Monte Carlo Counterfactual regret minimization applied to Clue

by
Benjamin Martin

A THESIS

submitted to

Oregon State University

Honors College

in partial fulfillment of
the requirements for the
degree of

Honors Baccalaureate of Science in Computer Science
(Honors Scholar)

Presented June 12, 2020
Commencement June 2021

AN ABSTRACT OF THE THESIS OF

Benjamin Martin for the degree of Honors Baccalaureate of Science in Computer Science presented on June 12, 2020. Title: Monte Carlo counterfactual regret minimization applied to Clue.

Abstract approved:_____

Prasad Tadepalli

This document analyzes the application of Monte Carlo Counterfactual Regret Minimization (MCCFR) in the game of Hasboro's Clue. As a partially observable stochastic multiplayer game, Clue is well-suited for MCCFR methods. MCCFR has previously been shown to be effective in beating top human players around the world in No-Limit Texas Hold'em. We have found that an MCCFR agent proves to be superior in win rate over a heuristic model counting alternative and two baseline random agents using choice sampling and regret matching.

Key Words: clue, cluedo, MCCFR, monte carlo, partially observable games

Corresponding e-mail address: ben.pat.martin98@gmail.com

Monte Carlo Counterfactual regret minimization applied to Clue

by
Benjamin Martin

A THESIS

submitted to

Oregon State University

Honors College

in partial fulfillment of
the requirements for the
degree of

Honors Baccalaureate of Science in Computer Science
(Honors Scholar)

Presented June 12, 2020
Commencement June 2021

Honors Baccalaureate of Science in Computer Science project of Benjamin Martin presented on June 12, 2020.

APPROVED:

_____

Prasad Tadepalli Mentor, representing Department of Computer Science

_____

Eric Walkingshaw Committee Member, representing Department of Computer Science

_____

Alan Fern, Committee Member, representing Department of Computer Science

_____

Toni Doolen, Dean, Oregon State University Honors College

I understand that my project will become part of the permanent collection of Oregon State University, Honors College. My signature below authorizes release of my project to any reader upon request.

_____

Benjamin Martin, Author

Monte Carlo Counterfactual Regret Minimization applied to Clue

Benjamin Martin
Electrical Engineering and Computer Science
Oregon State University
Corvallis, OR, USA
martinb3@oregonstate.edu

Abstract

This document analyzes the application of Monte Carlo Counterfactual Regret Minimization (MCCFR) in the game of Hasboro's Clue. As a partially observable stochastic multiplayer game, Clue is well-suited for MCCFR methods. MCCFR has previously been shown to be effective in beating top human players around the world in No-Limit Texas Hold'em. We have found that an MCCFR agent proves to be superior in win rate over a heuristic model counting alternative and two baseline random agents using choice sampling and regret matching.

Index Terms

clue, cluedo, MCCFR, monte carlo, partially observable games

Fig. 1. The 2002 edition of the game of Clue, featuring six suspect tokens, six weapons, and 21 cards corresponding to six suspects, six weapons, and nine rooms. The rooms are depicted on the game board, which is not pictured here.

I. Introduction

Hasboro's game *Cluedo* or *Clue* in North America is a murder-mystery board game. Each player takes on the role of a detective, attempting to piece together the clues of the death of the owner of a large estate, who was found to be murdered. The 2002 Clue edition consists of 21 cards: 6 suspect cards, 6 weapon cards, and 9 room cards. Players attempt to gain knowledge of the identity of three cards that are kept in a secret 'solution file' which contain the identity of the murderer, the weapon, and the room where the murder occurred. Clue, being a multiplayer turn-based stochastic partially observable game, is well-suited for MCCFR methods. MCCFR is an iterative algorithm which traverses the game tree by sampling actions at nodes along the game tree instead of exploring the entire tree. MCCFR employs the same regret updating and regret matching techniques as counterfactual regret minimization (CFR). We have used Choice Sampling (CS) MCCFR coupled with abstracting the game state space to reduce memory overloads.

Compared to several varieties of heuristic agents, we have found that MCCFR has proven to be superior. In 3-player games where 2 of the agents are based on a heuristics algorithm, and the third agent is the MCCFR agent, we found that the MCCFR agent wins slightly more than $1/3$ of the time when playing against agents using the most robust heuristics.

Clue, being a partially observable stochastic game, is a perfect target for the application of MCCFR. MCCFR has been used with a high degree of success in other imperfect-information games including Liar's Dice, [6] No-Limits Texas Hold'Em, [5] and Kuhn Poker. We have extended the work from No-Limits Texas Hold'Em [5] solving by adopting their blueprinting strategies (the MCCFR module of their Libratus agent) to advance play. Clue is also a straightforward game to simulate and observe as opposed to more complex partially observable games, such as Hearts.

II. Background

A. Domain

Before the game begins, one card of each type (room, suspect, and weapon) is randomly selected and set aside and placed in the 'solution' file. The remaining 18 cards are then shuffled together distributed as evenly as possible between the remaining players. Players keep their own cards hidden to themselves. Players take turns making "suggestions" which are then refuted by other players with cards from their hands. Once a player makes a suggestion which consists of a room card, a weapon card, and a suspect card, each player clockwise from the player who made the suggestion must attempt to refute the suggestion or pass. If they can refute, they must privately show one of the cards from their hand that match one of the suggested cards to the suggester. Once a suggestion is refuted or all players pass, play continues to the next player, who then may make a suggestion. Each player may make one accusation per game. If they accuse and correctly identify the three hidden cards, then they win the game. Otherwise, they are eliminated from the game and must continue refuting suggestions, but may no longer make an accusation. For the full rules of the game, refer to the 2002 edition rulebook. [3] Traditionally, players are limited to making suggestions only in the room which their own character pawns visit. For ease of application, we have abridged the rules such that there are no movement actions or game board. Instead, players may choose to make a suggestion for any room on their turn, uninhibited by movement. Additionally, players cannot make an accusation. Instead, players win the game automatically once they suggest the correct set of 'solution' cards i.e., a player wins a game of Clue when they make a suggestion that all players pass on and they do not hold any of the suggested cards in their own hand.

Initially, the total number of valid models is

$$6 * 6 * 9 * \frac{18!}{\prod_{i=1}^{P} H_i!}$$

Where P is the number of players in the game, and H_i is the number of cards in player i's hand. For a 6-player game where each player is dealt 3 cards, the number of valid models is over 44 trillion. This formula is arrived at by multiplying the number of possible solution files (6 suspect cards, 6 weapon cards, and 9 room cards, 1 of each which is independently selected at random) which yields $6*6*9 = 324$. The remaining 18 cards can then be arranged in any permutation, but must be divided by the number of ways in which the same cards can be rearranged in each player's own hand. Note that the following condition must be true:

$$\sum_{i=1}^{P} H_i = 18$$

For different versions of clue (where there may be more or less room cards, weapon cards, or suspect cards), these conditions do not hold.

In order to avoid leaking knowledge or information to other players in the game of Clue, the agent must play in a somewhat random fashion to avoid disclosing its own cards or the cards that it knows about. To expand upon this point, we have shown that greedy heuristic agents that suggest the most likely combination of cards in a game of Clue can be exploited by a heuristic exploitation agent. As an example to show how a deterministic policy can be punished, consider a naive poker agent. This agent will bet proportionally to the strength of its hand; it will bet high when it has a good hand, and it will bet low when it has a bad hand, and it will fold when the bet size grows larger than the expected value of the hand. This naive poker agent is missing a very human component: the *bluffing* element. It is playing in a greedy way which would

B. Existing Clue research

Clue has previously been analyzed using Monte Carlo policy rollout algorithm which used a baseline heuristic policy to begin evaluating the likelihood of a set of cards appearing in the solution file, and tweaking the hyperparameters of both the horizon and the number of simulations to run. The authors found that the policy rollout improved each baseline agents' performance when pitted against the unimproved baseline agent. Games were modeled using a Markov Decision Process (MDP) with modified transition functions to approximate the partially observable domain. [4]

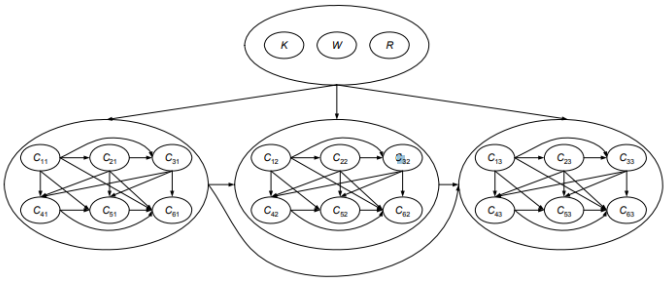

Fig. 2. Clue bayesian network as described in [2]

Clue has also been modeled and tested using a Bayesian Network (BN). The authors modified the rules of the game by fixing the player's hands with a certain quantity of each typology of card by their category of room, suspect, or weapon. The authors tested their model by providing an interactive, graphical interface which allows human players to play against the computer. Testing showed that the model had an advantage over human players with a win disparity. [2]

C. Monte Carlo counterfactual regret minimization

1) History: Valid sequences of actions in the game are called histories, denoted $h \in H$. A history is a terminal history, $h \in Z$ where $Z \subset H$, if its sequences of actions lead from root to leaf. A prefix can be obtained by taking a valid sequence of actions from h. Given h, the current player to act is denoted $P(h)$. Each information set contains one or more valid histories. The standard assumption is perfect recall: information sets are defined by the information that was revealed to each player over the course of a history, assuming infallible memory. In a game's play, there are two different types of nodes: a chance node, which denotes a random event not caused by any particular player. There is also a choice node, which is a state at which the player must take an action. Choice nodes are further logically divided up into opponent's choice nodes, and the player's choice nodes. In Clue, we express the utility at the terminal node of the history by assigning a value of -1 to any player who loses the game, and $+P$ to the winner, where P is the number of players in the game. A player then, who takes an action that causes them to lose a game in an otherwise winning position, would regret taking the winning action by $1 + P$.

2) Counterfactual regret minimization: In 2000 counterfactual regret minimization (CFR) was introduced as a means to track player regrets for their previous actions and play with a weighted bias towards positive regrets.

In counterfactual regret minimization, a strategy is a mapping of infoset I to action probabilities. All strategies for all players at time t is called a *strategy profile*, denoted by σ^t. The strategy for player i at *information set* (infoset) I then is denoted by $\sigma_i^t(I)$. An information set is the set of all information available to player i in the game at that point. Events which occur independently outside of player control are *chance nodes*. Chance nodes can be treated the same as a player – chance nodes have a constant strategy which is not updated over the course of the counterfactual regret minimization algorithm. For instance, in a 2-player game, the "chance" player would be denoted as player $i = 3$. The *counterfactual value* of a history h is:

$$v^{\sigma}(h) = \sum_{z \in Z_h} \pi^{\sigma}_{-i}(h)\pi^{\sigma}(h, z)u_i(z)$$

where Z_h is the set of all terminal histories that can be reached from h, and $\pi^{\sigma}(h, z)$ is the reach probability of reaching z from h assuming that all players follow the strategy profile σ.

The *instantaneous counterfactual regret* is as follows:

$$r_i(h, a) = v_i(\sigma_{I \to a}, h) - v_i(\sigma, h)$$

where $\sigma_{I \to a}$ is the exact same strategy profile as σ except that player i will always follow the pure strategy of taking action a when at infoset I.

The *cumulative counterfactual regret* after T time steps is as follows:

$$R_i^T(I, a) = \sum_{t=1}^{T} r_i^t(I, a)$$

where

$$r_i^t(I, a) = \sum_{h \in I} r(h, a)$$

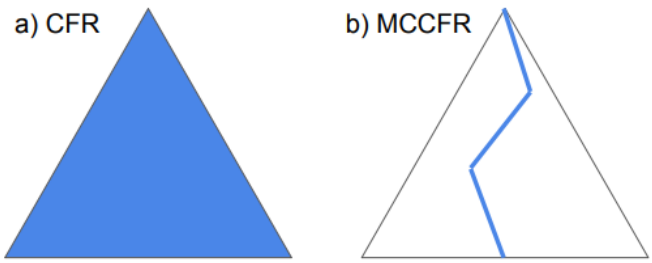

Fig. 3. Typical variants of CFR versus MCCFR exploring the game tree. While (a) CFR explores the entire tree, (b) MCCFR will sample opponent actions to reduce the space needed to evaluate an action by the player.

The strategy profile is then updated at every timestep. The following algorithm is also referred to as *regret matching*, where the probability of selecting action a is proportional to the positive cumulative regret of taking that action:

$$\sigma_i^{T+1}(I,a) = \begin{cases} \frac{R_i^{T,+}(I,a)}{\sum_{a \in A(I)} R_i^{T,+}(I,a)} & \text{if} \sum_{a \in A(I)} R_i^{T,+}(I,a) > 0 \\ \frac{1}{|A(I)|} & \text{otherwise} \end{cases}$$

where

$$R_i^{T,+}(I,a) = max(R_i^T(I,a), 0)$$

MCCFR in Texas Hold'Em uses an iterative strategy training over multiple games. In each iteration, it simulates a game as a player, called the traverser. The traverser would pick actions according to its strategy profile, and sample opponent actions in the game. After the game completes, the traverser then examines other paths where it could have chosen different actions and evaluates the value of picking other actions in the game. The traverser measures how much better it otherwise could have played by assigning a regret value to each other action. The action chosen by the traverser always has a regret of 0. In the future, actions with higher regret have a higher chance of being chosen, because the strategy profile is updated at the end of each game and is proportional to the regret of each action.

III. Adaptation of MCCFR to Clue

CFR guarantees that in a two-player zero-sum game, two players employing the CFR to update their respective strategies will reach a Nash Equilibrium with a large enough number of games. While Clue is not a two-player game, the game of Clue is not easily prone to "kingmaker" or "gang-up" tactics, which lends itself to still be applicable to CFR, regardless of the number of players.

In Clue, the public history of the game is represented with a 5 x h matrix, where h is the total number of suggestions made in a game. The first column denotes the player who made the suggestion, columns 2, 3, and 4 denote the identity of the cards for the suspect, weapon, and room respectively, and the final column denotes the player who refuted. If no player was able to refute, then the final column is left empty as NULL. This history is public knowledge, which means that all players have access to this information with perfect recall.

The infoset for each player in the game can be accurately created using two tables. First, the public history of the game as described above, and secondly, the *private knowledge* of the player at that point of the game. The knowledge of the player can be expressed as a 21 x P matrix, where each cell in the matrix denotes whether player p has card c. If the player knows that some player p doesn't have card c (either because the player passed, or through logical deduction), then the value of the cell is -1. If player p has card c, then the value of the cell is 1. Otherwise, the value of the cell is 0. This denotes that the player's knowledge on whether or not player p has card c is not yet conclusive. We can infer that any card which cannot be in any player's hand must be in the solution file.

Q@	Suspect	Weapon	Room	A@
Player 1	Plum	Knife	Study	Player 3
Player 2	White	Candlestick	Kitchen	Player 5
Player 3	Green	Revolver	Hall	NULL
...	...	...	...	...

Fig. 4. An example game in progress in the game of Clue. Player 1 began by suggesting Professor Plum with the Knife in the Study. Player 2 did not have any cards that matched the suggestion, and passed. Player 3 then showed a card to Player 1. Player 2 then made a suggestion of Mrs. White with the Candlestick in the Kitchen. Players 3 and 4 passed, and player 5 refuted, showing a card to Player 2. Player 3 then made a suggestion of Mr. Green with the revolver in the Hall which none of the other players could refute. Player 3 won the game if none of the 3 cards that she suggested were in their own hand (because they could automatically deduce that the 3 cards that they just suggested were in the solution file). This information is expressed in a compact format, but for simplicity's sake has been shown more explicitly here.

A. Implementation

MCCFR uses sampling to evaluate the counterfactual regrets of an action for a particular choice node in a player. The game tree is sampled using Chance Sampling (CS) – one of the simplest variants of MCCFR. Each choice node where a player may take an action is sampled following the weights of their current strategy profile and evaluated. Opponent's choice nodes and chance nodes are merely *queried*, that is, only a single sample is generated which creates the response that would have happened if the player had taken the action that they did. This technique can create high variance due to the high branching factor and low sampling of opponent's choices and random events There is continued discussion on how to reduce variance in the MCCFR algorithm to speed its training up. [7]

B. Game Abstraction

Due to the branching factor and memory cost associated with a long game duration like in Clue, the game's information states must be abstracted, or compactly expressed so as to avoid expensive memory and computational costs. Here we logically divide cards by their typology [2], and bucket information states where cards of the same typology are in identical positions relative to player order. Cards in Clue are reduced to three different typologies: room cards, suspect cards, and weapon cards. For example, a player holding 3 cards: Plum, Knife, Ballroom is functionally similar to holding Plum, Knife, Study, except for the fact that the ballroom card has been swapped with the study card. Since these situations can be calculated in exactly the same way albeit with different locations for the cards, then they can be bucketed in the state space of the strategy profile without loss of information.

IV. Experimental results

We developed 3 agents similar to what is used in [4]. The first agent is a random stateless agent. It will merely suggest any three cards every turn as long as those three cards are not in its own hand. We then have a second, stateful random agent. This agent will record all cards it has seen directly, and will never suggest a card that it has seen, but will otherwise make random suggestions. Finally, we have a greedy heuristic agent which will approximate the likelihood of each card in a solution file. It will greedily suggest the set of three cards that are most likely to be in the solution file. The heuristic agent works by model counting the number of valid solution models after reaching a specific game state, and checks to see which cards appear in the solution most often in the model. The heuristic agent starts by encoding all information in the infoset into a set of boolean variables, which can then be used to model count using standard SAT# model counting techniques. The heuristic agent relies on the assumption that all solution models are distributed uniformly random. Also since it works in a greedy way, the heuristic agent is vulnerable to revealing information on its own hand state and knowledge, which can be exploited.

Algorithm 1: Heuristic model counting agent

```
Result: Return action a for game state (infostate) s.
for each unique tuple of cards suspect, weapon, room do
    T[suspect, weapon, room] = model_count_valid_solutions(suspect, weapon, room);
    return max(T)
end
function MODEL_COUNT_VALID_SOLUTIONS(suspect, weapon, room)
if s.privateKnowledge.playerHasOne(suspect, weapon, room) then
    num = 0;
end
s.publicHistory.convertHistoryToSATKnowledge();
num = s.SATMODELCOUNT();
return num;
```

The MCCFR agent was trained and then tested against each agent in a 3-player setting, with a single MCCFR agent and two of the same benchmarking agent. In order to mitigate the bias of player order, the MCCFR agent would play first in a third of the games, second in another third of the games, and last in the final third of the games. The agent played in total 30,000 games against the two opposing agents. It played 10,000 games in 3 different positions each. The 95% confidence interval after 30,000 games played is ± 156.8 games or $\pm .523\%$ in win rate. The MCCFR was trained at several intervals, at which point the training was paused and the strength of the MCCFR agent was measured against the baseline random agents and the heuristic agent. After training, a copy of the training result at that point in time was saved and training resumed. The results measure the win rate of the MCCFR agent against the two opposing agents with respect to the number of games trained that the MCCFR was allotted. One traversal of the game tree in the CFR algorithm here is considered 1 game of training.

V. Future work

Although there has been groundwork laid in showing the promise of MCCFR solving in Clue games, it may perhaps be beneficial to develop more sophisticated agents to benchmark the MCCFR agents. It would also be beneficial to translate the

code from the propriety Matlab language into a more easily distributed and accessible language, such as Python or Java. There are also several optimization techniques which can be used to improve upon the MCCFR algorithm to improve its learning speed, using variance reduction [7] . There is a possibility of implementing linear CFR which will linearly scale up more recent regrets, and Regret-Based Pruning (RBP). Regret-based pruning is a technique where negative cumulative regrets are not zeroed out, but rather, are stored as they are. Any regrets which are very negative will be ignored and their regrets will not be updated for as many iterations as needed until it is possible that the action could be positive again. Then, the game tree can be traversed for that action and the regret can be reevaluated. There is also a consideration of adding ”subgame solving” when approaching near the end of the game. The static blueprinting strategy that this MCFFR agent can be great at approaching the myriad of options and possibilities early on in the game, but may not be as strong when it approaches game states that only occur a handful of times. Leveraging the extra knowledge it has at the end of the game, the agent would switch to using a granulated solving method by analyzing every possible action by the opponent. This approach has been used successfully in No-Limit Texas Hold-’Em. [5]

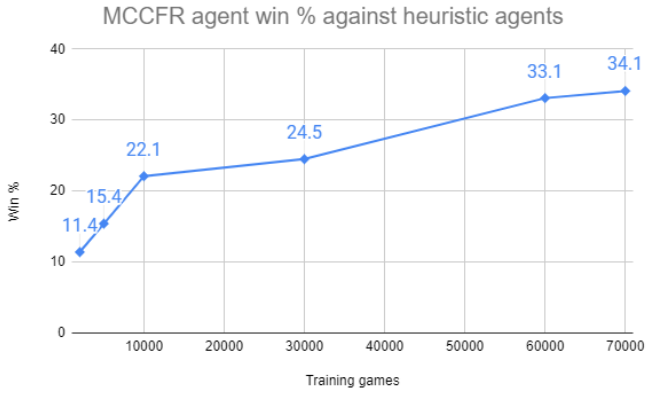

Fig. 5. Results of the MCCFR agent versus the heuristic agent, as a function of number of games won over games trained.

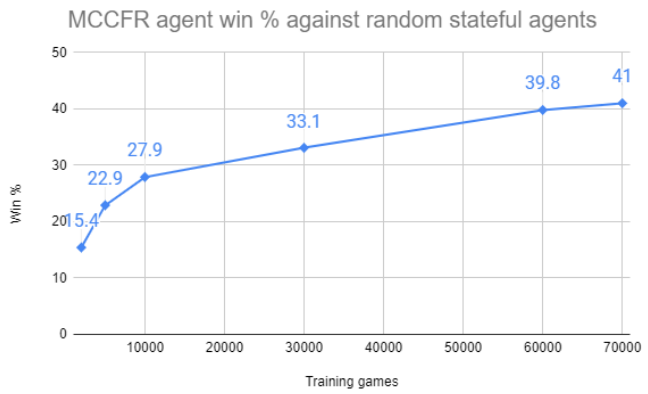

Fig. 6. Results of the MCCFR agent versus the random stateful agent, as a function of number of games won over games trained.

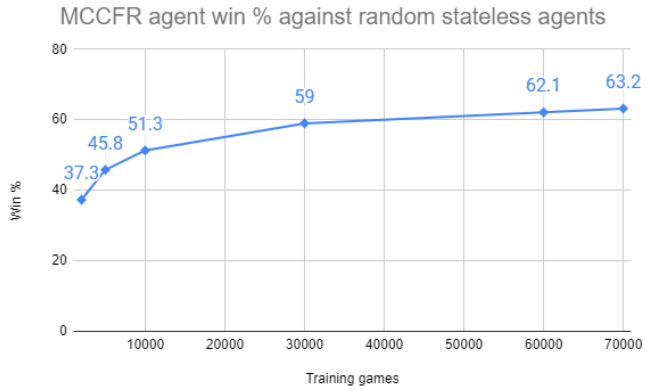

Fig. 7. Results of the MCCFR agent versus the random stateless agent, as a function of number of games won over games trained.

VI. Conclusion

We have found a robust agent that is proven to be superior over other benchmarking agents after sufficient training. The downside of this particular agent is the high training cost and time needed to train, which can be mitigated with enough computing resources and optimization of the algorithm.

References

[1] Alvim M.S., Chatzikokolakis K., Kawamoto Y., Palamidessi C. (2018) Leakage and Protocol Composition in a Game-Theoretic Perspective. In: Bauer L., Küsters R. (eds) Principles of Security and Trust. POST 2018. Lecture Notes in Computer Science, vol 10804. Springer, Cham

[2] C. Cai and S. Ferrari. "On the Development of an Intelligent Computer Player for CLUE: a Case Study on Preposterior Decision Analysis", *Proceedings of the 2006 American Control Conference Minneapolis, Minnesota, USA*, June 14-16, 2006. http://lisc.mae.cornell.edu/LISCpapers/ACC_CLUE_Chenghui06.pdf

[3] "Clue Instruction Book". *Hasboro, Inc*, 2002. https://www.hasbro.com/common/instruct/Clue_(2002).pdf

[4] E. Marshall. "An Empirical Evaluation of Policy Rollout for Clue", Oregon State University, July 2017.

[5] N. Brown and T. Sandholm. "Libratus: The Superhuman AI for No-Limit Poker". *International Joint Conference on Artificial Intelligence (IJCAI-17).* 2017.

[6] M. Ponsen, S. Jong, M. Lanctot. "Computing Approximate Nash Equilibria and Robust Best-Responses Using Sampling" *Journal of Artificial Intelligence Research* 42 (2011) 575–605, Dec. 2011.

[7] M. Schmid, *et. al.* "Variance Reduction in Monte Carlo Counterfactual Regret Minimization (VR-MCCFR) for Extensive Form Games using Baselines," *Association for the Advancement of Artificial Intelligence*, Sep. 2018.

[8] T. Neller and M. Lanctot. "An Introduction to Counterfactual Regret Minimization", Gettysburg College, July 9, 2013.