Implementing an agent with reinforcement learning elements for Scripts of Tribute card game

(Implementacja agenta z elementami uczenia przez wzmacnianie do gry karcianej Scripts of Tribute)

Rafał Stochel

Praca inżynierska

Promotor: dr Jakub Kowalski

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

2 września 2024

Abstract

The main idea for writing the thesis was to test machine learning methods in the card game environment. The "Scripts of Tribute" project was used for this purpose. It is an implementation of the "Tales of Tribute" minigame from "The Elder Scrolls Online" that allows you to write your own agents. Using artificial intelligence solutions, the computer program should obtain the best possible results during the game.

Created agent was submitted to the Tales of Tribute AI Competition (TOTAIC), held as part of the IEEE Conference on Games 2024 (IEEE CoG). It managed to take second place. Following text describes rules of the game, TOTAIC competition, designed program and its results.

Głównym zamysłem na napisanie pracy było przetestowanie metod uczenia maszynowego w środowisku gier karcianych. Użyto do tego celu projektu „Scripts of Tribute". Jest to implementacja minigry „Tales of Tribute" z produkcji „The Elder Scrolls Online" umożliwiająca pisanie własnych agentów. Przy użyciu rozwiązań z obszaru sztucznej inteligencji program komputerowy ma uzyskać jak najlepsze rezultaty podczas rozgrywki.

Stworzony agent został wysłany na zawody Tales of Tribute AI Competition (TOTAIC), odbywające się w ramach konferencji IEEE Conference on Games 2024 (IEEE CoG), plasując się na drugim miejscu. Poniższy tekst opisuje zasady gry, konkurs TOTAIC, stworzony program oraz uzyskane przez niego wyniki.

Contents

Chapter 1

Introduction

In recent years we have seen an increase in the achievements of artificial intelligence in playing games of all kinds, starting with the important event of constructing a Deep Blue [1] computer system by IBM. It was a machine that could play Chess at a master level. In 1996, a match was held between the reigning world champion Garry Kasparov and Deep Blue, which ended with a score of 4–2 for the human player. One year later, in 1997 there was a rematch, but this time computer player was the winner. In those days, it was a groundbreaking event that showed possibilities of artificial intelligence. Another significant program was AlphaGo [2] by DeepMind – board game Go agent. During *Future of Go Summit* event in 2017 it defeated Ke Jie, a number one ranked player in the world. Go is considered even more complicated for computers than Chess, due to the larger number of possible games states and a more abstract evaluation of the board.

The next complex problem approached by DeepMind was StarCraft II. This is a real–time strategy game with a theoretically infinite set of options to make. Resource management, buildings placement, selecting the appropriate strategy to deal with the enemy, micro management of units, all of these aspects had to be done by the machine. AlphaStar [3] was presented in 2019 and it was able to meet these requirements. What is more, it managed to beat top tier professional human players.

Card games seem like a great area to verify the capabilities of artificial intelligence. A computer program faces problems such as reasonable deckbuilding, searching through multiple game states, evaluation of current situation and predicting the opponent's possible moves. To test these ideas the Script of Tribute [4] project and related tournament – Tales of Tribute AI Competition [5] were created. They allow you to create your own bots and then compete with other programs. This paper describes an agent designed to check how reinforcement learning methods can help establish a good evaluation function, providing a better result in clashing with other programs.

Chapter 2

Scripts of Tribute

Scripts of Tribute (SoT) is an implementation of card minigame Tales of Tribute from The Elder Scrolls Online MMORPG. Project was done using C# programming language and .NET platform. Gameplay focuses on a duel between two players. The first person to score a certain number of in-game points wins.

2.1 Game rules

Game cards are splitted into eight decks, which are named after their Patrons. Before the game starts, players have to choose which decks will be available during the play (figure 2.3). Treasury deck can't be selected and is always available in the game. These decks will create a Tavern. This is a set of possible cards to buy during the play. Then the turn–based duel starts.

In general turn looks like this: five cards are added to the hand of player from the Draw Pile. If Draw Pile is empty, then Cooldown Pile cards are shuffled and moved into Draw Pile. Player can play Action and Agent cards from the hand. Alive Agents can be activated. Then player decides whether to spend resources on purchasing new cards (sometimes it is not worth to get anything) or on calling patrons (four selected before the game + Treasury). Tavern can only have 5 possible options at a time. Bought cards are replaced with new ones. Lastly, Power points can be used to attack enemy Agents. Figure 2.1 describes all the main objects in the game. Another example of gameplay is shown in figure 2.4.

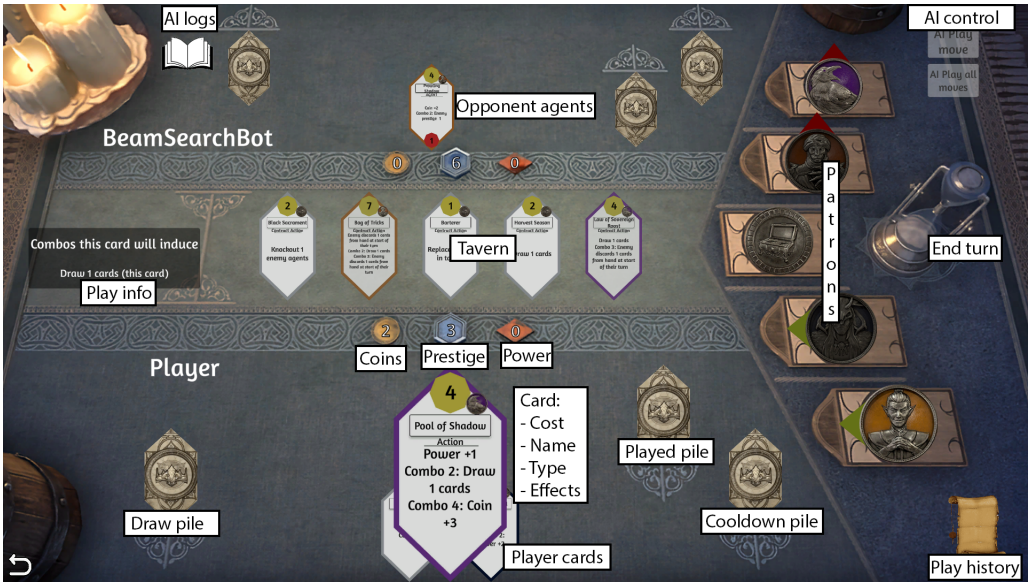

Figure 2.1: Scripts of Tribute GUI screenshot with described elements

There are three resources in the game:

- Coins – basic currency. With Coins we can call some Patrons and buy cards from the Tavern, which will extend our current deck. All gained Coins are cleared at the end of turn.
- Power – with Power we can call some Patrons or attack enemy Agents, possibly remove them from the board and move them to the Cooldown Pile. Remaining Power is converted to Prestige at the end of turn.
- Prestige – winning condition resource. Winner is determined based on these points. Usually obtained from Power. Sometimes can be gained directly from played cards or Patrons.

Each competitor starts with 10 cards in the deck, which are called Starters. They are mainly used to gain Coins. Cards are divided into four groups:

- Action – its effects are generated after it is played. Goes to the Played Pile after purchase. When turn is over, the Played Pile is moved to the Cooldown Pile.
- Agent – card that can be placed on the board. Each round, if Agent is alive, player can activate it, to perform its actions. Goes to the Cooldown Pile after purchase and after losing health points.
- Contract action – is played instantly after buying. Goes to the Discard Pile after playing. Cards remain on the Discard Pile to the end of the game.
- Contract agent – is placed immediately on the board after buying. Goes to the Discard Pile after losing health points.

Patrons are a significant tactic of the game. Every Patron except the Treasury, which is always neutral, can be in one of three states: favors player, neutral, favors enemy. By calling them, player can trigger various effects depending on the current state of the Patron. Activating it also changes the Patron's state to one that is more favorable to the player (usually by one step). Brief description of the available Patrons:

Patron	Activation cost	Effect
Ansei	2 Power	Gain 1 Coin and gain 1 Coin at the start of your turn if favored by Ansei
Duke of Crows	All Coins	Gain *Coins - 1* Prestige, can be activated only if not favored
Hlaalu	Sacrifice 1 card in play	Gain *Cost of sacrificed card - 1* Prestige
Orgnum	3 Coins	Gain Power based on number of owned cards
Pelin	2 Power	Move 1 Agent from your Cooldown Pile to Draw Pile
Rajhin	3 Coin	Add empty card to the opponent's deck
Red Eagle	2 Power	Draw 1 card
Treasury	Sacrifice 1 card in play	Add Writ of Coin card (+2 Coins effect) to your Cooldown Pile, this Patron is always neutral

Another important aspect of the game is Combos. When multiple cards from one deck are played during one turn, they can produce additional effects. Tracking your own and enemy's possible combinations is crucial to winning the duel.
Player can win the game in one of three possible ways:

- Get favor of all four deck Patrons.
- Gain 40 or more points of Prestige. The game will now enter "sudden death" mode. If any player has less Prestige points than his opponent at the end of his turn, he loses the game.
- Gain 80 or more points of Prestige. After that, the game immediately ends.

More detailed description of rules can be found on the Tales of Tribute fan-made sites[1].

[1] `https://eso-hub.com/en/guides/tales-of-tribute-guide`

2.2 SoT GUI examples

SoT project provides an application with graphical interface. It is incredibly helpful during fixing and improving the agent (figure 2.5). Program can be also used to practice your own Tales of Tribute skills. When selecting the opponent, we can also specify the game seed and the time the agent has to complete a turn (shown in figure 2.2).

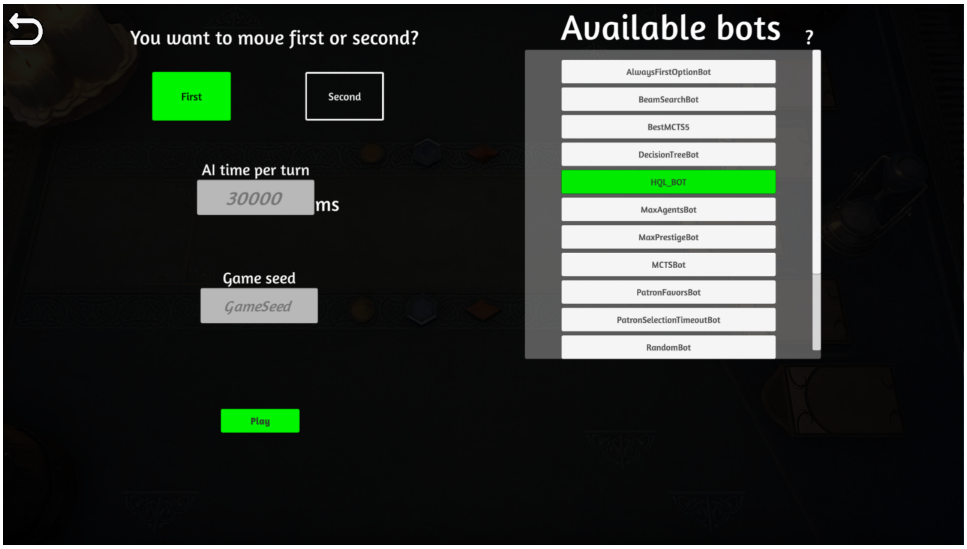

Figure 2.2: After choosing an option to start, we have to pick our enemy

Figure 2.3: Selecting patrons

Figure 2.4: Example gameplay

Figure 2.5: Logs are a great way to debug written bot

2.3 How to create an agent

To write a bot user has to create a new class that inherits from the provided "AI" class. The game engine initializes the instances of two bots that are going to clash. Then, at certain moments, it calls functions on both instances. An agent has to implement the following three methods:

- `PatronId SelectPatron(<PatronId> availablePatrons, int round)`
  Each new game begins with a call to this method. The agent has to select which Patron will be available during the game from the list of available Patrons.

`Round` describes when the method was called (four choices in total).

- `Move Play(GameState gameState, List<Move> possibleMoves, TimeSpan remainingTime)`
  An essential part of any agent. From the list of possible moves the bot should return one to play. The `GameState` object describes the current state of the board. The time spent on move selection should not exceed the value of the `remainingTime` variable. The game is lost if illegal move is returned.

- `void GameEnd(EndGameState state, FullGameState? finalBoardState)`
  Auxiliary function used to debug bot's behavior based on `EndGameState` and `FullGameState` objects. Gets called after the game is over.

When the program is done, we can test it in two ways: in the GUI described earlier or using `GameRunner` command line interface, shown in the figure 2.6. Basic command to run one game:
`GameRunner.exe BotA BotB`.

There are also additional flags to modify testing:

- `--n X` – run X games
- `--s X` – run game with specified seed
- `--t X` – run games on X threads

```
.\GameRunner.exe DecisionTreeBot MaxPrestigeBot -n100
Running 100 games - DecisionTreeBot vs MaxPrestigeBot

Initial seed used: 29864269499474592 58
Total time taken: 3803ms
Average time per game: 38,03ms

Stats from the games played:
Final amount of draws: 0/100 (0%)
Final amount of P1 wins: 84/100 (84%)
Final amount of P2 wins: 16/100 (16%)
Ends due to Prestige>40: 99/100 (99%)
Ends due to Prestige>80: 0/100 (0%)
Ends due to Patron Favor: 1/100 (1%)
Ends due to Turn Limit: 0/100 (0%)
Ends due to other factors: 0/100 (0%)
```

Figure 2.6: GameRunner output

Chapter 3

Agent implementation

The agent was written in C# (just like the original project), although the latest framework update allows the use of other programming languages. Created bot consists of three main parts to select the best move: searching through game states with Beam Search algorithm, usage of Q–learning method to determine the value of card, evaluating given state with a heuristic function. Patrons are selected randomly. The idea behind agent's implementation was to check whether a machine learning can be helpful in rating cards.

3.1 Beam Search algorithm

When agent is given multiple moves options in `Play()` method it must somehow decide, which one is the best. The game engine provides function `ApplyMove(Move move, ulong seed)`, to simulate the execution of a move. It has to be called by the instance of `GameState` or `SeededGameState` type. It returns a new game state and a list of new possible moves. Seed as an optional argument is used to shuffle new cards for the Tavern and the Draw Pile.
With this method we can check what effects each move will have. However, cards in the Draw Pile are placed randomly. Because of this, it is difficult to predict the opponent's moves, as we do not know exactly what hand he will have in his turn. Designed agent only searches through its own turn's options.

Some cards, when activated, create a choice for the player. For example replace n Tavern cards or move n cards from the Cooldown Pile to the top of the Draw Pile. Such actions can lead to a remarkable growth of the number of states. To overcome this problem, Beam Search[1] algorithm has been chosen as the search method. We want to keep only k nodes with the highest evaluation score. Without it, we might have exceeded the time limit for playing the move. The final node limit was selected by testing, more in the tests section.

[1] `https://en.wikipedia.org/wiki/Beam_search`

Additionally, to reduce the number of states to check, we can play "no choice cards" (a similar approach is described in [6]). These are simple moves that do not create any decisions to make and can be played instantly at the start of the turn. Of all 101 cards available in the game, 43 were marked as "no choice cards".
The agent plays moves in a specific order depending on their type:

- At first no choice cards.
- Then, the remaining Action and Agent cards that can create choice.
- Finally, remaining moves are evaluated, like buying new cards, attacking Agents, calling Patrons.

Beam Search procedure adapted for the SoT can be summarized using the following pseudocode:

1. Declare empty lists: *beamNodes*, *endNodes*.
2. Create nodes from given data (game state and list of possible moves) and add them to *beamNodes*.
3. While *beamNodes* is not empty:
   (a) Declare empty list: *childrenNodes*.
   (b) Iterate over *beamNodes*: expand the given node (by `ApplyMove()` function) and visit its children. If child is a final node (can't be expanded – has no child nodes) add it to the *endNodes* with calculated heuristic evaluation, otherwise add it to the *childrenNodes*.
   (c) Keep k (a predefined limit) nodes in *childrenNodes* with the best heuristic results. The remaining nodes are discarded from the list.
   (d) *beamNodes* = *childrenNodes*
   (e) Simplify nodes – iterate over *beamNodes*: play "no choice cards" on node's game state.
4. Iterate over *endNodes*, return move from the element with the highest heuristic score.

3.2 Heuristic

A crucial part of the agent. After reaching the end node in the search procedure, we need to know whether the current situation is favorable for us. To do this, various aspects of the board are considered. All parameters are multiplied by appropriate weights, to focus on the more significant elements of the game, e.g. Prestige is way more important than Coins during late game. The program calculates values

separately for the player and the opponent. Then, the final score for a given state is calculated as:

$$stateScore = playerScore - enemyScore$$

During the search procedure, every final state (where the only possible move is to end the turn) is assigned a heuristic value. The agent chooses final state with the highest heuristic score and plays move, from which this state originates.

1. Stage of the game
   The first thing the agent does is determine the current phase of the game. In the early turns, we want to focus on deck building. As the game progresses, Prestige becomes more valuable, as it is a condition for winning. For each stage a set of weights was prepared, based on my own gaming experience and tweaking the values during testing the agent.

   Formula to calculate current stage:

Stage	*turnCount* + *enemyPrestige* intervals
Start	[0, 10)
Early	[10, 20)
Middle	[20, 20)
Late	[30, 30+)

2. Cards and Board
   Things considered when calculating the score:

   - Coins, Power, Prestige
   - Number of cards from each deck. The higher the number, the greater the chance of making a card combination.
   - All cards in the deck and Agents that are active on board. To rate them, the value from Q–learning algorithm is used. More details in the Q–learning section. The Agent's score also depends on its health points.
     Extra points for up to 3 cards from the top of the Draw Pile (3 is the maximum number of cards that can be moved there from the Cooldown Pile).
     There is a small penalty for having starter-type cards in the late game. These cards yield only 1 Coin per play, which is not very useful at this point.
   - Cards available in the Tavern, that can be purchased by the agent.

3. Patrons
   Overall, the activation of Patrons is rewarded in itself, but some of them can have additional effects depending on the phase of the game:

   - Ansei – Coins earning Patron, only rewarded in the *Start* and *Early* phases.

- Duke of Crow – exchanges money for Prestige and can only be activated when it is neutral or favors the opponent. For this reason it should be played at the end of the game, to maximize its effect. For the *Start* and *Early* it is assigned huge negative values.
- Orgnum – gives Prestige for the number of cards the player owns, preferred during the *Late* game.

In addition, the player receives a penalty that increases with the number of activated opponent's Patrons.

3.3 Q–learning

When choosing parameter weights for the evaluation function, we rely on our own intuition. However, this can be tricky with cards, as they have long term effects on the game. To deal with this, we can delegate the calculation of the values to the computer. Evolutionary algorithms are often used for this purpose [7], [8]. Another idea is to use machine learning.

Card games are characterized by a high degree of randomness. In Scripts of Tribute, we only know what decks are available at the beginning of the game. The Draw Pile and the Tavern cards are prepared randomly each time. Therefore it it would be difficult to prepare a data set for training the agent. That's why Q–learning[2] was chosen, because it is a model-free algorithm. Due to the mentioned randomness, the probability of moving from one state to another is not taken into account. The agent collects data during the game and learns from it.

3.3.1 State, action and reward

The general idea of this algorithm is to track the performance of the agent in every possible state of the game. First, we need to define what exactly is the state of the game for our bot. Cards have clearly described effects. Their final outcome depends on the currently possible combinations with other cards. Since we want to measure the strength of cards, the state was defined as the number of cards in a given deck. Possible state values are {0, 1, ..., 19}. If the number of cards is higher, it is reduced to 19 (this only happens if our deck is flooded with Curse cards). Then, the action is to play a specific card. The reward function is used, to check how good this move was. In general, all effects caused by the card in a single turn are scored. More precisely, the following things are counted in the `ScoreForCompletedEffect()` reward function: Coins, Power and Prestige gained; cost of a card acquired from the Tavern; opponent's lost Prestige; cost of a destroyed card; points for drawing a card; points for opponent's discarding a card; cost of a card moved to the top of the

[2] `https://en.wikipedia.org/wiki/Q-learning`

Draw Pile; added Patron calls; created bonus card (a specific card from the Orgnum deck). Combining 20 possible state values and 113 game cards, we get a table with 2260 cells, that were initially filled with the number 5. This value resulted in fast learning time and good results. The agent updates the Q–table based on the cards played each turn.

3.3.2 Q–learning formula

Main formula to calculate new Q–values:

$$Q_{new}(S, A) = (1 - \alpha) \cdot Q(S, A) + \alpha \cdot [R(S, A) + \gamma \cdot \max_{a'} Q(S', a')]$$

Where variables are:

- S – state S of the game. The number of cards owned from a specific deck. A card from this deck will be played.
- A – action A that will be executed. Activated Agent or Action played during the turn.
- $R(S, A)$ – reward function to check how well the action A performed in the state S. Calculated as the sum of `ScoreForCompletedEffect()` function calls for each card's effect.
- α – learning rate. Value from the interval $[0, 1]$. Determines how fast the agent should learn, i.e. how much the old value is overwritten by the new one. Agent with $\alpha = 0$ does not learn at all, it relies on the knowledge it already has. When $\alpha = 1$, old values are discarded after each update.
- γ – discount factor. Value from the interval $[0, 1]$. If $\gamma = 0$, potential future rewards are not considered, only current situation is important. An agent with $\gamma = 1$ focuses on long-term strategy.
- $\max_{a'} Q(S', a')$ – maximum reward that can be obtained from the new state S' after action A, iterating over every possible action in S': a'.
  Normally, we get a new state after performing an action. However, in the current state–action design a new game state is reached by getting a card from the Tavern. Possible actions are represented as cards available in the Tavern, that come from the same deck as the played card. S' is is then defined as $S + 1$ (because the Tavern card was acquired).

3.3.3 Q–values examples

Example card and its values calculated by the Q–learning algorithm. All numbers in the table are truncated to two decimal places. Based on observations, sometimes

there are duels where the agent has even 30 cards at the end of the game. In such extreme situations it is difficult to achieve card combinations. The created agent does not take into account the total number of cards in the Q–learning algorithm, so the effects of the cards may begin to decrease in the late game. Limiting the purchase of new cards is an idea for future optimization. Sample card and its Q–value:
Midnight Raid, Action card, Red Eagle deck, Cost=4, effects:

- Activation: gain 3 points of Power.
- Combo 2: gain 2 points of Power.

State	Q–value
0	11.76
1	12.32
2	11.27
3	14.97
4	18.37
[5, 19]	5

Apparently, more than 4 cards from the Red Eagle deck were not reached when Midnight Raid was in play, so 5+ states were not visited. The default integer value of 5 is still present in Q–table cells. An incorrect number can be seen in the state=2. In this situation, the value should be greater than the previous state value and less than the next one. For more cards, see appendix A.

Chapter 4

Agent testing

4.1 Q–learning general tests

The most important test is what results the algorithm gets. The simplest way to check this is through duels with other agents. Four bots clashed and each match consisted of 1000 games:

- HQL_BOT – created agent.
- HQL_BOT SoT card tier – the card values from the Q–table were replaced with card tiers from the SoT project. Each card was assigned a tier.
  Original SoT card tiers:
  S = 50, A = 25, B = 10, C = 5, D = 1
  They have been scaled down to more closely match the other weights of the HQL_BOT's heuristics (more in appendix B):
  S = 10, A = 6, B = 3, C = 2, D = 1
- HQL_BOT card_value=1 – each card is assigned a value of 1.
- DecisionTreeBot – bot from Scripts of Tribute, added for reference.

vs	HQL_BOT	HQL_BOT SoT card tier	HQL_BOT card_value=1	DecisionTree Bot
HQL_BOT	-	73.3%	92.9%	78.6%
HQL_BOT SoT card tier	26.7%	-	79.3%	73.6%
HQL_BOT card_value=1	7.1%	20.7%	-	22.4%
DecisionTree Bot	21.4%	26.4%	77.6%	-

4.2 Rate of learning

This section shows the win rate and the learning process over the number of games played. The agent started dueling with the Q–table cells filled with the default value of 5. The learning rate and discount factor were set to $\alpha = 0.1$, $\gamma = 0.8$. DecisionTreeBot was chosen as the opponent. Results were measured after every 500 games and are shown in figure 4.1.
On the other hand, if we use 0 as the default value, the agent learns much slower. Now, the results were measured after every 1000 games – figure 4.2.

4.3 Beam Search performance

The agent again competes with DecisionTreeBot. In each test, the node limit of the Beam Search was changed (figure 4.3). Every match consisted of 500 games. As expected, as the number of maintained nodes increases, so does the computation time. However, the percentage of games won remains almost the same. We can assume that the initially best node will retain the best result after simulating the remaining possible moves.

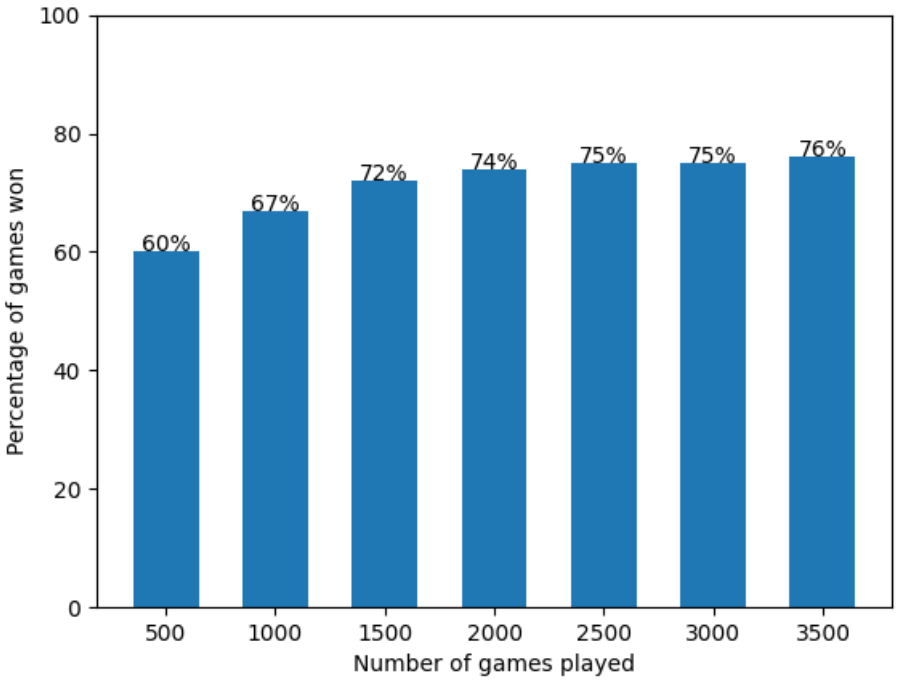

Figure 4.1: Initial Q–value = 5

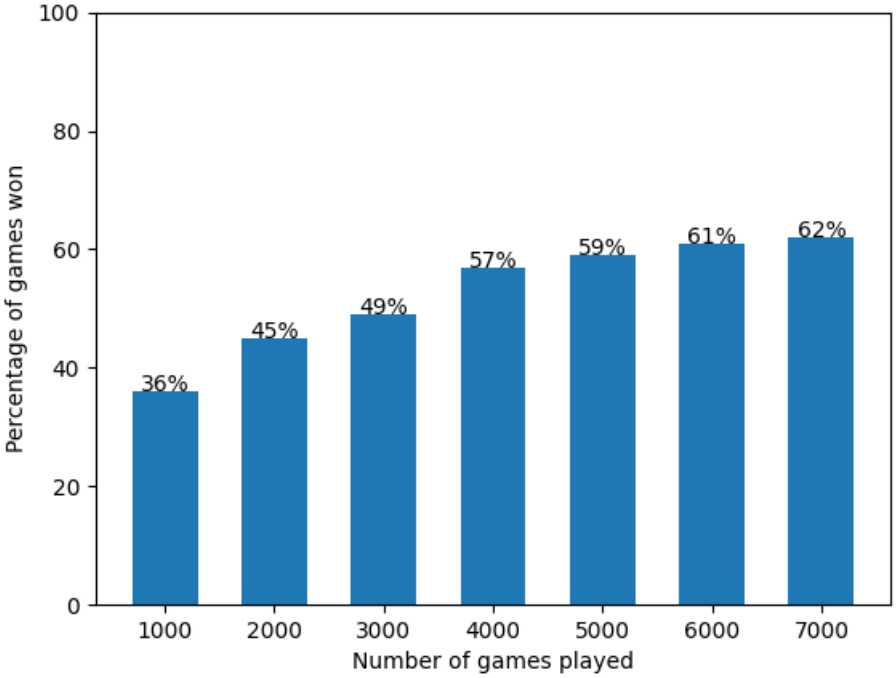

Figure 4.2: Initial Q–value = 0

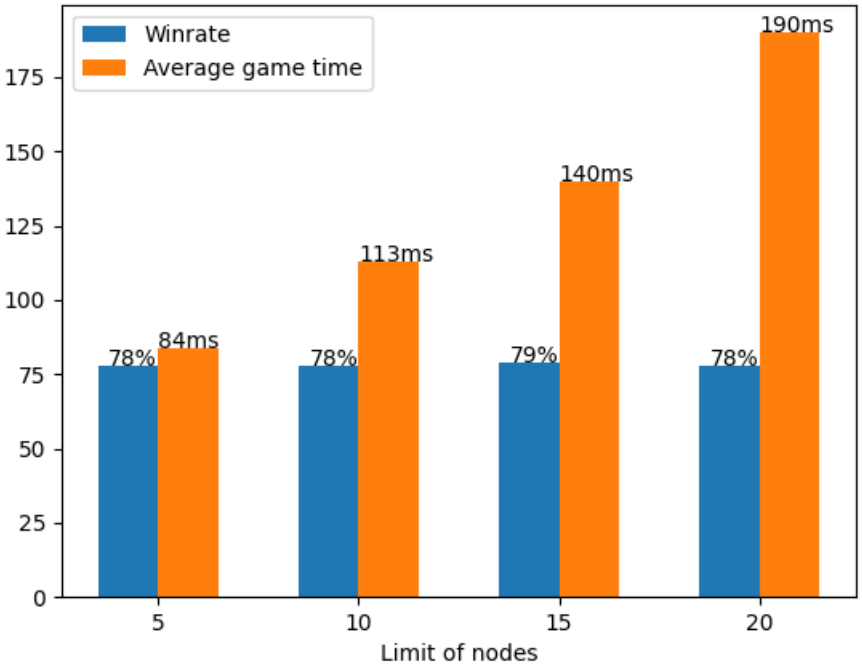

Figure 4.3: Node limits comparision

Chapter 5

Tales of Tribute AI Competition

Five teams took part in the 2024 edition of the competition by submitting one agent each. The tournament was played in a round-robin format. Duels between two bots consisted of about 600 games. HQL_BOT (Heuristic Q–learning Bot) presented in this thesis, came in second place. BestMCTS3 and BestMCTS5, which won the last year, are described in [6]. Both are variations on the general idea of Monte-Carlo Tree Search[1]. The next edition of the competition is planned for 2025.

In the version of the HQLBOT that was sent to the contest, the Q–table was initially filled with the number 5. It has been trained by playing most of the time against DecisionTreeBot. The number of training games was approximately 20,000. Although based on the previous section, we can see that after about 3500 games, the score doesn't change that much.

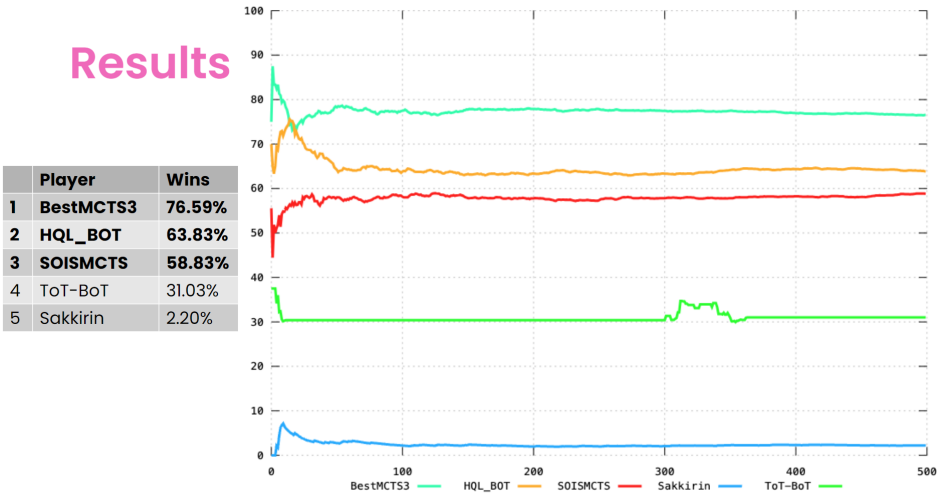

Figure 5.1: Winrate of each agent

[1] https://en.wikipedia.org/wiki/Monte_Carlo_tree_search

Detailed Results

vs.	**BestMCTS3**	**HQL_BOT**	**SOISMCTS**	ToT-BoT	Sakkirin
BestMCTS3		67.75%	62.07%	96.25%	98.72%
HQL_BOT	32.25%		60.34%	83.13%	97.84%
SOISMCTS	37.94%	39.66%		75.64%	97.92%
ToT-BoT	3.75%	16.87%	24.36%		80.77%
Sakkirin	1.28%	2.16%	2.08%	19.23%	

Figure 5.2: Result of each duel

Chapter 6

Conclusions

This paper presents the implementation of an agent for the card game Scripts of Tribute. The motivation was to test reinforcement learning methods. At the beginning, the rules of the game and how to create new agents are explained. It then describes how the created agent searches through multiple game states with the Beam Search method and evaluates them using a heuristic function. Later it focuses on the idea of the Q–learning algorithm, how it was adapted to the requirements of the game and how it improved the evaluation function. The last part contains the agent's tests and results, along with the summary of the Tales of Tribute AI Competition, where the described agent achieved second place.

Q–learning has proved to be helpful. Although the agent achieved decent results, many things can be done better, e.g. improving the weights of the heuristic function or improving the calculation of Q–values. The next thing worth trying is to use a different machine learning method, such as deep reinforcement learning with the ability to make major decisions during the game.

Bibliography

[1] M. Campbell, A. J. Hoane, and F. Hsu, “Deep Blue”, Artificial intelligence, vol. 134, no. 1, pp. 57–83, 2002.

[2] D. Silver et al., “Mastering the game of Go with deep neural networks and tree search”, Nature, vol. 529, no. 7587, pp. 484–489, 2016.

[3] M. Mathieu, S. Ozair, S. Srinivasan et al. ”AlphaStar Unplugged: Large–Scale Offline Reinforcement Learning”, arXiv:2308.03526.

[4] D. Budzki, D. Kowalik, and K. Polak, ”Implementing Tales of Tribute as a Programming Game”, Engineer’s thesis, University of Wrocław, 2023.

[5] J. Kowalski, R. Miernik, K. Polak, D. Budzki and D. Kowalik, ”Introducing Tales of Tribute AI Competition”, IEEE Conference on Games, pp. 1–8 (2024)

[6] A. Ciężkowski and A. Krzyżyński, “Developing Card Playing Agent for Tales of Tribute AI Competition”, Engineer’s Thesis, University of Wrocław, 2023.

[7] J. Kowalski and R. Miernik, “Evolutionary Approach to Collectible Card Game Arena Deckbuilding using Active Genes”, IEEE Congress on Evolutionary Computation (CEC), 2020, pp. 1–8.

[8] S. J. Bjørke and K. A. Fludal, ”Deckbuilding in Magic: The Gathering Using a Genetic Algorithm”, Master’s thesis, NTNU, 2017.

Appendix A

Q–values

Murder of Crows, Action card, Duke of Crows deck, Cost=4, effects:

- Activation: gain 1 Coin.
- Combo 2: gain 2 Coins and 2 points of Power.
- Combo 3: gain 2 points of Power.

State	Q–value
0	5.87
1	7.66
2	7.60
3	6.97
4	11.15
5	10.47
6	7.43
7	6.88
8	5.55
[9, 19]	5

As the number of all cards increases (we can assume this because the agent keeps buying cards), it becomes difficult to achieve better effects with card combinations. That is why the values decrease from state=5.
Ansei Assault, Action card, Ansei deck, Cost=9, effects:

- Activation: gain 5 points of Power OR get a card from the Tavern that costs up to 9 Coins.

State	Q–value
0	5
1	5.95
[2, 19]	5

The agent was only able to check one state for a card that was as expensive as the Ansei Assault card.
Storm Shark Wavecaller, Agent card, Orgnum deck, Cost=5, effects:

- Activation: gain 2 points of Power.
- Combo 2: remove up to 1 card from the Tavern.

State	Q–value
0	4.9
1	5.16
2	5.06
3	5.06
4	5.60
5	4.88
6	6.75
7	4.16
8	5
9	4.20
10	4.20
[11, 19]	5

Orgnum deck offers the opportunity to obtain bonus cards. For this card, even State=10 has been visited.

House Embassy, Action card, Hlaalu deck, Cost=8, effects:

- Activation: gain 7 Coins.
- Combo 2: get a card from the Tavern that costs up to 7 Coins.

State	Q–value
0	5
1	15.97
2	13.03
3	15.67
4	15.08
5	15.74
6	12.28
7	6.88
[8, 19]	5

Another expensive card, but this time it was more preferred by the agent.

Appendix B

Heuristic weights

Table B.1: HQL_BOT heuristic weights

Parameter \ Stage	Start	Early	Middle	Late
Prestige	15	16	20	20
Power	14	15	19	19
Coins	2	1	1	1
Patron	18	18	18	18
Card (having more cards is beneficial)	5	5	4	4
Tavern card	1	1	1	1
Draw Pile card	1	1	1	1
Curse	-3	-3	-3	-3
Combo	4	4	4	4
Active agent	12	12	12	12
Agent's health points	4	4	4	4
"Gold" card	0	-2	-4	-6
Ansei	3	2	1	1
Duke of Crows	-1000	-1000	0	20
Orgnum	1	1	3	5