DanZero: Mastering GuanDan Game with Reinforcement Learning

1st Yudong Lu
Institute of Artificial Intelligence
Hefei Comprehensive Nation Science Center;
University of Science and Technology of China
Hefei, China
luyudong@mail.ustc.edu.cn

2nd Jian zhao
University of Science and Technology of China
Hefei, China
zj140@mail.ustc.edu.cn

3rd Youpeng Zhao
University of Science and Technology of China
Hefei, China
zyp123@mail.ustc.edu.cn

4th Wengang Zhou
Institute of Artificial Intelligence
Hefei Comprehensive Nation Science Center;
University of Science and Technology of China
Hefei, China
zhwg@ustc.edu.cn

5th Houqiang Li
Institute of Artificial Intelligence
Hefei Comprehensive Nation Science Center;
University of Science and Technology of China
Hefei, China
lihq@ustc.edu.cn

Abstract—The use of artificial intelligence (AI) in card games has been a widely researched topic in the field of AI for an extended period. Recent advancements have led to AI programs exhibiting expert-level gameplay in complex card games such as Mahjong, DouDizhu, and Texas Hold'em. This paper aims to develop an AI program, named DanZero, for GuanDan, an exceptionally complex card game that involves four players competing and cooperating in a long process to upgrade their level quickly. Developing AI for GuanDan is challenging due to its large state and action space, long episode length, and uncertainty in the number of players. To address these challenges, we propose DanZero, the first AI program for GuanDan, that employs reinforcement learning using a distributed framework for training. Our framework consists of two processes: the Actor Process and the Learner Process. In the Actor Process, we design state features and generate samples through agents’ self-play. In the Learner Process, we update the model using the Deep Monte-Carlo Method. We trained DanZero for 30 days, utilizing 160 CPUs and 1 GPU to develop the program successfully. We compared DanZero’s performance with eight baseline AI programs based on heuristic rules, and our results indicate DanZero’s exceptional performance. We further tested DanZero with human players and demonstrated its ability to perform at a human level. The code for DanZero can be found in the supplementary material.

Index Terms—GuanDan, Reinforcement Learning, Deep Q Learning, Monte Carlo Simulation

I. INTRODUCTION

The use of games as a benchmark for assessing the strength of artificial intelligence (AI) has generated considerable interest in the field of machine learning, particularly in reinforcement learning [1]–[3]. Recent advances in reinforcement learning have led to significant progress in various games, including board games like Go [4], [5] and chess [6], card games such as Texas Hold’em [7]–[9] and Mahjong [10] and video games such as StarCraft [11] and DOTA [12]. However, games with imperfect information and a large state and action space still pose a challenging problem for reinforcement learning.

This study aims to develop an AI program for GuanDan, a card game popular in China with over 20 million active players that has received limited attention in the literature. The game involves cooperation and competition under a partially observable environment, with four players split into two groups playing against each other using two decks of cards. The game’s complexity is due to its larger state and action space, as demonstrated by the significantly higher information set size and count compared to other games, including 4-player Mahjong, and a much larger action and legal action space, as shown in Figure 1. Although recent progress has been made in AI for chess and card games [10], [13]–[16], developing an AI program for GuanDan remains a challenging and unresolved issue.

In general, GuanDan is challenging due to the following issues:

• **The state space and action space of GuanDan is large:** In a GuanDan game, there are two decks of pokers used while many existing games that have been studied only use one deck of cards. What’s more, the suits of cards are important in GuanDan as the combination of cards is affected by this factor while suits of cards are usually ignored when dealing with some card games such as DouDizhu. In addition, there exist “wild card” and “level card” in GuanDan, making it much more difficult to play...
compared with other card games.

- **The length of one episode of GuanDan game is long:** GuanDan involves a concept of “leveling up”, and the game ends only when a team of players has upgraded over level A so that one episode of the game contains several rounds. In each episode, each agent in GuanDan has to make over 100 decisions while the number of decisions that agent in other card games needs to make in one episode is generally not higher than 20.

- **The number of players in the GuanDan game may change as the game progresses:** At the beginning, there are four players in GuanDan, two of whom make up a team against the team of the remaining two players. One round of the GuanDan game terminates only when both players of any team have emptied their hand cards, making this game very complex. For example, if one player has emptied his hand cards, the following game will be turned into an unequal situation, i.e., two players cooperate against the rest single player.

- **The number of legal actions under a state is uncertain:** In the game of Guandan, due to the existence of special card combinations and “wild card”, there can be more than 5000 legal actions at the beginning state of a game, but this number rapidly decreases to fewer than 50 as cards are played from the hand. This feature makes action modeling very challenging.

Most existing AI programs for GuanDan are based on heuristic rules and many carefully designed techniques are applied to deal with different situations. For example, a player may give priority to cards that are small or can not form special card types when leading the first trick. When a player plays cards passively to cover cards of other players, using cards of the same card type is prior to playing bombs. There’s one work [17] that tried to develop an AI program for GuanDan which adopts Upper Confidence Bound Apply to Tree (UCT) algorithm but it performs only slightly better than random agents, i.e., agents choose actions randomly from legal action set.

In this work, we propose an AI system for GuanDan called DanZero with reinforcement learning. Considering the large state and action spaces, classical value-based reinforcement learning algorithms such as Deep Q-Learning (DQN) [18] will probably suffer from overestimating issue [19]. Similarly, policy gradient methods such as A3C [20] also perform unsatisfactorily in issues with large action space as they fail to leverage action features. To this end, we propose to adopt the Monte Carlo method enhanced by neural networks, which can utilize the action features and approximate true values without bias [21]. What’s more, we adopt feature encoding techniques to process the state and action features and implement a distributed self-play reinforcement learning framework to facilitate training. Integrating these techniques, our AI system for GuanDan is able to beat all other existing algorithms, proving the effectiveness of our methods.

**II. RELATED WORK**

This section provides a background on the concept of imperfect-information games and highlights the use of reinforcement learning to develop game AI in such contexts.

Imperfect-information games are characterized by hidden information and randomness, making them more representative of real-world scenarios that involve stochasticity. Consequently, handling imperfect-information games poses more challenging and significant research questions compared to perfect-information games. In the classical imperfect-information games, poker games, Counterfactual Regret Minimization (CFR) [22] and its variants, where a model of game is required to traverse the game tree during computation, are often adopted. To handle a large-scale imperfect-information game, learning an abstraction of state or action space to reduce the game to a manageable size is often applied [9], [23], [24]. However, as GuanDan involves cooperation and competition at the same time and the number of players may change as the game goes on, such a complex setting poses great challenges to these classical algorithms in poker games. Although utilizing deep neural networks to generalize across states helps CFR methods obviate the need for abstractions [25], [26], this family of algorithms still have difficulty dealing with games with large state and action space.

Monte Carlo simulation relies on repeated sampling and statistical analysis to simulate real physical scenes and get approximate solutions to problems [27]. With the help of a large amount of data, the deviation of the simulated values will be very small. Therefore, this approach performs well in environments that contain hidden information or randomness [28], [29]. In this work, we use the Monte Carlo method to estimate the value of legal actions in each state. At the same time, different rewards are designed for different stages of the game according to the rules, accelerating the convergence of Monte Carlo’s estimation method.

At the moment, other methods are still using heuristic rules or CFR methods to build the Guandan AI. These schemes mainly use human priority knowledge, stipulating that intelligent bodies make reasonable decisions in various cases. At present, the most capable Guandan AI will classify the hand card, artificially sets the value setting of some legal actions in different circumstances, and select action through value comparison. Most of the current solutions do not consider using reinforcement learning algorithms to improve Guandan AI skills.

Different from CFR methods which rely on game-tree traversals, reinforcement learning can help models learn skills through interactions with the environment so this technique is very suitable for large-scale games. Thanks to the development of reinforcement learning, there is recently a growing trend in utilizing this technique to solve imperfect-information games. For instance, AI programs for famous large-scale games such as DOTA [12], StarCraft [11] and Honor of King have been developed and achieved amazing performance [30]. As for card games, reinforcement learning has also been successfully
adopted in Mahjong, Texas Hold’em, DouDizhu, and so on [7], [8], [10], [13], [15]. What’s more, reinforcement learning can be combined with many other techniques such as search [31] and rule-enhance [32] and shows satisfactory performance. Considering the advantages of this powerful technique, we choose reinforcement learning to develop an AI program for the unsolved GuanDan game.

III. PRELIMINARY

In this section, we give a brief introduction to the basic rules of GuanDan, including the card-playing phase and tribute phase. More detailed rules can be found on Wiki 1.

A. Basic Card-play Knowledge

The card types of this game are listed in Figure 2. There are four suits in the cards used in the GuanDan game, including Hearts (H), Spades (S), Diamonds (D), and Clubs (C). The basic rank of single cards is, from high to low, Red Joker (RJ), Black Joker (BJ), Level cards, A, K, Q, J, 10, 9, 8, 7, 6, 5, 4, 3, 2. When forming Tubes, Plates, Straights, or Flushes, Aces can be seen as 1, just below 2. For Solos, Pairs, Triples, Tubes, Plates, Full Houses, and Straights, cards can only cover the same type. Full Houses are ranked by the points of the Triple part. Bombs can cover these mentioned combinations and Bombs with more cards can cover Bombs with fewer cards. If the numbers of cards are the same between two Bombs, they are also ranked by their points. Flush Straights can cover Bombs with less than six cards; the relationship between this kind of card type is determined by the points. Finally, Joker Bombs can beat any card type.

B. Rules in Guandan

In a GuanDan game, there are two decks of standard pokers used, including Jokers, and four players sitting around a square table, each of whom has 27 cards in hand. The players sitting opposite each other belong to the same camp. What’s more, there exists the concept of “leveling up”, “level cards” and “wild cards”. To be specific, both camps of this game are on their own level which starts from 2 to A. The first round of a GuanDan game is always played at Level 2 and the levels of the subsequent rounds are determined by the level of the camp who have won in the previous round. Cards of the same rank as the level of the current round are called “level cards” and they rank just below Jokers when being played singly. These kinds of cards can also be used at their ranks in natural order when making up other combinations. In addition, level

---

1https://en.wikipedia.org/wiki/Guandan#Playing
cards in Heart are “wild cards” and they can be utilized in place of any cards needed to make up a combination except for Jokers. The camp that first levels up over level ‘A’ will win the game. To this end, one GuanDan game usually contains several rounds.

Players play cards in counterclockwise order and the player leading the first trick can play any type of card from his hand. The other players can play cards of the same type or bombs, which are larger than cards played by the previous player, or they can choose to pass. A trick continues until three players pass in succession or the player who played the last cards leads in the next trick. Such procedures repeat until three players have no cards left or players of the same camp have emptied their cards and this round ends. The first one emptying the cards is called the Banker, and other players are called the Follower, the Third, and the Dweller according to the order of their emptying the cards, respectively. Only the team of the Banker can promote the level and the promoted number can be 3 or 2 levels, the level of A can not be skipped. At the round with level A, a camp can only win when the Banker’s partner is the Follower or the Third.

In this section, our DanZero will be described in three aspects: the heuristic rules, the model architecture design, and the training algorithm.

A. Heuristic Rules

When people or AI play Guandan to the second and subsequent round of one episode, there exists a Tribute Phase. In this phase, the Dweller of the previous round has to pay a Tribute to the Banker by giving his biggest single card other than the wild card. After that, the Banker has to give a single card back with a point not higher than 10 to ensure that each player has the same number of cards. Because the logic of this stage is quite different from playing cards, we utilize heuristic rules to make decisions in the Tribute phase. The following describes the heuristic rules we use during the Tribute phase.

As the Tribute that the Dweller has to pay is decided by the rule of the game, we just have to figure out how we choose which card to return tribute. In this process, we have two main aspects to consider, one is to try to ensure the strength of their own hand and the integrity of some special card types, and the other is to make the return of the card as far as possible not to enhance the strength of the opponent’s hand or improve the integrity of their hand. Based on these considerations, the player should first combine his cards and give priority to the combinations of Bombs and Straight Flush. After the combination, if there exists a card, whose point is lower than 10, that can only make up a Single card type, just return one card back with a point not higher than 10 to ensure that each player has the same number of cards. Because the logic of this stage is quite different from playing cards, we utilize heuristic rules to make decisions in the Tribute phase. The following describes the heuristic rules we use during the Tribute phase.

B. Model Architecture Design

The key in our model architecture design is taking all relevant information and candidate action as input and outputting the state-action value. We encode each card combination with a 54-dimensional vector, corresponding to the 54 cards in poker. There are 3 possible values for each element in the vector, i.e., \{0, 1, 2\}, indicating the number of cards of the corresponding suits and points. An example is shown in Figure 3. In this way, the feature encodes the suit information, and the dimension size is acceptable.

The feature of the state is composed of a vector with 513 dimensions, and their physical meanings are listed as follows (from the view of one player):

- \(0 − 53\): our current hand.
- \(54 − 107\): the remaining cards, i.e., all the cards except our current hand and all played cards.
- \(108 − 161\): the last move and the cards that we are going to play must be able to cover this combination of cards. If we have to lead the trick, these dimensions are set to zero.
• [162 – 215]: the last move of partner. If the last move of a teammate is “pass”, this vector is set to zero. If the partner has finished his hand cards, these dimensions are set to be -1.
• [216 – 299]: the number of remaining cards of the other three players which are recorded in the order of playing cards.
• [300 – 461]: the played cards of the other three players which are recorded in the order of playing cards.
• [462 – 501]: the level of our team and opponent team.
• [501 – 513]: flag for wild cards, namely, whether we have wild cards in hand and whether these cards can make up Bombs, Straight Flushes, Straights, or other card types except Single and Joker Bomb.

As for the action features, they are also represented with a 54-dimensional vector. The network that we adopt consists of several layers of Multi-Layer Perception (MLP) and the input is the concatenation of the state and action features, which is a 567-dimensional vector. The output of the network is the Q-value for one state-action pair. Figure 4 demonstrates how we divide different regions to form the state vector and the architectures of the network that we adopt.

**Algorithm 1: Process of Actor**

1: Initialize environment ENV;
2: Initialize model $M$ with random parameters;
3: for Episodes=1,2,3,... do
4: Initial state $s_0 = \text{ENV.reset}()$;
5: Set $t = 0$;
6: while not done do
7: for Agent=1,2,3,4 do
8: Calculate the legal actions set $A$;
9: Choose action $a^i$ by $\epsilon$-greedy;
10: Collect set of trajectories by interacting with the environment:
11: $\tau^i_t = f(\tau^i_{t-1}, a^{-i})$;
12: end for
13: $t = t + 1$;
14: end while
15: Assign a value $r$ for every sample;
16: For each trajectory $(\tau_t, a_t, Q(\tau_t, a_t), r_t)$, save it to replay buffer $B$;
17: Update model $M$ with period $I$;
18: end for
value for every sample according to the result of this round. To be specific, for the winning team, samples of their trajectory will be assigned +3, +2 and +1 when the partner of the Banker is the Follower, the Third and the Dweller, respectively. And the samples of losing camp will be assigned corresponding negative value. When the level of the last round of one game is A, the reward changes to 0 when the Banker’s partner is not the Dweller as they can not win the game in this case. The agent trajectory data tuple \((\tau,a,Q(\tau,a),r)\) is sent to the learner to train the model after one episode terminates. Considering the self-play procedure, each episode will produce 4 trajectories.

Need to add that, in the second and subsequent round of one episode, there exists a Tribute phase. Because the logic of Tribute is quite different from playing cards, we utilize heuristic rules to make decisions in this phase. What’s more, in this phase data will not be saved. Detailed rules are available in supplement materials.

2) Learner: The learner is responsible for network update. In each iteration, the learner receives the collected episode data from the actors and the data is stored in a buffer. The learner samples a batch of data from the buffer to update the network with Deep Monte Carlo. Deep Monte Carlo is an effective value-based algorithm especially in such episodic and reward-sparse tasks which has a very large state and action space. However, considering the transportation delay in distributed reinforcement learning, it is necessary to preprocess the state-action value sent by the actor as the following equation shows:

\[
Q_p(\tau,a) = \text{clip}(\frac{Q(\tau,a;\theta_1)}{Q(\tau,a;\theta_a)} + 1 - \lambda, 1 + \lambda) * Q(\tau,a;\theta_a),
\]

where \(Q(\tau,a;\theta_a)\) refers to the state-action value predicted by the \(\theta_a\) parameter on the actor side, and \(Q(\tau,a;\theta_1)\) refers to the state-action value predicted on the learner side. \(\lambda\) is a hyperparameter and the definition of clip function is shown as follows:

\[
\text{clip}(x, x_{\text{min}}, x_{\text{max}}) = \begin{cases} 
    x_{\text{max}}, & x > x_{\text{max}} \\
    x_{\text{min}}, & x_{\text{min}} \leq x \leq x_{\text{max}} \\
    x, & x < x_{\text{min}}
\end{cases}
\]

Using the above preprocessing formula can remove the samples with a large distribution difference against the model on the learner side, so that the given target value is time-sensitive. Then we construct an optimization function as follows:

\[
\text{Loss} = \frac{1}{N} \sum_{i=1}^{N} [Q_p(\tau,a) - r]^2,
\]

where \(N\) represents the batch size.

By adopting the Distributed Q-Learning, we can parallelize multiple actor processes so that the training process of our AI system is efficient. The overall algorithm framework is summarized in Algorithm 1 and 2.

V. EXPERIMENTS

In this section, we compare the performance of DanZero with state-of-the-art rule-based methods on the GuanDan benchmark. Also, in order to more intuitively demonstrate the strength of our AI, we test DanZero against human players. Our AI system is trained on a server with 4 Intel(R) Xeon(R) Gold 6252 CPU @ 2.10GHz and GeForce RTX 3070 GPU in Ubuntu 16.04 operating system.

A. Experimental Setup

In order to evaluate the performance of our AI program, we launch tournaments between our model and different baseline algorithms. To be specific, two agents of one team in GuanDan game utilize our model and another team adopts baseline algorithm. By initializing different games and executing the test for many times, we can achieve objective evaluation results.

We perform a mild hyper-parameter search on Q-learning and use the best setting for the shared hyper-parameters for all methods. An overview of hyper-parameters for each method is listed in Table I. What’s more, we adopt 80 agents for training.

| Hyper parameters (%) | Meaning                                      | Value |
|----------------------|----------------------------------------------|-------|
| Epsilon              | Probability of random exploration in actor   | 0.01  |
| Mempool size         | Memory pool size in learner                  | 65536 |
| Batch size           | Batch size per training                      | 32768 |
| Training freq        | Number of receptions between each training   | 250   |
| Lamda                | Range of clip Q                             | 0.65  |
| lr                   | Learning rate                               | 0.001 |
| Optimizer            | Optimizer in training                       | RMS   |
| Actor num            | Number of actor                             | 80    |
| Actor core           | CPU core per actor                          | 2     |
| MLP layer            | Number of MLP layers                        | 4     |
| MLP node             | Number of MLP nodes per layer               | 512   |
| Activation functions | Activation functions per layer              | tanh  |

TABLE I: The hyperparameters used in our AI system.

B. Performance against 8 Rule-based Bots and Each Other

To show how our model performs in the training process, we save a checkpoint every 24 hours. We evaluate these checkpoints by playing 1000 games with 8 rule-based bots. Considering that one GuanDan game contains several rounds, playing 1000 games is enough to reveal the performance of an AI program objectively. To be mentioned, the baseline rule-based bots that we compare are the top 8 agents in the first
Chinese Artificial Intelligence for GuanDan Competition so that their strengths can be guaranteed. Their implementations are available at this website \(^2\).

The average win rate of our model against different baselines is shown in Figure 5 and the indexes of baselines represent their ranks in the competition. Due to the long process of GuanDan games, 1000 matches are enough to reflect the strength of the model. It can be observed that our AI system DanZero is able to achieve significantly better performance than those rule-based agents. Specifically, DanZero has absolute superiority over the rule-based bots except for baseline 1, baseline 2 and baseline 4 after enough training. What’s more, considering that the nature of high variance of this game, achieving the win rate of 80% is an obvious preponderance so that our DanZero also performs much better than the other three baseline algorithms.

It’s interesting that baseline 2 achieves the best result against our model instead of baseline 1 and baseline 3 seems to perform worse than baseline 4 and baseline 6. To figure out this phenomenon, we also conduct evaluations between all the AI programs and the results are shown in Table II. It can be observed that the performance of different baselines does not quite correspond to their ranks after enough evaluation, which can account for the above phenomenon partly. However, the overall performance of baseline 1, baseline 2 and baseline 4 is equivalent but baseline 2 obviously performs better than the other two against DanZero. We assume that this is because there exists a restraint relationship between different policies. In other words, a weak bot is also possible to beat a strong bot if its policy is just suitable to target the opponent’s weakness. This phenomenon is very common in AI programs using heuristic rules in fact. However, our DanZero can still deal with these agents which have different styles and achieve obvious superiority, proving the effectiveness of our methods.

In addition, we also test the effectiveness of flag for “wild cards” in the state feature. We conduct abrasive experiment that removes these dimensions in state features and Table II reports the results, which is represented by “Our-”. It can be observed that removing these information degrades the performance of our model, proving that adding this feature helps the model better grasp the use of “wild cards”.

C. Human Evaluation

Apart from comparisons with strong rule-based bots, we also evaluate the real performance of DanZero against human players.

Ten human players are all excellent graduate students who are very intelligent and flexible in thinking. Although they are not professional Guandan players, they have a certain level of gameplay and a good understanding of the rules. In the competition, we let two human players as teammates to play against two AIs, and each pair of human players plays twenty rounds against the AIs. In the test, two students were new to Guandan. They lost 17 out of 20 rounds against the AIs. Four students had been playing for one to two years. They lost 30 out of 40 rounds against the AIs. Four students had been playing for three years or more. They lost 24 out of 40 rounds against the AIs. In total, the AIs won 71 rounds out of 100 games. When we invited the highest-level players we could find to compete, the AIs won 24 out of 40 rounds, achieving a 60% win rate in both cases.

VI. Conclusion

This paper presents an AI system for GuanDan, an imperfect-information game that poses significant challenges, such as a large state and action space and an unknown number of players. To address these challenges, we employ Deep Monte-Carlo Methods as the primary algorithm, characterizing the state information and utilizing a distributed self-play paradigm, resulting in a strong RL bot called DanZero. To demonstrate the effectiveness of our method, we compare DanZero with state-of-the-art rule-based baselines, and our AI program’s outstanding performance highlights its efficacy. Additionally, we perform human evaluation, showing that DanZero reaches human-level performance. We aim to establish our work as a benchmark for further research on the GuanDan game.

For future work, we will explore avenues to enhance our AI system. Specifically, our current model requires 30 days of training, and we will investigate methods to accelerate the training process.

REFERENCES

[1] I. P. Pinto and L. R. Coutinho, “Hierarchical reinforcement learning with monte carlo tree search in computer fighting game,” IEEE Transactions on Games (TOG), vol. 11, no. 3, pp. 290–295, 2018.

[2] D. Perez-Liebana, J. Liu, A. Khalifa, R. D. Gaina, J. Togelius, and S. M. Lucas, “General video game ai: A multitrack framework for evaluating agents, games, and content generation algorithms,” IEEE Transactions on Games (TOG), vol. 11, no. 3, pp. 195–214, 2019.

---

\(^2\)http://gameai.njupt.edu.cn/gameaicompetition/guandan_machine_code/index.html
| Win Rate (%) | baseline8 | baseline7 | baseline6 | baseline5 | baseline4 | baseline3 | baseline2 | baseline1 | Ours- | Ours |
|--------------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-------|-------|
| baseline8    | 26.72     | 1.14      | 100.00    | 0.00      | 13.10     | 0.00      | 0.00      | 0.00      | 0.00  | 0.00  |
| baseline7    | 73.28     | 42.52     | 100.00    | 6.93      | 15.31     | 0.00      | 0.00      | 0.00      | 0.00  | 0.00  |
| baseline6    | 98.86     | 57.48     | 97.67     | 18.40     | 47.96     | 28.19     | 15.64     | 0.00      | 0.00  | 0.00  |
| baseline5    | 0.00      | 0.00      | 2.33      | -         | 0.00      | 0.00      | 0.00      | 0.00      | 0.00  | 0.00  |
| baseline4    | 100.00    | 93.07     | 81.60     | 100.00    | 83.15     | 36.42     | 55.71     | 17.32     | 12.55 | -     |
| baseline3    | 86.90     | 84.69     | 52.04     | 100.00    | 16.85     | -         | 12.89     | 14.28     | 0.00  | 0.00  |
| baseline2    | 100.00    | 100.00    | 71.81     | 100.00    | 63.58     | 88.19     | -         | 54.44     | 25.33 | 17.39 |
| baseline1    | 100.00    | 100.00    | 86.36     | 100.00    | 54.29     | 86.78     | 45.56     | -         | 12.77 | 9.82  |
| Ours-        | 100.00    | 100.00    | 100.00    | 100.00    | 82.68     | 100.00    | 74.67     | 87.23     | -     | 46.55 |
| Ours         | 100.00    | 100.00    | 100.00    | 100.00    | 87.45     | 100.00    | 82.61     | 90.12     | 53.45 | -     |

TABLE II: The average performance of the compared algorithms by playing 1000 episodes of GuanDan. The win rate of each row is achieved by test between the bot in the first column against other algorithms. “Ours-” represents the abrasive model that removes the flag for wild cards. The results of “Our” and “Ours-” are achieved after training for 30 days.