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*

*Department of Computer Science, University of York, UK. {dh1135, kyd500, pjy500, james.walker}@york.ac.uk

†Microsoft Research, Cambridge, UK. sam.devlin@microsoft.com

‡Institute of Analytics & Data Science, University of Essex, UK. ssamot@essex.ac.uk

Abstract—Throughout scientific history, overarching theoretical frameworks have allowed researchers to grow beyond personal intuitions and culturally biased theories. They allow to verify and replicate existing findings, and to link disconnected results. The notion of self-play, albeit often cited in multiagent Reinforcement Learning, has never been grounded in 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. On a simple environment, we qualitatively measure how well a subset of the captured self-play methods approximate this solution when paired with the famous PPO algorithm. We also provide insights on interpreting quantitative metrics of performance for self-play training. Our results indicate that, throughout training, various self-play definitions exhibit cyclic policy evolutions.

I. INTRODUCTION

In the classical single agent reinforcement learning (RL) scenarios described by [1], where a stationary environment is modelled 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 multi-agent system, for an agent's optimal strategy is dependent on behaviours 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.

To approximate this solution, traditional multi-agent RL (MARL) methods would train and benchmark a policy against a set of preexisting fixed agents, using as a success metric the relative performance against these agents. These methods rest on two assumptions. Firstly, the availability of benchmarking policies at training and testing time. 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 features many flaws. if this benchmarking set of is too small, the trained policy may overfit to the behaviour 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, favouring empirical results.

What about the cases in which we don't have access to these learning resources? Such as when developing a new game for which no prior expert information is known, and for which any hand-crafted evaluation functions yields a fruitless policy. A priori methods such as optimistic policy initialization are still permitted [2]. Yet, under such constraints, there is little room to compute a set of good benchmarking policies, let alone a set of dominating policies.

Authors such as [3] began experimenting on self-play (SP). SP is an open-ended learning training scheme which arises in the context of multi-agent training. A SP training scheme trains a learning agent *purely* by simulating plays with itself, or with policies which have been generated during training. These generated policies can dynamically build a set of benchmarking policies during training. Such 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 [4] or dynamical systems analysis [5].

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 the creation of more nuanced and efficient contributions. Incremental efforts on existing and future contributions can now be captured on a shared language. This paper constitutes a first step towards defining a generalizing framework under which SP MARL methods can be inspected. Our contributions:

- A generalizing framework defined under formal notation to describe SP algorithms in MARL.
- A unifying definition under the presented framework of some prevalent SP algorithms from the literature.
- 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. [3] 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 [6] featured SP to learn a policy using $TD(\lambda)$ [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 [7]. More recently, AlphaGo [8] used a combination of supervised learning on expert moves and SP to beat the world champion Go player. This algorithm was later refined [9], removing the need for expert human moves. A policy was learnt purely by using an mix of supervised learning on moves generated by SP and Monte Carlo Tree Search (MCTS), as presented in [10]. These works echo the sentiment that superhuman AI needs 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. [11] relaxes 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 behaviour of some of the environment's agents. The authors claim that such version of SP aims at training a policy which is able to defeat random older versions of itself, ensuring continual learning. This notion of a "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 [12]. This is of genuine importance, given that most RL algorithms' convergence properties heavily rely on the assumption of a stationary environment [13]. 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 behaviours inhabits the environment, a

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

powerful assumption in transfer learning [14] and multi-task learning [15]. In fact, there already are multitask meta-RL algorithms which assume knowledge of a distribution over MDPs which the agent is being trained on, such as RL^2 [16]. Note that a SP algorithm featuring a growing set of "historical" policies will introduce a non-stationary 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 non-stationary.

Recently, [17] 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 is there to learn from a policy. AlphaStar [18] reached Grandmaster level in StarCraft II with various policies by using a combination of various SP algorithms [19]. Part of their training pipeline relied on training a set of "exploiter" policies which focus on exploitining specific policies under training, relaxing the need for them to be robust to all opponents.

[20] Defines 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 (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 [21] [4] towards existing multiagent solutions [5]. Our contribution shares the spirit of creating a generalised 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, vectors ($\boldsymbol{\pi} \in \mathbb{R}^n$). Bold uppercase, matrices ($\boldsymbol{A} \in \mathbb{R}^{n \times n}$).

A. Normal form games

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. $\forall i \in \{n\}$, $\pi_i \in BR(\pi_{-i})$. 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 ($\Pi_1 = \ldots = \Pi_n$) 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** [4] is a meta-game 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 $a_{i,j} = w_{i,j} - \frac{1}{2}$, shifting the range of each entry from $[0, 1]$ to $[-\frac{1}{2}, \frac{1}{2}]$. Symmetrical 2-player zero-sum games represented by an antisymmetric matrix $\boldsymbol{A}$ feature a unique maxent Nash [4], a fact we will use in Section V. Finally, the **relative population performance** [4] is a population-level meassure 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, $v = 0$ indicates both populations are equivalent.

B. Multiagent Reinforcement Learning

Let E represent a multi-agent system with n agents and a reward discount factor γ. This environment E features a state space S, a joint observation space $O = O_1 \times \ldots \times O_n$ and a joint action space $A = A_1 \times \ldots \times A_n$, 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 each agent 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 $\Pi = \Pi_1 \times \ldots \times \Pi_n$ 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 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:

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

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.

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 π', taken from a set of *fixed* policies $\pi' \subseteq \pi^o$, will define the agents' behaviour for the next episode. This *excludes* the agent whose behaviour 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 π^o. This intuition is formally captured in Algorithm 1, which presents a SP scheme inside a Partially Observable Stochastic Game (POSG) loop. Algorithm 1 defines an *n-player*, *general-sum*, *partially-observable* environment. The steps belonging to the SP scheme have been highlighted in orange.

A. Framework definition

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

- $\pi^o \subseteq \Pi_i$; The **menagerie**. A set of policies from which agents' behaviour will be sampled. This set always includes the currently training policy π. A constraint is

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*: $\pi \in \Pi_i$

```
1  π° = {π} ;                    // Menagerie initialization
2  for e = 0, 1, 2, ... do
3  │   π′ ∼ Ω(π°, π) ;            // Sample from menagerie
4  │   π = π′ ∪ {π};
5  │   s₀, o₀ ∼ ρ₀;
6  │   for t = 0, ..., termination do
7  │   │   aₜ ∼ π(oₜ);
8  │   │   s_{t+1}, o_{t+1} ∼ P(sₜ, aₜ);
9  │   │   rₜ ∼ R(sₜ, aₜ);
10 │   │   t ← t + 1;
11 │   end
12 │   π ← update(π);
13 │   π° ∼ G(π°, π) ;            // Curate menagerie
14 end
15 return π;
```

placed over π^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.

- $\Omega(\pi' \in \Pi_{-i} | \pi^o \subseteq \Pi_i, \pi \in \Pi_i) \in [0, 1]$; where $\pi' \subseteq \pi^o$; The **policy sampling distribution**. A probability distribution over the menagerie π^o, the set of available policies. It is conditioned on the menagerie π^o and the current policy π being trained. It chooses which policies, apart from π, will inhabit the environment's agents. *[handwritten: Policies that will be trained on; Prob of throws → vector]*

- $G(\pi^o \subseteq \Pi_i | \pi^o \subseteq \Pi_i, \pi \in \Pi_i) \in [0, 1]$; The **curator** or gating function, of the menagerie. A possibly stochastic function whose parameters are the current training policy π and a menagerie π^o. The curator serves two purposes, which complex curators could break into two functions: *[handwritten: new archive]*

  - G decides if the current policy π will be introduced in the menagerie. *[handwritten: selects if]*
  - G decides which policies in the menagerie, $\pi \in \pi^o$, will be discarded from the menagerie.

The curator bears resemblance with the notion of Hall of Fame from evolutionary algorithms [22]. 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.*

---

[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.

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 [23], and is required for policy gradient algorithms [24].

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

C. Self-play as an approximation to the multiagent solution

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

The policy sampling distribution Ω and the gating function G are tools by which a menagerie π^o can be computed and curated over time. Self-play can be conceived as a bottom up approach towards computing a set of policies, π^o, to be used as a proxy for the entire policy space Π in equation 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 equation 1.

[4] introduces 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 π^o a menagerie:

1) Naive Self-Play: The is the oldest and simplest SP algorithm, originating in [3]. 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 behaviour. To capture this, the policy sampling distribution Ω puts all probability weight to the latest π.

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

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.

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

2) δ-Uniform Self-Play: Introduced by [11] 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.

With this in mind, we present a novel policy sampling distribution, named δ-Limit Uniform, that gives increased probability to later policies. An attempt to amend the δ-Uniform bias. Figure 1 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 towards earlier policies.

Let $|\boldsymbol{\pi}_n^o|$ be the size of the menagerie at the beginning of the n-th episode. π_e is the e-th 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 n-th SP episode are computed as

$$\rho_e^n = \frac{1}{|\boldsymbol{\pi}_n^o|(|\boldsymbol{\pi}_n^o| - e)^2}, \quad (2) \qquad p_e^n = \frac{\rho_e^n}{\sum_{i=0}^{|\boldsymbol{\pi}_n^o|} \rho_i^n}. \quad (3)$$

2) Qualitative Metric for the Menagerie's Efficiency: A visual metric, aimed at understanding how well a menagerie approximates the entire policy space. Policies can be characterised by the behaviours/state trajectories they produce when acting in the multi-agent 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 multi-dimensional, 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 towards understanding where, inside the embedded state trajectory space, the agent has spent most time on during training. It is valuable providing we can label some subsets of the embedding space with high-level understanding of what is happening throughout the state trajectories.

- **Time Window-Avegared 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 re-visit some areas of the trajectory space, which can help identify catastrophic forgetting and cyclic policy evolutions.

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 enables 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.

VII. EXPERIMENTAL DETAILS

A. Experiment description

We now present the environments, evaluation metrics, RL algorithms and SP schemes used in our experiment.

1) Environment: Repeated imperfect Recall Rock Paper Scissors (RirRPS) as introduced in [28]. An extended form, imperfect imformation, two-player, zero-sum, simultaneous version of Rock Paper Scisors. The agent which obtains the highest cumulative reward by the end of the last repetition is considered the winner. Ties are broken uniformly at random. We choose a *repeated* game and not a single round because repeatability introduces explotability, which increases with the number of repetitions. In our experiments we use 10 repetitions, with a recall of the last 3 joint actions.

RirRPS is a highly (but not fully) cyclic game. In fully cyclic games, improving an agent's performance against another agent is always counterbalanced by a decrease in performance against other possible agents, implying that invididual agent improvement is inconsequential [4].

TABLE I: PPO hyperparameters used for both experiments.

Hyperparameter	Qualitative study	Quantitative study
Horizon (T)	2048	128
Adam stepsize	3×10^{-4}	10^{-5}
Num. epochs	10	10
Minibatch size	64	16
Discount (γ)	0.99	0.99
GAE parameter (λ)	0.95	0.95
Entropy coeff.	0.01	0.01
Clipping parameter (ϵ)	0.2	0.2

2) Algorithmic choices: For our qualitative studies we used Proximal Policy Optimization [29] where the underlying policy is represented by either by a feedforward neural network (MLP-PPO) or a recurrent architecture (RNN-PPO). Four our quantitative studies we only use MLP-PPO.

3) Self-Play choices: We train a PPO agent on a 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 conveged

and (2) the number of episodes $n_{matches}$ that will be used to compute the aforementioned winrate. We used $w = 72\%$, $n_{matches} = 50$.

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

4) Evaluation metrics:

*a) **Winrate matrices**:* SP algorithms train / modify a policy π overtime. 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 π_t^o. Thus, we can generate a population which represents the evolution of the policy training under an SP algorithm overtime, $\boldsymbol{\pi_{sp}} = [\pi_{t_0}, \pi_{t_1}, \ldots]$. By examining the evaluation matrix generated from this population $\boldsymbol{W_{\pi_{sp}}}$ we can quantitatively examine if different SP algorithms suffer from catastrophic forgetting or cyclic policy evolutions.

*b) **Evolution of relative population performance**:* As introduced in Section III, we shall use the relative population performance as a direct meassure of the relative quality between the populations spawned by two different SP algorithms. We are interested in how this relative performance evolves overtime. 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 $\boldsymbol{A_{\pi_{sp_1}, \pi_{sp_2}}}$ between both populations.
3) Compute $\boldsymbol{A_{sub}} = \{\boldsymbol{A_{1...i \times 1...i}} : i \in \{n\}\}$, 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 [30]. 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 paper, to compute the winrate for an entry $w_{i,j}$ in an empirical winrate matrix $\boldsymbol{W}$ we use 30 simulations.

VIII. RESULTS

A. Qualitative analysis

Figure 2 shows the 2D t-SNE state trajectory embeddings for all combinations of SP algorihtm & RL algorithm introduced in the previous section. Each training session lasted for a $1e4$ episodes on the RirRPS environment.

Each SP agent using naive SP and $\delta = 0$-Limit Uniform exhibits cyclic catastrophic forgetting as their time window-averaged trajectories in the embedded space display cyclic movement, whereas $\delta = 0$-Uniform's time window-averaged trajectories seem less affected.

Especially in the cases of the $\delta = 0$-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 picks, indicating that recurrent policies seem to further spread the menagerie over the whole policy space to some greater extent compared to feedforward policies.

Comparing $\delta = 0$ Uniform and $\delta = 0$-Limit Uniform SPs, we can observe a progressive and somewhat ordered exploration of the policy space by the former. $\delta = 0$-Uniform's Time Window-averaged SP episode trajectories visit each fixed agent clusters one by one. Since the former biases towards earlier policies that have entered the menagerie when sampling opponent, we hypothesize that this time-related bias is entering in synergy with the learning rate of the trained policy. Indeed, after behaving like a Rock Agent (green cluster), the trained policy starts to behave like a Paper Agent (purple cluster) as the Rock Agent-behaving policies that have entered in the menagerie progressively starts to be sampled as opponent. Both the MLP-PPO- and RNN-PPO-equipped agents exhibit that cyclic and ordered exploration of the embedding space.

B. Quantitative analysis

The results from Figure 3 are metrics gathered on a single training run due to the computational requirements of averaging results over many runs. However, the behaviour captured is a representative sample of many training runs.

Each row i of winrate matrix $\boldsymbol{W}$ represents the winrates of policy at checkpoint i against all other policies checkpointed 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 is always 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.

We turn our focus to the winrate matrices from Figure 3. 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 in Figure 3.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.

Figure 3.C shows the evolution of the policy training under $\delta = 0.5$-Uniform (Half history). This policy attempts to compute a best response against the later half of its history. We see that on average, for a given row i, the corresponding policy tends to win against policies $j \in [\frac{i}{2}, i-1]$. Note that 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 in RirRPS.

$\delta = 0$-Uniform (Figure 3.D), whose underlying policy attempts a best response against its entire history, shows a
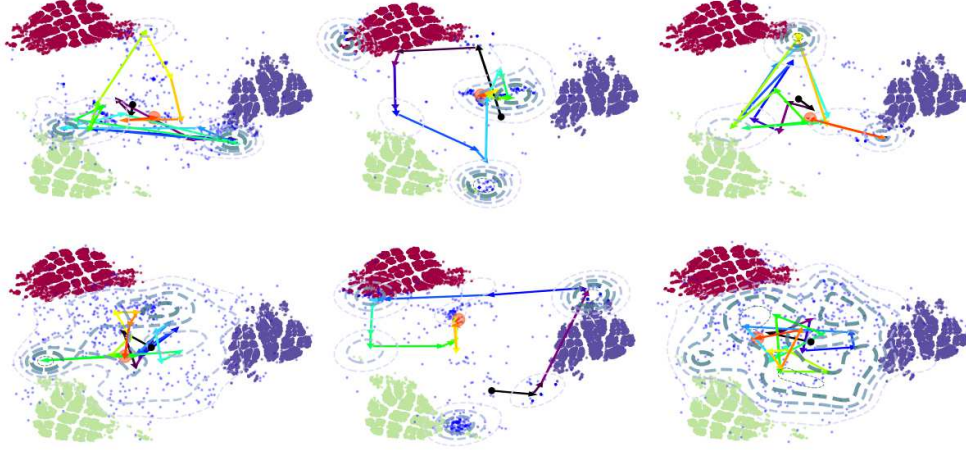
Fig. 2: Density Heightmap and Time Window-averaged SP-induced of episode trajectories in the computed 2D t-SNE state trajectory embedding space. **Top-Left:** Naive SP with MLP-PPO. **Bottom-Left:** Naive SP with RNN-PPO. **Top-Centre:** $\delta = 0$-Uniform SP with MLP-PPO. **Bottom-Centre** $\delta = 0$-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 represents 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 colour transitions.
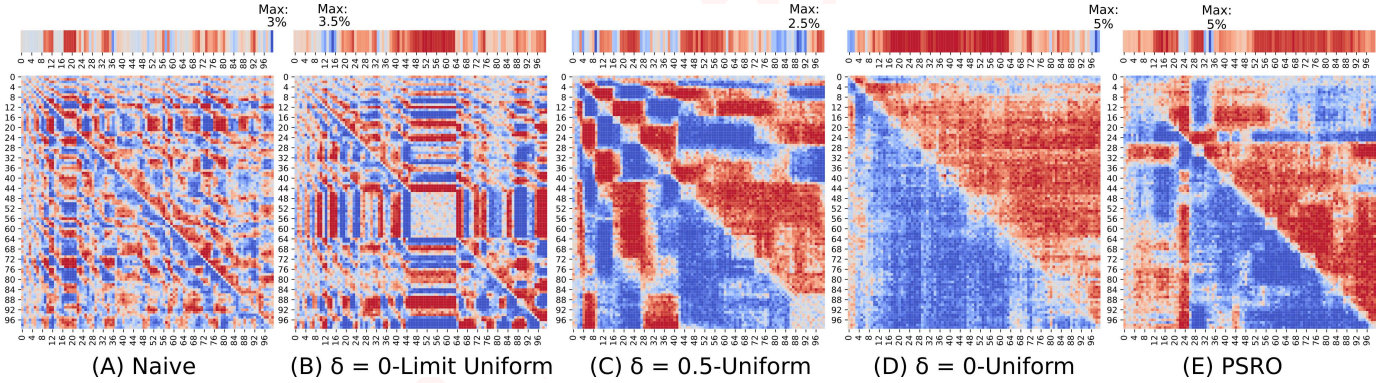

Fig. 3: Empirical winrate matrices showing the evolution of 6 policies where each one is being trained via a different SP algorithm in RirRPS. For every SP training process, we sample a policy after every policy update for a total of 100 policy checkpoints and 12800 training episodes. Treating each matrix 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.

close-to-ideal empirical winrate matrix insofar as any given policy i beats most previous versions of itself and loses against later ones. Also, for any subgame of Figure 3.D the largest concentration of support under Nash consistently lays on the latest policies.

PSRO's winrate matrix, depicted in Figure 3.E, does follow a positive trend, although less so than $\delta = 0$-Uniform, as checkpoints beyond the 37th lose against policies 20 to 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 Figure 3.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 game from Figure 3.E.

IX. DISCUSSION

a) Cyclic policy evolutions: As expected, naive SP clearly features a cyclic policy evolution. As previously stated, an ideal SP would yield policies that always beat previous ones. In contrast, almost all checkpoints obtained during naive SP training cycle between losing and winning against

previous and future checkpoints. This is further evidenced by the support under Nash from Figure 3.A, where under Nash equilibrium many policies share the highest amount support (around 3%). We observed similar cyclic behaviour in $\delta = 0$-Limit Uniform in Figure 3.B and in $\delta = 0.5$-Limit Uniform (not shown). Which may entail that δ-Limit Uniform SPs over-correct the bias towards earlier policies, matching our observations on the previous qualitative analysis.

$\delta = 0$-Uniform (Full history) does not exhibit a cyclic policy evolution. Full history tends towards generating monotonically better policies. That is, on average any given checkpoint is able to obtain positive winrates when matched against previous checkpoints. Hence, we claim that full history SP is an apt SP scheme for cyclic environments, given enough computational time. On the other hand, policies trained via $\delta = 0.5$-Uniform SP seem to struggle to reliably defeat all previous versions of themselves. Hence in RirRPS we see that by excluding the earlier half of the history, the latest policy becomes exploitable by earlier policies. Note that half history's winrate matrix features more pronounced winrates (entries are either closer to 100% or 0% winrate) than the equivalent entries in full history's winrate matrix. 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 allow earlier policies to exploit it.

PSRO (Figure 3.E) does show signs of forgetting, as stated in the previous section. As training progresses, later policies begin to lose against previous ones, with this effect growing larger overtime.

b) Relative population performances: Figure 4 shows the evolution of population performances comparing the performance of naive SP against the SP algorithms from Figure 3. Interestingly, when we look at the evolution of relative population performance we notice that it converges near zero for all SP algorithms. For RirRPS, this implies that the populations generated by all SP as training progresses are of similar quality, furthering the idea that in highly cyclic games individual policy improvement is not meaningful, even when there is potential to exploitation due to repetitions 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 Figure 3.

A possible reason why naive SP performs evenly or positively against all other SP algorithms is that early on in training it quickly cycles through rock / paper / scissors policies, and from those three policies it is possible to compose almost any policy in RirRPS.

c) Fragility of PSRO's oracle hyperparameters: Small changes in the oracle's hyperparameters (winrate threshold w, window size of match outcomes $n_{matches}$) can quickly lead to unfeasibly long training times (too many policies added to the menagerie) or arguably degenerate behaviour by the SP algorithm (the curator never introduces new policies into the menagerie). In the worst case scenario, the training policy will never convege towards a best response against the initial (randomly initialized) policy in the menagerie. We show in Table II a sweep over both hyperparameters in RirRPS. Most
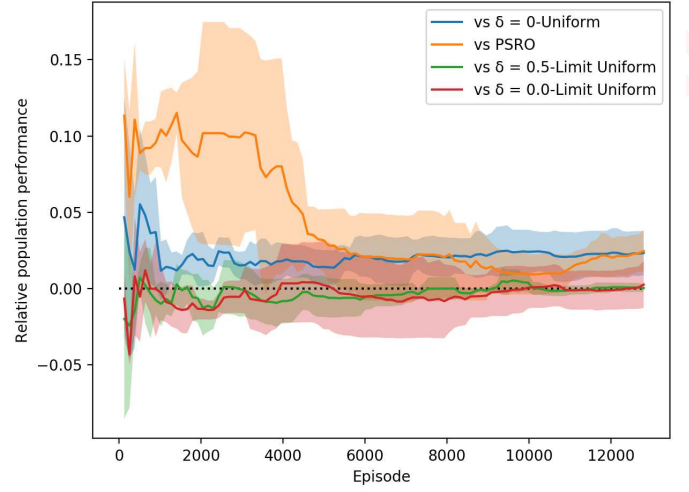

Fig. 4: Evolution of relative population performance of naive SP against 4 other SP schemes. The cyclic behaviour of naive SP quickly discovers how to play Rock, Paper and Scissors, which are enough to generate a Nash equilibrium, which explains the initial positive relative performance.

of the training time is spent inside of the meta-game solver $\mathcal{M}$, which leads us to believe that a less computationally intensive meta-game solver should be used.

A Nash Equilibrium in RirRPS is to act randomly, which we argue is likely the behaviour of the training policy at the beginning of training. Hence, it is highly unlikely that a policy will obtain a high enough winrate against this random policy to be added to the menagerie, making it difficult for the learning policy to discover policies which differ from random play. This means that the policy will not discover how to exploit policies beyond random play.

TABLE II: Hyperparameter sweep time profiling for 12k episodes in RirRPS. Columns $\mathcal{M}$ and W_{π^o} represent the percentage of training time spent on computing a meta-game solution and updating the meta-game respectively.

Hyperparameter values	$\mathcal{M}$	W_{π^o}	Total training time	$|\pi^o|$
(60%, 30)	95%	2%	>2d	332
(70%, 30)	95%	5%	1d 12h 44m	294
(75%, 30)	70%	25%	1h 38m	139
(80%, 30)	59%	32%	55m	118
(85%, 30)	0%	0%	4m	1
(70%, 45)	69%	24%	58m	112
(75%, 45)	2%	9%	5m	19
(80%, 45)	0%	0%	4m	1
(70%, 50)	67%	26%	1h 7m	123

d) Extrapolation: RirRPS is a *2-player*, *zero-sum* and *simultaneous* game. Our experimental results may not extend to *n-players* or *general-sum* games. Moreover, following the dimensionality-based definition of the complexity of a game [4], the lower-bound on the complexity of RPS and, incidentally, RirRPS are rather low. Therefore, it would be interesting to compare current results with games of verifiably greater lower-bound on their complexity, such as RoboSumo [31].

X. Conclusions & Future Work

Building on our original work [28], this paper presents a general framework in which to define SP training schemes. This is done by formalizing the notion of a menagerie, a policy sampling distribution and a curator (gating) function. This framework is framed as 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 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.

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.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 1998.

[2] M. C. Machado, S. Srinivasan, and M. Bowling, "Domain-independent optimistic initialization for reinforcement learning," 10 2014.

[3] A. Samuel, "Some studies in machine learning using the game of checkers," *Ibm Journal*, vol. 3, no. 3, p. 210, 1959. [Online]. Available: http://pages.cs.wisc.edu/{~}dyer/cs540/handouts/samuel-checkers.pdf

[4] D. Balduzzi, M. Garnelo, Y. Bachrach, W. Czarnecki, J. Perolat, M. Jaderberg, and T. Graepel, "Open-ended Learning in Symmetric Zero-sum Games."

[5] S. Omidshafiei, C. Papadimitriou, G. Piliouras, K. Tuyls, M. Rowland, J.-B. Lespiau, W. M. Czarnecki, M. Lanctot, J. Perolat, and R. Munos, "{\alpha}-rank: Multi-agent evaluation by evolution," *arXiv preprint arXiv:1903.01373*, 2019.

[6] G. Tesauro, "Temporal Difference Learning and TD-Gammon." *Commun. ACM*, vol. 38, pp. 58–68, 1995.

[7] ——, "Neurogammon: a neural-network backgammon program," *1990 IJCNN International Joint Conference on Neural Networks*, pp. 33–39 vol.3, 1990. [Online]. Available: http://ieeexplore.ieee.org/document/5726779/

[8] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. Van Den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017. [Online]. Available: http://dx.doi.org/10.1038/nature24270

[9] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm," pp. 1–19, 2017.

[10] T. Anthony, Z. Tian, and D. Barber, "Thinking Fast and Slow with Deep Learning and Tree Search," no. Il, pp. 1–19, 2017. [Online]. Available: http://arxiv.org/abs/1705.08439

[11] T. Bansal, J. Pachocki, S. Sidor, I. Sutskever, and I. Mordatch, "Emergent Complexity via Multi-Agent Competition," vol. 2, pp. 1–12, 2017. [Online]. Available: http://arxiv.org/abs/1710.03748

[12] G. J. Laurent, L. Matignon, and N. L. Fort-Piat, "The world of independent learners is not markovian," *International Journal of Knowledge-Based and Intelligent Engineering Systems*, vol. 15, no. 1, pp. 55–64, 2011.

[13] M. Asai, "Is multiagent deep reinforcement learning the answer or the question? A Brief Survey," *Igaku Toshokan*, vol. 48, no. 1, pp. 95–99, 2001.

[14] R. S. Sutton, A. Koop, and D. Silver, "Sutton-role-of-tracking," pp. 1–8, 2007. [Online]. Available: papers://d471b97a-e92c-44c2-8562-4efc271c8c1b/Paper/p522

[15] M. E. Taylor and P. Stone, "Transfer Learning for Reinforcement Learning Domain.pdf," vol. 10, pp. 1633–1685, 2009.

[16] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel, "RL\$ˆ2\$: Fast Reinforcement Learning via Slow Reinforcement Learning," pp. 1–14, 2016. [Online]. Available: http://arxiv.org/abs/1611.02779

[17] G. B. B. C. V. C. P. D. C. D. D. F. Q. F. S. H. C. H. R. J. S. G. C. O. J. P. M. P. H. P. d. O. P. J. R. T. S. J. S. J. S. S. S. I. S. J. T. F. W. S. Z. Openai, Christopher Berner.

[18] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, pp. 1–5, 2019.

[19] M. Jaderberg, W. M. Czarnecki, I. Dunning, L. Marris, G. Lever, A. G. Castaneda, C. Beattie, N. C. Rabinowitz, A. S. Morcos, A. Ruderman, N. Sonnerat, T. Green, L. Deason, J. Z. Leibo, D. Silver, D. Hassabis, K. Kavukcuoglu, and T. Graepel, "Human-level performance in first-person multiplayer games with population-based deep reinforcement learning."

[20] M. Lanctot, V. Zambaldi, A. Gruslys, A. Lazaridou, K. Tuyls, J. Perolat, D. Silver, and T. Graepel, "A Unified Game-Theoretic Approach to Multiagent Reinforcement Learning," no. Nips, 2017. [Online]. Available: http://arxiv.org/abs/1711.00832

[21] P. Muller, S. Omidshafiei, M. Rowland, K. Tuyls, J. Perolat, S. Liu, D. Hennes, L. Marris, M. Lanctot, E. Hughes *et al.*, "A generalized training approach for multiagent learning," *arXiv preprint arXiv:1909.12823*, 2019.

[22] 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, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 174–188.

[23] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized Experience Replay," pp. 1–21, 2015. [Online]. Available: http://arxiv.org/abs/1511.05952

[24] R. J. Williams, "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning," *Machine Learning*, vol. 8, no. 3, pp. 229–256, 1992.

[25] D. Balduzzi, "Re-evaluating Evaluation," no. Nips, 2018.

[26] M. Van Der Ree and M. Wiering, "Reinforcement learning in the game of Othello: Learning against a fixed opponent and learning from self-play," *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning, ADPRL*, pp. 108–115, 2013.

[27] L. V. D. Maaten and G. Hinton, "Visualizing Data using t-SNE," vol. 9, pp. 2579–2605, 2008.

[28] D. Hernandez, K. Denamganaï, Y. Gao, P. York, S. Devlin, S. Samothrakis, and J. A. Walker, "A generalized framework for self-play training," in *2019 IEEE Conference on Games (CoG)*. IEEE, 2019, pp. 1–8.

[29] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[30] M. Rowland, S. Omidshafiei, K. Tuyls, J. Perolat, M. Valko, G. Piliouras, and R. Munos, "Multiagent evaluation under incomplete information," in *Advances in Neural Information Processing Systems*, 2019, pp. 12 270–12 282.

[31] M. Al-Shedivat, T. Bansal, Y. Burda, I. Sutskever, I. Mordatch, and P. Abbeel, "Continuous adaptation via meta-learning in nonstationary and competitive environments," 2017.