Development of a Reinforcement Learning Agent and a Rule-Based Bot in Balatro

Frank Assumma, Benjamin Hinchcliff, Luca Ornstil, and Sean Sponsler
Cal Poly,
San Luis Obispo, United States
{fassumma,bhinchli,lornstil,ssponsle}@calpoly.edu

***Abstract*—The hit poker inspired roguelike game Balatro presents a unique challenge for AI agents due to the complexity of its environment, combining long term planning elements with dynamic short term synergies, similar to other games such as Slay the Spire. We introduce two methods for developing an agent, a rule-based method built around flushes, and a reinforcement learning method based on a Deep Q-Learning Network, achieving an average round reached of 11.2 and just over 1 respectively. We discuss the limitations of our work as well as future directions for better modeling of Balatro using learning techniques.**

I. INTRODUCTION

Balatro is a single-player poker-themed roguelike deck-building game where players construct hands using standard poker combinations while utilizing modifiers such as jokers, tarot cards, planet cards, and spectral cards to influence scoring and gameplay. Players progress through rounds by meeting point thresholds with randomized card draws and modifiers making each run unique.

Gameplay alternates between a game phase, where players form hands to reach a score goal, and a shop phase, where they purchase jokers and modifiers to improve future hands. Each game consists of eight antes with increasing difficulty and a structured sequence of small blinds, big blinds, and boss blinds. The combination of randomness, resource management, and long-term planning makes Balatro a complex strategic game.

This paper explores the development of two AI approaches for Balatro: a reinforcement learning (RL) agent, which learns optimal strategies through periodic rewards, and a rule-based flush agent, which follows a deterministic strategy focusing on building flushes. These bots provide insight into AI decision-making in a stochastic, highly strategic environment.

Designing AI for Balatro presents significant challenges, including randomness in card and modifier draws, the need for long-term planning, and the complexity of hand and modifier interactions. Additionally, the game presents a very high branching factor in the decision process, with some nondeterministic elements present. Our work aims to test if a reinforcement learning agent can adapt to these challenges while comparing it to a structured, rule-based approach. The findings contribute to broader research on AI and strategy games, where adaptability and resource management are crucial.

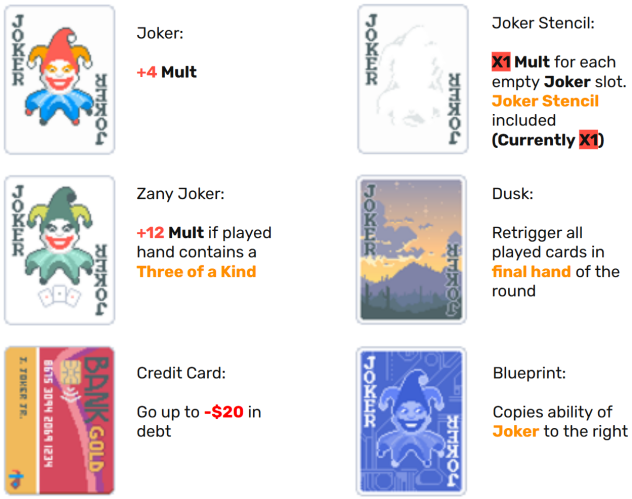

Fig. 1. Six selected examples showcasing the diversity of Jokers in Balatro, chosen from the game's total of 150 unique Jokers

II. BACKGROUND

A. Gameplay Mechanics

In Balatro, players construct hands using standard poker combinations, but the game introduces unique modifiers such as jokers, tarot cards, planet cards, and spectral cards to influence scoring and gameplay. The objective is to progress through rounds by meeting or exceeding certain point thresholds with each playthrough offering randomized cards, jokers, and values.

The game is comprised of two phases that alternate: the game phase, and the shop phase. Every game begins in the game phase, where the player must make five-card poker hands out of eight cards they are dealt. Each blind has a fixed number of discards and hands that may be played to reach the point goal. Once the player is at the point goal, they receive cash for each remaining discard and for interest on their current cash. The player then enters the shop phase, where they can purchase jokers and other modifier cards that enhance cashflow, certain hands, individual cards, point multipliers, and more. Figure 1 illustrates a selection of these diverse jokers and effects.

Each game is separated into eight antes with three game phases (blinds) and three shop phases each. The first blind from each phase is the small blind, the second blind from each phase is the big blind, and the final blind from each

phase is the boss blind. Once the boss blind at each ante is defeated, the player moves onto the next ante. After defeating all eight antes, the player wins. If the player runs out of hands and does not reach the point requirement in the game phase, they lose and must restart.

Throughout this paper, we will refer to rounds by a round count. As there are three blinds per ante, round count is calculated as $3(n-1)+k$, where n is the current ante, and k is 1 for the small blind, 2 for the big blind, and 3 for the boss blind. Reaching round 3 indicates that an agent made it to the boss blind of the first ante. Reaching round 10 indicates that an agent made it to the small blind of the fourth ante.

B. Strategy

Beyond core mechanics, Balatro is distinguished by its deep strategic layers and high replayability. Its randomized elements and unique modifier system ensure that every run is different, encouraging experimentation and strategic adaptation.

In human play, Balatro requires players to focus and build a strategy that performs well. This involves complex decision-making during both game phases, consideration of randomness in multiple avenues, and long-term planning to ensure scored points are scaling at a level comparable to the game's difficulty.

This creates a very dynamic and therefore challenging environment where players must make a build that has a high level of synergy and is able to consistently score high, while factoring in the inherent randomness of the game. Developing an AI capable of mastering such a game is both a significant challenge and a valuable opportunity to advance research in AI game-playing agents. Unlike deterministic games like chess, where the entire game state is visible, Balatro introduces elements of uncertainty and incomplete information, requiring sophisticated decision-making algorithms.

III. RELATED WORK

The development of AI agents for card games has been an active area of research. Yilmaz and Öztürk explored the use of reinforcement learning combined with supervised learning to create self-learning agents for the card game Batak, the Turkish variation of Bridge [1]. Their study highlights the potential of hybrid learning methods in complex game environments. Similarly, projects such as $balatrobot$ and $Balatro - Bot$ provide foundational frameworks for bot development in Balatro. However, these implementations primarily rely on rule-based strategies with limited adaptability to the game's dynamic environment [2], [3].

Research on poker-playing bots offers valuable insights into handling incomplete information, a crucial challenge in Balatro. Ashi [4] developed AI planning techniques for poker, while advancements in Monte Carlo Tree Search (MCTS) and Reinforcement Learning (RL) have demonstrated effectiveness in navigating complex, stochastic environments [5]. These techniques will play a central role in our approach to developing an AI agent for Balatro, enabling it to address the game's strategic and probabilistic challenges.

Beyond card games, prior research on Roguelike deck-building games, such as Slay the Spire, provides additional relevant insights. Mathijssen [6] applied machine learning techniques to predict game success based on card selections throughout a run, an approach that could be adapted to predict the effectiveness of Balatro strategies. Additionally, studies on reinforcement learning in dynamic Roguelike environments have explored methods for adapting to rapidly shifting objectives, which may prove essential given the strategic complexity of Balatro [7]. Recent work has also investigated the application of large language models (LLMs) to Slay the Spire, albeit with limited success, suggesting an alternative avenue of exploration in Balatro [8].

While Balatro shares elements with traditional card games and Roguelike deck-builders, no prior research has explored AI-driven strategies for the game. Existing Balatro bots primarily use simple rule-based approaches, and no work has examined reinforcement learning in this context. Our research represents the first attempt to apply RL to Balatro, alongside a novel rule-based flush bot, to analyze strategic decision-making in the game's stochastic environment.

IV. METHODS

A. Bot Framework and Game Modding

In order to interface the game, we chose to use the balatrobot API [2], an independently developed project for interfacing between AI agents and Balatro. Initially, we forked the balatrobot repository and made a series of adjustments to work with various operating systems and setups. This included proper interfacing with Steamodded and Lovely, the respective modding framework and injector of modified game logic utilized in the project. Through this setup, we were able to extract game state information, execute actions programmatically, and implement custom decision-making logic.

While balatrobot provided the foundation for bot automation, the game's randomized elements and unique modifier system introduced significant complexity. Each run features randomized booster packs, card upgrades, shop offerings, and blind modifiers, all of which contribute to a high branching factor that complicates planning-based approaches. Due to these factors, we limited our shop phase interactions to only purchasing and selling jokers. Tarot, planet, and spectral cards introduce additional variance, often requiring context-dependent decision-making that would be too difficult to model within a rule-based or reinforcement learning framework.

B. Decision-Making Constraints and Rule-Based Approach

In the play phase alone, a player can play or discard between 1-5 cards out of 8 choices. Therefore, without any modifiers present, there are $(8 + \binom{8}{2} + \binom{8}{3} + \binom{8}{4} + \binom{8}{5}) * 2 = 438$ possible actions for each turn. Beginning with 4 hands and 4 discards per round, a player has up to 8 turns in each round. Consequently, if a player were to use all hands and discards to reach the goal, there are 438^8 possible actions during a round. As players can increase their hand and discard limit as

well as their hand size as the game progresses, the turn count and action count generally increases, leading to an even higher action space.

Given Balatro's procedural generation and deep decision trees, Monte Carlo Tree Search (MCTS) and full play phase and shop phase reinforcement learning were initially considered as strong candidates for decision-making. MCTS, in particular, is commonly used for turn-based decision-making in games with well-defined action spaces like Go and chess. For example, DeepMind utilized MCTS in its development of AlphaGo, the first Go agent to defeat world champion Go players [9]; however, despite its advantages, MCTS proved impractical due to technical limitations imposed by the balatrobot API.

While MCTS relies on simulating future game states to evaluate decision paths, the balatrobot API does not allow arbitrary state manipulation or lookahead rollouts. Since MCTS fundamentally depends on running multiple simulated plays per decision step, this restriction makes it impossible to implement MCTS effectively within the current API framework. Additionally, interactions with the game are limited to predefined actions, preventing the kind of controlled simulations necessary for MCTS-based planning.

If future developments are made to the balatrobot API that allow for more flexible state manipulation, MCTS could become a viable alternative for decision-making. This will be expanded upon more in the future works section.

Given the above constraints, we opted to develop two separate strategies for comparison: a rule-based strategy that focused on maximizing flush consistency and joker efficiency and a reinforcement learning strategy which learned in the play phase but utilized a rule-based purchasing strategy in the shop phase.

C. Rule-Based Flush Bot

The development of the rule-based flush bot began with a baseline implementation, forked from the initial balatrobot repository. This initial bot followed a simple strategy:

- Flush Priority: it prioritized playing flushes whenever possible.
- Discard Optimization: If a flush was not playable and discards were available, the bot discarded all cards that were not of the most frequently held suit, ensuring that no more than five cards remained.
- Fallback Hand: If a flush was not playable and no discard was available, the bot played all cards not of the most frequently held suit

However, this initial version lacked the ability to purchase jokers, reroll the shop, or sell jokers, significantly limiting its effectiveness.

To improve performance, we introduced the following modifications:

1) Joker Purchasing: The first major improvement was enabling the bot to purchase filtered jokers, avoiding those that incentivized non-flush hands.
2) Shop Rerolling: To further optimize joker acquisition, we provided the bot with the ability to reroll the shop.
3) Joker Selling: The bot was also modified to sell jokers selectively to maintain an optimized set of jokers. However, very few jokers are actually sold to prevent the issue of frequent playing with only four active jokers. Since the bot can only sell jokers at the start of each round, excessive selling previously led to prolonged rounds with an unfilled joker slot.

To address the joker selling challenges, the bot distinguishes between strong jokers and weaker jokers. The weaker jokers are those that are acceptable purchases but may lose effectiveness later in the game. During testing, an overabundance of weaker jokers led to frequent selling and suboptimal joker slot usage. To mitigate this, only a select few jokers are classified as weaker. These jokers provide early-game benefits but become less effective later on. This refined strategy ensures that the bot regularly maintains all five joker slots filled, improving consistency and long-term effectiveness.

D. Reinforcement Learning Bot

In addition to the rule-based flush bot, we explored a Deep Q-Learning Network (DQN) to learn optimal strategies through reinforcement learning. A DQN is a reinforcement learning algorithm that combines Q-Learning with deep neural networks to handle environments with large states and action spaces [10]. The DQN framework was designed to model Balatro's decision-making process as a Markov Decision Process (MDP), where each game state was represented as a feature vector, and the agent learned to maximize long-term rewards through experience.

1) State Representation: The state space was constructed to capture the main relevant aspect of the game environment: the hand composition. The hand is encoded using a multi-hot encoding, one-hot encoding each card as a fixed number between 1 and 52. We utilize the sum of the cards, allowing for multiples of the same card. By encoding this feature, the network was able to assess game state and predict optimal actions based on learned Q-values. In the future, we would like to capture more aspects of the game environment, like remaining discards, remaining hands, the point goal, and the current joker state.

2) Action Space: The DQN's action space was designed to balance learning efficiency and legality of moves, preventing excessive branching complexity. Instead of a naive approach where the DQN directly selects a full hand, the action space was structured as the state space multiplied by five, representing the model's preference for each of the five potential hand slots. The network outputs weights corresponding to these preferences, which are then softmaxed and masked, ensuring that only legally playable cards are considered. Invalid choices are zeroed out, and a final card selection is made by sampling from the resulting probability distribution.

This structure was implemented to prevent the DQN from repeatedly making illegal moves, which previously hindered learning. Additionally, if duplicate selections occurred, the bot

TABLE I
TABLE 1: REWARD BONUSES OF EACH HAND TYPE PROVIDED TO THE DQN

Hand Type	Reward Bonus
Full House	20
Flush	15
Straight	15
Three of a Kind	10
Two Pair	5
Pair	-2
High Card	-5

would automatically play a hand with fewer than five cards rather than failing to execute a valid action.

During the shop phase, a rule-based heuristic was still in place, allowing the DQN to purchase select jokers based on predefined criteria while preventing excessive spending via controlled shop rerolls. Joker selling was also introduced to make room for stronger options.

Notably, the decision to exclude tarot, planet, and spectral cards from the action space was made to reduce variance in state transitions and focus on core deck and joker interactions.

While an alternative encoding using indices and a binary mask could potentially improve efficiency, the chosen approach ensures that the model does not rely on specific card indices and instead the actual card, reinforcing a more generalizable strategy.

3) Reward Function: A well-defined reward function was critical to the DQN's performance. The reward structure was designed as follows:

- Hand Type Reward: Immediate reward based on the type of hand chosen, encouraging higher-scoring hand types
- Chip Reward: Immediate reward based on the chip difference between the current played hand and the last played hand
- Resource Usage Bonus: A reward that indicates the bots' use of hands and discards. Fewer remaining hands and discards lead to a higher resource usage bonus, steering the bot towards improving its hand type and quality by playing a hand with a high chip count.

The hand type reward was represented with a key-value mapping, as is shown in Table I.

Additionally, the calculation for resource usage bonus is provided in Equation 1, where λ represents the scaling factor, D is the initial discard count, d is the current discards remaining, H is the initial hand count, and h is the current hands remaining.

$$bonus = \lambda(\frac{D-d}{D} + \frac{H-h}{H}) \tag{1}$$

This structure reinforced searching for high-scoring hands and chip counts while discouraging suboptimal hand types.

4) DQN Architecture: We implemented a deep feedforward neural network with six fully connected layers for function approximation:

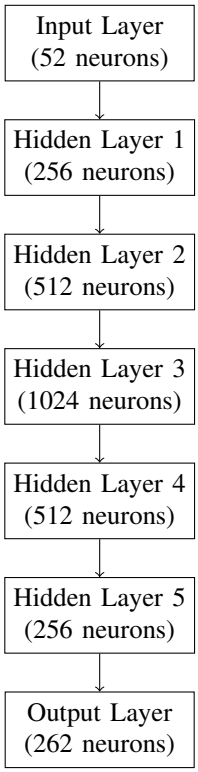

Fig. 2. DQN architecture showing each layer and the neuron count at that layer

- Input: State vector of dimension equal to the number of observations (52 cards in a deck)
- Hidden Layers: ReLU-activated layers increasing up to 1024 neurons before contracting.
- Output: Q-values for the number of actions, representing estimated future rewards for each decision.

Figure 2 shows the network architecture used in our research. Note that the input number of observations in the network is 52, the number of cards in a deck, and the number of actions is represented by the number of cards (52) multiplied by the maximum number of cards per hand (5) added to the amount of options there are in a play – discarding or playing a hand (2). This leads to 262 possible actions. Also note that jokers with modifiers that change these values were ignored in the shop to keep constant inputs and outputs.

Our DQN implementation consists of two neural networks: a policy network, which is actively trained to approximate the optimal Q-values, and a target network, which is a delayed copy of the policy network used to generate stable target Q-values. The target network is updated incrementally using a soft update strategy with a small update rate ($\tau = 0.005$) to prevent instability in training.

The DQN training was guided by experience replay and a separate target network to stabilize learning. Hyperparameters included:

- Batch Size: 128
- Discount Factor (γ): 0.99

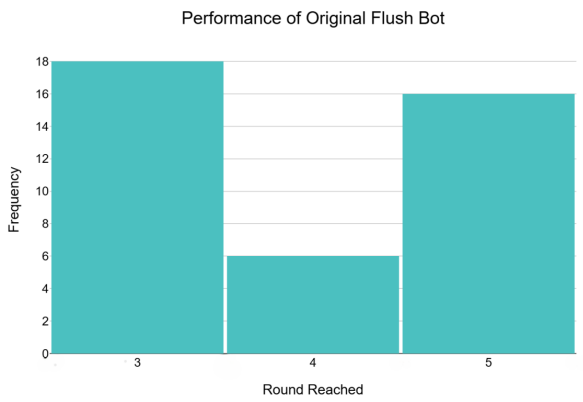

Fig. 3. Histogram of rounds reached by the original flush bot in 40 runs

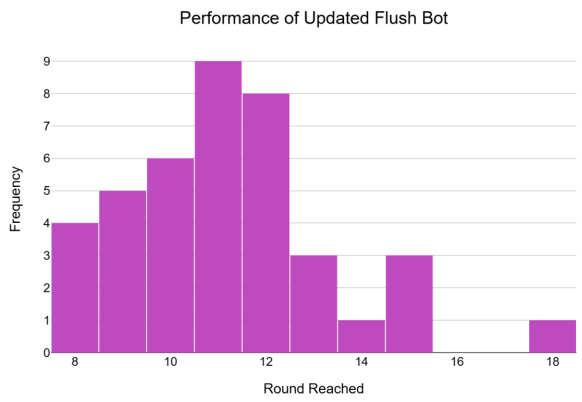

Fig. 4. Histogram of rounds reached by the updated flush bot in 40 runs

- Exploration Rate (ϵ): Decayed from 0.9 $\longrightarrow$ 0.05 over 1000 episodes
- Optimizer: AdamW
- Learning Rate: $1 * 10^{-4}$
- Replay Buffer Size: 2500

During training, actions were selected using an ϵ-greedy strategy, with exploration decreasing over time.

V. Evaluation / Results

A. Rule-Based Flush Bot

To evaluate the impacts of updates made to the rule-based flush bot, we conducted 40 test runs for both the original and the updated versions, seen in Figure 3. The original flush bot, which lacked the ability to purchase jokers, achieved an average round of 3.95, with the most common stopping point being round 3 (n=18). The bot never progressed beyond round 5, as clearing the Big Blind of Ante 2 (1200 points) proved infeasible given its limitations. Without access to multipliers or chip-enhancing effects from jokers, the bot's score per flush hand remained within the high-200s to low-300s, making it highly unlikely to consistently reach the required thresholds for later rounds. Observationally, the bot was able to form at least one flush per round with high probability, and on average, three flushes per round, but achieving more than three was rare.

In contrast, the updated flush bot, which incorporated joker purchasing, shop rerolling, and joker selling, demonstrated a significant performance improvement, reaching an average round of 11.2 across 40 runs as seen in Figure 4.

The bot most frequently reached round 11 (n=9) and round 12 (n=8), with its best recorded performance reaching round 18 (Boss Blind of Ante 6). Among the implemented enhancements, joker purchasing had the most substantial impact, allowing the bot to construct significantly stronger flushes. With five active jokers, a single flush could score several thousand points rather than a few hundred, a substantial increase. Despite these improvements, the updated bot's performance remained highly dependent on variance, which influenced success in three primary ways:

1) Flush Formation: The number of flushes successfully built per round remained a major factor in performance.
2) Joker Availability: The quality and presence of useful jokers in the shop significantly affected score potential.
3) Boss Blind Matchups: Certain boss blinds posed a significant challenge for the bot by restricting the bot to playing certain hands. For example, "The Eye" boss blind forces playing a unique hand type only once per round, severely hindering the ability of our bot to advance.

To mitigate the variability of joker availability, two additional stratefies were implemented: rerolling the shop and selling weaker jokers to create space for stronger options. Testing indicated that limiting shop rerolls to two was optimal, as three rerolls often disrupted the game's expected timing for shop interactions. Furthermore, the bot immediately ended the shop phase upon making a purchase to streamline execution. Joker selling, though a more moderate improvement, also proved to reduce variance and provide long-term benefits by removing early-game jokers that became less effective in later rounds.

Future improvements could further reduce the bot's reliance on favorable random number generation (RNG). Notably, expanding the bot's purchasing capabilities to include consumables, like booster packs, tarot cards, planet cards, and spectral cards, could enhance joker acquisition rates and boost hand scores. However, implementing these features requires additional decision-making logic for selecting optimal cards, which was beyond the scope of our research.

B. Deep-Q Learning Agent

For the evaluation of the DQN agent, we decided to track both the training performance in terms of training loss and the game performance in terms of the following metrics:

- The occurrences of a given hand-type: The agent should be able to learn patterns as hand-types from our supplemental reward structure. We can track which hands the bot plays the most often to determine if it is straying from

known random probabilities of hands in poker, towards learned structures which give higher rewards.

- The final round reached by the bot: At a high-level, this helps to show the bot performance as a whole in terms of how far it is able to reach within the game over the period it is playing.
- The score of the last hand played: A deviation beyond the metric of the final round reached is required to understand where the bot is failing within each round in terms of score.

Figure 5 is a histogram representing our first metric, showing the count of the occurrences of each hand-type. The bot evidently favors pairs and two-pairs as hand-types which corresponds to the fact that pairs have by far the largest probability of appearing in any given poker hand. While additional reward tuning could stray the bot away from choosing pairs over time, Balatro is a unique environment where the collection of certain Jokers could directly incentivize specific hand-types.

Figure 6 is a histogram representing our second metric, showing the count of the final round reached from a given epoch. With an overwhelming total of 1012, it was rare for the bot to proceed, as it reached the second round 11 times, the third round 2 times, and on a fortunate run, reached round 5 one time. Evidently, the bot performance represents the failure to properly identify patterns in the poker-specific action set, as the first round is encountered without the availability of Jokers which could drastically increase the score obtained during the round. When the bot passes the first round however, the increased score threshold for consecutive antes requires the purchase of Jokers, which have an extremely wide variance of assistance towards performance, as some require the use of specific hand-types for their operation to trigger, while others may provide a universal multiplier to score.

Figure 7 is a histogram representing our final metric, showing the distribution of scores for the final hand played by the bot, with an average score of 106.2 and a standard deviation of 47.6. We are able to see that the largest score distribution is correlated with our findings on the hand-types most often played by the bot, as the hand score for high card, pair, and two-pair line up with the peaks on the histogram.

In terms of training performance, we captured the loss of the DQN agent over the course of its exploratory period before approaching the floor threshold of 0.05. Figure 8 shows that the Huber loss of the agent over its training period did not improve significantly with time.

We suspect this inability to properly learn patterns over time is a result of the size and sparsity of the action space. This causes many outputs from the model to not have any measurable effect on the loss, which diminishes learning from any single step. While the action space was designed to be configuration-independent, this design choice may have ultimately overcomplicated decision-making to the detriment of overall performance. Addressing this issue, whether through modifying the action space, refining reward structures, or incorporating more advanced learning techniques, presents a strong avenue for future research.

VI. FUTURE WORK

Given the infancy of the space and lack of work on Balatro, there are several key directions for future work. First, improving the Balatro botting framework would provide a stronger foundation for testing different algorithms, particularly MCTS, and allow for the use of better reinforcement learning APIs. MCTS relies on an effective forward model, which the current framework lacks, making it difficult to implement effectively. Developing a dedicated forward model would allow for deeper search-based decision-making and potentially stronger overall performance. Furthermore, industry-standard reinforcement learning APIs like Gymnasium are very challenging to interface with the current botting framework. Developments in the botting API could allow for more optimized RL training systems, ultimately improving results.

For reinforcement learning approaches, a limitation of our DQN agent was its action modeling. The large joker space

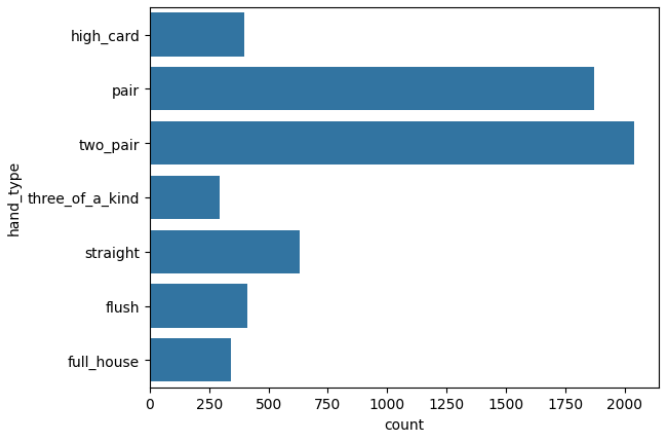

Fig. 5. Histogram of played hand-types

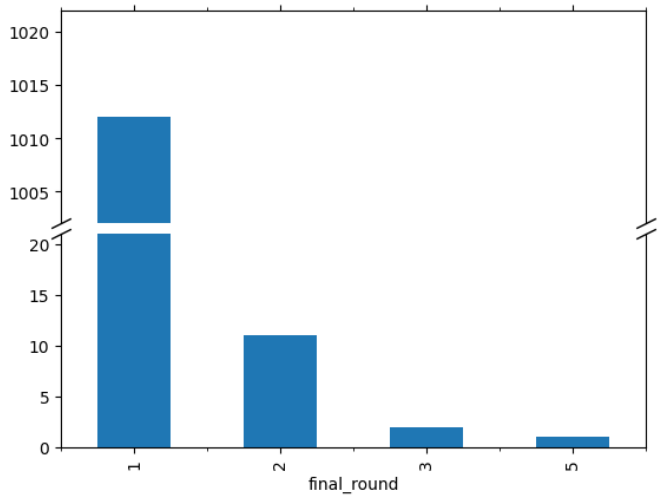

Fig. 6. Histogram of final round reached

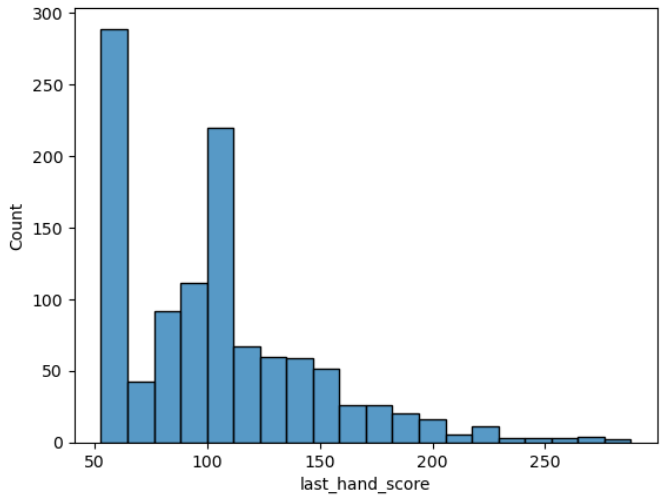

Fig. 7. Histogram of score of last played hand

likely suffers from sparsity, making it difficult for the model to repeatedly encounter and learn from similar decisions. Future research should explore shrinking the action space, possibly by filtering or clustering joker-based decisions, or by shrinking the joker pool. Additionally, expanding the state representation could improve the agent's ability to recognize useful patterns over time. This expansion could involve exploration of remaining hands, remaining discards, point goals, and more in addition to the hand encoding to provide the model with more patterns.

Beyond DQN-specific improvements, alternative reinforcement learning techniques like poolicy gradient methods, actor-critic architectures, or hybrid approaches incorporating MCTS, may offer more robust learning. Experimenting with different reward structures is also a promising direction, as the current system may not adequately reinforce long-term strategy over immediate gains.

Lastly, expanding the rule bot's state representation and purchasing capabilities could lead to significant improvements

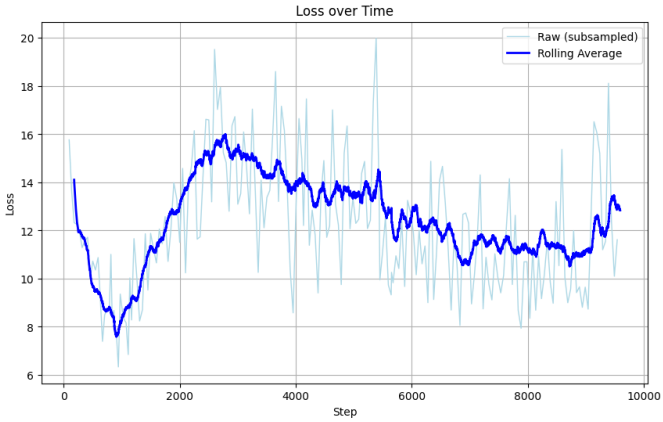

Fig. 8. Huber loss over training period

in performance. While our current updates focus on joker purchases, shop rerolls, and selective joker selling, future iterations could extend the work to consumables such as booster packs, tarot cards, planet cards, and spectral cards. Here, developments in consumables could enhance both joker acquisition and hand scoring potential. Additionally, access to game state information like the boss blind type could allow for better decision-making and planning by the flush bot, avoiding potential round-ending bosses.

By addressing these areas, future research can refine both rule-based and reinforcement learning approaches to develop a more sophisticated Balatro-playing agent.

VII. CONCLUSIONS

Balatro presents a complex challenge for AI agents, requiring both long-term strategic planning and short-term adaptability, In this work, we explored two approaches for developing an AI agent: a rule-based flush bot and a Deep Q-Learning Network (DQN) reinforcement learning agent. The rule-based flush bot demonstrated a significant improvement over its original version, achieving an average round score of 11.2 with the addition of joker purchasing, shop rerolling, and joker selling mechanics. Meanwhile, the DQN found challenges learning effective strategies, likely due to the high-dimensional and sparse nature of the action space, and challenges in reward signal propagation.

Our results highlight the difficulties of applying reinforcement learning to Balatro without a more structured framework. Future work should prioritize the development of a more accessible and forward botting API for use with diverse model setups. Additionally, focus should be placed on refining the representation of the action space, testing alternative reinforcement learning techniques, and integrating a forward model for more effective decision making, such as MCTS. Improvements to the rule-based bot, including expanding purchasing capabilities to consumables and more robust state representation, could provide a stronger baseline for comparison.

Ultimately, our findings suggest that while rule-based methods can be effective with domain-specific knowledge, learning-based approaches require more refined modeling to navigate Balatro's complexity. Developing a comprehensive AI for Balatro remains an open challenge, and future research will be essential in bridging theh gap between handcrafted strategies and adaptive learning techniques.

REFERENCES

[1] Y. Yilmaz and O. Öztürk, "Reinforcement learning in card game environments using monte carlo tree search," in *2019 4th International Conference on Computer Science and Engineering (UBMK)*. IEEE, 2019, pp. 1–6.

[2] besteon, "balatrobot," accessed: 2025-01-26. [Online]. Available: https://github.com/besteon/balatrobot

[3] R. Elliston, "Balatro-bot," accessed: 2025-01-26. [Online]. Available: https://github.com/raienelliston/Balatro-Bot

[4] M. Ashi, "Ai planning for poker player," Master's thesis, University of Stuttgart, 2022, accessed: 2025-01-26. [Online]. Available: https://elib.uni-stuttgart.de/bitstream/11682/12840/3/thesis_ashi.pdf

[5] T. Vodopivec, S. Samothrakis, and B. Ster, "On monte carlo tree search and reinforcement learning," *Journal of Artificial Intelligence Research*, vol. 60, pp. 881–936, 2017.

[6] C. Mathijssen, "Predicting a successful run in slay the spire," Master's thesis, Tilburg University, 2023. [Online]. Available: https://arno.uvt.nl/show.cgi?fid=169629

[7] A. Sestini, A. Kuhnle, and A. D. Bagdanov, "Deep policy networks for NPC behaviors that adapt to changing design parameters in roguelike games," *CoRR*, vol. abs/2012.03532, 2020. [Online]. Available: https://arxiv.org/abs/2012.03532

[8] B. Bateni and J. Whitehead, "Language-driven play: Large language models as game-playing agents in slay the spire," in *Proceedings of the 19th International Conference on the Foundations of Digital Games*, ser. FDG '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3649921.3650013

[9] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016. [Online]. Available: https://doi.org/10.1038/nature16961

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015. [Online]. Available: https://doi.org/10.1038/nature14236