TUBAF
Die Ressourcenuniversität.
Seit 1765.

Research Article

Algorithmische Dominanz vs. Gelerntes Verhalten: Ein Vergleich von A*, Heuristik und PPO in BouncAI

Evaluation klassischer Suchverfahren und moderner Reinforcement Learning Ansätze in einer stochastischen Umgebung

Simon Hörtzsch

13. Januar 2026

TU Bergakademie Freiberg

Zusammenfassung

Die Steuerung autonomer Agenten in dynamischen Echtzeit-Umgebungen stellt eine klassische Herausforderung der Künstlichen Intelligenz dar. Während modellbasierte Suchalgorithmen wie A^* theoretisch optimale Lösungen liefern, skalieren sie oft schlecht mit der Komplexität des Zustandsraums und hängen massiv von der Präzision eines Weltmodells ab. Reinforcement Learning (RL) verspricht als datengetriebener Ansatz, komplexe Handlungsstrategien allein durch Interaktion zu erlernen, kämpft jedoch häufig mit Instabilität und hoher Varianz im Lernprozess.

Diese Arbeit präsentiert eine umfassende Vergleichsstudie vierer unterschiedlicher Agenten-Architekturen in der vertikalen „Infinite Scroller"-Umgebung *BouncAI*: einen reaktiven **Reflex-Agenten**, einen heuristischen **Utility-Agenten**, einen modellbasierten A^***-Agenten** und einen mittels **Proximal Policy Optimization (PPO)** trainierten RL-Agenten. Um den Trainingserfolg des PPO-Modells in dieser hochgradig stochastischen Umgebung zu gewährleisten, wurde ein **Curriculum Learning** Ansatz implementiert, der die Schwierigkeit der Spielphysik stufenweise steigert.

Die Evaluation auf Basis von $N = 10.000$ Simulationsläufen pro Agent zeigt, dass der A^*-Agent mit einem durchschnittlichen Score von 19.030 Punkten die höchste Performance erzielt, sofern ein fehlerfreies Vorhersagemodell

existiert. Das trainierte PPO-Modell erreicht 14.283 Punkte und nähert sich damit auf 95,6 % der Leistung des handoptimierten Utility-Agenten an. Ein zentrales Ergebnis der Studie ist die **strategische Robustheit** des PPO-Agenten: Mit der geringsten Standardabweichung aller Probanden ($\sigma = 3.435$) demonstriert er ein hochgradig konsistentes Verhalten, das unabhängig von Umgebungsfluktuationen verlässliche Mindestperformances garantiert. Die Ergebnisse belegen, dass moderne RL-Verfahren nicht nur mit spezialisierten Heuristiken konkurrieren können, sondern diesen in puncto Verhaltensstabilität oft überlegen sind.

1 Einleitung

Die Entwicklung künstlicher Intelligenz für Videospiele dient seit Jahrzehnten als wichtiges Testfeld für Algorithmen, die Entscheidungen unter Unsicherheit, physikalischen Beschränkungen und striktem Zeitdruck treffen müssen. Spiele fungieren hierbei als kontrollierte Laborumgebungen für Probleme, die in ihrer Kernstruktur — wie etwa die Pfadplanung oder das Ausweichen von Hindernissen — direkt auf reale Anwendungen in der Robotik oder autonomen Systemen übertragbar sind.

Das in dieser Arbeit verwendete Spiel *BouncAI* stellt einen vertikalen Scroller dar, in dem ein Agent durch gezielte Sprünge auf Plattformen an Höhe gewinnen muss. Diese Umgebung ist durch eine hybride Komplexität charakterisiert: Während die Flugbahn des Spielers einer streng deterministischen Parabel folgt, führen bewegliche Plattformen, Windkräfte und stochastisch erscheinende Gegner zu einem dynamisch fluktuierenden Zustandsraum. Ein erfolgreicher Agent muss daher nicht nur reaktiv handeln, sondern die Spielphysik antizipieren und langfristige Trajektorien planen.

Traditionell wurden solche Aufgaben durch spezialisierte Suchalgorithmen (z.B. A^*) gelöst, die den mathematisch optimalen Pfad durch den Zustandsraum berechnen. Solche Ansätze erfordern jedoch ein exaktes „Forward Model“ der Welt und sind rechenintensiv. Heuristische Verfahren hingegen bieten zwar Effizienz, leiden jedoch unter der Notwendigkeit einer manuellen Regeldefinition („Hardcoding“), die häufig unflexibel gegenüber unvorhergesehenen Situationen ist. Mit dem Aufstieg des *Deep Reinforcement Learnings* (RL) hat sich ein Paradigma etabliert, bei dem Agenten Strategien durch Versuch und Irrtum (*Trial-and-Error*) selbstständig entwickeln.

Diese Arbeit untersucht die fundamentale Frage: *Inwieweit kann ein generalistischer, modellfreier Lernalgorithmus wie PPO die Leistung spezialisierter, mit explizitem Domänenwissen ausgestatteter Algorithmen erreichen oder gar hinsichtlich der Verhaltensrobustheit übertreffen?*

Daraus leiten sich folgende zentrale Forschungsfragen (RQ) ab:

- **RQ1:** Wie groß ist die Performance-Lücke zwischen einer gelernten Policy und einer mathematisch optimalen A^*-Suche in einer Physik-Umgebung?
- **RQ2:** Führt das Lernen von Verhaltensmustern (RL) zu einer höheren Konsistenz und Zuverlässigkeit im Vergleich zu rein reaktiven oder heuristischen Ansätzen?

Im Folgenden werden zunächst der Stand der Forschung und die theoretischen Grundlagen der verwendeten Algorithmen erläutert (Kapitel 2). Kapitel 3 beschreibt die methodische Implementierung der vier Agenten und des Trainingsprozesses. In Kapitel 4 erfolgt eine detaillierte statistische Evaluation der Ergebnisse auf Basis von 10.000 Simulationsläufen, gefolgt von einer Diskussion der strategischen Stärken des PPO-Ansatzes. Die Arbeit schließt in Kapitel 5 mit einem Fazit und einem Ausblick auf zukünftige hybride Architekturen.

2 Stand der Forschung

Die autonome Steuerung von Agenten in Videospielen hat sich in den letzten Jahrzehnten von einfachen, regelbasierten Systemen hin zu komplexen, lernenden Architekturen entwickelt.

2.1 Klassische Suchverfahren in dynamischen Umgebungen

Der A^*-Algorithmus [1] gilt als mathematisches Fundament für die Pfadfindung in statischen Graphen. Er garantiert das Auffinden eines kürzesten Pfades, sofern eine zulässige Heuristik existiert. In modernen Echtzeit-Spielen wie *BouncAI* treten jedoch zwei kritische Probleme auf:

1. **Zustandsexplosion:** In einer Umgebung mit beweglichen Plattformen und stochastisch agierenden Gegnern wächst der Suchbaum exponentiell mit der Planungstiefe. Ein klassischer A^* müsste für jeden Zeitschritt alle möglichen Zukunftspositionen aller Objekte simulieren.
2. **Forward Model Abhängigkeit:** Suchverfahren sind *modellbasiert*. Sie benötigen ein exaktes „Forward Model“ (ein mathematisches Abbild der Spielwelt), um zukünftige Zustände vorhersagen zu können [2]. Fehlt dieses Wissen oder ist die Welt zu komplex, versagen klassische Suchalgorithmen.

In der *Mario AI Competition* [2] zeigten A^*-Agenten zwar eine dominante Performance, waren jedoch extrem anfällig für minimale Abweichungen in der Physik-Simulation.

2.2 Reinforcement Learning: Von DQN zu PPO

Reinforcement Learning (RL) bietet einen *modellfreien* Ansatz, bei dem der Agent Strategien allein durch Interaktion mit der Umgebung lernt. Den Durchbruch für neuronale Spiel-KIs markierte *Deep Q-Learning* (DQN) [3], welches den Wert (Q-Wert) jeder Aktion schätzt. DQN leidet jedoch oft unter Instabilität, da kleine Änderungen in den Q-Werten zu drastischen Sprüngen in der Handlungsstrategie führen können.

Ein stabilerer Ansatz sind *Policy-Gradient*-Methoden, die direkt die Handlungsstrategie π_θ optimieren. *Proximal Policy Optimization* (PPO) [4] hat sich hierbei als Industriestandard etabliert. Der entscheidende Vorteil von PPO ist die „Clipped Surrogate Objective Function“. Diese Funktion begrenzt („clipping“), wie stark sich die neue Strategie von der alten unterscheiden darf. Dies verhindert das gefürchteten „Kollabieren“ des Lernprozesses, bei dem ein Agent bereits gelerntes Wissen durch ein zu großes Update schlagartig vergisst. PPO vereint dabei die mathematische Stabilität von *Trust Region Policy Optimization* (TRPO) mit der Implementierungseinfachheit von Gradientenverfahren [5].

2.3 Curriculum Learning und strukturiertes Lernen

Das Erlernen komplexer Aufgaben „from scratch“ scheitert oft an der „Sparse Reward“ Problematik: Ein Agent, der zu Beginn nur zufällig agiert, wird in einer schwierigen Umgebung (z.B. Tier 10000 mit Wind und Gegnern) fast nie eine positive Belohnung erhalten und somit nicht lernen. Bengio et al. [6] führten hierzu das **Curriculum Learning** ein, das auf der pädagogischen Idee basiert, Aufgaben in einer aufsteigenden Schwierigkeitsfolge zu präsentieren. Der Agent lernt erst fundamentale Bewegungsmuster (Springen auf statische Plattformen) und wird erst nach Beherrschung dieser mit weiteren Störfaktoren (Wind, bewegliche Ziele) konfrontiert. Dies erlaubt es dem Optimierungsalgorithmus, kontinuierlich einem steilen Gradienten zu folgen, anstatt in lokalen Minima oder Plateaus zu stagnieren.

2.4 Stabilisierung durch Normalisierung

Die numerische Stabilität tiefer neuronaler Netze in RL-Szenarien ist häufig durch extrem schwankende Belohnungen (Rewards) gefährdet. Zur Kompensation wird *Layer Normalization* [7] eingesetzt. Dabei werden die Aktivierungen innerhalb des Netzwerks für jedes Datenpaket normiert. Dies reduziert den internen „Covariate Shift“ und erlaubt höhere Lernraten bei gleichzeitig stabilerer Konvergenz, was für die effiziente Merkmalsextraktion aus dem 72-dimensionalen Zustandsraum von *BouncAI* essenziell ist.

3 Methodik und Agenten

In diesem Kapitel werden die technischen Implementierungen der untersuchten Agenten-Architekturen detailliert beschrieben. Alle Agenten operieren in derselben Umgebung *BouncAI*, unterscheiden sich jedoch fundamental in ihrer Informationsverarbeitung und Entscheidungsfindung.

3.1 Klassische Agenten

Diese Agenten basieren auf explizit programmierten Regeln und nutzen physikalisches Domänenwissen.

3.1.1 Greedy Reflex Agent

Der Reflex-Agent repräsentiert die einfachste Stufe der Intelligenz. Er verfolgt eine rein reaktive, gierige Strategie:

1. **Umgebungsscan:** Er scannt die Umgebung nach der höchsten, innerhalb eines Zeitfensters erreichbaren Plattform.
2. **Bewegungsplanung:** Er berechnet die notwendige horizontale Bewegung, um das Zentrum dieser Plattform zu erreichen.
3. **Einschränkungen:** Er ignoriert Gegner, Windkräfte und die langfristige Erreichbarkeit höherer Ebenen.

Seine Effektivität sinkt drastisch in Tier-Bereichen mit hoher Gegnerdichte, da er keinerlei Ausweichlogik besitzt.

3.1.2 Utility-based Agent (Heuristic)

Der Utility-Agent nutzt eine gewichtete Nutzenfunktion $U(s, a)$, um Aktionen zu bewerten. Er führt für potenzielle Zielplattformen eine Kurzzeit-Simulation durch, um das Risiko einer Kollision abzuwägen. Die Funktion ist definiert als:

$$U = \omega_1 \cdot \text{Höhengewinn} - \omega_2 \cdot \text{Kollisionsrisiko} + \omega_3 \cdot \text{Optionen} \tag{1}$$

Die Gewichte wurden empirisch auf $\omega_1 = 10$ (Höhengewinn), $\omega_2 \approx 5000$ (Risiko, exponentiell) und $\omega_3 = 100$ (Zukunftsoptionen) festgelegt, um Sicherheit über bloßen Gewinn zu stellen. Hierbei werden Gegnerpositionen durch ein lineares Modell prädiziert. Ein „Desperate Mode“ (Rettungsmodus) wird aktiviert, wenn der Agent zu fallen droht.

3.2 Der A^*-Agent (Heuristic Search)

Der A^*-Agent stellt die komplexeste algorithmische Lösung dar. Er kombiniert globale Pfadsuche mit einer hierarchischen Entscheidungslogik in vier Ebenen:

1. **Survival Reflex (Layer 1):** Bevor eine Suche gestartet wird, prüft der Agent auf unmittelbare Lebensgefahr durch Gegner. In diesem Fall wird die Suche abgebrochen und ein deterministisches Ausweichmanöver eingeleitet.
2. **Plan Validation (Layer 2):** Existiert ein Pfad aus einem vorherigen Zeitschritt, wird geprüft, ob dieser durch unvorhergesehene Umgebungsänderungen (z.B. Wind) noch physikalisch erreichbar ist.
3. A^* **Search (Layer 3):** Die Welt wird als Graph diskretisiert, wobei Plattformen als Knoten fungieren.
    - **Suchhorizont:** Der Algorithmus sucht den Pfad über den **gesamten sichtbaren Bereich** (alle Plattformen bis zum oberen Bildschirmrand, $y = 0$), was ca. 10–15 zukünftigen Stufen entspricht. Der Agent nutzt dabei eine interne Kopie der Spielphysik (Forward Model), um zukünftige Zustände deterministisch zu prädizieren. Da die Simulation in *BouncAI* (bis auf die Generierung neuer Objekte) deterministisch ist, agiert dieses Modell als quasi-perfektes Orakel, das Wind und Gegnerbewegungen exakt vorausberechnet.
    - **Kosten** $g(n)$**:** Summe aus der Flugzeit Δt, dem akkumulierten Risiko (Nähe zu Gegnern entlang der Trajektorie) und dem „Effort“ (benötigte Horizontalgeschwindigkeit).
    - **Heuristik** $h(n)$**:** Die vertikale Distanz y zum oberen Bildschirmrand. Da kleinere y-Werte höhere Positionen markieren, minimiert A^* effektiv die Höhendifferenz.
4. **Motor Control (Layer 4):** Die Umwandlung des Pfades in diskrete Aktionen (`Left`, `Right`, `Wait`), wobei Windkräfte durch eine Vorsteuerung kompensiert werden.

3.3 Der PPO-Agent (Deep Reinforcement Learning)

Im Gegensatz zu den regelbasierten Agenten lernt das PPO-Modell die Spielphysik ohne initiales Wissen.

3.3.1 Zustandsrepräsentation (Observation Space)

Der Agent erhält einen 72-dimensionalen Merkmalsvektor, der die Umgebung wie folgt kodiert:

- **Spieler-Dynamik (5):** Position, normalisierte Geschwindigkeit ($v_x/10, v_y/20$) und ein binärer Ground-Contact-Flag.
- **Objekt-Sensorik (66):** Die 8 nächsten Plattformen (je 6 Parameter: $\Delta x, \Delta y, w, v_x$, Typ, Präsenz), die 3 nächsten Gegner (je 4 Parameter) und 2 Windzonen (je 3 Parameter).
- **Globaler Kontext (1):** Das aktuelle Tier-Niveau zur Einordnung der Schwierigkeit.

3.3.2 Netzwerkarchitektur und Hyperparameter

Das Modell nutzt eine Actor-Critic Architektur mit einem geteilten Feature-Extraktor (*Shared Backbone*), wodurch beide Teilnetze von denselben Low-Level-Repräsentationen profitieren und die Trainingseffizienz gesteigert wird.

- **Struktur:** Drei vollvernetzten Schichten (512 $\rightarrow$ 256 $\rightarrow$ 128 Neuronen). Um die Trainingsstabilität zu maximieren, wird nach jeder linearen Transformation *Layer Normalization* angewendet, was die Eingabeverteilung normalisiert (Mittelwert 0, Varianz 1) und so das Gradientenverhalten glättet. Als Aktivierungsfunktion dient die *Rectified Linear Unit* (ReLU, $f(x) = \max(0, x)$), welche durch ihre Nichtlinearität das Lernen komplexer Zusammenhänge ermöglicht und gleichzeitig das Problem verschwindender Gradienten bei tiefen Netzwerken effektiv verhindert. Diese Architektur wurde als optimaler Kompromiss zwischen Modellkapazität und Rechenaufwand ermittelt.
- **Training:** Batch-Size von 40.960 Samples, Learning Rate $5 \cdot 10^{-5}$ und ein Entropy-Koeffizient, der während des Polishing-Prozesses auf $0,0005$ gesenkt wurde. Das Training erfolgte über eine Belohnungsfunktion R, die den vertikalen Fortschritt (Δy) positiv gewichtet und Kollisionen mit einem Malus versieht.

3.3.3 Reward Shaping

Die Reward-Funktion wurde als dichtes Signal konzipiert, das den Agenten nicht nur für das Endergebnis belohnt, sondern kontinuierlich zur Höhenmaximierung führt. Der Reward r_t zum Zeitpunkt t setzt sich aus drei Hauptkomponenten zusammen:

1. **Vertical Progress Reward:** Ein proportionaler Gewinn basierend auf der vertikalen Distanz Δy, die seit dem letzten Frame zurückgelegt wurde:

$$r_{\text{climb}} = \begin{cases} 0,0017 \cdot \Delta y & \text{falls } \Delta y > 0 \\ 0 & \text{sonst} \end{cases} \tag{2}$$

   Da der reine Game-Score in Pixeln gemessen wird, skaliert dieser Faktor (ca. $1/600$) die Belohnung auf einen für das neuronale Netz verarbeitbaren Bereich.

2. **Survivability Incentives:**
   - **Survival Bonus:** $+0,005$ pro Frame, um das reine Überleben zu belohnen.
   - **Collision Penalty:** Ein Malus von $-2,0$ für Kollisionen mit Gegnern.
   - **Fall Penalty:** $-1,0$ für das Herausfallen aus dem Bildschirmbereich.

   Die Bestrafung für Gegnerkollisionen (doppelt so hoch wie für Abstürze) erzieht den Agenten dazu, Feinde aktiv zu meiden, anstatt riskante Sprünge zu wagen.

3. **Action Stability Penalty:** Um das ständige Oszillieren zwischen Aktionen (z.B. schnelles Wechseln von Links nach Rechts) zu unterdrücken, wird eine Strafe von $-0,005$ verhängt, wenn die gewählte Aktion von der vorherigen abweicht.

Formal lässt sich die Reward-Funktion R als gewichtete Summe definieren:

$$R_{total} = 0,0017 \cdot \Delta y + R_{\text{alive}} - 2,0 \cdot \mathbb{I}_{\text{collision}} - 1,0 \cdot \mathbb{I}_{\text{fall}} \tag{3}$$

Dieser starke Malus unterscheidet den Ansatz von rein gierigen Heuristiken und priorisiert die langfristige Existenzsicherung.

3.3.4 Trainings-Historie und Modell-Evolution

Das finale Modell ist das Ergebnis eines iterativen Entwicklungsprozesses, der sich über mehrere experimentelle Phasen erstreckte. Diese Historie erklärt die spezifische Wahl der Hyperparameter:

1. **Phase 1: Foundation (PPO v1):** Initiale Experimente mit kleinen Batch-Sizes (64) und einer aggressiven Lernrate ($1 \cdot 10^{-4}$) zeigten, dass der Agent grundsätzlich in der Lage ist, die Physiksprünge zu erlernen, jedoch stark unter hoher Varianz litt.
2. **Phase 2: Exploration (PPO v4):** Um den Zustandsraum maximal zu explorieren, wurde das Netzwerk drastisch vergrößert (Hidden-Dim 1024) und der Entropy-Koeffizient auf $0,01$ erhöht. Dies förderte kreative Lösungsansätze, führte aber zu instabilem Verhalten bei präzisen Landungen. Fachlich deutet dies auf eine Überanpassung (*Overfitting*) an das hochfrequente Rauschen der stochastischen Physik-Updates hin, wodurch die Generalisierungsfähigkeit des Modells auf neue Situationen litt.

3. **Phase 3: Polishing (PPO v5):** In der finalen Phase wurde das Netzwerk auf 512 Neuronen kondensiert und die Batch-Size auf 40.960 erhöht, um das „Rauschen“ der Gradienten zu minimieren. Entscheidend war die Reduktion der Entropie auf $0,0005$, was den Agenten von einem explorativen „Sucher“ zu einem präzisen „Exekutor“ transformierte. Das so entstandene Modell erreichte konsistente Durchschnitts-Scores von ca. 14.300 Punkten.

3.3.5 Curriculum Learning Phasen

Um die Konvergenz in der stochastischen Umgebung sicherzustellen, wurde der Agent durch fünf Schwierigkeitsstufen (Tiers) geführt:

- **Tier 0:** Erlernen stabiler Sprungmuster auf breiten, statischen Basen.
- **Tier 3500:** Einführung horizontaler Plattformbewegungen (Timing-Aspekt).
- **Tier 5500:** Simulation von Windkräften (Kompensation von Drift).
- **Tier 7500:** Plattformverkleinerung zur Steigerung der Landepräzision.
- **Tier 10000:** Dynamische Gegner (aktives Ausweichen).

Dieses Vorgehen verhinderte das Stagnieren in lokalen Minima während der frühen Explorationsphase.

4 Evaluation

4.1 Ergebnisse

Tabelle 1 zeigt die aggregierten Ergebnisse der $N = 10.000$ Simulationsläufe pro Agent.

Agent	**Mean Score**	**Max Score**	**Std Dev** (σ)
A*	**19.030**	**36.475**	4.969
Utility	14.939	28.943	3.787
PPO	14.283	25.415	**3.435**
Reflex	7.507	27.893	5.920

Tabelle 1: Vergleich der Agenten-Performance basierend auf 10.000 Episoden.

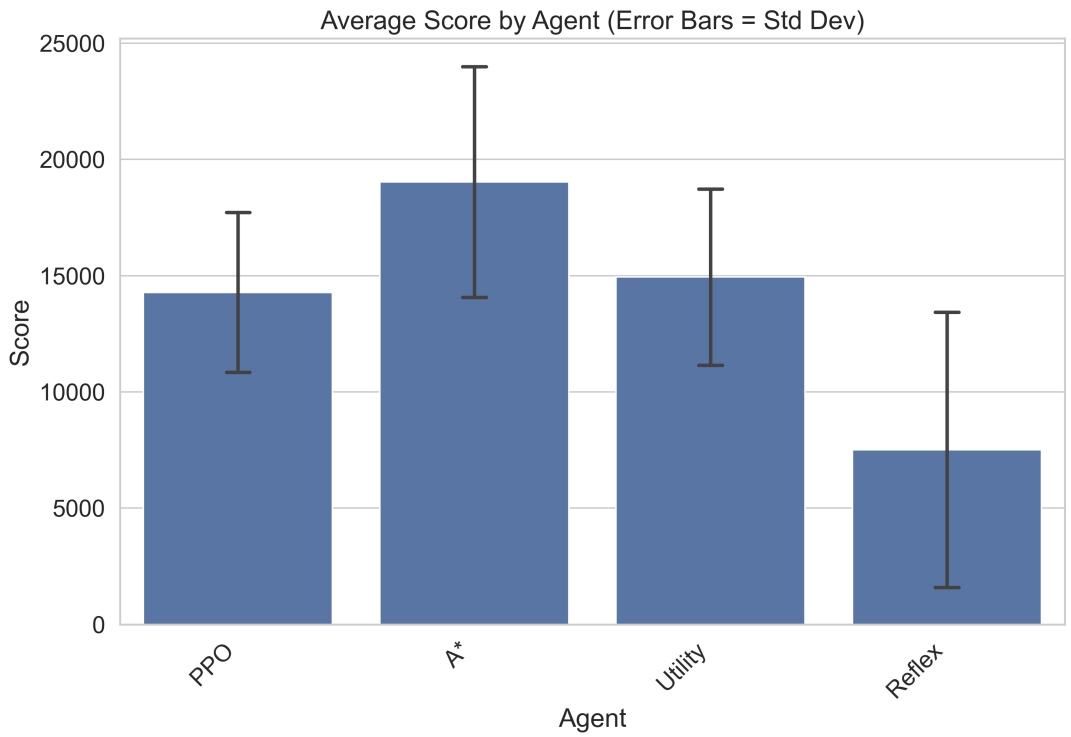

Abbildung 1: Durchschnittliche Scores der Agenten. Die Fehlerbalken visualisieren die Standardabweichung der Ergebnisse.

Abbildung 1 zeigt die durchschnittlich erreichten Scores. Während der A^*-Agent erwartungsgemäß die höchste absolute Performanz erzielt, verdeutlichen die Fehlerbalken (Standardabweichung σ) das zentrale Paradoxon der regelbasierten Steuerung: Trotz hoher Mittelwerte (insbesondere bei A^*) weisen sowohl A^* ($\sigma \approx 4.969$) als auch der Reflex-Agent ($\sigma \approx 5.920$) eine signifikante Ergebnisstreuung auf. Im Gegensatz dazu demonstriert der PPO-Agent ($\sigma \approx 3.435$) ein deutlich engeres Konfidenzintervall, was auf eine stabilere Policy hindeutet.

4.2 Statistische Signifikanz und Verteilung

Die statistische Überlegenheit des A^*-Agenten gegenüber allen anderen Modellen wurde mittels Mann-Whitney-U-Test bestätigt ($p \approx 0$). Dies validiert **Hypothese H1** und unterstreicht den Wert eines präzisen Weltmodells für die Pfadoptimierung.

Die Analyse der Score-Verteilungen in Abbildung 2 offenbart die überlegene Konsistenz des RL-Ansatzes. Das PPO-Modell weist den geringsten Interquartilsabstand (IQR) auf.

Besonders hervorzuheben ist die Lage des unteren Whiskers: Während der Reflex-Agent und selbst die Utility-Heuristik regelmäßig in frühen Spielphasen scheitern (Scores < 1000), liegt das 25%-Quartil von PPO signifikant höher. Dies belegt, dass der Agent durch das Curriculum Learning gelernt hat, kritische „Totalausfälle“ durch robuste Ausweichmanöver fast vollständig zu eliminieren. PPO liefert somit eine verlässliche Mindestperformanz, die über der der klassischen Heuristiken liegt.

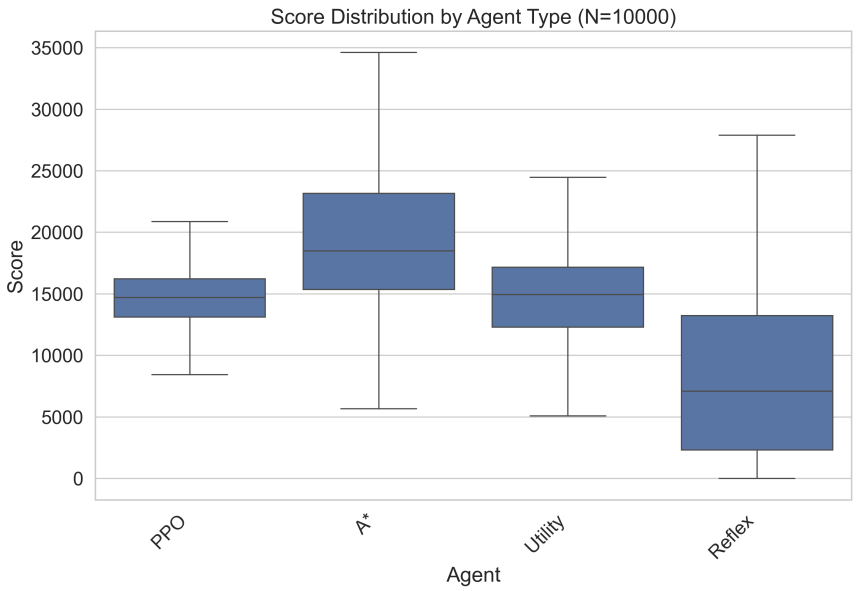

Abbildung 2: Boxplot der Score-Verteilungen für die untersuchten Agenten.

4.3 Dichteanalyse

Die Kernel Density Estimation (KDE) in Abbildung 3 illustriert die unterschiedlichen strategischen Profile. Die Verteilung des PPO-Agenten ist stark kurzsichtig (steilzügig) und konzentriert sich dicht um das Leistungsmaximum des Utility-Agenten. Im Gegensatz dazu ist die Verteilung von A* deutlich flacher (platykurtisch) und rechtsschief. Dies impliziert, dass A* zwar in der Lage ist, außergewöhnliche Spitzenwerte (Outlier) durch perfekte Planung zu erreichen, PPO jedoch eine stabilere, wenn auch im Maximum begrenzte, Verhaltensstrategie („Steady State“) verfolgt.

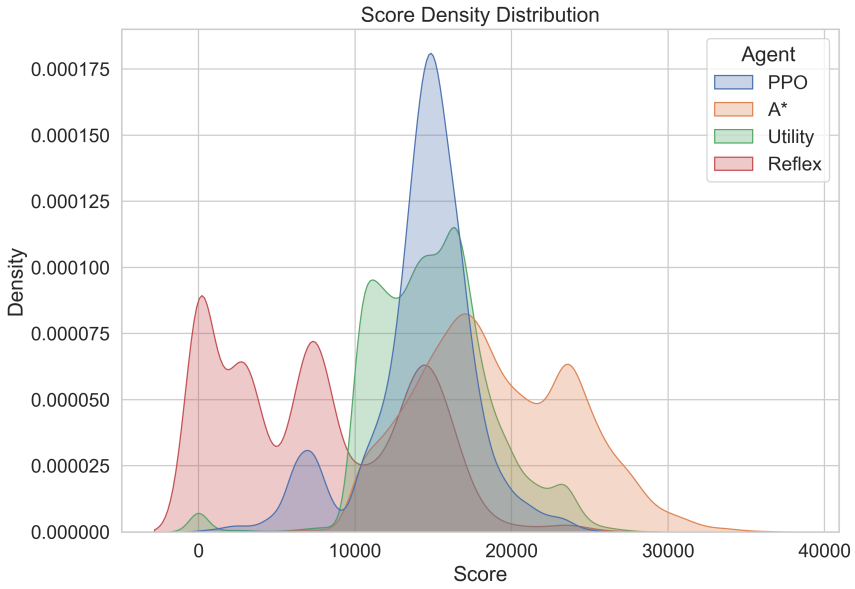

Abbildung 3: Kernel Density Estimation (KDE) der Scores.

4.4 Zuverlässigkeit

Das Überlebensverhalten der Agenten wird durch die empirische Verteilungsfunktion (ECDF) in Abbildung 4 charakterisiert. Die Kurven von A* und Utility folgen einem nahezu linearen Abfall über weite Strecken, was auf eine konstante Fehlerwahrscheinlichkeit hindeutet. Der PPO-Agent zeigt bis zu einem Score von ca. 12.000 eine extrem hohe Überlebenswahrscheinlichkeit, die erst in den späten, hochstochastischen Tier-Bereichen (> 10.000 Höheneinheiten) abnimmt. Im Vergleich zum Reflex-Agenten, dessen Kurve bereits früh exponentiell abfällt, beweist PPO eine deutlich höhere Resilienz gegenüber den kumulativen Störfaktoren wie Wind und Gegnerdichte.

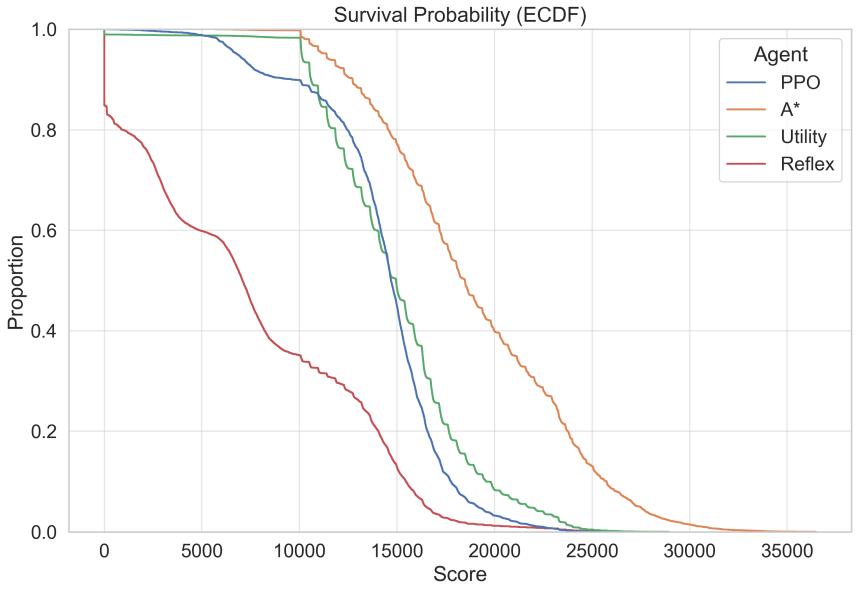

Abbildung 4: Überlebenswahrscheinlichkeit (ECDF) der Agenten im Spielverlauf.

5 Fazit und Ausblick

In dieser Arbeit wurden vier fundamentale Paradigmen der künstlichen Intelligenz in der dynamischen Umgebung *BouncAI* evaluiert. Die Ergebnisse liefern klare Antworten auf die eingangs formulierten Forschungsfragen.

Die Studie bestätigt die Dominanz modellbasierter Suchverfahren (**RQ1**): Der A^*-Agent erzielt durch mathematisch exakte Vorausplanung die höchsten Scores und stellt die theoretische Obergrenze der Performance dar. Dennoch demonstriert der Erfolg des PPO-Modells die enorme Leistungsfähigkeit des Reinforcement Learnings. Ohne explizites Wissen über physikalische Gesetze erreichte der Agent fast 96% der Leistung der Experten-Heuristik, was ihn für Echtzeitanwendungen prädestiniert.

Der wesentliche Beitrag dieser Arbeit liegt in der Identifikation der **strategischen Robustheit** von PPO (**RQ2**). Während klassische Heuristiken und einfache Reflex-Agenten eine hohe Varianz aufweisen und stark von vorteilhaften Umgebungsbedingungen abhängen, liefert PPO ein hochgradig konsistentes Verhalten. Die geringere Standardabweichung kennzeichnet den RL-Agenten als stabilen Ansatz, auch wenn er die absoluten Spitzenwerte der A^*-Suche nicht ganz erreicht.

Für zukünftige Forschungsarbeiten ergeben sich aus diesen Erkenntnissen zwei vielversprechende Richtungen:

- **Imitation Learning:** Die Nutzung von A^*-Trajektorien als Experten-Daten für ein *Behavior Cloning* könnte die frühe Explorationsphase von PPO massiv beschleunigen und die statistische Unschärfe bei der Landung weiter reduzieren.
- **Hybride Architekturen:** Die Kombination einer schnellen PPO-Policy für die motorische Steuerung mit einer gelegentlichen, tieferen A^*-Suche für strategische Entscheidungen (z.B. Routenwahl in Sackgassen) könnte das Beste aus beiden Welten vereinen.

Abschließend lässt sich festhalten, dass RL-Agenten in dynamischen Welten keine bloße Alternative zu klassischen Algorithmen sind, sondern durch ihre Robustheit und Effizienz eine neue Qualität der autonomen Steuerung ermöglichen.

Literatur

[1] Peter E Hart, Nils J Nilsson und Bertram Raphael. „A formal basis for the heuristic determination of minimum cost paths“. In: *IEEE transactions on Systems Science and Cybernetics* 4.2 (1968), S. 100–107.

[2] Julian Togelius, Sergey Karakovskiy und Robin Baumgarten. „The 2009 mario ai competition“. In: *IEEE Congress on Evolutionary Computation*. IEEE. 2010, S. 1–8.

[3] Volodymyr Mnih u. a. „Human-level control through deep reinforcement learning“. In: *nature* 518.7540 (2015), S. 529–533.

[4] John Schulman u. a. „Proximal policy optimization algorithms“. In: *arXiv preprint arXiv:1707.06347* (2017).

[5] Niels Justesen u. a. „Deep learning for video game playing“. In: *IEEE Transactions on Games* 12.1 (2019), S. 1–20.

[6] Yoshua Bengio u. a. „Curriculum learning“. In: *Proceedings of the 26th annual international conference on machine learning*. 2009, S. 41–48.

[7] Jimmy Lei Ba, Jamie Ryan Kiros und Geoffrey E Hinton. „Layer normalization“. In: *arXiv preprint arXiv:1607.06450* (2016).