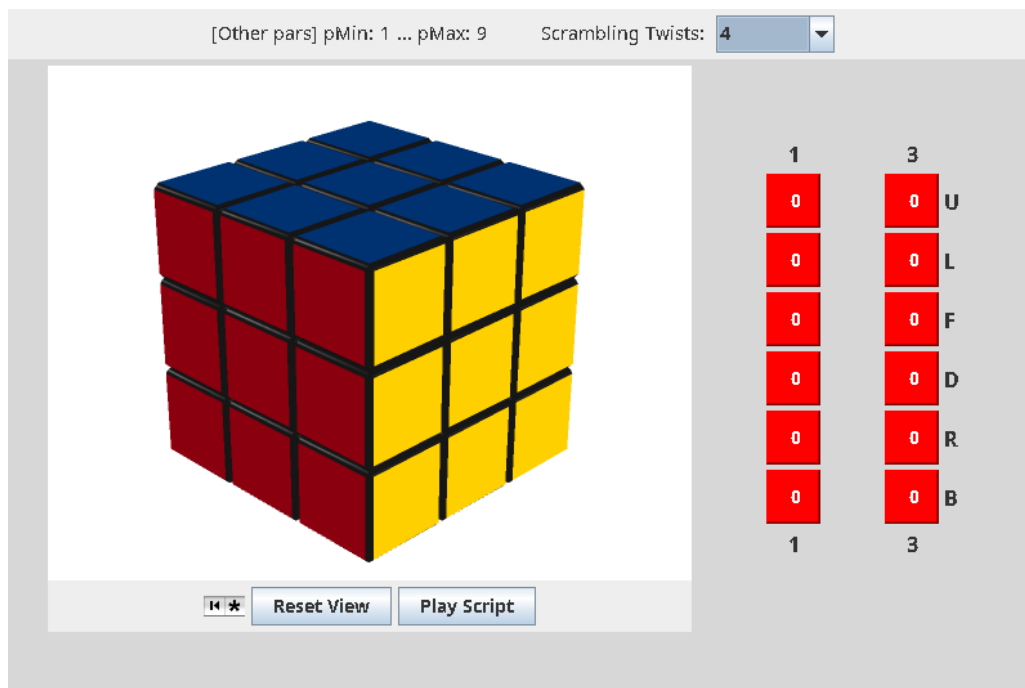


Abbildung 2: Die implementierte 3D-Darstellung mit CubeTwister-Integration im GBG-Framework mit interaktiver Würfelansicht

Der Vergleich zwischen der ursprünglichen 2D-Darstellung (siehe Abbildung 1 in Kapitel 2.1) und der neu implementierten 3D-Visualisierung (Abbildung 2)<sup>2</sup> verdeutlicht den qualitativen Unterschied zwischen den Visualisierungsansätzen:

<sup>2</sup>Video zur Veranschaulichung: <https://th-koeln.sciebo.de/s/iVlla5PMCWbRYre>

- **Räumliche Dimension:** Während die 2D-Darstellung lediglich eine abgewinkelte, flache Repräsentation des Würfels bietet, erlaubt die 3D-Visualisierung eine realitätsnahe, räumliche Darstellung. Diese erhöht die Intuitivität erheblich, da sie der tatsächlichen Erfahrung mit einem physischen Rubik's Cube entspricht.
- **Interaktivität:** Die 3D-Darstellung ermöglicht es, den Würfel aus verschiedenen Perspektiven zu betrachten, indem der Nutzer ihn frei rotieren kann.
- **Navigationsmöglichkeiten:** Der implementierte Zeitstrahl (der bei Einfügen eines Skripts erscheint) erlaubt eine gezielte Navigation durch die Bewegungssequenzen, was eine detaillierte Analyse der Lösungswege ermöglicht.

Trotz der technischen Einschränkungen bei der vollständigen Integration stellt die erreichte 3D-Visualisierung einen substantiellen Fortschritt gegenüber der ursprünglichen 2D-Darstellung dar und bietet eine deutlich verbesserte Basis für die visuelle Analyse von Lösungsstrategien.

### 3.7 Prozessablauf und Protokollierung der Spiele

Aufgrund der Schwierigkeiten bei der direkten Integration von CubeTwister in das GBG-Framework wurde eine systematische Protokollierung der Spielabläufe implementiert, um eine nachträgliche Analyse zu ermöglichen:

#### 1. Spielablauf im GBG-Framework:

- Laden eines trainierten KI-Agenten für den jeweiligen Würfeltyp (Pocket Cube oder Rubik's Cube)
- Konfiguration des MCTS-Wrappers mit 100 Iterationen
- Festlegung der gewünschten Scramble-Länge (zwischen 5 und 13 Zügen)
- Initiierung des Spiels durch Drücken der "Play"-Taste

#### 2. Automatisierte Protokollierung:

- Generierung einer zufälligen Scramble-Sequenz durch das Framework
- Versuch des KI-Agenten, eine Lösung zu finden
- Automatische Speicherung der folgenden Daten in einer Log-Datei:
  - Die angewandte Scramble-Sequenz
  - Die vom KI-Agenten gefundene Lösungssequenz
  - Erfolgsindikator (gelöst/nicht gelöst)
  - Länge der Lösungssequenz

Diese Protokollierung bildete die Grundlage für die im folgenden Abschnitt beschriebene alternative Lösung, bei der statt einer direkten Integration externe Visualisierungswerkzeuge zur Analyse eingesetzt wurden.

### 3.8 Alternative Lösung mit CubeTwister-Standalone und Twizzle

Die Erfahrungen mit der Integration der CubeTwister-Bibliothek führten zu einer wichtigen Erkenntnis: Da CubeTwister bereits als JAR-Paket extrahiert worden war, konnte es auch direkt als eigenständige Anwendung ausgeführt werden, ohne eine komplexe Integration in das GBG-Framework zu erfordern. Diese Einsicht führte zur Entwicklung einer pragmatischen Alternative.

Anstatt weiter zu versuchen, die komplexe Integration der CubeTwister-Bibliothek in das GBG-Framework zu bewerkstelligen, wurde eine Workflow-basierte Lösung entwickelt, die die Stärken beider Systeme nutzt, ohne sie direkt zu koppeln:

- **GBG-Framework als Basis:** Das Framework generiert und protokolliert weiterhin die Scramble-Sequenzen sowie die Lösungssequenzen der KI-Agenten.
- **CubeTwister-Standalone als primäres Vergleichsmittel:** Die vom GBG-Framework protokollierten Sequenzen werden in die eigenständige Version von CubeTwister übertragen, um dort mit dem integrierten Kociemba-Solver Referenzlösungen zu generieren und zu visualisieren.

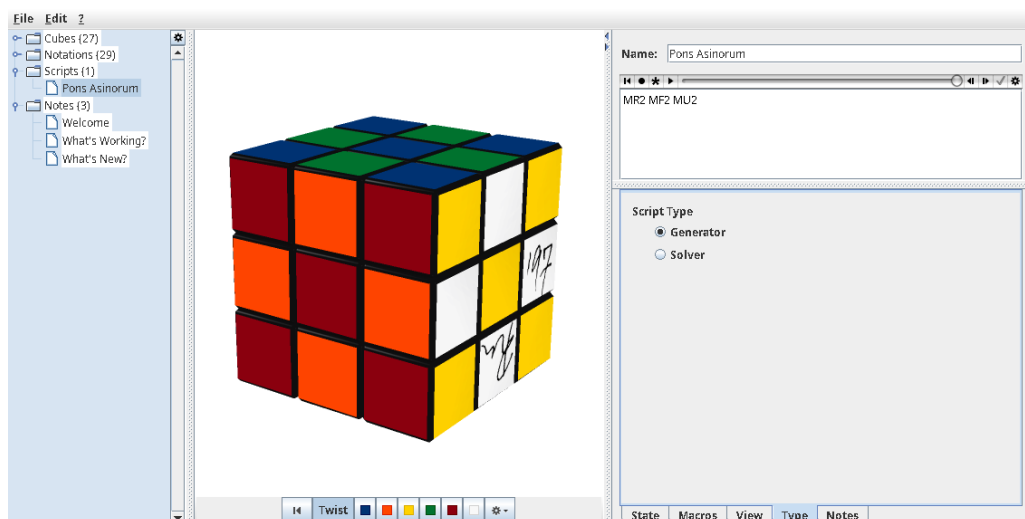


Abbildung 3: Eigenständige Anwendung von CubeTwister mit Visualisierung einer Würfelsequenz

Die Verwendung von CubeTwister als eigenständige Anwendung bot mehrere Vorteile gegenüber der Bibliotheksintegration:

- **Vollständige Funktionalität:** Alle Funktionen, einschließlich des Kociemba-Solvers, stehen ohne Einschränkungen zur Verfügung.
- **Stabile Ausführung:** Keine Kompatibilitätsprobleme zwischen Framework und CubeTwister.
- **Flexibilität:** Möglichkeit, verschiedene Aspekte der Lösungen separat zu analysieren.

Da mit dieser Entscheidung die Notwendigkeit entfiel, CubeTwister aufgrund seiner Java-Kompatibilität als einziges Visualisierungswerkzeug zu verwenden, wurde auch eine im Kapitel 2.3 identifizierte leistungsfähige Alternative einbezogen:

- **Twizzle als zusätzliches Vergleichsmittel und Visualisierungstool:** Twizzle, basierend auf der modernen cubing.js-Bibliothek [Cubb; Twi], wurde als ergänzendes Werkzeug in den Workflow integriert. Es bietet eine modernere und benutzerfreundlichere Oberfläche, eine ansprechendere Visualisierung und einen eigenen Solver als zusätzliches Vergleichsmittel.

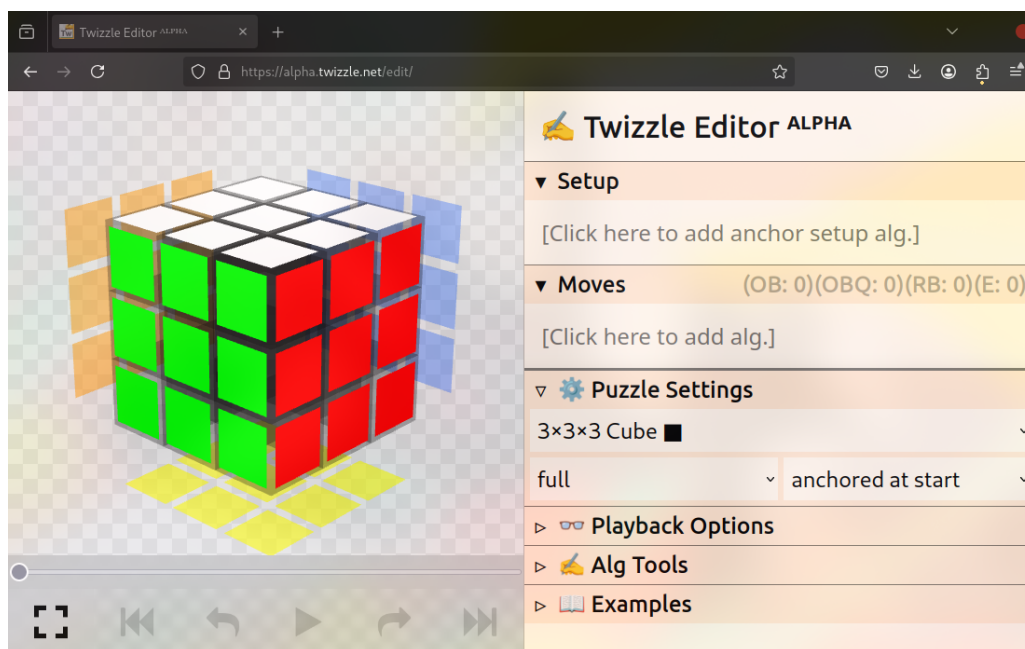


Abbildung 4: Twizzle-Weboberfläche mit 3D-Darstellung eines Rubik's Cube und Bewegungssteuerung

Der vollständige Workflow umfasst folgende Schritte:

1. Durchführung und Protokollierung der Tests im GBG-Framework
2. Extraktion der Scramble-Sequenzen aus den Protokolldateien

3. Eingabe der identischen Scramble-Sequenzen in CubeTwister-Standalone und Twizzle
4. Generierung von Referenzlösungen durch die integrierten deterministischen Solver
5. Tabellarische Erfassung und Vergleich der Lösungslängen
6. Detaillierte visuelle Analyse ausgewählter Fälle zur Identifikation von Mustern und Besonderheiten

Für die abschließende Analyse und Visualisierung werden sämtliche Lösungen - die KI-Lösung, die Kociemba-Lösung aus CubeTwister und die Twizzle-Lösung - tabellarisch aufgezeichnet und ausgewertet. Hierbei können auffällige Muster oder Unterschiede identifiziert und zur Veranschaulichung in Twizzle oder CubeTwister visualisiert werden.

Diese Kombination aus drei komplementären Werkzeugen - dem GBG-Framework für KI-basierte Lösungen, CubeTwister-Standalone für den Kociemba-Algorithmus und Twizzle für moderne Visualisierung und zusätzliche Validierung - ermöglichte trotz der gescheiterten direkten Integration einen fundierten Vergleich zwischen den verschiedenen Lösungsansätzen.

Diese vergleichende Analyse bildet die Grundlage für die Beantwortung der in Kapitel 1 formulierten Forschungsfragen, insbesondere zur Frage, inwiefern sich die im GBG-Framework trainierten KI-Agenten hinsichtlich ihrer Lösungsqualität von etablierten deterministischen Algorithmen unterscheiden.

Der detaillierte Testaufbau, die Bewertungskriterien und die Ergebnisse der Vergleichsanalyse werden im folgenden Kapitel beschrieben.

## 4 Testaufbau und Durchführung

Der praktische Vergleich zwischen den KI-basierten und deterministischen Lösungsansätzen stellt einen zentralen Aspekt dieser Arbeit dar. In diesem Kapitel wird der Aufbau und die Durchführung der vergleichenden Analyse beschrieben.

### 4.1 Testmethodik

Um die in Kapitel 1 formulierten Forschungsfragen zu beantworten, wurde ein systematischer Testaufbau entwickelt. Dieser zielt darauf ab, zwei zentrale Aspekte der Lösungsqualität zu erfassen:

1. **Lösungsrate:** Der Prozentsatz der erfolgreich gelösten Würfel bei einer gegebenen Scramble-Länge zeigt, wie zuverlässig die KI den Würfel unter zunehmender Komplexität lösen kann.
2. **Lösungseffizienz:** Die durchschnittliche Anzahl der benötigten Züge zeigt, wie optimal der gewählte Lösungsweg ist.

Für die Tests wurden folgende Parameter festgelegt:

- **Scramble-Längen:** 5 bis 13 Züge
  - Die untere Grenze von 5 Zügen wurde gewählt, da Probleme mit weniger Zügen oft trivial zu lösen sind
  - Die obere Grenze von 13 Zügen entspricht der maximalen Einstellung im GBG-Framework
- **Stichprobengröße:** Für jede Scramble-Länge wurden 25 Testläufe durchgeführt.
- **Würfeltypen:** Sowohl der  $2 \times 2 \times 2$  Pocket Cube als auch der  $3 \times 3 \times 3$  Rubik's Cube wurden mit dem in Kapitel 1.3.3 beschriebenen MCTS-Wrapper bei 100 Iterationen getestet.
- **Abbruchkriterium:** Ein Testlauf wurde als nicht gelöst gewertet, wenn nach 51 Zügen keine Lösung gefunden wurde

Jeder Testlauf wurde protokolliert und umfasste folgende Schritte:

- Generierung einer zufälligen Scramble-Sequenz der vorgegebenen Länge
- Lösung des verdrehten Würfels durch den jeweiligen KI-Agenten
- Abbruch und Wertung als nicht gelöst, wenn nach 51 Zügen keine Lösung gefunden wurde

Zu jedem Testlauf wurden folgende Daten protokolliert:

- Die angewandte Scramble-Sequenz
- Die vom KI-Agenten gefundene Lösungssequenz
- Ob das Puzzle erfolgreich gelöst wurde oder nicht
- Die Länge der Lösungssequenz

Die Scramble-Sequenzen wurden anschließend als Eingabe für zwei etablierte deterministische Lösungsalgorithmen verwendet:

- CubeTwister mit integriertem Kociemba-Solver
- twsearch, ein Kommandozeilentool aus dem Twizzle-Ökosystem [Cubc]

Erste Analysen zeigten, dass beide deterministische Ansätze identische Ergebnisse in Bezug auf Lösungsrate und Lösungseffizienz liefern, weshalb sie in der Auswertung zusammengefasst werden konnten.

Zur systematischen Auswertung der gesammelten Daten wurde ein zweistufiger Prozess entwickelt:

1. **Datenkonsolidierung:** Ein speziell entwickeltes Python-Skript extrahierte die relevanten Informationen aus den Protokolldateien und konsolidierte sie in einem strukturierten Format.
2. **Vergleichsanalyse:** Ein weiteres Skript ermittelte die Länge der Vergleichslösungen aus den deterministischen Algorithmen und erstellte eine tabellarische Zusammenfassung in einem Spreadsheet für die abschließende Datenauswertung.
3. **Visuelle Nachvollziehbarkeit:** Die vollständigen Scramble- und Lösungssequenzen wurden im Spreadsheet gespeichert, um als Basis für eine visuelle Analyse zu dienen.

## 4.2 Ergebnisse

Die Auswertung der Testdaten zeigt deutliche Unterschiede zwischen den KI-basierten und deterministischen Lösungsansätzen. Die Ergebnisse lassen sich in zwei Hauptkategorien einteilen:

### 4.2.1 Lösungsrate

Wie erwartet konnten die deterministischen Algorithmen alle getesteten Würfel unabhängig von der Scramble-Länge lösen.

Auch der KI-Agent erreichte beim Pocket Cube ( $2 \times 2 \times 2$ ) eine Lösungsrate von 100%. Somit gibt es für diesen Würfel keine erkennbare Grenze der Lösungsfähigkeit – im getesteten Rahmen wurde jede Konfiguration gelöst.

Beim Rubik's Cube ( $3 \times 3 \times 3$ ) zeigten sich hingegen folgende Muster:

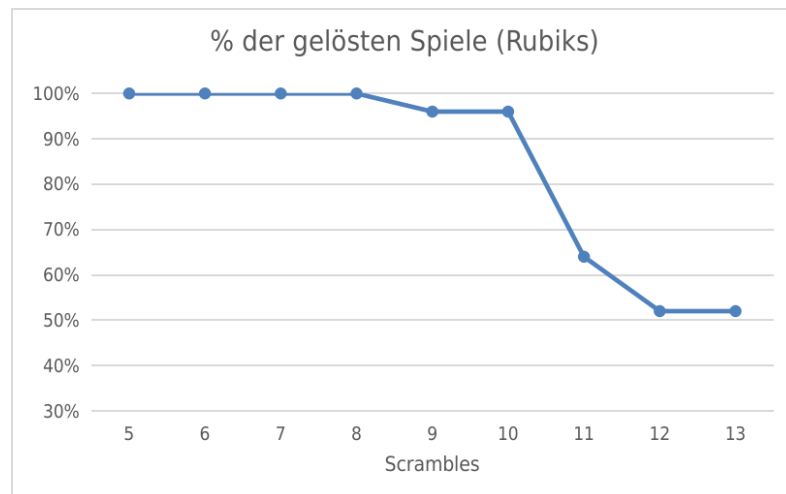


Abbildung 5: Lösungsrate für Rubik's Cube ( $3 \times 3 \times 3$ )

- Bis zu einer Scramble-Länge von 10 Zügen bleibt die Erfolgsrate über 95%
- Zwischen 10 und 11 Zügen ist ein scheinbar drastischer Abfall der Erfolgsrate von über 95% auf unter 65% zu beobachten. Dieser Abfall könnte jedoch optisch überbetont sein, da der Wert von 96% bei Scramble=10 ungewöhnlich hoch ausfällt. Ein Vergleich mit [Kon23], Abbildung 15, legt nahe, dass bei einer größeren Stichprobengröße bei gleicher Scramble-Länge nur ca. 82% erreicht werden. Im Kontext eines gleichmäßigeren Verlaufs erscheint der Rückgang damit weniger abrupt.
- Bei 12 und 13 Zügen stabilisiert sich die Erfolgsrate bei knapp über 50%
- Dieses Verhalten deutet auf eine kritische Schwelle in der Problemkomplexität hin, ab der die Leistungsfähigkeit des KI-Agenten deutlich nachlässt

### 4.2.2 Lösungseffizienz

Hinsichtlich der Lösungslänge ergaben sich folgende Erkenntnisse:

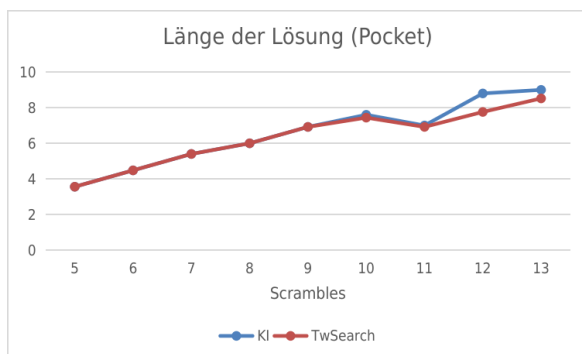


Abbildung 6: Durchschnittliche Lösungslänge für Pocket Cube ( $2 \times 2 \times 2$ )

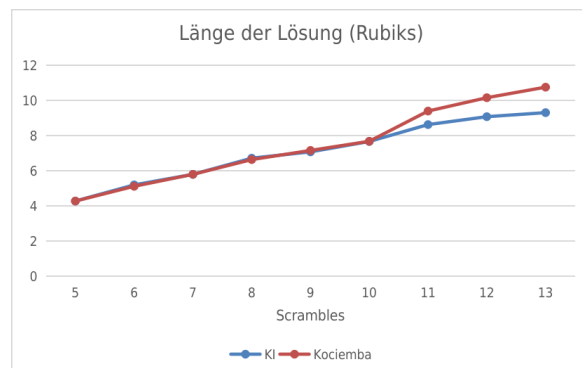


Abbildung 7: Durchschnittliche Lösungslänge für Rubik's Cube ( $3 \times 3 \times 3$ )

- **Pocket Cube:**

- Bei geringen Scramble-Längen (5-11 Züge) sind die Lösungslängen von KI-Agent und deterministischem Algorithmus nahezu identisch
- Ab 12 Zügen werden die Lösungen des KI-Agenten tendenziell ineffizienter

- **Rubik's Cube:**

- Bei den Lösungslängen des  $3 \times 3 \times 3$  Würfels ist ein interessantes Muster zu beobachten
- Die Lösungskurven verlaufen zunächst nahezu deckungsgleich
- Ab 11 Zügen scheinen die KI-Lösungen kürzer zu werden als die deterministischen Lösungen
- Dieses scheinbare Paradoxon lässt sich jedoch durch die sinkende Lösungsrate erklären

**Wichtige methodische Anmerkung:** Die Berechnung der durchschnittlichen Lösungslänge berücksichtigt nur erfolgreiche Lösungen. Bei steigender Scramble-Länge kann der KI-Agent zunehmend nur noch die einfacheren Fälle lösen, während die komplexeren Fälle (die potenziell längere Lösungswege erfordern würden) aus der Berechnung herausfallen, da sie nicht gelöst wurden. Dies führt zu einem Selektionseffekt: Die verbleibenden gelösten Fälle sind tendenziell diejenigen, die einfacher zu lösen waren und daher kürzere Lösungswege ermöglichen. Da die deterministischen Algorithmen hingegen alle Fälle lösen konnten (100% Erfolgsrate), fließen hier auch die komplexeren Fälle mit längeren Lösungswegen in die Durchschnittsberechnung ein. Bei korrekter Interpretation bedeutet

dies, dass die KI-Agenten bis zur kritischen Schwelle von 10-11 Zügen ähnlich effiziente Lösungen wie die deterministischen Algorithmen finden können, bei höherer Komplexität jedoch ähnlich wie beim Pocket Cube ineffizienter werden.

### 4.2.3 Vergleichsanalyse der erfolgreich gelösten Fälle

Um dem beschriebenen Selektionseffekt entgegenzuwirken, wurde eine zusätzliche Analyse durchgeführt, bei der ausschließlich jene Fälle berücksichtigt wurden, die sowohl von den KI-Agenten als auch von den deterministischen Algorithmen erfolgreich gelöst wurden. Die Schnittmenge gleicht den Selektionseffekt aus, der entsteht, wenn bei höherer Komplexität nur die einfacheren Fälle für die KI ausgewertet werden.

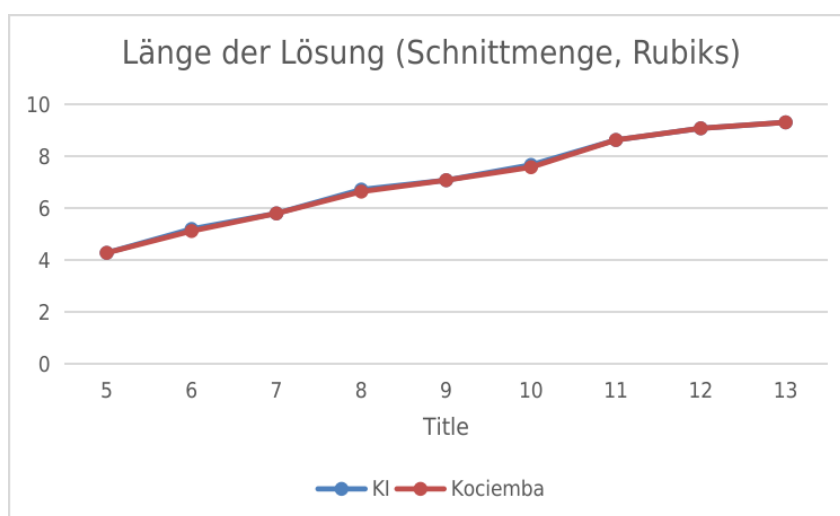


Abbildung 8: Vergleich der Lösungslängen für die Schnittmenge erfolgreich gelöster Fälle ( $3 \times 3 \times 3$ )

Die Ergebnisse bestätigen die Hypothese: Betrachtet man nur identische Ausgangskonfigurationen, so erreichen die KI-Agenten eine mit den deterministischen Algorithmen nahezu identische Lösungseffizienz. Damit rücken die Ergebnisse stärker in Einklang mit den Beobachtungen am Pocket Cube, bei dem bei voller Lösungsrate nur geringe Einbußen in der Lösungseffizienz zu beobachten waren.

Dies zeigt, dass die KI-Agenten für die Fälle, die sie lösen können, tatsächlich effiziente Lösungsstrategien finden. Ihre Hauptlimitation liegt weniger in der Qualität der Lösungen, sondern in der abnehmenden Lösungsrate bei zunehmender Problemkomplexität.

Es wird bestätigt, dass ein Selektionseffekt besteht. Allerdings zeigt sich kein erwarteter Verlust an Effizienz bei höherer Komplexität, und die Ergebnisse beider Kandidaten sind nahezu identisch. Aufgrund der erzielten Ergebnisse beim Pocket Cube kann man jedoch vermuten, dass es beim KI-Agenten dennoch einen geringen Effizienzverlust bei hoher Komplexität geben könnte, wenn er in der Lage wäre, diese komplexen Fälle zu lösen.

### 4.3 Analyse und Interpretation der Ergebnisse

Die Ergebnisse zeigen, dass die KI-Agenten bei moderater Problemkomplexität (bis etwa 10 Züge) mit den deterministischen Algorithmen mithalten können. Bei höherer Komplexität zeigen sich jedoch zunehmend die Grenzen des verwendeten Reinforcement-Learning-Ansatzes.

Zur Einordnung der Ergebnisse ist anzumerken, dass modernste Deep Learning-Ansätze wie DeepCubeA [Ago+19] eine Erfolgsrate von 100% für zufällig verdrehte Würfel erreichen, wobei in etwa 60% der Fälle ein kürzester Pfad zur Lösung gefunden wird. Allerdings verwenden diese Ansätze deutlich komplexere Netzwerkarchitekturen und benötigen entsprechend mehr Rechenressourcen. Der hier durchgeführte Vergleich fokussiert sich auf die Leistungsfähigkeit des ressourceneffizienteren n-tuple-basierten Ansatzes des GBG-Frameworks im Vergleich zu etablierten deterministischen Algorithmen.

Die systematische Analyse der protokollierten Sequenzen ermöglicht tiefere Einblicke in die unterschiedlichen Lösungsstrategien. Interessanterweise zeigte sich, dass bereits die tabellarische Darstellung der Lösungssequenzen ausreichend ist, um Muster wie Wiederholungsschleifen (wie “U U U U U...”) oder ineffiziente Zugfolgen zu identifizieren. Die visuelle Rekonstruktion in Twizzle bietet zwar eine anschaulichere Darstellung, liefert jedoch keine wesentlichen analytischen Mehrwerte gegenüber der direkten Sequenzanalyse. Diese qualitative Analyse ergänzt die quantitativen Ergebnisse und hilft, systematische Unterschiede zwischen den Lösungsansätzen zu identifizieren.

Die systematische Durchführung der Tests nach der beschriebenen Methodik lieferte eine umfangreiche Datenbasis zur Beantwortung der Forschungsfragen. Die Archivierung der vollständigen Sequenzen eröffnet zudem die Möglichkeit einer detaillierten visuellen Nachanalyse: Für jeden getesteten Fall können sowohl der Ausgangszustand (Scramble) als auch die gewählten Lösungswege (KI-basiert und deterministisch) in Twizzle oder anderen Visualisierungswerkzeugen rekonstruiert werden.

Während die visuelle 3D-Darstellung des Würfels für das allgemeine Verständnis intuitiver ist, zeigte sich überraschenderweise, dass für die detaillierte Analyse der algorithmischen Muster die reine Betrachtung der protokollierten Sequenzen oft ausreichend und in manchen Fällen sogar effektiver ist. So lassen sich beispielsweise Wiederholungsmuster oder ineffiziente Zugfolgen direkt in der tabellarischen Notation erkennen, ohne dass eine 3D-Visualisierung notwendig wäre. Die Kombination aus quantitativer Analyse und qualitativer Sequenzbetrachtung ermöglicht somit ein umfassendes Verständnis der unterschiedlichen Lösungsansätze.

## 5 Zusammenfassung und Ausblick

Die vorliegende Arbeit befasste sich mit der Verbesserung der Analyse von Rubik's-Cube-Algorithmen durch einen visuellen Ansatz und den systematischen Vergleich zwischen Reinforcement-Learning-Methoden und deterministischen Lösungsalgorithmen. Im Folgenden werden die zentralen Erkenntnisse zusammengefasst und ein Ausblick auf mögliche zukünftige Forschungsrichtungen gegeben.

### 5.1 Zusammenfassung der Arbeit

**Einleitung (Kapitel 1):** In der Einleitung wurden die Grundlagen des Reinforcement Learning erläutert und dessen Anwendung auf Puzzle-Spiele, insbesondere den Rubik's Cube, motiviert. Die besonderen Herausforderungen des Rubik's Cube als Testumgebung für KI-Algorithmen wurden dargestellt und das General Board Game (GBG) Framework als wichtige Plattform für diese Forschung eingeführt. Die Problemstellung der Arbeit wurde definiert: die bestehende 2D-Darstellung des Würfels im GBG-Framework ist unintuitiv, und es fehlt ein systematischer Vergleich zwischen KI-basierten und deterministischen Lösungsansätzen. Auf Basis dieser Problemstellung wurden drei zentrale Ziele formuliert: Die Entwicklung einer interaktiven 3D-Visualisierung, die Integration eines deterministischen Referenzalgorithmus und die vergleichende Analyse beider Lösungsansätze.

**Analyse potenzieller Visualisierungswerkzeuge (Kapitel 2):** Die Limitationen der ursprünglichen 2D-Darstellung des Würfels im GBG-Framework wurden detailliert analysiert und Anforderungen an ein verbessertes Visualisierungswerkzeug definiert. Nach einer umfassenden Recherche wurden potenzielle Werkzeuge in drei Kategorien eingeteilt: webbasierte JavaScript-Lösungen, plattformunabhängige Anwendungen und Java-basierte Werkzeuge. Eine systematische Bewertung nach vier Hauptkriterien (native Java-Kompatibilität, integrierter Solver, Unterstützung für verschiedene Würfeltypen und visuelle Qualität) führte zur Auswahl von CubeTwister als vielversprechendstes Werkzeug für die Integration.

**Implementation des Visualisierungswerkzeugs (Kapitel 3):** Die technischen Herausforderungen bei der Integration von CubeTwister in das GBG-Framework wurden ausführlich dargestellt. Diese umfassten die veraltete Distributionsform, fehlende Dokumentation und keinen direkten Quellcode-Zugriff. Der Implementationsprozess über Extraktion der JAR-Datei und Reverse-Engineering-Ansätze wurde beschrieben, ebenso wie der Versuch einer Integration über den ScriptPlayer von CubeTwister. Die erreichten Implementationsfortschritte wurden bilanziert und die entstandene 3D-Visualisierung im Vergleich zur ursprünglichen 2D-Darstellung bewertet. Schließlich wurde eine alternative Lösung präsentiert, die CubeTwister als eigenständige Anwendung und Twizzle als ergänzendes Visualisierungswerkzeug nutzt, um die technischen Integrationsprobleme zu

umgehen.

**Testaufbau und Durchführung (Kapitel 4):** Der systematische Testaufbau zur vergleichenden Analyse von KI-basierten und deterministischen Lösungsansätzen wurde detailliert dargestellt. Die Testmethodik umfasste die Untersuchung von Lösungsrate und Lösungseffizienz für Scramble-Längen von 5 bis 13 Zügen, mit je 25 Testläufen pro Konfiguration. Die empirischen Ergebnisse zeigten, dass die KI-Agenten bei moderater Problemkomplexität konkurrenzfähig sind, bei höherer Komplexität aber signifikante Leistungseinbußen aufweisen. Besonders auffällig war der kritische Schwellenwert von 10-11 Zügen beim Rubik's Cube, ab dem die Erfolgsrate dramatisch abfiel. Die Analyse der Lösungseffizienz zeigte, dass die KI-Agenten bei niedrigen bis mittleren Scramble-Längen ähnlich effiziente Lösungen wie deterministische Algorithmen finden können. Bei zunehmender Komplexität sinkt jedoch ihre Lösungsrate deutlich, während die erfolgreich gelösten Fälle weiterhin effiziente Lösungswege aufweisen – ein Effekt, der durch die selektive Lösbarkeit einfacherer Fälle bei höherer Komplexität erklärt werden kann.

## 5.2 Beantwortung der Forschungsfragen

Die in Kapitel 1 formulierten Forschungsfragen können auf Basis der erzielten Ergebnisse wie folgt beantwortet werden:

1. **Interaktive 3D-Visualisierung:** Die Integration einer 3D-Visualisierung in das GBG-Framework konnte teilweise realisiert werden. Trotz der technischen Herausforderungen wurde eine interaktive 3D-Darstellung des Würfels implementiert, die gegenüber der ursprünglichen 2D-Ansicht deutliche Vorteile bietet. Die vollständige Integration scheiterte an technischen Hindernissen, konnte jedoch durch einen alternativen Workflow kompensiert werden, der eigenständige Visualisierungswerkzeuge in den Analyseprozess einbezieht.
2. **Lösungsqualität der KI-Agenten:** Die systematische Analyse zeigte, dass die im GBG-Framework trainierten KI-Agenten bei moderater Problemkomplexität (bis etwa 10 Züge) konkurrenzfähige Lösungen im Vergleich zu deterministischen Algorithmen liefern können. Bei zunehmender Komplexität zeigen sich jedoch signifikante Unterschiede: Die Lösungsrate der KI-Agenten fällt drastisch ab, während deterministische Algorithmen weiterhin 100% der Fälle lösen. Interessanterweise bleiben die Lösungswege der KI-Agenten für die noch lösbaren Fälle effizient – ein Effekt, der durch die selektive Lösbarkeit tendenziell einfacherer Konfigurationen erklärt werden kann, während komplexere Fälle nicht mehr gelöst werden.
3. **Stärken und Schwächen der Lösungsansätze:** Die KI-basierten Ansätze zeigen ihre Stärken in der Anpassungsfähigkeit und der Fähigkeit, gänzlich ohne domänenspezifisches Wissen zu lernen. Ihre Schwächen liegen in der begrenzten Generalisie-

rungsfähigkeit bei hoher Problemkomplexität und der fehlenden Garantie optimaler Lösungen. Die deterministischen Algorithmen bieten dagegen hohe Zuverlässigkeit, Optimalität und Vorhersagbarkeit, sind aber auf spezifisches Domänenwissen angewiesen und weniger flexibel in der Anwendung auf andere Problemfelder.

4. **Beitrag der 3D-Visualisierung:** Die implementierte 3D-Visualisierung bietet eine intuitivere Darstellung der Würfelzustände im Vergleich zur ursprünglichen 2D-Ansicht. Während die räumliche Darstellung und die interaktiven Elemente das allgemeine Verständnis verbessern, zeigte die Forschung überraschenderweise, dass für die detaillierte Analyse von KI-Lösungsstrategien die systematische Protokollierung und tabellarische Darstellung der Zugsequenzen oft ausreichend und in manchen Fällen sogar effektiver ist. Algorithmische Muster wie Wiederholungsschleifen oder ineffiziente Zugfolgen lassen sich direkt in der notierten Sequenz erkennen. Die 3D-Visualisierung dient somit primär der verbesserten Benutzerfreundlichkeit und weniger als analytisches Werkzeug.

### 5.3 Kritische Bewertung der Ergebnisse

Die Ergebnisse der Arbeit unterliegen einigen Limitationen, die bei der Interpretation berücksichtigt werden sollten:

- **Technische Integration:** Die vollständige Integration von CubeTwister in das GBG-Framework konnte nicht abgeschlossen werden. Dies schränkt die direkte Anwendbarkeit der 3D-Visualisierung im Rahmen des Frameworks ein und erfordert derzeit noch manuelle Zwischenschritte in der Analyse.
- **Eingeschränkter Parameterraum:** Die Tests wurden mit spezifischen Konfigurationen der KI-Agenten durchgeführt. Weitere Parameter wie unterschiedliche Lernraten, Netzwerkarchitekturen oder MCTS-Einstellungen wurden nicht systematisch variiert, was die Generalisierbarkeit der Ergebnisse einschränkt.
- **Beschränkte Stichprobengröße:** Mit 25 Testläufen pro Konfiguration ist die Stichprobengröße zwar ausreichend für grundlegende Schlussfolgerungen, könnte aber für statistische Feinanalysen erweitert werden.
- **Fokus auf zwei Würfeltypen:** Die Untersuchung beschränkte sich auf den  $2 \times 2 \times 2$  Pocket Cube und den  $3 \times 3 \times 3$  Rubik's Cube. Andere Varianten oder höherdimensionale Puzzle wurden nicht betrachtet.

Trotz dieser Einschränkungen liefert die Arbeit wertvolle Erkenntnisse zum Vergleich von KI-basierten und deterministischen Lösungsansätzen für den Rubik's Cube und verbessert durch die 3D-Visualisierung die Analysemöglichkeiten im GBG-Framework.

## 5.4 Offene Probleme und zukünftige Forschungsrichtungen

Aus den Ergebnissen und Erfahrungen dieser Arbeit ergeben sich mehrere vielversprechende Ansatzpunkte für zukünftige Forschungsarbeiten:

- **Vollständige Integration:** Die begonnene Integration der 3D-Visualisierung in das GBG-Framework könnte vervollständigt werden, möglicherweise durch eine Neuentwicklung mit modernen Java-3D-Bibliotheken statt der Verwendung veralteter Software wie CubeTwister.
- **Hybride Lösungsansätze:** Die Kombination von KI-basierten und deterministischen Ansätzen könnte erforscht werden, um die jeweiligen Stärken zu nutzen. Beispielsweise könnte ein KI-Agent für die initiale Exploration des Lösungsraums mit einem deterministischen Algorithmus für die finale Optimierung kombiniert werden.
- **Verbesserung der KI-Agenten:** Die identifizierten Schwächen der KI-Agenten bei höherer Problemkomplexität könnten durch erweiterte Trainingsmethoden, alternative Netzwerkarchitekturen oder angepasste Reinforcement-Learning-Algorithmen adressiert werden.
- **Transferlernen:** Die Untersuchung, inwieweit Agenten, die für einen bestimmten Würfeltyp trainiert wurden, ihr Wissen auf andere Varianten übertragen können, bietet interessante Forschungsperspektiven zur Generalisierungsfähigkeit von Reinforcement-Learning-Methoden.
- **Erweiterte Visualisierungstechniken:** Die Entwicklung fortschrittlicherer Visualisierungstechniken, die nicht nur den Würfelzustand, sondern auch die internen Bewertungsfunktionen der KI-Agenten darstellen können, würde tiefere Einblicke in die Entscheidungsprozesse ermöglichen.
- **Anwendung auf andere Puzzle:** Die entwickelten Methoden könnten auf andere komplexe Puzzle wie das 15-Puzzle, Sokoban ähnlich wie bei [Ago+19] oder höherdimensionale Würfel übertragen werden, um die Generalisierbarkeit der Ergebnisse zu untersuchen.

## 5.5 Fazit

Die vorliegende Arbeit hat einen Beitrag zum besseren Verständnis der Leistungsfähigkeit von KI-basierten Lösungsansätzen für den Rubik's Cube im Vergleich zu klassischen deterministischen Algorithmen geleistet. Durch die Entwicklung einer verbesserten visuellen Darstellung wurde die intuitive Erfassbarkeit der Würfelzustände erhöht, was besonders für Bildungszwecke von Bedeutung ist. Für die Forschung erwies sich interessanterweise

die systematische Protokollierung und Analyse der Zugsequenzen als ebenso wertvoll wie die visuelle Darstellung selbst.

Die empirischen Ergebnisse zeigen, dass Reinforcement-Learning-Methoden bei moderater Problemkomplexität durchaus mit deterministischen Algorithmen konkurrieren können, bei steigender Komplexität jedoch an ihre Grenzen stoßen. Diese Erkenntnis kann als Ausgangspunkt für die Entwicklung verbesserter KI-Ansätze dienen, die auch bei höherer Komplexität zuverlässige und effiziente Lösungen liefern.

Insgesamt verdeutlicht die Arbeit das Potenzial der Kombination von visuellen Analysemethoden und systematischen Vergleichsstudien für das Verständnis komplexer Entscheidungsprobleme und der Leistungsfähigkeit verschiedener Lösungsstrategien. Sie legt damit eine Grundlage für weitere Forschungsarbeiten, die auf eine vertiefte Integration von KI-Methoden und visueller Analyse abzielen.

## 6 Quellenverzeichnis

- [Ago+19] Forest Agostinelli u. a. „Solving the Rubik’s cube with deep reinforcement learning and search“. In: *Nature Machine Intelligence* 1.8 (Aug. 2019), S. 356–363. ISSN: 2522-5839. DOI: [10 . 1038 / s42256 - 019 - 0070 - z](https://doi.org/10.1038/s42256-019-0070-z). URL: <https://www.nature.com/articles/s42256-019-0070-z>.
- [Ani] *AnimCubeJS - the Rubik’s Cube Animation Simulator*. URL: <https://animcubejs.cubing.net/animcubejs.html#usage> (besucht am 02. 12. 2024).
- [Ara23] Kevin Arauz. *kjarj54/Proyecto-CuboRubik*. 29. Nov. 2023. URL: <https://github.com/kjarj54/Proyecto-CuboRubik> (besucht am 21. 03. 2025).
- [Cuba] *CubeTwister*. URL: <https://www.randelshofer.ch/cubetwister/> (besucht am 10. 11. 2024).
- [Cubb] *cubing/cubing.js: A library for displaying and working with twisty puzzles. Also currently home to the code for Twizzle*. URL: <https://github.com/cubing/cubing.js> (besucht am 21. 03. 2025).
- [Cube] *cubing/twsearch*. 7. Apr. 2025. URL: <https://github.com/cubing/twsearch> (besucht am 07. 04. 2025).
- [Far25] Andrew Farkas. *HactarCE/Hyperspeedcube*. 17. März 2025. URL: <https://github.com/HactarCE/Hyperspeedcube> (besucht am 21. 03. 2025).
- [Gao24] Tingran Gao. *trgao10/Cubik*. 8. Juli 2024. URL: <https://github.com/trgao10/Cubik> (besucht am 21. 03. 2025).
- [Gon24] Vasu Gondaliya. *vasu-gondaliya/rubiks-cube*. 5. Okt. 2024. URL: <https://github.com/vasu-gondaliya/rubiks-cube> (besucht am 10. 11. 2024).
- [Jav] *Java Web Start*. In: *Wikipedia*. Page Version ID: 246251089. 27. Juni 2024. URL: [https://de.wikipedia.org/w/index.php?title=Java\\_Web\\_Start&oldid=246251089](https://de.wikipedia.org/w/index.php?title=Java_Web_Start&oldid=246251089) (besucht am 26. 03. 2025).
- [Kon23] Wolfgang Konen. *Towards Learning Rubik’s Cube with N-tuple-based Reinforcement Learning*. 2023. arXiv: [2301.12167 \[cs.LG\]](https://arxiv.org/abs/2301.12167). URL: <https://arxiv.org/abs/2301.12167>.
- [Kuz24] Andrey Kuzmin. *wOrm/elm-cubik*. 17. Mai 2024. URL: <https://github.com/wOrm/elm-cubik> (besucht am 10. 11. 2024).
- [McA+] S. McAleer u. a. „Solving the Rubik’s Cube with Approximate Policy Iteration“. In: *International Conference on Learning Representations* (). URL: <https://par.nsf.gov/biblio/10120458>.

- 
- [SB18] Richard S. Sutton und Andrew G. Barto. *Reinforcement Learning, second edition: An Introduction*. Google-Books-ID: sWV0DwAAQBAJ. MIT Press, 13. Nov. 2018. 549 S. ISBN: 978-0-262-03924-6.
- [Twi] *Twizzle*. URL: <https://alpha.twizzle.net/> (besucht am 21.03.2025).
- [Wan24] Yifeng Wang. *doodlewind/freecube*. 30. Okt. 2024. URL: <https://github.com/doodlewind/freecube> (besucht am 10.11.2024).
- [Wol25] WolfgangKonen. *WolfgangKonen/GBG*. original-date: 2016-11-14T10:58:36Z. 31. März 2025. URL: <https://github.com/WolfgangKonen/GBG> (besucht am 09.04.2025).
- [Zha24] Zihao Zhang. *zzh8829/Cubez*. 4. Nov. 2024. URL: <https://github.com/zzh8829/Cubez> (besucht am 21.03.2025).

# Erklärung über die selbständige Abfassung der Arbeit

Ich versichere, die von mir vorgelegte Arbeit selbständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht.

Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Köln, 10.04.2025

S. Wohler

---

(Ort, Datum, Unterschrift)