Balatro-KI

Jul Lang[a*†] , Dan Lut[a‡†]

[a]

1. Thema

1.1. Einleitung

Balatro ist ein pokerinspiriertes Roguelike-Strategiespiel des Indie-Entwicklers LocalThunk. Trotz seines limitierten Budgets wurde es innerhalb von kürzester Zeit ein Überraschungshit auf Steam. Auch wir entwickelten schnell eine Leidenschaft, und haben mittlerweile beide je um die 250 Stunden in das Spiel investiert. Als sich also bei diesem Projekt die Möglichkeit bot uns mehr damit zu beschäftigen, mussten wir diese Gelegenheit nutzen. So entstand die Idee, eine KI zu entwickeln, die das Spiel eigenständig spielen (und hoffentlich meistern) kann.

1.2. Fragestellung

Eine natürliche Option für die Fragestellung wäre, ob wir es schaffen, eine KI zu trainieren, die das Spiel schlägt, was bei Balatro heisst, den "Ante" (ein Zusammenschluss aus drei etwas unterschiedlichen Runden) 8 zu schlagen, auch wenn danach noch ohne Einschränkungen weiterspielen kann. Andererseits gibt es auch einige andere Werte, die man in Balatro versuchen könnte zu maximieren, zum Beispiel das Geld, den Score, oder die Anzahl gewonnene Runden. Wir haben uns für letzteres entschieden, da es weniger "alles oder nichts" ist als die erste Option, aber dennoch den zentralen Aspekt des Spieles verfolgt, im Gegensatz zu Geld und Score, die eher ein Mittel zum Zweck sind. Somit lautet unsere zentrale Fragestellung: **"Ist es möglich eine Balatro KI zu programmieren? Wie viele Runden schafft sie innerhalb von dieser limitierten Zeit?"** Nichtsdestotrotz sind wir natürlich auch sehr interessiert an den Taktiken der KI, welche Wechselwirkungen entdeckt werden und welche Aspekte des Spiels die KI meistern wird.

2. Planungsphase

2.1. Deep Reinforcement Learning: Geschichte

Historisch hatte die KI-Trainingsmethode "Deep Reinforcement Learning" grosse Erfolge im Bereich der Spiel-KIs, so konnte z.B. die KI AlphaGo von DeepMind im Jahre 2016 erstmals den Go Weltmeister schlagen! [1]
Auch wenn Deep-RL zwischenzeitlich etwas in den Hintergrund rückte, hat es momentan eine kleine Renaissance und blüht nach grossem Erfolg in LLMs wie Deepseek R1 [2] wieder auf.

2.2. Typus der KI

Da es sich bei Balatro dank seiner vielfältigen Interaktionen zwischen Spielkarten, Modifikationen, Jokers usw. um ein ähnlich komplexes Spiel wie Go handelt, scheint uns die Trainingsmethode die dieses meisterte eine gute Wahl, spezifisch eine **Online Deep Reinforcement Learning KI** [3], da es keinen Datensatz von Spielen gibt, den die KI imitieren könnte um "offline" zu trainieren.
Es beginnt damit, dass der Agent den Spielzustand erhält indem er sich befindet. Daraufhin wählt der Agent eine Handlung, z.B. eine Pokerhand spielen, welche sich auf die Umgebung, das Spiel, auswirkt und deren Zustand verändert. Hat die Handlung nach manuell gewählten Kriterien einen guten Effekt auf den Spielzustand, so erhält die KI eine Belohnung. Dieser Prozess wiederholt sich

* Corresponding author. Address: . Email: jullanggit@proton.me
† These authors contributed equally to this work.
‡ Corresponding author. Address: . Email: zadowcloud.0@gmail.com

kontinuierlich und durch Trial & Error lernt die KI welche Handlungen vorteilhaft sind und passt sein Verhalten an [4], [5].

2.3. Zeitplan
Wir entschieden uns bis zum 11. so gut wie möglich die Umgebung funktionsfähig zu machen und bis zum 18. die eigentliche KI fertigzustellen. (Steht noch etwas ausführlicher im Zwischenbericht)

3. Erfahrungen

3.1. Umgebung

3.1.1. Suche
Nach einer ersten Suche stossen wir auf einen Balatro Mod [6], welcher als Umgebung geeignet sein könnte: Er stellt eine API für Handlungen und Spielzustand zur Verfügung. Da Lua, die Programmiersprache von Balatro und dessen Mods, nicht für seine KI-Fähigkeiten bekannt ist, entschieden wir uns, diese API von Rust aus zu konsumieren und dort das Burn framework [7] zu verwenden um die KI zu trainieren. Leider stellte sich heraus das der Mod essenzielle Informationen, wie zum Beispiel Kartenmodifikationen, nicht liefert. Daraufhin haben wir versucht, diese direkt aus dem internen Spielzustand zu extrahieren [8], was aber selbst nach vielen Stunden Analyse desselben und Debugging immernoch häufig aus unerklärlichen Gründen einfror. Zudem hatte der Mod einige Bugs bei denen er ebenfalls einfror und keine Handlungen mehr annahm.
In Anbetracht der verhältnismässig kurzen Deadline haben wir uns schweren Herzens entschieden diesen Ansatz hinter uns zu lassen und auf eine andere Umgebung umzuschwenken: Eine Kopie von Balatro in Python [9], welche einen weitaus sinnvoller strukturierteren, und besser dokumentierten internen Spielzustand hat, und die mit vergleichsweise wenig arbeit eine gute Umgebung bieten konnte. Vereinfachend kommt dazu, dass Python der Industrie-Standard für KI ist und es somit viel Tutorials und Dokumentation für die Entwicklung verschiedener KI-Modelle hat. Als Framework haben wir uns für PyTorch [10] entschieden, da es als anfängerfreundlicher als die grosse Alternative, TensorFlow [11], beschrieben wird [12].

3.1.2. Implementation [13]
Um die Umgebung tatsächlich funktionstüchtig zu machen, implementierten wir zuerst einige Standardfunktionen, welche uns erlauben, die Umgebung zu initialisieren, neuzustarten und Aktionen auszuführen. Dies war für den Anfang genug, doch im Verlauf des Trainings wurde uns klar, dass die Menge an möglichen Aktionen der KI schlichtweg zu gross, und die Menge der legalen Aktionen zu klein waren: Wenn sich die KI in einer Runde befand waren lediglich 478 der 2^{43} möglichen Aktionen legal. Aufgrund dieser unmöglichen Wahrscheinlichkeit, sahen wir uns gezwungen, manuell legale Aktionen zu erzwingen. Auch wenn wir uns aus Zeitgründen nicht schafften, tatsächlich alle illegalen Aktionen zu verbieten, gelang es uns genug von diesen auszuschliessen, um der KI eine reale Chance zu geben, das Spiel zu meistern. Die letzte Verantwortung der Umgebung war es, die gespielten Aktionen aufzunehmen, damit wir sie zu einem späteren Zeitpunkt analysieren können.

3.2. Architektur
Im Bereich der Spiel-KIs hat es eine riesige Auswahl an erfolgreichen Architekturen, wie zum Beispiel Deep-Q-Networks [14], [15] mit seinen Erweiterungen wie Double-DQN [16], Dueling-DQN [17] und Rainbow [18], eine andere Option wären die Actor-Critic Architekturen, welche die die Advantage-Actor-Critic Familie [19] und die Algorithmen der Proximal Policy Optimization [20] beinhalten. Wir haben uns für letztere entschieden, da sie als sehr leistungsstark gelten, und ihre kontrollierten Updates der Policy für ein stabileres Training sorgen.

4. Ergebnisse

Nach etwa 22′000 Instanzen und ungefähr einem Tag Training, erreichte die KI den Boss von Ante 4, also fast die Hälfte des Spieles! Es war sehr unterhaltsam der KI dabei zuzusehe, wie sie kleine Dinge über das Spiel herausfand. Recht schnell merkte sie bereits, dass die Karten links grösseren Werten entsprechen und folglich ein klein wenig mehr Punkte geben. Auch scheint sie simple Pokerhände wie Pair oder Three of a Kind zu verstehen. Das Verständnis von den tatsächlichen Jokern lässt momentan noch sehr viel zu wünschen übrig, kein Wunder, es gibt 150 davon! Noch schlimmer sind natürlich jegliche Booster Packs, die KI bleibt dort oftmals stecken, da sie nicht weiss, was sie mit Param 2 anstellen sollte. Es wäre sicher noch sehr interessant zu sehen, wie sich die KI mit weiterem Training und Verbesserungen entwickelt.

4.1. Schlussfolgerung

Die Frage, ob es denn möglich sei, eine Balatro KI zu coden, lässt sich also mit einem klaren Ja beantworten. Wir mussten auf einen Python Port zurückgreifen, da das tatsächliche Spiel uns zu viele Kopfschmerzen bereitete, jedoch bleibt das Konzept des Spieles ja dasselbe. Unsere KI schaffte es, wie erwähnt, bis zu Ante 4 was hier der Runde 11 entsprach und einer erforderten Punktzahl von 20′000.

5. Reflexion

5.1. Planung

Wie korrekt von Herr Forny vorhergesagt, war unsere Planung zu knapp bemessen. Die eigentliche KI funktionierte zwar bereits ungefähr am 20, also etwa 2 Tage später als vorhergesagt, jedoch waren eigentlich alle der Aktionen der KI illegal. Daher mussten wir uns für eine relativ lange Zeit damit beschäftigen so gut wie möglich illegale Aktionen nicht zur Verfügung zu stellen, bei einem Spiel wie Balatro wird das jedoch sehr schnell sehr komplex. Auch tauchten einige Anomalien auf, die wir nicht mehr beheben konnten, es ist aber dennoch sehr interessant über diese zu sprechen!

5.2. Anomalien

Statt von Bestrafungen abzuschrecken wies unsere KI manchmal auch auf ein etwas… masochistisches Verhalten auf, sie machte nämlich oftmals weiter mit derselben illegalen Handlung, obwohl sie jederzeit durch eine andere Handlung weitermachen könnte. In einer Instanz erreichte die Gesamtbelohnung ungefähr −9′000, das sind ungefähr 1′800 illegale Handlungen!! Das passierte nahezu bei jeder Instanz bei welcher ein Planet Pack geöffnet wurde, da hier das Param 2 leer sein muss. Interessanterweise fand die KI sogar einen Bug in der Python Version! Mit einem bestimmten Blind Skip Tag, sind alle Rerolls im Shop permanent gratis. Wir werden dies vermutlich dem Entwickler noch melden.

5.3. Lieblingsversuch

Wir fanden überraschenderweise einen Lieblingsversuch, auch hier blieb die KI stecken, dieses Mal im Shop, da sie bereits 5 von 5 Jokern hatte und somit keine neuen erwerben konnte. Das ging so weit, dass sie schliesslich einen Grossteil ihrer Joker verkaufte um etwas neues zu kaufen. Da war jedoch der Joker “Credit Card” dabei, welcher der KI erlaubte bis zu 20 Dollar im Minus zu sein. Nach dem Fiasko war die KI schliesslich verschuldet, konnte nichts neues kaufen und ihr blieb weniger als zuvor. In der nächsten Runde starb sie einen poetischen Tod.

“The day a blind man sees. The first thing he throws away is the stick that has helped him all his life.”

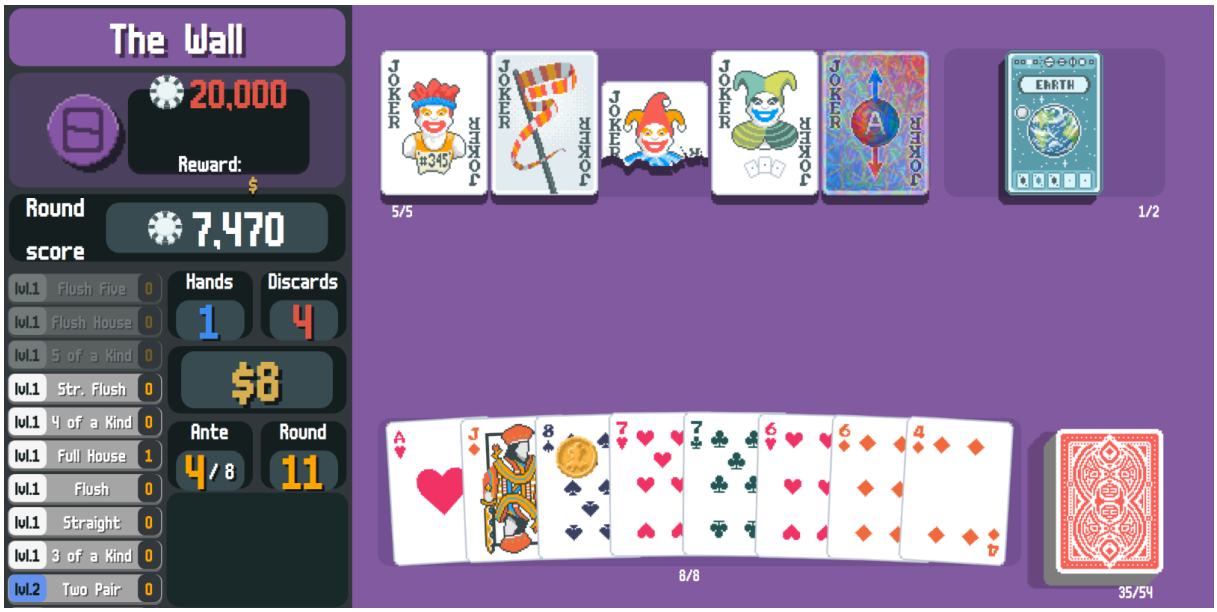

Figure 1. Der finale Spielzustand des besten Runs der KI

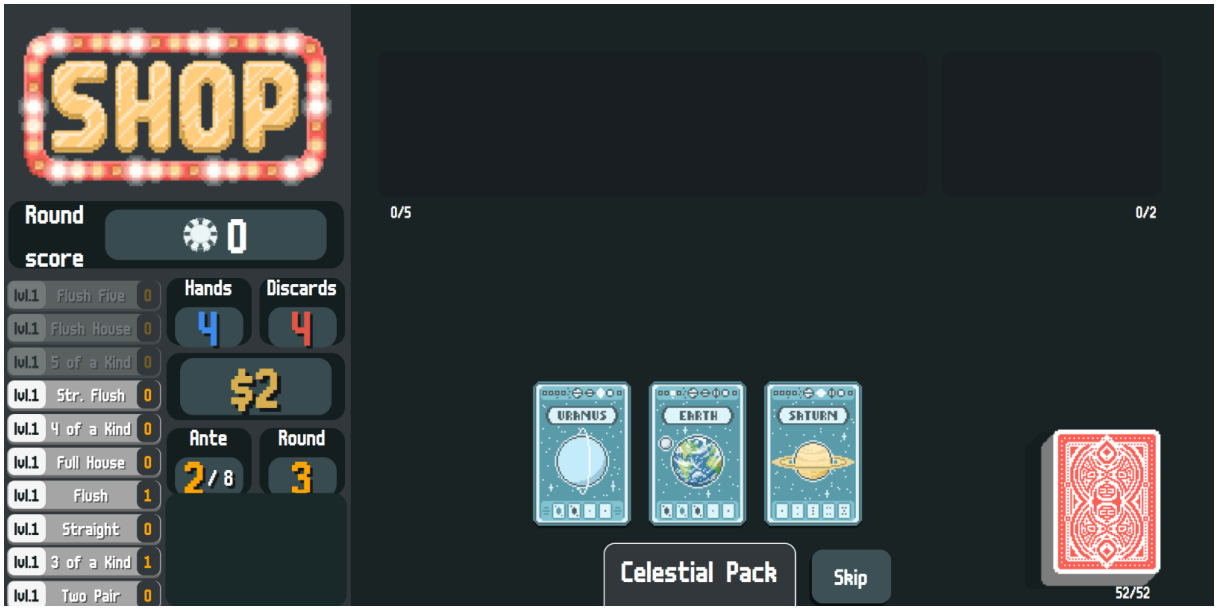

Figure 2. Der Spielzustand des Planet-Pack Param 2 Problems

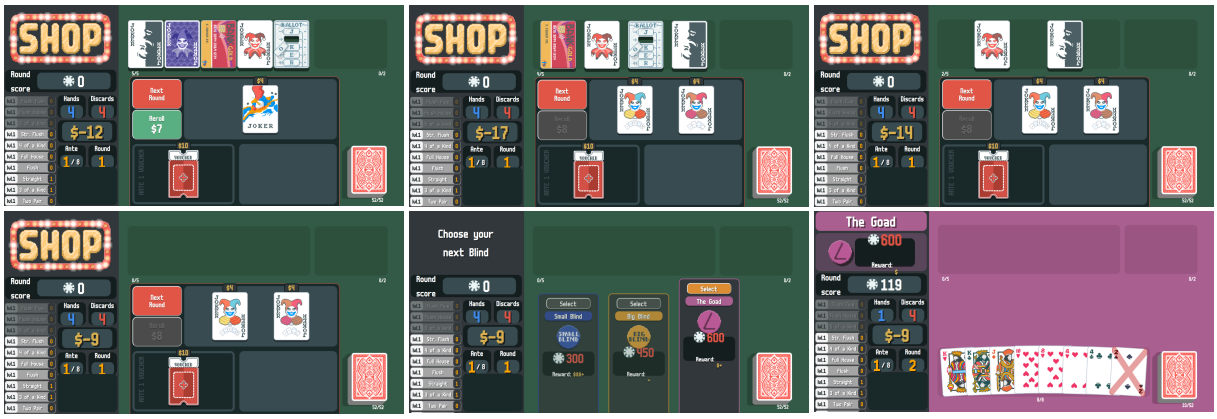

Figure 3. Der Spielverlauf unseres Lieblingsversuches

Bibliography

[1] DeepMind, “AlphaGo.” [Online]. Available: https://deepmind.google/research/projects/alphago/

[2] Deepseek, “Deepseek R1 Release 2025/01/20.” [Online]. Available: https://api-docs.deepseek.com/news/news250120

[3] Hugging Face, “Deep RL Course – Online vs. Offline Reinforcement Learning.” [Online]. Available: https://huggingface.co/learn/deep-rl-course/unitbonus3/offline-online

[4] Wikipedia contributors, “Reinforcement Learning.” [Online]. Available: https://en.wikipedia.org/wiki/Reinforcement_learning

[5] Studyflix, “Reinforcement Learning.” [Online]. Available: https://studyflix.de/informatik/reinforcement-learning-8509

[6] besteon, “Balatrobot - A Botting API for Balatro.” [Online]. Available: https://github.com/besteon/balatrobot

[7] Tracel AI, “Burn – Flexible Deep Learning Framework for Rust.” [Online]. Available: https://burn.dev/

[8] J. Lang, “balatro-bot – Extracting Game State from Balatro.” [Online]. Available: https://github.com/jullanggit/balatro-bot

[9] pjpuzzler, “python-balatro – A Python implementation of the hit roguelike deckbuilder Balatro by LocalThunk.” [Online]. Available: https://github.com/pjpuzzler/python-balatro

[10] PyTorch Development Team, “PyTorch.” [Online]. Available: https://pytorch.org/

[11] TensorFlow Development Team, “TensorFlow.” [Online]. Available: https://www.tensorflow.org/

[12] M. Shivanandhan, “PyTorch vs TensorFlow – Which is Better for Deep Learning Projects?.” [Online]. Available: https://www.freecodecamp.org/news/pytorch-vs-tensorflow-for-deep-learning-projects/

[13] J. Lang and D. Lut, “A Balatro AI written in Python.” [Online]. Available: https://github.com/jullanggit/python-balatro-ai

[14] V. Mnih *et al.*, “Playing Atari with Deep Reinforcement Learning.” [Online]. Available: https://arxiv.org/abs/1312.5602

[15] V. Mnih *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015, doi: 10.1038/nature14236.

[16] H. van Hasselt, A. Guez, and D. Silver, “Deep Reinforcement Learning with Double Q-learning.” [Online]. Available: https://arxiv.org/abs/1509.06461

[17] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, “Dueling Network Architectures for Deep Reinforcement Learning.” [Online]. Available: https://arxiv.org/abs/1511.06581

[18] M. Hessel *et al.*, “Rainbow: Combining Improvements in Deep Reinforcement Learning.” [Online]. Available: https://arxiv.org/abs/1710.02298

[19] V. Mnih *et al.*, “Asynchronous Methods for Deep Reinforcement Learning.” [Online]. Available: https://arxiv.org/abs/1602.01783

[20] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal Policy Optimization Algorithms.” [Online]. Available: https://arxiv.org/abs/1707.06347