A Comparison of Self-Play Algorithms Under a Generalized Framework

Daniel Hernandez, Kevin Denamganaï, Sam Devlin, Spyridon Samothrakis, and James Alfred Walker, *Senior Member, IEEE*

***Abstract*—The notion of self-play, albeit often cited in multiagent reinforcement learning as a process by which to train agent policies from scratch, has received little efforts to be taxonomized within a formal model. We present a formalized framework, with clearly defined assumptions, which encapsulates the meaning of self-play as abstracted from various existing self-play algorithms. This framework is framed as an approximation to a theoretical solution concept for multiagent training. Through a novel qualitative visualization metric, on a simple environment, we show that different self-play algorithms generate different distributions of episode trajectories, leading to different explorations of the policy space by the learning agents. Quantitatively, on two environments, we analyze the learning dynamics of policies trained under different self-play algorithms captured under our framework and perform cross self-play performance comparisons. Our results indicate that, throughout training, various widely used self-play algorithms exhibit cyclic policy evolutions and that the choice of self-play algorithm significantly affects the final performance of trained agents.**

***Index Terms*—Emergent phenomena, machine learning, multiagent systems.**

I. Introduction

IN THE classical single-agent reinforcement learning (RL) scenarios described by [1], where a stationary environment is modeled by a Markov decision process (MDP), a solution concept can be defined. MDPs are solved by computing a policy which yields the highest possible episodic reward. However, it is not clear how to define a pragmatic solution concept when training a single policy in a multiagent system, for an agent's optimal strategy is dependent on behaviors of the other agents that inhabit the environment. An initial solution is to compute the expected reward obtained by a given policy defined over the *entire* set of all possible other policies in the environment, which is intractable in all but toy scenarios.

Manuscript received December 30, 2019; revised November 24, 2020; accepted January 28, 2021. Date of publication February 11, 2021; date of current version June 16, 2022. This work was supported in part by the EPSRC Centre for Doctoral Training in Intelligent Games and Game Intelligence (IGGI) under Grant EP/L015846/1; and in part by the Digital Creativity Labs, Jointly Funded by EPSRC/AHRC/Innovate U.K., under Grant EP/M023265/1. *(Corresponding author: Daniel Hernandez.)*

Daniel Hernandez, Kevin Denamganaï, and James Alfred Walker are with the Department of Computer Science, University of York, York YO10 5DD, U.K. (e-mail: danielhp95@gmail.com; kyd500@york.ac.uk; james.walker@york.ac.uk).

Sam Devlin is with Microsoft Research, Cambridge CB1 2FB, U.K. (e-mail: sam.devlin@microsoft.com).

Spyridon Samothrakis is with the Institute of Analytics & Data Science, University of Essex, Colchester CO4 3SQ, U.K. (e-mail: ssamot@essex.ac.uk).

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TG.2021.3058898.

Digital Object Identifier 10.1109/TG.2021.3058898

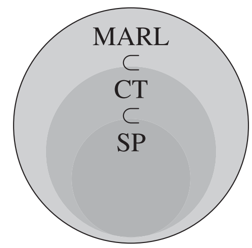


Fig. 1. Inclusion relationship of MARL training schemes

In order to train one or more policies to act in a multiagent environment, multiagent RL (MARL) algorithms, also known as training schemes, are used. A MARL training scheme is the process by which one or more policies are trained in a multiagent environment. There are two clearly defined categories which differ in the assumptions made about the accessibility and capacity to modify agents' policies in an environment: decentralized and centralized training (CT). In decentralized training, each agent is assumed to be independent from other agents, learning a policy by processing only their own local environment signals (observations and rewards), with no external global module to coordinate policy learning between agents. In centralized training, there is an additional entity overlooking the entire system of agents. This module can take many forms, such as a centralized critic [2] advising all learning agents on how to update their policies. We focus on a set of centralized systems that train a *single* policy by choosing which other *fixed* policies will define the behavior of the other agents in the environment [3], [4]. This inclusion between MARL training schemes is visually captured in Fig. 1.

Traditionally, such MARL methods that use a centralized policy-deciding module would train a single policy by matching it against a set of *pre-existing* and fixed policies, using as a success metric, the relative performance against these fixed agents. These methods rest on two assumptions: first, the availability of benchmarking policies to train and test against; secondly, these existing policies dominate, in a game theoretical sense, most of the policy space. Thus, it would not be necessary to compute the expectation over the entire policy space, using as a proxy an expectation over the preexisting policies. However, this approach is flawed. If this benchmarking set is too small, the trained policy may overfit to the behavior of the agents it was trained with, and thus prone to being exploitable by other policies. Furthermore, the validity of the last assumption is rarely formally justified, favoring empirical results.

As early as the 1950s, authors such as Samuel [5] began experimenting on self-play (SP), an open-ended [6] MARL centralized training scheme. An SP training scheme trains a learning agent *purely* by simulating plays with a copy of itself, or *fixed* policies generated during training. These generated policies can dynamically build a set of benchmarking policies during training. Such a set can potentially be curated to remove dominated or redundant policies. Once we leave behind the limiting approach of training against a fixed and known set of policies in favour of SP, it is of paramount importance to define meaningful metrics to inform this open-ended learning process. Fortunately, recent years have seen the introduction of metrics for multiagent evaluation, stemming from game theory [6] or dynamical systems analysis [7].

Historically, SP lacks a formal definition, and notation is often not shared among researchers. This has led to isolated, and sometimes conflicting, conceptions of what constitutes SP as a training scheme in MARL. It is our firm belief that a formally grounded framework with rigorous and unified notation will strengthen the field of SP MARL and allow for incremental efforts on existing and future contributions to be captured on a shared language. This article constitutes a first step toward defining a generalizing framework under which SP MARL methods can be inspected. Our contributions are as follows:

1) a generalizing framework defined under formal notation to describe SP algorithms in MARL;
2) a unifying definition under the presented framework of some prevalent SP algorithms from the literature;
3) a qualitative and quantitative study of some SP algorithms.

II. Related Work

The notion of SP has been present in the game playing AI community for over half a century. Samuel [5] discusses the notion of learning a state-value function to evaluate board positions in the game of checkers, to later inform a 1-ply tree search algorithm to traverse more effectively the search space. This learning process takes place as the opponent uses the same state-value function, both playing agents updating simultaneously the shared state-value function. Such training fashion was named self-play. The TD-Gammon algorithm [8] featured SP to learn a policy using TD(λ) [1] to reach expert level backgammon play. This approach surpassed previous work by the same author, which derived a backgammon playing policy by performing supervised learning on expert datasets [9]. More recently, AlphaGo [10] used a combination of supervised learning on expert moves and SP to beat the world champion Go player. This algorithm was later refined [11], removing the need for expert human moves. A policy was learnt purely by using a mix of supervised learning on moves generated by SP and Monte Carlo Tree Search (MCTS), as presented in [12]. These works echo the sentiment that superhuman AI need not be limited or biased by preexisting human knowledge.

It is often assumed that a training scheme can be defined as SP if, and only if all agents in an environment follow the same policy, corresponding to the latest version of the policy being trained. Meaning that, when the learning agent's policy is updated, every single agent in the environment mirrors this policy update. We refer to this SP method as *naive* SP. Bansal *et al.* [13] relax this assumption by allowing some agents to follow the policies of "past-selves". Instead of replicating the same policy over all agents, the policy of all of the non-training agents can *also* come from a set of *fixed* "historical" policies. This set is built as training progresses, by taking *checkpoints*[1] of the policy being trained. At the beginning of a training episode, policies are uniformly sampled from this "historical" policy set and define the behavior of some of the environment's agents. The authors claim that such a version of SP aims at training a policy which is able to defeat random older versions of itself, ensuring continual learning. This notion of "choosing policies from a historical set" allows for two decision points: 1) which agents will be added into this "historical" set of policies and 2) which of these agents will populate the environment. Different takes on 1) and 2) spawn different SP algorithms.

From this scenario, consider the following: each *combination* of fixed policies sampled as opponents from the "historical" dataset can be considered as a separate MDP. This is because by leaving a single agent learning in a stationary environment, the fixed agents' influence on the environment is stationary [14]. This is of genuine importance, given that most RL algorithms' convergence properties heavily rely on the assumption of a stationary environment. SP algorithms can leverage the assumption that they are using SP, so they can provide the learning agent with a label denoting which combination of agent behaviors inhabits the environment, a powerful assumption in transfer learning [15] and multitask learning [16]. In fact, there are already multitask meta-RL algorithms, which assume knowledge of a distribution over MDPs that the agent is being trained on, such as RL^2 [17]. Note that an SP algorithm featuring a growing set of "historical" policies will introduce a nonstationary distribution over the policies that will inhabit the environment during training. It ensues that the distribution over the set of MDPs encountered by the training agent becomes nonstationary.

Recently, Berner *et al.* [18] trained a team of RL agents using SP to achieve superhuman level performance in the competitive team-based game of Dota 2. During training, the team would play 80% of the games using *naive* SP while the remaining 20% were played against "past-selves." The probability of facing any of these previous policies depends on a per-policy metric (which is updated during training) evaluating how much there is to learn from a policy. AlphaStar [19] reached Grandmaster level in StarCraft II with various policies by using a combination of various SP algorithms [20]. Part of their training pipeline relied on training a set of "exploiter" policies, which focus on exploiting specific policies under training, relaxing the need for them to be robust to all opponents.

[1] For deep RL, this is equivalent to freezing the weights of the neural networks to represent an agent's policy.

Lanctot *et al.* [4] define the Policy-Space Response Oracles (PSRO) family of algorithms, unifying various game theoretical algorithms for multiagent training. PSRO algorithms tackle this problem by iteratively generating monotonically stronger policies relative to an existing set of policies. These algorithms iterate over the following loop: a meta-game (see definition in Section III) is defined over the current set of policies, for which a "solution" is computed, and from this solution, one or more policies are added to the set of policies. The choice of solution concept and the procedure to generate new policies from this concept is the differentiating factor between PSRO algorithms. There are current efforts to show convergence properties of some PSRO algorithms [6], [21] toward existing multiagent solutions [7]. Our contribution shares the spirit of creating a generalized framework to encompass existing algorithms, but with a focus on MARL literature instead of game theory.

III. Preliminary Notation

Cursive lowercase letters represent scalars (n). Bold lowercase letters represent vectors ($\boldsymbol{\pi} \in \mathbb{R}^n$). Bold uppercase letters represent matrices ($\boldsymbol{A} \in \mathbb{R}^{n \times n}$).

A. Normal Form Games and Winrate Matrices

A *normal form game* is a tuple (Π, U, n) where n is the number of players, $\Pi = (\Pi_1, \ldots, \Pi_n)$ is the set of joint policies, one for each player. $U : \Pi \to \mathbb{R}^n$ is a payoff table mapping each joint policy to a scalar utility for each player.

Rational players try to maximize their own expected utility. Each player i does so by selecting a policy from Π_i or equivalently by sampling from a mixture (distribution) over them $\pi_i \in \Delta(\Pi_i)$. The *value* v_i for a player i given a policy vector $\boldsymbol{\pi}$ is the expected payoff obtained by player i if all players follow $\boldsymbol{\pi}$, $v_i = U_i(\boldsymbol{\pi})$.

A (possibly mixed) policy π_i is a best response for player i against all other players' policies $\boldsymbol{\pi_{-i}}$ if playing π_i yields player i the highest possible payoff against strategies $\boldsymbol{\pi_{-i}}, \pi_i \in BR(\boldsymbol{\pi_{-i}})$. A Nash equilibrium is a policy profile (one policy for each player) such that each player's policy is a best response against all other player policies. $\forall i \in \{n\},\ \pi_i \in BR(\boldsymbol{\pi_{-i}})$. A Nash equilibrium is maximally entropic (maxent Nash) if each player's policy is maximally indifferent between actions with the same empirical performance.

A game is zero-sum if $\forall \pi \in \Pi,\ \mathbf{1} \cdot U(\Pi) = 0$, otherwise it is a general-sum game. A game is symmetric if all players feature the same policy set ($\Pi_1 = \cdots = \Pi_n$) and the payoff associated to each joint policy depends only on the policies and not on the identity of the players. 2-Player normal form games ($n = 2$) are typically defined by a tuple ($\boldsymbol{A}, \boldsymbol{B}$), where $\boldsymbol{A} \in \mathbb{R}^{|\Pi_1| \times |\Pi_2|}$ gives the payoff for player 1 (row player), and $\boldsymbol{B} \in \mathbb{R}^{|\Pi_1| \times |\Pi_2|}$ gives the payoff for player 2 (column player). If $\boldsymbol{B} = \boldsymbol{A^T}$ the game is symmetric. Most importantly for us, if $\boldsymbol{B} = -\boldsymbol{A}$, the game is zero-sum. Exploiting this equality, 2-player zero-sum games are often represented by a single matrix $\boldsymbol{A}$ containing the payoffs for player 1.

Given a vector of n agents $\boldsymbol{\pi}$ for an arbitrary game, also known as a population, let $\boldsymbol{W_\pi} \in \mathbb{R}^{n \times n}$ denote an *empirical winrate matrix* also known as a *meta-game*. The entry $w_{i,j}$ for $i, j \in \{n\}$ represents the winrate of many head-to-head matches of policy π_i when playing against policy π_j for the given game. A meta-game can be thought of as an abstraction of the underlying game, in which a players' actions consist of choosing policies from the population rather than primitive game actions. A meta-game's empirical winrate matrix $\boldsymbol{W_\pi}$ for a given population $\boldsymbol{\pi}$ can be considered as a payoff matrix for a 2-player zero-sum game. It is possible to define an empirical winrate matrix over two (or more) populations $\boldsymbol{W_{\pi_1,\pi_2}}$, such that each player chooses agents from a different population. An *evaluation matrix* [6] is a metagame represented by an antisymmetric matrix $\boldsymbol{A}$. One can turn an empirical winrate matrix $\boldsymbol{W}$ into an antisymmetric matrix by performing the element-wise operation $a_{i,j} = w_{i,j} - 1/2$, shifting the range of each entry from $[0, 1]$ to $[-1/2, 1/2]$. Symmetrical 2-player zero-sum games represented by an antisymmetric matrix $\boldsymbol{A}$ feature a unique maxent Nash [6], a fact we will use in Section V.

Finally, the *relative population performance* [6] is a population-level measure of performance. Given two populations $\boldsymbol{\pi_1}, \boldsymbol{\pi_2}$, it yields a single scalar value comparing the performance of $\boldsymbol{\pi_1}$ against $\boldsymbol{\pi_2}$. It is computed by generating an evaluation matrix for both populations $A_{\boldsymbol{\pi_1},\boldsymbol{\pi_2}}$ which is then treated as a 2-player zero-sum game. A Nash equilibrium is then computed $(\boldsymbol{n_{\pi_1}}, \boldsymbol{n_{\pi_2}})$ for the zero-sum game defined by $A_{\boldsymbol{\pi_1},\boldsymbol{\pi_2}}$. The relative population performance is the value v for the meta-player 1: $v = \boldsymbol{n_{\pi_1}} \cdot A_{\boldsymbol{\pi_1},\boldsymbol{\pi_2}} \cdot \boldsymbol{n_{\pi_2}}^T$. A positive v indicates that $\boldsymbol{\pi_1}$ wins on average against population $\boldsymbol{\pi_2}$, with the opposite being true if v is negative, and $v = 0$ indicates both populations are equivalent. In later sections, we scale this metric to lay between -1 ($\boldsymbol{\pi_2}$ dominates $\boldsymbol{\pi_1}$) and 1 (vice versa).

B. Multiagent Reinforcement Learning

Let E represent a multiagent system with n agents and a reward discount factor γ. This environment E features a state-space S, a joint observation space $O = O_1 \times \cdots \times O_n$, and a joint action space $A = A_1 \times \cdots \times A_n$, where O_i and A_i represent the observation and action space for the ith agent respectively. Let the (potentially stochastic) mapping from observations to actions $\pi_i : O_i \to A_i$ represent the policy for the ith agent, and $\boldsymbol{\pi} = [\pi_1, \ldots, \pi_n]$ the joint policy vector, containing the policy for *all* agents in E. The joint policy vector $\boldsymbol{\pi}$ can also be regarded as a distribution over the joint action space conditioned on the joint observation space $\boldsymbol{\pi} : O \to A$. Let $\Pi = \Pi_1 \times \cdots \times \Pi_n$ be the joint policy space, where Π_i is the policy space for agent i. As before, let Π_{-i} denote the joint policy space for all agents except agent i.

The solution to this environment E for an agent i is to compute a policy (π_i^*) which maximizes its expected reward obtained when acting in an environment across the *entire* set of all possible other policies Π_{-i} in the environment

$$\pi_i^* = \underset{\pi_i \in \Pi_i}{\arg\max} \int_{\boldsymbol{\pi_{-i}} \subseteq \Pi_{-i}} \mathbb{E}_{\boldsymbol{a_t} \sim \boldsymbol{\pi}; s_{t+1}, r_t \sim P(s_t, \boldsymbol{a_t})} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right]. \tag{1}$$

An iteration, or episode, of the classical MARL loop goes as follows. The environment presents all agents with a vector containing all individual agent observations $\boldsymbol{o_t} = [o_t^1, \ldots, o_t^n]$ based on its state s_t. The vector containing the actions of all agents is sampled from the joint policy vector $\boldsymbol{a_t} \sim \boldsymbol{\pi}(\boldsymbol{o_t})$. The environment then executes the action vector $\boldsymbol{a_t}$, transitioning to a new state s_{t+1} and yielding both a new observation $\boldsymbol{o_{t+1}}$ and a reward vector $\boldsymbol{r_t}$ containing an observation and reward for each agent. This loop is repeated until a terminal state is reached, after which a new episode begins.

C. Transitivity and Cycles in 2-Player Zero-Sum Games

Balduzzi *et al.* [6] show that 2-player zero sum games can be broken down into transitive components (skill levels) and cyclic components (viable strategies at a given skill level). By analyzing the effect of SP algorithms on a heavily cyclic game, we can observe how an algorithm deals with the obstacle of catastrophic forgetting, learning to beat new policies by unnecessarily deteriorating its performance against known policies. Analyzing SP algorithms on a heavily transitive games gives us information about the speed at which these algorithms increase the skill of the learning agent [22].

Algorithm 1: (POSG) RL Loop With Self-Play.

Input: *Environment*: $(S, A, O, \mathcal{P}(\cdot,\cdot|\cdot,\cdot), \mathcal{R}(\cdot,\cdot), \rho_0)$
Input: *Self-Play Scheme*: $(\Omega(\cdot|\cdot,\cdot), G(\cdot|\cdot,\cdot))$
Input: *Policy to be trained*: $\pi \in \Pi_i$

$\boldsymbol{\pi^o} = \{\pi\}$; `// Menagerie initialization`
for $e = 0, 1, 2, \ldots$ **do**
  $\boldsymbol{\pi'} \sim \Omega(\boldsymbol{\pi^o}, \pi)$; `// Sample from menagerie`
  $\boldsymbol{\pi} = \boldsymbol{\pi'} \cup \{\pi\}$;
  $s_0, \boldsymbol{o_0} \sim \rho_0$;
  for $t = 0, \ldots,$ *termination* **do**
    $\boldsymbol{a_t} \sim \boldsymbol{\pi}(\boldsymbol{o_t})$;
    $s_{t+1}, \boldsymbol{o_{t+1}} \sim P(s_t, \boldsymbol{a_t})$;
    $\boldsymbol{r_t} \sim \boldsymbol{R}(s_t, \boldsymbol{a_t})$;
    $t \leftarrow t + 1$;
  end
  $\pi \leftarrow update(\pi)$;
  $\boldsymbol{\pi^o} \sim G(\boldsymbol{\pi^o}, \pi)$; `// Curate menagerie`
end
return π;

IV. Generalized Self-Play Framework

Here we present the mathematical formulation, and required assumptions, for a formal framework which encapsulates the notion of self-play in the context of MARL. It allows for the creation and comparison of existing and future SP algorithms.

Self-play training schemes can be conceived as modules which extend the MARL loop by introducing a functionality prior to, and after, every episode. Let π be the only policy being trained throughout the MARL loop. An SP scheme envelops the MARL loop by first deciding which policies $\boldsymbol{\pi'}$, taken from a set of *fixed* policies $\boldsymbol{\pi'} \subseteq \boldsymbol{\pi^o}$, will define the agents' behavior for the next episode. This *excludes* the agent whose behavior is defined by π. Once the episode ends, a function G decides whether or not the (possibly updated) policy π will be introduced in the pool of available policies $\boldsymbol{\pi^o}$. This intuition is formally captured in Algorithm 1, which presents an SP scheme inside a Partially Observable Stochastic Game (POSG) loop. Algorithm 1 defines a *n-player*, *general-sum*, *partially observable* environment. The steps belonging to the SP scheme are highlighted in orange.

The attentive reader will realize that Algorithm 1 only updates policy π once at the end of each episode, and this does not allow for temporal difference (TD) learning [1] algorithms to be trained, as these rely on updating policy π after each environment step. One can readily move the *update* functionality to be called on the inner loop (during gameplay). We have decided on keeping it on the outer loop for readability purposes.

A. Framework Definition

We define an SP module or training scheme by formalizing the notions of the *menagerie* $\boldsymbol{\pi^o}$, the *policy sampling distribution* Ω, and the *gating function* G. Specified by the tuple $< \Omega(\cdot|\cdot,\cdot), G(\cdot|\cdot,\cdot) >$:

1) $\boldsymbol{\pi^o} \subseteq \Pi_i$; the *menagerie*. A set of policies from which agents' behavior will be sampled. This set always includes the currently training policy π. A constraint is placed over $\boldsymbol{\pi^o}$. All of its elements must be derived, at least indirectly, from π, the policy being trained. Hence, all policies in the menagerie are elements of π's policy space. The menagerie *can* change as training progresses by the curator function described below.
2) $\Omega(\boldsymbol{\pi'} \in \Pi_{-i} | \boldsymbol{\pi^o} \subseteq \Pi_i, \pi \in \Pi_i) \in [0, 1]$, where $\boldsymbol{\pi'} \subseteq \boldsymbol{\pi^o}$; the *policy sampling distribution*. A probability distribution over the menagerie $\boldsymbol{\pi^o}$, the set of available policies. It is conditioned on the menagerie $\boldsymbol{\pi^o}$ and the current policy π being trained. It chooses which policies, apart from π, will inhabit the environment's agents.
3) $G(\boldsymbol{\pi^{o\prime}} \subseteq \Pi_i | \boldsymbol{\pi^o} \subseteq \Pi_i, \pi \in \Pi_i) \in [0, 1]$; The *curator* or gating function, of the menagerie. A possibly stochastic function whose parameters are the current training policy π and a menagerie $\boldsymbol{\pi^o}$. The curator serves two purposes, which complex curators could break into two functions.
   a) G decides if the current policy π will be introduced in the menagerie.
   b) G decides which policies in the menagerie, $\pi \in \boldsymbol{\pi^o}$, will be discarded from the menagerie.

The curator bears resemblance with the notion of Hall of Fame from evolutionary algorithms [23], as Hall of Fame algorithms also consider the problem of curating a policy set over time.

B. Assumptions

Our SP framework explicitly assumes the following.

Assumption 1.1: The policies present in the environment can either be exact copies of the policy being trained, or policies derived indirectly from it, taken from the menagerie.

Assumption 1.2: Prior, during, and after a training episode, the SP module has access to the agents' policy representations,[2] allowing any-time read and write rights for all policies.

The definitions above capture the minimal structure of all SP training schemes. However, it is possible to condition both the policy sampling distribution Ω and curator G on any other variables. For instance, it could be interesting to define an SP algorithm whose components are conditioned on episode trajectories, which has proved useful in RL research [24], and is required for policy gradient algorithms [25].

Our SP framework does not make any assumptions on the environment with which the policies interact.

Assumption 2: There exists a set of policies, $\boldsymbol{\pi} \subseteq \Pi$, significantly smaller than the entire original policy space, $|\boldsymbol{\pi}| \ll |\Pi|$, which we can use as a proxy for Π in equation 1. If so, the integration over the policy space becomes computationally tractable, making (1) computationally solvable.

The policy sampling distribution Ω and the gating function G are tools by which a menagerie $\boldsymbol{\pi^o}$ can be computed and curated over time. Self-play can be conceived as a bottom-up approach toward computing a set of policies, $\boldsymbol{\pi^o}$, to be used as a proxy for the entire policy space Π in (1). The obvious fact that an agent cannot act according to a policy outside its policy space means that a menagerie can only contain policies of a single policy space. Consequently, for environments with disjoint policy spaces, SP may be unable to serve as an approximate solution to (1).

Balduzzi *et al.* [6] introduce the notion of the *gamescape*, a polytope which geometrically encodes interactions between agents for zero-sum games. They derive a set of algorithms whose goal is to grow and curate an approximation to this polytope. We draw parallels between their work and the idea of using SP algorithms to compute a proxy for a target policy space.

V. Self-Play Algorithms

We demonstrate the generalizing capabilities of our framework by presenting four prevalent SP schemes from MARL literature. Let π be a policy being trained, and $\boldsymbol{\pi^o}$ a menagerie.

1) Naive Self-Play: The is the oldest and simplest SP algorithm, originating in [5]. The premise is that every agent in the environment is populated with the latest version of the policy being trained. All agents share the same behavior. To capture this, the policy sampling distribution Ω puts all probability weight to the latest π.

$$\Omega(\boldsymbol{\pi'}|\boldsymbol{\pi^o}, \pi) = \begin{cases} 1 & \forall \pi' \in \boldsymbol{\pi'}: \ \pi' == \pi \\ 0 & \text{otherwise.} \end{cases}$$

In this degenerate scenario, the gating function G always deterministically inserts the latest version of the training policy into the menagerie, discarding the previous menagerie entirely.

$$G(\boldsymbol{\pi^o}, \pi) = \{\pi\}.$$

[2] If the policies are being represented by a neural network. Access to the policy representation means access to the neural network topology and weights.

2) δ-Uniform Self-Play: Introduced by [13] and mentioned in Section II. This SP scheme treats the menagerie as a set of "historical" policies. The authors wanted to create an SP scheme that ensured continual learning by training a policy which could consistently beat random older versions of itself.

Let $M = |\boldsymbol{\pi^o}|$ be the size of the menagerie, and let $\delta \in [0, 1]$ denote the percentage threshold on the oldest policy to be considered as a potential candidate to be sampled from $\boldsymbol{\pi^o}$ by Ω. Thus, $\delta = 0$ corresponds to all policies in the menagerie being considered as candidates, and $\delta = 1$ only allows the last policy introduced in the menagerie to be sampled by Ω. After computing the set of candidate policies following this criteria, the authors use a uniform distribution to sample from it.

$$\Omega(\boldsymbol{\pi'}|\boldsymbol{\pi^o}, \pi) = Uniform(\delta M, M).$$

The gating function G used in δ-Uniform self-play is fully inclusive and deterministic. After every episode, it always inserts the training policy into the menagerie.

$$G(\boldsymbol{\pi^o}, \pi) = \boldsymbol{\pi^o} \cup \{\pi\}.$$

3) Population-Based Training Self-Play: As introduced in [20], population-based training SP is a parallel SP algorithm influenced by evolutionary algorithms. Each agent is independently learning on their own SP-augmented MARL loop. The menagerie, initialized with a population of random policies, is shared among all learning agents. The menagerie is treated as the population of an evolutionary algorithm.

The policy sampling distribution chooses opponents from the menagerie which are similar in skill to the currently training agent, where agent skill is measured by Elo ratings.

The gating function is analogous to the selection, crossover, and mutation phases of an evolutionary algorithm. It modifies and changes the menagerie by dropping low-performing agents and introducing evolved versions of the existing population.

4) Policy-Spaced Response Oracles (PSRO): A family of algorithms introduced in [4]. Such algorithms maintain an empirical winrate matrix $\boldsymbol{W_{\pi^o}}$ generated from a menagerie $\boldsymbol{\pi^o}$, and are parameterized via the choice of two functions.

- $\mathcal{M}(\boldsymbol{W_{\pi^o}} \in \mathbb{R}^{|\boldsymbol{\pi^o}| \times |\boldsymbol{\pi^o}|}) \in \Delta(\boldsymbol{\pi^o})$. The meta-game solver, which takes a meta-game and outputs a "meta-game solution," a distribution over the policies of the menagerie.
- $\mathcal{O}(\pi \in \Pi, \boldsymbol{\pi'} \in \Delta(\boldsymbol{\pi^o})) \in \Pi$. The oracle, which takes a distribution over policies $\boldsymbol{\pi'}$, a starting policy π and derives a new policy π^* which performs better against $\boldsymbol{\pi'}$ than π.

The function of the meta-game solver $\mathcal{M}$ is captured by our policy sampling distribution Ω, as they both output a probability distribution over a set of policies, the menagerie. After the oracle computes a new policy, it is added to the meta-game, and the empirical winrate matrix $\boldsymbol{W_{\pi^o}}$ is updated via game simulations.

$\mathcal{M}$ operates on a meta-game generated by doing head-to-head matches between all policies in the menagerie, whereas a policy sampling distribution Ω operates directly on the menagerie. In this article, we use $\mathcal{M}$ = maxent-Nash [26]. As stated in Section III, we can turn a winrate matrix into an antisymmetric evaluation matrix, which we know has a unique maxent-Nash.

This uniqueness feature is valuable for consistent interpretability. Other alternatives exist [7], [21].

$$\Omega(\boldsymbol{\pi}'|\boldsymbol{\pi}^{\boldsymbol{o}}, \pi) = \mathcal{M}(\textit{meta-game}(\boldsymbol{\pi}')).$$

The functionality of the oracle can be anything that generates a new policy, such as an RL algorithm or evolutionary algorithm among other options. Upon completion of the oracle function, a new policy is added to the meta-game. To this extent, the oracle $\mathcal{O}$ and our curator function G are analogous in so far as both functions decide when a policy is introduced in the menagerie. The curator has the advantage of removing policies from the menagerie.

The extent to which PSRO and our framework overlap is left for future work.

VI. Novel Contributions

In this section, we present a novel policy sampling distribution that alleviates on the shortcomings of the δ-uniform sampling distribution and a novel qualitative metric for the efficiency of the menagerie when it comes to using it as a proxy to the whole policy space. This shows how minimal incremental changes to existing methods, within the context of a general framework, can lead to improvements.

1) δ-Limit Uniform Policy Sampling Distribution: In supervised learning approaches, training datasets are fixed before training commences. This yields a stationary distribution from which training examples are drawn. RL suffers from sequential and correlated data collection during training, rendering a nonstationary distribution over training samples.

We analyze a property of the δ-Uniform SP algorithm. As stated earlier, it aims to generate an agent which can defeat *random* past versions of itself. However, this is affected by the sequential data collection curse of RL methods. By sampling uniformly at random from a menagerie, we observe a bias of the policies sampled from Ω towards earlier policies. Intuitively, earlier policies are sampled more often by virtue of being electable to sampling more times than recently added policies.

Computing a policy which generalizes against a broad set of policies is desirable. However, we worry that by sampling earlier policies too often the learning policy will be biased towards interacting with, often random, initial agents. This worry is furthered by empirical evidence stating that, in certain board games, the quality of the fixed policies being used during training is directly proportional to potential quality of the policy being trained [3].

With this in mind, we present a novel policy sampling distribution, named δ-Limit Uniform, that gives increased probability to later policies. This is an attempt to amend the aforementioned bias induced by δ-Uniform. Fig. 2 shows the histograms of the number of samples per policy for both $\delta = 0$-Uniform and $\delta = 0$-Limit Uniform, clearly showing how the δ-Limit Uniform distribution avoids biasing toward earlier policies.

Let $|\boldsymbol{\pi}_{\boldsymbol{n}}^{\boldsymbol{o}}|$ be the size of the menagerie at the beginning of the nth episode. π_e is the eth policy to have entered the menagerie (asserting $e \leq n$). The logit probability ρ_e^n and normalized probability p_e^n of sampling π_e for the nth SP episode are computed

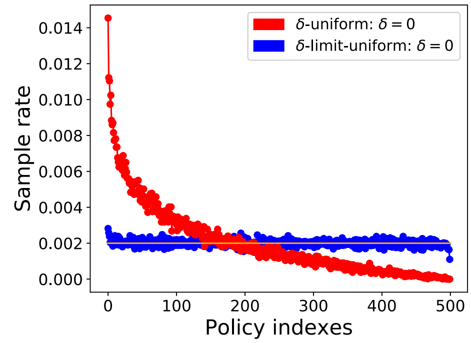


Fig. 2. Histograms of sample rates for policies inside a menagerie for two sample training runs. The horizontal orange line represents a $Uniform(0500)$ distribution.

as

$$\rho_e^n = \frac{1}{|\boldsymbol{\pi}_{\boldsymbol{n}}^{\boldsymbol{o}}|(|\boldsymbol{\pi}_{\boldsymbol{n}}^{\boldsymbol{o}}| - e)^2} \tag{2}$$

$$p_e^n = \frac{\rho_e^n}{\sum_{i=0}^{|\boldsymbol{\pi}_{\boldsymbol{n}}^{\boldsymbol{o}}|} \rho_i^n}. \tag{3}$$

2) Qualitative Metric for the Menagerie's Efficiency: A visual metric, aimed at understanding how well a menagerie approximates the entire policy space (shown in Fig. 3). Policies can be characterized by the behaviors/state trajectories they produce when acting in the multiagent environment. Thus, assessing the span of the state trajectories induced by the SP training enables an assessment of the span of the policies living inside the menagerie, which is what we mean by assessing how well a menagerie approximates the whole policy space. This visual display comes from a 2D embedding of the state trajectories experienced by an agent during each training episode. We use t-SNE [27] to project the multidimensional, environment-specific representation of state trajectories unto a 2D space. Other dimensionality reduction algorithms can be used. We propose two visual cues.

- *Density Heightmap*: Visualization of the density function yielded by the embedded state trajectories, computed via a kernel density estimation method. Intuitively, it gives insight toward understanding where, inside the embedded state trajectory space, the agent has spent most time on during training.
- *Time Window-Averaged SP-Induced Trajectories*: Visualization of the temporal evolution of the average embedded trajectory/episode for an agent during training. Computed by uniformly dividing the time-sorted embedded trajectories in buckets, with the window-averaged trajectory being the median trajectory, computed in the 2D embedding space, of each bucket. Intuitively, it displays which parts of the embedded trajectory space the agent has traversed throughout training. This cue can be used to visually assess to what extent an agent is prone to revisit some areas of the trajectory space, which can help identify catastrophic forgetting and cyclic policy evolutions.

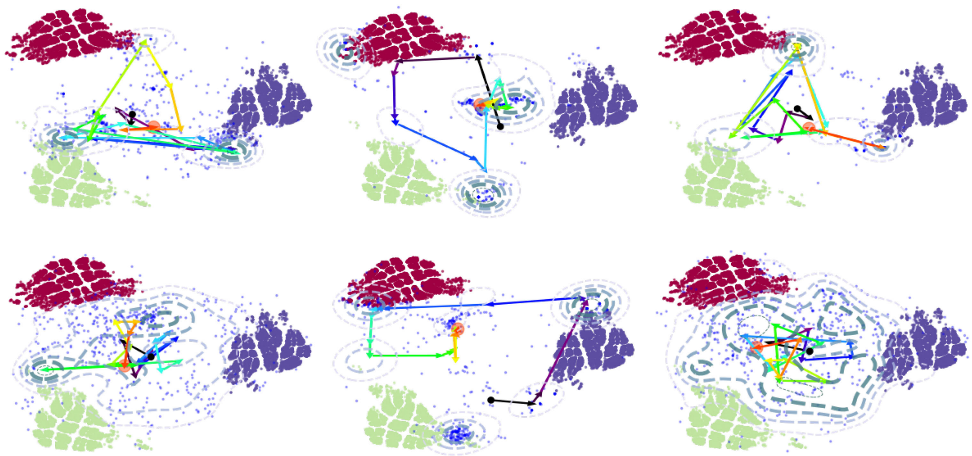


Fig. 3. Density heightmap and time window-averaged SP-induced of episode trajectories in a 2D t-SNE state trajectory embedding space. Top-Left: Naive SP with MLP-PPO. Bottom-Left: Naive SP with RNN-PPO. Top-Center: $\delta = 0$-Uniform SP with MLP-PPO. Bottom-Centre $\delta = 0$-Uniform SP with RNN-PPO. Top-Right: $\delta = 0$-Limit Uniform SP with MLP-PPO. Bottom-Right: $\delta = 0$-Limit Uniform SP with RNN-PPO. RirRPS environment, $1e4$ SP training episodes. Green-, purple-, and red-colored clusters are embeddings of state trajectories resulting of pitting, respectively, RockAgent, PaperAgent, and ScissorsAgent against a RandomAgent. The scattered blue dots represent the individual projection of each one of the $10e4$ trajectories. Their density heightmaps are represented through dashed contours. The time-sorted training trajectories experienced by the SP agents were divided into 20 time-windows, and a centroid (median trajectory) was computed for each. Consecutive centroids have been linked by arrows, creating the time window-averaged SP-induced episode trajectories. Starting at the black dot, their progression is highlighted via the rainbow color transitions. For extra details, see the Appendix.

t-SNE-projected representations vary depending on the data used as input. For our purposes, it means that if we were to separately embed two sets of different state trajectories, we might not be able to meaningfully compare both separate embeddings. We tackle this problem with two measures.

1) We compute a basis of possible state trajectories using some environment-specific heuristics that enable the basis to span over most of the whole state trajectory space. The number of basis state trajectories computed is of the same order as the number of state trajectories generated during training.
2) When comparing two or more sets of state trajectories generated by different algorithms, we compute the embeddings of each algorithm-induced state trajectories all at once via an *aggregated* set of state trajectories. Thus, it allows for meaningful comparisons across state trajectory embeddings from different algorithms.

This metric does not come without issues. It is only valuable providing we can label some subsets of the embedding space with high-level understanding of what is happening throughout the state trajectories. Similarly, computing a basis of trajectories is a nontrivial, environment-specific task. Furthermore, stochasticity coming from environment dynamics could make these visualizations harder to understand. Thus, this metric is most suitable for simple and deterministic environments.

VII. Experimental Details

A. Experiment Description

1) Environments

a) Repeated imperfect recall rock paper scissors: (RirRPS) introduced in [28]. A repeated, imperfect information, simultaneous version of Rock Paper Scissors. The agent which obtains the highest cumulative reward by the end of the last repetition is considered the winner. We use 10 repetitions, with a recall of the last three joint actions. Ties are broken uniformly at random. RirRPS is a highly cyclic game, with a small amount of transitive skill coming from the *repeated* aspect of the game, which introduces exploitability that increases with the number of repetitions. We also chose this environment because it lends itself to be visualized by the qualitative metrics introduced in Section VI.

b) Connect Four: A sequential game with a high degree of transitivity as empirically demonstrated in [22]. Connect Four is not a symmetrical game; when training or benchmarking agents, we randomly assign agent positions to enforce this symmetry.

2) Evaluation Metrics

a) Winrate matrices: SP algorithms train/modify a policy π over time. We can consider an SP scheme sp as a generative process, which we can query at any time t to obtain the latest version of π being trained by sp, $\pi_t \sim sp$. This is analogous to creating checkpoints in training at which to freeze a copy of the policy π being trained. We only freeze π_t and not the menagerie $\boldsymbol{\pi_t^o}$. Thus, we can generate a population which represents the evolution of the policy training under an SP algorithm over time, $\boldsymbol{\pi_{sp}} = [\pi_{t_0}, \pi_{t_1}, \dots]$. By examining the evaluation matrix generated from this population, $\boldsymbol{W_{\pi_{sp}}}$, we can quantitatively examine if different SP algorithms 1) suffer from catastrophic forgetting and 2) the speed at which the learning agent is improving upon previous versions.

b) Evolution of relative population performance: As introduced in Section III, we shall use the relative population performance as a direct measure of the relative quality between the populations spawned by two different SP algorithms. We are interested in how this relative performance evolves over time. Below we describe the algorithm to obtain such evolution. Given a set of SP training algorithms SP:

1) for each $sp \in SP$, sample a population $\boldsymbol{\pi_{sp}}$ of n agents;
2) for each population pair $(\boldsymbol{\pi_{sp_1}}, \boldsymbol{\pi_{sp_2}})$, $sp_1, sp_2 \in SP$, compute an evaluation matrix $\boldsymbol{A}_{\boldsymbol{\pi_{sp_1}}, \boldsymbol{\pi_{sp_2}}}$ between both populations;
3) compute $\boldsymbol{A_{sub}} = \{\boldsymbol{A_{1 \dots i \times 1 \dots i}} : i \in \{n\}\}$, which represents all submatrices of $A_{\boldsymbol{\pi_{sp_1}}, \boldsymbol{\pi_{sp_2}}}$;
4) compute the evolution of relative population performance associated with each submatrix $\boldsymbol{A_i} \in \boldsymbol{A_{sub}}$, $\boldsymbol{v_{sp_1, sp_2}} = [v_{\boldsymbol{A_i}}] \in \mathbb{R}^n$.

Evaluation matrices are expensive to compute: $O(n^2)$, where n is the population size. There is current research on reducing the computational load of generating evaluation matrices [29]. The procedure outlined above uses a single evaluation matrix to compute the relative population performances for all submatrices, meaning that we can recycle the empirical winrate matrix used to generate the evaluation matrices. Throughout this article, to compute the winrate for an entry $w_{i,j}$ in an empirical winrate matrix $\boldsymbol{W}$, we use 30 simulations.

3) Algorithmic choices

For our qualitative studies, we used proximal policy optimization (PPO) [30], where the underlying policy is represented by either by a feedforward neural network (MLP-PPO) or a recurrent architecture (RNN-PPO). For our quantitative studies, we only use MLP-PPO. See the Appendix for extra details.

4) Self-Play choices

We train a PPO agent on an SP-extended MARL loop as shown in Algorithm 1.

- Naive SP.
- δ-Uniform and δ-Limit Uniform, where the value of δ is specified each time.
- PSRO ($\mathcal{M} =$ maxent-Nash, $\mathcal{O} =$ best response). Such oracle is governed by two hyperparameters, which play a role in determining whether the training agent has converged to a best response: 1) The winrate $w \in [0, 1]$ at which it is considered that the current agent has converged and 2) the number of episodes n_{matches} that will be used to compute the aforementioned winrate. For all experiments, we used $w = 72\%$, $n_{\text{matches}} = 50$.

For all SP training schemes, the initial menagerie contains a copy of the initial policy, with randomly initialized weights.

VIII. Results and Discussion

A. Qualitative Analysis

Fig. 3 shows the 2D t-SNE state trajectory embeddings for Naive, $\delta = 0$-Uniform, and $\delta = 0$-Limit Uniform SP schemes, for both agent architectures introduced in the previous section. Each training session lasted for a $1e4$ episodes on the RirRPS environment.

We begin by observing $\delta = 0$-Uniform (Fig. 3, middle column). $\delta = 0$-Uniform's time window-averaged SP episode trajectories visit each fixed agent clusters one by one. Indeed, after behaving like a RockAgent (green cluster), the trained policy starts to behave like a PaperAgent (purple cluster) as the RockAgent-behaving policies that have entered in the menagerie progressively starts to be sampled as opponents. Both the MLP-PPO- and RNN-PPO-equipped agents exhibit that cyclic and ordered exploration of the embedding space. In contrast, agents trained using naive SP and $\delta = 0$-Limit Uniform exhibit erratic exploration of the embedded space, as depicted by the sharp turns in their time window-averaged trajectories.

Comparing the top and bottom rows of Fig. 3, with a focus on $\delta = 0$-Limit Uniform and Naive SPs, the Density Heightmaps of RNN-PPO seem to be made of plateaus whereas the ones of MLP-PPO are made of sharp peaks, indicating that recurrent policies seem to further spread the menagerie over the whole policy space to some greater extent compared to feedforward policies.

Based on this qualitative projections, we have gathered evidence that the choice of SP training scheme does affect the way that a learning agent explores the joint policy space.

B. Quantitative Analysis

The results from Fig. 4 are metrics gathered on a representative training run. The agents used to generate Fig. 3 are different from those used to generate Fig. 4, thus we advice caution when comparing both figures. For details on these differences, see the Appendix.

Each row i of winrate matrix $\boldsymbol{W}$ represents the winrates of policy at checkpoint i against all other policies generated during training. Thus, for any given row i, the entries left of the diagonal ($w_{i,j} \forall j < i$) indicate winrates against policies from earlier checkpoints in training, or older policies. Conversely, entries right of the diagonal ($w_{i,j} \forall j > i$) denote winrates of policy i against later checkpoints, or newer policies. Diagonal entries represent the winrate of a policy against itself, which we trivially set at 50%. An ideal training scheme which would always compute monotonically better policies as training progressed would yield a winrate matrix where the lower triangular indices would show positive winrates (higher than 50%) and the upper triangular would show negative winrates (lower than 50%). In other words, a policy would always win against previous versions of itself, and lose against newer ones. Similarly, from an ideal winrate matrix, we would get a Nash support which posits little to no probability mass unto the earlier policies, and increases for later policies.

C. Winrate Matrices Analysis

1) Naive and $\delta = 0.0$-Limit: We turn our focus to the winrate matrices from Fig. 4. As discussed, Naive SP uses as opponent an identical version of the policy being trained, and thus the underlying RL algorithm tries to compute a best response against itself. This is clearly manifested in the winrate matrix for RirRPS in Fig. 4(1.A). The entries just left of the diagonal show positive winrates, and those just right of the diagonal show negative winrates. This means that the training policy learns how to beat the last version of itself. Take any row from Fig. 4(1.A), most checkpoints obtained during naive SP training cycle between losing and winning against previous and future checkpoints. These are indications that as the training policy progresses through

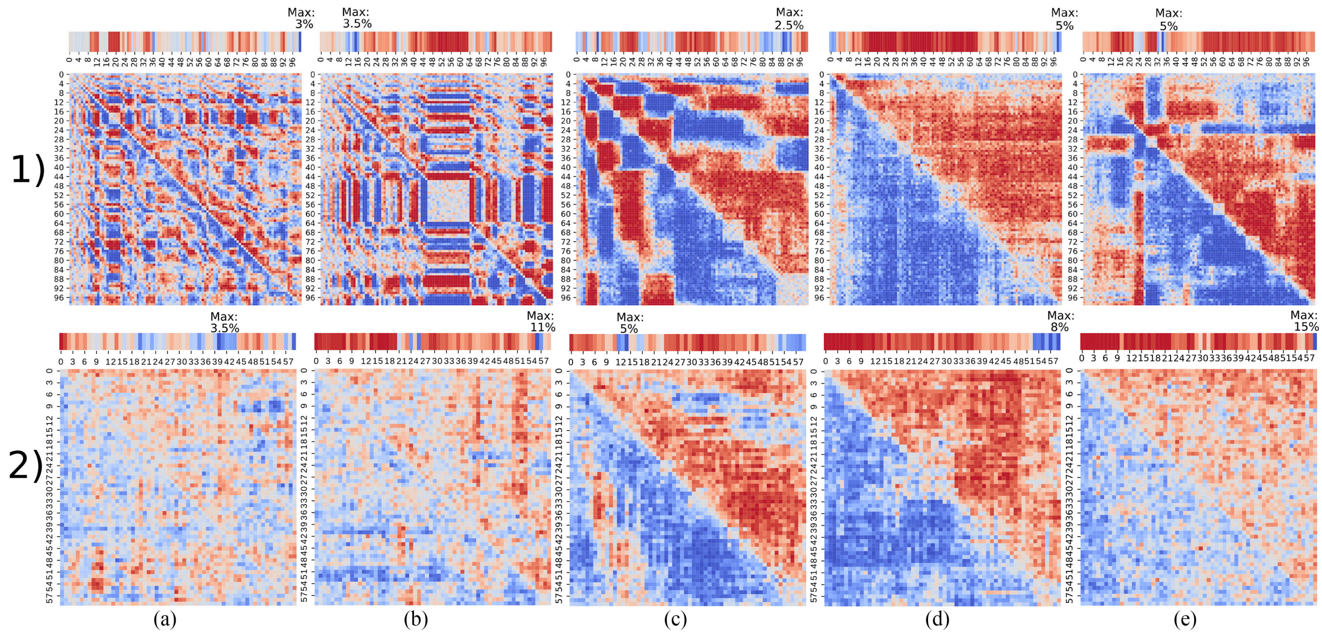


Fig. 4. Empirical winrate matrices showing the evolution of five policies where each one is being trained via a different SP algorithms. Row 1) RirRPS. Row 2) Connect Four. For every SP training process, we sample a policy after every one policy update on RirRPS (for greater granularity) and two updates for Connect Four (for greater variety), totalling 100 checkpoints for RirRPS and 60 for Connect Four. Treating each matrix as the payoff matrix for a symmetrical 2-player zero-sum game, we present on top of of each matrix the support received by each policy on the Nash equilibrium of such game. This support gives a measure of quality of each individual policy with respect to the other policies in the population. Blue/red indicates positive/negative winrates for column player. (a) Naive. (b) $\delta = 0$-limit uniform. (c) $\delta = 0.5$-uniform. (d) $\delta = 0$-uniform. (e) PSRO.

training, it does not learn to exploit previously encountered agents. This is further evidenced by the support under Nash from Fig. 4(1.A), where under Nash equilibrium, many policies share the highest amount support (around 3%). We observed similar cyclic behavior in $\delta = 0$-Limit Uniform [in Fig. 4(1.B)] and in $\delta = 0.5$-Limit Uniform (not shown), which may entail that δ-Limit Uniform SPs overcorrect the bias toward earlier policies, matching our observations on the previous qualitative analysis. Looking at Naive SP for Connect Four, the winrate matrix of Fig. 4(2.A), with most winrates laying around 50%, show that policy improvement during training is slow. There *is* a small gradual internal improvement, as the lower triangular matrix features an average entry of 53%. $\delta = 0$-Limit Uniform yields very similar results, as seen in Fig. 4(1.B/2.B). Combined with the similarities with Naive SP found in our qualitative analysis, we believe that our proposed δ-Uniform training schemes are indeed overcorrecting the bias toward sampling earlier policies and putting too much emphasis on sampling later ones, leading to behaviors that resemble Naive SP.

2) $\delta = 0.5$-Uniform: Fig. 4(1.C) shows the evolution of the policy training under $\delta = 0.5$-Uniform. This policy attempts to compute a best response against the later half of its history. Fig. 4 features more pronounced winrates (entries are either closer to 100% or 0% winrate) than the equivalent winrate matrix from $\delta = 0.0$-Uniform's Fig. 4(1.D). This is a result of $\delta = 0.5$-Uniform's menagerie being smaller than $\delta = 0$-Uniform's counterpart, which leads the training policy to overfit against the policies in the menagerie, which in turn allows earlier policies to exploit it. We see that, on average, for a given row i, the corresponding policy tends to win against policies $j \in [^i/_2, i-1]$. Interestingly, policies immediately outside the moving window determined by the choice of $\delta = 0.5$ feature a negative winrate, suggesting the training policy does not generalize to policies outside of the menagerie. This is because, as these policies fall out of the menagerie, there is no pressure coming from the optimization process to maximize the learning agent's performance against them. This causes the underlying RL algorithm to update the policy parameters in a way that could be detrimental with respect to the performance of the training agent against policies outside of the menagerie. We were surprised to find that same effect is also present Connect Four, as shown by the red band in the winrate matrix of Fig. 4(2.C). Theoretically, in purely transitive games, one does not need to maintain a large pool of varied policies to ensure skill improvement [6]. We propose two explanations. First, that this effect showcases the cyclic components of Connect Four, if Connect Four is understood as a functional form game [6]. Second, the usage of function approximators (the neural networks which represent the policy) are introducing nontransitivities into the learning process. Focusing on Fig. 4(2.C), this exploitability by earlier policies is present in the Nash support, where policies with index [12–19] add up to 27% of the support under Nash.

3) $\delta = 0$-Uniform: Fig. 4(1.D/2.D), whose underlying policy attempts a best response against its entire history, shows a close-to-ideal empirical winrate matrix insofar as any given policy beats most previous versions of itself and loses against later ones. Also, for any subgame of Fig. 4(1.D/2.D), the largest concentration of support under Nash consistently lays on the latest policies. This is specially the case for the case of Connect Four, where the last seven policies accrue 49% of the support under Nash. $\delta = 0$-Uniform does not exhibit a cyclic policy evolution. Instead, it tends toward generating monotonically better policies. Hence, we claim that $\delta = 0$-Uniform SP is an apt SP scheme for cyclic environments, given enough computational time.

4) PSRO: The winrate matrix for RirRPS, depicted in Fig. 4(1.E), does follow a positive trend, although less so than $\delta = 0$-Uniform, as checkpoints beyond the 37th lose against policies 20–26, which worsens as later policies are introduced. Interestingly, the policy featuring the largest support under Nash is the 34th checkpoint. This does *not* necessarily mean that all 66 checkpoints that came after it were weaker in comparison. The quality of a policy (in terms of support under Nash) can vary greatly when policies are added or dropped from the population. For instance, if we consider a subgame of Fig. 4(E) taking only the first 92 checkpoints, we would find that the 92nd policy features the largest support under Nash around 3%, yet it falls around 1% on the final winrate matrix [Fig. 4(1.E)]. For the game of Connect Four, in most subgames from Fig. 4(2.E), the policy with the largest support under Nash is always the last one, with the last one featuring 15%. Note how the average winrate on the lower triangular indices from Fig. 4(2.E) is not as high as Fig. 4(2.D). This could mislead the reader into thinking that $\delta = 0$-Uniform generates higher quality policies, but such conclusions should be drawn from metrics that make comparisons not on the internal policy progression, but rather on comparisons across SP schemes.

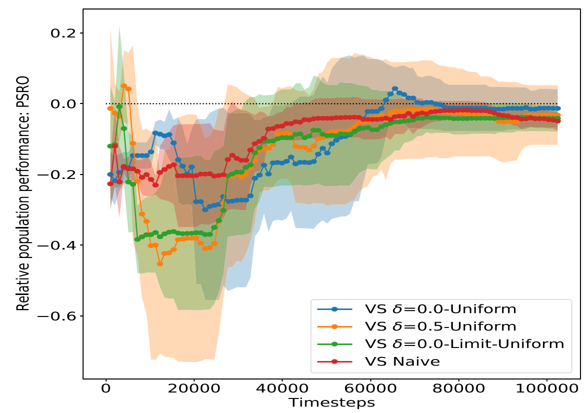
D. Relative Population Performances

We now use the relative population performance metric to make a quantitative comparison across different SP algorithms. Figs. 5(a) and (b), computed from the agents trained for Fig. 4, shows the evolution of the relative population performance between PSRO and the other four SP algorithms for RirRPS and Connect Four, respectively.

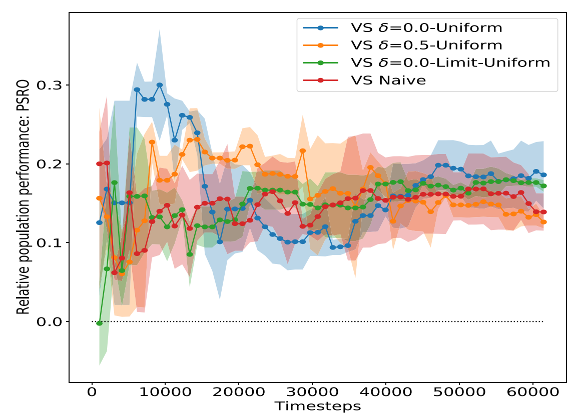
In Fig. 5(a), we notice that the relative population performance seems to converge near-zero for all SP algorithms. For RirRPS, this implies that the populations generated by all SP training processes are of similar quality, furthering the idea that in highly cyclic games, individual policy improvement is not meaningful, even when there is potential for exploitation due to repetitions[3] in RirRPS. However, we are surprised to find Naive SP performing better than $\delta = 0$-Uniform and PSRO, which is not obvious by just looking at the winrate matrices from Fig. 4(1). A possible reason is that the cyclic behavior of Naive SP quickly discovers how to play Rock, Paper and Scissors, which are enough to generate a Nash equilibrium. Conversely, an SP algorithm that scarcely introduces a new policy into its menagerie, such as PSRO, slows down the variety of trajectories experienced by the learning agent, taking longer to explore the policy space.

Fig. 5(b) showcases how PSRO is better suited for transitive environments, as it shows a positive relative population performance versus all other SP algorithms at all points during training. This difference in performance stabilizes on an average of 0.15 across all SP algorithms. We hypothesize that this difference would increase over time, meaning that the rate of policy performance improvement induced by PSRO is higher. This would require experiments with longer training times. However, with our computational budget of around 60 K environment timesteps, we can say that $\delta = 0.5$-Uniform and Naive SP were the most promising training schemes behind PSRO (lower relative performances against PSRO means that they were closer to being as strong as PSRO).

[3]By repetitions, we mean the multiple rounds of RPS within the game of RirRPS.



(a)



(b)

Fig. 5. Evolution of relative population performances of PSRO vs. all other SP schemes. Positive values indicate that PSRO performs better against an SP scheme, and negative values indicate the opposite. This experiment was repeated three times to obtain an estimate of the standard variation (shaded regions). (a) RirRPS. PSRO features negative relative population performances against all other SP algorithms, meaning that it trained a weaker sequence of policies. (b) Connect Four. PSRO outperforms all other SP schemes.

IX. Conclusion

Building on our original work [28], this article presented a general framework in which SP training schemes are defined. This is done by formalizing the notion of a menagerie, a policy sampling distribution and a curator (gating) function. This framework is framed as a theoretical approximation to a solution concept in MARL, under stated assumptions. The framework's generalizing capabilities have been showcased by capturing existing SP algorithms within it. We have also identified shortcomings of some of the captured methods, and have proposed methods which could potentially overcome the said issues. Through a qualitative study, we have showcased that, on a simple environment, different SP algorithms differ in how the joint policy space is explored. We have also carried out a quantitative analysis on 1) the evolution of policies being trained under different SP algorithms to discover cyclic policy evolutions and 2) the relative performance between various SP algorithms, on both a highly cyclic and highly transitive environment.

Future work will study other possibilities presented within the expressive capabilities of our SP framework. For instance, there is no research exploring which policy sampling distribution works best for different types of environments. Furthermore, it may even be possible to *learn* a policy sampling distribution or curator during training using meta-RL.

Acknowledgment

The authors would like to thank Jayesh K. Gupta for his insightful conversations and work on Nash averaging.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[2] C. Wen, X. Yao, Y. Wang, and X. Tan, "Smix (λ): Enhancing centralized value functions for cooperative multi-agent reinforcement learning," in *Proc. Conf. Artif. Intell.*, 2020, pp. 7301–7308.

[3] M. Van Der Ree and M. Wiering, "Reinforcement learning in the game of Othello: Learning against a fixed opponent and learning from self-play," *Proc. IEEE Symp. Adaptive Dyn. Program. Reinforcement Learn.*, 2013, pp. 108–115.

[4] M. Lanctot *et al.*, "A unified game-theoretic approach to multiagent reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 4190–4203.

[5] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM J. Res. Develop.*, vol. 3, no. 3, pp. 210–229, 1959.

[6] D. Balduzzi *et al.*, "Open-ended learning in symmetric zero-sum games," *CORR*, vol. abs/1901.08106, 2019. [Online]. Available: http://arxiv.org/abs/1901.08106

[7] S. Omidshafiei *et al.*, "α-rank: Multi-agent evaluation by evolution," *Sci. Rep.*, vol. 9, no. 1, pp. 1–29, 2019.

[8] G. Tesauro, "Temporal difference learning and TD-gammon," *Commun. ACM*, vol. 38, no. 3, pp. 58–68, 1995.

[9] G. Tesauro, "Neurogammon: A neural-network backgammon program," in *Proc. Int. Joint Conf. Neural Netw.*, 1990, pp. 33–39.

[10] D. Silver *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[11] D. Silver *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[12] T. Anthony, Z. Tian, and D. Barber, "Thinking fast and slow with deep learning and tree search," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5360–5370.

[13] T. Bansal, J. Pachocki, S. Sidor, I. Sutskever, and I. Mordatch, "Emergent complexity via multi-agent competition," in *Proc. Int. Conf. Learn. Representations*, 2018.

[14] G. J. Laurent, *et al.* "The world of independent learners is not Markovian," *Int. J. Knowl.-Based Intell. Eng. Syst.*, vol. 15, no. 1, pp. 55–64, 2011.

[15] R. S. Sutton, A. Koop, and D. Silver, "On the role of tracking in stationary environments," in *Proc. 24th Int. Conf. Mach. Learn.*, 2007, pp. 871–878.

[16] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey." *J. Mach. Learn. Res.*, vol. 10, no. 7, pp. 1633–1685, 2009.

[17] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel, "Rl^2: Fast reinforcement learning via slow reinforcement learning," *CoRR*, vol. abs/1611.02779, 2016.

[18] C. Berner *et al.*, "Dota 2 with large scale deep reinforcement learning," *CoRR*, vol. abs/1912.06680, 2019. [Online]. Available: http://arxiv.org/abs/1912.06680

[19] O. Vinyals *et al.*, "Grandmaster level in starcraft II using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[20] M. Jaderberg *et al.*, "Human-level performance in 3D multiplayer games with population-based reinforcement learning," *Science*, vol. 364, no. 6443, pp. 859–865, 2019.

[21] P. Muller *et al.*, "A generalized training approach for multiagent learning," in *Proc. Int. Conf. Learn. Representations*, 2019.

[22] W. M. Czarnecki *et al.*, "Real world games look like spinning tops," *CoRR*, vol. abs/2004.09468, 2020.

[23] M. Nogueira, C. Cotta, and A. J. Fernández-Leiva, "An analysis of hall-of-fame strategies in competitive coevolutionary algorithms for self-learning in RTS games," in *Learning and Intelligent Optimization*, G. Nicosia and P. Pardalos, Eds. Berlin, Germany: Springer, 2013, pp. 174–188.

[24] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *Proc. 4th Int. Conf. Learn. Representations*, San Juan, Puerto Rico, May 2–4, 2016.

[25] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, no. 3, pp. 229–256, 1992.

[26] D. Balduzzi, K. Tuyls, J. Perolat, and T. Graepel, "Re-evaluating evaluation," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 3268–3279.

[27] L. V. D. Maaten and G. Hinton, "Visualizing data using t-SNE," *J. Mach. Learn. Res.*, vol. 9, no. Nov., pp. 2579–2605, 2008.

[28] D. Hernandez *et al.*, "A generalized framework for self-play training," in *Proc. IEEE Conf. Games.*, 2019, pp. 1–8.

[29] M. Rowland *et al.*, "Multiagent evaluation under incomplete information," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 12 270–12282.

[30] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017.