Duale Hochschule Baden-Württemberg Mannheim

Ausarbeitung

Flappy Bird Game

Studiengang Wirtschaftsinformatik

Studienrichtung Data Science

Verfasser(in):	Ali Moutyrek, An-Phi Dang, Tobias Ludwig, Katharina Thiel
Matrikelnummer:	7507540, 7286643, 7558992, 5910231
Kurs:	WWI21DSA
Bearbeitungszeitraum:	08.05.2024 - 10.07.2024
Github-Repo:	https://github.com/TobiasLudwig02/RLFlappyBird

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

A2C	Advantage Actor-Critic
DQN	Deep Q-Learning
KI	Künstliche Intelligenz
MLP	Multiple Layer Perceptron
PPO	Proximal policy optimization
RL	Reinforcement Learning
TRPO	Trust Region Policy Optimization

1 Einleitung

Flappy Bird ist ein einfaches, aber äußerst beliebtes Spiel, das von dem vietnamesischen Entwickler Dong Nguyen entwickelt wurde. In diesem Spiel steuert der Spieler einen kleinen Vogel, der durch Tippen auf den Bildschirm zwischen Pipes navigieren muss, ohne diese zu berühren [1]. Das Ziel des Spiels ist es, den Vogel so lange wie möglich am Leben zu halten und dabei möglichst viele Pipes zu durchfliegen [1]. Trotz seiner einfachen Spielmechanik erfordert Flappy Bird ein hohes Maß an Geschicklichkeit und Reaktionsvermögen.
Dieses Environment bietet sich gut für das Trainieren eines Reinforcement Learning (RL) Agents an, welcher durch Interaktionen mit seiner Umgebung optimale Entscheidungen treffen soll, um eine maximale Belohnung zu erhalten [1]. Dabei kommt eine Rewardfunktion zum Einsatz, bei dem der Agent positive Rückmeldungen für erfolgreiche Aktionen und negative Rückmeldungen für Fehler erhält [2].
Das Ziel dieses Projekts ist es, verschiedene Modelle auf einem Flappy Bird Environment zu trainieren und zu evaluieren, welches der Modelle am besten für diesen Vorgang geeignet ist. Dafür werden verschiedene Modelle trainiert, sowie verschiedene Ansätze des Trainings verwendet und deren Ergebnisse miteinander verglichen.

2 Grundlagen

Um die optimale Methode für das Erlernen von Flappy Bird zu bestimmen, werden verschiedene RL-Modelle trainiert. Diese sind: Q-Learning, Deep Q-Learning, Advantage Actor-Critic und Proximal Policy Optimization. Um deren Güte korrekt vergleichen zu können, werden diese mit einem heuristischen Ansatz, das Baselinemodell, verglichen.

2.1 Q-Learning und Deep Q-Learning

Q-Learning ist ein zentraler Algorithmus im Reinforcement Learning, der Agenten optimale Strategien erlernen lässt, indem er erwartete Belohnungen für bestimmte Aktionen in bestimmten Zuständen schätzt [3]. Das Ziel ist es, eine Strategie zu entwickeln, die die kumulierte zukünftige Belohnung maximiert.
Deep Q-Learning (DQN) kombiniert Q-Learning mit tiefen neuronalen Netzwerken [4].
DQN kann direkt aus Rohdaten wie Pixeln eines Spiels lernen [5]. Während des Trainings werden die Netzwerkgewichte durch Backpropagation aktualisiert, um die Differenz zwischen vorhergesagten Q-Werten und tatsächlichen Belohnungen zu minimieren.
Die Eingabe des Emulators wird durch das Neuronalen Netzt approximiert, um die Bellmann-Gleichung lösen zu können (vgl. S. 3 [6]). Als Greedy- Strategie wird in diesem Fall 1 - ϵ verwendet.

2.2 Advantage Actor-Critic

Advantage Actor-Critic (A2C) ist eine fortschrittliche Methode im Bereich des Reinforcement Learnings, die sowohl die Vorteile von Policy-Gradient-Methoden als auch die von Value-Based-Methoden kombiniert. Der Hauptunterschied zu anderen Modellen besteht darin, dass A2C sowohl einen „Actor“ als auch einen „Critic“ verwendet. Der Actor aktualisiert die Policy, also die Strategie, wie Aktionen ausgeführt werden, basierend auf den von ihm geschätzten Gradienten. Der Critic bewertet die Aktionen des Actors und liefert

Feedback, welches zur Verbesserung der Policy beiträgt. Diese Arbeitsteilung führt zu stabileren und effizienteren Lernprozessen, da der Critic dem Actor dabei hilft, genauere und weniger variierende Gradienten zu berechnen. [7], [8]
Technisch gesehen beginnt der A2C-Algorithmus mit der Interaktion des Actors mit der Umgebung, wobei Aktionen zu Belohnungen und neuen Zuständen führen. Diese Daten werden im Rollout Buffer gespeichert. A2C ist eine synchrone, deterministische Variante des Asynchronous Advantage Actor Critic und verwendet mehrere Worker, um den Einsatz eines Replay Buffers zu vermeiden. Der Advantage wird basierend auf gesammelten Belohnungen und vom Critic geschätzten Werten berechnet, oft verfeinert durch Generalized Advantage Estimation. Die Policy und Wertfunktion werden durch mehrere Verlustberechnungen aktualisiert:

- **Policy-Loss**: Berechnet durch Multiplikation des Advantages mit der Wahrscheinlichkeit der ausgeführten Aktionen, um die Wahrscheinlichkeit von Aktionen mit hohem Advantage zu maximieren.
- **Value-Loss**: Minimiert die Differenz zwischen geschätzten und tatsächlichen Werten.
- **Entropie-Loss**: Maximiert die Entropie der Policy, um Exploration zu fördern.

Die Gradienten dieser Verluste werden berechnet und die Parameter der Netzwerke durch Gradienten-Clipping aktualisiert, um instabile Updates zu vermeiden. Im Trainingsprozess wird die Policy mit den im Rollout Buffer gesammelten Daten aktualisiert, was das Berechnen der Verluste und das Aktualisieren der Netzparameter umfasst. Der Lernprozess beinhaltet mehrere Trainingszyklen, wobei verschiedene Statistiken wie erklärbare Varianz oder Verluste aufgezeichnet werden. [9]

2.3 Proximal Policy Optimization

Proximal Policy Optimization (PPO) wurde von OpenAI entwickelt und zeichnet sich durch einfache Implementierung und hohe Effizienz aus [10]. PPO kombiniert die Stabilität von Trust Region Policy Optimization (TRPO) mit der Einfachheit und Effizienz von Ordnungsgradienten-Methoden. Dies wird durch die Einführung eines sogenannten „clipped surrogate objective“ erreicht, welches sicherstellt, dass die neue Policy nicht zu weit von der alten

Policy abweicht. Dies minimiert das Risiko von drastischen Leistungsabfällen nach einem Update. [11], [12]
Technisch besteht PPO aus mehreren Schritten:

- Festlegen einer Surrogate-Zielfunktion mit zufälligen Anfangsparametern, unter Verwendung der „clipped surrogate objective“.
- Sammeln von Daten durch Erfahrungen in der Umgebung mithilfe des „Rolloutbuffer“.
- Berechnen eines Advantage-Schätzers, um zu bestimmen, ob eine Aktion sinnvoll ist, wobei Normalisierung angewendet wird.
- Bilden der Surrogate-Zielfunktion über die gesammelten Erfahrungen als Erwartungswert unter Nutzung von Batch-Processing.
- Anwenden des stochastischen Gradienten Aufstiegs, um die verschiedenen Parameter zu aktualisieren.
- Durchführen mehrerer Epochen dieser Optimierungsschritte, im Gegensatz zu einfachen Policy-Gradienten-Methoden. [13]

Die Effektivität von PPO wurde in Studien an komplexen Spielen wie Dota 2 und Atari-Spielen nachgewiesen [14]. Diese Studien zeigen, dass PPO hohe Punktzahlen erzielen und eine stabile, vorhersehbare Leistung während des Trainings aufrechterhalten kann. Dies ist ein Vorteil gegenüber Algorithmen wie Q-Learning und DQN, die oft unter instabilen Lernprozessen leiden [15].

2.4 Das Gymnasium Flappy Bird Environment

Für die Entwicklung und Evaluierung von RL-Modellen sind standardisierte Testumgebungen von großer Bedeutung. Eine solche Environment bietet das Flappy Bird Environment von Gymnasium.[16] Dieses Environment bietet eine effiziente Möglichkeit, RL-Modelle zu trainieren und zu testen. Im Folgenden wird die Gymnasium-Environment für Flappy Bird im Detail erläutert, wobei ein besonderer Fokus auf die Belohnungsstruktur, die Parameter des Environments und die spezifischen Anpassungen gelegt wird.
In der Gymnasium Flappy Bird Environment erhält der Agent Belohnungen entsprechend seinem Fortschritt im Spiel. Eine positive Belohnung wird für jeden Frame vergeben, in

dem der Agent überlebt (+0,1 Punkte). Darüber hinaus erhält der Agent eine Belohnung von +1 Punkt für das erfolgreiche Passieren einer Pipe. Negative Belohnungen werden erteilt, wenn der Agent auf eine Pipe trifft oder den Boden berührt (-1,0 Punkte), sowie wenn er die Oberseite des Bildschirms berührt (-0,5 Punkte). Diese Struktur motiviert den Agenten, Pipes zu durchfliegen und möglichst lange im Spiel zu bleiben. [16]
Die Gymnasium-Environment für Flappy Bird verwendet insgesamt 12 Parameter, um den Zustand des Spiels zu beschreiben. Zu diesen Parametern gehören die x- und y-Position des Vogels, seine aktuelle Geschwindigkeit, die horizontale Entfernung zur nächsten Pipe sowie die y-Positionen der Ober- und Unterseiten der nächsten beiden Pipes. Weitere Parameter umfassen den aktuellen Punktestand, den Lebensstatus des Vogels, die letzte vom Agenten ausgeführte Aktion und die aktuelle Zeitstufe im Spiel. Diese Parameter bieten dem Agenten eine umfassende Sicht auf das Spielfeld und seine aktuelle Situation, was für die Entscheidungsfindung im RL-Training entscheidend ist. [16]

2.5 Erstellen eines eigenen Environments

Gegenüber der Gymnasium Flappy Bird Environment wird ein weiteres, eigenentwickeltes Environment verwendet. Diese Umgebung mit wird Mithilfe des Gymnasium Paketes erstellt, ist optisch schlanker als das Gymnasium Flappy Bird Environment und stellt die Pipes und den Vogel mit der Library Pygames dar [17].
Die Umgebung nimmt während des Spielens folgende Parameter an: Die x-Position des Vogels, die y-Position des Vogels, die Position der unteren Pipe und die Position der oberen Pipe. Dieses Environment ermöglicht es, die Algorithmen mit statischen Pipes (die Pipes befinden sich in jeder Episode an derselben Position) und variablen Pipes (mit einem Random Seed wird die Position der Pipes zufällig festgelegt) zu trainieren, sowie die Anzahl der durchflogenen Pipes wiederzugeben.

3 Modelle und Ergebnisse

Dieses Kapitel beschreibt die verwendeten Methoden sowie die daraus erzeugten Ergebnisse.
Es wird die Algorithmen-Sammlung Stable Baselines3 verwendet, welche sich durch Konsistenz, Zuverlässigkeit und Flexibilität zur Unterstützung von benutzerdefinierten Umgebungen und Algorithmen sowie Kompatibilität mit beliebten Bibliotheken wie OpenAI Gymnasium auszeichnet [18].

3.1 Baseline

Die Baseline ist folgender heuristischer Ansatz: Immer, wenn der Vogel unter eine bestimmte Höhe fällt (hier: 0,4), flattert er, um über dieser zu bleiben. Der Grund hierfür ist, dass das Gymnasium Environment eine Höhe von 1 hat, wodurch 0.4 knapp unter der Mitte liegt und der Vogel somit stetig in der Mitte flattert. Technisch wird die vertikale Position des Vogels, welches das Environment zurück gibt [16], dauerhaft mit dem definierten Schwellenwert verglichen und sollte der Vogel sich unterhalb dieses Schwellenwerts befinden, wird eine 1 für die Aktion des Flatterns zurückgegeben.
Dieser Ansatz dient als Vergleichsmaßstab, um die Effektivität und Leistungsfähigkeit der fortgeschrittenen Lernmethoden zu bewerten.

3.2 Testen des eigen erstellten Environment

Auf dem eigen erstellten Environment wird ein Q-Learning Algorithmus angewendet. Der Zustandsraum des hier verwendeten Q-Learning Agenten besteht aus 4 Dimensionen mit je 10 Intervallen, was 10.000 diskreten Zuständen entspricht. Bei dem Vergleich zwischen den Environments mit den statischen Pipes und den dynamischen Pipes fällt auf, dass der Algorithmus nur bei den statischen Rohren in der Lage ist, mehrere Pipes zu passieren. Der Grund hierfür ist, dass die Off-Policy Strategie des Q-Learnings bei zufälligen Pipe Positionen nicht in der Lage ist zu konvergieren und somit keine effiziente Strategie entwickeln kann. Dies deutet darauf hin, dass die in [2] erwähnte Velocity des Vogels für den

Algorithmus wichtig ist, um das Spiel zu erlernen.
Der Ansatz des Q-Learnings wurde aufgrund dieser Erkenntnis für das weitere Training verworfen. Um zu testen, ob diese Environment weiterverwendet werden sollte, wird zusätzlich ein DQN-Algorithmus angewendet. Da auch dieser keine überzeugenden Ergebnisse liefert (bei 2.000.000 Episoden ein Reward von durchschnittlich 1,8. Der Reward ist festgelegt als kontinuierlich +0,1, +1 für das Passieren einer Pipe und -1 fürs Sterben) deutet das auf ein unzureichendes Environment hin. Da das Gymnasium Environment mehr Parameter zurückgibt, kann davon ausgegangen werden, dass es für das Trainieren der Modelle besser geeignet ist und wird daher im Folgenden, anstatt des eigenen Environments verwendet.[16]

3.3 Auswahl der Modelle für das Gymnasium Flappy Bird Environment

Für die Implementierung der Gymnasium Environment wird eine angepasste Version des Gymnasium Flappy Bird Environments verwendet, welche im weiteren Verlauf angepasst wird.
Um verschiedene Modelle auf diesem zu testen, wird auf drei verschiedene Reinforcement Learning Modelle zurückgegriffen. Im Folgenden wird die jeweilige Auswahl erläutert.

Mit einem **PPO** Modell wird eine stabile und zuverlässige Lernleistung in verschiedenen Aufgabenbereichen des Reinforcement Learnings geboten. Dieses Modell kombiniert die Vorteile von Trust Region Policy Optimization (TRPO) und bietet gleichzeitig eine einfachere Implementierung [13], was zu stabilen und zuverlässigen Ergebnissen führt. Durch die Nutzung von Surrogate-Loss-Funktionen kann PPO effiziente und stabile Updates durchführen, was besonders für die dynamische und unstetige Umgebung von Flappy Bird vorteilhaft ist. PPO hat sich in vielen RL-Anwendungen als robust und effizient erwiesen, was seine Wahl für diese Aufgabe rechtfertigt [13].
DQN ist ein bewährtes Modell für RL, die sich durch ihre Fähigkeit auszeichnet, diskrete Aktionen in hochdimensionalen Zustandsräumen zu bewältigen [19]. Flappy Bird stellt eine Herausforderung für das DQN dar, da der Agent nur zwischen diskreten Aktionen (Flügelbewegung oder keine Bewegung) entscheiden muss. Dieses Netz ist nicht die bevorzugte Wahl für das Flappy Bird Environment. Es wird jedoch als Vergleich verwendet, da es eines der stärkeren vorhandenen Modelle des Reinforcement Learnings ist [19].

A2C ist ein weiteres populäres RL-Modell, das durch die Kombination von Policy- und Value-Function-Learning robuste Lernprozesse ermöglicht [20]. Der Vorteil von A2C liegt in seiner Fähigkeit, sowohl die Policy des Agenten als auch die Wertfunktion zu optimieren, was zu effizienten und schnellen Lernprozessen führt. Diese Eigenschaft ist besonders wichtig für das Spiel Flappy Bird, das schnelle Reaktionszeiten und präzise Entscheidungen erfordert [20].
Die Wahl der `MlpPolicy` als zugrunde liegendes Netzwerk für alle Modelle basiert auf ihrer Flexibilität und Fähigkeit, komplexe nichtlineare Funktionen zu approximieren. Multiple Layer Perceptrons (MLPs) sind in der Lage, verschiedene Eingabemuster effektiv zu verarbeiten und komplexe Abbildungen zu lernen, was für die dynamische Umgebung von Flappy Bird unerlässlich ist [21]. Darüber hinaus ermöglicht die `MlpPolicy` eine einfache Implementierung und Integration in verschiedene RL-Algorithmen, was sie zu einer idealen Wahl für diese Studie macht [21].

3.4 Ergebnisse

Im Folgenden werden die Ergebnisse der zuvor ausgewählten und erklärten Modelle dargestellt und erläutert. Das Vergleichsmaß ist hier der Reward, der zu unterschiedlichen Timestamps erzielt wird und wie zuverlässig dieser erzielt wird.

3.4.1 Modelle auf dem Gymnasium Flappy Bird Environment

Die vorher beschriebenen Modelle werden über jeweils zwei Millionen Timestamps trainiert. Diese Angabe wird gewählt, da in dieser Zeit eine deutliche Steigerung zu erkennen ist und das Training der Modelle in vertretbaren Zeit abgeschlossen werden kann. Der Reward ist folgendermaßen gesetzt: +1 für das Passieren einer Pipe, -1 für das Berühren des Bodens, -0,5 für das Berühren des oberen Randes und ein kontinuierlicher Reward von +0,1.
Die Baseline zeigt erwartungsgemäß keine signifikante Verbesserung des Rewards, sondern bleibt auf einem konstant niedrigen Niveau. Siehe Abbildung 3.1.
PPO zeigt eine signifikante Verbesserung des durchschnittlichen Rewards über die Zeit. Dieses Verhalten ist auf die effiziente und stabile Policy-Optimierung zurückzuführen, die durch den Einsatz von Surrogate-Loss-Funktionen erreicht wird [13]. Auffällig sind jedoch

die hohen Schwankungen, welche eigentlich durch das „clipped surrogate objectiv“ verhindert werden sollten.

Abbildung 3.1: Der Reward der Modelle PPO, DQN, A2C und Baseline ist auf der Gymnasium Flappy Bird Environment über 2 Mio. Timestamps dargestellt. Das am besten abschneidende Modell ist das PPO Modell, welches zeitweise einen Reward von über 60 erreicht, dabei aber stark schwankende Ergebnisse erzielt. Im Gegensatz dazu erreichen die Modelle DQN und Baseline kaum einen Reward von 10, bevor sie konvergieren. Das A2C Modell erreicht zeitweise einen Reward von 20 und ist nach den Timestamps noch nicht konvergiert.

Das DQN-Modell weist eine langsame, aber konstante Verbesserung des Rewards auf. Dies ist typisch für DQN, da es darauf angewiesen ist, Q-Werte für jede Aktion in jedem Zustand zu lernen, was bei komplexen Umgebungen wie Flappy Bird zeitaufwendig ist [19]. Die Leistung von DQN wird durch seine Fähigkeit begrenzt, langfristige Abhängigkeiten und Strategien effektiv zu lernen, was in der Flappy Bird Umgebung zu einer geringen Lernrate führt.
Das A2C-Modell zeigt eine Verbesserung des Rewards, jedoch langsamer als PPO. Dies liegt daran, dass A2C sowohl die Policy- als auch die Wertfunktion gleichzeitig optimiert, was zu einem im Vergleich zu PPO weniger effizienten Lernprozess führt, jedoch eine

bessere Stabilität bietet.

3.4.2 Modelle auf dem Gymnasium Flappy Bird Environment mit verändertem Reward

In diesem Teil wird die Gymnasium Environment mit einer neuen Rewardfunktion trainiert. Diese sieht folgendermaßen aus:

- Pipe passed = 100
- Gegen Decke = -5
- Gegen Pipe = -10
- Am Leben = 1

Im Vergleich zur Standard-Rewardfunktion wird der Reward für das Überwinden einer Pipe mal Hundert genommen, um den Agent zu „motivieren“ mehr durch die Pipes zu fliegen. Die weiteren Punkte werden jeweils verzehnfacht, um diese an den Reward für das Schaffen einer Pipe ein wenig anzunähern.

Das PPO-Modell erreicht, in Abbildung 3.2 zu sehen, schnell einen hohen Reward von bis zu 800. Die starken Schwankungen die es zuvor im durchschnittlichen Reward gab, treten hier weniger, bzw. später, auf. Obwohl die Belohnung für das Passieren von Pipes im Vergleich zum Sterben jetzt deutlich stärker belohnt wird, werden jetzt weniger Pipes überquert.
Das DQN-Modell zeigt eine moderate Steigerung des Rewards, jedoch langsamer als PPO. DQN benötigt mehr Zeit, um die langfristigen Belohnungen effektiv zu lernen und entsprechend zu handeln [19]. Genau wie vorher konvergiert der erhaltene Reward schnell.

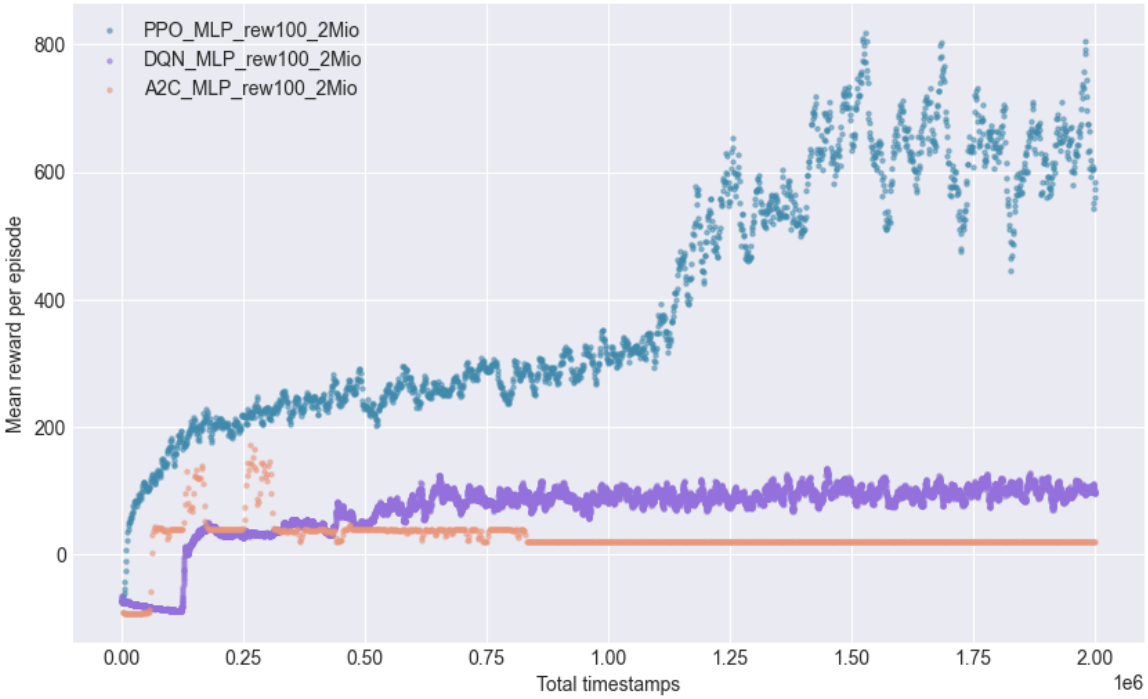

Abbildung 3.2: PPO, DQN und A2C über 2 Mio. Timestamps mit angepasster Rewardfunktion. PPO schneidet am besten ab, wobei es einen Reward von bis zu 800 erreicht. A2C schneidet am schlechtesten ab, bei einem Reward, der zwar am Anfang bis zu 200 erreicht, danach aber auf 0 sinkt. Der Algorithmus A2C lernt also keine effektive Strategie, um das Spiel zu spielen. DQN erreicht die 100 Punkte nach ungefähr 500.000 Timestamps. Genauso wie A2C wird das Modell danach nicht mehr besser. Die Anpassung der Rewardfunktion hat dazu geführt, dass die Modelle in derselben Trainingszeit schlechter sind.

Das A2C-Modell zeigt anfänglich eine schnelle Verbesserung, stabilisiert sich jedoch später auf einem niedrigen Plateau. Dies liegt daran, dass A2C sowohl die Policy als auch die Wertfunktion gleichzeitig optimiert, was zu einem schnelleren Lernen in der Anfangsphase führt, aber möglicherweise nicht die gleiche Langzeitstabilität wie PPO bietet [20]. Die veränderte Rewardfunktion scheint A2C daran zu hindern, wie zuvor stetig weiter zulernen. Auch bei der veränderten Rewardfunktion ist PPO das effektivste Modell.

3.4.3 Curriculum Learning

Curriculum Learning beinhaltet die schrittweise Anpassung der Schwierigkeit der Aufgabe. Bei diesem Vorgehen wurde der Abstand zwischen den Pipes alle 500.000 Trainingsschritte verringert:

- Erste 500.000 Schritte: Abstand = 200
- Nächste 500.000 Schritte: Abstand = 150
- Weitere 500.000 Schritte: Abstand = 125
- Letzte 500.000 Schritte: Abstand = 100 (Standardwert)

Diese Technik soll es dem Agenten ermöglichen, sich zuerst auf einfachere Aufgaben zu konzentrieren und dann allmählich komplexere Herausforderungen zu bewältigen.

Abbildung 3.3: Vergleich von zwei PPO Modellen, bei denen eines das Curriculum Learning anwendet. Das Modell mit Curriculum Learning erzielt zu Beginn höhere Rewards, da die Pipes einfach durchquert werden können. Nach 1 Mio. Timestamps liegt es jedoch auf dem Niveau des anderen Modells. Beide Modelle erzielen keine stabilen Rewards, sondern haben Schwankungen von 50 Punkten.

Wie in Abbildung 3.3 dargestellt, zeigt sas PPO-Modell ohne Curriculum Learning eine stetige, aber relativ langsame Verbesserung des durchschnittlichen Rewards über die Zeit. Dies ist auf die konstante Schwierigkeit des Spiels zurückzuführen, die es dem Agenten erschwert, schnell effektive Strategien zu entwickeln [13]. Siehe
Das PPO-Modell mit Curriculum Learning zeigt eine deutlich schnellere Verbesserung des durchschnittlichen Rewards zu Beginn des Trainings. Dies liegt daran, dass der Agent zunächst auf einfacheren Aufgaben (größerer Abstand zwischen den Pipes) trainiert wird, was es ihm anfangs ermöglicht, grundlegende Fähigkeiten schneller zu erlernen [22].
Die schrittweise Erhöhung der Schwierigkeit sorgt dafür, dass der Agent seine Strategien kontinuierlich anpasst und verbessert. Die Schwankungen im mittleren Reward während des Trainings können auf die Anpassungsphasen des Agenten zurückgeführt werden, wenn die Schwierigkeit erhöht wird. Nach den 2 Mio. Timestamps haben beide Modelle ein ähnliches Niveau erreicht. Schlussendlich wird kein signifikant besseres Ergebnis durch das Curriculum Learning erzielt.

4 Zusammenfassung und Ausblick

In dieser Arbeit wurden verschiedene Ansätze für das Trainieren von RL-Modellen getestet. Am schlechtesten hat das DQN Modell abgeschnitten, da hier der durchschnittliche Reward mit der Zeit stagniert. Das A2C Modell hat bessere Ergebnisse erzielt, ist jedoch stark von der Rewardfunktion abhängig. Es ist anzumerken, dass es die von allen Modellen stabilsten Ergebnisse liefert. Das PPO-Modell hat die besten Ergebnisse erzielt. Dies liegt an der Policy-Optimierung mittels Surrogate-Loss-Funktion, die unabhängig von der Belohnungsfunktion zu hervorragenden Resultaten führt. Zudem zeigt sich, dass ein Environment, dass möglichst viele Beobachtungen an das Modell zurückgibt, entscheidend für das erfolgreiche Training von Modellen ist.
Für die weitere Forschung wird empfohlen, PPO-Modelle über eine größere Anzahl von Timestamps zu trainieren, um die Strategie des Modells weiter zu verbessern. Es ist außerdem ratsam, ein Hyperparametertuning bei der Methode DQN durchzuführen. Für das A2C-Modell kann mithilfe einer größeren Anzahl von Timestamps noch eine bessere Strategie für das Spiel erlernt werden.

Literaturverzeichnis

[1] S. Wei, „Reinforcement Learning for Improving Flappy Bird Game," *Highlights in Science, Engineering and Technology*, Jg. 34, S. 244–249, Feb. 2023. DOI: 10.54097/hset.v34i.5479.

[2] Z. He, Y. Zhang und D. Zhao, „Flappy Bird Game Based on Reinforcement Learning Q-Learning Algorithm," *Highlights in Science, Engineering and Technology*, Jg. 34, S. 222–225, Feb. 2023. DOI: 10.54097/hset.v34i.5475.

[3] E. Duryea, M. Ganger und W. Hu, „Exploring Deep Reinforcement Learning with Multi Q-Learning," *Intelligent control and automation*, Jg. 07, Nr. 04, S. 129–144, Jan. 2016. DOI: 10.4236/ica.2016.74012. Adresse: https://doi.org/10.4236/ica.2016.74012.

[4] Y. Guo, „Enhancing Flappy Bird Performance With Q-Learning and DQN Strategies," *Highlights in Science, Engineering and Technology*, Jg. 85, S. 396–402, März 2024. DOI: 10.54097/qrded191.

[5] R. Rosalina, A. Sengkey, G. Sahuri und R. Mandala, „Generating intelligent agent behaviors in multi-agent game AI using deep reinforcement learning algorithm," *International Journal of Advances in Applied Sciences*, Jg. 12, S. 396, Dez. 2023. DOI: 10.11591/ijaas.v12.i4.pp396-404.

[6] V. Mnih, K. Kavukcuoglu, D. Silver et al., *Playing atari with deep reinforcement learning*, 19. Dez. 2013. arXiv: 1312.5602[cs]. Adresse: http://arxiv.org/abs/1312.5602 (besucht am 10. 07. 2024).

[7] I. Grondman, L. Busoniu, G. A. D. Lopes und R. Babuska, „A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, Jg. 42, Nr. 6, S. 1291–1307, 2012. DOI: 10.1109/TSMCC.2012.2218595.

[8] Y. Xiao, X. Lyu und C. Amato, „Local Advantage Actor-Critic for Robust Multi-Agent Deep Reinforcement Learning," in *2021 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*, 2021, S. 155–163. DOI: 10.1109/MRS50823.2021.9620607.

[9] D. „stable-baselines3/stable$_baselines3/a2c/a2c.pyatmasterDLR-RM/stable-baselines3$." (), Adresse: https://github.com/DLR-RM/stable-baselines3/blob/master/stable_baselines3/a2c/a2c.py (besucht am 10. 07. 2024).

[10] S. Yang, M. Barlow, T. Townsend et al., „Reinforcement Learning Agents Playing Ticket to Ride–A Complex Imperfect Information Board Game With Delayed Rewards,“ *IEEE Access*, Jg. 11, S. 60 737–60 757, 2023. DOI: 10.1109/ACCESS.2023.3287100.

[11] „Proximal Policy Optimization — Spinning Up documentation.“ (), Adresse: https://spinningup.openai.com/en/latest/algorithms/ppo.html (besucht am 09. 07. 2024).

[12] J. Schulman, F. Wolski, P. Dhariwal, A. Radford und O. Klimov, *Proximal Policy Optimization Algorithms*, 2017. arXiv: 1707.06347 [cs.LG]. Adresse: https://arxiv.org/abs/1707.06347.

[13] J. Schulman, F. Wolski, P. Dhariwal, A. Radford und O. Klimov, „Proximal Policy Optimization Algorithms,“ *arXiv preprint arXiv:1707.06347*, 2017.

[14] J. Ristovska und D. Šoberl, „Human-assisted reinforcement learning demonstrated on the Flappy Bird Game,“ Okt. 2023. DOI: 10.26493/scores23.13.

[15] Y. Guo, „Enhancing Flappy Bird Performance With Q-Learning and DQN Strategies,“ *Highlights in Science, Engineering and Technology*, Jg. 85, S. 396–402, März 2024. DOI: 10.54097/qrded191.

[16] „flappy-bird-gymnasium.“ (29. Feb. 2024), Adresse: https://pypi.org/project/flappy-bird-gymnasium/.

[17] „Pygame Front Page — pygame v2.6.0 documentation.“ (), Adresse: https://www.pygame.org/docs/ (besucht am 10. 07. 2024).

[18] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto und N. Dormann, *Stable Baselines: A Fork of OpenAI Baselines, Implementing Reinforcement Learning Algorithms*, Accessed: 2024-07-10, 2018. Adresse: https://stable-baselines.readthedocs.io/en/master/index.html.

[19] V. Mnih, K. Kavukcuoglu, D. Silver et al., „Human-level control through deep reinforcement learning,“ *Nature*, Jg. 518, S. 529–533, 2015. Adresse: https://api.semanticscholar.org/CorpusID:205242740.

[20] V. Mnih, A. P. Badia, M. Mirza et al., „Asynchronous methods for deep reinforcement learning,“ *International conference on machine learning*, S. 1928–1937, 2016.

[21] Y. LeCun, Y. Bengio und G. Hinton, „Deep learning,“ *Nature*, Jg. 521, Nr. 7553, S. 436–444, 2015.

[22] Y. Bengio, J. Louradour, R. Collobert und J. Weston, „Curriculum learning,“ *Proceedings of the 26th annual international conference on machine learning*, S. 41–48, 2009.