AlphaHearts Zero: Implementing AlphaZero techniques for Imperfect Information Trick-Taking Games

Andrew Wei

A Senior Project presented in partial fulfillment of the requirements for the Bachelor of Science in Computer Science and Economics

Yale University
May 2022

Advisor: James Glenn

AlphaHearts Zero: Implementing AlphaZero techniques for Imperfect Information Trick-Taking Games

Andrew Wei

May 2022

Abstract

The AlphaZero technique has seen remarkable success in evaluating perfect information games. However, owing to the difficulty of evaluating information sets rather than states, little work has been done in applying this technique to imperfect information games. In this work, I apply the AlphaZero technique to Hearts, an imperfect information trick-taking game. I create an agent ("AlphaHearts Zero") capable of playing Hearts at a human level, outclassing baseline agents and approaching the performance of cheating agents. In particular, the agent is capable of being trained *tabula rasa* without any prior knowledge or preloaded heuristic.

AlphaHearts Zero utilizes the Perfect Information Monte Carlo (PIMC) technique to convert in-game information sets into perfect information states, and then applies a version of the Monte Carlo Tree Search (MCTS) algorithm enhanced with a Deep Q-Network (DQN). The DQN neural network is trained by self-play through deep Reinforcement Learning (RL), as per other AlphaZero implementations. Self-play RL is able to steadily increase the performance of AlphaHearts Zero over many iterations.

Owing to the fact that the AlphaHearts Zero is trained without any prior knowledge, the methods demonstrated in this paper in Hearts show the viability of both PIMC and AlphaZero in a wide range of imperfect information games. Possibilities for future exploration include a specialized inference system to more accurately convert information sets into perfect information states, a more efficient tree search algorithm, further refinement of the neural network and an improved self-play procedure.

This senior project consists of this report, as well as attached code files.

Background

Motivation

Many advances have been made in the field of computational games in recent years. Starting with *Deep Blue versus Garry Kasparov* in Chess in 1997 to *AlphaGo versus Lee Sedol* in Go in 2016, computers have reached and exceeded human play levels in most perfect information games. With the development of AlphaZero 2017 and MuZero in 2019, superhuman play for many perfect information games are within reach. The combination of Monte Carlo Tree Search with a Neural Network with has been highly successful in a variety of perfect information games.

Despite advances in the field of perfect information games, superhuman play remains out of reach for most imperfect games, especially trick-taking games. Although they have reached expert-level play in games such as Bridge, they remain at a level below their top human competitors in most competitions. In this paper, I will apply the current state-of-the-art system (AlphaZero) in perfect information games to Hearts, an imperfect information trick-taking game.

Hearts

Hearts is a deterministic (except for the initial distribution of cards) imperfect information trick-taking game typically played by four players and a standard deck of 52 cards. Each player is dealt 13 cards, face-down.

Every round, the first player will play a single card face-up on the table (called "leading"). The other players must play a single card of the same suit in response, unless they do not have a card of the same suit, in which case they are "void" of the suit and may play whatever card they wish. The highest card of the leading suit wins the trick and leads the next round. This continues until all cards are played (in 13 tricks).

The objective of the game of hearts is to take as few points as possible. A player takes points if they win tricks where Hearts and the Queen of Spades have been played. Each heart won in a trick is worth 1 point, while the Queen of Spades is worth 13 points. On the other hand, if a player takes all 26 points, otherwise called "shooting the moon", they get 0 points and all other players get 26 points. Multiple games are typically played until a player reaches a certain points threshold.

Some other variants of the game play with the rule that no player may lead with hearts until hearts have been played on a prior trick. Other variants include a rule where players exchange three cards with their opponents before gameplay. These variants are not within the scope of this project.

Past Literature in Hearts

There have been a small number of papers in the game of hearts. In his 1998 paper Perkins[8] detailed two different algorithms, II-Max n and MC-Max n, which can be used for search in general imperfect information games, and demonstrated their application to the game of hearts. These algorithms were extensions of $\max^n$ search[7] for imperfect information games, which was in itself an extension of minimax search.

In their 2005 paper, Fujita et al.[4] used reinforcement learning in a neural network to play the game of hearts. Although the paper did not use tree search, it did use a neural network that evaluated the value of certain perfect information states, and used MCMC sampling of these values to determine the value of an information set.

In the realm of TD-learning, Furnkranz et al.[9] did some initial work, by using a neural network to learn a mapping from state characteristics to general strategy, which is then mapped onto a selection of moves, allowing for gameplay to be learned from a small selection of features. This was expanded upon by Sturtevant et al.[13], who developed an extensive set of features to conduct TD-learning.

Alternatives to Tree Search for Imperfect Information Games

In recent years, Monte Carlo Tree Search (MCTS) has been the preferred method of choice for solving a variety of games. Although classical MCTS is used in perfect information games, various adaptations of it exist to solve imperfect information games.

In a 2012 paper, Cowling et al.[3] propose several techniques, collectively known as Information Set Monte Carlo Tree Search (ISMCTS). In particular, one technique, MO-ISMCTS, is noted by the authors as the most "theoretical defensible", but proof of convergence to the Nash equilibrium has yet to be achieved. MO-ISMCSTS will be out of scope for my senior project, but will form the basis for my CPSC 674 final project.

Furtak et al.[5] explore the idea of Perfect Information Monte Carlo (PIMC) and Imperfect Information Monte Carlo (IIMC). Together, they propose Recursive PIMC technique (RecPIMC) — a recursive IIMC search variant based on perfect information evaluation. This technique was shown to perform extremely well in Skat (another trick-taking game). Other results by Pipan[10] demonstrate the superiority of PIMC in other imperfect information games as well.

Recent Work in Similar Trick-Taking Games

Recent advances in other trick-taking games mean that there is increasing optimism that these techniques can be used in Hearts as well.

Through the use of PIMC, Buro et al.[2] developed several novel techniques in improving state evaluation in Skat. In particular, the use of PIMC allowed for the development of a sense of inference, which was critical to playing the game. Therefore, this agent was able to reach expert-level strength in several online tournaments.

In addition, Shi et al.[11] was able to master the game of GongZhu at an expert-level using an AlphaZero approach (writing a program called ScrofaZero). ScrofaZero was unique in the sense that it utilized stratified sampling of card distributions in order to boost results.

Methodology

AlphaZero for Perfect Information Games

The AlphaZero method utilizes Monte Carlo Tree Search at its core, with a neural network to help guide random playouts. Each neural network is trained on past gameplay, and evaluated against prior iterations of the network - every iteration, the network is kept if it performs better against prior iterations, and replaced elsewise. This process continues indefinitely, resulting in superhuman play in games such as Chess and Go.[12]

The game of Hearts differs the standard implementation of AlphaZero in two key areas. Firstly, Hearts is a four-player game, while most attempts to games addressed are two-player. Secondly, AlphaZero is concerned with perfect information games, while Hearts is imperfect information. To contend with the first issue, I alter the Reinforcement Learning procedure to allow for multi-agent self play. To contend with the latter issue, I implement the Perfect Information Monte Carlo method, which reduces information sets to perfect information states.

Perfect Information Monte Carlo

The Perfect Information Monte Carlo (PIMC) technique translates imperfect information games to their perfect information equivalents, allowing for the use of perfect information algorithms such as minimax, MCTS, and in this instance AlphaZero. Upon reaching an information set, a Perfect Information Monte Carlo agent will sample a certain number of possible states within said set, and determine the perfect information value of each move. The move that performs best is then selected.

The success of a PIMC agent depends on the strength of its sampling module and perfect information value module. In human play, individuals are capable of predicting player cards on the basis of what has been played in the past, as well as what would be logical for them to play. They are thus able to sample from the information set in a more effective manner. However, computers have a limited ability for inference - in my project, I focus primarily on the development of a more successful perfect information move value module. Using simple uniform random sampling for Hearts is acceptable, primarily because of its high disambiguation factor - the number of nodes in a player's information set shrinks with regard to the depth of the tree.[6]

Monte Carlo Tree Search

I used a standard MCTS with a the upper confidence bound (UCB) bandit algorithm[1] to evaluate positions in the game. Specifically, for each already-fully-explored node, we choose to visit the node

$$\text{argmax}_i \quad \frac{w_i}{n_i} + K\sqrt{\frac{\ln N}{n_i}}$$

where w_i is the total proportion of points avoided at the node i (the "reward" of the game), n_i is how many times node i has been visited, N is the total visits to the node this round, and K is the exploration constant.

The total reward at node i, designated as w_i, is calculated by:

$$w_i = \sum_{k=1}^{n_i} \frac{26 - \text{Number of points won in Game } k}{26}$$

The default rollout policy utilizes a neural network to guide rollouts. For every node with unexplored children, we choose to visit each unexplored node with the probability:

$$\text{softmax}(q(s,a) \times \mathbb{I}(\text{legal move}))$$

Typically, it is standard for the legal move mask to be applied after the softmax layer, or for the softmax layer to be implemented as part of the neural network. However, owing to the structure of the neural network, such a setup would preserve information about illegal moves in the probability network.

The alteration to the MCTS rollout policy is the primary source of the performance increase of the neural network-enhanced MCTS agent. Although this change may seem subtle, the alteration to the rollout policy has significant consequences - by guiding rollout play in a more realistic direction, the neural network-enhanced MCTS agent is better able to simulate gameplay and search the game tree in a more intelligent and efficient manner.

Other implementations of AlphaZero techniques have used neural networks to alter the implementation of the UCB algorithm. This may further improve the search algorithm and lead to better gameplay - certainly an area for future study.

Multi-agent Reinforcement Learning

The neural networks used in the agents are trained through a process of self-play reinforcement learning. Every epoch, four AIs will play 64 games against each other, and then train in a single batch. Data from batches are preserved for 3 rounds. The network is trained on a Adam optimizer with lr = 0.001 and β = (0.3; 0.999). A Huber loss function is used due to the high variance of Hearts game play and decreased sensitivity to outliers. Networks train against a target network for stability purposes.

In order to bootstrap training, each neural network is initially bootstrapped against a Cheating MCTS network in order to hasten training speed. The initial bootstrapping did not affect final results, but did save many hours in training time.

After every training epoch, networks are evaluated against each other. Each network plays in the evaluation phase according to the following rules (sorted in order of training time):

- **Perfect Information:** This player will play the maximum legal Q value move of the Perfect Information state.
- **PIMC:** This player will play the average maximum legal Q value move of 10 determinizations.
- **Cheating MCTS:** This player will play the best move as determined by a MCTS search tree with 20 iterations of the perfect information state.
- **PIMC MCTS:** This player will play the average maximum best move as determined by a MCTS search tree with 20 iterations per determinized state and 10 different determinized states.

The search depth in each training step is constrained by the amount of time it takes for each iteration to run. However, the experimental results demonstrate that the networks trained by Perfect Information do not perform significantly worse than networks trained by more complex methods. This signifies that training using a deeper MCTS search tree may not lead to a substantial increase in performance, in comparison to the amount of additional training time added.

Summary of Terminology

To summarize, this project will generate two players for evaluation in the next section:

- **Cheating AlphaHearts Zero:** This player will play Monte Carlo Tree Search with playouts enhanced with the results from the neural network.
- **PIMC AlphaHearts Zero:** This player will generate 10 determinizations per information set, then play as the Cheating AlphaHearts Zero for each determinization. The move with the highest value will then be played.

As discussed in the "Experimental Details" section, the Cheating AlphaHearts Zero player utilizes 200 MCTS iterations and the PIMC AlphaHearts Zero player utilizes 200 MCTS iterations and 10 determinizations. The PIMC AlphaHearts Zero runs in approximately 1 second per turn, which is acceptable for play against Humans. With faster MCTS and Hearts code, this could definitely lead towards speed enhancements and more feasible iterations.

Empirical Results

Evaluation System

All agents are evaluated against each other in an arena-based head-to-head format. Each player was copied, and each copy was placed in either the North/South positions or the East/West positions, where they would play 200 games against each other (This involves 100 newly generated starting hands, and each starting hand would be played by each pairing twice). Total points collected by each pairing was recorded. However, the game was not cooperative and all agents assumed they were on different teams.

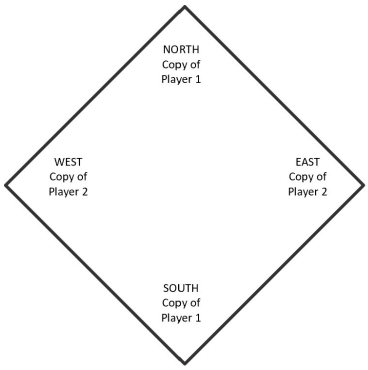

Figure 1: Evaluation System. Total point differential is calculated by (North + South points) - (East + West points)

In the self-play section of the code, the gameplay from the evaluation is recorded and used to train the next iteration of Neural Networks. This is done in order to use computational power more efficiently.

Heuristic and Testing AIs

To evaluate our agents, we build a series of Heuristics-based AIs:

- **Random Player:** This player will play any legal move with an uniform random probability.
- **Simple Heuristic Player:** This player will play a simple greedy strategy.

- **Complex Heuristic Player, without Cheating:** This player will play a more complex rules-based strategy, taking into account what is on the board. This player will play at a human beginner-level strategy,

- **Complex Heuristic Player, with Cheating:** This player will play a more complex rules-based strategy, but with the added ability to examine hidden cards of the player. This player is strong against humans, but it can be exploited over time.

In addition, our agents are evaluated against two baseline tree-search based-agents. These baseline tree-search agents are evaluated using the same amount of iterations and same search depth as the Q-network enhanced agents.

- **PIMC Player (Determinized MCTS):** This player will implement the Perfect Information Monte Carlo algorithm, with a Monte Carlo Tree Search being expanded for each determinization. The algorithm will search the game tree 100 times, over 10 determinizations. This agent plays at a human level. The PIMC Player is analogous to the PIMC AlphaHearts Zero player with no neural network to enhance playouts.

- **Cheating MCTS:** This player will play the game using Monte Carlo Tree Search, using hidden information. The algorithm will search the game tree 100 times. This player is strong against humans. The Cheating MCTS Player is analogous to the Cheating AlphaHearts Zero player with no neural network to enhance playouts.

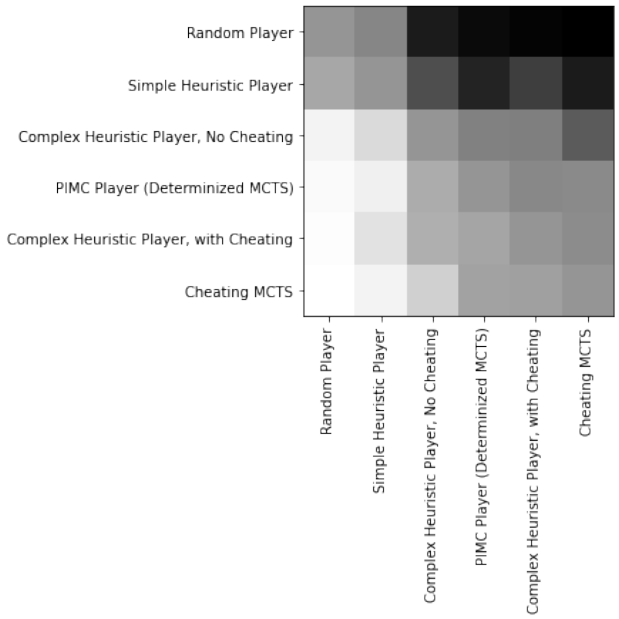

Figure 2: Heatmap of results of gameplay over 100 games. A black square would indicate a high win rate of the column player against the row player, and the opposite is true for white squares.

Each of the agents are evaluated against other agents, with the results being displayed in Figure 2.

Performance vs Heuristic and Testing AIs

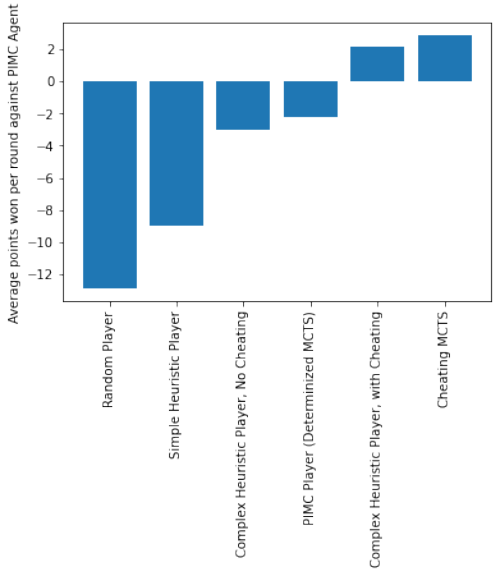

Figure 3: Results of Baseline Neural Network (fc-1) against Heuristic and Testing agents

The use of the AlphaZero Hearts technique yields dividends in player performance. Although it is unable to match the level of cheating agents, it outperforms all other benchmark imperfect information agents. Furthermore, the addition of the neural network makes up for around half of the gap between the baseline PIMC agent and the cheating agents, demonstrating that the incorporation of the neural network leads to a substantial improvement to human levels.

Neural Network Training Method

As discussed in the section on Multi-agent Reinforcement Learning, there were four different training evaluation rules used in the training process to evaluate the strength of neural networks. Each of these four networks are trained for 25 epochs, and evaluated in a PIMC (non-cheating) setting against the Cheating Complex Player for 500 games. In addition, the results for the standard PIMC-MCTS agent without the neural networks are also included.

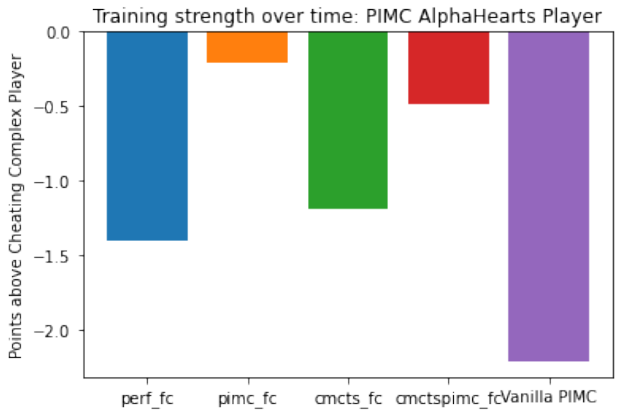

Figure 4: Playing strength of various different PIMC players against Cheating Heuristic Player. Each Neural Network is trained using a different method

The results demonstrate that training methods that take into account imperfect information actions perform significantly better than training methods that assume perfect information. Furthermore, by taking into account imperfect information, agents are able to further narrow the gap with Cheating Players, in comparison with the standard PIMC-MCTS agent.

The use of perfect information in training runs contrary to the results from the paper by Shi et al. in Gongzhu.[11] Although eliminating the hidden information may lead to a more stable network, a neural network trained under the assumption of perfect information does not adapt as well to the reality of PIMC play.

Despite this, training a network on imperfect information takes an order of magnitude more time than training on imperfect information. Further speed enhancements to the code are needed to obtain more practical results.

Performance over time

In order to validate that networks were improving through self-play, networks were sampled every 5 epochs and evaluated in the Cheating AlphaHearts Zero framework against the Cheating Complex Player. 100 symmetric games were played for each evaluation.

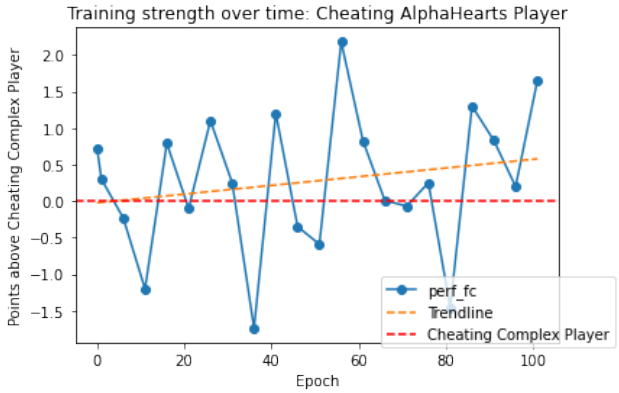

Figure 5: Training Strength over time of MCTS with Neural Network.

The diagram demonstrates that the Self-Play module is able to gradually improve the performance of the AlphaHearts Zero player. However, results are noisy and inconsistent - more training epochs and evaluation iterations are needed for more consistent results.

Discussion

Human Experience

Strengths and Weaknesses

Anecdotally, it is observed that the agent is generally more successful at generating favorable but unexpected endgame scenarios. In particular, it is able to occasionally catch its opponents off guard and hand off the Queen of Spades when they are least expecting it (ie. when leading with a low card of another suit). Furthermore, the agent is also able to execute what at the time seem like poor moves in favor of long-term gains. For example, leading with high hearts early may seem like a poor move at first glance, but actually may set up a favorable later-game scenario which leads to the opponent taking the Queen of Spades. Humans are often not able to execute this type of "intuitive" play.

On the other hand, the agent has a bias against taking leadership in a trick, especially in the early game when certain suits are almost definitely "safe". This is most likely due to the fact that the search tree has

incentives to search almost all branches of the tree, which lead it to negatively bias segments of the tree with a lot of very bad choices but a few good ones, rather than a smaller amount of medium choices. Re-weighting the exploration factor or changing the exploration factor dynamically based upon the Q-values should lead to a lessening of this problem.

Furthermore, the agent appears to have a higher probability of selecting high variance plays, relative to humans. An example of a high variance play would be playing Ace or King of Spades in the penultimate turn of the trick, betting that the player after will not have the Queen of Spades. Although this is not inherently problematic (because the agent is trained to only optimize to minimize score), this can present issues since the actual agent is not acting in perfect information and since the true Q-values are often themselves highly uncertain. A potential solution to this issue would be to penalize the Q-values against high variance plays, as well as skewing the PIMC player away from selecting plays that are high variance.

Occasionally, the agent may execute what may only be described as a blunder. An example of this would be playing the Ace of Hearts when hearts are on the table, significantly more lower cards are available, and it is impossible to shoot the moon - and then following up by playing the lower heart next. It is observed that this tendency can be reduced with more determinizations (and more computational time), which implies that this is probably due to the fact that the agent uses PIMC with uniform random sampling. Through the course of random sampling, determinizations may turn out in a way such that what is in reality a blunder may seem like an incredibly intelligent move. This issue can be more intelligently resolved through an improved sampling system, such as the stratified sampling used in Shi et al. [11].

Finally, the agent has a tendency to under-optimize certain obvious optimizations in the game. For example, it can waste its safer cards in situations where more dangerous cards would suffice (such as the early game), or in another example it can refuse to pass points onto a player even when they have points available. This situation can occur when the agent does not search deeply enough to see the value of holding onto a safer distribution of cards. This is most likely an issue originating in MCTS in imperfect information, as it has been observed in other papers (Cowling, Powley and Whitehouse 2012). Alternatively, the agent could penalize worse positions directly in position evaluation.

Shooting the Moon

Perhaps due to the fact that the agent has a higher probability of selecting high variance plays, it is fairly effective at disrupting what is the highest-variance play of all - shooting the moon. Agents will usually wrest control of the tricks once players are close to shooting the moon. This is even more surprising, considering that agents are not equipped with a sampling module and are unable to read opponent intentions - playing Ace of Hearts in the early game is a fairly clear indication of shooting the moon to all human players, but is completely indiscernible to the agent.

The high disruption of shoot the moon attempts may be related to the incentive structure of the agents. In human play, when it is obvious that a player will shoot the moon, players will nonetheless hesitate on which player should take the burden of disrupting the opposing player's plans by taking points. This is because even though human players are officially graded on the number of points they take, most human players will measure their points relative to other players.

Agents are also very hesitant at shooting the moon. Firstly, this is because their self-play opponents, other agents, are very good at disrupting shoot the moon attempts - internal opponent modelling views these attempts as very unlikely to succeed. Secondly, this is because shooting the moon is viewed as a neutral play, and agents are only concerned with their own score. If shooting the moon grants -26 to the player and 0 to all other players, it is significantly more likely that agents will attempt to shoot the moon.

Conclusion

Overall, AlphaHearts Zero capable of playing Hearts at a human level, outclassing baseline agents and approaching the performance of cheating agents. AlphaHearts Zero is capable of reaching this level despite starting from zero knowledge - thus, the methods demonstrated in this paper can be applied to a wide range of imperfect information games.

Further Work

There are several areas of future work:

- **Specific Sampling:** The current sampling algorithm samples uniformly randomly from all possible states in the information set. However, in reality, the distribution of states per information set is uneven - certain states are significantly more likely given game dynamics and player behavior. Improving the sampling algorithm to include this information would definitely improve the performance of the PIMC system.
- **Stratified Sampling:** The current uniform random sampling system is fairly inefficient, owing to the fact that a significant amount of power is spent determining the locations of irrelevant cards. Specific rules could be put into place to ensure that important cards are distributed in a manner that is representative.
- **Penalizing Variance:** As explained in the prior section, the existing AlphaHearts Zero player generally makes riskier plays that are optimal. Certain measures could be put into place to discourage said behavior.
- **Neural Network enhancements:** Certain improvements to the network could improve the training speed - for example, by altering the input space / adding additional features. Furthermore, more research is needed into determining the optimal architecture.
- **Self-play enhancements:** Certain self-play procedure hyperparameters could be altered to check the impact on training. Examples of hyperparameters that could be changed include the points per game cutoff for swapping to a new network (currently 0.5), the amount of iterations per epoch (currently 64 games), the batch size (currently 32), the batch training frequency (currently once per 4 states), and the value for γ (currently 0.99).
- **Tree search enhancements:** Although the optimal exploration constant was determined, there could be an opportunity to use the neural network to influence the UCB selection mechanism. Furthermore, the tree search algorithm could also be made significantly more efficient by moving the code from Python onto C++, as well as cutting out unnecessary segments in MCTS such as the search conducted when there are no options or in shallow trees.
- **Faster code:** One significant constraint on this project was the speed of the base game itself. All elements of the game algorithm could also be made significantly more efficient by moving the code from Python onto C++.

Experimental Details

Time Benchmarking

As mentioned before, the PIMC AlphaHearts Zero agent runs in approximately 1 second per turn, which is acceptable for play against humans. The agent runs significantly faster as the game tree gets shallower later on in the game. This translates to around half a minute for an all-AI game to be simulated.

The full training and evaluation epoch cycle can take anywhere from 30 minutes to 2 hours, depending on the complexity of the network and the evaluation method. However, it is important to keep in mind that the code running the MCTS and PIMC segments are run in Python and not fully optimized. A more efficient

language such as C++ could substantially improve training speed and results.

All programs were run on the Yale HPC cluster with a standard NVIDIA RTX 2080 Ti GPU.

Optimal amount of determinizations

An experiment was conducted to determine the optimal amount of PIMC determinizations required to play the game of Hearts against the Complex Cheating Player. It is well-understood that an increased amount of PIMC determinizations will lead to a better approximation of the true distribution of states in the information set. However, owing to computing power and time limitations, a reasonable amount of determinizations must be selected.

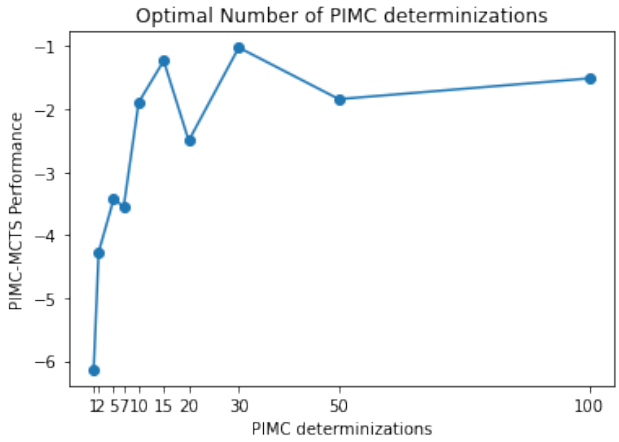

Figure 6: Average amount of points won against the Cheating Complex Player, for a PIMC player with standard MCTS and a given amount of determinizations

The results in Figure 6 show that although increased determinizations do lead to better game performance, gains are fairly minimal after we increase the number of determinizations past 10. Thus, we set the number of determinizations to be 10 for both training and testing.

Optimal amount of MCTS searches

An experiment was conducted to determine the optimal amount of MCTS iterations required to play the game of Hearts against the Complex Cheating Player. As with the case with PIMC determinizations, increasing the amount of MCTS iterations will lead to more accurate gameplay, but will require more computational power.

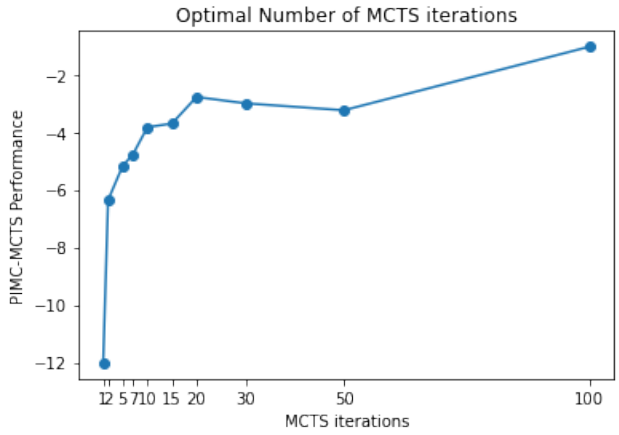

Figure 7: Average amount of points won against the Cheating Complex Player, for a PIMC player with standard MCTS and a given amount of MCTS iterations

The results in Figure 6 demonstrate that increased MCTS iterations leads to better game performance, but the gains slow down after 20 iterations. Thus, we set the number of MCTS to be 20 for training. However, owing to the fact that performance does not approach the level of the Cheating Complex Player until 100 iterations, the number of MCTS iterations will be set to 100 for testing purposes.

Optimal UCB exploration constant

An experiment was conducted to determine the optimal UCB exploration constant K for the game of Hearts. Although the theoretical optimum value for K is $\sqrt{2}$, empirical calculations can give a better value.

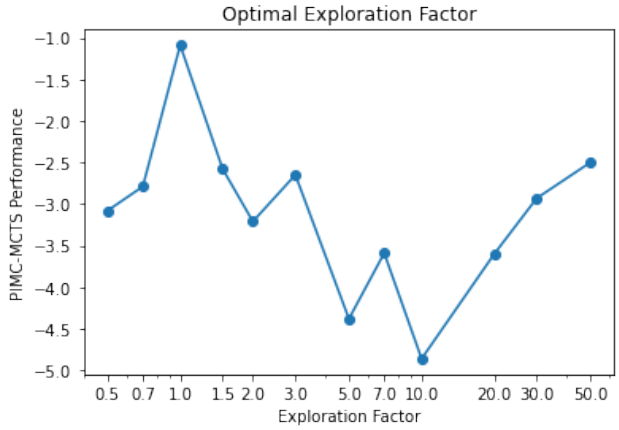

Figure 8: Average amount of points won against the Cheating Complex Player, for a PIMC player with standard MCTS and a set exploration factor.

The results in Figure 8 demonstrate that $K = 1$ should be used for Hearts.

Neural Network Output

To validate the results and give greater insights into the inner workings of our network, I gather statistics for all cards across all games. To run the test, I gather the network output results for the starting positions of 1000 randomly generated games. The Q-value of each card represents the predicted number of points you will win in the entire game if you lead with a certain card.

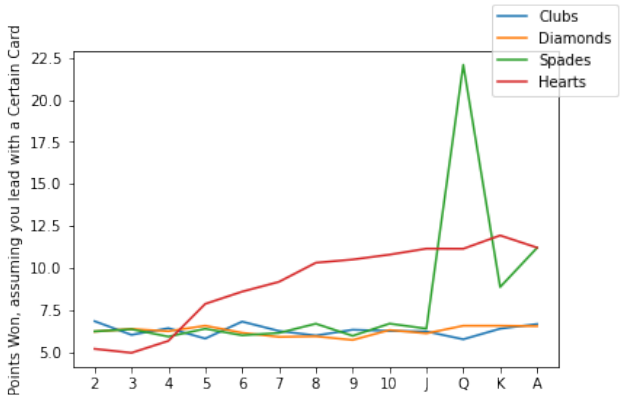

Figure 9: Card Value in Starting Position

On average, a player can expect to win 6.5 points per game. This is confirmed by the fact that most cards have a value of around that in the beginning. Since Clubs and Diamonds and low Spades contain no points in the first round since players are highly unlikely to be void in those suits, there is generally no difference in the value of high or low cards.

As expected, leading with the Queen of Spades is one of the most foolish decisions that can be made by a player, and the Q-network predicts that it will lead to a significant amount of points being lost per player. The same is true with the King and Ace of Spades. Furthermore, leading with higher hearts will generally lead to other players dumping their hearts on you, which will lead to increased points.

Finally, the neural network demonstrates that leading with low hearts generally leads to the best results. This is due to the fact that it is almost guaranteed that the burden of leading will be passed onto the next players, and that the burden of hearts will be passed onto other players. This is a legal move in our variant of Hearts, but in other variants players are not allowed to start with hearts unless it has been "broken".

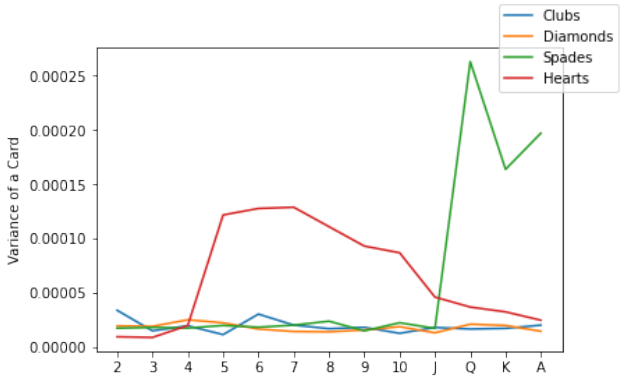

Figure 10: Card Variance in Starting Position

The two types of cards with the highest variance in the starting position are high spades and middle hearts. The reasons behind the high variance of high spades are fairly obvious, but the high variance of middle hearts are less so. If a player plays Five of Hearts, there is a chance that all other players behind them will play the Two, Three and Four. However, there is also a chance that one of the other players will have more low hearts, and some other player will be forced to play a card higher than the Five - explaining the high variance.

Network Inputs and Outputs

The input for the Neural Networks is a $52 \times 4 + 52 \times 4 + 14 \times 4 = 472$ dimensional vector, encoding all information necessary to play the perfect information version of the game.

- **Player Hand Encoding:** This segment of the neural network input is of size 52×4. For each element of the input vector, the vector element is 1 if the player has the indicated card in their hand and 0 elsewise.
- **Current Trick Encoding:** This segment of the neural network input is of size 52×4. For each element of the input vector, the vector element is 1 if the player in question has played the currently indicated card for the current trick and 0 elsewise.
- **Hand History Encoding:** This segment of the neural network input is of size 14×4. For each element of the input vector, the vector element is 1 if the player in question has won the trick containing this card and 0 elsewise. There are only 14 cards that we are concerned with: the Queen of Spades and all 13 hearts.

The output for the Neural Networks is a 52-dimensional vector, encoding the Q-values for the state and the action of playing a card. Since most cards are illegal or impossible to play, a mask is applied to the output of the Neural Network.

Network Architectures

There were three neural network architectures considered in the project, in order of complexity:

- **Fully Connected Three Layer (fc):** A fully connected network with 3 hidden layers and ReLU activation functions.
- **PV Network 16 (PV16):** A fully connected network with 16 hidden layers, 5 skip connections and ReLU activation functions. Adopted from Shi et al.[11]
- **PV Network 24 (PV24):** A fully connected network with 24 hidden layers, 9 skip connections and ReLU activation functions. Adopted from Shi et al.[11]

Evaluation Results with more Complex Neural Networks

In order to validate that all networks were improving through self-play, and to provide a comparison between the networks, I generated a figure similar to Figure 5 for the three aforementioned neural network architectures.

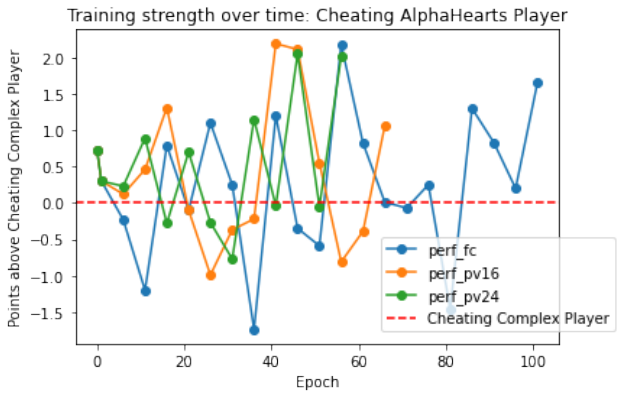

Figure 11: Training Strength over time of MCTS with Neural Network. Epochs > 65 and > 55 are not available for PV16 and PV24, respectively

There appears to be no significant difference between the more simple and complex neural networks, and the latter networks do take significantly more time to train with the limited computational resources available. However, since it appears that none of the aforementioned have reached their training plateaus yet, it appears that the question of network architecture is unresolved as of now. It it is to be expected that as more complex features are learned, the more complex neural networks should outperform the simpler versions.

Acknowledgements

I am grateful for my advisor Prof. James Glenn for his support throughout the entire course of my Senior Project. I also thank Sun Youran and Theodore J. Perkins for their assistance, as well as my DUS Philipp Strack and the CSEC registrar Sabrina Whiteman.

References

[1] S. Bubeck and N. Cesa-Bianchi. Regret analysis of stochastic and nonstochastic multi-armed bandit problems, 2012. URL `https://arxiv.org/abs/1204.5721`.

[2] M. Buro, J. R. Long, T. Furtak, and N. Sturtevant. Improving state evaluation, inference, and search in trick-based card games. In *In Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI2009*, 2009.

[3] P. Cowling, E. Powley, and D. Whitehouse. Information set monte carlo tree search. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(2):120–143, 2012. ISSN 1943-068X. doi: 10.1109/TCIAIG.2012.2200894.

[4] H. Fujita and S. Ishii. Model-based reinforcement learning for partially observable games with sampling-based state estimation. *Neural computation*, 19:3051–87, 12 2007. doi: 10.1162/neco.2007.19.11.3051.

[5] T. Furtak and M. Buro. Recursive monte carlo search for imperfect information games. pages 1–8, 08 2013. ISBN 978-1-4673-5308-3. doi: 10.1109/CIG.2013.6633646.

[6] J. Long, N. R. Sturtevant, M. Buro, and T. Furtak. Understanding the success of perfect information monte carlo sampling in game tree search, 2010.

[7] C. A. Luckhardt and K. B. Irani. An algorithmic solution of n-person games. In *In Proc. of the Fifth National Conference on Artificial Intelligence (AAAI-86*, pages 158–162. Morgan Kaufmann Publishers, 1986.

[8] T. Perkins. Two search techniques for imperfect information games and application to hearts. *CMPSCI Technical Report*, 98-71, 1998.

[9] B. Pfahringer, H. Kaindl, S. Kramer, and J. Fürnkranz. Learning to make good use of operational advice.

[10] C. M. Pipan. Application of the monte-carlo tree search to multi-action turn-based games with hidden information. Master's thesis, Air Force Institute of Technology, 2021.

[11] N. Shi, R. Li, and S. Youran. Scrofazero: Mastering trick-taking poker game gongzhu by deep reinforcement learning. *CoRR*, abs/2102.07495, 2021. URL `https://arxiv.org/abs/2102.07495`.

[12] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm, 2017. URL `https://arxiv.org/abs/1712.01815`.

[13] N. R. Sturtevant and A. M. White. Feature construction for reinforcement learning in hearts.