AI Project Report: Hearts Game

Yifan Cai, Junxian Guo, ZhiHang Wang, MinXiang Shen, etc

January 26, 2024

1 Introduction

1.1 Rules of Hearts

Microsoft Hearts is a classic trick-taking card game that can be played by four players with a standard 52-card deck, excluding the jokers. The deck is divided into four suits: hearts, diamonds, clubs, and spades, with hearts being the primary focus of the game. Each heart card is worth one point and the Queen of Spades is worth thirteen points. The final goal is to accumulate the fewest points during the game which will end after one of the player reaches a goal point such as 100 points.

Obviously, each player will receive thirteen cards at first, and 13 rounds make up one turn, the game will keep on going unless in one turn, one player reaches more than the goal points (e.g. 100). The values of the cards are in order of 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, and A. Firstly, select the dealer (i.e. the first player to play), and the dealer must first select three unwanted cards to pass to another opponent. For the order, one should pass the card to the player on the left side in the first turn, pass to the player on the right-hand side in the second turn, pass to the player sitting opposite in the third turn and no pass in the fourth turn, and so on. After receiving a pass from the dealer, the corresponding player also needs to pass three cards to the dealer.

There are also three special rules during the game: firstly, player who catches Plum Blossom 2 must first play Plum Blossom 2. Secondly, you cannot play a heart or spade Q in the first round of the game. Finally, Only after having played a heart in the previous round can one play a heart first in the current round (unless you only have one suit of heart left in your hand).

The players will play the cards in a clockwise direction. Each player must follow the same suit of cards in sequence. If there are no cards of the same suit as the deal, any card can be played. The player with the largest point in the same suit will win this round and receive all the cards in this round. The winning player will play the cards first in the next round. After 13 rounds are finished, the player with highest score will become the dealer in the next turn.(our rule is strictly base on `https://baike.baidu.com/item/%E7%BA%A2%E5%BF%83%E5%A4%A7%E6%88%98/875494?fr=ge_ala`, 百度百科: 红心大战. so, you can read that as well)

1.2 Reason to choose the topic

Our selection of the Hearts game theme aligns seamlessly with the knowledge acquired during our coursework. Classic algorithms, namely Monte Carlo, Q-learning, greedy, and random (baseline), were employed to execute the game.

Our investigation revealed that existing AI projects online predominantly focus on elementary games such as 2048, Flappy Bird, MountainCar-v0, and Snake. The reinforcement learning techniques employed in these projects primarily concentrate on foundational aspects of exploration and exploitation. In contrast, Hearts, a confrontational game involving four players, introduces a novel challenge due to its nature as an incomplete information game where opponents' hands and playing styles remain unknown. Our algorithm is compelled to infer potential decisions of adversaries based solely on the cards played and the composition of our own hand. This gives rise to a notably intricate decision tree, dynamic alterations in the gaming environment, and elevated coding exigencies. It is worth noting that due to the complexity of Hearts, we encountered a lack of open source materials online, so most of the code implementation and algorithm realization in this article came from the innovative writing of the group.

Our team endeavors to leverage this project as an opportunity for self-imposed challenges and innovation. A distinctive facet of our innovative work lies in the application of the Q-learning algorithm to the Hearts game. Our investigation unearthed a scarcity of instances where Q-learning has been implemented in strategic card games, possibly due to the intricate state-space and diverse set of possible actions inherent in such games. However, our team has successfully surmounted these challenges through the abstraction of actions and the development of distinct state classes, thereby presenting a novel and creative solution to this task.

1.3 Reference and Team Innovation

Due to the high difficulty of Hearts, it was challenging to find reference materials online. Our code framework was sourced from `https://github.com/pjgao/MonteCarloHeartsAI`, which is an unfinished codebase, only defined the rules of Hearts and provided a sample of the Monte Carlo algorithm. It is crucial to note that the Monte Carlo algorithm in this code does not adhere to the game rules. In other words, its decision design does not align with Hearts rules, resulting in incorrect card plays. The conclusions drawn from the original code are therefore not reliable. Nevertheless, it served as a reference for understanding the Monte Carlo concept, and we made extensive modifications and debugging.

In addition to this, we independently made various contributions: 1. We reconstructed the Monte Carlo algorithm, incorporating Hearts rules into the algorithm to ensure correct card plays.

2. We expanded our algorithms to include Q-learning (core formula from class PPT), greedy (decision-making based on gaming techniques from BaiduBaike), and random (baseline for assessing algorithm performance). These three algorithms were applied to the other three players, enhancing opponent intelligence and increasing the difficulty of the game.

3. We implemented both text-based (terminal) and graphical (pygame) outputs. After each round, we printed the card plays and remaining hand information for each player in the terminal.

Additionally, using the pygame library, we achieved a visualized graphical interface, enhancing the user experience (see Figure).

4, The original Monte Carlo assumes that all other players use the Monte Carlo algorithm by default. During the execution, we introduce hyperparameters that allow the algorithm to adjust based on the actual game. When leading by a significant margin in scores, it tends to assume that the opponents are using naive algorithms, with the aim of applying the concept of reinforcement learning, which estimates actual situations during the exploration process.

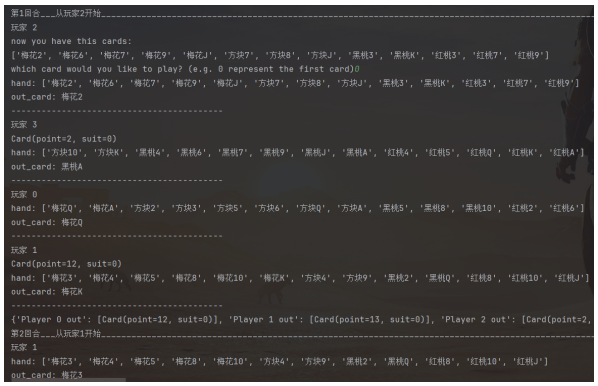

Figure 1: Human VS AI

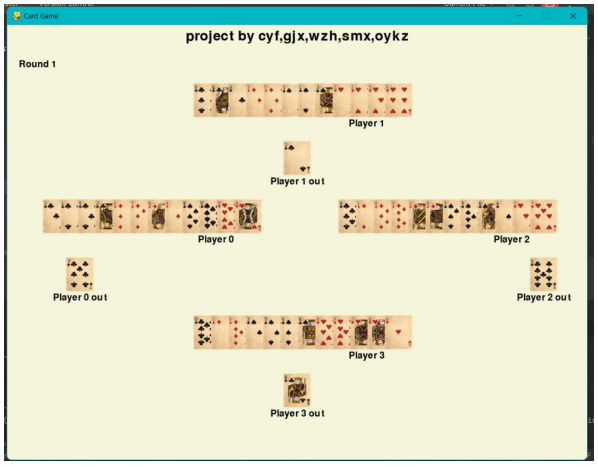

Figure 2: The graphical output.

2 Methodology

2.1 Greedy Algorithm

Our algorithm is designed to follow a strategy based on the current hand situation, aiming to "play high cards whenever possible while avoiding taking points." The algorithm categorizes into three scenarios based on the order of card plays:

First to play: If the red hearts haven't been played yet, and the smallest card in hand is not a red heart, prioritize not playing a red heart. If the Queen of Spades hasn't been played

yet, play the smallest Spade. If the Queen of Spades has already been played, choose the highest card.

Not the last to play: Based on the highest rank of the cards played in the current round, if you don't have a card of that suit, prioritize playing the Queen of Spades. If you do have a card of that suit, follow the game rules to choose your card, avoiding playing high-ranking cards.

Last to play: If you don't have a card of the suit in question, prioritize playing the Queen of Spades. If you do have a card of that suit, decide your strategy based on whether a high-ranking card (Queen of Hearts or a high-ranking red heart) has appeared.

The main problem of this algorithm is that it is very 'short sighted', we stick on not receiving points right now and use too many small cards, so in future, we will loss a lot.

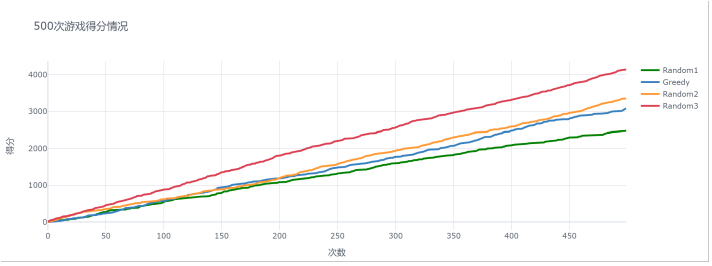

Figure 3: greedy vs random

2.2 MCTS

In our study of applying the Monte Carlo Tree Search (MCTS) method for optimizing card-playing strategies in Hearts, we began by identifying all legal card-playing options available at the current step. We then established a fixed time window to conduct multiple simulations of the game within this period, with the aim of identifying the playing strategy with the highest win rate. The win rate is calculated as the ratio of the number of wins to the total number of simulations conducted.

During the simulation process, we first clone the current game state to ensure that the simulation does not interfere with the actual game. Then, we randomly distribute the unknown cards in the game, transforming the problem with uncertainty into a deterministic one for subsequent simulation. After the preprocessing is completed, the simulation begins. Typically, MCTS consists of four phases: Selection, Expansion, Simulation, and Backpropagation. In games with large state spaces like Go, the Rollout phase usually starts from the root of the state tree with pure random simulation to address the issue of the intractable state space. However, in the application of Hearts, since our defined state space is "players + cards played," with a maximum fixed depth of 52, we can simplify the MCTS model by combining the Selection and Rollout phases.

During the Selection phase, before every possible next move has been explored at least once, we randomly select the next move. Once all possible moves have been explored, we choose the next step based on the UCB1 (Upper Confidence bounds for Trees) formula. The UCB1 formula is given by:

$$UCB1(i) = \bar{V}_i + C\sqrt{\frac{2\ln n}{n_i}} \tag{1}$$

where:

- $\bar{V}_i$ is the winning rate of the arm i.
- C is a constant that controls the degree of exploration. A higher value for C encourages more exploration of less tried arms. We set it as $\sqrt{2}$
- n is the total number of times all arms have been played.
- n_i is the number of times arm i has been played.

The simulation then continues until a win or loss is determined. During this process, if an unexplored node is encountered, it is expanded (Expansion) and added to the search tree. Lastly, for all nodes along the traversed path, their state information is updated: the number of visits is incremented by one, and if the simulation result is a win, the win count of that node is also incremented by one.(The process is as shown in the flowchart (figure4)

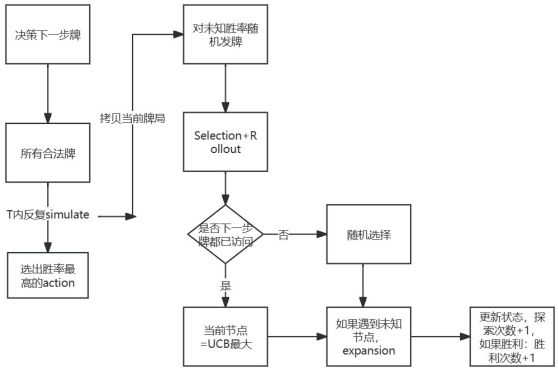

Figure 4: The flowchart of the MCTS

In the experimental phase of our study, we observed that although the MCTS algorithm significantly outperformed the greedy algorithm in efficacy, it required a longer running time. In the game environment of Hearts, the amount of points does not affect the final outcome of the game —winning by 100 points is considered the same as winning by 1 point. Based on this understanding, we introduced a hyperparameter threshold into the algorithm. When the MCTS algorithm has already secured a certain lead over other players in previous rounds, reaching this threshold, we appropriately reduce the time allocated for simulating games, thereby sacrificing some accuracy for increased speed.(figure5)

Furthermore, given that the model includes player states, all players in the simulation use the same MCTS strategy, which implicitly assumes that opponents also use the MCTS strategy for playing cards. However, this assumption is not always correct; opponents' play may be random or even naive. Therefore, by adjusting the hyperparameter threshold, we optimized the model so that when the MCTS algorithm has a significant lead, it is more inclined to treat opponents as

Figure 5: The left and right sides respectively show the performance and runtime of the original Monte Carlo algorithm and the accelerated Monte Carlo algorithm.

playing randomly or naively. This adjustment not only speeds up the algorithm but also has the potential to improve its performance further.(figure6)

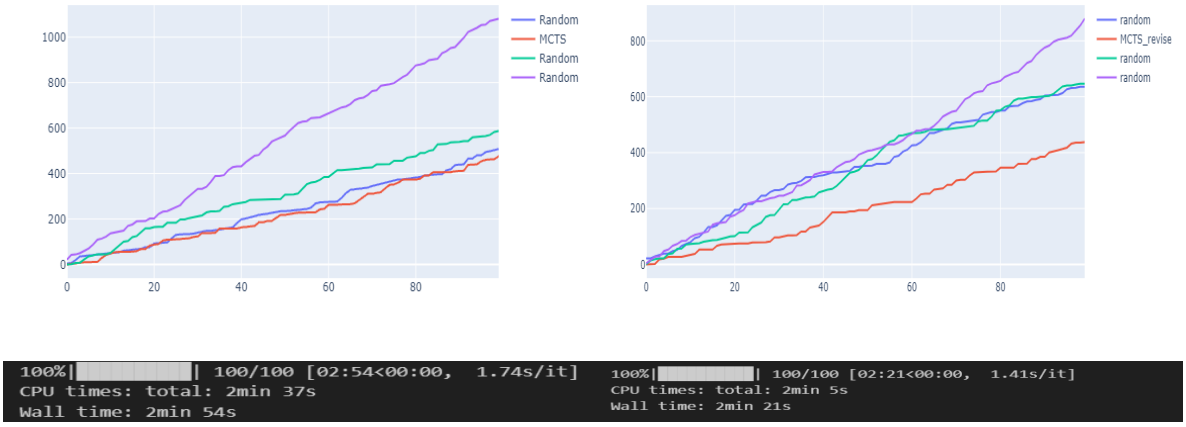

Figure 6: The left and right sides respectively show the performance and runtime of the accelerated original Monte Carlo algorithm and the modified Monte Carlo algorithm under the condition that the opponent is random.

Of course, when two players both using the MCTS algorithm meet, they are expected to be closely matched adversaries. Since the score difference between them will not be significant, the adjusted model does not trigger the aforementioned threshold adjustment mechanism when encountering equally skilled players using the MCTS strategy. Therefore, in such cases, the performance of both models is essentially the same.(figure7)

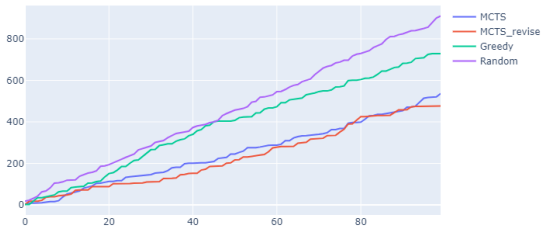

Figure 7: The result of two models playing as opponents in the same game.

2.3 Q-learning

The Q-learning algorithm stands out as an innovative contribution from our team. Traditional Q-learning faces challenges in card games due to the extensive number of states and actions involved. According to our investigation, there is no existing implementation of Q-learning for card games. Taking an unconventional approach, our team made substantial improvements to traditional Q-learning. Our Q-learning is composed of states and actions. We designed 12 distinct states based on players' varying situations:

· State 1: First player to play, no spades or max spade value greater than Q, and no one has played the spade Q.

· State 2: First player to play, small spades exist, and no one has played the spade Q.

· State 3: First player to play, spade Q has been played.

· State 4: Second or third player to play, no cards of the same suit played this round, and a spade Q is in hand.

· State 5: Second or third player to play, no cards of the same suit played this round, and no spade Q in hand.

· State 6: Second or third player to play, cards of the same suit played this round, and a smaller card than the current max is in hand.

· State 7: Second or third player to play, cards of the same suit played this round, and only cards larger than the current max are in hand.

· State 8: Last player to play, no cards of the same suit played this round, and a spade Q is in hand.

· State 9: Last player to play, no cards of the same suit played this round, and no spade Q in hand.

· State 10: Last player to play, cards of the same suit played this round, and no pig or hearts in the cards played this round.

· State 11: Last player to play, cards of the same suit played this round, and pig or hearts present, and small cards are in hand.

· State 12: Last player to play, cards of the same suit played this round, and pig or hearts present, and no small cards are in hand.

These 12 states represent classic situations players encounter during the game, forming a

comprehensive set. In different states, players often need to contemplate different strategies to achieve optimal results. Our actions are divided into two categories: good actions and bad actions, corresponding to a short-sighted greedy strategy focused on avoiding immediate point receiving and a long-term strategy of reserving small cards to prevent future points, respectively. In other words, our action design doesn't directly provide specific card plays but rather selects different playing styles, followed by invoking corresponding functions to determine the card plays. The following two examples contrast the different strategies of bad actions and good actions in some states.

2.3.1 Example Analysis

State 2: Greedy strategy involves playing the smallest spade to deplete opponents' spades, force the spade Q, and avoid playing high cards due to the fear of opponents playing hearts. For our "bad action," if it's an early round (<4), we play the smallest spade less than Q, aiming to deplete opponents' spades and not overly concerned about taking one or two points, with a focus on not having high cards when everyone plays hearts in the future.

State 3: Greedy strategy involves playing the smallest card in all suits to avoid point losses. Our "bad action" categorically considers: - If there are spades, play the largest spade. - If there are no spades, play a small heart (<=6). - If there are no small hearts and it's an early round, play high cards in all suits. The objective is to ensure not having high-point cards when everyone plays hearts in the future, not just worrying about taking one or two points at the moment.

2.3.2 Algorithm Explanation

We have designed a card-playing agent based on the Q-learning strategy, which evaluates and learns actions in specific states using the following formula (which is an improvement to the lecture PPT) :

$$sample = R(action) + R(points\,in\,last\,round) + \gamma \cdot \max_{a'} Q(s', a') \tag{2}$$

$$Q(s, a) \longleftarrow (1 - \alpha) \cdot Q(s, a) + \alpha \cdot (sample) \tag{3}$$

In this context, s and s' represent the states before and after taking an action, respectively, a and a' represent the actions taken in the corresponding states, R is the reward function, and Q represents the Q-value. α and γ are hyperparameters predetermined before program execution.

Regarding action a: To facilitate unified management, the action space corresponding to each state is abstracted into two categories: "good" actions, aligning with the greedy intuition of that state, and "bad" actions, considering long-term decisions. In other words, we consciously reserve certain small cards for the future, not overly concerned about avoiding points in the initial rounds. For different states, the corresponding "good" and "bad" actions have different implementation details. In general, in the early rounds, we prefer taking "bad" actions for strategic foresight, and in the later rounds, we use a greedy algorithm to minimize points. After deciding which approach (action) to use, we invoke the corresponding function to determine the specific card play.

Regarding reward function, we divide it into two parts: R(action) and R(points in the last round):

R(action) represents the scores for different actions. R("good") and R("bad") are initially set to 20 and 25, respectively (as we want the agent to be forward-thinking initially and greedy later). Subsequently, each round, $R(\text{"good"}) = R(\text{"good"}) + \text{round}/2$; $R(\text{"bad"}) = R(\text{"bad"}) - \text{round}/2$.

R(points in the last round) represents the points obtained by the algorithm's card play in the previous round, denoted as x. The updating formula is as follows:

$$R(x) = -\theta x$$

where θ is a customizable parameter. The significance of this approach is that when the algorithm learns that it lost points in the previous round, it will penalize that decision, reducing trust in it for subsequent rounds. The number of algorithm choosing good and bad action is as follow, as shown in picture, we will choose less bad action in later rounds:

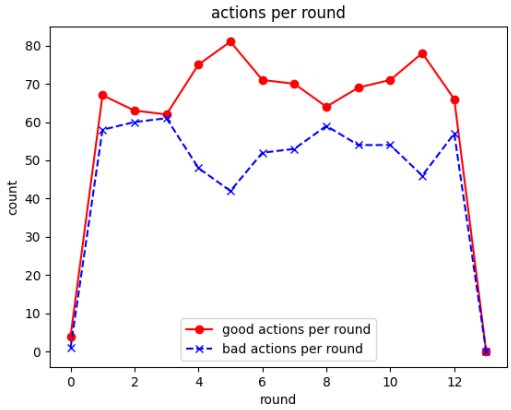

Figure 8: number of choosing which action in several rounds

Regarding Q-values: We initialize all Q-values to 0, continuously updating them during Q-learning. Since it is not possible to predict the next round's state directly after each card play, i.e., given s and a, predicting s' is not possible. To address this, when in state s_t, $s_t = s'_{t-1}$, allowing us to update $Q(s_{t-1}, a_{t-1})$. In other words, we keep a record of each decision's state and action. When the agent is in a new state, we update the Q-value of the previous state. Considering the large state space and the fact that each game has only 13 rounds, i.e., 13 chances to update Q-values per game, we set Q-values as learnable parameters, saved in a .json file for pre-training. In the function $Game.AutoPlay()$, setting the variable $Train$ = True initiates training mode. Each Q-learning agent updates its Q-values as the game progresses, saving them in $Q\_values\_trained\_i.json$ after the game ends. Setting the variable $load\_params$ = True initializes Q-values using the pre-saved values; otherwise, the default is 0.

As shown below, our innovative Q-learning algorithm is worse than MCTS but better than greedy and random.

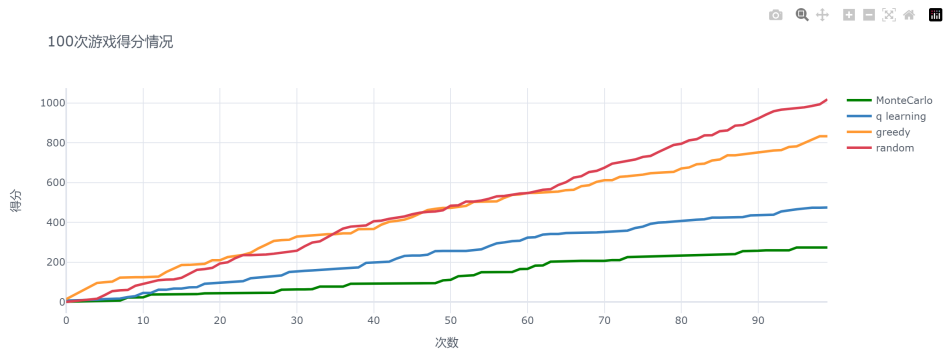

Figure 9: our innovative Q-learning algorithm is not bad

3 Experiment and result

experiments ploted: 200 times

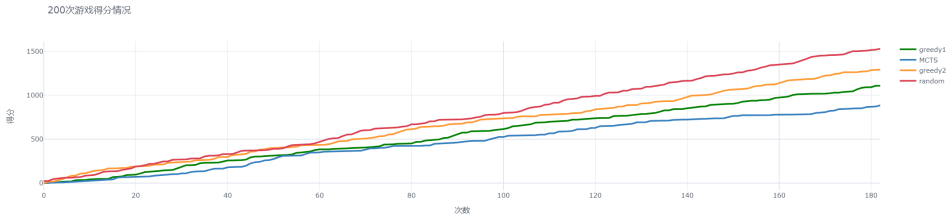

Figure 10: MCTS+Random+Greedy*2

```
CPU times: total: 35.5 s
Wall time: 4min 53s
```

Figure 11: time for MCTS

```
CPU times: total: 40.2 s
Wall time: 4min 17s
```

Figure 12: time for revised MCTS

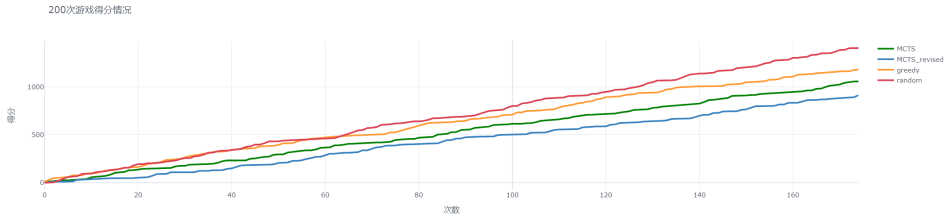

Figure 13: mcts1+mcts2+Random+greedy(compared on score)

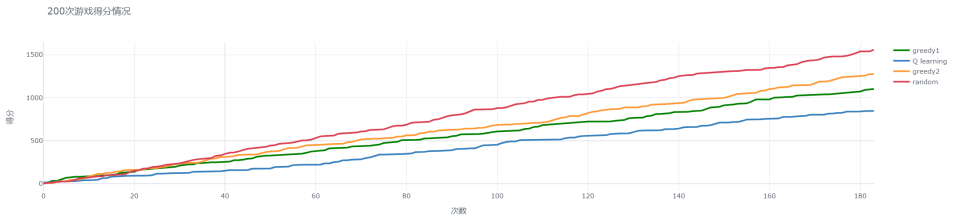

Figure 14: Qlearning+Random+greedy*2

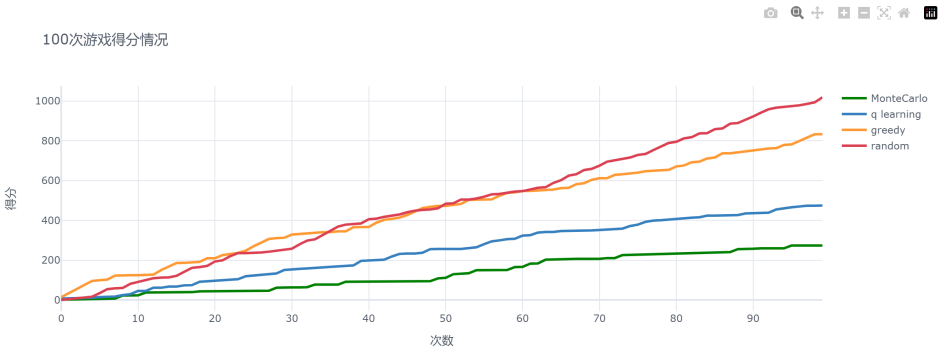

Figure 15: mcts+qlearning+random+greedy

4 Future Work

In MCTS, different branches of the search tree can be explored independently, allowing for efficiency improvements through parallel computation. We think parallelization in MCTS can occur at two levels:

Parallelization of Simulations: During the simulation phase, different simulations can run in parallel on separate processing units. Each processing unit independently executes simulations, and the results are then communicated to the main thread for integration.

Parallelization of Tree Expansion: The tree structure in MCTS is dynamically expanded, and different subtrees can be expanded in parallel on distinct processing units. This allows different parts of the search tree to be explored independently.

For Q-learning: We can further refine the state space by considering additional factors, such as incorporating opponents' card play preferences and utilizing the current score situation as part of the reward assignment criteria. For instance, when trailing in score, we can assign a higher reward to the Q-values resulting from "bad actions."

For Q-learning: we can expand the range of possible actions beyond the simplistic "good action" and "bad action" categorization. This may involve introducing more nuanced playing strategies, such as: Cautious Play: Taking a pessimistic view that opponents may collaborate against the player. Bold Play: Making daring moves, such as assuming opponents won't play certain high-value cards early. Strategic Play for the Long Term: Similar to existing "bad actions," emphasizing reserving small cards for future rounds. Immediate Focus Play: Similar to a greedy approach, concentrating on avoiding points in the present.

References

For Hearts rules : https://github.com/pjgao/MonteCarloHeartsAI, 红心大战规则框架(注意原文件运行结果错误)

For how to win the game : https://baike.baidu.com/item/%E7%BA%A2%E5%BF%83%E5%A4%A7%E6%88%98/875494?fr=ge_ala, 百度百科

For idea of Q-learning : lecture PPT

For greedy algorithm : https://www.huotun.com/game/395890.html, 红心大战游戏, 想赢就要知道这些技巧 (从菜鸟到高手)

For MCTS: lecture PPT