A Survey on Self-play Methods in Reinforcement Learning

Anonymous authors
Paper under double-blind review

Abstract

Self-play, characterized by agents' interactions with copies or past versions of themselves, has recently gained prominence in reinforcement learning (RL). This paper first clarifies the preliminaries of self-play, including the multi-agent reinforcement learning framework and basic game theory concepts. Then, it provides a unified framework and classifies existing self-play algorithms within this framework. Moreover, the paper bridges the gap between the algorithms and their practical implications by illustrating the role of self-play in different scenarios. Finally, the survey highlights open challenges and future research directions in self-play. This paper is an essential guide map for understanding the multifaceted landscape of self-play in RL.

1 Introduction

Reinforcement learning (RL) represents a significant paradigm (Sutton & Barto, 2018) within machine learning (ML), focused on optimizing decision processes through interaction with an environment. In RL, the environment is modeled as a Markov decision process (MDP), where agents observe states, take actions, receive rewards, and cause transitions to new states. The primary goal of RL algorithms is to derive the optimal policy that yields the maximum expected accumulated reward over time. Deep RL extends traditional RL by employing deep neural networks as function approximators to handle high-dimensional state spaces, contributing to breakthroughs in various complex tasks (Mnih et al., 2013).

Moreover, transitioning from single-agent to multi-agent reinforcement learning (MARL) introduces complex dynamics (Rashid et al., 2018; Mahajan et al., 2019; Yu et al., 2022). In MARL, the interdependence of agents' actions presents significant challenges, as the environment appears non-stationary to each agent. The main issues in MARL are coordination, communication, and equilibrium selection, particularly in competitive scenarios. These challenges often lead to difficulties in achieving convergence, maintaining stability, and efficiently exploring the solution space.

With the help of game theory, a mathematical framework that models the interactions between multiple decision-makers, self-play emerges as an elegant solution to some inherent challenges in MARL. Self-play provides an approach where an agent interacts with copies or past versions of itself (Samuel, 2000; Bansal et al., 2018). This method promises a more stable and manageable learning process. The capabilities of self-play extend to a wide range of scenarios, including its high-profile applications in Go (Silver et al., 2016; 2017; 2018; Schrittwieser et al., 2020), chess (Silver et al., 2018; Schrittwieser et al., 2020), poker (Moravčík et al., 2017; Heinrich et al., 2015), and video games (Berner et al., 2019; Vinyals et al., 2019). In these scenarios, it has developed strategies that surpass human expertise. Although the application of self-play is extensive and promising, it is accompanied by limitations, such as the potential convergence to suboptimal strategies and significant computational requirements(Silver et al., 2016; 2018).

Although some research adopts a broad perspective through empirical game-theoretic analysis (EGTA)(Wellman et al., 2025), relatively few comprehensive surveys focus solely on self-play algorithms. Among them, a study develops an algorithmic framework for self-play, but it does not incorporate the Policy-Space Response Oracle (PSRO) series of algorithms (Hernandez et al., 2021), while a separate study focuses exclusively on PSRO (Bighashdel et al., 2024), without considering other self-play algorithms. Another study examines the theoretical safety of self-play (DiGiovanni & Zell, 2021). While these studies are

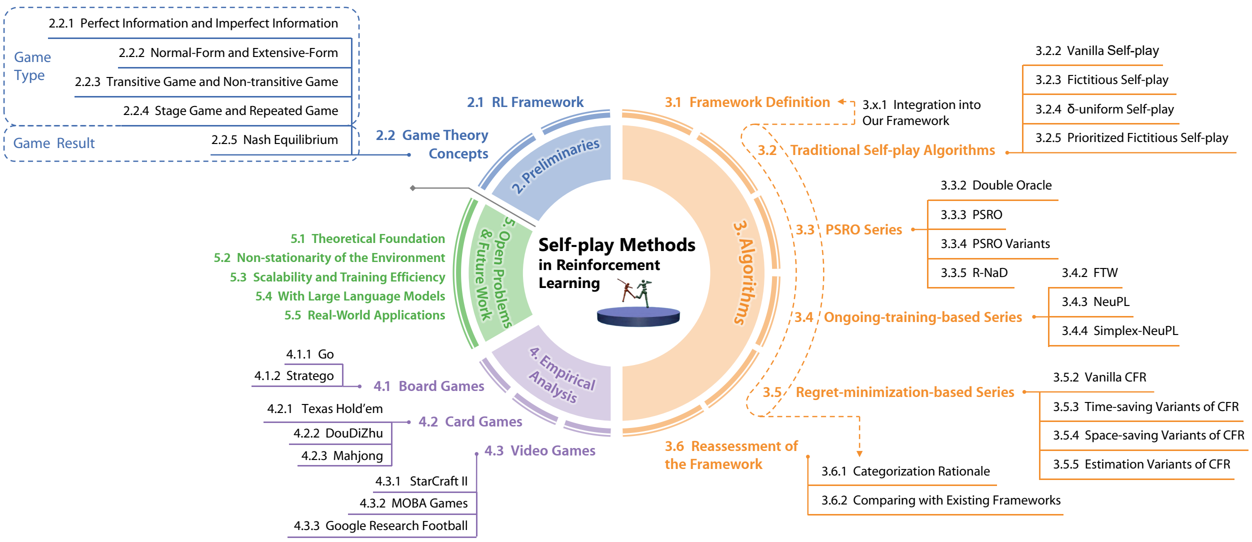

Figure 1: Overview of our survey.

valuable, they do not provide a comprehensive perspective that fully captures the breadth and depth of self-play. Therefore, this survey aims to fill this gap.

The survey is organized as follows. Sec. 2 introduces the background of self-play, including the RL framework and game theory concepts. Sec. 3 proposes a unified framework and then categorizes existing self-play algorithms into four categories based on this framework. In Sec. 4, a comprehensive analysis illustrates how self-play is applied in various scenarios. Sec. 5 describes open problems in self-play and explores future research directions. Finally, Sec. 6 concludes the survey on self-play. A more detailed overview of our survey is depicted in Fig. 1, which further illustrates the relationships among the subsections.

2 Preliminaries

This section first introduces the framework of RL, where an agent learns to make decisions by interacting with an environment to maximize cumulative rewards. We then present basic game theory concepts, which analyze strategic interactions between rational decision-makers and provide tools to study outcomes based on players' strategies.

2.1 RL Framework

In RL, the environment is modeled as an MDP, where the Markovian assumption states that the environment's evolution is fully determined by its current state, eliminating the need to consider past states. An agent interacts with the environment by taking actions, which result in different states and associated rewards. MDPs can be extended to multi-agent settings, known as Markov games (MGs) (Littman, 1994) or stochastic games (Shapley, 1953). We focus on the most general form: partially observable Markov games (POMGs), where multiple agents interact with the environment, and each agent receives only individual observations instead of the full state, meaning they have limited access to the environment's state.

A POMG $\mathcal{G}$ is defined by $\mathcal{G} = (\mathcal{N}, \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{P}, \mathcal{R}, \gamma, \rho)$. $\mathcal{N} = \{1, \cdots, n\}$ denotes n agents. $\mathcal{S}$ is the state space. $\mathcal{A} = \prod_{i=1}^{n} \mathcal{A}_i$ is the product of the action space of each agent. Similarly, $\mathcal{O} = \prod_{i=1}^{n} \mathcal{O}_i$ is the product of the observation space of each agent. $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ denotes the transition probability from one state to another given the actions of each agent. $\mathcal{R} = \{\mathcal{R}_1, \cdots, \mathcal{R}_n\}$, where $\mathcal{R}_i : \mathcal{S} \times \mathcal{A}_i \to \mathbb{R}$ denotes the reward function of agent i. $\gamma \in [0, 1]$ is the discount factor. $\rho : \mathcal{S} \to [0, 1]$ describes initial state distribution. If it is a fully cooperative setting, agents can share the same reward function (Yu et al., 2022; Rashid et al., 2020; Foerster et al., 2018; Kuba et al., 2021). When $n = 1, \mathcal{O}_i = \mathcal{S}$, the POMG returns to the MDP.

More concretely, in RL, agents interact with the environment through the following procedure: At each discrete time step t, each agent i receives an observation $o_{i,t}$ from the environment and selects an action based on a stochastic policy $\pi_{\theta_i} : \mathcal{O}_i \times \mathcal{A}_i \to [0, 1]$, where θ_i represents the parameters. After receiving the joint actions $\mathbf{a}_t = (a_{1,t}, \cdots, a_{n,t})$, the environment transitions from the current state s_t to a subsequent state s_{t+1} according to the transition function $\mathcal{P}$ and sends a reward $r_{i,t+1}$ to every agent i. The ultimate goal of agent i is to maximize the expected discounted cumulative rewards: $\mathbb{E}_{\pi_{\theta_i}}[\sum_{t=0}^{\infty} \gamma^t r_{i,t}]$.

2.2 Game Theory Concepts

2.2.1 Perfect Information and Imperfect Information

In a game with **perfect information**, only one player moves at a time. Each player comprehensively understands the current game state, the complete history of moves made, and the potential future outcomes. If these conditions are not met, the game is considered to have **imperfect information** (Lucchetti, 2011; Mycielski, 1992). An example of a game with perfect information is Go. In Go, both players have full access to the game state at all times, including the positions of all stones and the identity of the next player. In contrast, Texas Hold'em poker is a game with imperfect information. Players cannot see their opponents' private cards, creating uncertainty and requiring decisions based on probabilistic reasoning, bluffing, and interpreting others' actions.

2.2.2 Normal-Form and Extensive-Form

The normal form and extensive form are two distinct representations of games in game theory. If a game $\mathcal{G}$ is represented in the **normal-form**, it can be expressed by $\mathcal{G} = (\mathcal{N}, \mathbf{\Pi}, \mathbf{u})$. $\mathcal{N} = \{1, 2, \cdots, n\}$ denotes the players. The set $\mathbf{\Pi} = \Pi_1 \times \cdots \times \Pi_n$ represents the pure strategy space for all players. A **pure strategy** specifies a particular and deterministic action for a player in the game. A vector $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_n) \in \mathbf{\Pi}$ is called a **strategy profile**. A **mixed strategy** assigns a probability distribution over the set of pure strategies. For player i, a mixed strategy is represented by $\sigma_i \in \Delta(\Pi_i)$, where Δ denotes the probability simplex. $\mathbf{u} = (u_1, \cdots, u_n)$, where $u_i : \mathbf{\Pi} \to \mathbb{R}$, is a **utility function** that assigns a real-valued **payoff** to each player i. If $\forall \boldsymbol{\pi} \in \mathbf{\Pi}, \sum_i u_i(\boldsymbol{\pi}) = 0$, the game is a **zero-sum** game, otherwise it is a **general-sum** game. If $\Pi_1 = \cdots = \Pi_n$ and the payoffs are invariant under any permutation of the players' strategies, the game is a **symmetric game**. If a finite set of players (especially two players) are involved and each player has a finite set of strategies, a normal-form game can be directly depicted in a matrix.

In *two-player zero-sum symmetric normal-form games*, both players share the same pure strategy space, denoted by Π, such that $\Pi = \Pi_1 = \Pi_2$. Since the utility function satisfies $u_1(\pi_i, \pi_j) = -u_2(\pi_i, \pi_j)$, we can simplify the utility to a single function u, where for $\pi_i, \pi_j \in \Pi$, if π_i beats π_j, then $u(\pi_i, \pi_j) = -u(\pi_j, \pi_i) > 0$. The **evaluation matrix** captures the game outcomes by detailing the results of different strategies when they are played against each other: $A_\Pi = \{u(\pi_i, \pi_j) : \pi_i, \pi_j \in \Pi \times \Pi\}$.

If a game is represented in the **extensive-form**, it is expressed sequentially, illustrating the sequence of moves, choices made by the players, and the information available to each player during decision-making. Typically, a game in the extensive-form is represented by a game tree. This tree demonstrates the sequential and potentially conditional nature of decisions. Moreover, if player i has **perfect recall**, it means that player i remembers which action they have taken in the past. A game $\mathcal{G}$ represented in the extensive-form can be expressed by $\mathcal{G} = (\mathcal{N} \cup \{c\}, H, Z, P, \mathcal{I}, A, \mathbf{u})$. $\mathcal{N} = \{1, 2, \cdots, n\}$ denotes a set of players. c is **chance** node and can be regarded as a special player. H represents a set of possible **histories** and $Z \subseteq H$ is a set of **terminal histories**. Order of moves is represented by a function $P(h) \in \mathcal{N} \cup \{c\}$ to indicate which player is to move, where $h \in H$. $\mathcal{I}$ denotes **information set partitions** and $\mathcal{I}_i$ denotes the information set partitions for player i. This implies that in an imperfect information game, if player i reaches a history $h \in I_i$, where $I_i \in \mathcal{I}_i$ is a specific **information set**, player i cannot distinguish which particular history $h \in I_i$ it is encountering. Action space is represented by $A(h)$ for a non-terminal history $h \in H$. For all non-terminal histories h within an information set I_i, the available actions are the same; otherwise, they are distinguishable. Therefore, we use $A(I_i)$ to represent the available actions for the information set I_i. Utility functions is denoted by $\mathbf{u} = (u_1, \cdots, u_n)$, where $u_i : Z \to \mathbb{R}$. Together, these components define the structure and dynamics of an extensive-form game. Moreover, A strategy profile in an extensive-form game can be expressed by $\boldsymbol{\pi} = (\pi_1, \cdots, \pi_n)$, where π_i maps each $I_i \in \mathcal{I}_i$ to a probability distribution over $A(I_i)$. A **subgame** of an extensive-form game is a portion of the game that starts from a single initial node, includes all successors of any node within the subgame, and contains all nodes in the same information set as any node in the subgame.

The Prisoner's Dilemma is a classic example in game theory. The game outcomes are as follows:

- If one player confesses (C) and the other lies (L), the confessor will serve 1 year in jail, while the liar will serve 8 years.
- If both players choose to confess, they will each serve 7 years behind bars.
- If both players choose to lie, they will each serve only 2 years behind bars.

The classic scenario is known as the *simultaneous* Prisoner's Dilemma, where two players must simultaneously decide whether to confess or lie without knowing the other's choice. Games played in this manner are referred to as **static games**. The normal-form representation is suitable for them, as it captures the simultaneous nature of decision-making (as depicted in Fig. 2a). Another variant is the *sequential* Prisoner's Dilemma, where the second player decides with knowledge of the first player's action. Games of this nature are called **dynamic games**. The extensive-form representation is well-suited for them, as it can clearly illustrate the sequence of moves and the information available to each player at each decision point (as shown in Fig. 2b).

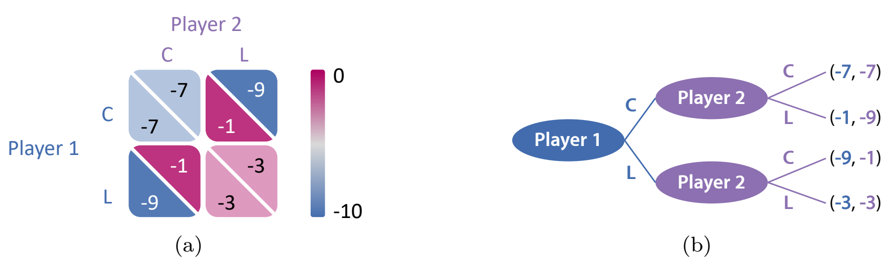

Figure 2: The example of Prisoner's Dilemma. (a) Matrix representation of *simultaneous* Prisoner's Dilemma in normal-form. (b) Game tree representation of *sequential* Prisoner's Dilemma in extensive-form.

Beyond normal-form and extensive-form games, the analysis of complex games often involves a higher-level abstraction: the **meta-game**. The meta-game can be viewed as an advanced form of a normal-form game, but it focuses on the selection of policies (also called strategies) rather than isolated actions. Each player chooses from a set of policies, rather than a set of actions. This set of policies is referred to as the **policy population**. **Meta-strategies** are mixed strategies that assign probabilities to the policies within the policy population in the meta-game.

2.2.3 Transitive Game and Non-transitive Game

For the sake of simplicity, we restrict our focus to *two-player zero-sum symmetric games*. In a **transitive game**, the strategies or outcomes follow a transitive relationship. Formally, $\forall \pi_i, \pi_j, \pi_k \in \Pi$, if $u(\pi_i, \pi_j) > 0$ and $u(\pi_j, \pi_k) > 0$, then it must follow that $u(\pi_i, \pi_k) > 0$. This transitive property simplifies the strategic landscape, allowing for an ordinal ranking of strategies. Conversely, in a **non-transitive game**, $\exists\, \pi_i, \pi_j, \pi_k \in \Pi$ such that $u(\pi_i, \pi_j) > 0$ and $u(\pi_j, \pi_k) > 0$, but $u(\pi_i, \pi_k) \leq 0$. This introduces a cyclic relationship among strategies, thereby complicating the game. The complexity often results in a mixed-strategy equilibrium, where players randomize their choices among multiple strategies to maximize their expected payoff. A classical example of a non-transitive game is Rock-Paper-Scissors, in which no single strategy uniformly dominates all others.

2.2.4 Stage Game and Repeated Game

A **stage game** (or **one-shot game**) is a game that is played only once, namely a one-shot interaction between players. A famous example of a stage game is the Prisoner's Dilemma. A **repeated game** is derived from a stage game played multiple times. Formally, a repeated game based on a stage game $\mathcal{G}$ is defined by playing $\mathcal{G}$ for T periods, where T can be finite or infinite. The strategies in a repeated game are history-contingent, meaning they can depend on the entire sequence of past plays. An example of a repeated game is Texas Hold'em, where players engage in multiple rounds, and each player's strategy may evolve based on previous rounds, betting patterns, and the history of interactions among players.

2.2.5 Nash Equilibrium

For simplicity, π_i denotes the strategy of player i, and π_{-i} denotes the strategies of all players other than player i. Given π_{-i}, player i's **best response (BR)** is the strategy that maximizes player i's payoff:

$$BR_i(\pi_{-i}) = \arg\max_{\pi_i} u_i(\pi_i, \pi_{-i}). \tag{1}$$

A strategy π_i^* is an **ϵ-BR** to strategies π_{-i} if:

$$u_i(\pi_i^*, \pi_{-i}) \geq u_i(BR_i(\pi_{-i}), \pi_{-i}) - \epsilon, \tag{2}$$

where ϵ is a pre-specified threshold.

A strategy profile $(\pi_1^*, \pi_2^*, ..., \pi_n^*)$ is a **Nash equilibrium (NE)** if, for each player i:

$$u_i(\pi_i^*, \pi_{-i}^*) \geq u_i(\pi_i, \pi_{-i}^*), \forall \pi_i, \tag{3}$$

meaning that no player can benefit by changing their strategy unilaterally, given the strategies of others. In other words, an NE is a situation where each player's strategy is a BR to the others' strategies.

A strategy profile $(\pi_1^*, \pi_2^*, ..., \pi_n^*)$ is an **ϵ-NE** if, for each player i:

$$u_i(\pi_i^*, \pi_{-i}^*) \geq u_i(\pi_i, \pi_{-i}^*) - \epsilon, \forall \pi_i, \tag{4}$$

meaning that no player can increase their payoff by more than ϵ by unilaterally changing their strategy.

However, computing NE is generally intractable in complex games, so some researchers utilize α-Rank (Omidshafiei et al., 2019) and Correlated Equilibrium (CE) (Aumann, 1974) as alternatives. Some studies also resort to Replicator Dynamics (Taylor & Jonker, 1978) to analyze strategy evolution.

3 Algorithms

Based on existing self-play work (Hernandez et al., 2021; Lanctot et al., 2017; Liu et al., 2022c; Garnelo et al., 2021), we propose a self-play framework (Algo. 1) that boasts enhanced expressivity and superior generalization capabilities. For simplicity, our framework is illustrated for *symmetric games*. All players share a policy population with a fixed maximum size. In each iteration, a newly initialized policy is trained, while opponent policies are sampled from the population. After training, the new policy is added to the population. The updated policy population is then evaluated, and then the opponent sampling strategy is recalculated for the next iteration.

This section is organized as follows: In Sec.3.1, we provide a formalized description of our framework. We then categorize self-play algorithms into four primary groups: traditional self-play algorithms (Sec. 3.2), the PSRO series (Sec. 3.3), the ongoing-training-based series (Sec. 3.4), and the regret-minimization-based series (Sec. 3.5). We analyze how these four categories align with our framework and introduce corresponding algorithms for each category. To make it more straightforward, we highlight the representative classic algorithms from these and present them in Table 1 for comparison within our framework. In Sec. 3.6, we compare the four categories, explain our classification rationale, and highlight the distinctions between our framework and existing ones to demonstrate its greater generality.

3.1 Framework Definition

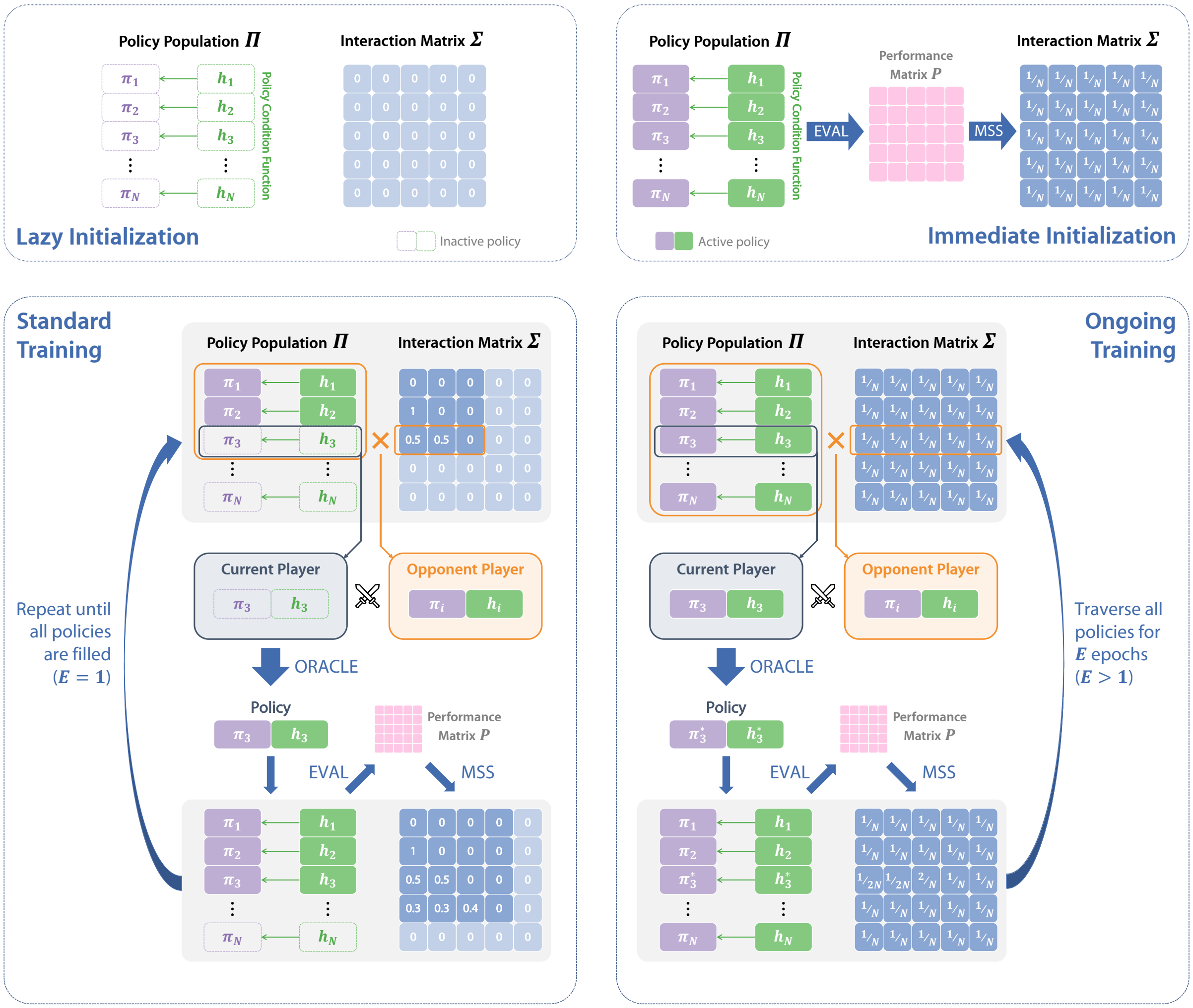

Figure 3: Illustration of our framework. The top row shows two different initialization methods for the policy population Π and the interaction matrix Σ: lazy initialization and immediate initialization (corresponding to Line 1 in Algo. 1). The bottom row illustrates two training paradigms: standard training and ongoing training (corresponding to Lines 2~9 in Algo. 1). We categorize self-play algorithms into four types: traditional self-play, the PSRO series, the ongoing-training-based series, and the regret-minimization-based series. Among these, traditional self-play, the PSRO series, and the regret-minimization-based series utilize lazy initialization and standard training, whereas the ongoing-training-based series employs immediate initialization and ongoing training. For a detailed description and analysis, please refer to Sec.3 and Table1.

In Algo. 1, we define a unified self-play framework based on Liu et al. (2022c); Lanctot et al. (2017); Garnelo et al. (2021); Hernandez et al. (2021). We combine visual illustrations (Fig. 3) with detailed textual descriptions to explain the key processes of our framework. Next, we provide an in-depth explanation of these processes:

- $\Pi := \{\pi_i(\cdot|h(i))\}_{i=1}^N$: Each policy π_i in the **policy population** Π is conditioned on a **policy condition function** $h(i)$, which provides supplementary information for certain algorithms. We denote the policy $\pi_i(\cdot|h(i))$ as π_i^h. Note that i refers to the ith policy in the population, not the ith player, and N denotes the **policy population size**, not the number of players. Π can be initialized (Line 1 in Algo. 1) in two ways: lazy initialization and immediate initialization. In **lazy initialization**, Π starts with N placeholder policies, with the actual policies being initialized during

Algorithm 1 A unified framework of self-play.

```
1: Initialize Π, Σ
2: for e ∈ [[E]] do
3:   for σ_i ∈ Σ do
4:     Initialize π_i^h
5:     π_i^h ← ORACLE(π_i^h, σ_i, Π)          ▷ Compute the oracle.
6:     P ← EVAL(Π)                            ▷ Get policies' performance.
7:     Σ ← MSS(P)                             ▷ Update the interaction matrix.
8:   end for
9: end for
10: return Π, Σ
```

the training iterations (Line 4 in Algo. 1). In **immediate initialization**, Π is initialized upfront with N real policies, which may be randomly generated or pre-trained models. A policy π_i^h is considered **inactive** if it is a placeholder; otherwise, it is considered **active**.

- $\Sigma := \{\sigma_i\}_{i=1}^N \in \mathbb{R}^{N \times C_1}$: The **interaction matrix** Σ consists of rows σ_i, where each $\sigma_i \in \mathbb{R}^{C_1}$ represents the **opponent sampling strategy** of policy i, indicating how to sample opponents' policies. The dimension C_1 varies according to the sampling method. For instance, σ_i can represent the probability distribution over opponent policies in the policy population Π, where $C_1 = N^{n-1}$ and n denotes the number of players. Alternatively, σ_i can be viewed as parameters of a sampling network. Specially, in a two-player game, if $C_1 = N$ and σ_{ij} represents the probability that policy i is optimized against policy j, Σ can be depicted in **directed interaction graphs** (Fig. 4). Σ will be initialized (Line 1 in Algo. 1) based on the initialization of Π. If Π uses lazy initialization, Σ will be initialized as an empty matrix. If Π uses immediate initialization, Σ will be calculated based on Π. We will discuss this in more detail later.

- For epoch $e \in [[E]]$ (Line 2 in Algo. 1): E denotes the total number of epochs for the entire policy population. For example, if the algorithm only introduces new policies into the population without updating existing ones, then $E = 1$. This implies that only the *inactive* policy is likely to be chosen for training in each iteration and turned into an *active* one, while *activa* policies remain unchanged. Conversely, if *active* policies are updated multiple times throughout the algorithm, then $E > 1$. Indeed, E accurately reflects the number of updates performed.

- Initialize π_i^h (Line 4 in Algo. 1): The initialization of π_i^h can vary depending on the algorithm being used. It can be initialized randomly or by leveraging pre-trained models (Silver et al., 2016; Vinyals et al., 2019) or through a recently updated policy.

- ORACLE(π_i, σ_i, Π) (Line 5 in Algo. 1): The ORACLE is an abstract computational entity that returns a new policy adhering to specific criteria. Here, we divide ORACLE into three types. (1) One type is the BR oracle, which is designed to identify the optimal counter-strategies against an opponent's strategy, including finding NE (McMahan et al., 2003). However, it often requires considerable computational effort. (2) To alleviate the computational demands, the approximate best response (ABR) oracle is introduced, using techniques such as RL (Algo. 2), evolution-theory-based methods (Algo. 3) or regret minimization methods (Algo. 4). (3) Other specially crafted ORACLEs are either tailored to specific meta-strategy solvers (MSSes) (Muller et al., 2020; Marris et al., 2021) or introduced to enhance diversity (Balduzzi et al., 2019; Perez-Nieves et al., 2021; Liu et al., 2021). Some details of Algo. 2, 3 and 4 need to be mentioned. Algo. 2 is a RL method that can be widely used across different categories. Moreover, off-policy RL algorithms typically require a replay buffer to gather samples; however, they are not explicitly included for simplicity. Algo. 3 is specifically utilized by Regularized Nash Dynamics (R-NaD) (Perolat et al., 2021) and Algo. 4 is specifically utilized by the regret-minimization-based series. They will be introduced in detail in Sec. 3.3.5 and Sec. 3.5 respectively.

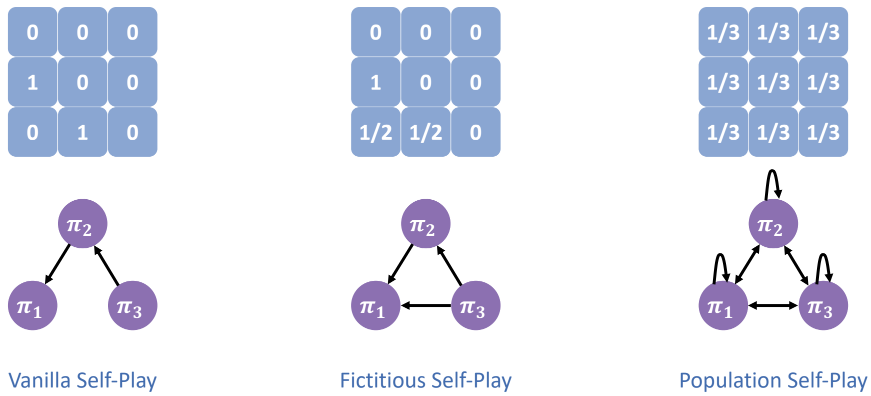

Figure 4: Interaction matrix Σ examples. When $C_1 = N$ and the opponent sampling strategy σ_{ij} represents the probability that policy i is optimized against policy j, the interaction matrix Σ can be depicted in directed interaction graphs. Here, we consider three representative self-play algorithms. In the **Top** section, we define the interaction matrix $\Sigma \in \mathbb{R}^{3\times 3}$ as $\{\sigma_i\}_{i=1}^{3}$. In the **Bottom** section, we present directed interaction graphs where the outgoing edges from each node are equally weighted, and their weights collectively sum to one. The relationship between the **Top** and **Bottom** sections is established through directed edges: an edge directed from node i to node j with a weight of σ_{ij} signifies that policy i is optimized against policy j with a probability of σ_{ij}. Note that this figure is reproduced from Liu et al. (2022c) and this concept is initially proposed by Garnelo et al. (2021).

- EVAL(Π) (Line 6 in Algo. 1): Evaluating the policy population Π. There are multiple evaluation metrics available to assess the performance of each policy. The performance matrix is represented as $\mathcal{P} := \{p_i\}_{i=1}^{N} \in \mathbb{R}^{N\times C_2}$. p_i is the performance of policy i, and the dimension C_2 depends on the evaluation metric used. For instance, p_i can be depicted as the relative skill like Elo ratings ($C_2 = 1$) or can be depicted as the payoff tensor ($C_2 = N^{n-1}$, where n is the number of players). Specially, in two-player symmetric zero-sum games, the expected payoffs can serve as the evaluation metric. In such cases, $\mathcal{P}$ is a square matrix ($C_2 = N$).

- $MSS : \mathbb{R}^{N\times C_2} \rightarrow \mathbb{R}^{N\times C_1}$ (Line 7 in Algo. 1): A **meta-strategy solver (MSS)** MSS takes the **performance matrix** $\mathcal{P}$ as its input and produces a new interaction matrix Σ as its output. If the MSS produces a constant matrix irrespective of the input, the evaluation step (Line 6 in Algo. 1) can be skipped to reduce computations. Moreover, if Π uses immediate initialization, Σ will be initialized (Line 1 in Algo. 1) by first evaluating the performance and then applying the MSS.

3.2 Traditional Self-play Algorithms

Traditional self-play algorithms involve agents improving their strategies by repeatedly playing against themselves, allowing them to explore various strategies and enhance their decision-making abilities without external input. The simplest form involves agents training against their most recent version to identify weaknesses. Other approaches involve training against a set of strategies from different iterations, enabling agents to develop robust and adaptive strategies. This section will explain how traditional self-play algorithms fit into our framework and introduce representative traditional self-play methods, ranging from simpler forms to more complex ones.

3.2.1 Integration into Our Framework

Traditional self-play algorithms can be incorporated into our proposed framework (Algo. 1) with the following settings. **First**, the policy population Π utilizes lazy initialization because the policy population in traditional self-play algorithms is intended to grow with each iteration. **Second**, we set $E = 1$ because only the *inactive*

Algorithm 2 Compute the oracle in RL.

Require: π_i^h ▷ Policy i is being trained.
Require: σ_i ▷ Opponent sampling strategy of policy i.
Require: Π ▷ Policy population.

```
while π_i^h is not valid do
    for trajectory τ ∈ [[τ_max]] do
        sample π_opp^h ~ P(σ_i)                ▷ Policies of opponents.
        π = (π_i^h, π_opp^h)
        s_0, o_0 ~ ρ                           ▷ Initial state.
        for t ∈ [[t_max]] do
            a_t ~ π(o_t)
            s_{t+1}, o_{t+1} ~ P(s_t, a_t)
            r_t ← R(s_t, a_t)
        end for
        π_i^h ← update(π_i^h)                  ▷ Using RL algorithms
    end for
end while
return π_i^h
```

Algorithm 3 Compute the oracle in evolution theory.

Require: π_i^h ▷ Policy i is being trained.
Require: σ_i ▷ Opponent sampling strategy of policy i.
Require: Π ▷ Policy population.

```
sample π_opp^h ~ P(σ_i)                        ▷ Policies of opponents.
π_reg = (π_i^h, π_opp^h)                       ▷ Regularization policies.
Transformed the reward according to π_reg.
π_i^h ← Use replicator dynamics to play the reward-transformed game until convergence.
return π_i^h
```

Algorithm 4 Compute the oracle in regret matching.

Require: π_i^h ▷ Policy i is being trained.
Require: σ_i ▷ Opponent sampling strategy of policy i.
Require: Π ▷ Policy population.

```
for each player j do
    sample π_opp^h ~ P(σ_i)                    ▷ Policies of opponents.
    π = (π_i^h(j), π_opp^h(−j))
    Use regret matching to play the game and update h(i) with new regret minimization information.
end for
return π_i^h
```

policy can be trained in each iteration, then turning into an *active* policy. Here, the policy population size N serves as the upper limit for the number of *active* policies in the population. In other words, we use N iterations to optimize the policy. **Third**, the strategy being trained π_i^h can be initialized in a general manner. For instance, the strategy can be initialized randomly, learning from scratch. More often, π_i^h is initialized by $\pi_{i-1}(\cdot|h(i-1))$, allowing incremental learning and adaptation based on the most current trained policy to accelerate convergence. **Fourth**, the policies in traditional self-play algorithms don't need supplementary information, we set the policy condition function $h(i) = \emptyset$. **Fifth**, the MSSes of traditional self-play algorithms are straightforward, often yielding a constant interaction matrix Σ that eliminates performance evaluation. Only the MSS for prioritized fictitious self-play (described in Sec. 3.2.5) requires the performance matrix; nonetheless, it does not need to solve for complex game outcomes like NE.

Next, we outline the traditional self-play schemes. For simplicity, we operate under the following assumption:

Assumption 1. *In a two-player symmetric game, $C_1 = N$, with σ_{ij} denoting the probability that policy i is optimized in response to policy j which leads to $\sum_{j=1}^{N} \sigma_{ij} = 1,\ \forall i$.*

Based on Assumption 1, we can further deduce the following important corollary:

Corollary 1. *In traditional self-play algorithms, the interaction matrix Σ is a lower triangular matrix.*

Proof. The policy population gradually increases over time. When policy i is selected for training, only policy j $(j \leq i)$ has already been trained and holds meaningful outcomes. Other policies are *inactive* policies. As a result, we exclusively select policy j, where $j \leq i$, to serve as the opponent for policy i. □

3.2.2 Vanilla Self-play

In vanilla self-play (Samuel, 2000), agents are trained by competing against their latest versions. Thus, the MSS of vanilla self-play:

$$MSS(\mathcal{P})_{ij} = \begin{cases} 1, \ \text{if } j = i-1 \\ 0, \ \text{otherwise} \end{cases}. \tag{5}$$

Regardless of $\mathcal{P}$, the MSS produces the same interaction matrix. Although this MSS is straightforward, it is utilized in many other algorithms, so we refer to it as **vanilla MSS**. Vanilla self-play is effective in transitive games, but it can lead the agent to cyclic learning patterns in non-transitive games. It may also lead to an unstationary problem, causing the system to get stuck in a local optimum.

A more modern version of vanilla self-play is to combine it with off-policy RL. In each iteration, the policy to be trained, π_i^h, is initialized using the previous policy $\pi_{i-1}(\cdot|h(i-1))$. Typically, each player's policy is fixed, and samples are collected into a buffer by vanilla self-play. These samples remain useful for training, even if they were generated by an earlier policy. This allows the agent to leverage past experiences and maintain more stationary training. Once the policy is updated, it will be added to the policy population.

3.2.3 Fictitious Self-play

Fictitious Play (FP) (Brown, 1951) is a learning algorithm in game theory where each player best responds to the empirical frequency of the strategies used by the opponent. If the opponent's strategy is static, FP can find the NE. Based on FP intuition, Fictitious Self-Play (FSP) (Heinrich et al., 2015) is introduced to make agents play against past versions of themselves to learn optimal strategies to improve the robustness of vanilla self-play. Neural Fictitious Self-Play (NFSP) (Heinrich & Silver, 2016) is a modern variant that combines FSP with deep learning techniques. It uses neural networks to approximate the BRs. In the original versions of NFSP and FSP, two distinct types of agent memory are used: one to record the agent's own behavior and the other to capture the opponent's behavior.

In more recent approaches (Lanctot et al., 2017), random sampling is frequently employed to approximate the opponents' average strategy, eliminating the need to maintain two separate types of agent memory. Thus, the MSS of FSP:

$$MSS(\mathcal{P})_{ij} = \begin{cases} \frac{1}{i-1}, \ \text{if } j \leq i-1 \\ 0, \ \text{otherwise} \end{cases}. \tag{6}$$

In FSP, the MSS continues to generate a constant interaction matrix. Compared to vanilla self-play, this approach enhances the robustness of the policy by sampling older versions of its own policies to be used as opponent strategies.

3.2.4 δ-uniform Self-play

δ-uniform self-play, introduced by Bansal et al. (2018), uses the hyper-parameter δ, ranging from 0 to 1, to select the most recent $1-\delta$ percentage of policies for uniform sampling to generate opponent policies. When policy i is in the training phase, according to Corollary 1, only the previous $i-1$ policies are relevant. To select opponents for policy i, we sample from the range of policies between $[\lceil \delta(i-1) \rceil, i-1]$, where $\lceil \cdot \rceil$ denotes the ceiling function, which rounds up the input to the nearest integer greater than or equal to it. When $\delta = 0$, the system retains the complete historical memory, whereas $\delta = 1$ implies that only the most recent policy is utilized. Thus, the MSS of δ-uniform self-play:

$$MSS(\mathcal{P})_{ij} = \begin{cases} \frac{1}{f(i)}, \text{ if } \lceil \delta(i-1) \rceil \leq j \leq i-1 \\ 0, \text{ otherwise} \end{cases}, \tag{7}$$

$$f(i) = i - \lceil \delta(i-1) \rceil. \tag{8}$$

In δ-uniform self-play, The MSS generates a constant interaction matrix. *Specially*, if $\delta = 0$, it corresponds to FSP, and if $\delta = 1$, it corresponds to vanilla self-play.

3.2.5 Prioritized Fictitious Self-play

Prioritized Fictitious Self-Play (PFSP) (Vinyals et al., 2019) leverages a priority function to allocate a higher probability of selection to agents with higher priorities. Here, $\mathcal{P}$ represents the winning rates, specifically defined as $\mathcal{P}_{ij} = \text{Prob}(\pi_i \text{ beats } \pi_j)$. The MSS of PFSP is given by Algo. 5. The function $f : [0,1] \to [0, \infty)$ is a priority function. For example, $f(x) = (1-x)^p$ with $p > 0$ indicates that stronger policies against the currently being trained policy have a higher chance of being chosen as opponents. Alternatively, $f = x(1-x)$ implies that players of similar levels are more likely to be chosen as opponents. Additionally, in broader terms, $\mathcal{P}$ can also be assessed by other metrics. A similar MSS in Berner et al. (2019) can be utilized to allocate higher probabilities to strategies that perform better or are more challenging to defeat.

Algorithm 5 PFSP meta-strategy solver.

function $\mathcal{F}(\mathcal{P})$
  $\sigma_{i+1,j} \leftarrow \frac{f(\mathcal{P}_{i,j})}{\sum_{j \leq i} f(\mathcal{P}_{i,j})}$, if $j \leq i$
  Append zeros to σ_{i+1} until its length is N.
return Σ

3.3 PSRO Series

Similar to traditional self-play algorithms, the PSRO series of algorithms starts with a single policy and gradually expands the policy space by incorporating new oracles. These oracles are policies that approximate optimal responses to the current meta-strategies of other agents.

3.3.1 Integration into Our Framework

The PSRO series of algorithms can also be integrated into our proposed framework (Algo. 1). **First**, similar to traditional self-play algorithms, we also utilize lazy initialization to initialize Π. **Second**, we also set $E = 1$ and N can be considered as the upper limit for the policy population size in the original PSRO algorithms. **Third**, in the context of the PSRO series of algorithms, the strategy of our player π_i^h can also be initialized in a general manner. **Fourth**, we set $h(i) = \emptyset$ since the PSRO series of algorithms do not use any policy condition function for their policies. **Fifth**, it's crucial to highlight that our framework diverges from the traditional PSRO model (Lanctot et al., 2017) in how σ_i is defined. In contrast to being the meta-strategy for policy, in our framework, σ_i is the opponent sampling strategy. It means that σ_i here represents the opponent's meta-strategy against policy i for the PSRO series of algorithms. **Sixth**, compared with traditional self-play methods, the MSSes of the PSRO series are often more complex. For example, some MSSes incorporate concepts from different types of game equilibria (McMahan et al., 2003; Muller et al., 2020; Marris et al., 2021). **Lastly**, we also simply follow Assumption 1. We can derive the Corollary 2 using a similar proof as Corollary 1.

Corollary 2. *In the PSRO series of algorithms, the interaction matrix Σ is a lower triangular matrix.*

3.3.2 Double Oracle

Double Oracle (DO) (McMahan et al., 2003) can be only applied to two-player normal-form games. In this context, we can utilize the payoff matrix as the evaluation matrix. The interaction matrix can be initialized with all zeros, reflecting the initial absence of interactions between strategies. The MSS of DO can then be outlined as described in Algo. 6. The opponent sampling strategy σ_i corresponds to the opponent's NE strategy of the restricted game. Therefore, the oracle in DO is a BR rather than an ABR, computing the BR against the current NE opponent strategy of the restricted game. In the context of two-player normal-form games, DO theoretically can achieve the NE of the full game.

Algorithm 6 NE-based meta-strategy solver.

```
function F(P)
    σ_{i+1} ← SOLVE-NASH(P_{1:i,1:i})  ▷ Opponent's NE meta-strategy.
    Append zeros to σ_{i+1} until its length is N.
return Σ
```

We take rock-paper-scissors as an example, a symmetric two-player game. The policy condition function $h(i) = \emptyset$. The policy population size $N = 4$. The policy population Π uses lazy initialization and the interaction matrix Σ is initialized with all zeros. Assume that in the **first** iteration, the policy π_1 is set to "rock". Obviously, the NE-based MSS returns $\sigma_2 = (1, 0, 0, 0)$. In the **second** iteration, the BR oracle adds "paper" to Π as π_2. Consequently, the NE-based MSS yields $\sigma_3 = (0, 1, 0, 0)$, since "paper" always beats "rock." In the **third** iteration, noting that σ_3 selects only "paper" and never "rock," the BR oracle adds "scissors" as π_3. The NE-based MSS then solves the game and returns $\sigma_4 = (1/3, 1/3, 1/3, 0)$. In the **final** iteration, the BR oracle adds a mixed strategy that assigns a uniform probability over "rock," "paper," and "scissors" as π_4. The final interaction matrix is:

$$\Sigma = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 \end{bmatrix} \tag{9}$$

This process demonstrates how the DO algorithm iteratively expands the strategy set within our framework. Also, using DO as an example, we show that, compared with traditional self-play algorithms, the PSRO series employs more sophisticated MSSes to better capture the essence of the game, albeit at a higher computational cost.

3.3.3 PSRO

PSRO (Lanctot et al., 2017) extends DO to more complex games beyond just two-player normal-form games. This approach first introduces the concept of the MSS, which is a broader concept than simply computing NE. The MSS framework allows for a more flexible representation of strategic solutions in various game settings. Many variants of PSRO focus on designing new MSSes to better capture different aspects of strategic play in these more complex games. In the original PSRO framework, the oracle is computed using RL techniques, similar to those described in Algorithm 2. This allows the algorithm to effectively handle large and intricate strategy spaces, making it applicable to various game scenarios.

3.3.4 PSRO Variants

The traditional PSRO algorithm has been the subject of numerous extensions in recent research. Some studies focus on making PSRO more computationally tractable in different scenarios or achieving fast convergence. By establishing a hierarchical structure of RL workers, Pipeline PSRO (McAleer et al., 2020) achieves the parallelization of PSRO and simultaneously provides guarantees of convergence. Efficient PSRO (Zhou et al., 2022) introduces the unrestricted-restricted (URR) game to narrow the selection of opponent policies, thereby preventing the need for meta-game simulations in traditional PSRO. Similar to Pipeline PSRO, Efficient PSRO is equipped to solve URR in parallel. In addition, unlike traditional PSRO, which confines the integration of population strategies to the game's initial state, Extensive-Form Double Oracle (XDO) (McAleer et al., 2021) allows this integration across all information states. It ensures a linear convergence to an approximate NE based on the number of information states, enhancing its tractability in extensive-form games. Online Double Oracle (ODO) (Dinh et al., 2022) integrates the no-regret analysis of online learning with DO to improve both the convergence rate to an NE and the average payoff. Anytime PSRO (McAleer et al., 2022b) and Self-play PSRO (McAleer et al., 2022a) are designed to incorporate less exploitable policies into the policy population, facilitating faster convergence. Moreover, as the number of agents grows, determining BRs becomes exponentially challenging. Mean-field PSRO (Muller et al., 2022) has been introduced to address this complexity in mean-field games. In addition, due to the computational intractability of solving NE in multi-player general-sum games (Daskalakis et al., 2009; Daskalakis, 2013) and the selection problem in NE (Goldberg et al., 2013), Müller et al. put forward α-PSRO (Muller et al., 2020). Instead of NE, α-rank (Omidshafiei et al., 2019), which is unique and polynomial-time solvable in multi-player general-sum games, is introduced in the MSS and a preference-based best response (PBR) oracle is incorporated in the approach. Similar to α-PSRO, Marris et al. (2021) proposes Joint Policy-Space Response Oracle (JPSRO) to tackle multi-player general-sum games. It utilizes CE and coarse correlated equilibrium (CCE) as alternatives to NE to make it computationally tractable in multi-player general-sum games and designs the CE and CCE-based MSS.

Other research focuses on policy diversification, as deriving a single policy in transitive games is less meaningful. Balduzzi et al. (2019) introduces an open-ended framework in two-player zero-sum games. This framework enhances the diversity of the strategy population and introduces a gamescape, which geometrically represents the latent objectives in games for open-ended learning. The study proposed two algorithms: Nash response PSRO_{N} and rectified Nash response PSRO_{rN}. Both algorithms utilize an asymmetric payoff matrix as their performance evaluation metric. Similarly to DO, they employ the Nash-based MSS (Algo. 6). Compared to PSRO_{N}, PSRO_{rN} incorporates an additional step within the ABR oracle to focus on the opponents they beat or tie and ignore the opponents they lose to. Perez-Nieves et al. (2021) uses determinantal point process (Kulesza et al., 2012) to assess diversity and introduces diverse PSRO by incorporating a diversity term into the PSRO oracle. This modification can also be easily implemented in FP and α-PSRO. Similarly, Liu et al. (2021) introduces the concepts of behavioral diversity and response diversity and incorporates them into the PSRO oracle. Policy Space Diversity PSRO (Yao et al., 2023) defines a diversity metric named population exploitability that helps to achieve a full-game NE.

3.3.5 R-NaD

Although R-NaD (Perolat et al., 2021) is initially described as leveraging evolutionary game theory with a regularization component. Here, we categorized it into the PSRO series with a unique oracle computation technique (Algo. 3), executed in two stages: the first stage involves transforming the reward based on the regularization policy to make it policy-dependent, and the second stage applies replicator dynamics (Taylor & Jonker, 1978) until convergence to a fixed point. The oracle added to the policy population Π in each iteration is derived from the reward-transformed game, not the original problem's oracle. Nonetheless, this approach ensures that the policy converges to the NE of the original game when the game is monotone. The MSS of R-NaD is the vanilla MSS, as described in Equ. (5). This equation illustrates that the fixed point reached in each iteration, the oracle, is utilized as the regularization policy for the next iteration.

3.4 Ongoing-training-based Series

In the PSRO series of algorithms, two key challenges arise. First, truncating the ABR operators during each iteration is often necessary when operating with a limited budget, introducing sub-optimally trained responses into the population. Second, relearning basic skills in every iteration is redundant and becomes untenable when confronted with increasingly formidable opponents (Liu et al., 2022c). To address these challenges, the ongoing-training-based series of algorithms emphasizes the repeated ongoing training of all policies. In other words, instead of only *inactive* policies being selected for training, all *active* policies are also likely to be chosen for training.

3.4.1 Integration into Our Framework

We can incorporate these ongoing-training-based series of algorithms into our proposed framework (Algo. 1) using the following settings: **First**, we use immediate initialization to initialize Π because, in the ongoing-training-based series, all policies in the policy population are trained together, rather than the policy population growing with each iteration. **Second**, we set $E = E_{max} > 1$, which represents the maximum number of epochs to optimize each policy within the policy population. In other words, each unique policy undergoes iterative training for E_{max} times. **Third**, since each policy undergoes training for E_{max} times, we utilize $\pi_i(\cdot|h(i))$ to initialize π_i^h. This means that policy updates are self-referential. **Lastly**, under Assumption 1, different from Collory 1 and 2, due to the continuous training process of all policies, we derive Collory 3.

Corollary 3. *In the ongoing-training-based series of algorithms, the interaction matrix Σ is generally* ***not*** *a lower triangular matrix.*

Proof. When policy i is selected for training, policy k ($k \geq i$) has already been actually initialized and is therefore considered an *active* policy. Furthermore, in epoch e ($e > 1$), policy k has already been updated and holds significant meaning. As a result, policy k, where $k \geq i$, is likely to be chosen as the opponent for policy i. Consequently, the interaction matrix Σ is generally **not** a lower triangular matrix. □

3.4.2 FTW

Quake III Arena Capture the Flag is a renowned 3D multi-player first-person video game where two teams vie to capture as many flags as possible. The For The Win (FTW) agent (Jaderberg et al., 2019) is designed to perform at human-level proficiency in this game. A pivotal aspect of FTW is its employment of the ongoing-training-based self-play in RL. Specifically, it trains a population of N different policies in parallel, which compete and collaborate with each other. When policy i is undergoing training, FTW samples both its teammates and adversaries from the population. *Specially*, in scenarios where each team comprises a single member, it can be seamlessly integrated into our framework using the subsequent MSS:

$$MSS(\mathcal{P})_{ij} = \frac{1}{N}. \tag{10}$$

This essentially means that the interaction graph is densely connected. Moreover, all policies draw upon a unified policy network parameterized by ϕ. Hence, $\pi_i(\cdot|h(i))$ can be aptly depicted as $\pi_\phi(\cdot|h(i))$. Furthermore, since these policies are not conditioned on external parameters, it is straightforward to represent the conditioning function $h(i) = \emptyset$.

3.4.3 NeuPL

Neural Population Learning (NeuPL) (Liu et al., 2022c) introduces another critical innovation: it employs a unified conditional network, where each policy is adjusted against a specific meta-game mixture strategy. This is instrumental in enabling transfer learning across policies. Owing to NeuPL's reliance on a unified conditional network parameterized by θ, $\pi_i(\cdot|h(i))$ can be succinctly represented as $\pi_\theta(\cdot|h(i))$. Since NeuPL policies depend on the opponent sampling strategy σ_i, we define $h(i) = \sigma_i$.

3.4.4 Simplex-NeuPL

Simplex-NeuPL (Liu et al., 2022a), which builds on NeuPL, is designed to achieve *any-mixture optimality*, which signifies that the formulated policy should exhibit flexibility across a diverse spectrum of adversaries, including those that might not present equivalent competitive prowess. To model the population learning process from a geometric perspective, Simplex-NeuPL introduces the concept of the population simplex. Analogously to its predecessor, Simplex-NeuPL integrates a conditional network to characterize the policy, represented as $\pi_\theta(\cdot|h(i))$ conditioned on the opponent sampling strategy $h(i) = \sigma_i$. Intriguingly, there is a slight possibility that σ_i does not originate from the MSS. Instead, it is drawn as a sample from the population simplex. This sampling mechanism results in greater robustness.

3.5 Regret-minimization-based Series

Another line of self-play algorithms is based on regret minimization. Unlike other categories, this series prioritizes accumulated payoff over time rather than focusing solely on a single episode, which is common in repeated games. For example, in games like Texas Hold'em or Werewolf, players must use deception, concealment, and bluffing to aim for overall victories rather than just winning a single game. Traditional regret-minimization-based self-play typically doesn't use RL, but later studies have combined them for improved performance. This section will also discuss traditional regret-minimization methods to lay the groundwork for understanding their integration with RL.

3.5.1 Integration into Our Framework

We can also integrate the regret-minimization-based series of algorithms into our proposed framework (Algo. 1) with the following settings: **First**, similar to traditional self-play algorithms and the PSRO series, we use lazy initialization to initialize the policy population Π. **Second**, we set $E = 1$, and N is regarded as the maximum iteration to optimize the policy. **Third**, we initialize π_i^h using $\pi_{i-1}(\cdot|h(i-1))$ to utilize the most current trained policy. More specifically, $h(i) = h(i-1)$ and $\pi_i^h = \pi_{i-1}^h$. **Fourth**, in each iteration i, $h(i)$ represents the specific elements that regret-minimization-based self-play algorithms need to store. Note that this category relies heavily on the information in $h(i)$. For instance, in Counterfactual Regret Minimization (CFR) (Zinkevich et al., 2007), it is necessary to store counterfactual regrets for every action in every information set for each player within $h(i)$. Once $h(i)$ is determined, the corresponding policy is also defined. We will discuss CFR in detail in Sec. 3.5.2. **Fifth**, the ABR operator (Algo. 4) incorporates regret-related information into $h(i)$. Unlike the original CFR, Algo. 4 updates the regrets of each player sequentially. This means that the regrets of player 2 are updated after considering the already-updated regrets of player 1. This adjustment has been shown to accelerate convergence empirically and possesses a theoretical error bound (Burch et al., 2019). Additionally, each π_i^h represents the strategies for all players, and in iteration i, player j uses the strategy $\pi_i^h(j)$. **Lastly**, the MSS of the regret-minimization-based series is the vanilla MSS, as described in Equ. (5). Also, we can derive Collory 4.

Corollary 4. *In the regret-minimization-based series of algorithms, the interaction matrix Σ is a lower triangular matrix. More specifically, it is a **unit lower shift matrix**, with ones only on the subdiagonal and zeroes elsewhere.*

Proof. Regret minimization-based algorithms always use the latest strategy for training. In other words, at iteration i, π_{i-1}^h is consistently chosen as the opponent policy. Consequently, the interaction matrix Σ is a unit lower shift matrix. □

3.5.2 Vanilla CFR

Regret measures the difference between the best possible payoff and the actual payoff. The regret matching algorithm (Hart & Mas-Colell, 2000) optimizes decisions by selecting strategies based on accumulated positive **overall regrets**, with higher overall regret strategies being more likely chosen to correct past underperformance. After each round, players update the overall regret values for each strategy. Strategies will converge to an approximate NE. However, this method primarily applies to normal-form games because computing overall regret in extensive-form games is challenging.

Zinkevich et al. (2007) proposes CFR for extensive-form games. Here, it is referred to as vanilla CFR to distinguish it from later advancements. Vanilla CFR maintains the strategy and the **immediate counterfactual regrets** for each information set. Theoretically, this regret is specific to each information set, and the aggregation of these immediate counterfactual regrets is an upper bound for the overall regret. Therefore, the problem can be decomposed into numerous more minor regret minimization problems.

Next, there is a more detailed description of vanilla CFR. In this survey, π_i denotes the strategy at iteration i for consistency, whereas the original paper uses σ. Furthermore, we use j to represent player j, and $\pi_i(j)$ denotes the policy used by player j at iteration i. The **counterfactual value** $v_j(\pi, I)$, which represents the expected value upon reaching the information set I when all players, except for player j, adhere to the strategy π:

$$v_j(\pi, I) = \sum_{h \in I, h' \in Z} \text{Prob}_{-j}^{\pi}(h) \text{Prob}^{\pi}(h, h') u_j(h'). \tag{11}$$

It is an unnormalized form of the **counterfactual utility** in the original paper. Based on this counterfactual value, the immediate counterfactual regret is defined as:

$$R_{j,\text{imm}}^{T}(I) = \frac{1}{T} \max_{a \in A(I)} \sum_{i=1}^{T} \left(v_j(\pi_i|_{I \to a}, I) - v_j(\pi_i, I) \right), \tag{12}$$

where $\pi_i|_{I \to a}$ denotes player j choose action a with probability 1 at iteration i. Moreover, the **positive immediate counterfactual regret** is:

$$R_{j,\text{imm}}^{T,+}(I) = \max(R_{j,\text{imm}}^{T}(I), 0). \tag{13}$$

The player j's average overall regret R_j^T is defined as:

$$R_j^T = \frac{1}{T} \max_{\pi(j)^* \in \Pi(j)} \sum_{i=1}^{T} \left(u_j(\pi(j)^*, \pi_i(-j)) - u_j(\pi_i) \right). \tag{14}$$

Theoretically, average overall regret R_j^T is bounded by the sum of positive immediate counterfactual regrets:

$$R_j^T \leq \sum_{I \in \mathcal{I}_j} R_{j,\text{imm}}^{T,+}(I). \tag{15}$$

Thus, the problem can be simplified from minimizing the overall regret to minimizing the immediate counterfactual regrets in each information set separately. For simplicity, we often drop "immediate" in discussions and refer directly to **counterfactual regrets**. The counterfactual regret can also be recorded as follows:

$$R_j^T(I, a) = \frac{1}{T} \sum_{i=1}^{T} \left(v_j(\pi_i|_{I \to a}, I) - v_j(\pi_i, I) \right), \tag{16}$$

$$R_j^{T,+}(I, a) = \max\{R_j^T(I, a), 0\}. \tag{17}$$

The regret matching algorithm (Hart & Mas-Colell, 2000) determines the strategy at each information set:

$$\pi_j^{T+1}(I)(a) = \begin{cases} \frac{R_j^{T,+}(I,a)}{\sum_{a \in A(I)} R_j^{T,+}(I,a)} & \text{if denominator} > 0 \\ \frac{1}{|A(I)|} & \text{otherwise} \end{cases}. \tag{18}$$

It's worth noting that in some studies, the normalization factor $\frac{1}{T}$ is omitted in Equ. (12), (14) and (16).

Vanilla CFR has several shortcomings. Firstly, it requires traversing the entire game tree in each iteration, computationally intractable for larger game trees. Although some efforts have focused on game abstraction to reduce the size of the game tree, greater abstraction can lead to decreased performance. Second, it requires storing counterfactual regrets $R^i(I, a)$ for every action a in every information I at each iteration i. These values are stored in $h(i)$ within our proposed framework (Algo. 1), leading to significant storage challenges.

3.5.3 Time-saving Variants of CFR

Many studies focus on enhancing the time efficiency of CFR. The first approach involves modifying the regret calculation to increase its speed. CFR+ (Tammelin, 2014) implements regret-matching+ by storing non-negative regret-like values $R_j^{+,i}(I, a)$, rather than $R_j^i(I, a)$. Additionally, CFR+ updates the regrets of each player sequentially and adopts a weighted average strategy. Moreover, Brown & Sandholm (2019a) introduces the concept of weighted regrets and develops Linear CFR (LCFR) and Discounted CFR (DCFR). The second approach involves adopting sampling methods. While it requires more iterations, each iteration is shorter, reducing the overall convergence time. Monte Carlo CFR (MCCFR) (Lanctot et al., 2009) divides terminal histories into blocks, with each iteration sampling from these blocks instead of traversing the entire game tree. This allows for calculating sampled counterfactual values for each player, leading to counterfactual regrets that, in expectation, match those of the Vanilla CFR. Vanilla CFR is a specific case where all histories are divided into just one block. MCCFR typically manifests in three forms: outcome-sampling MCCFR, where each block corresponds to a single history; external-sampling MCCFR, which samples opponent and chance nodes; and chance-sampling MCCFR, only focusing on chance nodes. Moreover, Johanson et al. (2012) expands on chance-sampling by categorizing based on the handling of public and private chance nodes. Some studies also focus on learning how to reduce the variance of MCCFR to speed up convergence (Schmid et al., 2019; Davis et al., 2020). In addition to these two primary approaches, other studies have identified that warm starting (Brown & Sandholm, 2016) and pruning (Brown & Sandholm, 2015; Brown et al., 2017; Brown & Sandholm, 2017) can also accelerate convergence.

3.5.4 Space-saving Variants of CFR

In perfect information games, decomposition reduces problem-solving scale by solving subgames. However, in imperfect information games, defining subgames is challenging due to their intersection with information set boundaries. CFR-D (Burch et al., 2014) is a pioneering method for decomposing imperfect information games into the main component called the trunk and subgames defined by forests of trees that do not divide any information sets. In each iteration, CFR is applied within the trunk for both players, and a solver is used to determine the counterfactual BR in the subgame. The process includes updating the trunk with the counterfactual values from the subgame's root and updating the average counterfactual values at this root, while the solution to the subgame is discarded. CFR-D minimizes storage needs by only saving $R^i(I_*, a)$ for information sets in the trunk and at each subgame's root, which is denoted by I_*, trading off storage efficiency against the time required to resolve subgames. Similar thoughts are echoed in the Continue-Resolving technique used by DeepStack (Moravčík et al., 2017) and the Safe and Nested Subgame Solving technique used by Libratus (Brown & Sandholm, 2018). We will discuss these approaches in Sec. 4.2.1, exploring their application to Texas Hold'em.

3.5.5 Estimation Variants of CFR

Although these CFR variants advance the field, they can't directly solve large imperfect-information extensive-form games due to their reliance on tabular representations. The typical approach involves abstracting the original game, applying CFR to the abstracted game, and translating strategies back to the original. This abstraction is game-specific and relies heavily on domain knowledge. Additionally, smaller abstractions often yield suboptimal results. Given these challenges, Waugh et al. (2015) introduces Regression CFR (RCFR), which employs a shared regressor $\varphi(I, a)$ to estimate counterfactual regrets. Nevertheless, using regression trees as the regressor limits RCFR's applicability to miniature games, and the necessity for manually crafted features remains a drawback. After advantage-based regret minimization (ARM) (Jin et al., 2018) merges CFR with deep RL in single-agent scenarios, a growing body of research has focused on applying CFR in conjunction with neural networks to multi-agent scenarios. Double Neural CFR (Li et al., 2018) utilizes two neural networks: one for estimating counterfactual regrets and another for approximating the average strategy. In a similar vein, Deep CFR (Brown et al., 2019) leverages an advantage network $V(I, a|\theta_p)$ to estimate counterfactual regrets with each player having a distinct hyperparameter θ_p and employs $\pi(I, a|\theta_\pi)$ for strategy estimation after the training process of the advantage network. Since these two networks are trained in sequence rather than concurrently, the strategy for each intermediate iteration remains conditioned on the output of the advantage network: $h(i) = V(I, a|\theta_p)$. Despite similarities, Deep

Table 1: Overview of representative self-play algorithms within our framework.

Algorithms	MSS	h(i)	Categories	E	Initialization of Π	Initialization of π_i^h	Suitable Game
Vanilla Self-play	Equ. (5)	$\emptyset$	Traditional Self-play	$E=1$	Lazy	General	Stage Game
FSP	Equ. (6)						
δ-Uniform Self-play	Equ. (7)						
PFSP	Algo. 5						
DO	NE-based (Algo. 6)	$\emptyset$	PSRO Series	$E=1$	Lazy	General	Stage Game
PSRO	General						
α-PSRO	α-rank-based						
JPSRO	(C)CE-based						
R-NaD	Equ. (5)						
FTW	Equ. (10)	$\emptyset$	Ongoing-training-based	$E>1$	Immediate	$\pi_i^h \leftarrow \pi_i(\cdot\|h(i))$	Stage Game
NeuPL	General	σ_i					
Simplex-NeuPL	General	σ_i					
Vanilla CFR	Equ. (5)	$R^i(I,a)$	Regret-minimization-based	$E=1$	Lazy	$\pi_i^h \leftarrow \pi_{i-1}(\cdot\|h(i-1))$	Repeated Game
CFR+		$R^{+,i}(I,a)$					
CFR-D		$R^i(I_*,a)$					
RCFR		$\varphi(I,a)$					
Deep CFR		$V(I,a\|\theta_p)$					

CFR distinguishes itself from Double Neural CFR through its data collection and proven effectiveness in larger-scale poker games. Single Deep CFR (SD-CFR) (Steinberger, 2019) demonstrates training an average strategy network is unnecessary, with only an advantage network required. Building on the foundation of SD-CFR, DREAM (Steinberger et al., 2020) utilizes a learned baseline to maintain low variance in a model-free setting when only one action is sampled at each decision point. Moreover, advantage regret-matching actor-critic (ARMAC) (Gruslys et al., 2020) incorporates the thought of retrospective policy improvement.

3.6 Reassessment of the Framework

After introducing these four categories of self-play algorithms, we will further compare them in this section, explain the rationale behind categorizing the algorithms, and summarize the representative algorithms in Table 1. Moreover, we will illustrate the differences between our framework and other frameworks to demonstrate why our proposed framework is more general.

3.6.1 Categorization Rationale

Traditional self-play algorithms and the PSRO series share many similarities. Initially, they require only one randomly initiated policy, and the policy population expands as training progresses. Therefore, in our framework, we use lazy initialization to initialize the policy population and set $E = 1$ for these two categories. The interaction matrix is typically a lower triangular matrix (Corollary 1 and Corollary 2). The primary difference between the PSRO series and traditional self-play algorithms is that the PSRO series employs more complex MSSes to handle tasks with intricate game-theoretical requirements. For example, α-PSRO (Muller et al., 2020) specifically utilizes an α-rank-based MSS to tackle multi-player general-sum games. In other tasks, traditional self-play algorithms are more commonly used to reduce the computational cost associated with complex MSSes in the PSRO series.

Unlike the two previously mentioned categories, the ongoing-training-based series adopts a different paradigm. Instead of gradually expanding the policy population and relying on newer policies to be stronger, this approach strengthens all policies simultaneously at each epoch. This method alleviates issues such as early truncation and repeated skill relearning that occur in the above two categories. To integrate this category into our framework, immediate initialization is used for the policy population, and $\pi_i(\cdot|h(i))$ is utilized to initialize π_i^h to ensure that policy updates are self-referential. Also, the interaction matrix is generally not a lower triangular matrix (Corollary 3).

Lastly, the regret-minimization-based series focuses on overall performance over time rather than on individual episodes, making it particularly suitable for repeated games. For example, Texas Hold'em is a classic repeated game where players adjust their strategies based on past interactions and use tactics involving deception and bluffing. The main goal of this training process is to update the regrets associated with different strategies, which in turn demands significant storage resources. Our framework uses $h(i)$ to store this information. Since the policies are determined by $h(i)$, only the most recent policy is relevant. Therefore, the interaction matrix is a unit lower shift matrix (Corollary 4). We also do not need actually to initialize the whole policy population and only need to use $\pi_{i-1}(\cdot|h(i-1))$ to initialize π_i^h in the training process.

3.6.2 Comparing with Existing Frameworks

Our framework is built upon PSRO (Lanctot et al., 2017) and NeuPL (Liu et al., 2022c). Here, we outline the differences between our framework and these existing frameworks. The primary distinction between our framework and PSRO is the use of an interaction matrix $\Sigma := \{\sigma_i\}_{i=1}^N \in \mathbb{R}^{N\times C_1}$ to represent the opponent sampling strategy, allowing for the integration of more complex competitive dynamics. Moreover, in our framework, σ_i denotes the opponent sampling strategy, which specifies how to sample opponents' policies against policy i rather than being the meta-strategy of policy i. Additionally, our framework incorporates a policy condition function $h(i)$, making it more general than NeuPL, where policies are conditioned on σ_i. This enhancement gives our framework greater expressiveness. Furthermore, we describe how to compute the oracle (Line 5 in Alg. 1) in three different ways (Alg. 2, Alg. 3 and Alg. 4) to provide a clearer understanding. Also, to the best of our knowledge, our framework is the first self-play framework to integrate the regret-minimization-based series, which is a significant self-play paradigm.

4 Empirical Analysis

In this section, we introduce iconic applications of self-play by categorizing the scenarios into three distinct groups: board games, which typically involve perfect information; card games, which usually involve imperfect information; and video games, which feature real-time actions rather than turn-based play. We then illustrate how self-play is applied in these complex scenarios and provide a comparative analysis of these applications in Table 2.

4.1 Board Games

Board games, the majority of which are perfect information games, were previously revolutionized by the introduction of two essential techniques: position evaluation and Monte Carlo tree search (MCTS) (Coulom, 2006; Kocsis & Szepesvári, 2006). These methodologies, with minor modifications, demonstrated superhuman effectiveness in solving board games such as chess (Campbell et al., 2002), checkers (Schaeffer et al., 1992), othello (Buro, 1999), backgammon (Tesauro & Galperin, 1996), and Scrabble (Sheppard, 2002). In contrast, the application of these techniques to the game of Go with an estimated 2.1×10^{170} legal board configurations, only enabled performance at the amateur level (Bouzy & Helmstetter, 2004; Coulom, 2007; Baudiš & Gailly, 2011; Enzenberger et al., 2010; Gelly & Silver, 2007). In light of this, our discussion will specifically focus on the game of Go to illustrate the application of self-play. In addition to Go, we will explore Stratego, a board game with imperfect information, in contrast to most board games that only involve perfect information.

4.1.1 Go

Go is an ancient board game, which is played on a grid of 19x19 lines, where two players alternately place black and white stones aiming to control the largest territory. The paradigm of Go artificial intelligence (AI) is revolutionized with the launch of DeepMind's AlphaGo series (Silver et al., 2016; 2017; 2018; Schrittwieser et al., 2020), which leveraged the power of self-play to significantly elevate performance. In AlphaGo (Silver et al., 2016), the training regime can be split into three stages. In the first stage, supervised learning with expert data trains a fast policy network $p_\pi(a|s)$ for rollouts in MCTS and a precise policy network $p_\sigma(a|s)$. The second stage employs self-play combined with RL to get a refined policy network $p_\rho(a|s)$ based on $p_\sigma(a|s)$ and subsequently trains a value network $v_\theta(s)$. Specifically, $p_\rho(a|s)$ is refined by competing against a randomly selected historical version $p_{\rho^-}(a|s)$, similar to the MSS shown in Equ. (6), while $v_\theta(s)$ is trained using the game samples collected by self-play of $p_\rho(a|s)$. In the third stage, MCTS integrates the policy and value networks to select actions.

Based on AlphaGo, AlphaGo Zero (Silver et al., 2017) does not require any expert data except game rules. It utilizes only one network $f_\theta(s)$ to concurrently predict the action probabilities and the state value. Self-play is employed to generate data and refine $f_\theta(s)$ with the current best policy competing against itself, a process analogous to the MSS referenced in Equ. (5). A new policy to be incorporated into the policy pool must surpass a 55 percent win rate against its predecessor. AlphaZero (Silver et al., 2018) extends AlphaGo Zero to include games beyond Go, such as Chess and Shogi. Concerning the self-play procedure, the only difference between AlphaZero and AlphaGo Zero is that AlphaZero utilizes the newly updated network to generate samples by self-play without validation. Building upon AlphaZero, MuZero (Schrittwieser et al., 2020) takes learning from scratch to the next level, even operating without predefined game rules. The self-play process in MuZero operates similarly to that in AlphaZero. In practice, in addition to excelling in board games, MuZero also achieves state-of-the-art performance in Atari games.

4.1.2 Stratego

Unlike most board games, which are perfect information games, Stratego, a two-player imperfect information board game, distinguishes itself by incorporating elements of memory, deduction, and bluffing. This complexity is further amplified by the game's long episode length and many potential game states, estimated 10^{535} (Perolat et al., 2022). The game is divided into two phases: the deployment phase, where players secretly arrange their units, and the game-play phase, where the objective is to deduce the opponent's setup and capture their flag. The depth and computational complexity of the game remained a challenge until breakthroughs such as DeepNash (Perolat et al., 2022) showed promising advances in AI's ability to tackle it. DeepNash scales up evolution-theory-based self-play method R-NaD (Perolat et al., 2021) (discussed in Sec. 3.3.5) to neural R-NaD. It employs a neural network with four heads: one for value prediction, one for the deployment phase, one for piece selection, and one for piece displacement. Neural Replicator Dynamics (Hennes et al., 2020) is utilized to obtain the approximate fixed-point policy. DeepNash holds the third-place ranking among all professional Gravon Stratego players and wins nearly every match against existing Stratego bots.

4.2 Card Games

Unlike board games, card games are typically imperfect information games with a larger state space and greater complexity. Here, we introduce three representative card games: Texas Hold'em, DouDiZhu, and Mahjong, to illustrate how self-play is utilized in card games. Texas Hold'em is a repeated game rather than a stage game, so the focus is on the overall win-loss outcome rather than on individual rounds. Consequently, it is particularly well-suited for regret-minimization-based self-play methods, which aim to minimize overall regret rather than regret on a per-round basis. In contrast, DouDiZhu and Mahjong are typically regarded as stage games, so studies often employ self-play methods that do not rely on regret minimization.

4.2.1 Texas Hold'em

Texas Hold'em, a popular poker game with 2-10 players, is known for its strategic depth and bluffing elements. The two-player variant is called **heads-up Texas Hold'em**. The gameplay begins with each player receiving two private cards (hole cards), followed by a round of betting. Subsequently, three community cards (the flop) are revealed, leading to another betting round. This is followed by dealing a fourth (the turn) and a fifth community card (the river), each accompanied by further betting rounds. The objective is to construct the best five-card poker hand from any combination of hole cards and community cards. Texas Hold'em offers two betting formats: limited betting and no-limit betting. The latter one is noted for its complexity, allowing players to bet any amount up to their entire stack of chips. While simplified versions such as **Kuhn Poker** and **Leduc Poker** serve valuable roles in theoretical analysis, we will focus on the algorithms designed to compete in full Texas Hold'em.

Heads-up limit Texas Hold'em (HULHE), the simplest form with approximately 3.16×10^{17} game states, was not solved until the introduction of Cepheus (Bowling et al., 2015). It utilizes fixed-point arithmetic with compression and regret-minimization-based self-play method CFR+ (Tammelin, 2014) to address the issues of storage and computation, respectively, resulting in superhuman performance. Deep CFR (Brown et al., 2019) combines regret-minimization-based self-play method CFR with deep neural networks. Furthermore, some studies do not adopt the regret-minimization-based series of self-play; instead, they use the traditional self-play method. NSFP (Heinrich & Silver, 2016) introduces a self-play method combined with end-to-end RL training, also achieving competitive performance in HULHE. Poker-CNN (Yakovenko et al., 2016) utilizes self-play to learn convolutional networks to solve a video version of HULHE.

After solving HULHE, research focus shifts to **heads-up no-limit Texas Hold'em (HUNL)** with significantly larger game states, approximately 10^{164} (Johanson, 2013). Thus, traversing the game tree as Cepheus does in HULHE is impossible. DeepStack (Moravčík et al., 2017) is a regret-minimization-based self-play method combined with continual re-solving. This method focuses on a subtree of limited depth and breadth and estimates the outcomes of the furthest reaches of this subtree. Libratus (Brown & Sandholm, 2018) develops a blueprint strategy, leveraging an enhanced version of regret-minimization-based self-play method MCCFR (Lanctot et al., 2009). Recursive Belief-based Learning (ReBeL) (Brown et al., 2020) introduces the public belief state (PBS), transforming the imperfect-information game into a perfect-information game with continuous state space. ReBeL utilizes the regret-minimization-based self-play method CFR-D, combined with RL and search, to train both the value and policy networks. Unlike the algorithms mentioned above, AlphaHoldem (Zhao et al., 2022a) does not use regret-minimization-based self-play. Instead, it proposes a K-best self-play method that always selects the top K agents, based on their ELO ratings (Elo & Sloan, 1978), to generate samples.

Pluribus (Brown & Sandholm, 2019b), based on Libratus, addresses **six-player no-limit Texas Hold'em**. Similar to Libratus, Pluribus utilizes a regret-minimization-based self-play method MCCFR (Lanctot et al., 2009) to develop its blueprint strategy. It conducts a depth-limited search before executing actions. Different from Libratus, Pluribus maintains a streamlined policy pool of only four strategies, assuming that its opponents might adjust their strategies during gameplay among these four strategies, which allows Pluribus to manage complexity more efficiently.

4.2.2 DouDiZhu

DouDizhu (a.k.a. Fight the Landlord) is a three-player strategic card game. In this game, one player takes on the role of the landlord and competes against the other two players, the peasants. The game is played in two main stages: the bidding stage and the card play stage. During the bidding stage, players vie to become the landlord. During the card play stage, players take turns playing cards in various combinations, intending to be the first to empty their hands. DouDiZhu is characterized by imperfect information, as players can only see their own cards and the cards played. The essence of the game lies in cooperation among peasants and competition between the two factions. It has an estimated $10^{76} \sim 10^{106}$ possible game states (Zha et al., 2019) and an action space comprising 27472 possible moves (Yang et al., 2022).

DeltaDou (Jiang et al., 2019) is the first algorithm to achieve expert-level performance in DouDizhu. Similar to AlphaZero (Silver et al., 2018), DeltaDou utilizes vanilla self-play to generate samples to train policy and value networks. DouZero (Zha et al., 2021) also uses vanilla self-play to generate data. It reduces training costs by opting for a sampling method over the tree search approach. Based on DouZero, DouZero+ (Zhao et al., 2022b) incorporates opponent modeling, assisting the agent in making more informed decisions. PerfectDou (Yang et al., 2022) utilizes vanilla self-play under a Perfect-Training-Imperfect-Execution framework.

4.2.3 Mahjong

Mahjong has evolved into various global variants, including the famous Japanese version known as Riichi Mahjong. This game is typically played by four players who must navigate both the visible aspects of the game, such as discarded tiles, and the hidden elements, like their own hand and the unseen public tiles. The strategic depth and complexity of Mahjong pose significant challenges. Despite ongoing research, these findings have yet to reach expert human levels (Kurita & Hoki, 2020; Gao et al., 2019; Mizukami

& Tsuruoka, 2015). Difficulties are navigating incomplete information, dynamically adapting strategies to multiple opponents, and contending with complex winning rules and an enormous number of possible game states, estimated at 10^{169} (Zha et al., 2019). Suphx (Li et al., 2020) is recognized as one of the first algorithms to master Mahjong, achieving a performance level comparable to expert human players, precisely 10 dan on Tenhou (Tsunoda), the most popular online Mahjong platform. Initially, Suphx employs supervised learning, utilizing expert data to train its model. It then advances its capabilities through vanilla self-play combined with RL. Similarly, NAGA (Village), developed by Dwango Media Village, and LuckyJ (Tencent), designed by Tencent, have also achieved the rank of 10 dan on Tenhou. Furthermore, LuckyJ has even defeated human professional players.

4.3 Video Games

In contrast to traditional board games and card games, video games often feature real-time actions, long trajectories, and increased complexity stemming from a broader range of actions and observations. We illustrate representative video games showcasing self-play's impact.

4.3.1 StarCraft II

StarCraft is a real-time strategy (RTS) game. It has three distinct species: the Terrans, Zerg, and Protoss, each with unique units and strategic options that enhance the gameplay complexity. Renowned for its balanced gameplay, strategic depth, and intense competitiveness, the game challenges players to gather resources, construct bases, and build armies. Victory requires meticulous planning and tactical execution, with defeat occurring when a player loses all their buildings. AlphaStar (Vinyals et al., 2019) dominates the 1v1 mode competitions in StarCraft II and has defeated professional players. Its framework is similar to AlphaGo (Silver et al., 2016), initially utilizing supervised learning to train the policy with expert data. Subsequently, it uses a hierarchical self-play method combined with end-to-end RL to train the networks. More specifically, The proposed self-play method divides all the agents into three types: main agents, league exploiters, and main exploiters. It maintains a policy pool of past players that records all these types of agents. **Main agents** engage in traditional self-play algorithms like FSP and PFSP, competing against main agents themselves and other agents in the policy pool. They are periodically added to the pool and never reset. **League exploiters** also use a traditional self-play method PFSP to play against all policy pool agents, added to the pool if they show a high win rate and potentially reset to expose global blind spots. **Main exploiters** only compete with main agents to improve their robustness, are added to the pool after achieving a high win rate or specific training steps, and are reset upon each addition. Among those three agent types, the main agent is the core agent and embodies the final AlphaStar strategy. However, the introduction of three agent types significantly increases the training computation. Further studies (Han et al., 2020; Wang et al., 2021; Huang et al., 2024) have enhanced the league self-play training procedure.

4.3.2 MOBA Games

Multiplayer Online Battle Arena (MOBA) games are a popular video game genre that blends RTS with role-playing elements. In typical MOBA games, two teams of players control their unique characters, known as heroes, and compete to destroy the opposing team's main structure, often referred to as the base. Each hero has distinct abilities and plays a specific role within the team, such as Warrior, Tank, or Support. Managing multiple lanes and battling under the fog of war, which obscures parts of the map, are critical aspects of the gameplay.

OpenAI Five (Berner et al., 2019) defeated the world champion team in a simplified version of Dota 2 that featured a limited pool of heroes and certain banned items. The training process introduces a self-play method that combines two traditional self-play algorithms: with an 80% probability of engaging in vanilla self-play and a 20% probability of employing a technique similar to the PFSP used in AlphaStar (Vinyals et al., 2019). This technique selects each policy from the policy pool based on its quality score, which is continuously updated from competition results throughout training. Higher quality scores increase the likelihood of a policy being selected. OpenAI Five also requires extensive training resources.

Another notable MOBA game, especially viral in China, is Honor of Kings. The 1v1 mode is conquered by Ye et al. (2020b), which boasts a significant winning rate against top professional players. It utilizes a traditional self-play method δ-uniform self-play to train the RL networks. The 5v5 mode was later mastered by Ye et al. (2020a) as well. Unlike OpenAI Five, this work expands the hero pool to 40 heroes, exponentially increasing the possible combinations of lineups. It proposes a new self-play method referred to as curriculum self-play learning (CSPL). Specifically, the training process is divided into three stages. The first stage involves training fixed lineups through self-play and utilizing human data to balance the two teams to aid policy improvements. The second stage employs multi-teacher policy distillation to produce a distilled model. The final stage uses this distilled model as the initial model for another round of self-play with randomly picked lineups. This approach defeats professional player teams. The self-play generated data is also used to learn compelling lineup drafting by utilizing MCTS and neural networks (Chen et al., 2021).

4.3.3 Google Research Football

Google Research Football (GRF) (Kurach et al., 2020) is an open-source football simulator emphasizing high-level actions. It initially offers two scenarios: the football benchmark and the football academy with 11 specific tasks. Here, we focus exclusively on the football benchmark because it presents a more complex scenario that better demonstrates the effects of self-play. GRF is particularly challenging due to the need for cooperation among teammates and competition against opposing teams. It features long trajectories with 3000 steps per round, stochastic transitions, and sparse rewards.

WeKick (Ziyang Li, 2020) claimed victory in the GRF competition on Kaggle (Google, 2020), which simplifies the game dynamics by allowing competitors to control only one player, either the ball carrier on offense or the nearest defender on defense. It employed self-play strategies similar to those used in league training (Vinyals et al., 2019). It initializes its opponent policy pool using strategies developed through RL and Generative Adversarial Imitation Learning (GAIL) (Ho & Ermon, 2016) to facilitate the training process.

Further research delves into the full football game rather than the simplified version. Team-PSRO (McAleer et al., 2023) extends PSRO series of self-play algorithms to team games, outperforming baselines in the 4v4 version of the full GRF. In the context of the 11v11 version, where the goalkeeper is rule-based controlled, TiKick (Huang et al., 2021a) utilizes vanilla self-play to collect samples and then employs imitation learning. Fictitious Cross-Play (FXP) (Xu et al., 2023) proposes a new self-play method similar to AlphaStar (Vinyals et al., 2019). It introduces two populations: the main population and the counter population. Policies in the counter population improve solely by cross-playing with policies in the main population as opponents, while policies in the main population engage in playing with policies from both populations. FXP achieves a win rate of over 94% against TiKick. TiZero (Lin et al., 2023), a follow-up to TiKick, combines curriculum learning with FSP and PFSP (Vinyals et al., 2019) to avoid expert data reliance, achieving a higher TrueSkill rating (Herbrich et al., 2006) than TiKick.

5 Open Problems and Future Work

Self-play methods have shown remarkable performance by leveraging iterative learning and adapting to complex environments. However, significant challenges and open questions remain, presenting opportunities for further research and development.

5.1 Theoretical Foundation

Although NE has been shown to exist in games with finite players and finite actions (Nash et al., 1950), computing NE with self-play algorithms in larger games remains challenging and consequently, many studies aim to achieve approximate NE (Li et al., 2024). However, in some cases, even computing an approximate NE is difficult (Daskalakis, 2013). Some research has resorted to higher levels of equilibrium, such as CE (Marris et al., 2021) and α-rank (Muller et al., 2020). Although many algorithms are developed with theoretical foundations, there is often a gap in complex real-world scenarios. For example, although self-play algorithms like AlphaGo (Silver et al., 2016), AlphaStar (Samvelyan et al., 2019), and OpenAI Five (Berner et al., 2019) achieve empirical success, they lack formal game theory proofs behind their effectiveness. Future work

Table 2: Overview of representative studies in empirical analysis.

Category	Game	Number of Players	Perfect Information	Number of Game States	Algorithms	Self-play Method	Search	Expert Data
Board Game	Chess	2	✓	10^{45}	AlphaZero	Vanilla SP	✓	×
					MuZero	Vanilla SP	✓	×
	Go	2	✓	10^{360}	AlphaGo	FSP	✓	✓
					AlphaGo Zero	Vanilla SP	✓	×
					AlphaZero	Vanilla SP	✓	×
					MuZero	Vanilla SP	✓	×
	Stratego	2	×	10^{535}	DeepNash	R-NaD	×	×
Card Game	HULHE	2	×	10^{17}	Cepheus	CFR+	✓	×
					NFSP	NFSP	×	×
	HUNL	2	×	10^{164}	DeepStack	CFR+	✓	×
					Libratus	MCCFR	✓	×
					ReBel	CFR-D / FSP	✓	×
	Six-player No-limit Texas Hold'em	6	×	$> 10^{164}$	Pluribus	MCCFR	✓	×
	DouDiZhu	3	×	$10^{76} \sim 10^{106}$	DeltaDou	Vanilla SP	✓	×
					DouZero	Vanilla SP	×	✓
					PerfectDou	Vanilla SP	×	×
	Mahjong	4	×	10^{169}	Suphx	Vanilla SP	✓	✓
Video Game	StarCraft II	2	×	/	AlphaStar	FSP & PFSP	×	✓
	Dota 2	10	×		OpenAI Five	Vanilla SP & PFSP	×	×
	Honor of Kings	10	×		MOBA AI	δ-uniform SP	✓	✓
	Google Research Football	22	✓		TiZero	FSP & PFSP	×	×

should focus on bridging this gap by developing new theoretically sound and practically effective algorithms or proving the theoretical underpinnings of existing successful self-play algorithms in complex environments.

5.2 Non-stationarity of the Environment

In the self-play framework, the opponents are a vital component of the environment, and the strategies of the opponent players evolve as training progresses. This evolution can cause the same strategy to lead to different results over time, creating a non-stationary environment. This problem is also shared by the MARL area. Future research should aim to develop self-play algorithms that are more robust and can adapt to changing conditions. For example, incorporating opponent modeling (Zhao et al., 2022b) into self-play can help agents anticipate changes in opponent strategies and adjust their own strategies proactively, making them more robust to environmental changes.

5.3 Scalability and Training Efficiency

The scalability of self-play methods faces significant challenges as the number of teams and players within those teams increases. As the number of participants grows, the complexity of interactions explodes. For example, in OpenAI Five (Berner et al., 2019), the hero pool size is limited to only 17 heroes. MOBA AI (Ye et al., 2020a) extends this to a 40-hero pool with the help of curriculum learning, but it still cannot cover the entire hero pool available in the actual game. One potential solution is leveraging players' inherent connections to optimize the learning process. For instance, using graph-based models to represent and exploit the relationships between players can help manage and reduce the complexity of large-scale multi-agent environments. These scalability issues are fundamentally rooted in the limited training efficiency of self-play methods from two aspects including computation and storage. The first issue is computational efficiency, which is induced by the iterative nature of self-play where agents repeatedly play against themselves or past versions. Moreover, although forming more complex populations and competitive mechanisms (Samvelyan et al., 2019) can enhance the intensity and quality of training, it further exacerbates the demand for computational resources. Techniques such as parallel computing, distributed learning, and more efficient neural network architectures could be explored to address these challenges. The second issue is storage because self-play requires maintaining a policy pool. Even when using a shared network architecture, storing the parameters of large models can be problematic. This issue is particularly pronounced in regret-minimization-based self-play algorithms, which must store the regrets for each information set and potential action. Managing both the computational load and storage requirements is essential for improving the overall training efficiency and scalability of self-play methods.

5.4 With Large Language Models

With their remarkable capability and emergent generalizability, large language models (LLMs) have been regarded as a potential foundation for achieving human-level intelligence (Achiam et al., 2023), and self-play methods have been proposed to fine-tune LLMs, enhance LLMs' reasoning performance, and build LLM-based agents with strong decision-making abilities. Post-training fine-tuning (Ouyang et al., 2022; Bai et al., 2022) is a key step in aligning LLM with more desired behaviors, but it requires a huge amount of human-annotated data. To reduce reliance on human-annotated data, Self-play fIne-tuNing (SPIN) (Chen et al., 2024) introduces a self-play mechanism to generate training data using the LLM itself and fine-tuning the LLM by distinguishing self-generated responses from human-annotated data. Another line of work (Munos et al., 2024; Swamy et al., 2024; Wu et al., 2024) formulate the alignment problem as a two-player constant-sum game and use self-play methods to solve the game. Some other work (Huang et al., 2022; Yuan et al., 2024) also utilizes model-generated data to fine-tune LLMs with minimal human annotations. The idea of self-improvement has also been applied to improve the reasoning ability of LLMs. Self-Play of Adversarial Game (SPAG) (Cheng et al., 2025) observes that self-play in a two-player adversarial language game called Adversarial Taboo can boost the LLM's performance on various reasoning benchmarks. Besides improving the capability of LLMs, self-play methods have also contributed to building LLM-based agents with strong strategic abilities. A representative work is Cicero (, FAIR) which achieves human-level play in the game of Diplomacy by combining language models with an RL policy trained by self-play. Cicero uses the self-play policy to produce an intended action and prompts the language model to generate languages conditioned on the policy's intent. Another work (Xu et al., 2024) also combines LLM with self-play policy but takes a different approach by first prompting the LLM to propose multiple action candidates and then using the self-play policy to produce the action distribution over these candidates. Despite recent progress, applying self-play with LLMs is still underexplored and requires further research.

5.5 Real-World Applications

Self-play is particularly effective in addressing some problems abstracted from real-world situations through its iterative learning approach. In the field of economics, for instance, self-play is used to enhance supervised learning models in multi-issue bargaining tasks (Lewis et al., 2017). Furthermore, self-play proves beneficial in solving combinatorial optimization problems (COPs) such as the traveling salesman problem (TSP) and the capacitated vehicle routing problem (CVRP) (Wang et al., 2024). Within the domain of traffic, self-play facilitates the development of human-like autonomous driving behaviors (Cornelisse & Vinitsky, 2024) and enables vehicles to learn negotiation strategies to merge on or off roads (Tang, 2019), although currently within 2D simulators.

However, a notable challenge with self-play is its impracticality for direct application in real-world scenarios. Its iterative approach requires extensive trial and error, which is computationally demanding and often involves actions that are impractical or unsafe outside of a controlled simulation. Therefore, self-play is often carried out in simulators. The effective deployment of self-play in real-world applications boils down to overcoming the Sim2Real gap. For example, for tasks where the Sim2Real gap is less pronounced, self-play can be well employed to enhance video streaming capabilities (Huang et al., 2021b), and to address the image retargeting problem (Kajiura et al., 2020). Moreover, EvoPlay (Wang et al., 2023) leverages self-play to design protein sequences, utilizing the advanced AlphaFold2 simulator (Jumper et al., 2021) to narrow the Sim2Real gap. Similarly, in heterogeneous multi-robot systems, self-play is utilized for competitive sports tasks like humanoid football (Liu et al., 2022b; Haarnoja et al., 2024), quadruped competition (Xiong et al., 2024), and multi-drone volleyball (Xu et al., 2025), with significant efforts dedicated to Sim2Real transitions for real-world success (Gao et al., 2023).

6 Conclusion

The burgeoning field of self-play in RL has been systematically explored in this survey. The core idea of self-play that agents interact with copies or past versions of themselves has proven to be a powerful approach for developing robust strategies across various domains. Before delving into the specifics of self-play, this paper first elucidates the MARL framework and introduces the game theory background of self-play. Moreover, the paper presents a unified framework and categorizes existing self-play algorithms into four main categories: traditional self-play algorithms, the PSRO series, the ongoing-training-based series, and the regret-minimization-based series. We provide detailed explanations on how these four groups are seamlessly integrated into our proposed framework, ensuring a comprehensive understanding of their functionalities. The transition from theory to application is underscored by a rigorous analysis of self-play's role within various scenarios, such as board games, card games, and video games. Despite the successes of self-play in many areas, challenges remain, such as convergence to suboptimal strategies and substantial computational demands. Also, future work can focus on how to integrate self-play with LLMs and achieve real-world applications. In conclusion, self-play is vital to modern RL research, offering key insights and tools for developing advanced AI systems. This survey serves as an essential guide for researchers and practitioners, paving the way for further advancements in this field.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Robert J Aumann. Subjectivity and correlation in randomized strategies. *Journal of mathematical Economics*, 1(1):67–96, 1974.

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022.

David Balduzzi, Marta Garnelo, Yoram Bachrach, Wojciech Czarnecki, Julien Perolat, Max Jaderberg, and Thore Graepel. Open-ended learning in symmetric zero-sum games. In *International Conference on Machine Learning*, pp. 434–443. PMLR, 2019.

Trapit Bansal, Jakub Pachocki, Szymon Sidor, Ilya Sutskever, and Igor Mordatch. Emergent complexity via multi-agent competition. In *International Conference on Learning Representations*, 2018.

Petr Baudiš and Jean-loup Gailly. Pachi: State of the art open source go program. *Advances in computer games*, pp. 24–38, 2011.

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

Ariyan Bighashdel, Yongzhao Wang, Stephen McAleer, Rahul Savani, and Frans A Oliehoek. Policy space response oracles: a survey. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, pp. 7951–7961, 2024.

Bruno Bouzy and Bernard Helmstetter. Monte-carlo go developments. *Advances in Computer Games: Many Games, Many Challenges*, pp. 159–174, 2004.

Michael Bowling, Neil Burch, Michael Johanson, and Oskari Tammelin. Heads-up limit hold'em poker is solved. *Science*, 347(6218):145–149, 2015.

George W Brown. Iterative solution of games by fictitious play. *Act. Anal. Prod Allocation*, 13(1):374, 1951.

Noam Brown and Tuomas Sandholm. Regret-based pruning in extensive-form games. *Advances in neural information processing systems*, 28, 2015.

Noam Brown and Tuomas Sandholm. Strategy-based warm starting for regret minimization in games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

Noam Brown and Tuomas Sandholm. Reduced space and faster convergence in imperfect-information games via pruning. In *International conference on machine learning*, pp. 596–604. PMLR, 2017.

Noam Brown and Tuomas Sandholm. Superhuman ai for heads-up no-limit poker: Libratus beats top professionals. *Science*, 359(6374):418–424, 2018.

Noam Brown and Tuomas Sandholm. Solving imperfect-information games via discounted regret minimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 1829–1836, 2019a.

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

Noam Brown, Christian Kroer, and Tuomas Sandholm. Dynamic thresholding and pruning for regret minimization. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

Noam Brown, Adam Lerer, Sam Gross, and Tuomas Sandholm. Deep counterfactual regret minimization. In *International conference on machine learning*, pp. 793–802. PMLR, 2019.

Noam Brown, Anton Bakhtin, Adam Lerer, and Qucheng Gong. Combining deep reinforcement learning and search for imperfect-information games. *Advances in Neural Information Processing Systems*, 33: 17057–17069, 2020.

Neil Burch, Michael Johanson, and Michael Bowling. Solving imperfect information games using decomposition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28, 2014.

Neil Burch, Matej Moravcik, and Martin Schmid. Revisiting cfr+ and alternating updates. *Journal of Artificial Intelligence Research*, 64:429–443, 2019.

Michael Buro. From simple features to sophisticated evaluation functions. In *Computers and Games: First International Conference, CG'98 Tsukuba, Japan, November 11–12, 1998 Proceedings 1*, pp. 126–145. Springer, 1999.

Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. Deep blue. *Artificial intelligence*, 134(1-2): 57–83, 2002.

Sheng Chen, Menghui Zhu, Deheng Ye, Weinan Zhang, Qiang Fu, and Wei Yang. Which heroes to pick? learning to draft in moba games with neural networks and tree search. *IEEE Transactions on Games*, 13 (4):410–421, 2021.

Zixiang Chen, Yihe Deng, Huizhuo Yuan, Kaixuan Ji, and Quanquan Gu. Self-play fine-tuning converts weak language models to strong language models. In *International Conference on Machine Learning*, pp. 6621–6642. PMLR, 2024.

Pengyu Cheng, Tianhao Hu, Han Xu, Zhisong Zhang, Yong Dai, Lei Han, Xiaolong Li, et al. Self-playing adversarial language game enhances llm reasoning. *Advances in Neural Information Processing Systems*, 37:126515–126543, 2025.

Daphne Cornelisse and Eugene Vinitsky. Human-compatible driving partners through data-regularized self-play reinforcement learning. *arXiv preprint arXiv:2403.19648*, 2024.

Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pp. 72–83. Springer, 2006.

Rémi Coulom. Computing "elo ratings" of move patterns in the game of go. *ICGA journal*, 30(4):198–208, 2007.

Constantinos Daskalakis. On the complexity of approximating a nash equilibrium. *ACM Transactions on Algorithms (TALG)*, 9(3):1–35, 2013.

Constantinos Daskalakis, Paul W Goldberg, and Christos H Papadimitriou. The complexity of computing a nash equilibrium. *Communications of the ACM*, 52(2):89–97, 2009.

Trevor Davis, Martin Schmid, and Michael Bowling. Low-variance and zero-variance baselines for extensive-form games. In *International Conference on Machine Learning*, pp. 2392–2401. PMLR, 2020.

Anthony DiGiovanni and Ethan C Zell. Survey of self-play in reinforcement learning. *arXiv preprint arXiv:2107.02850*, 2021.

Le Cong Dinh, Stephen McAleer, Zheng Tian, Nicolas Perez-Nieves, Oliver Slumbers, David Henry Mguni, Jun Wang, Haitham Bou Ammar, and Yaodong Yang. Online double oracle. *TMLR: Transactions on Machine Learning Research*, 2022.

Arpad E Elo and Sam Sloan. The rating of chessplayers: Past and present. 1978.

Markus Enzenberger, Martin Müller, Broderick Arneson, and Richard Segal. Fuego—an open-source framework for board games and go engine based on monte carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(4):259–270, 2010.

Meta Fundamental AI Research Diplomacy Team (FAIR)†, Anton Bakhtin, Noam Brown, Emily Dinan, Gabriele Farina, Colin Flaherty, Daniel Fried, Andrew Goff, Jonathan Gray, Hengyuan Hu, et al. Human-level play in the game of diplomacy by combining language models with strategic reasoning. *Science*, 378 (6624):1067–1074, 2022.

Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Shiqi Gao, Fuminori Okuya, Yoshihiro Kawahara, and Yoshimasa Tsuruoka. Building a computer mahjong player via deep convolutional neural networks. *arXiv preprint arXiv:1906.02146*, 2019.

Yuan Gao, Junfeng Chen, Xi Chen, Chongyang Wang, Junjie Hu, Fuqin Deng, and Tin Lun Lam. Asymmetric self-play-enabled intelligent heterogeneous multirobot catching system using deep multiagent reinforcement learning. *IEEE Transactions on Robotics*, 2023.

Marta Garnelo, Wojciech Marian Czarnecki, Siqi Liu, Dhruva Tirumala, Junhyuk Oh, Gauthier Gidel, Hado van Hasselt, and David Balduzzi. Pick your battles: Interaction graphs as population-level objectives for strategic diversity. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 1501–1503, 2021.

Sylvain Gelly and David Silver. Combining online and offline knowledge in uct. In *Proceedings of the 24th international conference on Machine learning*, pp. 273–280, 2007.

Paul W Goldberg, Christos H Papadimitriou, and Rahul Savani. The complexity of the homotopy method, equilibrium selection, and lemke-howson solutions. *ACM Transactions on Economics and Computation (TEAC)*, 1(2):1–25, 2013.

Google. Google research football competition 2020. `https://www.kaggle.com/c/google-football`, 2020.

Audrūnas Gruslys, Marc Lanctot, Rémi Munos, Finbarr Timbers, Martin Schmid, Julien Perolat, Dustin Morrill, Vinicius Zambaldi, Jean-Baptiste Lespiau, John Schultz, et al. The advantage regret-matching actor-critic. *arXiv preprint arXiv:2008.12234*, 2020.

Tuomas Haarnoja, Ben Moran, Guy Lever, Sandy H Huang, Dhruva Tirumala, Jan Humplik, Markus Wulfmeier, Saran Tunyasuvunakool, Noah Y Siegel, Roland Hafner, et al. Learning agile soccer skills for a bipedal robot with deep reinforcement learning. *Science Robotics*, 9(89):eadi8022, 2024.

Lei Han, Jiechao Xiong, Peng Sun, Xinghai Sun, Meng Fang, Qingwei Guo, Qiaobo Chen, Tengfei Shi, Hongsheng Yu, Xipeng Wu, et al. Tstarbot-x: An open-sourced and comprehensive study for efficient league training in starcraft ii full game. *arXiv preprint arXiv:2011.13729*, 2020.

Sergiu Hart and Andreu Mas-Colell. A simple adaptive procedure leading to correlated equilibrium. *Econometrica*, 68(5):1127–1150, 2000.

Johannes Heinrich and David Silver. Deep reinforcement learning from self-play in imperfect-information games. *arXiv preprint arXiv:1603.01121*, 2016.

Johannes Heinrich, Marc Lanctot, and David Silver. Fictitious self-play in extensive-form games. In *International conference on machine learning*, pp. 805–813. PMLR, 2015.

Daniel Hennes, Dustin Morrill, Shayegan Omidshafiei, Rémi Munos, Julien Perolat, Marc Lanctot, Audrunas Gruslys, Jean-Baptiste Lespiau, Paavo Parmas, Edgar Duéñez-Guzmán, et al. Neural replicator dynamics: Multiagent learning via hedging policy gradients. In *Proceedings of the 19th international conference on autonomous agents and multiagent systems*, pp. 492–501, 2020.

Ralf Herbrich, Tom Minka, and Thore Graepel. Trueskill™: a bayesian skill rating system. *Advances in neural information processing systems*, 19, 2006.

Daniel Hernandez, Kevin Denamganai, Sam Devlin, Spyridon Samothrakis, and James Alfred Walker. A comparison of self-play algorithms under a generalized framework. *IEEE Transactions on Games*, 14(2): 221–231, 2021.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *Advances in neural information processing systems*, 29:4565–4573, 2016.

Jiaxin Huang, Shixiang Shane Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. Large language models can self-improve. *arXiv preprint arXiv:2210.11610*, 2022.

Ruozi Huang, Xipeng Wu, Hongsheng Yu, Zhong Fan, Haobo Fu, Qiang Fu, and Wei Yang. A robust and opponent-aware league training method for starcraft ii. *Advances in Neural Information Processing Systems*, 36, 2024.

Shiyu Huang, Wenze Chen, Longfei Zhang, Shizhen Xu, Ziyang Li, Fengming Zhu, Deheng Ye, Ting Chen, and Jun Zhu. Tikick: Towards playing multi-agent football full games from single-agent demonstrations. *arXiv preprint arXiv:2110.04507*, 2021a.

Tianchi Huang, Rui-Xiao Zhang, and Lifeng Sun. Zwei: A self-play reinforcement learning framework for video transmission services. *IEEE Transactions on Multimedia*, 24:1350–1365, 2021b.

Max Jaderberg, Wojciech M Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castaneda, Charles Beattie, Neil C Rabinowitz, Ari S Morcos, Avraham Ruderman, et al. Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865, 2019.

Qiqi Jiang, Kuangzheng Li, Boyao Du, Hao Chen, and Hai Fang. Deltadou: Expert-level doudizhu ai through self-play. In *IJCAI*, pp. 1265–1271, 2019.

Peter Jin, Kurt Keutzer, and Sergey Levine. Regret minimization for partially observable deep reinforcement learning. In *International conference on machine learning*, pp. 2342–2351. PMLR, 2018.

Michael Johanson. Measuring the size of large no-limit poker games. *arXiv preprint arXiv:1302.7008*, 2013.

Michael Johanson, Nolan Bard, Marc Lanctot, Richard G Gibson, and Michael Bowling. Efficient nash equilibrium approximation through monte carlo counterfactual regret minimization. In *Aamas*, pp. 837–846, 2012.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

Nobukatsu Kajiura, Satoshi Kosugi, Xueting Wang, and Toshihiko Yamasaki. Self-play reinforcement learning for fast image retargeting. In *Proceedings of the 28th ACM international conference on multimedia*, pp. 1755–1763, 2020.

Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pp. 282–293. Springer, 2006.

Jakub Grudzien Kuba, Ruiqing Chen, Muning Wen, Ying Wen, Fanglei Sun, Jun Wang, and Yaodong Yang. Trust region policy optimisation in multi-agent reinforcement learning. *arXiv preprint arXiv:2109.11251*, 2021.

Alex Kulesza, Ben Taskar, et al. Determinantal point processes for machine learning. *Foundations and Trends® in Machine Learning*, 5(2–3):123–286, 2012.

Karol Kurach, Anton Raichuk, Piotr Stańczyk, Michał Zając, Olivier Bachem, Lasse Espeholt, Carlos Riquelme, Damien Vincent, Marcin Michalski, Olivier Bousquet, et al. Google research football: A novel reinforcement learning environment. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 4501–4510, 2020.

Moyuru Kurita and Kunihito Hoki. Method for constructing artificial intelligence player with abstractions to markov decision processes in multiplayer game of mahjong. *IEEE Transactions on Games*, 13(1):99–110, 2020.

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

Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Pérolat, David Silver, and Thore Graepel. A unified game-theoretic approach to multiagent reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

Mike Lewis, Denis Yarats, Yann Dauphin, Devi Parikh, and Dhruv Batra. Deal or no deal? end-to-end learning of negotiation dialogues. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 2443–2453, 2017.

Hanyu Li, Wenhan Huang, Zhijian Duan, David Henry Mguni, Kun Shao, Jun Wang, and Xiaotie Deng. A survey on algorithms for nash equilibria in finite normal-form games. *Computer Science Review*, 51: 100613, 2024.

Hui Li, Kailiang Hu, Zhibang Ge, Tao Jiang, Yuan Qi, and Le Song. Double neural counterfactual regret minimization. *arXiv preprint arXiv:1812.10607*, 2018.

Junjie Li, Sotetsu Koyamada, Qiwei Ye, Guoqing Liu, Chao Wang, Ruihan Yang, Li Zhao, Tao Qin, Tie-Yan Liu, and Hsiao-Wuen Hon. Suphx: Mastering mahjong with deep reinforcement learning. *arXiv preprint arXiv:2003.13590*, 2020.

Fanqi Lin, Shiyu Huang, Tim Pearce, Wenze Chen, and Wei-Wei Tu. Tizero: Mastering multi-agent football with curriculum learning and self-play. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, pp. 67–76, 2023.

Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994*, pp. 157–163. Elsevier, 1994.

Siqi Liu, Marc Lanctot, Luke Marris, and Nicolas Heess. Simplex neural population learning: Any-mixture bayes-optimality in symmetric zero-sum games. In *International Conference on Machine Learning*, pp. 13793–13806. PMLR, 2022a.

Siqi Liu, Guy Lever, Zhe Wang, Josh Merel, SM Ali Eslami, Daniel Hennes, Wojciech M Czarnecki, Yuval Tassa, Shayegan Omidshafiei, Abbas Abdolmaleki, et al. From motor control to team play in simulated humanoid football. *Science Robotics*, 7(69):eabo0235, 2022b.

Siqi Liu, Luke Marris, Daniel Hennes, Josh Merel, Nicolas Heess, and Thore Graepel. Neupl: Neural population learning. *arXiv preprint arXiv:2202.07415*, 2022c.

Xiangyu Liu, Hangtian Jia, Ying Wen, Yujing Hu, Yingfeng Chen, Changjie Fan, Zhipeng Hu, and Yaodong Yang. Towards unifying behavioral and response diversity for open-ended learning in zero-sum games. *Advances in Neural Information Processing Systems*, 34:941–952, 2021.

Roberto Lucchetti. *A primer in game theory*. Società Editrice Esculapio, 2011.

Anuj Mahajan, Tabish Rashid, Mikayel Samvelyan, and Shimon Whiteson. Maven: Multi-agent variational exploration. *Advances in neural information processing systems*, 32, 2019.

Luke Marris, Paul Muller, Marc Lanctot, Karl Tuyls, and Thore Graepel. Multi-agent training beyond zero-sum with correlated equilibrium meta-solvers. In *International Conference on Machine Learning*, pp. 7480–7491. PMLR, 2021.

Stephen McAleer, John B Lanier, Roy Fox, and Pierre Baldi. Pipeline psro: A scalable approach for finding approximate nash equilibria in large games. *Advances in neural information processing systems*, 33:20238–20248, 2020.

Stephen McAleer, John B Lanier, Kevin A Wang, Pierre Baldi, and Roy Fox. Xdo: A double oracle algorithm for extensive-form games. *Advances in Neural Information Processing Systems*, 34:23128–23139, 2021.

Stephen McAleer, John Banister Lanier, Kevin Wang, Pierre Baldi, Roy Fox, and Tuomas Sandholm. Self-play psro: Toward optimal populations in two-player zero-sum games. *arXiv preprint arXiv:2207.06541*, 2022a.

Stephen McAleer, Kevin Wang, John Lanier, Marc Lanctot, Pierre Baldi, Tuomas Sandholm, and Roy Fox. Anytime psro for two-player zero-sum games. *arXiv preprint arXiv:2201.07700*, 2022b.

Stephen Marcus McAleer, Gabriele Farina, Gaoyue Zhou, Mingzhi Wang, Yaodong Yang, and Tuomas Sandholm. Team-psro for learning approximate tmecor in large team games via cooperative reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

H Brendan McMahan, Geoffrey J Gordon, and Avrim Blum. Planning in the presence of cost functions controlled by an adversary. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pp. 536–543, 2003.

Naoki Mizukami and Yoshimasa Tsuruoka. Building a computer mahjong player based on monte carlo simulation and opponent models. In *2015 IEEE Conference on Computational Intelligence and Games (CIG)*, pp. 275–283. IEEE, 2015.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Matej Moravčík, Martin Schmid, Neil Burch, Viliam Lisỳ, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 356(6337):508–513, 2017.

P Muller, S Omidshafiei, M Rowland, K Tuyls, J Pérolat, S Liu, D Hennes, L Marris, M Lanctot, E Hughes, et al. A generalized training approach for multiagent learning. In *ICLR*, pp. 1–35. ICLR, 2020.

Paul Muller, Mark Rowland, Romuald Elie, Georgios Piliouras, Julien Perolat, Mathieu Lauriere, Raphael Marinier, Olivier Pietquin, and Karl Tuyls. Learning equilibria in mean-field games: Introducing mean-field psro. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, pp. 926–934, 2022.

Remi Munos, Michal Valko, Daniele Calandriello, Mohammad Gheshlaghi Azar, Mark Rowland, Zhaohan Daniel Guo, Yunhao Tang, Matthieu Geist, Thomas Mesnard, Côme Fiegel, et al. Nash learning from human feedback. In *International Conference on Machine Learning*, pp. 36743–36768. PMLR, 2024.

Jan Mycielski. Games with perfect information. *Handbook of game theory with economic applications*, 1: 41–70, 1992.

John F Nash et al. Non-cooperative games. 1950.

Shayegan Omidshafiei, Christos Papadimitriou, Georgios Piliouras, Karl Tuyls, Mark Rowland, Jean-Baptiste Lespiau, Wojciech M Czarnecki, Marc Lanctot, Julien Perolat, and Remi Munos. α-rank: Multi-agent evaluation by evolution. *Scientific reports*, 9(1):9937, 2019.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

Nicolas Perez-Nieves, Yaodong Yang, Oliver Slumbers, David H Mguni, Ying Wen, and Jun Wang. Modelling behavioural diversity for learning in open-ended games. In *International conference on machine learning*, pp. 8514–8524. PMLR, 2021.

Julien Perolat, Remi Munos, Jean-Baptiste Lespiau, Shayegan Omidshafiei, Mark Rowland, Pedro Ortega, Neil Burch, Thomas Anthony, David Balduzzi, Bart De Vylder, et al. From poincaré recurrence to convergence in imperfect information games: Finding equilibrium via regularization. In *International Conference on Machine Learning*, pp. 8525–8535. PMLR, 2021.

Julien Perolat, Bart De Vylder, Daniel Hennes, Eugene Tarassov, Florian Strub, Vincent de Boer, Paul Muller, Jerome T Connor, Neil Burch, Thomas Anthony, et al. Mastering the game of stratego with model-free multiagent reinforcement learning. *Science*, 378(6623):990–996, 2022.

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, pp. 4295–4304. PMLR, 2018.

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder De Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Monotonic value function factorisation for deep multi-agent reinforcement learning. *The Journal of Machine Learning Research*, 21(1):7234–7284, 2020.

Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 44(1.2):206–226, 2000.

Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim GJ Rudner, Chia-Man Hung, Philip HS Torr, Jakob Foerster, and Shimon Whiteson. The starcraft multi-agent challenge. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 2186–2188, 2019.

Jonathan Schaeffer, Joseph Culberson, Norman Treloar, Brent Knight, Paul Lu, and Duane Szafron. A world championship caliber checkers program. *Artificial Intelligence*, 53(2-3):273–289, 1992.

Martin Schmid, Neil Burch, Marc Lanctot, Matej Moravcik, Rudolf Kadlec, and Michael Bowling. Variance reduction in monte carlo counterfactual regret minimization (vr-mccfr) for extensive form games using baselines. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 2157–2164, 2019.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

Lloyd S Shapley. Stochastic games. *Proceedings of the national academy of sciences*, 39(10):1095–1100, 1953.

Brian Sheppard. World-championship-caliber scrabble. *Artificial Intelligence*, 134(1-2):241–275, 2002.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

Eric Steinberger. Single deep counterfactual regret minimization. *arXiv preprint arXiv:1901.07621*, 2019.

Eric Steinberger, Adam Lerer, and Noam Brown. Dream: Deep regret minimization with advantage baselines and model-free learning. *arXiv preprint arXiv:2006.10410*, 2020.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Gokul Swamy, Christoph Dann, Rahul Kidambi, Zhiwei Steven Wu, and Alekh Agarwal. A minimaximalist approach to reinforcement learning from human feedback. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 47345–47377, 2024.

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

Yichuan Tang. Towards learning multi-agent negotiations via self-play. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pp. 0–0, 2019.

Peter D Taylor and Leo B Jonker. Evolutionary stable strategies and game dynamics. *Mathematical biosciences*, 40(1-2):145–156, 1978.

Tencent. Luckyj: Deep learning mahjong ai. https://twitter.com/LuckyJ1443836.

Gerald Tesauro and Gregory Galperin. On-line policy improvement using monte-carlo search. *Advances in Neural Information Processing Systems*, 9, 1996.

Shingo Tsunoda. Tenhou. https://tenhou.net/.

Dwango Media Village. Naga: Deep learning mahjong ai. https://dmv.nico/ja/articles/mahjong_ai_naga/.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

Chenguang Wang, Zhouliang Yu, Stephen McAleer, Tianshu Yu, and Yaodong Yang. Asp: Learn a universal neural solver! *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

Xiangjun Wang, Junxiao Song, Penghui Qi, Peng Peng, Zhenkun Tang, Wei Zhang, Weimin Li, Xiongjun Pi, Jujie He, Chao Gao, et al. Scc: An efficient deep reinforcement learning agent mastering the game of starcraft ii. In *International conference on machine learning*, pp. 10905–10915. PMLR, 2021.

Yi Wang, Hui Tang, Lichao Huang, Lulu Pan, Lixiang Yang, Huanming Yang, Feng Mu, and Meng Yang. Self-play reinforcement learning guides protein engineering. *Nature Machine Intelligence*, 5(8):845–860, 2023.

Kevin Waugh, Dustin Morrill, James Bagnell, and Michael Bowling. Solving games with functional regret estimation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015.

Michael P Wellman, Karl Tuyls, and Amy Greenwald. Empirical game theoretic analysis: A survey. *Journal of Artificial Intelligence Research*, 82:1017–1076, 2025.

Yue Wu, Zhiqing Sun, Huizhuo Yuan, Kaixuan Ji, Yiming Yang, and Quanquan Gu. Self-play preference optimization for language model alignment. *arXiv preprint arXiv:2405.00675*, 2024.

Ziyan Xiong, Bo Chen, Shiyu Huang, Wei-Wei Tu, Zhaofeng He, and Yang Gao. Mqe: Unleashing the power of interaction with multi-agent quadruped environment. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5918–5924. IEEE, 2024.

Zelai Xu, Yancheng Liang, Chao Yu, Yu Wang, and Yi Wu. Fictitious cross-play: Learning global nash equilibrium in mixed cooperative-competitive games. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, pp. 1053–1061, 2023.

Zelai Xu, Chao Yu, Fei Fang, Yu Wang, and Yi Wu. Language agents with reinforcement learning for strategic play in the werewolf game. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 55434–55464, 2024.

Zelai Xu, Chao Yu, Ruize Zhang, Huining Yuan, Xiangmin Yi, Shilong Ji, Chuqi Wang, Wenhao Tang, and Yu Wang. Volleybots: A testbed for multi-drone volleyball game combining motion control and strategic play. *arXiv preprint arXiv:2502.01932*, 2025.

Nikolai Yakovenko, Liangliang Cao, Colin Raffel, and James Fan. Poker-cnn: A pattern learning strategy for making draws and bets in poker games using convolutional networks. In *Proceedings of the aaai conference on artificial intelligence*, volume 30, 2016.

Guan Yang, Minghuan Liu, Weijun Hong, Weinan Zhang, Fei Fang, Guangjun Zeng, and Yue Lin. Perfectdou: Dominating doudizhu with perfect information distillation. *Advances in Neural Information Processing Systems*, 35:34954–34965, 2022.

Jian Yao, Weiming Liu, Haobo Fu, Yaodong Yang, Stephen McAleer, Qiang Fu, and Wei Yang. Policy space diversity for non-transitive games. *Advances in Neural Information Processing Systems*, 36:67771–67793, 2023.

Deheng Ye, Guibin Chen, Wen Zhang, Sheng Chen, Bo Yuan, Bo Liu, Jia Chen, Zhao Liu, Fuhao Qiu, Hongsheng Yu, et al. Towards playing full moba games with deep reinforcement learning. *Advances in Neural Information Processing Systems*, 33:621–632, 2020a.

Deheng Ye, Zhao Liu, Mingfei Sun, Bei Shi, Peilin Zhao, Hao Wu, Hongsheng Yu, Shaojie Yang, Xipeng Wu, Qingwei Guo, et al. Mastering complex control in moba games with deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 6672–6679, 2020b.

Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of ppo in cooperative multi-agent games. *Advances in neural information processing systems*, 35:24611–24624, 2022.

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. Self-rewarding language models. *arXiv preprint arXiv:2401.10020*, 2024.

Daochen Zha, Kwei-Herng Lai, Yuanpu Cao, Songyi Huang, Ruzhe Wei, Junyu Guo, and Xia Hu. Rlcard: A toolkit for reinforcement learning in card games. *arXiv preprint arXiv:1910.04376*, 2019.

Daochen Zha, Jingru Xie, Wenye Ma, Sheng Zhang, Xiangru Lian, Xia Hu, and Ji Liu. Douzero: Mastering doudizhu with self-play deep reinforcement learning. In *international conference on machine learning*, pp. 12333–12344. PMLR, 2021.

Enmin Zhao, Renye Yan, Jinqiu Li, Kai Li, and Junliang Xing. Alphaholdem: High-performance artificial intelligence for heads-up no-limit poker via end-to-end reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 4689–4697, 2022a.

Youpeng Zhao, Jian Zhao, Xunhan Hu, Wengang Zhou, and Houqiang Li. Douzero+: Improving doudizhu ai by opponent modeling and coach-guided learning. In *2022 IEEE Conference on Games (CoG)*, pp. 127–134. IEEE, 2022b.

Ming Zhou, Jingxiao Chen, Ying Wen, Weinan Zhang, Yaodong Yang, Yong Yu, and Jun Wang. Efficient policy space response oracles. *arXiv preprint arXiv:2202.00633*, 2022.

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

Fengming Zhu Ziyang Li, Kaiwen Zhu. Wekick. `https://www.kaggle.com/c/google-football/discussion/202232`, 2020.

A Appendix

A.1 Abbreviations in Alphabetical Order

ABR Approximate Best Response

AI Artificial Intelligence

ARM Advantage-based Regret Minimization

ARMAC Advantage Regret-Matching Actor-Critic

BR Best Response

CCE Coarse Correlated Equilibrium

CE Correlated Equilibrium

CFR Counterfactual Regret Minimization

COP	Combinatorial Optimization Problem
CSPL	Curriculum Self-Play Learning
CVRP	Capacitated Vehicle Routing Problem
DCFR	Discounted Counterfactual Regret Minimization
DO	Double Oracle
EGTA	Empirical Game-Theoretic Analysis
FP	Fictitious Play
FSP	Fictitious Self-Play
FTW	For The Win
FXP	Fictitious Cross-Play
GAIL	Generative Adversarial Imitation Learning
GRF	Google Research Football
HULHE	Heads-Up Limit Texas Hold'em
HUNL	Heads-Up No-Limit Texas Hold'em
JPSRO	Joint Policy-Space Response Oracle
LCFR	Linear Counterfactual Regret Minimization
LLM	Large Language Model
MARL	Multi-Agent Reinforcement Learning
MCCFR	Monte Carlo Counterfactual Regret Minimization
MCTS	Monte Carlo Tree Search
MDP	Markov Decision Process
MG	Markov Game
ML	Machine Learning
MOBA	Multiplayer Online Battle Arena
MSS	Meta-Strategy Solver
NE	Nash Equilibrium
NeuPL	Neural Population Learning
NFSP	Neural Fictitious Self-Play
ODO	Online Double Oracle
PBR	Preference-based Best Response
PBS	Public Belief State
PFSP	Prioritized Fictitious Self-Play
POMG	Partially Observable Markov Game
PSRO	Policy-Space Response Oracle
RCFR	Regression Counterfactual Regret Minimization
ReBeL	Recursive Belief-based Learning
RL	Reinforcement Learning
R-NaD	Regularized Nash Dynamics
RTS	Real-Time Strategy
SD-CFR	Single Deep Counterfactual Regret Minimization

SPAG	Self-Play of Adversarial Game
SPIN	Self-Play fIne-tuNing
TSP	Traveling Salesman Problem
URR	UnRestricted-Restricted
XDO	Extensive-Form Double Oracle