Euchre AI

August Nord
Earlham College
Richmond, Indiana, USA

Abstract

Euchre is a cooperative-style, imperfect information card game played using a subset of cards from a standard playing card deck. Players are paired with a partner and attempt to win the most points for their team while not being able to know the cards in anyone's hand but their own. These elements are perhaps what has made it a popular game in the United States Midwest, but they also pose challenges when trying to create artificial intelligence agents to play the game. This paper will first cover the basics of the game, its strategies, and the previous research done in this field. Next, it will overview the methods, such as Reinforcement Learning and Monte Carlo Tree Search, that we explored in the creation of our agent models. And finally, it will share the results and conclusions made from testing these agents.

1 Introduction

Games have long been a popular field of study in computer science as they provide a space in which the behavior of methods and algorithms can be studied. Further, there are two main categories of board and card games: those of "perfect information" and those of "imperfect information." Perfect information refers to games where the complete state of the game is viewable to every player. Among these, chess and go are probably the most well known. In Euchre and other imperfect information games, however, some of the information about the current game state is hidden from the player. Thus, a player must use the information they have—primarily the cards in their hand and the cards that have been played—to make an assessment on what card might be the best move. They do this while not knowing how the other cards are distributed between the deck and the other players.

Another aspect that makes Euchre particularly interesting to study is that it is played with a partner. This shifts the emphasis from each individual greedily trying to win to cooperating with each other in order to come out victorious together.

Euchre has been researched before [1] [4] [5] [6], but many approaches focus on only the main part of game play—the playing phase. This study aims to broaden this scope as it takes a look at a slightly smaller, but equally important, aspect of the game—the calling phase. We will create agents that use a number of methods that have been studied, and then test their performance against each other to see which win most frequently. By combining this work together, we hope to create the algorithmic foundation for a more satisfying playable version of the game.

2 Background

2.1 Game Basics

As mentioned, there are two phases in a hand of Euchre: the calling phase and the playing phase. Before either of these begin, the four players are dealt five cards apiece with the remaining four cards placed aside as the "kitty." Next, the top card of the kitty is flipped up and the calling phase begins. Each individual has a chance, in turn, to decide if they would like the suit of this card to become the trump suit. A player who calls asserts that, based on their current hand and with the help of their partner, they believe they can win the round. If the suit is called, the player who dealt picks up the face-up card and discards one from their hand face down. If the suit is not called, players have a chance, in turn, to decide if they would like to call one of the other three suits as trump. If nothing is called, either the dealer is forced to call or the hand is redealt.

Once trump has been decided, the playing phase begins. Play happens in five rounds, or "tricks," similar to play in the card games Hearts, Spades, and Bridge. In these tricks, each player chooses one card to play in turn. Once everyone has had a chance to play, whoever played the highest value card takes the trick for their team. The team who takes three or more of the five tricks scores points from that hand. Typically, a number of hands are played until one team reaches 10 points.

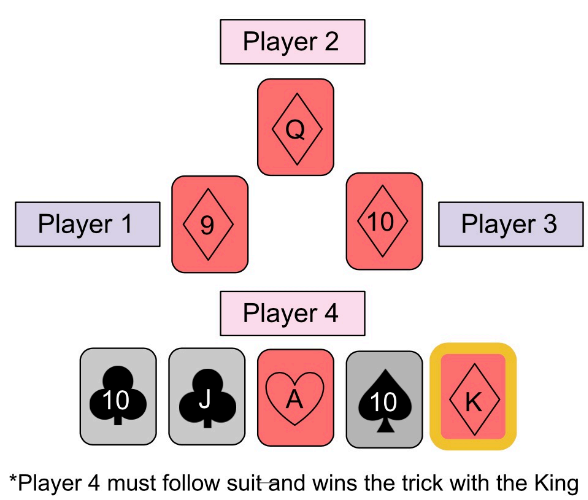

Figure 1. Sample hand from a Euchre Game

2.2 Know the Rules

At any point in the playing phase, a player has at most five available cards to choose from. However, even when this is the case, often the rules dictate that there are fewer than five valid plays. Due to this small number of choices, simply having an agent that knows the rules of the game allows for surprisingly good performance. In his thesis, Holmes plays agents who have been given the rules against those whom had to learn them through training observation. He finds that overwhelmingly the rule-following agents preform better [4].

Similarly, in Seelbinder's thesis, he creates a RANDOM agent that simply picks randomly between valid plays. He is surprised to find it often beats his agents that play with simple HIGH and LOW strategies [6].

2.3 Cooperative Play

The HIGH agent and it's slightly more sophisticated HIGH! counterpart always play high or always play high if they have any chance at winning, respectively. In this way, winning each hand is prioritized. These two perform well, but lack when it comes to cooperating with the partner. Since tricks taken by the partner also count for the team, to play your high cards when your partner already has the trick is often a poor decision in the long run. Thus, Seelbinder explores a User Friendly (UF) agent that yields to the partner in such instances. However, he finds that this agent has the opposite problem and is instead too passive. His UF.5 agent combines the aggressive HIGH! and passive UF strategies to form an even stronger agent. In terms of cooperation, Seelbinder also finds that having teams that play different strategies can also be effective. For example, the team of LOW and HIGH outperformed a double RANDOM team. All this evidence indicates that in whatever approach we take, a balance between passive and aggressive play styles amongst teammates must be fine tuned.

2.4 MDPs

One common way to frame the game of Euchre is as a set of Markov Decision Processes (MDPs). In an MDP, we define the state of the game, a number of actions that could be taken, resulting rewards from those actions, and finally the new state of the game. Our goal then is to apply a variety of methods to figure out which action leads to the highest reward. In the playing phase of Euchre, this looks like deciding which card is the best card to play to win the round. However, due to hidden state information (i.e., not knowing what cards each player has), we cannot be sure what the eventual reward will be, but rather must use what we know to make a guess. This particular case is referred to as Partially Observable MDPs (POMDPs), as the states are not fully observable [7].

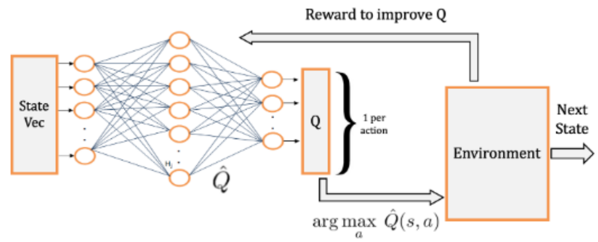

Figure 2. The Structure of Deep Q-Learning [5]

3 Methods

3.1 Reinforcement Learning

Reinforcement Learning (RL) is one way to tackle MDPs and POMDPs. RL techniques set the agent up to experiment around the domain and then use what it learns based on the rewards of those simulations to create a structure of how to behave in a way that maximizes rewards. In Euchre, this is more complex than the HIGH! agent in that it looks not just at short term reward of winning the trick, but aims to maximize rewards across the whole hand. In his Euchre research, Pugh [5] explores two types of RL: Deep Q-learning (DQL) and Neural Fictitious Self-Play (NFSP).

In RL, the letter q is often used to refer to the expected reward of any state, action pair. Q-learning is a set of techniques then that looks to find the optimal q-value. Often this is done by updating a table of q-values as a series of simulations is run. DQL is a variant of this, that instead uses a neural network to try to find the optimal q-value for the state, action pair. Pugh's example of this structure can be seen in Figure 1. Note the neural network that takes in the state information and returns a set of q-values.

NFSP is a RL approach introduced by Heinrich and Silver [3]. It uses two neural networks to learn. The first trains from information playing against other agents. The second, does a supervised learning using the data from its own play. It combines knowledge of these two styles together to decide on the best course of action.

Though the DQL and NFSP agents do not vastly outperform rule-based models, they do perform well enough to merit more exploration. Pugh's data set is available on github and is an extension of a program called RLCard [8]. This toolkit is also openly sourced on github. It is designed specifically to allow for exploration of RL techniques in card games.

3.2 Monte Carlo Search Trees

Another approach for handling imperfect information is to run a set of simulations where you pretend that all the information is known. This method is referred to as determinization or Monte Carlo sampling [2]. In Euchre, this would look like the system imagining a number of possible hands, running games to determine what the best card to play in each game is, and then averaging that out to decide what the best move would be. This is extended into Monte Carlo Search Trees (MCST) which also runs a series of simulations to make an estimate. MCST views the game as a search tree and the actions as the edges in that tree. It takes the current node or state, selects one of its children, and runs a simulation of random moves from that point to see if it reaches a win. If it does, it positively updates that node's probability of winning. If it doesn't, it negatively updates it. On this first pass through, since the probability is only based on the one simulation, it is likely not very accurate. However, as more and more decisions are explored, it is gradually able to gain a better picture of the state space. It then selects the action with the highest probability of winning. This algorithm can be considered anytime as after that first pass it is able to return a solution.

In his blog post, Bravender explores Information Set MCST in the Euchre domain. This slight variation uses the play history to randomly distribute what cards the other players might have and runs the MCTS using those [1].

4 Design

This project focuses on the calling phase of Euchre which, as mentioned, is an area that has seen less research. Yet, the calling phase is as crucial a part of the game as the playing phase. Deciding whether to call trump or not is a key decision that influences the whole round. We believe that even applying a rule-based approach to this part of the game would create a new baseline from which techniques

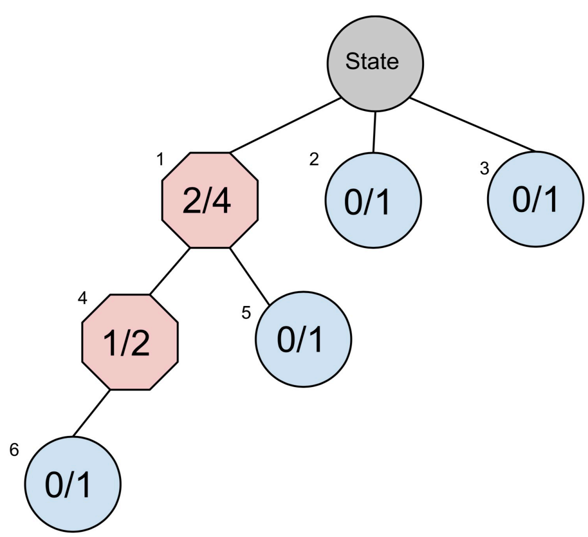

Figure 3. A Monte Carlo Search Tree after 6 simulations. The red nodes are those which, when explored, led to a win. Blue node simulations were losses. Note that the wins and losses of the child nodes affect the parent node's win probability.

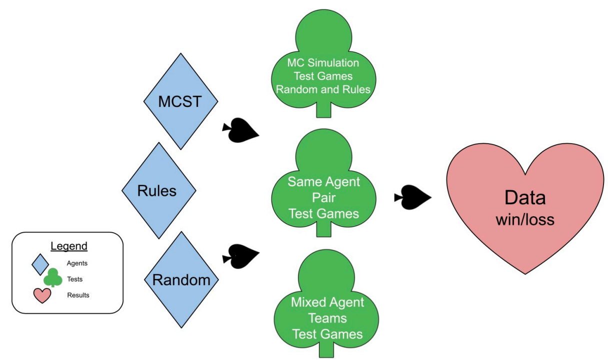

Figure 4. Data Architecture Diagram

like MCTS and RL can further be compared. Beyond the decision to call or not, knowledge from the calling phase can be used to gain insight on what the other players hands possibly looks like which can effect the probabilities in the playing phase. For example, if someone passes on picking up the highest heart, it is then improbable that they have high hearts in their hand; Otherwise, they would not have passed.

4.1 Environment

We started by utilizing the code Pugh [5] implements, which, as mentioned, is an extension of the RLCard toolkit [8]. This code environment was developed for concepts of Reinforcement Learning to be applied to various card game domains. Pugh took the step of extending it to include a Euchre environment and a Euchre rule-based agent. Using this as a base, we modified the environment to allow for a MCST agent as well as updating the hand scoring to more accurately mirror typical scoring rules.

4.2 The Agents

We worked with three different agents: a random agent, a rules agent, and a MCST agent.

The random agent functions first by randomly choosing to call trump or not. While playing, it likewise picks a card between the valid options at random. Our environment never allows for invalid plays.

Slightly more sophisticated, the rules agent attempts to mirror human play by following a set of rules that guide its play. For example, many people play that if they have three or more of a suit in their hand, they call that suit as trump. Another common strategy it uses is to always lead the highest card if possible.

The MCST agent functions as discussed in Section 3.2 of this paper. Each time it runs a simulation play, it creates a new euchre environment instance using the information available to the agent about the current game state. It then randomly deals out the remaining possible cards to the other playing agents. Next, the hand is run to completion as if all agents

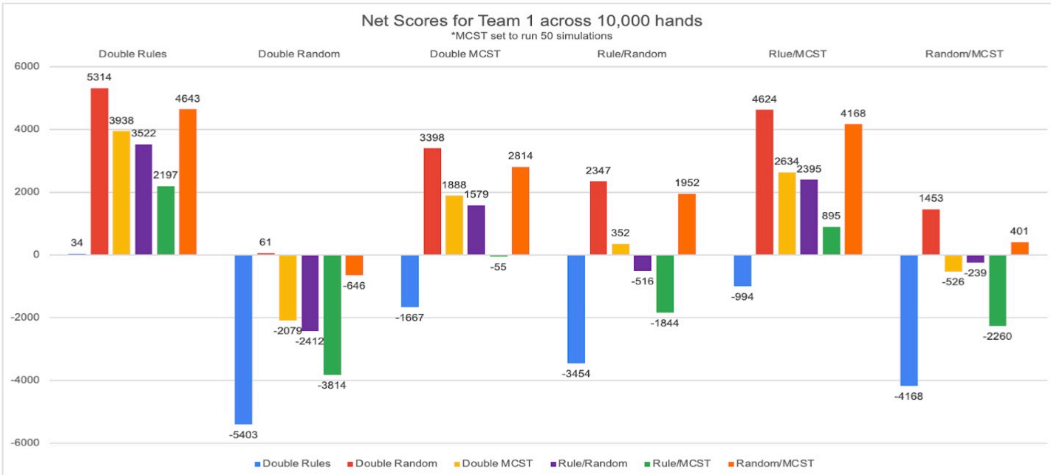

Figure 5. Results of different team pairings playing 10,000 hands. Number reflects the cumulative net score for team one

are playing randomly, and the result is back-propagated up through the tree of possible actions. Since the possible actions the MCST agent will have available to make on their turn is dependent on what actions the other agents take, the steps the other agents take are also simulated using random play.

4.3 Testing

Once all agents were created and running, we played them against each other in a strategic series of games. This process looked much like the testing Seelbinder [6] did with his agents, with the key difference that the games began at the calling phase, not the playing phase. Two key types of test games were tried: those with same agent pairs and those with mixed-agent pairs. For this round of testing, every match-up played 10,000 hands. The Rule and Random agents can perform this series of tests in a matter of minuets, but due to the large number of simulations in the algorithm, the MCST agent tests took many hours to complete. Further wishing to examine the functionality of the MCST agent, we ran 100 hands against a Double Random team with increasing number of simulations from 0 to 200. Ideally, this would result in a monotonic increase of scores as the number of simulations increases. Lastly, we attempted to improve the MCST agent by running the simulations using Rule agents as opposed to Random ones. This version was also tested against Double Random on increasing simulation numbers 0 to 200.

5 Results

The results of the first test are presented in Figure 5. The given number represents the net score for team one which played the strategy as labeled across the top against each other pair. To further contextualize, we note that the theoretical range of these values is -20,000 to 20,000 as the minimum score for each hand is -2 (representing the case of 2 points for team two) and the maximum score is 2. We see that the Double Rule agent team consistently won and the Double Random agent team consistently lost. We had expected the Double MCST agent team to outperform all the other agents, but it's performance was not as good as the Rule agent. Considering the high computational resources it takes to run the MCST agent, we conclude that the Rules agent is a significantly better option. The results of the increasing simulation number tests using both the MCST agent that runs simulations using Random agents and the one that runs using Rule agents are shown in Figure 6. In general, the Rules version does appear to do slightly better than the Random one, though the variance is high for both cases. There is an upward trend as simulations increase, with the agents rarely scoring negatively after about 30 simulations. This information is useful for future testing as the simulation number is a large contributing factor to the computational time, and this test justifies not using a larger simulation number.

6 Future Work

Significant infrastructure for a DQL agent has already been put in place by Pugh and the developers of RLCard, however, we find that the euchre code needs to be updated to match the newer implementations of this agent in the package. A level of testing and fine tuning could then be done comparing a DQL agent to the agents explored in this study.

It would be ideal if as the number of simulations increased on the MCST agent, the resulting score was non-decreasing. We imagine the extreme variance we see represented has a lot to do with the imperfect information aspect of euchre. For our determinization, we simply used a random redealing of cards, but there are some clues from the cards played as to what cards might be in the other players hands that could

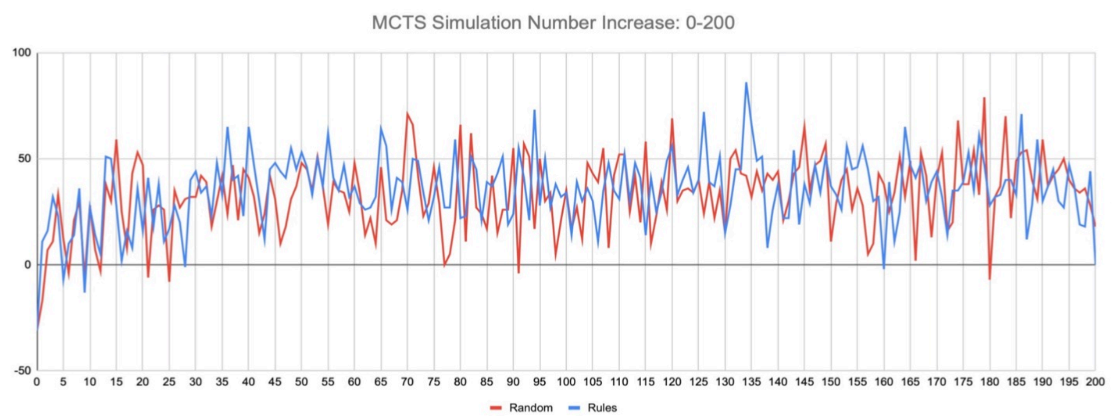

Figure 6. Results of Double MCST vs. Double Random playing 100 hands

be accounted for. Further testing could also be done to tune the exploration weight of the MCST algorithm.

Additionally, our environment does not implement a number of special case rules that are standard while playing Euchre. Most notably, the model could be expanded to include functionality for “going alone”, a special game case where only three of the four players play and up to four points are at stake.

Lastly, we would love to see this work expanded into a playable version where a human player can interactively play with the computer Euchre agents.

Acknowledgments

I would like to thank David Barbella and Micah Nord for providing detailed feedback throughout the process of creating the project proposal. And thanks to Charlie Peck for his guidance on the project and paper.

References

[1] Dan Bravender. 2017. Over 1 billion tricks played - Information Set Monte Carlo Tree Search Euchre simulation database. https://dan.bravender.net/2017/6/9/Over_1_billion_tricks_played_-_Information_Set_Monte_Carlo_Tree_Search_Euchre_simulation_database.html. Accessed: Oct. 9, 2023.

[2] Peter I. Cowling, Edward J. Powley, and Daniel Whitehouse. 2012. Information set monte carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games* 4, 2 (2012), 120–143.

[3] Johannes Heinrich and David Silver. 2016. Deep reinforcement learning from self-play in imperfect-information games. *arXiv preprint arXiv:1603.01121* (2016).

[4] Michael J. Holmes. 2001. *Machine learning in euchre: a comparison of techniques*. Ph. D. Dissertation. University of Northern Iowa.

[5] Eli Pugh. 2020. Learning to Play Euchre With Model-Free Reinforcement Learning. (2020).

[6] Benjamin E. Seelbinder. 2012. *Cooperative Artificial Intelligence in the Euchre Card Game*. University of Nevada, Reno.

[7] Stelios Triantafyllou and Goran Radanovic. 2023. Towards computationally efficient responsibility attribution in decentralized partially observable MDPs. *arXiv preprint arXiv:2302.12676* (2023).

[8] Daochen Zha, Kwei-Herng Lai, Yuanpu Cao, Songyi Huang, Ruzhe Wei, Junyu Guo, and Xia Hu. 2019. Rlcard: A toolkit for reinforcement learning in card games. *arXiv preprint arXiv:1910.04376* (2019).