Gongzhu AI based on Reinforcement Learning

Yue Zhang
Student ID: 300223245
Carleton University & University of Ottawa
yuezhang17@cmail.carleton.ca / yzha1030@uottawa.ca

Abstract

Gongzhu is an intriguing imperfect-information card game, characterized by its complex dynamics of collaboration and confrontation among four players. Despite its rich strategic depth, Gongzhu has been largely overlooked in the reinforcement learning (RL) community, and there are no existing benchmark environments, algorithms, or evaluation metrics specifically designed for the game. To address this gap, this project develops a Gongzhu environment with a user-friendly graphical interface (GUI). Additionally, I introduce a manual feature extraction (MFE) scheme that aids in the training of deep neural networks, alongside the benchmark deep Monte-Carlo (DMC) algorithm. I also propose a novel Elo-based evaluation metric that effectively captures the collaborative dynamics of the game, with a particular focus on the margin of victory (MOV). Empirical results demonstrate the advantages of MFE-trained models over hard-coded random and greedy models, highlighting the potential of this approach.

1 Introduction

Reinforcement learning (RL) algorithms have long been used to develop artificial intelligence (AI) for multiplayer games. Extensive research has been conducted on two-player, perfect-information games, leading to significant advancements. From search-based models like Deep Blue[10] and GPS Shogi[21] to RL-based, self-learning approaches like AlphaGo[22] and AlphaZero[22], and more advanced model-free methods like MuZero[18], AI has achieved superhuman performance, often surpassing top human players within a relatively short training period.

In contrast, the study of **imperfect-information** games, where players must make decisions with hidden or uncertain information, remains a significant challenge. Unlike perfect-information games, these environments require agents to reason under uncertainty, model opponents' strategies, and adapt dynamically to changing conditions. Moreover, many imperfect-information games involve more than two players, which further increases the complexity of the problems. Many open questions remain, making imperfect-information games a fertile ground for future research in AI and reinforcement learning.

This project explores the application of RL algorithms to **Gongzhu**, a traditional Chinese card game that involves four players and a standard 52-card deck. As an imperfect-information game, Gongzhu presents unique challenges due to its complex scoring system and the strategic collaboration required among players. These features make Gongzhu a particularly difficult game to solve, highlighting the significance of this project in advancing RL techniques for handling games with incomplete information and intricate dynamics.

The contributions of this project are summarized as follows:

Project source code is available on `https://github.com/sdawzy/gongzhu`

1. Developed **a novel Gongzhu environment** based on OpenAI Gymnasium, accompanied by **a graphical user interface (GUI)** named GongzhuGUI using React Native[1]. This environment accurately simulates Gongzhu with customizable options, such as opponent strength and declaration phrases. Additionally, the GUI provides a visual representation of the game dynamics, enabling the general public to enjoy Gongzhu while also facilitating the evaluation of different Gongzhu AIs.
2. Designed **a manual feature extraction (MFE) scheme** for Gongzhu that accelerates training with Deep Monte Carlo (DMC) while achieving performance comparable to fully automated feature learning.
3. Developed **an Elo-based evaluation system with Marginal of Victory (MOV)** for Gongzhu AIs, effectively capturing the game's complex imperfect and incomplete information dynamics. This system assesses AI performance in the presence of teammates and opponents with unknown strategic rationales, providing a robust measure of AI effectiveness.

1.1 Background: Rules of Gongzhu

Gameplay

Gongzhu is played with 4 players and a standard deck of 52 cards. In the beginning, cards are dealt equally to all players, with each player receiving 13 cards. Players take turns playing one card each. The first card determines the suit for that round. **Players must follow suit if possible**. If a player cannot follow suit, they may play any card. The highest card of the lead suit wins the round, and the winner **collects all cards played**. Note that Ace is the largest and 2 is the smallest in any suit.

Scoring Mechanism

Some cards are scored, as shown in Table 1. For convenience, usually Queen of Spades is called the Pig, Jack of Diamonds is called the Sheep, Ace of Hearts is called the Big Blood, and 10 of Club is called the Doubler. The score of a player is the **sum of all collected scored cards**. Moreover, if one collects the Doubler (10 of Clubs), their score is doubled. In addition, **if one collects all Hearts, the scores of Hearts will become positive**; if one does not have any scored cards except 10 of Clubs, their score is 50.

Cards	Hearts A	Hearts K	Hearts Q	Hearts J	Hearts 10–5	Hearts 4–2	Spades Q	Diamonds J
Score	-50	-40	-30	-20	-10	0	-100	$+100$

Table 1: Card Values and Scores

Game Objective

A player and the player directly opposite you are teammates. **One team's score is the combined total of both players' individual scores**. The objective of Gongzhu is to **maximize the score difference between one team and the other team**.

Declaration Mechanism

Optionally, players may choose to declare certain special cards before the first round begins. During the declaration phase, players take turns deciding **whether to declare any special cards from their hand**. The four special cards are the Pig, the Sheep, the Big Blood, and the Doubler. A special card can be declared in two ways: either secretly (closely) or openly. A secretly declared card has double the effect (×2), while an openly declared card has quadruple the effect (×4).

To maintain game balance and prevent openly declared cards from being too powerful, openly declared cards cannot be played in the first round of their suit—unless the player has no other card in that suit. For example, a player cannot play an openly declared Pig in the first round of spades, or an openly declared Doubler in the first round of hearts, unless it is their only card of that suit.

[1]Available in `https://sdawzy.github.io/gongzhu/`

(For the scope of this project, I will not include this declaration mechanism)

1.2 State, Action, and Reward Formulation

Based on the game rules, Gongzhu can be formulated as a multi-agent, imperfect-information game. In this project, I define the environment from the perspective of a single agent, such that its interaction with the environment can be modeled as a **Markov Decision Process (MDP)** under partial observability. Each playing card c is represented as a one-hot vector of length 52, allowing any collection of cards—such as a hand or a set of collected cards—to be encoded as a binary vector $v \in \{0, 1\}^{52}$.

Let the four players be denoted as $\{P_0, P_1, P_2, P_3\}$. At each time step t, the agent observes the complete game history $H_t = \{(c_j, \text{index}_j)\}_{j=0}^{t^*-1}$, where t^* is the number of cards played so far. Each tuple $(c_j, \text{index}_j) \in (\{0, 1\}^{52}, \{0, 1, 2, 3\})$ indicates that player P_{index_j} played card c_j at time step j. In addition, the agent observes its own current hand $h_t \in \{0, 1\}^{52}$. Given knowledge of the game rules and all past actions, the agent can deterministically infer additional public game state information, such as which cards have been played or collected by each player. Thus, the agent's **observed state** at time t is defined as: $s_t = (H_t, h_t)$. This formulation ensures that the environment is Markovian from the agent's perspective, as s_t contains all available and relevant information needed to select an action and predict future outcomes, given the rules.

The agent's **action** a_t is the selection of a valid card to play from its current hand, also represented as a one-hot vector in $\{0, 1\}^{52}$. Since each player plays exactly 13 cards per game, every episode contains exactly 13 decision steps (state-action pairs) per agent.

The **reward function** is sparse: all intermediate rewards are set to zero, and a terminal reward is given at the end of the game. Specifically, the reward $r \in \mathbb{R}$ is defined as the final score difference between the agent's team and the opposing team, scaled by a tunable hyperparameter. This design promotes long-term strategic planning rather than short-term gain.

1.3 Assumptions

The following assumptions are made throughout the scope of this project:

1. To ensure the project's relevance to real-world gameplay, **no assumptions are made about the optimality of the other players' strategies.** Moreover, each agent has no access to the strategies of their teammates or opponents. Therefore, agents of two players in the same team are not necessarily share the model.
2. **Players are not allowed to communicate with one another**. This constraint ensures that agents must make decisions solely based on their own observations of the current game state, defined as $s_t = (H_t, h_t)$.

2 Related Work

RL algorithms has been applied to imperfect information games such as No-Limit Hold'em [29, 25], Blackjack [14, 4], and Doudizhu [28, 11]. However, its application to Gongzhu remains largely under-explored. Existing efforts [26, 27] are limited by factors such as oversimplified objectives and scoring mechanisms [27], or the absence of reliable benchmarks [26, 27]. While related games like Contract Bridge have been more extensively studied [12, 17], with available benchmarks such as WBridge5 [5], most of these works focus on the bidding phase rather than the gameplay itself—primarily due to the greater modeling complexity involved in the latter.

2.1 Algorithms for Imperfect Information Games

Counterfactual Regret Minimization

The Counterfactual Regret (CFR) minimization framework [30] has been previously applied to imperfect information games such as No-Limit Hold'em [2, 16]. The main idea behind this framework is to estimate the difference in utility between the action actually taken and the best possible action (in hindsight), known as the counterfactual regret (CFR). By accumulating and minimizing these regrets over time, CFR converges toward a Nash equilibrium strategy in two-player zero-sum games.

In practice, however, estimating CFR with sufficient accuracy is often computationally expensive [16]. Moreover, the assumption underlying Nash equilibrium—that all players act rationally—conflicts with my assumption1. Therefore, **I did not adopt the CFR minimization framework in this project**.

Deep Monte-Carlo

Deep Monte-Carlo (DMC) is a generalization of classical Monte-Carlo (MC) methods that employs a deep neural network to approximate action-value functions $\hat{q}(s, a)$ [23]. To optimize a policy $\hat{\pi}$ using DMC, the following iterative procedure is repeated until convergence:

1. Sample an episode τ by acting according to the current policy $\hat{\pi}$, which is defined greedily with respect to the estimated action values: $\hat{\pi}(s) := \arg\max_a \hat{q}(s, a)$.
2. Update the estimates $\hat{q}(s, a)$ for all state-action pairs $(s, a) \in \tau$ using the observed returns from the episode.

In the context of imperfect information games, DMC has shown a strong ability to explore complex strategic dynamics effectively, especially in card games, while requiring a relatively modest computational budget [28]. For these reasons, **I selected DMC as the baseline algorithm for training my models**.

2.2 Evaluation Metrics

In zero-sum games, player strength is commonly evaluated using the Elo rating system [7]. In its standard form, each player is assigned a rating R, and the difference between two players' ratings determines their expected outcome. Specifically, the expected score of player a against player b is given by

$$E_{a\to b} = \frac{1}{1 + \exp\left(-\frac{R_a - R_b}{\sigma}\right)} \tag{1}$$

where σ is a scaling constant. After a game, player a's rating is updated based on the actual outcome $S_{a\to b}$, where $S = 1$ for a win, 0.5 for a draw, and 0 for a loss:

$$R'_a = R_a + K(S_{a\to b} - E_{a\to b}) \tag{2}$$

with K denoting the update rate (known as the $K-$factor). Game-specific evaluation metrics also exist, such as DouDizhu Evaluation [11] and Bridge International Match Points (IMP) [8]. However, these metrics are tailored to their respective games and cannot be directly applied to Gongzhu. Therefore, **the evaluation metric for this project is based on the Elo rating system**.

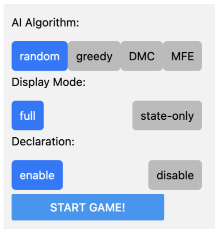
3 Approaches

3.1 Environment Implementation

I implemented the Gongzhu environment based on the OpenAI Gymnasium interface [24]. In addition to providing the core functionalities of the Gymnasium framework, I designed an Application Programming Interface (API) that enables external front-end applications to interact with the Gongzhu environment, including the GongzhuGUI described in Section 3.2. Moreover, the Gongzhu environment supports customizable declaration mechanism, allowing for flexibility in gameplay and enabling further research on Gongzhu AI.

3.2 GUI Implementation

To make my project accessible to the general public, I developed a graphical user interface (GUI) called GongzhuGUI using the React Native framework [9] and free illustrations from Irasutoya [15]. For entertainment purposes, GongzhuGUI allows users to play Gongzhu against different AI opponents with an optional declaration mode. Additionally, a built-in Gongzhu rules page helps beginners learn and enjoy the game. From a research perspective, GongzhuGUI provides a direct visualization of how a trained model plays, making its decision-making process more interpretable to the general audience. For instance, users can see the estimated action values of playing a particular card in real time. Some components of GongzhuGUI are shown in Figure 1.


AI Algorithm:

random | greedy | DMC | MFE

Display Mode:

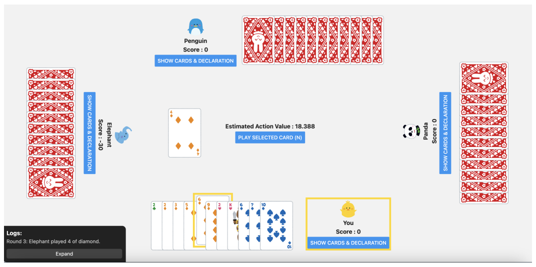
full | state-only

Declaration:

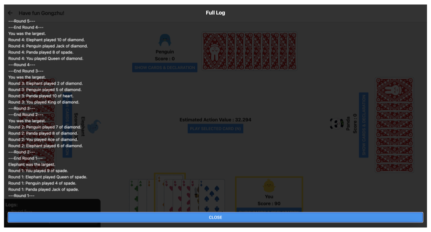
enable | disable

START GAME!

(a) Game Options



(b) Game Table

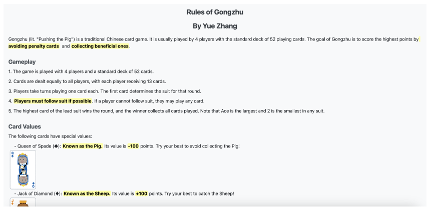

Have fun Gongzhu!

Full Log

```
---Round 5---
---End Round 4---
You was the largest.
Round 4: Elephant played 10 of diamond.
Round 4: Penguin played Jack of diamond.
Round 4: Panda played 8 of spade.
Round 4: You played Queen of diamond.
---Round 4---
---End Round 3---
You was the largest.
Round 3: Elephant played 2 of diamond.
Round 3: Penguin played 5 of diamond.
Round 3: Panda played 10 of heart.
Round 3: You played King of diamond.
---Round 3---
---End Round 2---
You was the largest.
Round 2: Penguin played 7 of diamond.
Round 2: Panda played 8 of diamond.
Round 2: You played Ace of diamond.
Round 2: Elephant played 6 of diamond.
---Round 2---
---End Round 1---
Elephant was the largest.
Round 1: You played 9 of spade.
Round 1: Elephant played Queen of spade.
Round 1: Penguin played 4 of spade.
Round 1: Panda played Jack of spade.
---Round 1---
```

CLOSE

(c) Game History


Rules of Gongzhu

By Yue Zhang

Gongzhu (lit. "Pushing the Pig") is a traditional Chinese card game. It is usually played by 4 players with the standard deck of 52 playing cards. The goal of Gongzhu is to score the highest points by **avoiding penalty cards** and **collecting beneficial ones**.

Gameplay

1. The game is played with 4 players and a standard deck of 52 cards.
2. Cards are dealt equally to all players, with each player receiving 13 cards.
3. Players take turns playing one card each. The first card determines the suit for that round.
4. **Players must follow suit if possible**. If a player cannot follow suit, they may play any card.
5. The highest card of the lead suit wins the round, and the winner collects all cards played. Note that Ace is the largest and 2 is the smallest in any suit.

Card Values

The following cards have special values:

- Queen of Spade (♠): **Known as the Pig.** Its value is **-100** points. Try your best to avoid collecting the Pig!
- Jack of Diamond (♦): **Known as the Sheep.** Its value is **+100** points. Try your best to catch the Sheep!

(d) Game Rules

Figure 1: Components of GongzhuGUI

3.3 Manual Feature Extraction

In theory, the agent can infer all relevant information solely from the game history H_t and its current hand h_t. However, in practice, learning to extract useful insights from raw data requires a highly complex and deep neural network, resulting in prohibitive training time. To mitigate this, I implemented Manual Feature Extraction (MFE) to pre-process the game state, distilling key features based on human decision-making heuristics in Gongzhu. This approach helps prune the model and significantly accelerates training. The extracted features are listed in Table2.

#	Feature	Shape
1	All played cards	$\{0,1\}^{52}$
2	Played cards by each player	$\{0,1\}^{4\times 52}$
3	Collected cards by each player	$\{0,1\}^{4\times 52}$
4	Whether the special cards have been played	$\{0,1\}^{4}$
5	Total number of cards played in each suit	$[0,13]^{4}$
6	Number of cards played in each suit by each player	$[0,13]^{4\times 4}$
7	Whether a player is guaranteed to be void in a suit	$\{0,1\}^{4\times 4}$
8	Whether a player has initiated each suit	$\{0,1\}^{4\times 4}$
9	Whether a player has a chance to collect all hearts	$\{0,1\}^{4}$
10	Whether the agent has cards higher than the special cards	$\{0,1\}^{4}$
11	Current score of each player	$[-600,800]^{4}$
12	Current score of each team	$[-600,800]^{2}$

Table 2: Manually extracted features used as input to the model. Features 1–6 represent general game state information observable by all players. Features 7–12 reflect more strategic or inferred knowledge commonly used in human decision-making. For example, Feature 9 identifies if any player has the opportunity to collect all hearts, i.e., no more than one player has collected hearts. If not, the agent may prioritize passing hearts to opponents rather than collecting them.

3.4 Neural Architecture

The agent's neural architecture consists of a Gated Recurrent Unit (GRU) and a Multi-Layer Perceptron (MLP) [28]. I chose GRU over the traditional LSTM due to its comparable performance while requiring a shorter training time [3]. Given a state $s_t = (H_t, h_t)$, the agent first processes the game history H_t through the GRU to obtain a feature vector of dimension 131, $GRU(H_t)$. This output is then concatenated with the flattened MFE output, $MFE(H_t)$, and the agent's hand h_t, forming the input vector for the MLP:

$$in_t = [GRU(H_t), MFE(H_t), h_t]$$

The concatenated vector in_t is then passed through an MLP with three hidden layers with hidden dimensions $[391, 255, 179]$, producing an output vector of length 52. Finally, the agent filters out illegal actions, selects the action with the highest estimated value among the legal options, and executes it greedily.

3.5 Training Procedure

Due to the absence of a real Gongzhu dataset, I opted to train the model using self-play algorithms. Initially, I implemented two baseline agents: a Random Agent and a Greedy Agent. The Random Agent selects a valid card uniformly at random, while the Greedy Agent follows a set of simple heuristics aimed at maximizing immediate gains. For example, when another player plays a heart, the Greedy Agent responds with the highest heart card that is still lower than the one played, if such a move is possible. These two agents are added to an agent pool, $\mathcal{A}$.

Next, I trained the agent described in Section 3.4 using the DMC algorithm [28]. In this framework, the learner agent is trained on batches of episodes sampled from parallel actor processes, each initialized with the same model parameters as the learner. These actors generate training data by playing against three agents randomly selected from the agent pool $\mathcal{A}$. The learner then updates its estimated action values using mean squared error (MSE) loss, optimized via traditional stochastic gradient descent (SGD).

To prevent the agent from overfitting to the weaknesses of specific opponents and to expose it to more diverse gameplay scenarios, I periodically added the learner agent to the agent pool $\mathcal{A}$. This selection was based on the K-best self-play strategy [29], ensuring that the agent continues to learn from stronger and more varied opponents over time.

3.6 Elo-based Evaluation with Margin of Victory

To evaluate the performance of different agents across diverse configurations, I designed an evaluation framework called the **Arena**. The Arena consists of multiple groups, where each group contains an equal number of players, all using the same agent model. Initially, all players are assigned the same Elo rating. The Arena proceeds by repeatedly selecting four players at random from the full pool to simulate a game. After each game, player ratings are updated using the standard Elo update rule. In the context of Gongzhu, which is played in teams of two, the rating of each team is computed as the average of the two players' individual ratings.

Since each agent model remains fixed during the evaluation phase, the true "skill" of each player is constant, justifying the use of the vanilla Elo update without adaptive terms. Furthermore, because it's difficult to assign individual credit within a team, both teammates receive the same rating update based on the team's performance. After running a large number of simulated games, we can estimate the relative strength of each agent model by analyzing the Elo rating statistics—particularly the mean and standard deviations (STDs)—of the players in each group.

In addition to win/loss outcomes, the **margin of victory (MOV)**—the score difference between teams—is also a critical performance indicator in Gongzhu, as the game objective is to maximize the score difference. To incorporate this into the Elo framework, I adopted the multiplicative MOV update scheme [13], which scales the rating updates based on the magnitude of the MOV. Specifically, let M denote the margin of victory for the winning team. The Elo update is then modified as follows:

$$R' = R + K(S - E)\log(1 + |M|) \tag{3}$$

This formulation ensures that games with larger margins of victory result in larger rating adjustments, encouraging agents to not only win but to win convincingly.

4 Empirical Studies

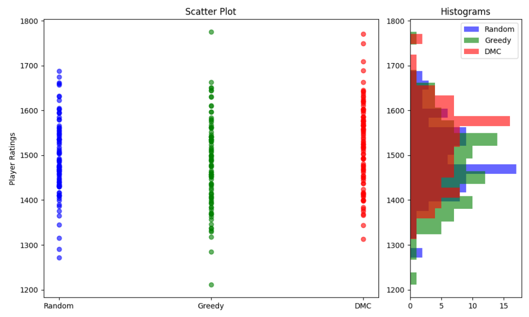
First, I trained a baseline DMC model without feature extraction, using a total of 10^6 samples. Next, I trained the MFE model following the procedure described in Section 3.5. Due to time constraints, training of the MFE model was limited to four iterations, each using 1.5×10^6 samples. The implementation of these training algorithms was inspired by DouZero [28], with $\gamma = 0.99$ and learning rate 0.0001.

For each evaluation using the Arena, I simulated 50,000 games. Constants related to Elo rating scheme are set to be the classical ones[7]: the initial rating of each player was set to 1500, with a scaling constant $\sigma = \frac{400}{\ln 10}$ and a K-factor of 16.

4.1 Evaluation Experiments

Firstly, I evaluated the performance of the baseline DMC model against the hard-coded Random and Greedy agents using the Arena. Each group has 100 players. As seen in Figure 2, the trained DMC model slightly outperforms the other two hard-coded agents, evidenced by the higher mean rating. However, the high STDs of all three agents indicate unstable performance and high volatility, suggesting that while the models show some level of success, there is significant variation in individual game outcomes.

Next, I evaluated the performance of the four generations of MFE agents against DMC and the two hard-coded agents. Each group consisted of 50 players, and the results are shown in Figure 3 and Table 3. On one hand, one MFE model outperformed all other agents, achieving an average Elo rating of 1522. On the other hand, the large STDs across all groups suggest significant inconsistency in agent performance. Interestingly, newer generations of MFE models did not necessarily outperform their predecessors. In fact, the MFE3 and MFE4 models had lower average ratings than all other models, which was an unexpected result.



	Random	Greedy	DMC
Mean	1501	1483	**1517**
STD	86.81	86.48	96.47

Figure 2: Random vs. Greedy vs. DMC

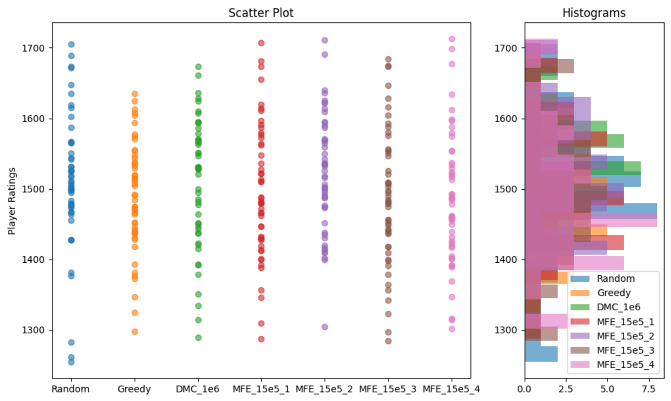
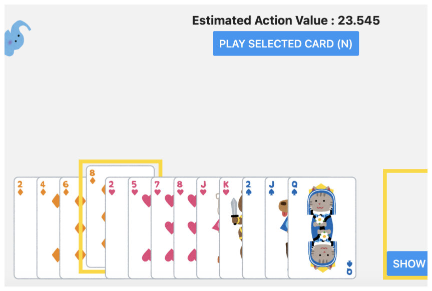


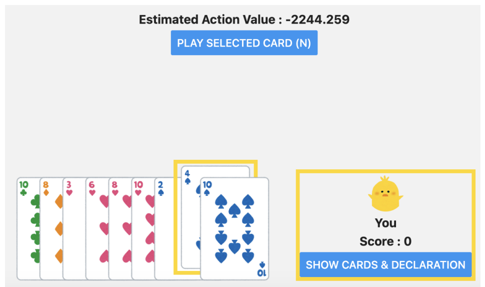
Figure 3: Random vs. Greedy vs. DMC vs. MFE

	Random	Greedy	DMC	MFE1	MFE2	MFE3	MFE4
Mean	1511	1490	1508	1497	**1522**	1488	1484
STD	95.82	77.72	91.06	94.74	83.08	96.75	94.42

Table 3: Random vs. Greedy vs. DMC vs. MFE



(a) A normal action value in the beginning of a game



(b) An unreasonable action value when some player obtained some score

Figure 4: Comparison of estimated action values in different stages of a game. The scale of estimated action value near end game was much larger than that in the beginning.

4.2 Analysis

Both of these experiments do not exhibit extraordinarily good performance from either the DMC or MFE models compared to the hard-coded agents. One direct reason for this could be the lack of sufficient training time. Typically, a high-performing RL model in card games requires training for weeks (or even months) on billions of samples [29, 28, 11, 14], whereas my models were trained on only several million samples. Therefore, the relatively limited training duration may have hindered the models' ability to fully capture the complexity of the game and optimize their performance.

In the MFE scheme, some extracted features may hinder the training process due to their unnormalized scales. For instance, I did not normalize the scale of the current scores, which can exceed 100, while most other features are binary. This issue is supported by the estimated action values. As shown in Figure 4, the estimated action values for cards are "normal" at the start of the game. However, once players collect cards with scores, these values grow drastically, sometimes exceeding theoretically possible values. This suggests that the unnormalized features may be distorting the model's learning process.

Furthermore, it is noteworthy that in both experiments, the performance of the Random Agent was not the worst; in fact, it even "defeated" the Greedy Agent in terms of average ratings! One plausible explanation lies in the assumption of my project that teammates in a team do not necessarily share the same decision strategies. The Random Agent, despite its seemingly ineffective strategy, may have benefited from its unpredictability, which could have disrupted the Greedy Agent's pattern-based decision-making. This unpredictability could lead to more favorable outcomes in certain scenarios, especially when the Greedy Agent's simple heuristics were countered by randomness. This may also impede other learnable models to extract consistent patterns from sampled game episodes.

5 Conclusions and Next Steps

This project introduces several contributions to the development of Gongzhu AI. First, I implemented a Gongzhu simulation environment along with an accessible graphical user interface, **GongzhuGUI**. This environment supports flexible game configurations, while the GUI allows a broader audience to interact with and evaluate different AI agents in an engaging and user-friendly way. Second, I proposed a **Manual Feature Extraction (MFE)** scheme for capturing informative features from Gongzhu game states. This scheme enables neural network-based agents to perform more effectively through training from deep Monto-Carlo (DMC). Finally, I designed an **Elo-based evaluation**

framework designed for Gongzhu that emphasizes an agent's ability to cooperate and compete with a diverse set of teammates and opponents.

However, there remains significant room for improvement. First, the performance of AI agents trained using DMC and MFE is inconsistent, with high variability across games. This underscores the need for further model refinement and more extensive training to enhance stability and reliability. Beyond algorithmic improvements, additional debugging and testing are also required for GongzhuGUI to ensure a polished and user-friendly experience before public release.

Next Steps

Specifically, some potential next steps are listed as follows:

- **Modification of the assumption.** As discussed in Section 4.2, the current assumption that teammates do not share the same model may hinder the learner model's ability to learn effectively. To make the project more tractable, this assumption should be revised so that both players on the same team use a shared model.
- **More training of models.** As previously discussed, the models require significantly more training time—potentially on an exponential scale—to achieve satisfactory performance. More training time could lead to more stable and effective policies, better generalization, and improved collaboration among agents.
- **Perspectives from game theory.** This project did not incorporate many perspectives from game theory, and neither the models nor the training algorithms employed game-theoretic techniques. A deeper exploration of game theory could provide valuable insights into the strategic dynamics of Gongzhu and potentially guide the design of more effective learning algorithms.
- **Adaptation of other algorithms.** In this project, I implemented a single training algorithm based on DMC, while vanilla DMC does not always guarantee smooth convergence. Therefore, I may explore alternative algorithms, such as Proximal Policy Optimization (PPO) [20] and Trust Region Policy Optimization (TRPO) [19], which are known for their stability and efficient convergence. Additionally, given the high variance of Gongzhu's rewards, distributional RL algorithms [1, 6] could offer valuable improvements by better capturing the distribution of returns and reducing training instability.
- **Better organization of codes.** Due to time constraints, the code developed for this project was neither well-organized nor thoroughly documented. To facilitate potential future collaboration with others, it will be important to revisit and restructure the codebase, ensuring clarity, maintainability, and proper documentation.
- **Integration of Declaration Mechanism.** In Gongzhu, the declaration mechanism is frequently used, making it an important aspect to study in terms of both dynamics and algorithm development. Given its similarity to the bidding process in Contract Bridge, existing models for bridge bidding [12, 17] may provide a useful starting point for developing algorithms to handle Gongzhu's declaration phase.

Acknowledgments

This project was completed as part of COMP5801: Intro Reinforcement Learning (Winter 2025) at Carleton University. I would like to express my sincere gratitude to Prof. Junfeng Wen for his detailed and insightful instruction throughout the course. I am also thankful to my classmates Tristan Gaudreault and Justin Zhang for their valuable feedback on this report. Additionally, I extend my appreciation to Prof. Yongyi Mao and my colleagues in his research group, who have played Gongzhu with me over the months and provided countless insights into the strategic depth of this fascinating game.

References

[1] Marc G. Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. *CoRR*, abs/1707.06887, 2017.

[2] Noam Brown and Tuomas Sandholm. Superhuman ai for heads-up no-limit poker: Libratus beats top professionals. *Science*, 359(6374):418–424, 2018.

[3] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

[4] Ron Coleman and Bashir Dahir. Toward gamification design of molecular-level filters through reinforcement learning of blackjack. In *2023 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 296–301, 2023.

[5] Yves Costel. Wbridge5. `http://www.wbridge5.com/`, 2021. Winner of the 2021 World Computer-Bridge Championship.

[6] Will Dabney, Mark Rowland, Marc G. Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. *CoRR*, abs/1710.10044, 2017.

[7] Arpad E. Elo. The proposed uscf rating system, its development, theory, and applications, August 1967.

[8] World Bridge Federation. *The Laws of Duplicate Bridge 2017*. World Bridge Federation, 2017.

[9] Meta (formerly Facebook). React native: A framework for building native apps using react. `https://reactnative.dev/`, 2015. Accessed: 2025-04-03.

[10] Feng-hsiung Hsu, Murray S. Campbell, and A. Joseph Hoane. Deep blue system overview. In *Proceedings of the 9th International Conference on Supercomputing*, ICS '95, page 240–244, New York, NY, USA, 1995. Association for Computing Machinery.

[11] Qiqi Jiang, Kuangzheng Li, Boyao Du, Hao Chen, and Hai Fang. Deltadou: Expert-level doudizhu ai through self-play. *International Joint Conference on Artificial Intelligence*, pages 1265–1271, 08 2019.

[12] Haruka Kita, Sotetsu Koyamada, Yotaro Yamaguchi, and Shin Ishii. A simple, solid, and reproducible baseline for bridge bidding ai, 2024.

[13] Stephanie Kovalchik. Extension of the elo rating system to margin of victory. *International Journal of Forecasting*, 36(4):1329–1341, 2020.

[14] Ziang Liu and Gabriel Spil. Learning explainable policy for playing blackjack using deep reinforcement learning (reinforcement learning), 2021.

[15] Takashi Mifune. Irasutoya: Free japanese-style illustrations. `https://www.irasutoya.com/`, 2012. Accessed: 2025-04-03.

[16] Matej Moravčík, Martin Schmid, Neil Burch, Viliam Lisý, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 356(6337):508–513, 2017.

[17] Zizhang Qiu, Shouguang Wang, Dan You, and MengChu Zhou. Bridge bidding via deep reinforcement learning and belief monte carlo search. *IEEE/CAA Journal of Automatica Sinica*, 11(10):2111–2122, 2024.

[18] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy P. Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model. *CoRR*, abs/1911.08265, 2019.

[19] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015.

[20] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[21] Game Programming Seminar. GPS Shogi. `https://gps.tanaka.ecc.u-tokyo.ac.jp/gpsshogi/`. Accessed: 2025-04-05.

[22] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[23] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 2nd edition, 2018.

[24] Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. Gymnasium: A standard interface for reinforcement learning environments, 2024.

[25] Ke Wang, Dongdong Bai, and Qibin Zhou. Deepholdem: An efficient end-to-end texas hold'em artificial intelligence fusion of algorithmic game theory and game information. In *2022 IEEE 8th International Conference on Computer and Communications (ICCC)*, pages 2313–2320, 2022.

[26] Licheng Wu, Qifei Wu, Hongming Zhong, and Xiali Li. "gongzhu" strategy based on convolutional neural network. In Fuchun Sun, Jianmin Li, Huaping Liu, and Zhongyi Chu, editors, *Cognitive Computation and Systems*, pages 117–128, Singapore, 2023. Springer Nature Singapore.

[27] Licheng Wu, Qifei Wu, Hongming Zhong, and Xiali Li. Mastering "gongzhu" with self-play deep reinforcement learning. In Fuchun Sun, Angelo Cangelosi, Jianwei Zhang, Yuanlong Yu, Huaping Liu, and Bin Fang, editors, *Cognitive Systems and Information Processing*, pages 148–158, Singapore, 2023. Springer Nature Singapore.

[28] Daochen Zha, Jingru Xie, Wenye Ma, Sheng Zhang, Xiangru Lian, Xia Hu, and Ji Liu. Douzero: Mastering doudizhu with self-play deep reinforcement learning. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 12333–12344. PMLR, 18–24 Jul 2021.

[29] Enmin Zhao, Renye Yan, Jinqiu Li, Kai Li, and Junliang Xing. Alphaholdem: High-performance artificial intelligence for heads-up no-limit poker via end-to-end reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(4):4689–4697, Jun. 2022.

[30] Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. In *Proceedings of the 21st International Conference on Neural Information Processing Systems*, NIPS'07, page 1729–1736, Red Hook, NY, USA, 2007. Curran Associates Inc.