Designing a Reliable *Crew* Member

Stanford CS238: Decision Making Under Uncertainty

Matthew Harvill
Department of Computer Science
Stanford University
`mharvill@stanford.edu`

Rajan Vivek
Department of Computer Science
Stanford University
`rvivek@stanford.edu`

Abstract

We demonstrate the effectiveness of a modified Monte Carlo Tree Search (MCTS) algorithm with the UCB-1 selection algorithm for learning to play a cooperative trick-taking game, *The Crew*. We show that our agent surpasses the performance of a random agent by a large margin at various game depths and is fairly robust to unreliable teammates. We also explore a deep learning approach inspired by Alpha Zero, discuss preliminary findings, and recommend future directions.

1 Introduction

While extensive prior work has explored reinforcement learning algorithms for adversarial games with both perfect information Silver et al. (2017), V. et al. (2018) and imperfect information Brown et al. (2020), fewer works have explored reinforcement learning for collaborative games with imperfect information. We explore this niche by developing an agent to play *The Crew: Mission Deep Sea*, a collaborative trick-taking game. A trick-taking game is one in which there are rounds of play, called tricks, each of which are evaluated to determine a winner or taker of that trick. Since this game is collaborative, all players win together or lose together based on whether everyone completes their own tasks in the fixed number of rounds. Example tasks include 'I will only win the first trick', 'I won't win any 1 cards', and 'I will win a 2 and a pink card in the same trick'. This is a sequential decision-making problem because the agent has to make good decisions for what cards to play based on what has happened in the previous rounds and what cards remain in its hand as well as the cards that remain distributed amongst the other players. Sources of uncertainty include state uncertainty– the agent doesn't know which cards are in which teammates' hands– as well as interaction uncertainty– the agent doesn't know how its teammates will play. We develop and characterize the performance of an agent that using Monte Carlo Tree Search to solve this problem.

Our contributions are as follows:

1. We formalize the game as a POMDP and develop a Monte Carlo Tree Search agent that it is able to significantly surpass the win rate of random agents during self-play.
2. We characterize the performance of our agent across various game depths (i.e. cards in each players' hand) and its robustness against unreliable teammates with a novel metric.
3. We present our initial findings for a deep learning-based agent modelled after Alpha Zero and recommend next steps for training such an agent.

2 Related Work

Monte Carlo Tree Search (MCTS), originally proposed by Coulom (2007), is a powerful online planning algorithm that is particularly effective for turn-based games. Silver and Veness (2010) present Partially Observable Monte Carlo Planning (POMCP), an extension to this algorithm that accounts for state uncertainty by tracking exploration and learned values of histories– sequences of actions and observations– rather than states. Many modern works have built upon these foundations including Van den Broeck et al. (2009) which proposes modifications to the standard MCTS selection and backpropogation strategies to handle opponent uncertainty. While not originally designed for POMDPs, AlphaZero (Silver et al., 2017) captured the world's attention for achieving superhuman performance at Go– a fully observable game – through the integration of Monte Carlo Tree Search and deep learning. Subsequent works such as Zhang et al. (2021) have adapted the Alpha Zero framework for POMDPs.

3 Approach

3.1 Game Set-Up

Each game of *The Crew* consists of each player being dealt N cards randomly selected from the play deck of 40 cards, where $N = 10$ for a standard 4-player game. M task cards are then randomly drawn from the task deck and revealed to all players. In a predetermined order, each player has the opportunity to accept a task or pass, indicating they do not want to take on any remaining tasks. The last players are required to take any tasks that remain. Each round consists of each player playing one card from their hand. The player with the "4 of sub" card in their starting hand begins the first round. Players must follow the same suit, i.e. "trump", of the starting player if possible. The player who plays the highest value "sub" card or– if no "sub" is played– highest card matching the trump suit wins the trick. The trick winner starts the next trick. All of our simulations use 4 players.

We establish a set of simplifications to the original *The Crew* rules to make the game more tractable for our MCTS agent:

1. We remove the game mechanic for communication between players via tokens. Thus, each player can only see their own hand.
2. We select only 5 tasks from the 100 in the original game and slightly modify some so that they work with smaller subsets of cards (for $N < 10$): *'I will win fewer tricks than the captain', 'I will win no yellow/green cards', 'I will win exactly two tricks', 'I will win a trick using a 9', 'I will win the green 7 and blue 8'*. In each game, $M = 2$ tasks are randomly sampled from these 5.
3. We reduce the number of cards in each player's hand from 10 to $N \in \{3, 4, 5, 6\}$.

3.2 POMDP Formulation

We formalize our game as a POMDP due to the inherent state uncertainty. Our state space consists of all permutations of 1) possible starting hands of size N sampled without replacement from the deck of 40 cards for each of 4 players, 2) possible task assignments consisting of 2 players each assigned 1 task sampled from 5 tasks, 3) tasks completed at a given round, 4) cards played in the current round, and 5) cards won by each player. Our action space consists of each of the 40 playing cards as well as the option to select a certain task or pass during the task selection round. (Of course, only a subset of the actions are valid for a given player at a given turn). The observation space consists of 1) a player's own hand, 2) the tasks selected for the group, 3) the task assignments for each player, and 4) other players' moves in a given round. Our agent receives a reward of 1 if the team completes all tasks by the end of the game and a reward of 0 otherwise. No discounting is used.

3.3 Monte Carlo Tree Search Approach

Monte Carlo Tree Search is an online planning technique that can be applied to POMDPs by estimating the value $Q(h, a)$ of history-action pairs. Each history consists of all previous actions and observations. The algorithm takes as input a belief b, rollout policy π, depth d, and factor c. Next, M simulations are run. Each simulation consists of sampling a starting state s, progressing d levels down the game tree by selecting actions according to the UCB-1 heuristic, and recording both the visit count $N(h, a)$ and average discounted reward $Q(h, a)$ of each history-action pair. The UCB-1 heuristic balances exploration and exploitation, where a larger c value results in more exploration. The UCB-1 heuristic is given by

$$Q(h, a) + c * \sqrt{\frac{logN(h)}{N(h,a}}, \text{ where } N(h) = \textstyle\sum_a N(h, a)$$

For training, we set $c = \sqrt{2}$ since this is the original value used for UCB-1 and the only nonzero reward is R = 1 for winning the game. We make a series of modifications to the algorithm for our implementation. First, our starting state s always consists of the beginning of a game with randomly-initialized hands. Thus, a belief b is not required, as we make the assumption that all cards not in one's hand are equally likely to be in the other players' hands. Next, each simulation continues until the game is won or lost, obviating the need for a rollout policy. Finally, to make the problem more tractable we replace each history with the current "visible" state consisting of the agent's current hand, the tasks left to be completed, the task assignments, and the number of players that have not yet played in the current round. When evaluating our variant of MCTS's policy, we set $c = 0$ to take the best action based on our previous exploration.

3.4 Alpha Zero-Inspired Approach

3.4.1 Alpha Zero Background

As a stretch goal, we aimed to determine whether an agent akin to Alpha Zero could master *The Crew*. Because the Alpha Zero framework was designed for fully-observable games, we experimented with relaxing our game by making all player hands public.

Alpha Zero uses a neural network p_θ that takes the current state of the s as input and outputs a predicted value of the state $v_\theta(s) \in [-1, 1]$ and policy $p_\theta(s)$ as a probability vector over all possible actions. The network engages in self-play, resulting in training examples (s_t, π_t, z_t), where s_t is a game state, π_t is an estimated target policy, and $z_t \in \{-1, 1\}$ represents whether the move ultimately led to a game win or loss. The network is trained using the loss function

$$l = \sum_t (v_\theta(s_t) - z_t)^2 - \pi_t * log(p_\theta(s_t))$$

To collect training data, the network engages in self-play using a modified Monte Carlo Tree Search algorithm. For each node the algorithm maintains the expected reward estimate $Q(s, a)$, the number of times the agent has taken action a in state s $N(s, a)$, and the network's predicted policy $P(s)$. Thee are used to calculate an upper confidence bound on our reward estimates:

$$U(s, a) = Q(s, a) + c_{puct} * P(s, a) * \frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)}$$

For each game simulation, an empty game tree is initialized and Monte Carlo search is run M times. If a previously unvisited node is encountered, $P(s)$ is initialized to $p_\theta(s)$, $Q(s, a)$ to 0, and $N(s, a)$ to zero. Next, the network's predicted value $v_\theta(s)$ is propogated upward to update $Q(s, a)$ for all parent nodes in the current game path. If a visited node is encountered, the search proceeds with the action that maximizes $U(s, a)$. After M simulations are completed, a target policy for each node is formulated as

$$\pi(s) = \frac{N(s,a)}{\sum_b N(s,b)}.$$

Thus, we get a training example for each visited state. The network is trained using training examples from many simulations. Next, it is evaluated at self-play, where each action is chosen by running MCTS from the current state and sampling from the resulting target policy. If the new network surpasses the performance of the old network, it replaces the old network. This process is repeated many times until the agent's performance converges.

3.4.2 Alpha-Zero Inspired *Crew* Agent

In our experiments, we used a transformer as our network Vaswani et al. (2017). This was motivated by the fact that our game has many distinct components including cards and player roles. Transformers are a powerful technique for learning rich representations of entities (i.e. tokens) that capture inter-entity relationships. We hoped this architecture would better model our game state space than a vanilla neural network.

We represented game states as a sequence of tokens. Each sequence contains information specifying all information in the game state, as described in Section 3.2. An example game state is:

"CLS CURRENT_PLAYER_A CURRENT_CAPTAIN_B TASK_0_A TASK_1_B TASK_0_COMPLETE TASK_1_INCOMPLETE GREEN_1_HAND_A ... SUB_4_PLAYED_B ... SUB_3_WON_C"

This represents that the current player to make a move is player A, the current game captain is player B, task 0 is assigned to player A and complete, task 1 is assigned to player B and incomplete, the card "Green 1" is in player A's hand, the card "Sub 4" has been played by player B in this round, and the card "Sub 3" has been won by player C. Each game state is 48 tokens long (standard game) and there are 179 unique tokens. The CLS token is placed at the beginning of every state sequence. This embedding of this token was used by a prediction head network at the top of the transformer to output action probabilities and state values. Our network consisted of three transformer encoder layers, a hidden size of 64, and 8 self-attention heads. Each agent training run consisted of training for 3 epochs on the previously generated examples from MCTS with a learning rate of 5e-4 and batch size of 32.

3.5 Evaluation Techniques

To evaluate each our agent in each game setting, we use three metrics:

1. Absolute win rate out of 10,000 games of self-play
2. The ratio of self-play win rate relative to stochastic agent win rate
3. Robustness to Teammate Unreliability (RTU)

We note that self-play is not necessarily representative of effective collaboration because an agent can overfit to collaborating with instantiations of itself. Thus, we formulate the Robustness to Teammate Unreliability (RTU) metric, which quantifies the degradation in performance of an agent as its teammates become increasingly more stochastic. The RTU metric is given by

$$RTU(agent) = \frac{\int_0^1 WinRate(Agent, Teammates(r))\, dr - WinRate(StochasticAgent, Teammates(0))}{WinRate(Agent, Teammates(1)) - WinRate(StochasticAgent, Teammates(0))}$$

where $Agent$ denotes an agent that selects the best action according to MCTS estimates and $Teammates(r)$ denotes a simulated team of reliability r where members select the best action with probability r and a random action otherwise.

This metric measures the normalized area between MCTS agent and stochastic agent win rate curves as teammates vary between complete reliability (which is equivalent to self-play) and complete stochasticity. An RTU score of 1 indicates that the agent's win rate does not degrade at all as teammate reliability decreases. We estimate the integral with a trapezoidal Reimann sum.

4 Results

The results below were attained after running 102,400 Monte Carlo rollouts using UCB1 from randomly initialized starting positions to game completion (Training iterations). Note that our Alpha-Zero Inspired Agent did not surpass random performance due to convergence issues. We do not report quantitative results for these experiments, but discuss the results further in our Analysis section.

Cards per Player	Stochastic Agent Win Rate	MCTS Agent Win Rate	Win Rate Ratio	RTU
3	0.047	0.220	4.68	0.48
4	0.058	0.166	2.87	0.57
5	0.047	0.078	1.67	0.60
6	0.043	0.053	1.24	0.79

Table 1: Monte Carlo Tree Search Agent Evaluation

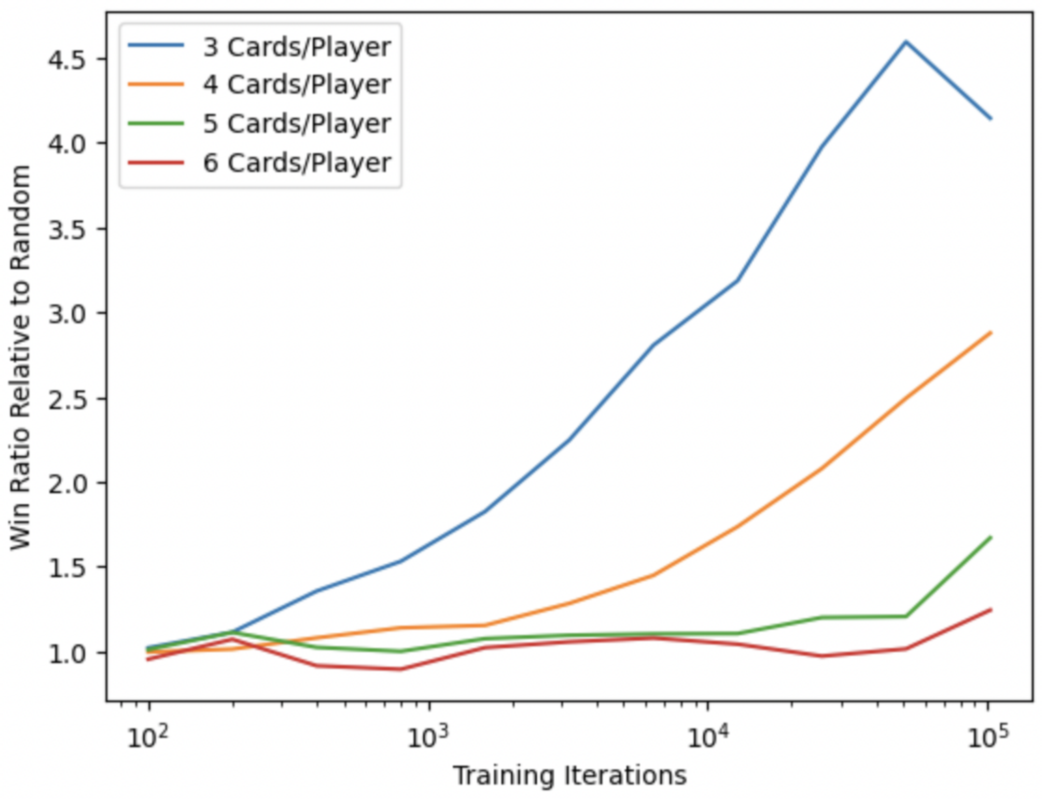

Figure 1: Win Rate Ratio of MCTS Agent to Stochastic Agent Throughout Training

5 Analysis

5.1 Monte Carlo Tree Search Agent

We observe that the MCTS agent outperforms the stochastic agent in all game settings tested, but the agent's gains diminish as the game depth (i.e the number of cards per player) increases. This diminishing increase in performance can be attributed to the greater difficulty in estimating action values in larger state and action spaces.

Interestingly, we observe that the stochastic win rate is highest for the 4 cards per player game. We believe this is because having fewer than four rounds forces the agent to win faster (placing more importance on each action). With more than 4 rounds, the players have more time to complete the tasks but there are more opportunities for fatal mistakes that lose the game. It appears that the 4 cards per player setup best balances these competing factors.

We observe that our agent is fairly robust in all game settings, achieving an RTU score greater than 0.5 when 4 or more cards per player are used. These RTU scores are visualized in Appendix A. We note that RTU tends to increase as the win rate ratio decreases. This is likely because a lower win rate ratio means that the agent has not overfitted to self-play games, where all players choose optimal actions. Future work should explore training the agent with unreliable teammates in order to decrease this overfitting and increase robustness.

5.2 Alpha-Zero Inspired Agent

Unfortunately we did not complete our stretch goal of implementing our Alpha Zero-inspired agent that masters *The Crew*. Our largest issue was the transformer's need for significant training data and slow convergence time. Transformers require very large amounts of training data in order to learn rich representations for each token, with modern transformer systems being trained on millions of token sequences (Vaswani et al., 2017). Our system was bottlenecked by the data generation rate of our single-thread MCTS implementation. While the transformer loss did decrease upon each training run, the MCTS data collection and training iteration time was too long for our agent to achieve above random win rates with our compute budget. In the future, we hope to demonstrate that such an agent can master self-play games of *The Crew* and be extended using techniques such as Zhang et al. (2021) to play the original version of the game where player hands are hidden.

6 Conclusion

We demonstrate that an agent that uses a modified version of Monte Carlo Tree search is able to surpass random performance in *The Crew* by a significant margin. We propose the Robustness to Teammate Unreliability metric to quantify agent robustness and show that our designed agent is fairly robust, with increased robustness in deeper games.

We show that our Monte Carlo technique does not scale well as the game depth increases and propose a path forward using an Alpha Zero-inspired agent. We recommend that further work explores our Alpha Zero formulation with greater computational resources.

7 Contributions

MH implemented the majority of the game simulator, developed evaluation scripts for our techniques, and implemented our Monte Carlo Agent.

RV helped implement the game simulator, experimented with Monte Carlo Tree search variations, and proposed and implemented the RTU metric. RV also implemented the Alpha Crew agent and experimented extensively with transformer training and Alpha Zero MCTS debugging.

Both team members equally contributed to formulating the project and experiments, plotting and analyzing results, and writing the report.

References

Guy Van den Broeck, Kurt Driessens, and Jan Ramon. 2009. Monte-carlo tree search in poker using expected reward distributions. In *Advances in Machine Learning*, pages 367–381, Berlin, Heidelberg. Springer Berlin Heidelberg.

Noam Brown, Anton Bakhtin, Adam Lerer, and Qucheng Gong. 2020. Combining deep reinforcement learning and search for imperfect-information games.

Rémi Coulom. 2007. Efficient selectivity and backup operators in monte-carlo tree search. In *Computers and Games*, pages 72–83, Berlin, Heidelberg. Springer Berlin Heidelberg.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. 2017. Mastering chess and shogi by self-play with a general reinforcement learning algorithm.

David Silver and Joel Veness. 2010. Monte-carlo planning in large pomdps. In *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc.

Sai Krishna G. V., Kyle Goyette, Ahmad Chamseddine, and Breandan Considine. 2018. Deep pepper: Expert iteration based chess agent in the reinforcement learning setting.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Yunsheng Zhang, Dong Yan, Bei Shi, Haobo Fu, Qiang Fu, Hang Su, Jun Zhu, and Ning Chen. 2021. Combining tree search and action prediction for state-of-the-art performance in doudizhu. pages 3413–3419.

8 Appendix

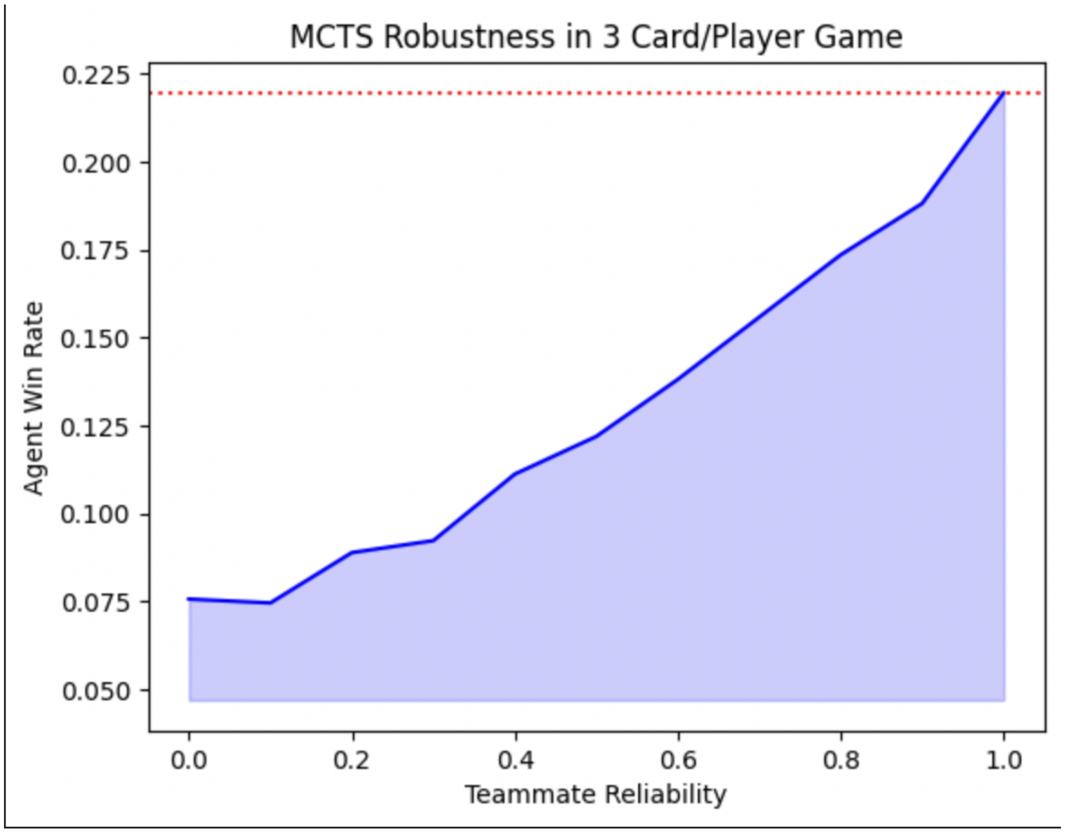

Figure 2: MCTS Agent robustness across varying levels of teammate reliability in the 3 card per player game setup. The shaded area indicates RTU = 0.48. Note that the shaded area is lower bounded by the average win rate of a stochastic agent.

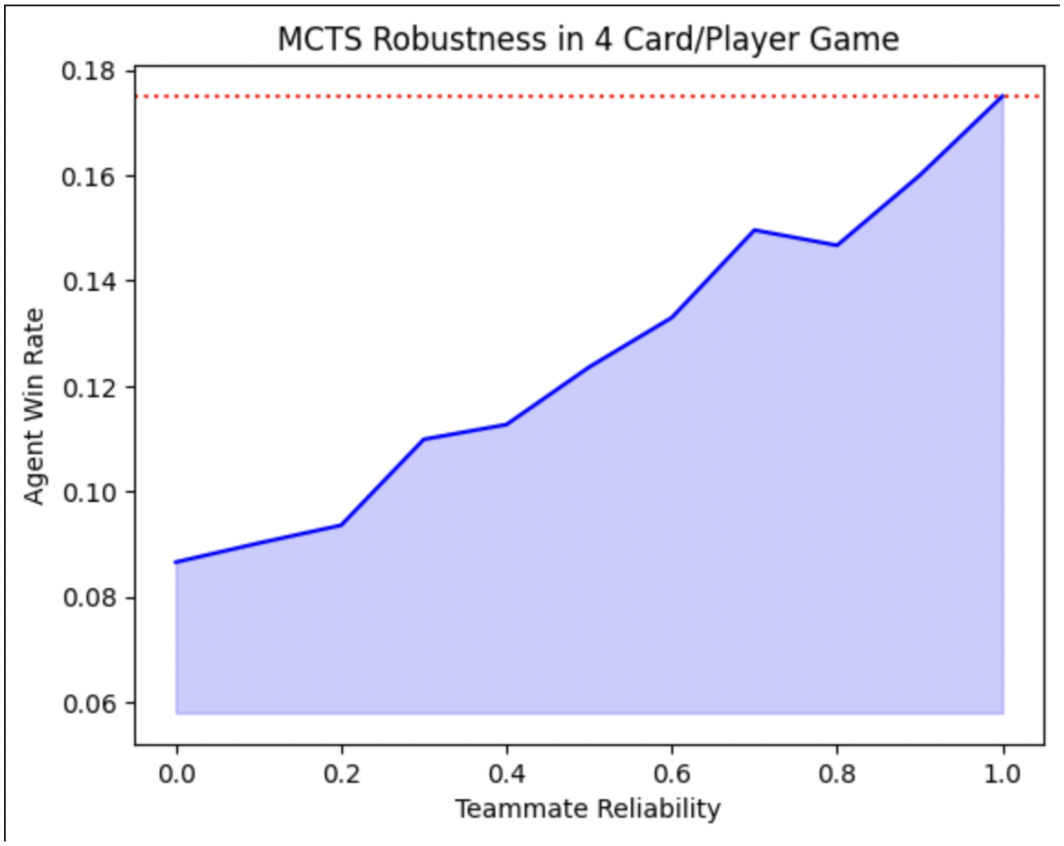

Figure 3: MCTS Agent robustness across varying levels of teammate reliability in the 4 card per player game setup. The shaded area indicates RTU = 0.57. Note that the shaded area is lower bounded by the average win rate of a stochastic agent.

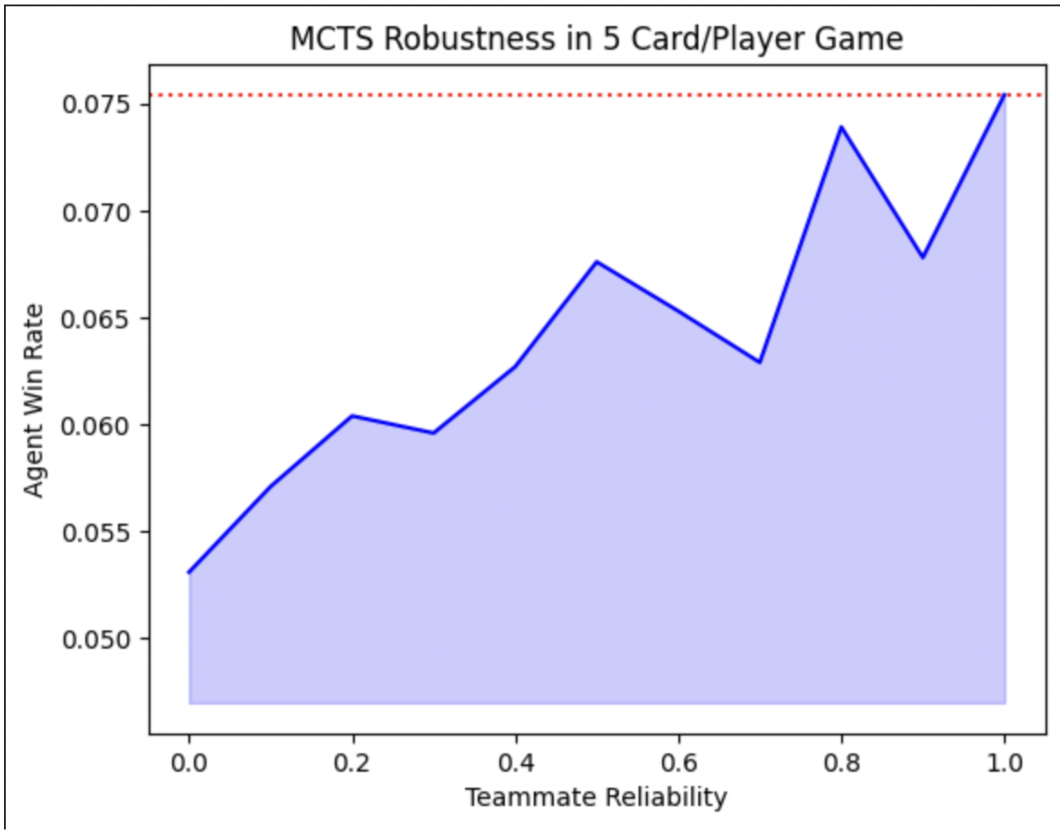

Figure 4: MCTS Agent robustness across varying levels of teammate reliability in the 5 card per player game setup. The shaded area indicates RTU = 0.60. Note that the shaded area is lower bounded by the average win rate of a stochastic agent.

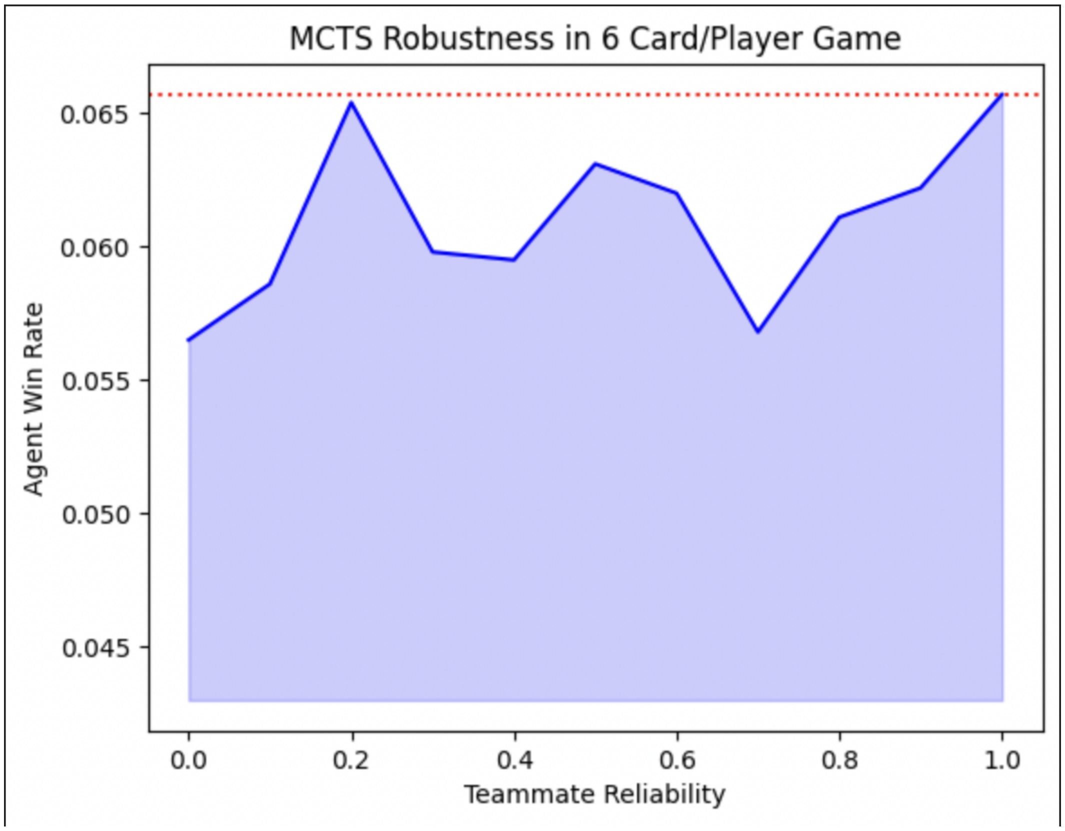

Figure 5: MCTS Agent robustness across varying levels of teammate reliability in the 6 card per player game setup. The shaded area indicates RTU = 0.79. Note that the shaded area is lower bounded by the average win rate of a stochastic agent.