Pocket ACEs: An Intelligent Poker Bot

Sam O'Nuallain
sonuallain@cs.umass.edu

Derek Lacy
dlacy@umass.edu

Parth Goel
pgoel@umass.edu

1 Introduction

In this project, we present *Pocket ACEs*, an (A)lpha Poker, (C)FR, and (E)volution based poker (s)trategy. Our agent combines several methods with a functional multi-armed bandit approach to perform well in two player limit Texas Hold'em against a diverse range of oppponents and generalize to previously unseen approaches from other teams. We introduce three foundational agents and a combined sampling approach:

- **Alpha Poker:** This agent was designed by Sam O'Nuallain, inspired by Alpha-Zero [9, 8]. Alpha-zero is a technique that at a basic level uses self-play to teach a policy network to play perfect information games using deep learning and search.
- **Counterfactual Regret Minimization (CFR):** This agent was designed by Derek Lacy. It implements a Monte Carlo variant of Counterfactual Regret Minimization (CFR) [13, 5], a game-theoretic algorithm designed to approximate a Nash equilibrium strategy in imperfect-information games.
- **Evolution:** This agent was designed by Parth Goel. This method implements a novel genetic algorithm that evolves a population of linear evaluators to learn competitive poker behavior.
- **Multi-Armed Bandit/Ensemble Approach – Thompson Sampling Algorithm:** Our combined approach was designed by Derek Lacy. We use the *Thompson Sampling Algorithm* [11], a method that probabilistically selects strategies based on past performance, allowing the player to adaptively choose the best approach in each scenario.

Driven by the idea that each of our base methods may perform well against different opponents by focusing on different goals, our aim is to create a player that learns to select the most suitable strategy for each new opponent.
ACE beats all individual agent strategies and converges to a mixed agent strategy within 20 rounds. We expect this combined approach to generalize well to the unseen agents other teams have created.

1.1 Related Work

State-of-the-art poker agents such as Libratus [3] and DeepStack [6] achieve superhuman performance in no-limit Texas Hold'em through game-theoretic reasoning and deep learning. These systems rely on counterfactual regret minimization [10], equilibrium approximation, and real-time neural evaluation, but require significant computational resources.
Reinforcement learning has also shown remarkable success in strategic domains. Notably, AlphaGo and AlphaZero [9, 8] demonstrated the effectiveness of self-play with Monte Carlo Tree Search (MCTS), though attempts to replicate this approach in poker were limited by time and resource constraints. Imitation learning [12] has been explored as an alternative, using expert-like agents to guide early learning.
Evolutionary algorithms offer a lightweight, flexible alternative for strategy learning. Prior work demonstrates their effectiveness in games through population-based search [1], co-evolution [7], and fitness shaping [4]. By evolving compact linear evaluators over curated features, we aim to balance performance and efficiency within our resource limits.
Each of these approaches show promise, but for different reasons. We combine these approaches

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

to leverage the strengths and check the weaknesses of each, forming a more general, robust Pocket ACES agent.

2 Model and Preliminaries

We designed a preliminary **Deterministic Monte Carlo Strategy (DMCS)** agent that utilizes a naive Monte Carlo tree search to run a number of simulations using a hand evaluation score and deciding its action based off a confidence threshold. This simple-but-effective agent provides a useful preliminary benchmark for our combined approach.

In the following, we describe the expected strengths and weaknesses of each agent and our justification and method for combining them.

The **Alpha Poker** agent abstracts the current poker state and transforms the state into a form readable by the underlying network using one-hot vectors and bucketed scalars. An example of the abstraction used is that community cards are represented by suit and rank count vectors. For example, 4C 8D 4C is represented as 2 clubs, 1 diamond, two 4's and one 8. The policy network has two output heads: the value and policy head. The value head learns to predict the value of a given state (the expected amount of chips won from playing on from the current state) and the policy head learns to predict a policy vector given the current state. The policy network is trained on the trajectories of two rules-based DMCS poker agents collected over 2 thousand 500-round games.

This method learns potentially deceptive and exploitive strategies by optimizing for short term gains. More details about model features, architecture and parameters are included in A.1.

Counterfactual Regret Minimization (CFR) [13] is a foundational algorithm for solving imperfect-information extensive-form games such as poker. CFR approximates a Nash equilibrium by simulating self-play and minimizing counterfactual regret over repeated iterations. Each player updates their strategy at every information set, where an information set represents a player's private view of the game, defined in our case by a state abstraction. More details are included in A.2.

We use *Monte Carlo CFR* (MCCFR) [5], specifically the *chance-sampling* variant, which samples a single chance outcome (e.g., board and private cards) per iteration. This significantly reduces computation and variance while preserving convergence guarantees. We also implement the more recent CFR$^+$ algorithm [10], which accelerates convergence by zeroing out negative cumulative regrets and using weighted averaging of strategies. Unlike heuristic or reward-optimizing approaches, CFR converges toward theoretically optimal, minimally exploitable strategies and is expected to perform robustly against opponents who rely on short-term tactics or deception.

We implement a novel **Evolutionary Genetic Algorithm**:
Selection: The top individuals based on fitness are retained as elites. Remaining agents are selected via tournament selection.
Crossover: Pairs of parents produce offspring by combining weights or passing down using a uniform crossover probability.
Mutation: Offspring weights are perturbed with random values under a fixed mutation rate.
Elitism: A small number of top-performing agents (5%) are preserved unchanged each generation to avoid forgetting strong policies.
Each agent evaluates the current game state using a set of 8 handcrafted features inspired by prior poker research [2]. The final action score is computed as a dot product between the agent's weight vector and the feature vector. The fitness of each individual is computed by averaging their performance over simulated matches. Fitness is computed as a weighted combination of baseline performance (70%) and self-play performance (30%). This evaluation strategy balances consistency with adaptability and shows to improve long-term strategic robustness. More information about features, fitness evaluation and parent-child evolution for this agent can be found in A.3.

Thompson Sampling is a Bayesian algorithm for solving multi-armed bandit problems by maintaining a probability distribution over the expected reward of each action (selecting single strategy for a round). At each time step, it samples a value from each strategy's learned reward distribution and selects the strategy with the highest sampled value, balancing exploration and exploitation based on uncertainty. Thompson Sampling is preferred over UCB for our work. It can be more exploitive and

able to converge on an optimal strategy with a limited number of interactions, like a single poker match of up to 500 rounds. We apply this algorithm to our ensemble approach, using it to select between each strategy and expecting it to quickly converge on an optimal strategy for for each new game and opponent.

3 Results

3.1 Alpha Poker

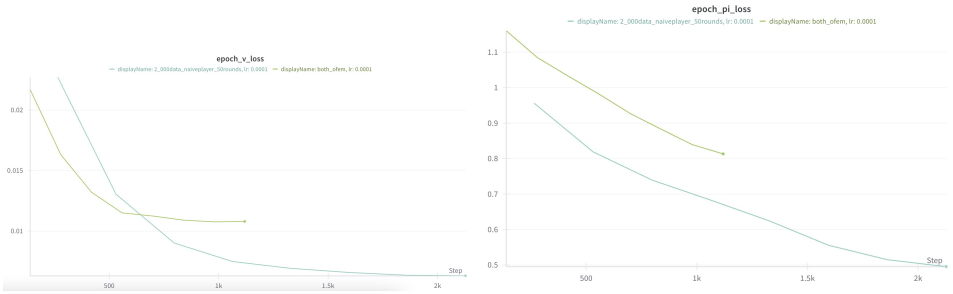

Figure 1: Value and policy head loss for Aggressive Player (green) and Naive Player (blue). The Naive Player has double the amount of data as the Aggressive Player, but both converge.

Due to time and resource constraints which made Alpha-Zero training infeasible (more details in A.1), we pivoted to using imitation learning to teach the policy network for Alpha Poker. Two DMCS agents were used as experts to collect poker trajectories from: Naive Player (NP) and Aggressive Player (AP). NP is the DMCS agent described in section 2, and AP is a version of NP that raises if the expected win rate is greater than or equal to 40% instead of 70%. Two datasets were collected from these agents playing poker: NP vs. NP and AP vs. NP for 1000 games of 50 rounds each. For every state and action taken in these games, the value of the (state, action) pair is computed as the amount of money lost/won at the end of the round.
Two networks were trained on these data. The first network θ_{naive} was trained on the data gathered from NP self-play, and the other θ_{mix} on a mix of the data from the NP self-play game and the NP vs. AP game. Both are evaluated on test splits held out from the collected data. θ_{NP} **achieved 75% accuracy on predicting actions given test states, which** θ_{mix} **achieved 56% accuracy. Both had a similar average MSE for predicting values of states.** Despite θ_{NP} having better test results, θ_{mix} performs better against Raise and Random players demonstrating that it is more robust to different strategies. For the tournament submission, we use θ_{mix}'s predicted policy vector to decide the next action given the current state.

3.2 CFR

We trained our Counterfactual Regret Minimization (CFR) agent over 9000 iterations. In practice CFR can be very computationally intensive. The original paper used up to 10^{12} information sets and trained for over 300 hours on 4 CPUs [13]. To attempt to make CFR a more feasible approach with limited time and compute, we implement a strong state abstraction based on the current round street, board texture, effective stack-to-pot ratio, and estimated win rate. This abstraction reduces the total number of information sets to approximately 3,500. Effective training requires both wide and deep exploration—visiting all information sets and doing so frequently enough to approach a Nash equilibrium involving many simulated game trajectories and tracking cumulative regrets and strategies at each decision point. As shown in figure 2, the **learning curve indicates convergence through broad exploration**. However, qualitative testing showed **difficulty defeating other agents—especially DMCS**—likely due to coarse state abstraction and limited training iterations. Nonetheless, we believe this approach can provide robustness against unseen opponents.

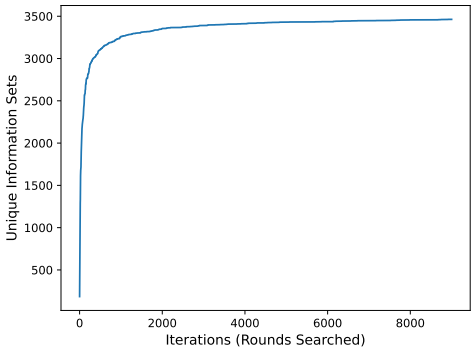

Figure 2: Learning curve for CFR showing the number of unique information sets visited as a function of total training iterations.

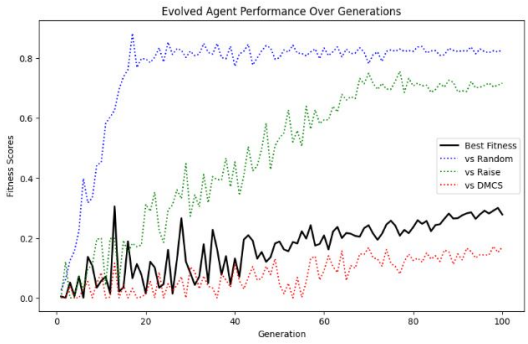

Figure 3: Fitness scores over generations.

3.3 Evolution

Across generations, we observed a steady increase in the best and average fitness scores. Figure 3 shows the progression of fitness over 100 generations for a population of 200, with a clear gap developing between the evolving population against the baseline agents. Agents quickly **learned to outperform the random and raise-only players however, performance against the DMCS agent improved more gradually**, suggesting that learning to exploit probabilistic rollouts required a combination of feature understanding and consistent strategy refinement.

Qualitative inspection of evolved agents revealed distinct behavioral patterns. Agents frequently folded under poor pot odds, raised more aggressively when stack ratios were high, and showed increased selectivity as the game progressed to later streets. Some agents developed bluffing-like behavior, raising with weaker hands when opponent aggression was low—suggesting that strategic adaptation was taking place. This training required heavy parallelization over CPU cores and took over 20 hours to conduct. Due to this heavy computation load, the agents weren't able to converge fully however, based on the trend of learning, further training would see consistent improvement to more optimal weights.

3.4 Multi-Armed Bandit Ensemble

Players	Random	Raise	DMCS	Average
AlphaPoker	0.9	0.7	0.0	0.53
CFR	0.6	0.0	**0.1**	0.23
Evolution	**1.0**	0.6	0.0	0.53
Ensemble	**1.0**	**0.8**	**0.1**	**0.63**

Table 1: Win rates over 10 games with 500 max rounds.

During the evaluations, qualitatively it appears that Thompson's sampling converges to a strategy of 1-2 agents within 20 rounds of poker. We realized that the DMCS agent performed well against our agents and could provide a good backup for when our agents struggle against specific opponent strategies and hence, was added into our final ensemble submission.

Each individual agent has different strengths and weaknesses from each other that can be exploited in combination by the ensemble agent. We expect this behavior to be more important when facing opponent agents created by other teams. It is possible that the CFR or Evolution alternatives, or linear combinations of all methods, will be more successful against agents we have not yet encountered. The goal of our multi-armed bandit ensemble method is to maximize our chances of generalizing to perform well against as many possible opponents as possible.

In conclusion, we demonstrate that our ensemble approach makes an effective AI Poker agent and provides a novel contribution to research in solving imperfect information problems.

4 References

References

[1] Zahra Beheshti and Siti Mariyam Hj Shamsuddin. A review of population-based meta-heuristic algorithms. *Int. j. adv. soft comput. appl*, 5(1):1–35, 2013.

[2] Darse Billings, Denis Papp, Jonathan Schaeffer, and Duane Szafron. Opponent modeling in poker. *Aaai/iaai*, 493(499):105, 1998.

[3] Noam Brown and Tuomas Sandholm. Superhuman ai for multiplayer poker. *Science*, 365 (6456):885–890, 2019.

[4] Mitchell K Colby and Kagan Tumer. Shaping fitness functions for coevolving cooperative multiagent systems. In *AAMAS*, volume 1, pages 425–432. Citeseer, 2012.

[5] Marc Lanctot, Kevin Waugh, Martin Zinkevich, and Michael Bowling. Monte carlo sampling for regret minimization in extensive games. *Advances in neural information processing systems*, 22, 2009.

[6] Matej Moravčík, Martin Schmid, Neil Burch, Viliam Lisý, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 356(6337):508–513, May 2017. ISSN 1095-9203. doi: 10.1126/science.aam6960. URL `http://dx.doi.org/10.1126/science.aam6960`.

[7] Jordan B Pollack and Alan D Blair. Co-evolution in the successful learning of backgammon strategy. *Machine learning*, 32:225–240, 1998.

[8] 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. Mastering chess and shogi by self-play with a general reinforcement learning algorithm, 2017. URL `https://arxiv.org/abs/1712.01815`.

[9] et al. Silver. Mastering the game of go with deep neural networks and tree search. *Nature*, 529 (7587):484–489, January 2016. doi: 10.1038/nature16961.

[10] Oskari Tammelin. Solving large imperfect information games using cfr+. *arXiv preprint arXiv:1407.5042*, 2014.

[11] William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.

[12] Maryam Zare, Parham M. Kebria, Abbas Khosravi, and Saeid Nahavandi. A survey of imitation learning: Algorithms, recent developments, and challenges, 2023. URL `https://arxiv.org/abs/2309.02473`.

[13] Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. *Advances in neural information processing systems*, 20, 2007.

A Additional Results

A.1 Alpha Poker

A.1.1 Features

- **Expected hand strength**: calculate win rate with 50 monte carlo simulations from the current hole and community cards (scalar).
- **Round count**: Current round count (scalar). Bucketed in groups of 5 for abstraction.
- **Community suit and rank counts**: Vector for suit and rank, counts how many are present in community cards. Abstracts community card state by not matching suit and ranks to cards.
- **Hole cards**: More fine-grained than community card representation, uses one hot encoding.
- **Pot amount**: Current amount in pot, standardized by big blind amount.
- **Street**: Current street, encoded as one-hot vector.
- **Pot odds**: Ratio of price to enter pot : pot + price. Allows agent to understand profitability of joining a pot. Standardized by big blind amount.
- **Player stacks**: Standardized by big blind amount.
- **Number of raises in current street**: This is an important feature to have so that the agent can understand opponent moves in the current street, and also how many raises they can make before they are cut off according to the rules.
- **Board texture**: Board texture uses common techniques in poker to identify certain conditions based on the current hole card and community cards. Encoded as a one-hot vector.

A.1.2 Model Parameters

After running the hyperparameter search for imitation learning on Naive Bayes trajectories, the following network architecture was found:

Three 100 dimension residual blocks serve as the base. From this base the policy and value heads are trained, each with one 64-dimensional hidden layer. SiLU is used for the activation function, and Layer Normalization and Dropout is used between all hidden layers of the network to decrease the chance of overfitting. These choices were made with the resource constraints of the tournament in mind.

A.1.3 Alpha-Zero Learning

Alpha Poker originally was meant to emulate Alpha-Zero training, by running self-play over many iterations to train a policy network to be an evaluation function for MCTS. While this method did show signs of learning and improvement, after running on a 4-core laptop for 24 hours only 1000 games of poker were played when these systems require closer to one million games to reach true convergence.

A.1.4 Imitation Learning

While θ_{naive} had a better test accuracy, θ_{mix} outperformed θ_{naive} when tested against agents in 10 500-round games. θ_{naive} struggles to beat the Raise Player even once since it is trained off of two rational agents playing each other, and assumes that an aggressive play implies a strong hand. θ_{mix} has learned that players can bluff, and beats raise player consistently. θ_{mix} also usually beats θ_{naive}. These findings support the hypothesis that combining a mix of experts for imitation learning makes the agent more robust to different strategies and allows the network to learn more.

Using θ_{naive} and θ_{mix} in MCTS to cutoff depth with the value head and choose actions with the policy head during simulation improves performance, but only when upwards of 50 simulations are used. Anything significantly below this amount actually decreases the model performance as measured by playing against the Random, Raise, and Naive players. Since it takes too long to run

MCTS with this many simulations, for the tournament we just take the highest probability action from the predicted policy of the θ_{mix} model.

A.2 Counterfactual Regret Minimization (CFR)

A.2.1 Overview

Counterfactual Regret Minimization (CFR) iteratively updates strategies by minimizing regret for not choosing optimal actions in each information set, conditioned on alternative game histories (counterfactuals). Over time, the average strategy converges toward a Nash equilibrium.

In practice, CFR must traverse a game tree with billions of states, which is computationally prohibitive. We implemented Monte Carlo CFR with chance-sampling [5], which samples individual trajectories to approximate full traversals, and CFR$^+$ [10], which accelerates convergence.

A.2.2 State Abstraction

To reduce computational cost, we abstract the state space to 4 key dimensions:

- **Street**: The current round (preflop, flop, turn, river).
- **Board Texture**: Suit redundancy and rank distribution.
- **Effective Stack-to-Pot Ratio (SPR)**: Captures implied odds.
- **Estimated Hand Strength**: Monte Carlo win rate of current hand.

This abstraction reduces the number of information sets to roughly 3,500, allowing tractable learning over 9000 iterations.

A.2.3 Training and Convergence

Figure 2 tracks exploration breadth across training. The curve indicates that CFR approaches full coverage of the reduced state space over 9000 iterations, trained over approximately 3–4 days. In contrast, more robust CFR implementations often operate on simplified toy versions of poker or run for millions of iterations with substantially larger information sets. To enable feasible training, we drastically reduced the state space in our implementation. Although CFR approaches near-complete coverage of the abstracted state space, limitations in training time (approx. 3 days) and abstraction granularity reduced its effectiveness against adaptive agents like DMCS.

A.3 Evolution

A.3.1 Overview

Our approach evolves a population of poker-playing agents using a genetic algorithm. Each agent is represented by a linear evaluator that maps a vector of normalized game features to a scalar score. This score determines the agent's action selection policy across different betting rounds in a heads-up No-Limit Texas Hold'em setting. The goal is to evolve feature weights that result in high-performing behavior against a combination of static baseline agents and dynamically changing opponents.

The evolution process begins with a population of randomly initialized agents. Over multiple generations, agents are evaluated via simulated poker matches, and their fitness is computed based on win rate and stack performance. The top-performing agents are selected for reproduction, and their weights are recombined using crossover and mutation operators to generate new agents. A small number of elite individuals are passed unchanged to preserve strong strategies.

A.3.2 Features

- **Hand Strength:** Estimated win rate using Monte Carlo simulation of hole cards against random hands.
- **Pot Odds:** Ratio of current pot to the cost of calling a bet.
- **Stack Ratio:** Proportion of the player's stack relative to the starting stack.

- **Board Texture:** Normalized indicator of suit concentration on the board (e.g., flush potential).
- **Street Number:** Encodes the current betting round (preflop, flop, turn, river) as a normalized value.
- **Number of Raises:** Count of raises in the current street, capped and normalized.
- **Opponent Aggression:** Proportion of aggressive actions (e.g., raises) taken by the opponent in the game so far.
- **Payout Estimate:** Product of hand strength and pot size, capped and normalized to account for expected reward.

A.3.3 Fitness Evaluation

- They play multiple matches (e.g., 3) against each static baseline opponent: a random player, a fixed raise-only player, and a naive Monte Carlo-based agent.
- They also play matches against the top 5 agents from the previous generation to introduce adaptive pressure and encourage meta-level learning.

Each match returns the relative stack delta of the agent, which is averaged across games and opponents. Fitness is computed as a weighted combination of baseline performance (70%) and self-play performance (30%), with the option to shift this weighting over time. This evaluation strategy balances consistency with adaptability and has been shown to improve long-term strategic robustness.

A.4 Figures

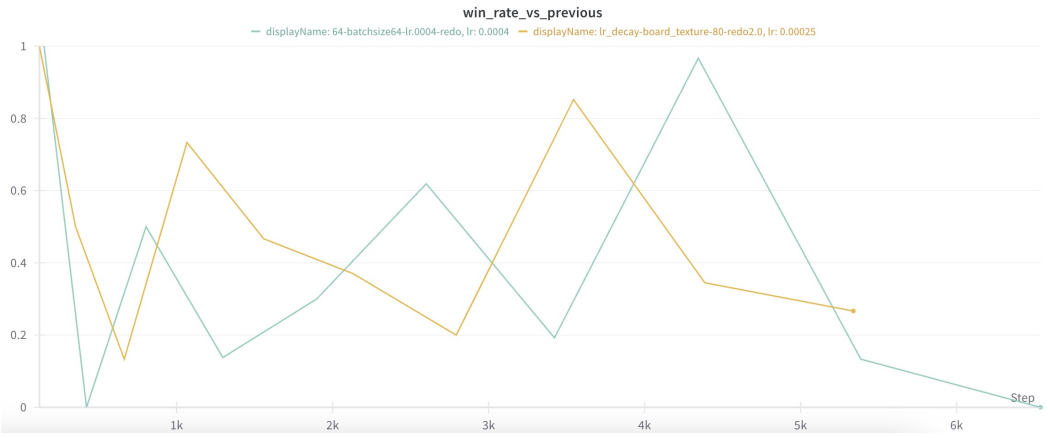

Figure 4: Win rate of two policy networks during Alpha-Zero training. After the end of each iteration of training, the network plays its previous iteration. If it wins more than 60% of games, the new network will be accepted. While the agent was learning to play and was beating previous iterations of the network as seen in the graph, there was not enough time to converge to an optimal strategy with this method.

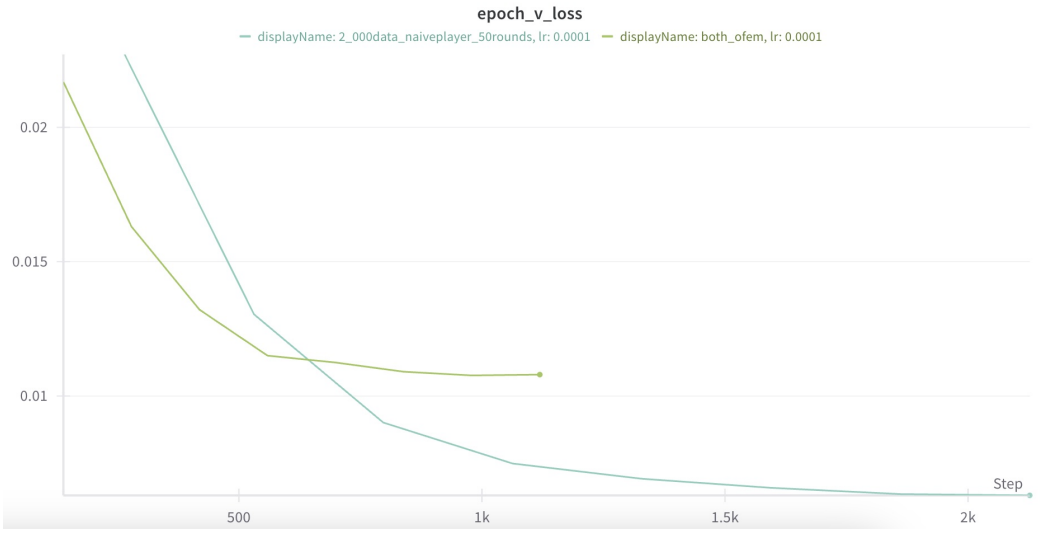

Figure 5: Value head loss for Aggressive Player (green) and Naive Player (blue).

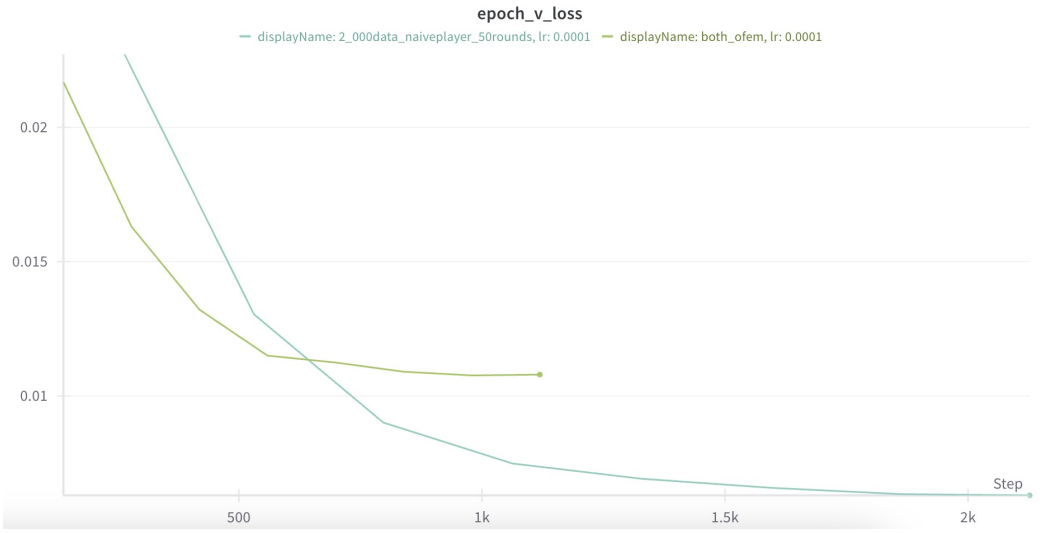

Figure 6: Policy head loss for Aggressive Player (green) and Naive Player (blue).

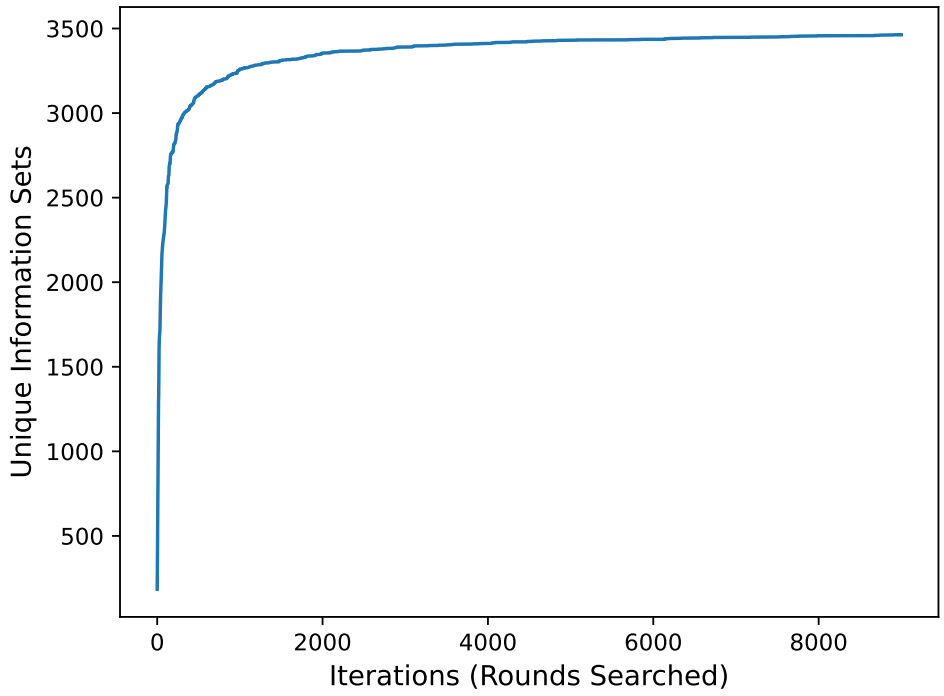

Figure 7: CFR learning Curve

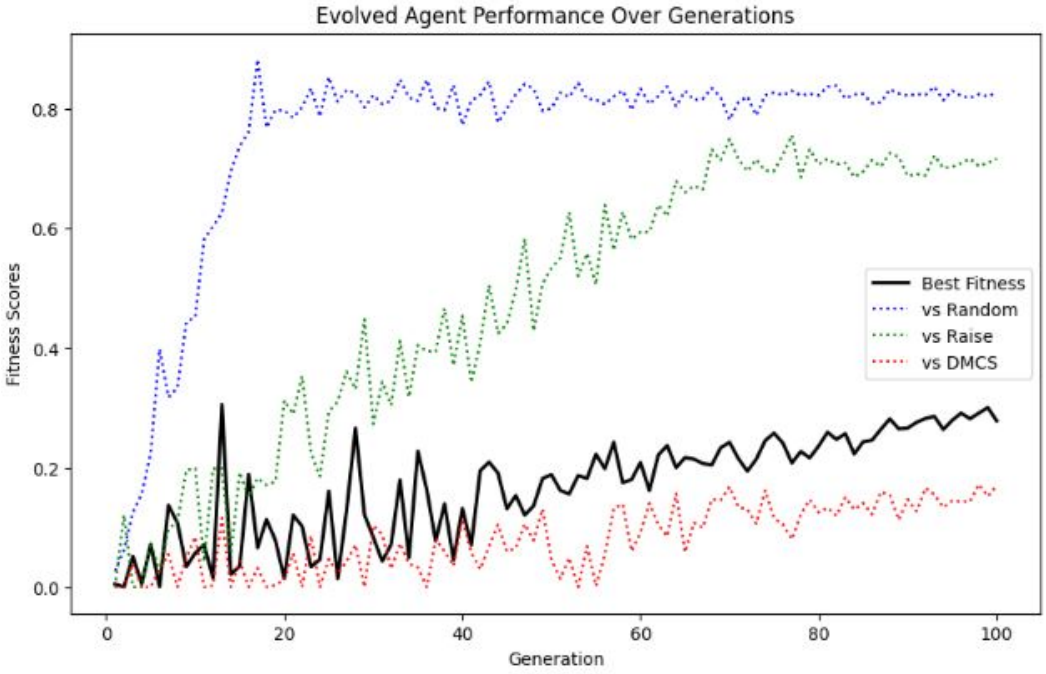

Figure 8: Evolution fitness scores over generations.