

Training a Reinforcement Learning Agent for Tales of Tribute

Sebastian Lashmet
Byram Hills High School
Armonk, United States
lashmets25@byramhills.net

Alexander Dockhorn
Faculty of EECS
Leibniz University Hannover
Hannover, Germany
dockhorn@tnt.uni-hannover.de

Abstract—Tales of Tribute is a strategic deck-building card game that presents unique challenges for artificial intelligence due to its complex action space, hidden information, and long-term planning requirements. In this work, we propose a reinforcement learning agent that learns to play Tales of Tribute without relying on handcrafted heuristics. Our approach introduces a generalizable game state representation and a scalable action evaluation mechanism based on preference vectors. We train our agent using Proximal Policy Optimization within the Scripts of Tribute framework and demonstrate competitive performance against established search-based agents, including Monte Carlo Tree Search. The results validate the viability of reinforcement learning in Tales of Tribute and highlight the potential of preference-based action evaluation in domains with large and variable action spaces. Further experiments will be required to test its viability in other card games.

Index Terms—Tales of Tribute, Deck-building Card Game, Reinforcement Learning

I. INTRODUCTION

The publication of the Legends of Code and Magic framework [1], [2] and its corresponding AI competition allowed AI developers to craft new solutions for the game Tales of Tribute (ToT). Originally released as a mini-game for *The Elder Scrolls Online*, ToT is a complex deck-building card game for a variety of reasons. Similar to many collectible/trading card games, it features profound strategic depth, hidden information (e.g., the hidden cards in the opponent's hand), and random elements (e.g., random card draws akin to poker). In addition, ToT players can “purchase” new cards mid-game to construct a dynamic deck of cards. This incorporates a new layer of long-term planning into the game, since bought cards remain with a player for many future turns, and allows players to dynamically adapt their deck to the opponent's strategy.

Previous research on AI for ToT has addressed the game's complexity using manually designed heuristics to guide search-based agents [2]. While effective, these heuristics encode human knowledge about strategy, limiting adaptability to changes in game rules or mechanics. In contrast, our study aims to reduce reliance on domain-specific decision heuristics by enabling an agent to learn its policy through reinforcement learning. While we do apply shaped rewards to support training stability, these are general-purpose and do not encode explicit game strategies or action preferences.

Therefore, the aim of this study is to create an agent that learns a strong policy for ToT without major loss of generality.

We achieve this by:

- a generalizable representation scheme for ToT's observation space,
- a scalable neural architecture that evaluates actions via learned preference vectors rather than fixed heuristics,
- and an empirical comparison of the trained model against the strongest available heuristic-based agents [1].

II. TALES OF TRIBUTE (TOT)

ToT is a deck-building card game released as part of the High Isle chapter of *The Elder Scrolls Online* [1]. Unlike many CCGs, ToT requires players to build their own unique decks throughout the course of the game. Successful agents must exhibit long-term planning capabilities as they balance short-term advantages with resourceful deck construction. In the following, we introduce its most important components that are relevant for this work:

Deckbuilding Throughout the game, players in Tales of Tribute accrue “Gold”, which they can use to buy cards from the “Tavern”. The Tavern is a collection of five cards that are placed in the middle of the board and replenished as cards are bought. Once a player buys a card (see Figure 1, that card becomes part of their Deck. For the rest of the game, the player will be free to use that card whenever they draw it. A core element of the game is the ability to discern which cards will be useful to a player, and whether they should be bought, discarded, or used at all.

Patrons At the beginning of the game, each player selects two “Patrons,” for a total of four. Patrons are in-game characters that appear on the side of both players' screens. These Patrons each come with their own smaller mini-decks that contribute cards to the Tavern.

Prestige Some cards in ToT, once played, allow a player to gain “Power”, which is represented by a counter in the center of the screen. At the end of a player's turn, all unused Power gets converted into “Prestige”. Ultimately, the first player to reach the goal of 40 Prestige wins the game, unless the opposing player also reaches 40 Prestige immediately after. In that case, the game will continue until one player reaches an amount of Prestige that the other cannot match.

Short Paper

Basic Strategy According to an internet gaming news and strategy discussion platform [3], it is best to focus on building a strong, consolidated board state at the beginning of the game. This means buying as many cards as possible that give the player lots of Gold. Prestige is relatively unimportant in the early stages of the game. This changes in the late game, when it becomes a good strategy to aggressively push for a surge in Prestige.

III. RELATED WORK

Currently, the best ToT agents use MCTS aided by heuristic evaluation functions. In particular, Cieżkowski & Krzyżyński's [4] state-of-the-art approach runs five MCTS trees, each searching until the first occurrence of imperfect information, and generates state evaluations with a heuristic function tuned by an evolutionary algorithm. Similarly, Kowalski et al.'s [1] strong baseline search agents search on a single tree and assign heuristic values to leaf nodes.

This method is currently the most successful. However, being developed by humans, there is no guarantee that the heuristic rules are optimal with respect to the task at hand. Furthermore, the heuristic functions' reliance on pre-existing human knowledge hinders the greater applicability of this technology. In contrast, self-learning ML methods are yet to be explored in ToT. Therefore, the goal of this paper is to develop the first self-taught agent for ToT.

In previous work, we have explored learning-based approaches in other card games. Our work on fuzzy multiset clustering [5], [6] supports strategy analysis and prediction based on deck structure, while the work on opponent modeling for Hearthstone [7] demonstrates the value of anticipating moves in partially observable games.

IV. DEVELOPING A TALES OF TRIBUTE RL AGENT

The first few actions in a Tales of Tribute game consist of the "drafting" phase, in which players select the Patrons that will be used for the game. It would be possible to modify our agent's architecture such that it learns which Patrons to select at the beginning of the game; however, this would likely result in the agent learning a few "favorite" Patrons that it almost never deviates from selecting. This would allow the agent to learn a playstyle entirely focused on a single Patron, undermining the versatility of RL that precipitated this research. Therefore, our agent begins the game by selecting its Patrons randomly.

Several major challenges need to be addressed when developing a ToT RL agent. First, we present our encoding for the agent's state and action space. Second, we describe the neural network architecture used to learn an action-evaluation function. Last but not least, we summarize general considerations on the design of the agent's training. The corresponding source code can be found at: <https://github.com/ADockhorn/Reinforcement-Learning-for-Scripts-of-Tribute.git>

A. Gamestate Representation

For each decision, the agent needs to receive information on the game's current state. The gamestate mainly consists of

Gold

For a player to use this card, they must first buy it from the Tavern for 6 gold.

Play Effects

When played, March on Hattu may have one of two effects at the player's discretion:

- A) "Gain 4 Power"
- B) "Draw Refresh – Return up to 3 cards of any type from your cooldown to the top of your draw"



Deck: Ansei

This card combos with other cards from the "Ansei Frandar Hunding" deck.

"Combo 2" Effects

If this is the second Ansei card played this turn, a player may gain 1 Power in addition to other card effects.

Fig. 1. March on Hattu is a card that can be bought, discarded, recovered, or used by either player. Representing the card requires representing each of the highlighted characteristics as well as the card's location on the board. (Card image taken from <https://elderscrollsonline.wiki.fextralife.com>)

cards that can be placed in different areas, e.g., the player's Hand or Deck. To process the data with a common densely connected neural network, we must parse the gamestate into a single vector of fixed length. To do this, we introduce a method to represent individual cards as smaller "card vectors." The observation is made up of many card vectors appended into one long gamestate vector, where different sections of that vector represent different areas of the game (e.g., Hand, Deck, Discard). Of course, this necessitates a method for creating the card vectors themselves.

a) Representing Individual Cards: Consistent card representations can be difficult to design, since ToT cards may have multiple effects simultaneously. Our workaround to this consists of encoding each card as a vector of dimension 14 representing the integer arguments of its play effects, plus another five components representing the card's Deck, Gold cost, type, and HP (for cards of the 'Agent' type; -1 otherwise). The play effects are repeated four times to represent the four levels of "combo effects" that some cards may have. The structure of such a vector is shown below in Table I. Lines 5–19 are repeated a total of four times to encode the four levels of card combination effects. This results in a vector of dimension 61.

b) Representing Board Areas: For each board area, we determine a maximum number of cards (either given by the rules or by play data) and assign it an appropriate amount of space in the observation. For example, a bundle of $61 \times 5 = 305$ elements corresponds to the five cards in the Tavern. Under the circumstances where there are fewer cards in a given area, we fill the rest of the designated slots with zeros, representing "null" cards that do not do anything. If there are somehow more cards than available space, the observation only considers the first n cards sent by the JSON-encoding from which the gamestate is parsed. Thus, the observation never grows or

TABLE I
STRUCTURE OF A CARD VECTOR.

Slot	Argument
1	Patron Deck of origin
2	Gold cost
3	Card type (e.g. “action”)
4	Card HP (“Health Points”)
5	Taunt (Boolean)
6	Gain n Gold
7	Gain n Power
8	Gain n Prestige
9	Opponent loses n Prestige
10	Replace n cards from Tavern
11	Acquire n cards from Tavern
12	Destroy n cards
13	Draw n cards
14	Opponent discards n cards
15	Recover n cards from discard
16	Knock out an opponent’s Patron
17	Call on n additional Patrons this turn
18	Create n “Summerset Sacking” cards
19	Heal n damage

shrinks, allowing the number of input features to remain fixed.

B. Action representation

Like the observation, the action space must retain a constant size in order to cooperate with a neural network. Further, it should be intuitive how the network output corresponds to a legal action. One approach to this problem, implemented by Xiao et al. [8] for the game *Hearthstone*, is to create a softmax distribution over all legal actions, zeroing out the probabilities for illegal ones in postprocessing. Another route is taken by Wang & Moh [9] in *Magic: the Gathering*, who trained a neural network to evaluate board positions and used it as an evaluation function in an MCTS implementation. However, these approaches don’t scale well with the large action spaces of these types of games: The former method suffers because it necessitates a massive output layer for increasingly complex games, while in the latter method, MCTS is hurt by a large branching factor and hidden information.

To approach this problem, we introduce a method for the agent to express its preference among different action qualities, which we describe here. As discussed above, any card in ToT can be expressed as a vector of dimension 61 that describes the integer arguments of its play effects. These effect vectors can also be generalized to describe legal actions. Specifically, the first 61 components of such a vector correspond to card attributes, which are interpreted as playing a card represented as such. The next 61 components correspond to buying a card in the same way. Finally, the last 6 components represent different Patron activations as well as Pass Turn. The vector assigns weights to each action attribute in this way, creating a fictitious action that can be easily compared with the vectors representing legal actions given by the gamestate. Such a vector is the output of the neural network. By taking the dot product of this “weighting” vector with vectors representing the actual legal actions, we arrive at scalars that encapsulate the agent’s preference to play a given action. This way, the space can

express an arbitrarily large number of moves without increasing network size, resulting in strong action space scalability.

This action formulation is novel in the context of card games. Similar methods have been used for drafting cards in *Magic: The Gathering* [10], but thus far, none have used it for choosing actions during the player-versus-player phase of a game. This method is beneficial because it allows the agent to evaluate all legal actions with only a single forward pass of the policy network, which sets it apart from state evaluation approaches like the one by Wang & Moh [9]. Furthermore, since the network output needs only to encode a single action preference vector, it scales well in games with very large and varied action spaces. The caveat is that, for this action formulation to work, the natural action space of the game must consist of actions that are easily represented as vectors representing their attributes.

a) *RL Reward Structure*: To create an RL agent for Tales of Tribute, we must define a reward function that the algorithm is trained to optimize. It is intuitive to simply define this as the agent’s win rate, or the percentage of games that it wins on average. However, in practice, this method was ineffective. With such a sparse reward—only one signal per entire game—the agent struggled to achieve any meaningful progress against any opponent more complicated than the framework’s RandomBot, which plays totally random moves. Through a series of early experiments, we found that a reward function of the following form allows for much more measurable progress:

$$Reward = \begin{cases} 0 + supplement & \text{Agent loses or ties} \\ 2 + supplement & \text{Agent wins} \end{cases}$$

$$supplement = \frac{1 - \exp[\frac{1}{5}(total\ Prestige)]}{1 - \exp[\frac{1}{5}80]}$$

V. EVALUATION

We train the agent using Kowalski et al.’s [1] *Scripts of Tribute* framework, which lets developers easily run many games of ToT, alongside OpenAI’s Stable Baselines 3 library [11] to provide an implementation of Proximal Policy Optimization. Early training experiments found that the agent learns fastest when it plays against opponents of comparable strength to itself, so we make use of multiple different training opponents at different stages of training:

- **RandomBot**: Plays moves entirely randomly. Extremely weak performance.
- **MaxPrestigeBot**: Plays the move that gives it the most Prestige on any given turn. Very aggressive playstyle.
- **BeamSearchBot**: Keeps track of k candidate actions and greedily searches their subgames for the heuristically best action.
- **Self-Play**: Loads a previous checkpoint of the proposed reinforcement learning agent. The selection is biased towards newer (on average better) models.
- **MCTSBot**: Searches through its own turn, then assigns a heuristic score to leaf nodes. This agent is the strongest and was only used for evaluation, not training.

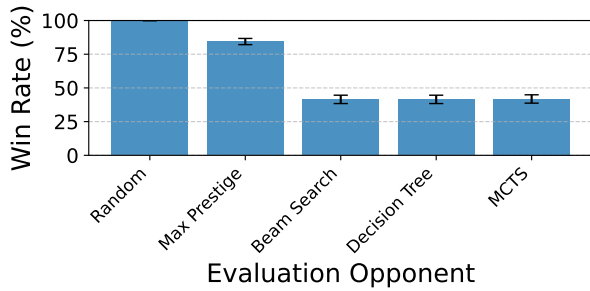


Fig. 2. Win Rates Against Baseline Agents. The vertical axis represents our agent’s win rate against five of the baseline bots, along with 95% confidence intervals for the true mean win rate. Overall, the proposed agent competes at a similar level to its search-based counterparts, achieving win rates slightly over 40% in most cases. Furthermore, it convincingly beats MaxPrestigeBot ($p < 0.001$) and wins all 1,000 games against RandomBot. As a result, we conclude that our agent competes with its more complex ToT-specialized opponents and defeats the less complex opponents convincingly.

All models shown in this paper were trained locally, on a laptop GPU, over the course of two weeks. The strongest model, used for evaluation, was trained for a total of 60 million timesteps—20 million of self-play, followed by 40 million against Beam Search. This constitutes a very small amount of computing power in comparison to studies like Xiao et al. [8]. Furthermore, the network architecture used was very small, with only two hidden layers of 64 neurons each.

After training, a total of 5,000 games of testing were run between our agent and a mix of RandomBot, MaxPrestigeBot, BeamSearchBot, MCTS, and Decision Tree (another strong search agent of the framework). For each of these opponents, we simulated 1,000 games with 500 games of our agent moving first, and 500 moving second. The proposed agent did not update its parameters during these evaluations. The results of this experiment are shown in Figure 2.

In the end, the agent achieves win rates around 40% against Kowalski et al.’s [1] strongest search-based algorithms. In addition, it easily defeats the simpler algorithms RandomBot and MaxPrestigeBot. Interestingly, despite losing on average against MCTSBot, the proposed agent beats Max Prestige more consistently than MCTS does: it won $84.40\% \pm 4.6\%$ of these matches, compared to MCTSBot’s $73.2 \pm 4.3\%$. These findings suggest that, while the agent does not outperform the strongest baseline agents, it can yield compatible performance with much reduced processing costs at run-time, and without the reliance on heuristic evaluations.

VI. DISCUSSION

We draw two main conclusions from our experiments. First, the agent consistently outperforms MaxPrestigeBot, despite being trained with a Prestige-based reward. If it had merely learned to greedily maximize Prestige, it would not surpass a bot designed to do exactly that. Its success ($p < 0.001$) suggests strategic behavior beyond reward shaping, likely favoring long-term gains over immediate rewards. This improvement only emerged after introducing shaped rewards, confirming their effectiveness.

Second, the use of preference vectors proves well-suited for large action spaces. This representation enables efficient training and evaluation of many actions with minimal computation, a critical advantage in CCGs. While MCTS performs better, having the possibility to train an agent to later run it without much additional computation requirements allows for integrating AI agents at a larger scale, e.g., as server-side opponents. To our knowledge, a preference vector-based selection remains largely unexplored in action selection for card games, aside from its use in drafting by Ward et al. [10].

VII. CONCLUSION AND FUTURE WORK

This study serves as proof-of-concept that reinforcement learning is effective in domains like ToT. Beyond its long-term strategic depth, the game’s unique deck building mechanics prevent reliance on a single strategy—players must adapt to different Patron decks and playstyles. Our results show that RL meets this challenge. In future work, we aim to explore applying similar strategies to deck-building or play in collectible card games like Magic: The Gathering or Hearthstone [12].

REFERENCES

- [1] Jakub Kowalski, Radosław Miernik, Katarzyna Polak, Dominik Budzki, and Damian Kowalik. Introducing tales of tribute ai competition. In *2024 IEEE Conference on Games (CoG)*, pages 1–8. IEEE, 2024.
- [2] Jakub Kowalski and Radosław Miernik. Summarizing strategy card game ai competition. In *2023 IEEE Conference on Games (CoG)*, pages 1–8. IEEE, 2023.
- [3] Jean. ESO Tales of Tribute Strategies — deltiasgaming.com. <https://deltiasgaming.com/eso-tes-of-tribute-strategies/>. [Accessed 06-05-2025].
- [4] Adam Cieżkowski and Artur Krzyżyński. *Developing Card Playing Agent for Tales of Tribute AI Competition*. Thesis, Uniwersytet Wrocławski, 2023. <https://jakubkowalski.tech/Supervising/Ciezkowski2023DevelopingCard.pdf>.
- [5] Alexander Dockhorn and Rudolf Kruse. Predicting cards using a fuzzy multisets clustering of decks. *International Journal of Computational Intelligence Systems (IJCIS)*, 13(1):1207–1217, August 2020.
- [6] Alexander Dockhorn, Tony Schwensfeier, and Rudolf Kruse. Fuzzy multisets clustering for metagame analysis. In *Proceedings of the 11th Conference of the European Society for Fuzzy Logic and Technology (EUSFLAT 2019)*, pages 536–543, August 2019.
- [7] Alexander Dockhorn, Max Frick, Ünal Akkaya, and Rudolf Kruse. Predicting opponent moves for improving hearthstone ai. In *17th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, IPMU 2018*, pages 621–632, 2018.
- [8] Changnan Xiao, Yongxin Zhang, Xuefeng Huang, Qinhan Huang, Jie Chen, et al. Mastering strategy card game (hearthstone) with improved techniques. In *2023 IEEE Conference on Games (CoG)*, pages 1–8. IEEE, 2023.
- [9] Dylan Wang and Teng-Sheng Moh. Hearthstone ai: Oops to well played. In *Proceedings of the 2019 ACM Southeast Conference*, pages 149–154, 2019.
- [10] Henry N Ward, Bobby Mills, Daniel J Brooks, Dan Troha, and Arseny S Khakhalin. Ai solutions for drafting in magic: the gathering. In *2021 IEEE conference on games (CoG)*, pages 1–8. IEEE, 2021.
- [11] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.
- [12] Alexander Dockhorn and Sanaz Mostaghim. Introducing the hearthstone-ai competition. *arXiv:1906.04238*, pages 1–4, May 2019.