Fog of War Chess Engine

Arunim Agarwal, Caleb Gupta,
Lucas Klein

University of Pennsylvania

Abstract: Fog of War Chess, also known as Dark Chess, is a chess variant of modest popularity where each player's vision is obscured, limiting their view to only the squares reachable by their pieces on the current turn. Currently, there is no game engine that is capable of playing Fog of War Chess, let alone at a high level. This comes in stark contrast to the majority of other chess variants that nearly all have engines that play at super-human levels. Several elements have led to the variant being a difficult problem in the field of artificial intelligence. Because the board becomes segmented into a region of known and unknown states, the variation transforms chess from a perfect information to imperfect information game. Due to the game tree being intractably large for the majority of game positions as a result of this imperfect information, we investigated using a model-free reinforcement learning algorithm to approach Nash equilibrium to prevent exploitable positions. This agent is called *Fog6200*, the first Fog of War chess engine. The work applies the implementation of a 2022 DeepMind paper titled "Mastering the Game of Stratego with Model-Free Multi-agent Reinforcement Learning" with a few key differences in memory structure, algorithmic and architecture alterations, and game-specific heuristic improvements.

1. INTRODUCTION

Imperfect information games with large-scale state spaces had been a large milestone for artificial intellgence agents for many years. *DeepNash*, as is presented in the DeepMind paper that introduced a top-3 global Stategо player in 2022, opened the door to many applications within this game structure and environment (Perolat et al., 2022). The innovation presented a significantly improved estimation for Nash equilibrium states that had been seen before, all without keeping an internal representation of the opponent's game state. Inspired algorithms can now have the structure to tackle large state-space problems in game theory including crowd modeling, games, smart grid, auction design, market problems, and much more. We propose one such extension in *Fog6200*, a game engine that is capable of playing Fog of War Chess, a variant to the game of chess. The agent is trained through many iterations of self-play within a larger neural-network architecture that repeatedly learns to modify its policy to approach a Nash equilibrium. This segments the training process into two main categories: The deep learning aspect that through self play is able to learn condensed game representation and action pairings, as well as the novel reinforcement learning algorithm that is based on Regularized Nash Dynamics (R-NaD) within the *DeepNash* architecture.

As described in (Perolat et al., 2020), the usual process of updating the policy by taking the gradient of a loss function or updating networks that guide MCTS was replaced by a three-step process of a reward transformation, follow the regularized leader, and updating the regularization policy. While this setup is general enough to represent both Stratego as well as Fog of War Chess, it is not the optimal setup for converging in a Nash equilibrium. Firstly, it does not include information about the game that was known in the past and since forgotten. This is represented by the movement of some piece to a square that it can no longer have an entire view of the board from. By keeping additional tensor representations of the previously known information, the agent is able to develop a weighted relationship between known and partially known squares, logarithmically decaying with the time since a known

spot. Further, the *DeepNash* algorithm has very little game specific information to specialize its capabilities, in either Stratego or Fog of War Chess. Therefore, a series of heuristics were developed as a way of improving the play and avoiding projectable mistakes that both improved training times and overall output. These heuristics included (among others) King Safety Positioning, Learning new information (and the idea of sacking for it when there are many pieces and not much information about the opponent), castling, focus on diddle of the board in the early game, protecting hanging pieces, queen safety, penalizing the movements of the same piece repetitively, suprise checks, taking of the King, alpha-beta pruning, Quiescence Searching, and static board evaluation functions.

2. BACKGROUND

2.1. FOG OF WAR CHESS STRATEGY

Several elements of strategy are distinctly different between chess and its Fog of War variation, which are key in analysis of what the agent is able to learn. This is distinctly important because no game tree models examinations that are possible in chess are available in the variant as it is unclear what the position of the opponent is, making it all the more important what strategies are built in. The most important feature that should be represented within any capable agent is the King safety. While this is also the case in normative chess, it becomes all the more important due to the unknown attacks that could be aimed toward the king. This is further distinct in that protection does not mean not getting attacked, but also requires more protection in that the attack could come from any unknown direction and select formations are optimal for that protection.

Another key element to the game that is not represented in chess is the misdirection that is applied in order to limit the information of the opponent. This is one of the reasons why better bounds on exploitability can be found with stochastic choosing of actions rather than a simple selection of the highest probability which would result in an opponent having improved predictions of the game state. Human Fog of War chess play often evolves in a more conservative, defensive manner, in line with this approach.

Another important aspect to the game is the movement of pieces to expand the vision of the board. In the perfect information standard version of chess, there is no incentive to move pieces in order to see what they see because the entire board is always visible. This furthers the value of active pieces from those that are involved in the game to those that can improve the knowledge state of the game.

In a similar vein, sacrifices in order to gain information about the state of the game now has a merit compared to the original game where such a concept would not make sense. In analyzing the strategies that come from this approach, it introduces several new chess openings that are losing by engine analysis in standard chess, but yield great results in Fog of War.

The last important difference is the weighed value of distinct pieces. In the optimal positioning on the board in Fog of War chess, a bishop would be able to reveal 13 additional squares on the board, a knight would be able to reveal 8 additional squares on the board, and a rook would be able to reveal 14 additional squares on the board. This has the effect of bishops becoming closer in value to that of a rook than that of a knight in standard chess. In being able to analyze which positions For of War chess engine gets into more often, we would be able to analyze its estimated values of piece worth.

2.2. NASH EQUILIBRIUM IN IMPERFECT INFORMATION GAMES

Nash Equilibrium, a concept initially developed by John Nash is a state in which each player cannot achieve a better reward by unilaterally deviating from their chosen strategy.

In imperfect information games, players do not have complete information about the game or the strategies chosen by their opponents. In imperfect information games, players make decisions based on their beliefs about the likely strategies of others, given the information available to them. For this reason, Nash equilibria for imperfect information games look very different compared to perfect information games. If you play a deterministic policy in an imperfect information game, the adversary can learn your exact policy and gain information about the game state. Thus Nash equilibria in imperfect information games consist of stochastic policies and have a probabilistic outcome.

2.3. Need for Nash Estimation

The challenge for Fog of War chess comes not only in that it is a large scale imperfect information game,

but in that there are also elements of asymmetry and position-dependency that make most of the neural network architectures of large space games ineffective. This has the result of a purely rule-defined action space that therefore cannot use certain algorithmic techniques such as AlphaGo. AlphaZero, however, was able to effectively learn super-human chess level in under a day of training time with the following setup of mathematical form. In place of static game valuation functions or heuristics, AlphaZero implements a deep neural network architecture $(p, v) = f_\theta(s)$ where the network takes the board position as an input and outputs a vector of move probabilities in the form of $p_a = Pr(a|s)$. The value for each state is then calculated with the expectation of the expected state at each given position which in turn yields a clear loss function for the neural network. AlphaZero also improves efficiency through various methods including the implementation of Monte-Carlo tree search (notably instead of Alpha-Beta pruning). Succinctly, the loss function of the neural network is defined as follows:

$$l = (z - v)^2 - \pi^T logp + c||\theta||^2$$

Note that z is the expected values of the state, v is the current value of the state, π is a vector representing a probability distribution over the possible moves either uniformly distributed or bias toward visit counts of states, p is move probabilities, c controls a L_2 regularization hyperparameter, and θ is initially randomized parameters. Further, note that this loss function uses a mean-squared error and cross-entropy losses.

The reason that we are not able to use this architecture, which would allow for a much stronger convergence guarantee is because the states that represent the board representation that must be maintained consist of both known and unknown elements. Further, this makes the exploration space for each possible action significantly larger because each of the possible opponent states all have distinct sets of possible actions that would have to be represented within the model. The aspect of the Monte Carlo tree search and any attempt at model pruning would also not lend itself to the imperfect information aspect of the game. Any learned representation would either have to capture all of the possible state combinations for a observed state tensor or would have to concede the aspect of a model-based learning algorithm for something model-free. The reason that the former option is not possible with current (or near future) technological constraints is because the state space of the game is on the order of magnitude of 10^{124} even without imperfect information, exponentially growing as the moves of the game progress and the potential representations of the board increase.

2.3.1. Follow the Regularized Leader

Follow the leader (FTL) is simply an algorithm with which the agent chooses the action that has produced the most amount of benefit in the past when in that same state. This is a natural algorithm for many reasonably sized state-space games, but fails in its ability to offer regret guarantees because the predictable approach allows for exploitability in these imperfect information games. This fails to product meaningful regret bounds. Follow the regularized leader (FTRL) produces bounds and unpredictability through its regularization function, $\Psi : A \to \mathbb{R}$.

Algorithm 1 Follow the Regularized Leader (Roth, 2023)

for $t = 1$ to T **do**
  Select an action:
  $a_t = \arg\min_{a \in A} \left(\sum_{s=1}^{t-1} \langle a, c^s \rangle + \Psi(a) \right)$
end for

This regularization function constrains the action so that it cannot change too drastically, reducing the exploitability of the agent. FTRL, can recover multiplicative weights updates, if it has the appropriate regularization function. Specifically, one like Gibbs-Shannon Entropy. It is used within the *Fog6200* infrastructure during the dynamics step where replicator dynamics are converging to a fixed policy.

3. RELATED WORK

3.1. Regret Minimization Algorithms

Regret is the standard by which we are able to judge whether an agent made the correct or incorrect decision based off of the other options that were available to the agent. Therefore, minimizing regret, or the shrinking of the difference between what is received from the actions compared to better alternatives, implies better gameplay and is the context with which any agent playing chess would optimize for. When both agents in a two-player zero-sum game minimize their regrets, a Nash equilibrium will form, where neither side can be exploited.

3.1.1. Libratus

Before the development of R-NaD there were several strategies for solving imperfect information that were able to perform at super-human level play in games of complexity rising up to that of No-Limit Poker. A fully generalizable model that does not take in any game-specific information yet was able to defeat top human professionals in HUNL is called Libratus and used three specific game theoretic modules for transitioning between game phases (Brown & Sandholm, 2018).

- **Game Abstraction** This section is used in order to simplify the game and compute state-action spaces with which game-theoric techniques that require smaller search spaces are able to work with. This provides strong early game play.
- **Fine-Grained Game Abstraction**. Abstractions are solved in conjunction with one another, noticeably distinct from the isolated state search techniques that are the standard in perfect information games. Sub-game nesting is implemented to expand the abstraction at points where actions outside of the abstraction space are played.
- **Self-Improver** Estimations on the actions of an opponent in order to perform more in depth searches. Performed with Monte Carlo Counterfactual Regret Minimization

These techniques have several noticeable improvements over the alternative imperfect information techniques, yet still had bottle-necking capabilities that led to us incorporating more structure from the R-NaD implementation. Firstly, it is a direct improvement over techniques that required the game and actions to be known before they are actually played. This is in the case of Counterfactual regret minimization plus, which both has to reason about the entire game as a whole instead of segmented pieces, and additionally is limited to games that it can fit in storage because it must store both the regret information and action probabilities for all information sets that exist within the game. To speak in orders of magnitude, Counterfactual regret minimization plus is able to converge within action spaces of approximately 10^{14} information points, even with strong compression - compared to the HUNL 10^{161} and the 10^{535} in DeepNash. (Brown et al., 2019)

4. APPROACH

4.1. Implementation Details

The general structure for the implementation of *Fog6200* is a R-NaD core that is running to find the Nash equilibrium policy within a game of training during self play, captured within a larger deep neural network framework. Both of the two steps are described in further details below.

4.1.1. Board Representation

Like AlphaGo Zero (Silver et al., 2017), the board state is encoded by spatial planes based only on the basic rules for each game. The actions are encoded by either spatial planes or a flat vector, again based only on the basic rules for each game.

4.1.2. Unexploitability

When thinking about an algorithm that would work in being able to effectively play against a specific type of player, there is an element of exploitability that can be made when optimizing for that style of play. For instance, in the case in which there is a player who always starts with the sequences of moves - e4 Nf3 d4 Nc3 - there becomes an established protocol that the engine would be able to recognize and adapt for. However, if the computer does not have an unexploitable method towards that set of moves then it would have relatively worse play in altered scenarios where the first four moves do not fall in that specific way. For this reason, it is important that a chess engine be unexploitable in its ability to play against any style of play or any subset of moves from the adversary. The way that *Fog6200* is able to train towards unexploitable policies is because it has a Nash equilibrium that is found with each iteration that is by construction, an attractor for the algorithm Let us take a deeper look at each of the algorithms individual components, from (Perolat et al., 2022).

4.2. DeepMind's R-NaD and extensions

We note that Follow the Regularized leader (with an entropic regularizer) is simply an instantiation of replicator dynamics. Further, by default, the "trajectories" defined by FTRL will "orbit" equilibrium strate-

gies, ie. be recurrent, assuming policy-independent rewards and an interior equilibrium (Mertikopoulos et al., 2017).

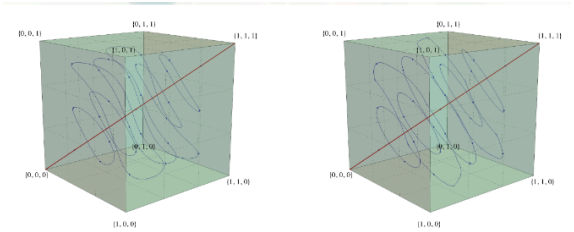

Figure 1. Evolution of the dynamics of FoReL in a 3-player zero-sum polymatrix game with entropic and Euclidean regularization (left and right respectively). The game considered is a graphical variant of Matching Pennies with three players. As can be seen, the trajectories of FoReL orbit the game's line of Nash equilibria (dark red). The kinks observed in the Euclidean case occur when the support of the trajectory of play changes; by contrast, the multiplicative weights dynamics (left) are interior, so they do not exhibit such kinks.

If we then introduce some policy-dependent reward regularization, we can induce convergence at the cost of giving up the precise Nash equilibrium. Note that we now have regularization as a function of action, via FTRL, and regularization as a function of the policy. This point of convergence can be used iteratively to get closer and closer to the Nash equilibrium.

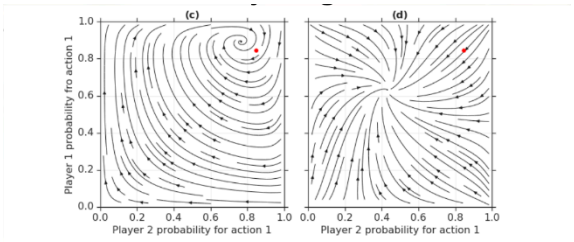

Figure 1. The trajectory plots for FoReL (plot (a)) and for the version with a reward transform with a parameter η multipliers 0.5, 1 and 10 (plots (b), (c), (d) respectively) in a biased matching pennies game (the payoff table for the first player is $[[1, -1], [-1, 10]]$). The red dot is the equilibrium policies of the original game.

An outline of the proof technique used by Perolat et. al is as follows: They define a distance to Nash equilibrium (units of "value") Then, they take a derivative of this distance metric with respect to time steps (iterations of the algorithm).

With a specific entropy-like regularization, you can show this derivative to be equivalent to value-differences between current and NE policy (guaranteed to be ≤ 0 by definition of Nash equilibrium), minus a KL divergence term (which is also ≥ 0 by KL divergence being positive).

Since this distance metric is bounded below, and the derivative is negative, the distance metric must converge to 0, though now to a NE of the regularized game.

Then, with the iterative process (set the regularization policy to be the converged-to policy), we get that the KL divergence between true NE and regularization policy is monotonically decreasing.

This process is described in more mathematical detail in the associated paper and a bit below.

4.2.1. Lyapunov method

This approach is needed to be able to converge to the Nash equilibrium within the replicator dynamics stepping stage. Consider a Lyapunov function to be defined within the context of ordinary differential equations of the form $\frac{d}{dt}y_t = \xi(y_t)$. Definitionally, f is a Lyapunov function if

$$\forall y \neq y^*, \frac{d}{dt}f(y_t) < 0$$

4.2.2. Regularized Nash Dynamics Algorithm

The regularization that exists with this Nash equilibiurm algorithm servers as a way to guarantee convergence and improve the rate at which is is gotten to. The R-NaD individual iteration has three main steps that it can be segmented into. These are the **Reward Transformation Step**, the **Dynamics Step**, and the **Update Step**. The first step essentially improves the rewards for the policy and actions of the player and their opponent based off of the regularization policy. The next step is simply the game play through Follow the Regularized Leader, while the last step updates what has been learned for the next iteration. Note that these three steps are repeated until Nash Convergence. This is the core of the learning algorithm and each of the three steps will be explained in more mathematical detail. Before stepping into each of the steps, let us define some of the terminology that will be used.

- $a^i \in A^i$: an action selected in player i's action space
- π^i: the policy with which player i plays an action (probability distribution over action space)
- $r^i(a^i, a^{-i})$: player i's reward for the simultaneous pair of actions

- $\mathcal{H}$: The Lyapunov function
- η: Regularization parameter (fixed)
- KL-Divergence: Kullback–Leibler divergence, common measurement for determining difference of two probability distributions

4.2.3. Reward Transformation Step

Note that the rewards that are being updated abide by the regularization policy $\pi_{\text{reg}} = (\pi^1_{\text{reg}}, \pi^2_{\text{reg}})$. The reward function that is used an iteratively updated is as follows:

$$r^i(\pi^i, \pi^{-i}, a^i, a^{-i}) = r^i(a^i, a^{-i}) - \eta \log(\frac{\pi^i(a^i)}{\pi^i_{\text{reg}}(a^i)}) + \eta \log(\frac{\pi^{-i}(a^{-i})}{\pi^{-i}_{\text{reg}}(a^{-i})})$$

Note that the regularization policy is initially randomized. Because of the paramaters that are passed in as changing the reward function, this transformation is policy-dependent.

4.2.4. Dynamics Step

A descriptive learning process such as Follow the Regularized Leader is used in order to improve the policies that are used in the first step to update the rewards. This is defined as follows:

$$\frac{d}{d\tau}\pi^i_\tau(a^i) = \pi^i_\tau(a_i)[Q^i_{\pi_\tau}(a_i) - \sum_{b_i} \pi^i_\tau(b^i) Q^i_{\pi_\tau}(b_i)]$$

Q is defined as the quality of the action that is being considered. It does this by measuring the 'fitness' of that action compared to that of the average of all of the other actions.

$$Q^i_{\pi_\tau}(a^i) = \mathbb{E}_{a^{-i}\ \pi^{-i}_\tau}[r^i(\pi^i_\tau, \pi^{-i}_\tau, a^i, a^{-i})]$$

The optimal fixed point, defined as π_{fix}, is guaranteed to converge by the following Lyapunov function:

$$H_{\pi_{\text{fix}}}(\pi) = \sum_{a^1 \in A^1} \pi^1_{\text{fix}}(a^1) \log(\frac{\pi^1_{fix}(a^1)}{\pi^1(a^1)}) + \pi^2_{\text{fix}}(a^2) \log(\frac{\pi^2_{\text{fix}}(a^2)}{\pi^2(a^2)})$$

4.2.5. Update Step

The update step uses regularization of the last fixed point in order to interpolate between the changes from the last iteration. Therefore, after initializing the arbitrarily regularization policy, all others are defined as $\pi_{n+1,reg} = \pi_{n,\text{fix}}$. This converges to π_{nash}. This distance to the Nash equilibrium is a distribution difference and can therefore use KL Divergence:

$$KL(\pi^1_{\text{nash}}, \pi^1_{n,\text{fix}}) + KL(\pi^2_{\text{nash}}, \pi^2_{n,\text{fix}})$$

There is a regularization parameter which both gives stable/faster convergence.

4.3. Neural network representations

Firstly, let us contrast the key differences in the imperfect information that exist between a game such as Stratego and Fog of War Chess. In Stratego, there is perfect information about ones own pieces (meaning both location and type), while there is inperfect information only in the types about ones opponents pieces. Contrarily, in Fog of War chess there is perfect information about ones own pieces (again relaying to location and type), perfect information about the opponents visualized pieces, imperfect information about some of the opponents pieces with regards to type and location, as well as semi-perfect information of both location and type of pieces that had recently been seen. Due to this distinctions, several approaches were needed to be included so that the essence of the algorithm would still hold value.

While similar to the Stratego AI in that it holds both piece-to-type assignments and private and public information tensors, they are distinct in how they are represented to be able to account for incomplete information gain and differing pieces of imperfect information. Note that a piece-to-type assignment for a player is the knowledge of every pieces position that is on the board and can be accounted for by a 8x8x7x10 board where the first two dimensions represent the row and column of the board, whereas the third dimension represents the piece assignment (pawn, rook, knight, bishop, queen, king, or empty), and where the last dimension represents a 9 move state history of the last time that it was seen (if it had even been seen, otherwise 10 is representative of unobserved). Let us define a tensor that represents the probabilities of this square having the assigned piece value at that given time stamp as follow:

$$Pub_i[r, c, k, t] = \left\{ \begin{array}{ll} 0, & \text{if no piece at (r,c)} \\ 1, & \text{if } k \text{ is at (r,c)} \\ \frac{9-t}{9}, & \text{if } k \text{ was at (r,c) } t \text{ ago} \\ \frac{\#\text{pieces}}{\#\text{unseen}_{r,c}}, & \text{if piece k is at (r,c)} \end{array} \right\} \quad (1)$$

Table 1: *Components of Agent Observation*

observation component	shape
Player's private information, Prv_i	8x8x7x10
Opponent's public information, Pub_{-i}	8x8x7x10
Player's public information, Pub_i	8x8x7x10
Percent of game expected to be complete	scalar
The last two moved pieces	7 x 2 height

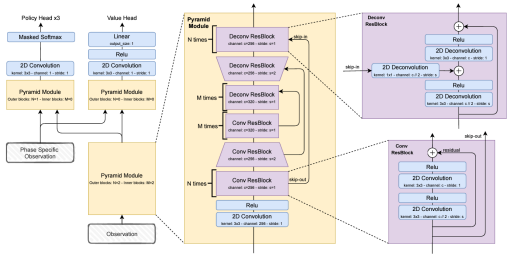

Figure 1: DeepNash Architecture

4.4. Heuristics

Heuristics are important to Fog of War chess in order to improve the training time of the algorithm and to nudge it in the right direction when there is not an obvious choice that is produced by the neural network. In order to not break the bounds and regret of. the algorithm, we must produce a new error bound, ϵ, such that the deviation in probability from the mean of any of the actions that are played differ at most that much after heuristics are applied. The follow heuristics are implemented in decreasing order of relevance to the game.

1. **King Safety Positioning**
   King Safety is of more importance than in standard variations of chess because the imperfect information element to the game means that any 'fogged' square that has either a diagonal, horizontal, vertical, or knight shape away from the king provides a potential attack that could end the game. Because this is high stakes relative to the game, this changes the outputted move probability distribution at a rate that is more significant. This heuristic is measured through counting the total number of squares that could have pieces that could attack the King

2. **Information Gain**
   Information gain is inherent to the deep learning architecture in that the algorithm will learn that moving pieces that develop a greater understanding of the board. However, we found that including this explicit heuristic improved the convergence time to more reasonable rates within the algorithm. This is implemented by performing single shot look ahead in the movement of every action in order to determine what the information gain would be from that specific action relative to its current state.

3. **Castle**
   Castling, similar to information gain, should be learned by the algorithm with enough iterations as it performs the dual service of protecting the king while also developing a piece. Again, this heuristic still slightly improved performance and convergence to reasonable rates of success when trained from scratch.

4. **Middle Board Focus**
   This, alike many of the other heuristics, is a key element of the Stockfish game engine which is able to perform extremely well on chess without the conventional deep net architectrues that have established as state-of-the-art. A larger on the middle of the board allows more developed pieces, larger information gain, and stronger attack against the opponent.

5. **No Hanging Pieces**
   A hanging piece is defined as one that is under attack and there is not a defending piece that would be able to capture back. This is an oversimplification of the issue because the opponent could have multiple attackers to the single defender, which could be unknown in the information space. This level of nuance is the aim of the neural network architecture as heuristics can only offer so much while still being time efficient or sufficiently complicated on their own.

6. **Quiescence Searching**
   Detect which moves have a high potential of changing the state balance of the game in a significant way

7. **Static Board Evaluation Function**
   This serves to look at the general state of the board and make assumptions about what kind of piece positioning would be better or worse without the context of the entire history of the game. This encourages positive piece mobility.

   a) **Pawn advancement**
   How far up the board has each pawn advanced. Reaching the opposite end is important because it promotes the pawn to a

different piece.

b) **Piece Mobility**
Separate for each of the pieces. How many different spaces can the given piece move to?

c) **Piece Threats**
Separate for each of the pieces. How many of the opponent's pieces are threatened by attack? This includes checks (which is a threat on the king)

d) **Piece Protects**
Separate for each of the pieces. How many of your own pieces are protecting the given piece to prevent it from being captured without repercussion?

5. RESULTS

5.1. Mini Fog-of-War

Due to computational constraints, we began the training runs on a few different models and techniques with the substantially reduced 4x4 version of the game. This smaller scale game has a subset of the pieces on the first row for each player. See Figure 4. We used this game because it is non-trivial to solve in the Fog of War setting, and retains the key aspects of the game: sequential IIG, possibility of losing knowledge after gaining it, and not knowing where the pieces might be. The latter two of these notably distinguish the game from Stratego.

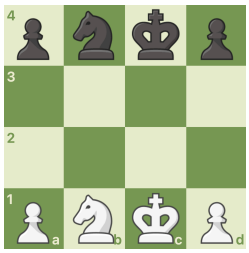

Figure 2: 4×4 version of Fog of War Chess

We trained both a straightforward MLP and a version of the convolution-based architecture shown in Figure 1, and found similar results, but a slightly faster convergence rate for the convolution based architecture. Both techniques had high win rates ($\approx$ 95%) against random opponents, made more impressive by the upper bound on wins against random players being significantly below 100%. [1]

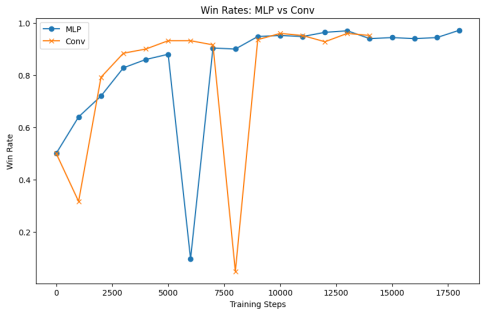

Figure 3: Win Rates vs. Random over the course of training for both architectures

5.2. Full Fog-of-War

We trained and evaluated a convolution-based architecture for the full dark chess game, but found that the training was quite unstable. We plot averaged high points below:

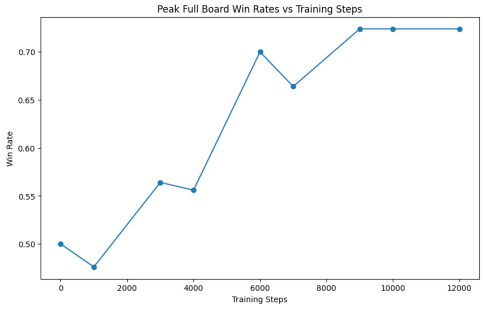

Figure 4: Win Rates vs. Random over the course of training for both architectures

REFERENCES

Brown, N., Lerer, A., Gross, S., & Sandholm, T. (2019, September). Deep counterfactual regret minimization. In K. Chaudhuri & R. Salakhutdinov (Eds.), *Proceedings of the*

[1] To support this, we found that at a simple analysis depth of two, the black player can not hope to win more than $63/64$ of their games against a random player. Given that the white player has 8 starting moves, 4 of which together can threaten both the black king's starting square and any square it may move to in one move, the best black can do is to randomize over their choice of king move (b3, c3, d3, or a non-king move), which must lead to a 1/8 chance of white lucking into a mating position, followed by another 1/8 chance of them randomly taking the king.

36th international conference on machine learning (pp. 793–802, Vol. 97). PMLR. https : / / proceedings . mlr . press / v97 / brown19b.html

Brown, N., & Sandholm, T. (2018). Superhuman ai for heads-up no-limit poker: Libratus beats top professionals. *Science*, *359*(6374), 418–424. https : / / doi . org / 10 . 1126 / science . aao1733

Mertikopoulos, P., Papadimitriou, C., & Piliouras, G. (2017). Cycles in adversarial regularized learning.

Oesterheld, C., Treutlein, J., Grosse, R., Conitzer, V., & Foerster, J. (2023). Similarity-based cooperative equilibrium.

Perolat, J., De Vylder, B., Hennes, D., Tarassov, E., Strub, F., de Boer, V., Muller, P., Connor, J. T., Burch, N., Anthony, T., McAleer, S., Elie, R., Cen, S. H., Wang, Z., Gruslys, A., Malysheva, A., Khan, M., Ozair, S., Timbers, F., … Tuyls, K. (2022). Mastering the game of stratego with model-free multiagent reinforcement learning. *Science*, *378*(6623), 990–996. https://doi.org/10.1126/science.add4679

Perolat, J., Munos, R., Lespiau, J.-B., Omidshafiei, S., Rowland, M., Ortega, P., Burch, N., Anthony, T., Balduzzi, D., Vylder, B. D., Piliouras, G., Lanctot, M., & Tuyls, K. (2020). From poincaré recurrence to convergence in imperfect information games: Finding equilibrium via regularization.

Roth, A. (2023). Learningingames(and gamesinlearning). https://www.cis.upenn.edu/~aaroth/GamesInLearning.pdf

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., & Hassabis, D. (2017). Mastering chess and shogi by self-play with a general reinforcement learning algorithm.

A. Pareto Related Ideas

Below, we explore a specific form of game to which R-NaD is applicable, and discuss Pareto optimality. From Similarity-based Cooperative Equilibrium (Oesterheld et al., 2023): "Let Γ be a two-player additively decomposable normal-form game. Then Γ has a Nash equilibrium that very weakly Pareto-dominates all other Nash equilibria. If Γ is furthermore symmetric, then in the Pareto-dominant equilibrium, both players receive the same utility." Where "we say that σ very weakly Pareto-dominates $\hat{\sigma}$ if for all i, we have that $u_i(\sigma) \geq u_i(\hat{\sigma})$."

Building on this, we claim two things:

1. This result holds for Sequential Imperfect Information Games as well.
2. R-NaD can converge to the very weak Pareto optimal Nash equilibrium, but does not necessarily.

From (Perolat et al., 2020), R-NaD is described to work in the context of "monotone" games, which are split into three categories (note that V is implicitly $V(h_{\text{init}})$, where h_{init} represents the history at the initial state:

1. 2-player zero-sum games, defined as satisfying $V^1_\pi = -V^2_\pi$ for all π.
2. zero-sum N-player polymatrix games, which can be decomposed into a sum of pairwise interactions $V^i_\pi = \sum_{j \neq i} \tilde{V}^i_{\pi^i, \pi^j}$, where $\tilde{V}^i_{\pi^i,\pi^j} = -\tilde{V}^j_{\pi^j,\pi^i}$.
3. Additively decomposable games, where $V^i_\pi = \bar{V}^i_\pi + \bar{V}^i_{\pi^{-i}}$

We now focus on the third category, additively decomposable games. We claim that sequential games satisfy the same theorem from Oesterheld 2023, and therefore have a very weak Pareto optimal Nash equilibrium.

This is fairly straightforward, as the proof follows the same structure as it does for normal form games, with actions replaced by policies. However, this only holds for 2-player games [2].

We begin from the definition: $V^i_\pi = \bar{V}^i_{\pi^i} + \bar{V}^i_{\pi^{-i}}$

Now say i would like to best respond to an arbitrary π^{-i}.

To do so they play some strategy in the set $S^i = \arg\max_{\pi^i} V^i_{\pi^i,\pi^{-i}} = \arg\max_{\pi^i}(\bar{V}^i_{\pi^i} + \bar{V}^i_{\pi^{-i}})$

$= \arg\max_{\pi^i}(\bar{V}^i_{\pi^i})$, as $\bar{V}^i_{\pi^{-i}}$ does not depend on π^i

These best responses are now independent of π^{-i}, and thus are best responses to all possible opposing strategies.

We can construct a specific Nash equilibrium as follows: select a policy $\bar{\pi}^i = \arg\max_{\pi^i \in S^i}(\bar{V}^{-i}_{\pi^i})$ for each player i. We can claim that the policy $\bar{\pi} = (\bar{\pi}^1, \bar{\pi}^2)$ is a Nash equilibrium, as each player is playing a best response to the other players' strategies.

Now, we can show that this Nash equilibrium is very weakly Pareto optimal over any other Nash equilibrium $\hat{\pi}$. First, we note that each player's strategy $\forall i, \hat{\pi}^i \in S^i$, as if it were not, then by definition of S^i, player i's reward could be increased by switching to a policy in S^i. From this fact, it follows that:

$\bar{V}^i_{\pi^i,\pi^{-i}} = \bar{V}^i_{\pi_i} + \bar{V}^i_{\pi^{-i}} = \max_{\pi^i \in S^i}(\bar{V}^i_{\pi^i}) + \max_{\pi^{-i} \in S^{-i}}(\bar{V}^i_{\pi^{-i}}) \geq \hat{V}^i_{\pi^i} + \hat{V}^i_{\pi^{-i}} = \hat{V}^i_{\pi^i,\pi^{-i}} \forall i$

Now, we move on to examine whether R-NaD is theoretically guaranteed to converge to this Pareto optimal in the two-player additively decomposable setting. We were not able to make substantial progress on this problem purely theoretically, but we note two things which give us a range within which to further explore:

First, in (Perolat et al., 2020), they claim that for any Nash equilibrium π^* of the original game, there is an initialized regularization policy and regularization strength that will cause the process to converge to π^*.

Second, we performed a simple empirical analysis on additively decomposable normal-form games and found that the policies rarely converge to the Pareto optimal equilibria because of the initialization- and hyper parameter-dependence of the process.

[2] We can show a short counterexample for the $N = 3$-player, action size $2, A = \{+, -\}$ case by considering the following game G: let $\bar{V}^i_{\pi^i} = 1 \forall i, \pi^i$, meaning any strategy confers each player some arbitrary constant reward to themself. This makes any strategy a Nash equilibria, since no unilateral strategy (change) can improve their reward. Now, again for simplicity, we define $\bar{V}^j_{\pi^i} = 0$ for $i = 2, 3, j \neq i$. This means nothing that players $2, 3$ do has any effect. Finally, let $\bar{V}^2_{\pi^1 = \pm} = \pm 1, \bar{V}^3_{\pi^1 = \pm} = \mp 1$. This makes any choice π^i suboptimal for either player 2 or 3, and thus makes any overall policy π both a Nash equilibrium and *not* Pareto optimal.