A Generalized Framework for Self-Play Training

Daniel Hernandez*, Kevin Denamganaï*, Yuan Gao†, Peter York*, 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
†Department of Information and Technology, Uppsala University, Sweden. alex.yuan.gao@it.uu.se
‡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. The results indicate that throughout training the trained policies exhibit cyclic evolutions, showing that self-play research is still at an early stage.**

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 dependant 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. Discouragingly, this policy set may not only be computationally intractable to process, it may even be infinite, if stochastic policies are allowed.

To approximate this solution, an existing family of multi-agent RL (MARL) methods 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 existing policies.

This work was funded by the EPSRC Centre for Doctoral Training in Intelligent Games and Game Intelligence (IGGI) EP/L015846/1.

Within the field of RL there are multiple methods for computing these benchmarking policies which must be available before training commences. To name a few, these preexisting policies can be computed using supervised learning on datasets of expert human moves to bias learning a policy towards expert human play [2] [3]; they can be tree-search based algorithms using hand-crafted evaluation functions or Monte Carlo based approaches if an environment model is present [4]. Some methods are as creative as deriving a strong policy by using off-policy methods on video replays [5] [6].

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 [7]. 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 [8] began experimenting on self-play (SP). SP is a 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.

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 notations 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 novel exemplary SP algorithm.
- A novel qualitative performance metric for SP training.

II. Related Work

The notion of SP has been present in the game playing AI community for over half a century. [8] 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 [9] 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 [10]. More recently, AlphaGo [11] used a combination of supervised learning on expert moves and SP to beat the world champion Go player. This algorithm was later refined [12], removing the need for expert human moves. A policy was learnt purely by using an elaborate mix of supervised learning on moves generated by SP and MCTS, as presented in [13]. These works echo the sentiment that superhuman AI needs not be limited or biased by preexisting human knowledge.

In the game of Othello, [14] experimented with training single agent RL algorithms using two different training schemes: SP and training versus a fixed opponent. Their results show that, depending on the RL algorithm used, learning by SP yields a higher quality policy than learning against a fixed opponent. Concretely, TD(λ) learnt best from self-play, but Q-learning performed better when learning against a fixed opponent. Similarly, [15] found that DQN [16], a deep variant of Q-learning, did not perform well when trained against other policies which were themselves being updated simultaneously, but otherwise performed well when training against fixed opponents. The environments used for their experiments differ too much to draw parallel conclusions from their results, one of them being a board game and the other a fast-paced fighting video game.

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. [17] 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.

[1]For deep RL, this is equivalent to freezing the weights of the neural networks used as part of the algorithm.

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 [18]. This is of genuine importance, given that most RL algorithms' convergence properties heavily rely on the assumption of a stationary environment [19]. Self-play 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 powerful assumption in transfer learning [20] and multi-task learning [21]. 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 [22]. 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, that the training agent will encounter, becomes non-stationary.

Similar ideas have also been independently discovered in other fields. Some methods in computational game theory directly tackle the idea of computing a strong policy[2] by iteratively constructing a set of monotonically stronger policies. In turns, iterated best responses better challenge the current policy to compute better responses to those, thus generating a stronger policy. Alternating fictitious play [23] iteratively computes a best response over set of policies that the learning agent expects the opponent agents to use. [24] devised a unifying game theoretical framework to capture this iterative best-response computation over a set of potential, or previously encountered, opponent agents. 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.

In psychology, [25] introduces the *adaptive staircase procedure*, where a learning agent is presented with a set of increasingly difficult tasks. After multiple successful trials at a task, the agent is promoted to harder tasks, otherwise it is demoted to easier ones. Such procedure was shown to prevent catastrophic forgetting [3] on trials outside its current level of difficulty, linking their results with [17] and SP. This was empirically demonstrated in the deep RL architecture UNREAL [26] for virtual visual acuity tests [27].

Unfortunately, the numerous empirical successes which motivate SP as a promising training scheme suffer from lack of formal proofs of convergence or even rate thereof [3]. We hope to provide a simple, yet powerful tool to analyze SP schemes in the next section.

[2]The notion of a policy in RL is roughly equivalent to that of a strategy in game theory, the term policy is used for consistency.

[3]In a multi-agent reinforcement learning context, catastrophic forgetting refers to the event of a policy dropping in performance against policies for which it used to perform favourably better.

III. Proposed 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.

Notation: Italic letters represent scalars (n), bold letters represent vectors ($\boldsymbol{\pi}$).

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 $O = \Pi_{i=1}^{n} O_i$ and a joint action space $A = \Pi_{i=1}^{n} A_i$, where O_i and A_i represent the observation and action space for the ith agent respectively. Let the mapping from observations to actions $\pi_i : O_i \rightarrow A_i$ 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 observation space $\boldsymbol{\pi} : O \rightarrow A$. Let $\Pi = \Pi_1 \times \ldots \times \Pi_n$ be the joint policy space, where Π_i is the policy space for agent i. Finally, given an agent i, 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:

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

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 $\boldsymbol{a_t}$, yielding a new environment state s_{t+1} and a reward vector $\boldsymbol{r_t}$ containing a reward for each agent. This loop is repeated until a terminal state is reached, after which a new episode begins.

Self-play can be conceived as a module which extends this loop by introducing a functionality prior to, and after, every episode. Let π be the only policy being trained throughout the MARL loop. A SP module envelops the RL loop by first deciding which policies $\boldsymbol{\pi'_e}$, taken from a set of *fixed* policies $\boldsymbol{\pi'_e} \subseteq \boldsymbol{\pi^o_e}$, will define the agents' behaviour before episode e begins. 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 $\boldsymbol{\pi^o_{e+1}}$. This intuition is formally captured in Algorithm 1. The SP module has been highlighted in orange, the black text represents the vanilla MARL loop.

A. Framework definition

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

Algorithm 1: (dec-POMDP) RL Loop with Self-Play.

```
Input: Environment: $(S, A, O, \mathcal{P}(\cdot,\cdot|\cdot,\cdot), \mathcal{R}(\cdot,\cdot), \rho_0)$
Input: Policy to be trained: $\pi(\cdot)$
1  $e \leftarrow 0$;
2  $\pi_e \leftarrow \pi$;
3  $\boldsymbol{\pi^o_0} = \{\pi_e\}$ ;            // Menagerie initialization
4  for $e = 0, 1, 2, \ldots$ do
5      $\boldsymbol{\pi'}_t \sim \Omega(\boldsymbol{\pi^o_e})$ ;        // Menagerie sampling
6      $\boldsymbol{\pi_e} = \boldsymbol{\pi'}_t \cup \{\pi_e\}$;
7      $t \leftarrow 0$;
8      $s_t, \boldsymbol{o_0} \sim \rho_0$;
9      repeat
10         $\boldsymbol{a_t} \sim \boldsymbol{\pi_e}(\boldsymbol{o_t})$;
11         $s_{t+1}, \boldsymbol{o_{t+1}} \sim P(s_t, \boldsymbol{a_t})$;
12         $\boldsymbol{r_t} \sim \boldsymbol{R}(s_t, \boldsymbol{a_t})$;
13         $t \leftarrow t + 1$;
14     until Episode termination;
15     $\pi_{e+1} \leftarrow update(\pi_e)$;
16     $\boldsymbol{\pi^o_{e+1}} \sim G(\boldsymbol{\pi^o_e}, \pi_e)$ ;     // Menagerie curation
17     $e \leftarrow e + 1$;
18 end
19 return $\pi_e$;
```

- $\boldsymbol{\pi^o} \subseteq \Pi$; 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 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. Because the menagerie may change after every episode, we denote $\boldsymbol{\pi^o_e}$ as the menagerie at the beginning of episode e.
- $\Omega(\boldsymbol{\pi'}|\boldsymbol{\pi^o}, \pi) \in [0, 1]$; $\Omega : \Pi \times \Pi \rightarrow \Pi$; where $\boldsymbol{\pi'} \subseteq \boldsymbol{\pi^o}$, $\pi \in \Pi$; 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.
- $G(\boldsymbol{\pi^{o\prime}}|\boldsymbol{\pi^o}, \pi) \in [0, 1]$; $G : \Pi \times \Pi \rightarrow \Pi$; Is the gating function, or curator of the menagerie. A possibly stochastic function whose parameters are the current training policy π and a menagerie $\boldsymbol{\pi^o}$. Its function is twofold:
  - G decides if the current policy π will be introduced in the menagerie.
  - 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 [28]. 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*[4]*. Allowing any-time read and write rights for all policies.*

The definitions above capture the minimal structure of an SP training scheme. 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 extremely useful in RL research [29], and is in fact mandatory for policy gradient algorithms [30].

Our SP framework only interacts with agent policies, making the choice of environment model orthogonal to any flavour of SP, and thus SP training is agnostic to the underlying environment model. With MMDPs, dec-POMDPs and Partially Observable Stochastic Games being some of the most used environment models in the literature.

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

Assumption 1: *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 equation 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 towards computing a set of policies, $\boldsymbol{\pi^o}$, 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.

[31] 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.

IV. Self-Play Algorithms

To demonstrate the generalizing capabilities of our framework, we present three prevalent self-play mechanisms extracted from the RL literature, and two novel contributions. Let π denote a policy being trained, and $\boldsymbol{\pi'}$ a menagerie:

1) Naive Self-Play: The is the oldest and simplest SP algorithm, originating in [8]. 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.

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

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) δ-Uniform Self-Play: This SP algorithm, introduced by [17] and briefly discussed in Section II, treats the menagerie as a set of "historical" policies. The authors wanted to create an SP algorithm that ensured continual learning by training a policy which could consistently beat random previous 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 that will be considered as a potential candidate to be sampled from the menagerie by Ω. Thus, $\delta = 0$ corresponds to all policies in the menagerie being considered as candidates and $\delta = 1$ only allowing the latest 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 [32], Population Based Training SP is a parallel self-play implementation highly 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 amongst 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. Skill is measured by computing the agent's Elo rating.

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.

V. Proposed Incremental Innovations

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.

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

1) δ-Limit Uniform policy sampling distribution: In traditional 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 non-stationary distribution of 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* 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 evidences 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 [14].

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 $|\boldsymbol{\pi}_{\boldsymbol{n}}^{\boldsymbol{o}}|$ 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

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

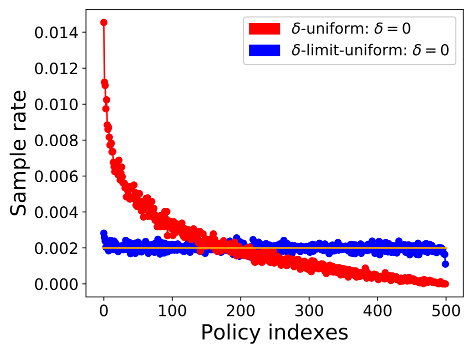

Fig. 1: Histograms of sample rates for policies inside a menagerie for two sample training runs. The horizontal orange line represents a $Uniform(0, 500)$ distribution.

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 [33] 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 Self-Play-induced episode 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 during throughout training time. 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 skirt 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.

VI. Experimental Details

In this section, we aim to qualitatively evaluate the effect of different SP training schemes when paired with two variants of PPO [34], a popular on-policy RL algorithm, using our novel qualitative metric assessing the efficiency of the menagerie when it comes to acting as a proxy to the whole policy space.

A. Experiment description

The rest of this section presents the environment, evaluation metrics, algorithms and SP schemes used in our experiment.

1) Environment: For our experiment, we implemented a repeated version of Rock Paper Scissors (RPS) with imperfect recall [35], named Repeated imperfect recall Rock Paper Scissors (RirRPS)[5]. This is an extended form, two-player, zero-sum, simultaneous game. 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.

A repeated game of RPS features the iconic discrete action space $A = \{Rock, Paper, Scissors\}$. To reduce the complexity of the state space, and to ensure a state representation of fixed length[6], we make our repeated RPS a game of imperfect recall. That is, the agents do not have access to the full history of previous actions, only a fraction of them. The state is represented by the last n_{recall} joint actions taken by the players. Thus, $s_t = (\boldsymbol{a}_{t-n_{recall}}, \dots, \boldsymbol{a}_{t-1})$. A placeholder empty move is used to populate the state when there have not been enough repetitions to fill the state representation yet. The environment's reward function is specified as: paper beats rock, rock beats scissors, scissors beat paper, and the winner of each repetition is attributed a reward of $+1$, and the loser a penalty of -1. In the following of this work, $n_{recall} = 3$.

2) Evaluation metrics: We focus our attention to using our novel qualitative metric introduced in Section V, based on projecting state trajectories encountered during agent training unto a 2D space. To stabilize the t-SNE projection, we need to compute a set of basis trajectories. This set is comprised of trajectories gathered from invidivually pitting *Rock*, *Paper* and *Scissors* agents against a *Random* agent. This set is used to help visualizing the natural and stable shape of the dimensionality reduced space of all the possible state trajectories.

TABLE I: PPO hyperparameters used for the experiment.

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

[5]We open source our implementation using the OpenAI's Gym environment [36] and make it available at: https://github.com/Danielhp95/gym-rock-paper-scissors

[6]Suitable for feedfowrad neural networks.

3) Algorithmic choices: We used two RL algorithms, both of them being slight variants of the Proximal Policy Optimization (PPO) algorithm [34]. We chose to use two architecture variants for the policy, one feedforward (MLP-PPO), and one recurrent(RNN-PPO), in order to demonstrate the generality of our framework[7]. Hyperparameters are presented in table I.

4) Self-Play choices: For each of the algorithms mentioned above, we trained an agent on a self-play extended MARL loop as shown in algorithm 1. We used 3 self-play schemes:

- Naive SP
- $\delta = 0$-Uniform SP.
- $\delta = 0$-Limit Uniform SP.

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

VII. Results

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 clearly exhibit cyclic catastrophic forgetting as their time window-averaged trajectories in the embedded space exhibit a cyclic behaviour, whereas $\delta = 0$-Uniform and $\delta = 0$-Limit Uniform SPs' time window-averaged trajectories seem both 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 the spreading of 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 as its Time Window-averaged SP episode trajectories in the embedding space are visiting one by one each fixed agent clusters. 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 progression in the embedding space. Free of the sampling bias, the latter SP exhibit a similarly ordered progression in the embedding space when looking at it with a coarser time window division, and with some quick alternation when looking at it with a finer time window division, as expected of a truely uniform opponent sampling distribution.

[7]In order to address any potential experimental reproduction issue, we have open-sourced our re-implementations of the aforementioned RL algorithms. We welcome our peers to collaborate with us in extending the number of algorithms that are available as well as to improve the overall architecture.

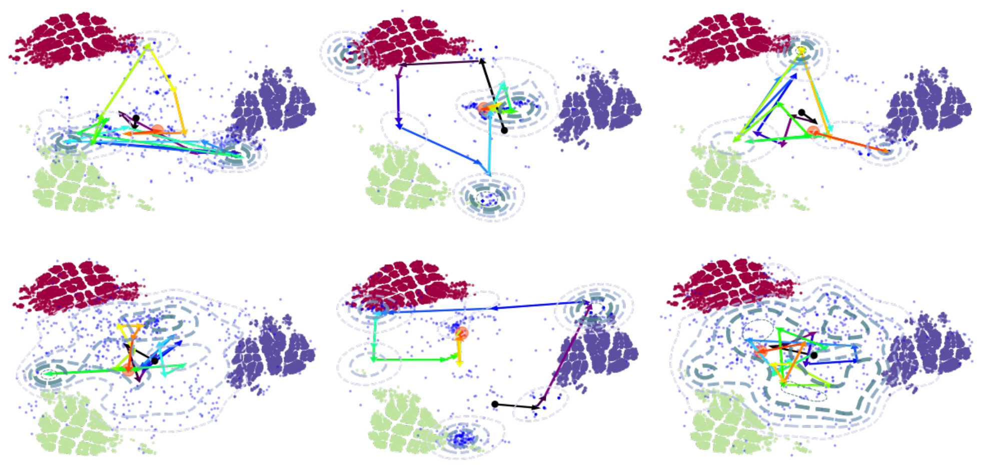

Fig. 2: Density Heightmap and Time Window-averaged SP-induced of episode trajectories in the computed 2D t-SNE state trajectory embedding space. **Top-Left:** Naive SP with MLP-PPO. **Bottom-Left:** Naive SP with RNN-PPO. **Top-Centre:** $\delta = 0$-Uniform SP with MLP-PPO. **Bottom-Centre** $\delta = 0$-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.

A. *Discussions*

*1) **Evaluation Metrics**:* It stands to reason that our proposed qualitative metric would yield a (visual) measure of the extent to which the menagerie is actually able to approximate the policy space. Our novel qualitative metric is a first step towards the construction of a robust, quantitative metric for the menagerie-induced spread over the whole policy space, for a given environment.

*2) **Stochastic vs Deterministic policies**:* Independently of which RL exploration scheme is used to help any RL agent explore an environment's state space, it stands to reason that the stochastic variations induced by the use of stochastic policies naturally enhance any RL agent's ability to explore the state space, compared to the use of deterministic policies. Thus, in the context of SP training, stochastic policies may be preferred over deterministic ones when the goal is to discover a menagerie which acts as a proxy for the whole policy space, due to increased exploration.

*3) **PPO Hyperparameters**:* PPO's horizon (T) is the hyperparameter that controls how many environment interactions will be gathered prior to a policy update. This hyperparameter controls the extent to which Algorithm 1's inner loop is operating with the stationary policies. Indeed, until (T) experiences have been gathered, the policy being trained by the PPO algorithm remains unchanged. Therefore, the horizon hyperparameter might affect the amount of opportunities the SP loop has to yield updated policies for the curator to consider. Depending on the SP scheme used, it can greatly affect the rate at which the menagerie spreads over the whole policy space. Many RL algorithms feature similar hyperparameters. We hypothesize that such hyperparameters will be critical in terms of generating a menagerie to act as a proxy to the whole policy space.

*4) **Extrapolation**:* Because our environment is a *2-player*, *zero-sum* and *simultaneous* game, our experimental results may not extend to *n-players* or *general-sum* games. However, our SP framework does allow such environments.

VIII. Conclusions & Future Work

In this paper we have presented 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 into it. We have also identified shortcomings of some of the captured methods, and have proposed methods which could potentially overcome said issues. We carried an experiment featuring a novel qualitative metric that denotes how well a menagerie expresses the entire policy space by visually displaying how much of the (embedded) state trajectory space an agent visits during training. Our results show that there are qualitative difference between how different SP algorithms traverse this space.

It will be interesting to carry out SP centered experiments which use more nuanced existing evaluation metrics, or to create evaluation metrics which better capture the quality of a menagerie to act as a proxy of an environment's policy space.

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.

ACKNOWLEDGEMENTS

The authors wish to thank Professor Peter Cowling for making this paper possible.

REFERENCES

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

[2] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of {Go} with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[3] G. Tesauro, "Practical Issues in Temporal Difference Learning," *Machine Learning*, vol. 8, no. 3-4, pp. 257–277, 1992. [Online]. Available: papers3://publication/uuid/4B128386-0184-4728-AC88-0F3385D1036F

[4] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of {Monte Carlo Tree Search} methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012.

[5] Y. Aytar, T. Pfaff, D. Budden, T. L. Paine, Z. Wang, and N. de Freitas, "Playing hard exploration games by watching YouTube," 2018. [Online]. Available: http://arxiv.org/abs/1805.11592

[6] K. Malysheva, Shpilman, "Learning to Run with Reward Shaping from Video Data," 2018.

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

[8] 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

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

[10] ——, "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/

[11] 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

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

[13] 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

[14] 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.

[15] V. Firoiu, W. F. Whitney, and J. B. Tenenbaum, "Beating the World's Best at Super Smash Bros. with Deep Reinforcement Learning," 2017. [Online]. Available: http://arxiv.org/abs/1702.06230

[16] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," pp. 1–9, 2013. [Online]. Available: http://arxiv.org/abs/1312.5602

[17] 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

[18] 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.

[19] 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.

[20] 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

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

[22] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel, "RL^2: Fast Reinforcement Learning via Slow Reinforcement Learning," pp. 1–14, 2016. [Online]. Available: http://arxiv.org/abs/1611.02779

[23] U. Berger, "Brown's original fictitious play," *Journal of Economic Theory*, vol. 135, no. 1, pp. 572–578, 2007.

[24] 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

[25] B. Treutwein, "Adaptive Psychophysical Procedures," 1995.

[26] C. Beattie, J. Z. Leibo, D. Teplyashin, T. Ward, M. Wainwright, H. Küttler, A. Lefrancq, S. Green, V. Valdés, A. Sadik, J. Schrittwieser, K. Anderson, S. York, M. Cant, A. Cain, A. Bolton, S. Gaffney, H. King, D. Hassabis, S. Legg, and S. Petersen, "DeepMind Lab," pp. 1–11, 2016. [Online]. Available: http://arxiv.org/abs/1612.03801

[27] J. Z. Leibo, C. d. M. D'Autume, D. Zoran, D. Amos, C. Beattie, K. Anderson, A. G. Castañeda, M. Sanchez, S. Green, A. Gruslys, S. Legg, D. Hassabis, and M. M. Botvinick, "Psychlab: A Psychology Laboratory for Deep Reinforcement Learning Agents," pp. 1–28, 2018. [Online]. Available: http://arxiv.org/abs/1801.08116

[28] 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.

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

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

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

[32] 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."

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

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

[35] M. Shafiei, N. Sturtevant, and J. Schaeffer, "Comparing UCT versus CFR in simultaneous games," *Proceeding of the IJCAI Workshop on General Game Playing*, 2009.

[36] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," pp. 1–4, 2016. [Online]. Available: http://arxiv.org/abs/1606.01540