FightLadder: A Benchmark for Competitive Multi-Agent Reinforcement Learning

Wenzhe Li[1], Zihan Ding[1], Seth Karten[1], and Chi Jin[1]

[1]Princeton University*

Abstract

Recent advances in reinforcement learning (RL) heavily rely on a variety of well-designed benchmarks, which provide environmental platforms and consistent criteria to evaluate existing and novel algorithms. Specifically, in multi-agent RL (MARL), a plethora of benchmarks based on cooperative games have spurred the development of algorithms that improve the scalability of cooperative multi-agent systems. However, for the competitive setting, a lightweight and open-sourced benchmark with challenging gaming dynamics and visual inputs has not yet been established. In this work, we present FightLadder, a real-time fighting game platform, to empower competitive MARL research. Along with the platform, we provide implementations of state-of-the-art MARL algorithms for competitive games, as well as a set of evaluation metrics to characterize the performance and exploitability of agents. We demonstrate the feasibility of this platform by training a general agent that consistently defeats 12 built-in characters in single-player mode, and expose the difficulty of training a non-exploitable agent without human knowledge and demonstrations in two-player mode. FightLadder provides meticulously designed environments to address critical challenges in competitive MARL research, aiming to catalyze a new era of discovery and advancement in the field. Videos and code at https://sites.google.com/view/fightladder/home.

1 Introduction

As an active branch of artificial intelligence (AI), deep reinforcement learning (DRL) has achieved significant success in various domains, including, but not limited to, strategic games (Silver et al., 2016; Li et al., 2020; Moravvcík et al., 2017; Vinyals et al., 2019; Berner et al., 2019), robotics control (Lillicrap et al., 2015; Andrychowicz et al., 2020b; Brohan et al., 2022), and large language models alignment (Ouyang et al., 2022). Underpinning these rapid advances are not only the development of sample-efficient RL algorithms but also the availability of well-designed benchmarks. These benchmarks provide environmental platforms, unify evaluation protocols, enable comparisons of state-of-the-art methods, motivate improved solutions, and guide

*Email: {wenzhe.li,zihand,sethkarten,chij}@princeton.edu.

Figure 1: FightLadder currently supports various cross-platform video fighting games: *Street Fighter II* (Genesis platform), *Street Fighter III* (Arcade platform), *Fatal Fury 2* (Genesis platform), *Mortal Kombat* (Genesis platform), and *The King of Fighters '97* (Neo Geo platform).

practical applications. As an example, policy proximal optimization (PPO) (Schulman et al., 2017) demonstrates its superior performance across different single-agent RL benchmarks, hence being considered as one of the most widely adopted single-agent RL algorithms (Andrychowicz et al., 2020a). In the realm of multi-agent reinforcement learning (MARL), while a series of benchmarks have also been proposed, most of them focus on fully cooperative settings. For competitive environments, some platforms simulate games with tabular representations and relatively simple dynamics, such as board games, while others, based on complex game engines, require significant computational resources and expert knowledge, such as Starcraft II and DOTA. To advance research on competitive multi-agent reinforcement learning (MARL) and transform game-theoretical results into practical applications, a fully competitive game platform that strikes the right balance between complexity, efficiency, and generality is urgently needed.

Multi-agent games are known to be more challenging than single-agent ones due to the additional non-stationarity introduced by the interactions with other players. Among different types of interactions, fully competitive settings can be rather difficult. People have a long history of designing and playing competitive games, as well as building strong AI opponents to make the game more challenging and hence intriguing. Previous AI research has investigated the solutions of competitive games using RL, but mostly for small-scale games like Backgammon (Tesauro et al., 1995) or other board games (Schrittwieser et al., 2020; Brown & Sandholm, 2018, 2019). Moreover, this line of work mostly uses state vectors as inputs, which is arguably easier than directly learning from raw pixel inputs that commonly appear in most popular video games. In contrast, this paper considers fighting games, which feature rich policy space, and significant depth in strategy — including catching specific timing, counter-attack by exploiting the stiffness of the opponents, managing energy resources, etc. Moreover,

these games also have a rather large number of characters with distinct move-sets which add another layer of complexity for AI agents to master the game. As a result, we are motivated to build a platform for a series of fighting games, with image inputs and complex fighting dynamics, to serve as a challenging competitive multi-player platform for the broad AI research community.

Apart from the game platform, the evaluation criteria and benchmark results for certain game settings are essential for boosting the field. MARL has been greatly investigated in the past few years for solving multi-player games, from both theoretical and empirical perspectives. A large number of algorithms have been proposed according to specific settings (Sunehag et al., 2017; Yu et al., 2022; Lowe et al., 2017; Silver et al., 2018; Lanctot et al., 2017; Vinyals et al., 2019; Ding et al., 2022). Nonetheless, for competitive game settings, there is a lack of unified evaluation criteria with thorough comparisons among different approaches.

In this work, we present FightLadder, a competitive two-player games benchmark. Our contributions are three-fold: We build the FightLadder platform to support five two-player fighting games, with ease to extend to other games in the future. The games support various observation spaces involving rendered images. Based on prior work, we provide implementations of the most popular algorithms for solving these competitive games, including an AlphaStar league training algorithm (Vinyals et al., 2019) and policy space response oracle (Lanctot et al., 2017). Furthermore, a unified evaluation framework with *Elo rating* and *exploitability* tests are provided alongside the game platforms and algorithm library. We report experimental results using the above toolkits to serve as the baselines for two-player competitive game settings. One important challenge of MARL is its diverse nature, which includes collaborative games, competitive games, two-player games, and multiplayer games, all of which have rather different problem structures, properties, and solution concepts. While it is promising to develop a unified solution that addresses them all together, in this work, we empirically demonstrate that *to some extent, existing methods are still limited in solving competitive two-player zero-sum games alone when combined with visual input, rich strategy space, and lack of extensive human demonstration*. We hope that FightLadder, which particularly focuses on this fundamental two-player setting, can serve as a stepping stone for the research community to develop effective self-play style algorithms to tackle it first before moving on to even more complicated scenarios, and inspire future directions that involve more types of interactions.

2 Related Work

MARL Environments. MARL environments can be categorized into three types according to the payoff structure of the game: *fully cooperative*, *fully competitive*, and *general*.

Existing environments for *fully cooperative* games are designed for various scenarios, including simulated games like MAMuJoCo (Peng et al., 2021), card games like Hanabi (Bard et al., 2020), video games like small-scale StarCraft SMAC (Samvelyan et al., 2019) and Google Research Football (Kurach et al., 2020), as well as practical scenarios like Traffic Junction (Sukhbaatar et al., 2016) in a grid world, Flatland (Mo-

hanty et al., 2020) for railway networks, network load balancing (Yao & Ding, 2022) and CityFlow (Zhang et al., 2019) for city traffic. Cooperative environments feature a single reward function shared by all agents, which makes them distinct from competitive games.

On the other hand, the *fully competitive* game benchmarks are relatively underdeveloped. Prior competitive environments are either on games with low-dimensional or discrete state space such as Pommerman (Resnick et al., 2018) and board games (Tesauro et al., 1995; Schrittwieser et al., 2020; Brown & Sandholm, 2018); or complex games with image input that require a significant amount of computational resources, such as Starcraft II (Vinyals et al., 2019) or DOTA (Berner et al., 2019). The fighting game environments proposed in this paper strike the right balance between complexity, efficiency, and generality. A few previous works also have explored fighting games: Go et al. (2023) focuses on developing an algorithm for a single fighting game—street fighter, as opposed to this paper which provides an environment that supports various fighting games. While Palmas (2022) provides a platform for fighting games, most of its efforts have been focused on the single-agent setting. It lacks explicit criteria for two-player scenarios with adaptive opponents, and does not provide a benchmark evaluating existing competitive MARL algorithms. Khan et al. (2022) focuses on fighting games in the blind setting where agents have to rely on acoustic inputs to play.

Finally, there are also a number of environments for *general* multiagent games that feature both cooperation and competition, including MPE (Mordatch & Abbeel, 2018), MAgent (Zheng et al., 2018), Hide-and-Seek (Baker et al., 2019), DMLab2D (Beattie et al., 2020), Arena (Song et al., 2020), Smarts (Zhou et al., 2020), Neural MMO (Suarez et al., 2021), PettingZoo (Terry et al., 2021), MATE (Pan et al., 2022), etc. Generic multi-agent general-sum games are rather challenging to evaluate — even the optimal solution concepts remain elusive. In contrast, the fully competitive setting considered in this paper presents clear game-theoretic properties and well-defined solution concepts. We also remark that while a number of the platforms above support several fully competitive games, they did not provide carefully designed evaluation toolkits as well as extensive baselines for competitive MARL algorithms.

MARL Algorithms and Evaluation. To solve multi-agent learning tasks, researchers have proposed algorithms and built libraries for ease of usage and evaluation. PyMARL (Samvelyan et al., 2019) is an initial MARL library built for solving SMAC tasks, while PyMARL2 (Hu et al., 2021) extends PyMARL with QMIX (Rashid et al., 2020). EPyMARL (Papoudakis et al., 2020) is also an extension of PyMARL, as a unified library for cooperative games supporting different learning paradigms including centralized and decentralized learning, value decomposition, etc. MARLlib (Hu et al., 2023) includes major cooperative MARL algorithms like VDN (Sunehag et al., 2017), MAPPO (Yu et al., 2022), MADDPG (Lowe et al., 2017), etc. More recent libraries include Pantheonrl (Sarkar et al., 2022), MAlib (Zhou et al., 2023), etc. These libraries mainly support MARL algorithms for cooperative games, lacking support for solving competitive games.

On the other hand, there is a line of research for solving competitive games with algorithms like self-play (Silver et al., 2018), fictitious play (Brown, 1951), Nash Q-

learning (Hu & Wellman, 2003; Ding et al., 2022), double oracle (McMahan et al., 2003), policy space response oracle (PSRO) (Lanctot et al., 2017) and league training (Vinyals et al., 2019). A unified benchmark remains missing to compare and evaluate the efficiency these algorithms on the same set of tasks, especially when combined with deep RL. This paper addresses this issue in the fully competitive setting. We concentrate on two-player zero-sum games, and propose a platform for fighting-style fully competitive games, along with a baseline implementation and evaluation of popular algorithms.

3 Multi-Agent Reinforcement Learning

FightLadder is designed to motivate novel algorithms for fully competitive two-player games in the domains of MARL and game theory. Markov Games (MGs) (Shapley, 1953) generalize single-player Markov Decision Processes (MDPs) into multi-player settings. Each player has its own utility and optimizes its policy to maximize the utility. The two-player zero-sum setting in MG represents a competitive relationship between the two players. With a shaped dense reward, the games can be generalized to general-sum.

We denote a finite-horizon two-player general-sum partially observable MG as $\mathrm{POMG}(\mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{B}, \mathbb{P}, \mathbb{O}, \{r\}_{i=1}^2, H)$. $\mathcal{S}$ is the state space, which can be partially observable and transformed through an observation emission function $\mathbb{O}$: $\mathcal{S} \to \mathcal{O}$ to the observation space $\mathcal{O}$. $\mathcal{A}$ and $\mathcal{B}$ are action spaces for two players, respectively. $\mathbb{P}(\cdot|s, a, b)$ is the state transition distribution, $r_i : \mathcal{S} \times \mathcal{A} \times \mathcal{B} \to \mathbb{R}$ is the reward function for the i-th player. In the zero-sum setting, two reward functions satisfy the zero-sum payoff structure $r_1 + r_2 = 0$. H is the horizon length. We denote the policies of two players as μ and ν, respectively. $V_i^{\mu,\nu} : \mathcal{S} \to \mathbb{R}$ represents the value function for player i evaluated with policies μ and ν, which can be expanded as the expected cumulative reward starting from the state s,

$$V_i^{\mu,\nu}(s) := \mathbb{E}_{\mu,\nu}\big[\textstyle\sum_{h=1}^{\infty} r_i(s_h, a_h, b_h)\big|s_1 = s\big].$$

In zero-sum games, we have $V_1^{\mu,\nu}(s) = -V_2^{\mu,\nu}(s), \forall s \in \mathcal{S}$ and define $V^{\mu,\nu}(s) = V_1^{\mu,\nu}(s)$ for simplicity.

Definition 3.1 (Best Response)**.** For any policy of the first player μ, there exists a *best response* (BR) against it from the second player, which is a policy $\nu^\dagger(\mu)$ satisfying $V_{2,h}^{\mu,\nu^\dagger(\mu)}(s) = \max_\nu V_{2,h}^{\mu,\nu}(s)$ for any $(s, h) \in \mathcal{S} \times [H]$. We denote $V_{2,h}^{\mu,\dagger} := V_{2,h}^{\mu,\nu^\dagger(\mu)}$ for simplification. $V_{2,h}^{\mu,\nu}(s)$ is the value function of the second player. BR against the second player can be defined similarly.

Definition 3.2 (Nash Equilibrium)**.** The *Nash equilibrium* (NE) in zero-sum setting is defined as a pair of policies $(\mu^\star, \nu^\star)$ satisfying the following minimax equation:

$$\max_\mu \min_\nu V^{\mu,\nu}(s) = V^{\mu^\star,\nu^\star}(s) = \min_\nu \max_\mu V^{\mu,\nu}(s).$$

Definition 3.3 (Exploitability)**.** The exploitability for a policy μ of the first player is defined as $V_2^{\mu,\dagger}(s_1) - V_2^{\mu^\star,\nu^\star}(s_1)$, i.e., the value of its BR policy $\nu^\dagger(\mu)$ or the

suboptimality gap from the NE value. The exploitability of the other side policy ν can be defined accordingly.

Note that NE strategies will always lead to zero exploitability, thus approaching the non-exploitable strategies is a reasonable pursuit for the game.

4 FightLadder

In this section, we present technical details of FightLadder. In the following part, we first introduce different game settings of FightLadder, followed by elaborating elements of MGs corresponding to the environment, and conclude with highlighting features of our benchmark.

4.1 Scenarios

FightLadder provides a flexible interface between modern game emulators (Murphy, 2013; Nichol et al., 2018) and algorithm developers. Thanks to its flexibility, FightLadder can support a wide range of classical fighting games over the past decades, including Street Fighter, Mortal Kombat, Fatal Fury, and The King of Fighters, some of which are still very popular nowadays. Figure 1 shows screenshots of several fighting games provided by FightLadder. With this diverse set of supported games, we can benchmark algorithms on various fighting scenarios differing in backgrounds, characters, and moving dynamics, which can further motivate novel algorithms that are general rather than overfitting to one specific game. For better readability and clarity, we would use Street Fighter as an example for illustration and evaluation in the rest of the paper. The other fighting games are very similar, and readers could refer to Appendix A.2 for more details. We name each scenario in the form *[game alias]_[character left]_vs_[character right]*, for example *sf_ryu_vs_ryu* in Street Fighter.

While FightLadder mainly focuses on the competitive two-player setting, the nature of fighting games allows it to be seamlessly deployed to the single-player scenario where the agent's task is to compete against a built-in game AI (e.g., *sf_ryu_vs_ryu(cpu)*). Under this single-player setting, users have the freedom to choose characters and set up the difficulty of the scripted AI opponent. Moreover, our benchmark also supports training in a much more challenging full-game scenario (e.g., *sf_ryu_full_game*), where the agent needs to defeat all 12 characters controlled by computers with the difficulty progressively increasing. As we shall see in later experiments, this scenario could also serve as a sanity check for our baseline algorithms to see whether they could learn effective behaviors from the environment.

4.2 State and Observations

We define the state space $\mathcal{S}$ as the complete set of attributes stored in the game emulator after each step of action. Same as human players, the agent is not allowed to access the underlying full state but can only access the observation space $\mathcal{O}$ of pixels, which forms a 128×100 RGB image corresponding to the rendered screen. This image includes the

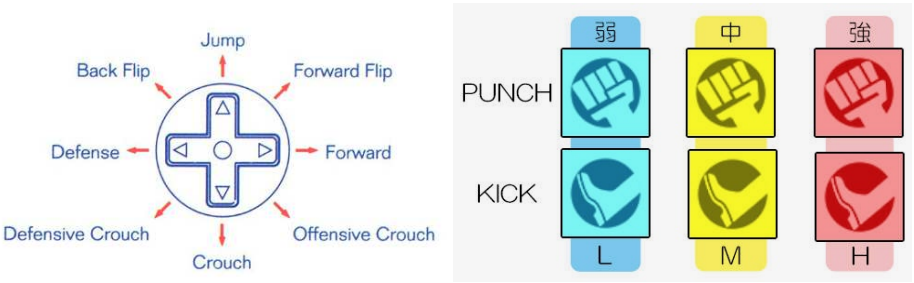

Figure 2: Motion and attack action spaces of fighting games. Images are adapted from Instruction Manual of Street Fighter II.

position and movement of both sides of the players, as well as the hit-point bar and the round timer on the top of the screen. At every step, a configurable number of images are stacked as the input of the agent.

While we use pixels as default observations, we also provide an interface for users to access additional information about the game status, including position, hit-point, and exact countdown number for agents on both sides. Users can leverage these attributes to better understand the agent's behavior or augment feature representations. More details are provided in Appendix A.2.

4.3 Action Space

In fighting games, two players share the same action space $\mathcal{A}$. The native *human action space* $\mathcal{A}_{\text{human}}$ is designed to mimic the joystick control of arcade games, which is a 12-dimensional binary space (['B', 'A', 'MODE', 'START', 'UP', 'DOWN', 'LEFT', 'RIGHT', 'C', 'Y', 'X', 'Z']) with each dimension representing a button being pressed or not. Note that due to the nature of fighting game engines, this space contains many redundant actions that are invalid, for instance, moving in opposite directions or moving and attacking at the same moment. To filter out these redundant actions and to construct a more structured space, we develop a categorical *transformed action space* $\mathcal{A}_{\text{trans}}$ through an encoding function $F : \mathcal{A}_{\text{human}} \rightarrow \mathcal{A}_{\text{trans}}$. Specifically, $\mathcal{A}_{\text{trans}}$ is the joint set of a direction move set $\mathcal{A}_{\text{motion}}$={defense, forward, jump, crouch, back flip, front flip, offensive crouch, defensive crouch} and an attack move set $\mathcal{A}_{\text{attack}}$={light punch, medium punch, hard punch, light kick, medium kick, hard kick}, as shown in Figure 2. Each action will remain a number of frames according to users' configuration. The games also have special techniques called *close attack*, i.e., Throws and Holds, which can be applied in certain regions near the opponent.

In addition to the standard move set, one signature element of fighting games is *special moves*, which is a kind of powerful attack or maneuver that requires the player to follow a specific action sequence (i.e., sequential keys combination, or combination of key holding and key pressing), with an example depicted in Figure 3. These moves usually have special properties (e.g., invincibility frames, larger coverage, etc.) and play a critical role in the strategy and depth of the game. They are especially useful for higher levels of play, from which players could create complex combos and outperform

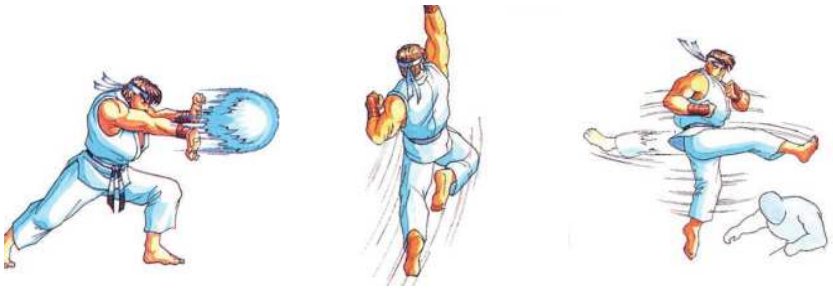

Figure 3: Example of special moves for character Ryu in StreetFighter II (left to right): Fireball, Dragon Punch, Hurricane Kick. Images are adapted from Instruction Manual of Street Fighter II.

opponents. However, we observe that learning to perform special moves from scratch can be challenging to baseline algorithms, as it requires the agent to memorize frames and actions in previous steps and accurately perform the next action in the action sequence of special moves. Moreover, the special moves can be different from character to character, which increases the difficulty of the game. Therefore, to alleviate this challenge, we also include hard-coded special move lists as one part of the action space so that the agent can directly access special moves with one single action.

4.4 Rewards

Sparse Reward. Both sides of the agents are to maximize their win rate for each round of the game. The *sparse reward* r_{sparse} assigns +1 for the winner and -1 for the loser at the end of each episode. In the sparse reward setting, all fighting games are two-player zero-sum games, which are theoretically guaranteed to exist at least one Nash Equilibrium (Filar & Vrieze, 2012), which directly induces a pair of non-exploitable policies.

Win Rate. For two players A and B, policy π_A winning against policy π_B can be defined as a reward relationship $r^A_{\text{sparse}}(\pi_A, \pi_B) > r^B_{\text{sparse}}(\pi_A, \pi_B)$ in a single match, with r^A_{sparse} and r^B_{sparse} as the sparse reward for players A and B in the zero-sum setting. The win rate is defined as the probability of winning as $p(\pi_A \succ \pi_B)$.

Shaped Dense Reward. While sparse reward is straightforward for evaluation, we discover that baseline algorithms could not effectively learn to behave well from such a sparse signal. To address this issue, we introduce a *shaped dense reward* r_{dense} for training, which is a weighted sum of the hit-point damage inflicted by the agent on the opponent and the damage it receives, together with a bonus (penalty) for winning (losing) the game. Specific format of this reward refers to Appendix A.1. The dense reward r_{dense} is chosen to coincide with the win rate of the policy, such that $\pi_A \succ \pi_B$ will always lead to $r^A_{\text{dense}}(\pi_A, \pi_B) > r^B_{\text{dense}}(\pi_A, \pi_B)$ in expectation. The dense reward also offers some flexibility, that the user can control the agent's aggressiveness by configuring the weighing scales in the reward function.

Table 1: FPS and memory usage of several open-sourced platforms.

Environment	Speed (FPS)	Memory (MB)
FightLadder (Ours)	1935.76	195.46
SMACv2	146.72	876.96
PettingZoo Atari	6268.18	32.13
DMLab2D	1144.27	47.41

4.5 Features

We remark on the following features of the proposed benchmark that could benefit MARL research.

Rich Strategy Space. One key feature of our benchmark is the rich strategy space as the nature of fighting games, which is particularly beneficial to the development of game-theoretical algorithms. To name a few, fighting games require players to consider **(a) character diversity:** each character has a unique skill set with different strengths and weaknesses, so one needs to master the strategy and counter-strategy of all possible opponents, and even reason how to select and order characters when they have the freedom to do so; **(b) complexity of mechanics:** fighting games are designed with sophisticate mechanics such as invincibility frame, hitboxes, and combo systems, which are challenging for micromanagement of characters; and **(c) adversarial opponents:** opponents may progressively adapt their policies to players' policies, thus finding non-exploitable policies is crucial in mastering fighting games.

Various Difficulty Levels. FightLadder provides several kinds of scenarios: single-player mode against one CPU player (e.g., *sf_ryu_vs_ryu(cpu)*), single-player mode full game (e.g., *sf_ryu_full_game*), two-player mode (e.g., *sf_ryu_vs_ryu*), team mode (supported in some games such as The King of Fighters). The difficulty levels are increasing in this order, as two-player mode (no CPU) introduces additional non-stationary (opponents can be adaptive), and team mode offers a richer strategy space. Moreover, FightLadder supports specifying **arbitrary difficulty levels** of CPUs and **arbitrary characters** for both the player and its opponent. This enriches the features of our platform and the diversity of strategy space.

Computational Efficiency. FightLadder also enjoys efficient computation for its usage, and the comparison with several other popular game environments is shown in Table 1. The frame rate is 13 times faster than SMACv2, with one-fourth usage of the memory. While FightLadder is less efficient than PettingZoo Atari, it provides more game complexity. The balance of complexity and low computational cost is important for evaluating algorithms at scale.

Fidelity and Popularity. FightLadder allows testing agents in full-length fighting games with an interface similar to human perception, thus providing a high-fidelity evaluation of competitive RL algorithms. Moreover, fighting games have been gaining

popularity since they were released, making it easier to test the learned RL agents against human expert players.

Open-Source and Compatibility. FightLadder is designed for the broad RL research community, so we make efforts to improve the ease of usage and make it accessible to all potential users. It is compatible with the Gym (Brockman et al., 2016) interface so that users can leverage off-the-shelf RL algorithms implementation.

Customization, Extension, and Flexibility. FightLadder is extremely flexible for configuration and extension. For customization, the users can customize action spaces (human/transformed action), reward functions (sparse/tunable shaped dense reward), number of frames to be observed per step, as well as access to additional information to help training. Moreover, our platform is built upon popular modern game emulators so that it is easy to extend to other games not provided by us. Specifically, it supports Gym Retro and MAMEToolkit, which already support a wide range of games. This extension capability of diverse games is provided by our platform with minimal engineering efforts. Please check our open-source project[1] for more details.

5 Evaluation Metrics

Versus Built-In Game AIs. Directly competing with the built-in AIs of the games provides a straightforward way of measuring policy performance. Typically, fighting games offer a hierarchical structure of levels, enabling players to adjust the difficulty setting (for example, Street Fighter features eight distinct levels). This structure allows for the empirical evaluation of the policy against the game's scripted AI at varying levels of challenge. It is important to acknowledge, however, that the limitations associated with hard-coded adversaries restrict the extent to which this metric can accurately reflect the policy's real capability. For brevity, we shall refer to such agents as CPU.

Elo Ratings. The skills of agents can be ranked through the FIDE rating system (Elo & Sloan, 1978), which is an incremental learning system that increases the Elo of winners and decreases the Elo of losers. The larger the difference in Elo between players A and B, the higher the probability that the player with the higher Elo, A, beats the player with the lower Elo, B. The Elo score calculation takes the following procedures:

First, the probability of player A winning is estimated with,

$$p_A := p(\pi_A \succ \pi_B) = (1.0 + 10^{\frac{\text{Elo}_B - \text{Elo}_A}{400}})^{-1}.$$

Then the Elo rating for player A as Elo_A will be updated with following formula:

$$\text{Elo}_A = \text{Elo}_A + k \cdot (\mathbb{1}[\text{winner} = A] - p_A),$$

where k is a constant of update rate. The update is symmetric for player B, as well as any other player in the ranking system.

[1] https://sites.google.com/view/fightladder/home

Versus AI Exploiters. As discussed in Section 3, exploitability (as Definition 3.3) measures the distance of a policy to the Nash equilibrium of the game. Specifically, the exploitability of a policy μ is measured by the win rate of its BR policy $\nu^\dagger(\mu)$ against μ, since $V^{\mu^\star,\nu^\star}(s_1) = 0$ for symmetric zero-sum game and $V_2^{\mu,\dagger}(s_1) = 1 \cdot p(\nu \succ \mu) + 0 \cdot p(\nu \preceq \mu) = p(\nu \succ \mu)$ for sparse reward setting. In practice, we can use any single-agent deep RL algorithm as an exploiter to approximately learn the BR policy $\nu^\dagger(\mu)$. For fair comparisons, we should use one consistent exploiter (same RL algorithm with same configurations) to evaluate the exploitability of different baselines.

Versus Human Players. While Definition 3.3 is a general metric to measure exploitability, it may be limited to the capability of deep RL algorithms in usage. Therefore, we also provide an interface for human players such that they can play with any learned model with convenience. This feature will show the strengths and weaknesses of agents directly and visibly, and motivate developers to improve their algorithms to be more non-exploitable in general. Given the remarkable success of modern RL algorithms outperforming expert human players in various video games (Mnih et al., 2013; Vinyals et al., 2019; Berner et al., 2019), we believe that FightLadder will emerge as a promising platform for the broad competitive MARL community and researchers will eventually build AI agents that could beat world champions in a much richer set of strategic games with significantly less engineering efforts.

6 FightLadder-Baselines

For the convenience of the community to evaluate existing methods and new algorithms on FightLadder platform, we open-source the implementation of several state-of-the-art (SOTA) competitive MARL algorithms, including independent learning (de Witt et al., 2020), two-timescale learning (Daskalakis et al., 2020), fictitious self-play (Heinrich et al., 2015), policy-space response oracle (Lanctot et al., 2017) and league training (Vinyals et al., 2019). Our codebase supports decentralized learning across multiple GPUs, and it is built upon Stable-Baselines3 (Raffin et al., 2021) so that users can leverage off-the-shelf implementations of RL algorithms. We choose proximal policy optimization (PPO) (Schulman et al., 2017) as the backbone policy optimization algorithm in our experiments. More details of baseline algorithms refer to Appendix B.

7 Results

In this section, we provide benchmark results on a selected game in FightLadder–the Street Fighter. We aim to answer the following questions through our benchmark: **(a)** Can existing RL algorithms solve the full video game in the single-player scenario? **(b)** How does the performance of state-of-the-art baseline algorithms in the two-player competitive setting compare? and **(c)** Does multi-agent training help to improve the non-exploitability?

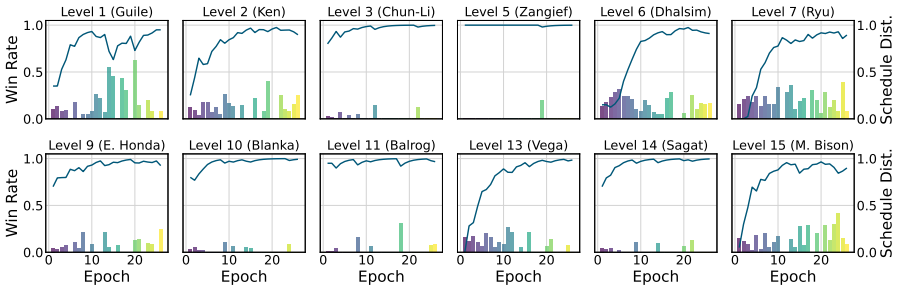

Figure 4: The win rate curves and scheduling distribution bar plot in *sf_ryu_full_game* via the proposed PPO with curriculum learning. Opponents of different characters are marked with different levels. Levels 4, 8, and 12 are omitted as they are bonus levels without fighting.

7.1 Single-Player Full Video Game

To answer question **(a)**, we evaluate PPO's performance in the scenario *sf_ryu_full_game* with human action space as a feasibility check. As mentioned in Section 4, this scenario requires the agent to learn a generalizable policy to compete against all different characters with increasing difficulty levels. *Curriculum learning* is applied to train the policy from easy to hard cases. Furthermore, to improve learning efficiency we develop a curriculum scheduler for opponent sampling to match with the learner after each epoch. More specifically, for the current learner L with policy π_L, we sample its opponent C from the entire character set $\mathcal{C}$, with the following inverse-weight scheduling distribution:

$$C \sim \Delta(\mathcal{C}) \propto 1 - p(\pi_L \succ \pi_C),$$

where $p(\pi_L \succ \pi_C)$ is the win rate of the learner against the opponent and $\Delta(\cdot)$ is the simplex. Intuitively, such a curriculum will encourage the agent to focus on the hardest opponents, similarly to prioritized experience play (Schaul et al., 2015). We defer other implementation details to Appendix C.

Figure 4 shows the performance of our proposed method during training. With 20 epochs of training (each epoch involves 10M training steps competing with opponents sampled from the curriculum scheduler in parallel), the agent is capable of defeating characters at each level with a win rate close to 1. In addition to beating each character with a high probability, the trained policy can complete the full video game with over 0.6 win rate, outperforming human players with hours of playing experience. This result shows that existing RL algorithms can already learn a well-behaved policy to solve the full single-player video game, which provides a good starting point for exploring the multi-agent setting.

As an additional experiment, we also test the inclusion of hard-coded special move lists in this setting with exactly the same algorithm implementation. Although it could be easier for the agent to learn more offensive policies, significant improvement in the

overall win rate is not observed. It indicates that the agents without encoded special moves can also effectively learn policies against CPUs. Constantly playing special moves will lead to a vulnerable situation for the agent, whereas the defensive strategy also matters greatly in the game. Moreover, given that an experienced human player can perform special moves easily (by executing the action sequences almost instantly), we do not think that hard-coded special move lists will become the advantage of trained agents over human players.

7.2 Performance of Two-Player Baseline Algorithms

To answer question **(b)**, we evaluate five SOTA algorithms mentioned in Section 6: independent PPO (IPPO), two-timescale IPPO (2Timescale), fictitious self-play (FSP), policy-space response oracles (PSRO), and league training (League) in the scenario *sf_ryu_vs_ryu*. IPPO and 2Timescale can be categorized into the independent learning paradigm, while FSP, PSRO, and League can be categorized into the population-based learning paradigm. For each algorithm, we initialize the population of agents with a pretrained policy in *sf_ryu_vs_ryu(cpu)* against the most difficult CPU[2]. We use the transformed actions $\mathcal{A}_{\text{trans}}$ with hard-coded special moves to unleash the full potential for agents. As a fair comparison, we use the same codebase (FightLadder-Baselines) and fix the hyperparameters of the backbone PPO algorithm. We train IPPO and 2Timescale for approximately 50M steps until the Elos saturate across all three seeds, FSP and PSRO for approximately 250M steps, and League for approximately 700M steps due to a larger population. A slice of the league during the league training process is visualized in Figure 5 Please refer to Appendix C for more implementation details.

For each algorithm, we report the training Elos of agents in the population during the course of training, respectively. The results are shown in Appendix D, which reveal that all baseline algorithms are improving their policies at the onset of training. Subsequently, IPPO and 2Timescale gradually converge and oscillate around the peak Elos, where FSP, PSRO, and League continue to increase their scores. This suggests that IPPO and 2Timescale may suffer from optimization issues during training and population-based methods may be more suitable for policy learning in fighting games.

To compare different baseline algorithms, we select the top ten agents (five on each left or right side) from each algorithm to form a new population, and compute the test Elos for this group of agents and CPU policies. We report the highest Elos for each algorithm in Table 2 and the distribution of these agents' Elos in Figure 6, where we find that League and PSRO significantly outperform other baselines, and population-based methods deliver better results than independent learning counterparts, which is aligned with our previous observation inspecting Elos of baselines individually. On the other hand, we notice that CPU policies may defeat most of the agents in this group except for a few best-performing agents, suggesting that it is still very challenging for existing SOTA algorithms to reach an advanced or superhuman level of performance in these fighting games. We also noticed that two sides of agents reveal asymmetric strengths in terms of Elos in both individual evaluation for each algorithm (Appendix D

[2]We do not pre-train in *sf_ryu_full_game* as *sf_ryu_vs_ryu* does not require skills to compete with other characters rather than Ryu.

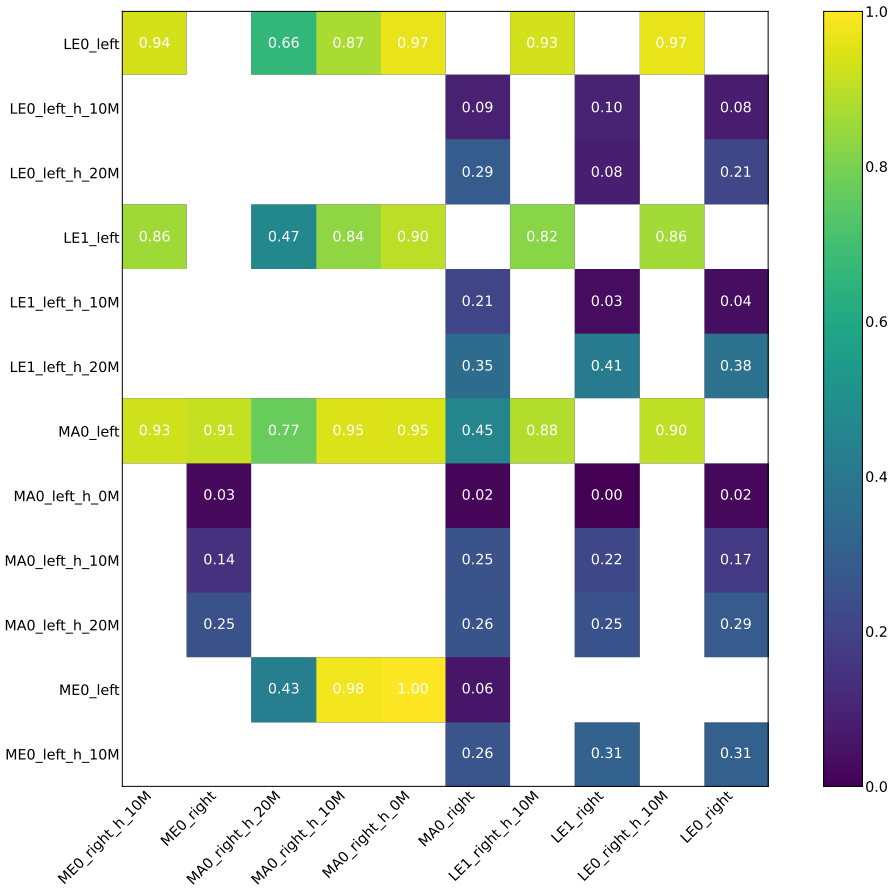

Figure 5: The payoff matrix for each pair of agents at a certain stage of League training. For league training, there is one main agent (MA), two league exploiters (LE0, LE1), and one main exploiter (ME) for each side (left or right). The name of each row indicates the agent information as `Character_Side_Checkpoint`. `Checkpoint=h_xM` represents a historical version of agent saved at `x` million steps. The value indicates the win rate of the left (row) player against the right (column) player. For instance, `ME0_right` wins all `MA0_left_h_xM` with high probability, indicating that main exploiters in the league can fully exploit previous main agents. Also the high win rate of `MA0_left` against all right agents (except `MA0_right`) shows that the main agent at current steps outperforms other agents in the league.

Figure 10-14) and overall evaluations across algorithms (Table 2). Such an imbalance may result from various factors, for instance, optimizing instability, variance from the population or Elos computation, etc, and can be an interesting research question for future work.

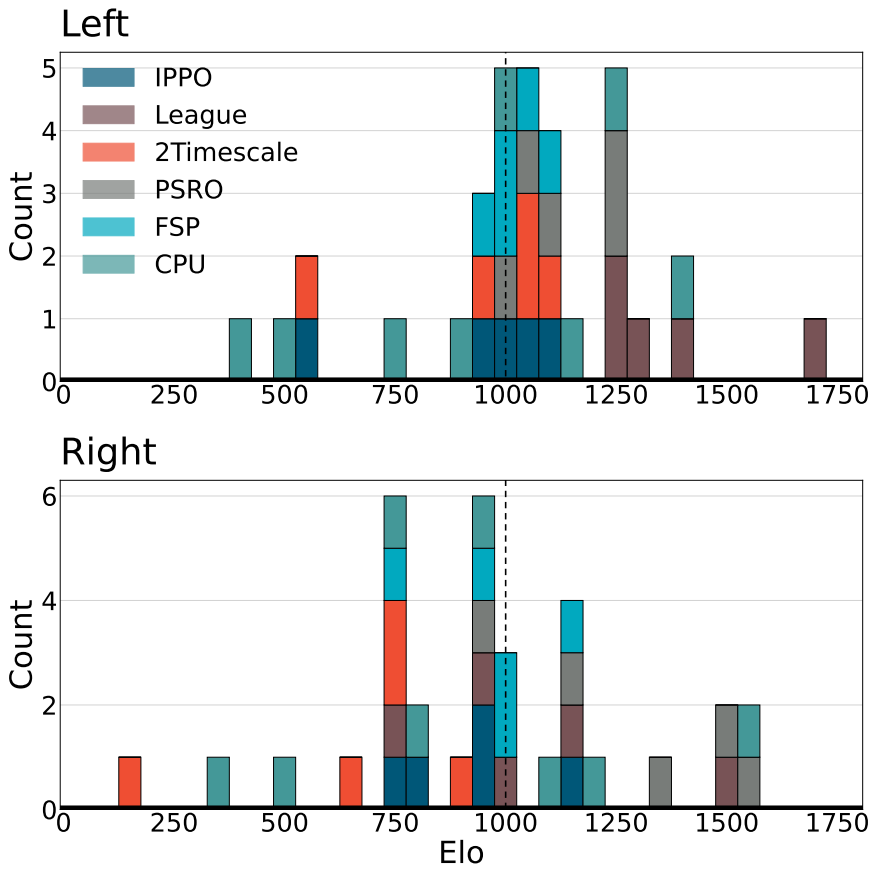

Figure 6: The distribution of Elo ratings for top ten agents from each baseline.

7.3 Non-Exploitability of Trained Agents

To answer question **(c)**, we measure the non-exploitability of baseline algorithms according to the evaluation approaches proposed in Section 5. More specifically, we choose models with the highest Elos from each two-player baseline algorithm respectively, and compare their exploitability with the single-player pretrained model used for initializing the population-based methods in Section 7.2.

The practical exploitability is calculated by setting the trained policy fixed on one side, and deploying a PPO agent on the other side as an exploiter. The PPO exploiter will be trained until convergence, and the success rate of the exploiter is the estimated exploitability of the original policy, according to Definition 3.3.

Single-Agent RL Exploiters. We use PPO as the algorithm for training exploiters, given its decent performance in both single-player and two-player scenarios shown in previous experiments. Table 3 shows the exploitability of comparing methods evaluated

Table 2: Comparison of training steps and the best Elo ratings among baselines, with CPU's Elos as references.

Method	**Training Steps** (Left/Right)	**Elo** (Left/Right)
IPPO	46M / 46M	1082 / 1164
League	647M / 630M	**1682** / **1503**
2Timescale	51M / 46M	1080 / 919
PSRO	176M / 161M	1262 / **1517**
FSP	262M / 244M	1079 / 1150
CPU	N/A	1395 / 1541

Table 3: Comparison of methods' exploitability. A lower number indicates the evaluated policy is more robust to exploitation.

Method	**Exploitability** (Left/Right)
IPPO	0.96 ± 0.03 / 0.91 ± 0.03
League	$\mathbf{0.94 \pm 0.05}$ / 0.94 ± 0.00
2Timescale	0.96 ± 0.02 / 0.90 ± 0.05
PSRO	0.97 ± 0.02 / $\mathbf{0.88 \pm 0.05}$
FSP	1.00 ± 0.00 / 0.95 ± 0.01
PPO	0.99 ± 0.02 / 0.99 ± 0.01

across three seeds, from which we observe that the single-player pretrained policy via PPO is easier to exploit and suffers from higher exploitability than almost all selected policies from two-player baselines. Therefore, this result indicates that two-player learning algorithms such as League and PSRO can help to improve the robustness of learned policies. On the other hand, the PPO exploiter eventually learns to beat policies from all baselines (with a win rate greater than 0.5), which means that none of these algorithms can result in the exact Nash equilibrium policies, or even close to it. Therefore, closing this gap is a challenging direction for future research.

Human Players as Exploiters. In addition to exploiting the learned models with RL algorithms, we also attempt to exploit their policies with human effort. During human evaluations, the evaluated models reveal some robustness to human players (e.g., defend when a human player attacks), but some simple strategies (e.g., defensive posture combined with low kicks at proper timing) could still defeat them rather consistently. Visualizations are provided in Appendix E.

Therefore, based on two exploiting experiments, we observe that *existing competitive MARL algorithms are found hard to learn non-exploitable strategies in competitive fighting games like Street Fighter*, thus raising a new challenge for the research community.

8 Conclusion and Limitation

In this paper, we present the FightLadder platform and evaluation benchmarks as a novel testbed for competitive MARL research. The platform supports various video action games including the popular Street Fighter series, with flexible support for new game integration.

We further provide experimental evaluations of present RL and MARL algorithms in both single-player and two-player modes of one specific game Street Fighter. In the single-player setting, we proposed a learning scheme based on curriculum learning. It trains a general RL agent that can consistently beat CPUs across different characters. In the two-player setting, the Elo rating and exploitability test are conducted as part of the proposed evaluation criteria. Our implementation of league training and PSRO provides stronger agents than FSP and IPPO in terms of Elo ratings. However, both single-agent RL and human players are capable of exploiting all agents learned by current widely adopted algorithms.

Our current work is limited to fully competitive two-player games. One important challenge of MARL is its diverse nature, which includes collaborative games, competitive games, two-player games, and multiplayer games, all of which have rather different problem structures and solution concepts. The more general setting, which involves more than two players and both cooperation and competition, is not yet explored and should be an important future direction. Although FightLadder supports multiple fighting games, our current results are mostly conducted on Street Fighter, and we are curious to see more results on other games.

This work motivates further research in developing more efficient and effective self-play algorithms finding non-exploitable strategies. We hope that our platform prompts general interest and more extensive research in competitive MARL and serves as a standard benchmark for developing practically useful self-play training paradigms.

Acknowledgements

This work was supported by Office of Naval Research N00014-22-1-2253, National Science Foundation Grant NSF-IIS-2107304, and National Science Foundation Graduate Research Fellowship Program under Grant No. DGE-2039656.

Impact Statement

This work may advance the field of game AI, thus has potentials to affect the gaming experience for human players. The strong AI agents for popular fighting games may attract people's attention to get involved in these games, or make them feel that the games can be even more challenging for human. Another positive impact is that our study promotes the research for robust systems against adversarial attacks.

References

Andrychowicz, M., Raichuk, A., Stańczyk, P., Orsini, M., Girgin, S., Marinier, R., Hussenot, L., Geist, M., Pietquin, O., Michalski, M., et al. What matters in on-policy reinforcement learning? a large-scale empirical study. *arXiv preprint arXiv:2006.05990*, 2020a.

Andrychowicz, O. M., Baker, B., Chociej, M., Jozefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020b.

Baker, B., Kanitscheider, I., Markov, T., Wu, Y., Powell, G., McGrew, B., and Mordatch, I. Emergent tool use from multi-agent autocurricula. *arXiv preprint arXiv:1909.07528*, 2019.

Bard, N., Foerster, J. N., Chandar, S., Burch, N., Lanctot, M., Song, H. F., Parisotto, E., Dumoulin, V., Moitra, S., Hughes, E., et al. The hanabi challenge: A new frontier for ai research. *Artificial Intelligence*, 280:103216, 2020.

Beattie, C., Köppe, T., Duéñez-Guzmán, E. A., and Leibo, J. Z. Deepmind lab2d. *arXiv preprint arXiv:2011.07027*, 2020.

Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Brohan, A., Brown, N., Carbajal, J., Chebotar, Y., Dabis, J., Finn, C., Gopalakrishnan, K., Hausman, K., Herzog, A., Hsu, J., et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

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

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

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

Daskalakis, C., Foster, D. J., and Golowich, N. Independent policy gradient methods for competitive reinforcement learning. *Advances in neural information processing systems*, 33:5527–5540, 2020.

de Witt, C. S., Gupta, T., Makoviichuk, D., Makoviychuk, V., Torr, P. H., Sun, M., and Whiteson, S. Is independent learning all you need in the starcraft multi-agent challenge? *arXiv preprint arXiv:2011.09533*, 2020.

Ding, Z., Su, D., Liu, Q., and Jin, C. A deep reinforcement learning approach for finding non-exploitable strategies in two-player atari games. *arXiv preprint arXiv:2207.08894*, 2022.

Domahidi, A., Chu, E., and Boyd, S. ECOS: An SOCP solver for embedded systems. In *European Control Conference (ECC)*, pp. 3071–3076, 2013.

Dresher, M., Shapley, L. S., and Tucker, A. W. *Advances in Game Theory.(AM-52), Volume 52*, volume 52. Princeton University Press, 2016.

Elo, A. E. and Sloan, S. The rating of chessplayers: Past and present. *(No Title)*, 1978.

Filar, J. and Vrieze, K. *Competitive Markov decision processes*. Springer Science & Business Media, 2012.

Foerster, J., Farquhar, G., Afouras, T., Nardelli, N., and Whiteson, S. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Go, S.-X., Jiang, Y., and Loke, D. K. A phase-change memristive reinforcement learning for rapidly outperforming champion street-fighter players. *Advanced Intelligent Systems*, 5(11):2300335, 2023.

Heinrich, J., Lanctot, M., and Silver, D. Fictitious self-play in extensive-form games. In Bach, F. and Blei, D. (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 805–813, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings.mlr.press/v37/heinrich15.html.

Hu, J. and Wellman, M. P. Nash q-learning for general-sum stochastic games. *Journal of machine learning research*, 4(Nov):1039–1069, 2003.

Hu, J., Jiang, S., Harding, S. A., Wu, H., and Liao, S.-w. Rethinking the implementation tricks and monotonicity constraint in cooperative multi-agent reinforcement learning. *arXiv preprint arXiv:2102.03479*, 2021.

Hu, S., Zhong, Y., Gao, M., Wang, W., Dong, H., Liang, X., Li, Z., Chang, X., and Yang, Y. Marllib: A scalable and efficient multi-agent reinforcement learning library. *Journal of Machine Learning Research*, 24(315):1–23, 2023.

Khan, I., Van Nguyen, T., Dai, X., and Thawonmas, R. Darefightingice competition: A fighting game sound design and ai competition. In *2022 IEEE Conference on Games (CoG)*, pp. 478–485. IEEE, 2022.

Kurach, K., Raichuk, A., Stańczyk, P., Zając, M., Bachem, O., Espeholt, L., Riquelme, C., Vincent, D., Michalski, M., Bousquet, O., 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.

Lanctot, M., Zambaldi, V., Gruslys, A., Lazaridou, A., Tuyls, K., Pérolat, J., Silver, D., and Graepel, T. A unified game-theoretic approach to multiagent reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

Li, J., Koyamada, S., Ye, Q., Liu, G., Wang, C., Yang, R., Zhao, L., Qin, T., Liu, T.-Y., and Hon, H.-W. Suphx: Mastering mahjong with deep reinforcement learning. *arXiv preprint arXiv:2003.13590*, 2020.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Lowe, R., Wu, Y. I., Tamar, A., Harb, J., Pieter Abbeel, O., and Mordatch, I. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30, 2017.

McMahan, H. B., Gordon, G. J., and Blum, A. 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.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Mohanty, S., Nygren, E., Laurent, F., Schneider, M., Scheller, C., Bhattacharya, N., Watson, J., Egli, A., Eichenberger, C., Baumberger, C., et al. Flatland-rl: Multi-agent reinforcement learning on trains. *arXiv preprint arXiv:2012.05893*, 2020.

Moravvcík, M., Schmid, M., Burch, N., Lisỳ, V., Morrill, D., Bard, N., Davis, T., Waugh, K., Johanson, M., and Bowling, M. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 356(6337):508–513, 2017.

Mordatch, I. and Abbeel, P. Emergence of grounded compositional language in multi-agent populations. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Murphy, D. Hacking public memory: Understanding the multiple arcade machine emulator. *Games and Culture*, 8(1):43–53, 2013.

Nichol, A., Pfau, V., Hesse, C., Klimov, O., and Schulman, J. Gotta learn fast: A new benchmark for generalization in rl. *arXiv preprint arXiv:1804.03720*, 2018.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

Palmas, A. Diambra arena: a new reinforcement learning platform for research and experimentation. *arXiv preprint arXiv:2210.10595*, 2022.

Pan, X., Liu, M., Zhong, F., Yang, Y., Zhu, S.-C., and Wang, Y. Mate: Benchmarking multi-agent reinforcement learning in distributed target coverage control. *Advances in Neural Information Processing Systems*, 35:27862–27879, 2022.

Papoudakis, G., Christianos, F., Schäfer, L., and Albrecht, S. V. Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks. *arXiv preprint arXiv:2006.07869*, 2020.

Peng, B., Rashid, T., Schroeder de Witt, C., Kamienny, P.-A., Torr, P., Böhmer, W., and Whiteson, S. Facmac: Factored multi-agent centralised policy gradients. *Advances in Neural Information Processing Systems*, 34:12208–12221, 2021.

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

Rashid, T., Samvelyan, M., De Witt, C. S., Farquhar, G., Foerster, J., and Whiteson, S. Monotonic value function factorisation for deep multi-agent reinforcement learning. *The Journal of Machine Learning Research*, 21(1):7234–7284, 2020.

Resnick, C., Eldridge, W., Ha, D., Britz, D., Foerster, J., Togelius, J., Cho, K., and Bruna, J. Pommerman: A multi-agent playground. *arXiv preprint arXiv:1809.07124*, 2018.

Samvelyan, M., Rashid, T., De Witt, C. S., Farquhar, G., Nardelli, N., Rudner, T. G., Hung, C.-M., Torr, P. H., Foerster, J., and Whiteson, S. The starcraft multi-agent challenge. *arXiv preprint arXiv:1902.04043*, 2019.

Sarkar, B., Talati, A., Shih, A., and Sadigh, D. Pantheonrl: A marl library for dynamic training interactions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 13221–13223, 2022.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

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

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

Song, Y., Wojcicki, A., Lukasiewicz, T., Wang, J., Aryan, A., Xu, Z., Xu, M., Ding, Z., and Wu, L. Arena: A general evaluation platform and building toolkit for multi-agent intelligence. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 7253–7260, 2020.

Suarez, J., Du, Y., Zhu, C., Mordatch, I., and Isola, P. The neural mmo platform for massively multiagent research. *arXiv preprint arXiv:2110.07594*, 2021.

Sukhbaatar, S., Fergus, R., et al. Learning multiagent communication with backpropagation. *Advances in neural information processing systems*, 29, 2016.

Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W. M., Zambaldi, V., Jaderberg, M., Lanctot, M., Sonnerat, N., Leibo, J. Z., Tuyls, K., et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*, 2017.

Tan, M. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, pp. 330–337, 1993.

Terry, J., Black, B., Grammel, N., Jayakumar, M., Hari, A., Sullivan, R., Santos, L. S., Dieffendahl, C., Horsch, C., Perez-Vicente, R., et al. Pettingzoo: Gym for multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 34: 15032–15043, 2021.

Tesauro, G. et al. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

Yao, Z. and Ding, Z. Learning distributed and fair policies for network load balancing as markov potential game. *Advances in Neural Information Processing Systems*, 35: 28815–28828, 2022.

Yu, C., Velu, A., Vinitsky, E., Gao, J., Wang, Y., Bayen, A., and Wu, Y. The surprising effectiveness of ppo in cooperative multi-agent games. *Advances in Neural Information Processing Systems*, 35:24611–24624, 2022.

Zhang, H., Feng, S., Liu, C., Ding, Y., Zhu, Y., Zhou, Z., Zhang, W., Yu, Y., Jin, H., and Li, Z. Cityflow: A multi-agent reinforcement learning environment for large scale city traffic scenario. In *The world wide web conference*, pp. 3620–3624, 2019.

Zheng, L., Yang, J., Cai, H., Zhou, M., Zhang, W., Wang, J., and Yu, Y. Magent: A many-agent reinforcement learning platform for artificial collective intelligence. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Zhou, M., Luo, J., Villella, J., Yang, Y., Rusu, D., Miao, J., Zhang, W., Alban, M., Fadakar, I., Chen, Z., et al. Smarts: Scalable multi-agent reinforcement learning training school for autonomous driving. *arXiv preprint arXiv:2010.09776*, 2020.

Zhou, M., Wan, Z., Wang, H., Wen, M., Wu, R., Wen, Y., Yang, Y., Yu, Y., Wang, J., and Zhang, W. Malib: A parallel framework for population-based multi-agent reinforcement learning. *Journal of Machine Learning Research*, 24(150):1–12, 2023.

A Details of FightLadder

A.1 Dense Reward

The shaped dense reward for the i-th agent at step t is defined as follows:

$$r_{i,t} = \alpha \left[\lambda(\text{HP}_{-i,t-1} - \text{HP}_{-i,t}) - (\text{HP}_{i,t-1} - \text{HP}_{i,t}) + r_{i,\text{bonus}} \right], \tag{1}$$

where α is a scaling factor, $\text{HP}_{i,t}$ denotes agent i's hit-point at step t and λ control the aggressiveness of learned agents, and $-i$ denotes the opponent agent. At the end of the game, the agent i will receive a bonus reward $r_{i,\text{bonus}}$, which is positively correlated to HP_i if it wins and negatively correlated to HP_{-i} if it loses. By default, we choose $\lambda = 3$ in SF2, FF2, and MK, and $\lambda = 1$ in SF3 and KOF97, for the consideration of practical performances.

A.2 Game Settings

Table 4 illustrates the observation, action, and rewards as well as other elements in the environment for all supported games — Street Fighter II (SF2), Fatal Fury 2 (FF2), Mortal Kombat (MK), Street Fighter III (SF3), and The King of Fighters '97 (KOF97).

Table 4: Specification of supported games in FightLadder.

	SF2	**FF2**	**MK**	**SF3**	**KOF97**
Observation (Pixels)	100×128×3	112×128×3	112×160×3	112×192×3	112×192×3
Human Action Supported	Yes	Yes	Yes	Yes	Yes
Transformed Action Supported	Yes	Yes	Yes	No	No
Shaped Dense Reward	Yes	Yes	Yes	Yes	Yes
Default Frames Per Step	8	8	8	3	3
Default Frames Stacked[3]	12	12	12	9	9
Additional Available Info	HPs, Countdown, Scoreboard, Positions	HPs, Countdown	HPs, Countdown, Scoreboard	HPs	HPs, Countdown, Positions, Power Status

A.3 Comparison of MARL Game Platforms

Table 5 compares our FightLadder with several popular MARL game platforms mostly focusing on competitive settings, in terms of observation space, action space, whether baseline methods are included and the number of agents in games. For the observation space, 'Continuous' indicates a vector-form latent state information of the game with continuous numerical values, and 'Image' indicates visual RGB information as observations. PommerMan (Resnick et al., 2018) uses grid environments therefore its observation only has discrete values. For the action space, most of the games only involves discrete action values except for Arena (Song et al., 2020). For the number of agents in these platforms, MPE provide diverse competitive settings like 1v1, 1vN, 1v1v1 and so on. MAgent includes 1 million agents competing againts each other, and for Neural MMO (Suarez et al., 2021) the number of agents is 256 or 1024. The team mode in our FightLadder and Arena supports the competitive settings of two teams, where each team includes multiple characters to be controlled by one team policy or separate agent policies.

[3]We uniformly sample the stacked frames as observations to improve the computational efficiency.

Table 5: Comparison of popular MARL game platforms.

Env	Observation Space	Action Space	Baselines	# Agents
MPE (Mordatch & Abbeel, 2018)	Continuous	Discrete	Yes	1v1, 1vN and 1v1v1...
MAgent (Zheng et al., 2018)	Continuous+Image	Discrete	Yes	1 million
Arena (Song et al., 2020)	Continuous+Image	Continuous/Discrete	Yes	1v1, NvN and team mode
Neural MMO (Suarez et al., 2021)	Continuous	Discrete	Yes	256 and 1024
PettingZoo Atari (Terry et al., 2021)	Continuous+Image	Discrete	No	1v1
PommerMan (Resnick et al., 2018)	Discrete	Discrete	No	2v2
FightLadder (Ours)	Continuous+Image	Discrete	Yes	1v1 and team mode

B Baseline Algorithms of FightLadder-Baselines

Independent Learning (IPPO). Independent learning is a straightforward extension of single-agent RL into MARL. It decomposes the joint optimization into individual ones for each agent while regarding all other agents as part of the environment. It can be implemented easily by simultaneously running single-agent RL algorithms for each player. Theoretically, this independent learning paradigm suffers from suboptimality (Tan, 1993; Foerster et al., 2018), because the environment becomes non-stationary while other agents are updating their policies. However, recent work (de Witt et al., 2020; Yu et al., 2022) finds that with modest hyperparameter tuning, IPPO can serve as a strong baseline compared to other state-of-the-art algorithms in some cooperative MARL tasks.

Two-timescale Learning (2Timescale). Two-timescale learning follows the independent learning paradigm, but requires two players to update gradients according to the two-timescale rule, i.e., one player uses a much smaller step size than the other one. As a result of this modification, two-timescale learning enjoys some nice theoretical properties — it is proven that under some mild assumptions, independent policy gradient algorithms satisfying two-timescale converge to a Nash equilibrium in two-player zero-sum stochastic games (Daskalakis et al., 2020).

Population-Based Methods. The independent learning framework is only training agents against the current version of their opponents, which may fail or converge slowly due to the lack of diversity (Dresher et al., 2016). Population-based methods are proposed to increase policy diversity by maintaining a pool of policies in previous iterations, and using them as a curriculum to update the current policy. More specifically, for t-th update, the agent μ^t plays with previous versions of its opponent $\tilde{\nu}$ sampled from the meta-strategy ρ_ν, which is a distribution over $\nu^0, \nu^1, \ldots, \nu^{t-1}$. Algorithm 1 presents the pseudo-code for general population-based methods. With different choices of sampling distribution, we can recover several state-of-the-art baselines:

- **Fictitious Self-Play (FSP),** where ρ_ν is the uniform distribution (Heinrich et al., 2015): Uniform$(\nu^0, \nu^1, \ldots, \nu^{t-1})$.
- **Policy-Space Response Oracles (PSRO),** where $(\tilde{\mu}, \tilde{\nu})$ are sampled from the meta-strategy (ρ_μ, ρ_ν) by solving Nash equilibrium of the payoff matrix game between $\mu^0, \mu^1, \ldots, \mu^{t-1}$ and $\nu^0, \nu^1, \ldots, \nu^{t-1}$ (Lanctot et al., 2017).

- **League Training (League),** where three types of agents — main agents, league exploiters, and main exploiters, are introduced into the population. Main agents train against themselves as well as all previous versions of agents in the population; league exploiters train against all previous agents; and main exploiters optimize the best response of main agents. Each type of agent adopts a different sampling distribution which is a mixture of self-play and prioritized fictitious self-play. We refer readers to (Vinyals et al., 2019) for more implementation details.

Algorithm 1 Population-Based Methods for MGs

Initialize policies $\mu^0 = \{\mu_h\}, \nu^0 = \{\nu_h\}, h \in [H]$
Initialize policy sets: $\mu = \{\mu^0\}, \nu = \{\nu^0\}$
Initialize meta-strategies: $\rho_\mu = [1.], \rho_\nu = [1.]$
for $t = 1, \ldots, T$ **do**
  if $t\%2 == 0$ **then**
    $\nu^t = \text{BEST\_RESPONSE}(\rho_\mu, \mu)$
    $\nu = \nu \bigcup \{\nu^t\}$
    Update ρ_ν according to specific algorithms
  else
    $\mu^t = \text{BEST\_RESPONSE}(\rho_\nu, \nu)$
    $\mu = \mu \bigcup \{\mu^t\}$
    Update ρ_μ according to specific algorithms
  end if
end for
Return $\mu, \rho_\mu, \nu, \rho_\nu$

C Experiment Details

C.1 Hyperparameters (Table 6 and 7)

C.2 Training Details

Figure 7, 8, and 9 report the payoff matrix of policies within the population for FSP, PSRO, and League, respectively, with the value representing the win rate of the left player against the right player. We trained all our agents on one server with 192 CPUs and 8 A6000 GPUs.

Hyperparameters	Value
feature extractor	CNN (Mnih et al., 2015)
rollout steps for each environment	512
batch size	1024
epochs per update	4
γ	0.94
GAE λ	0.95
learning rate	linear schedule from 2.5e-4 to 2.5e-6
clipping range	linear schedule from 0.15 to 0.025
advantage normalization	True
entropy coefficient	0.0
gradient clipping	0.5
value function coefficient	0.5

Table 6: Training hyperparameters for PPO, which is the backbone for both single-player and two-player algorithms in the experiment.

FSP		PSRO		League	
# envs per learner	24	# envs per learner	24	# envs per learner	24
steps for BR	10M	steps for BR	10M	steps for BR	10M
total steps	50M	total steps	250M	total steps	700M
# main agent	1	# main agent	1	# main agent	1
		Nash solver	ECOS (Domahidi et al., 2013)	# main exploiter	1
				# league exploiter	2

Table 7: Training hyperparameters for FSP, PSRO, and League. We omit the details of League's opponent scheduling here as it strictly follows the pseudocode provided in (Vinyals et al., 2019).

D Individual Elo Results

D.1 IPPO (Figure 10)

D.2 2Timescale (Figure 11)

D.3 FSP (Figure 12)

D.4 PSRO (Figure 13)

D.5 League (Figure 14)

E Visualization of Human Exploiters

Figure 15 visualizes how human players can exploit learned models with a simple strategy. Full videos are provided in the supplementary material.

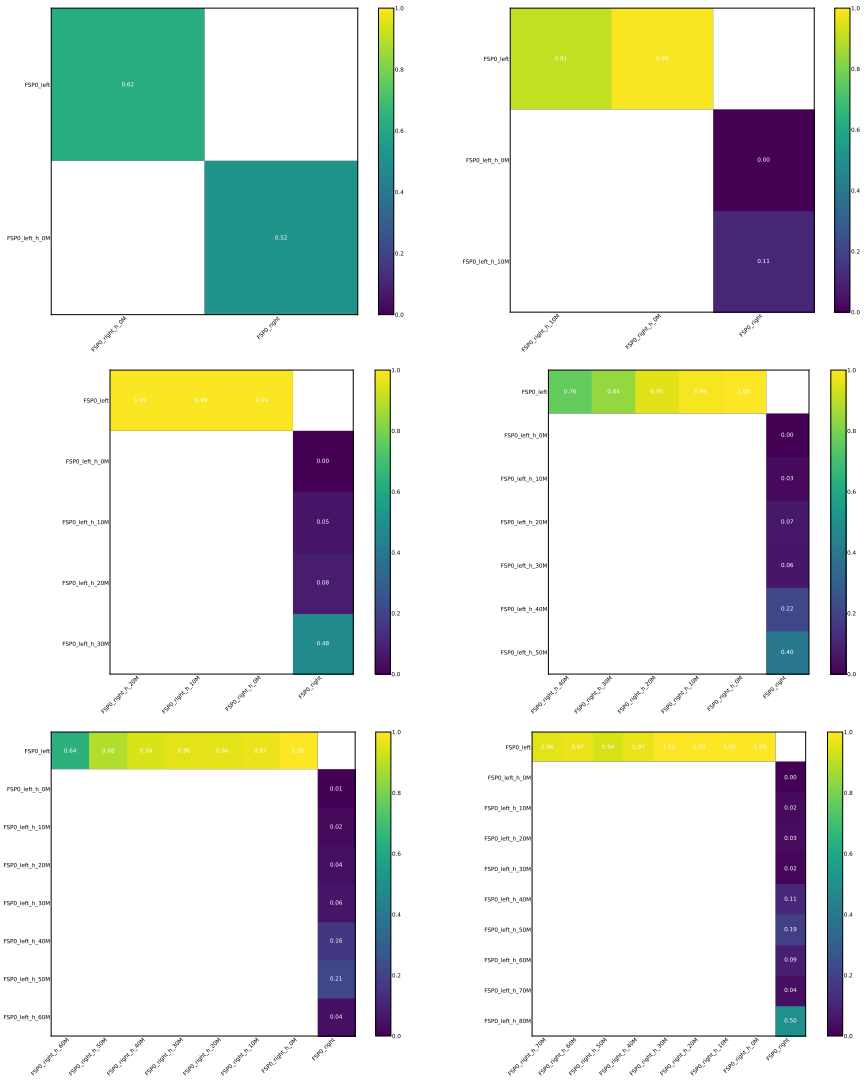

Figure 7: FSP details (training order from top left to bottom right): For FSP, there is one agent for each side (left or right). The name of each row indicates the agent information as `Character_Side_Checkpoint`. `Checkpoint=h_xM` represents a previous version of agent saved at `x` million steps. The value indicates the win rate of the left (row) player against the right (column) player.

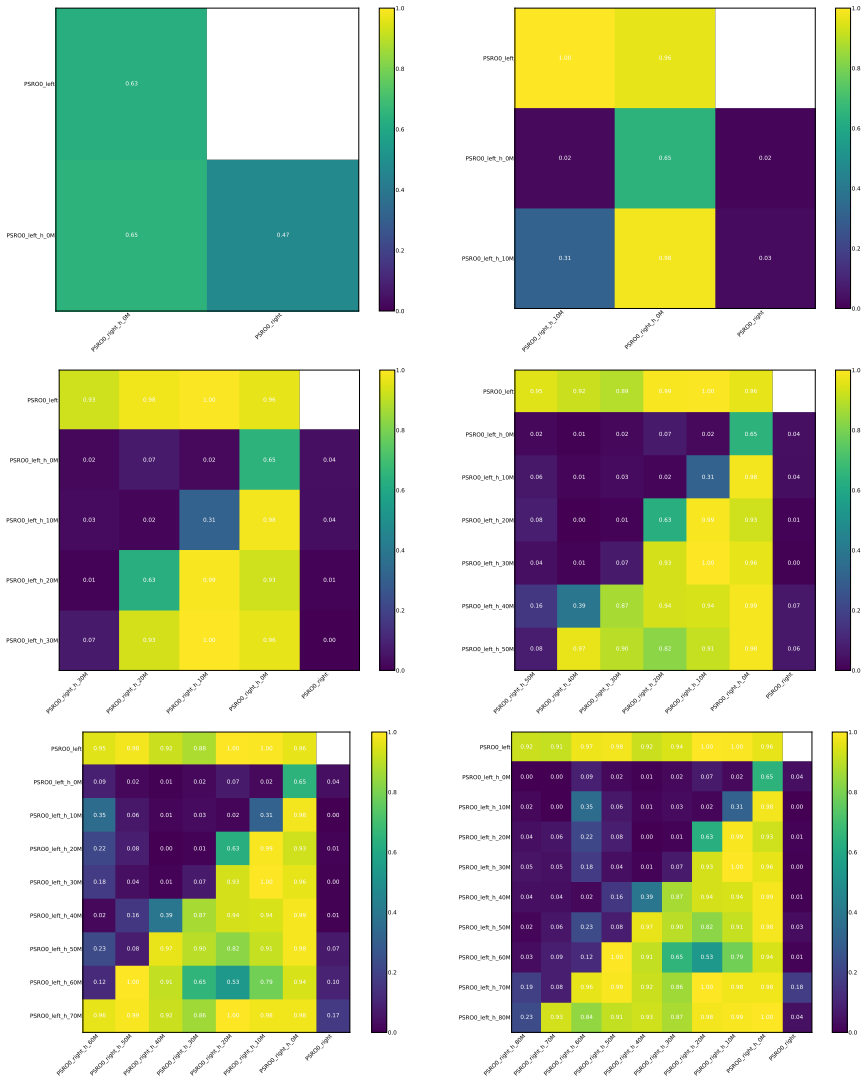

Figure 8: PSRO details (training order from top left to bottom right): For PSRO, there is one agent for each side (left or right). The name of each row indicates the agent information as `Character_Side_Checkpoint`. `Checkpoint=h_xM` represents a previous version of agent saved at `x` million steps. The value indicates the win rate of the left (row) player against the right (column) player.

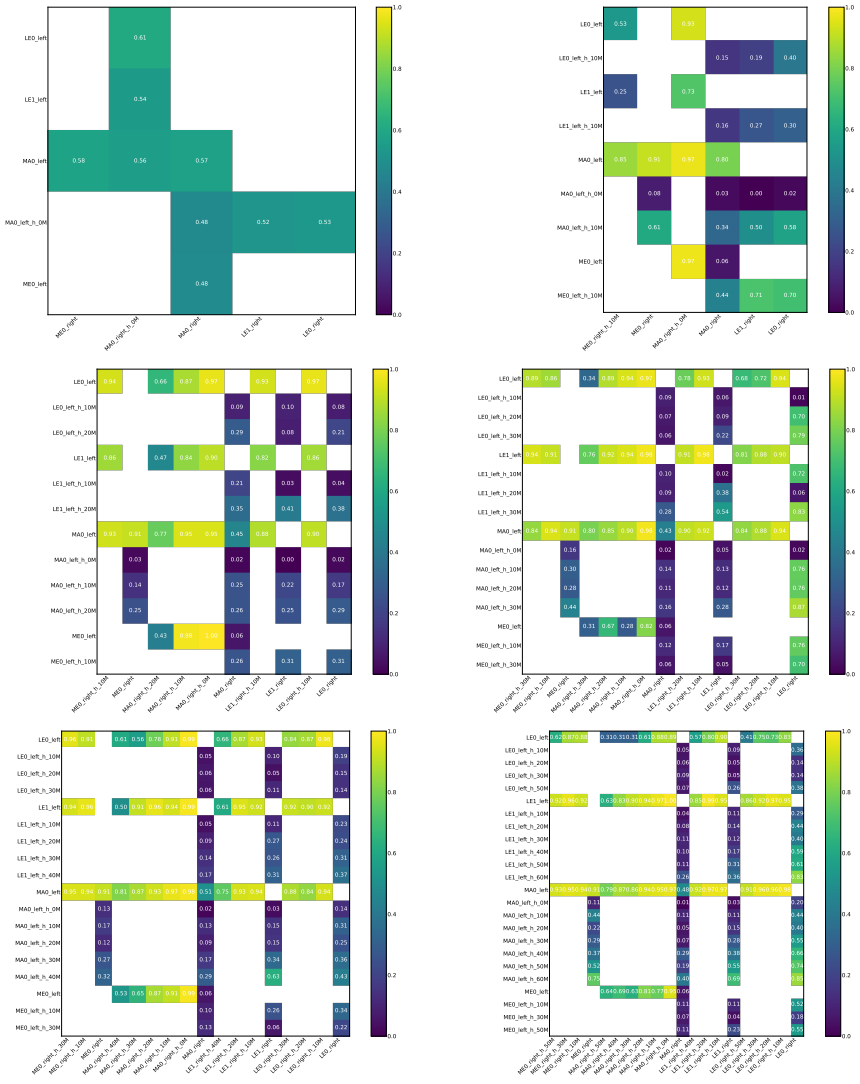

Figure 9: League training details (training order from top left to bottom right): For league training, there is one main agent (MA), two league exploiters (LE0, LE1), and one main exploiter (ME) for each side (left or right). The name of each row indicates the agent information as `Character_Side_Checkpoint`. `Checkpoint=h_xM` represents a previous version of agent saved at `x` million steps. The value indicates the win rate of the left (row) player against the right (column) player.

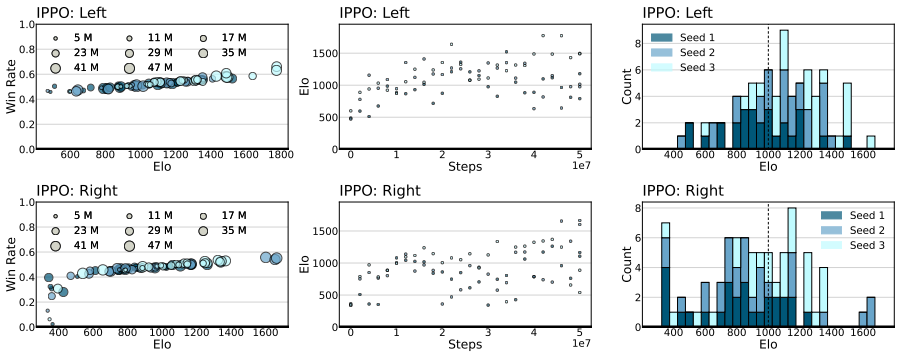

Figure 10: The Elo rating for the population of agents trained with IPPO algorithm. The upper three plots are for left-side player and the bottom three are for the right-side player. The Elo rating is plotted against the winning rate over matched policies (left figures), training steps (middle figures) and the number of policies (right figures).

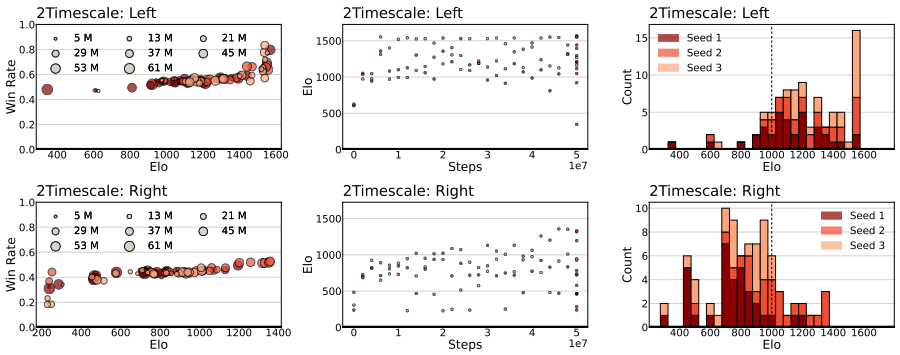

Figure 11: The Elo rating for the population of agents trained with 2Timescale algorithm. The upper three plots are for left-side player and the bottom three are for the right-side player. The Elo rating is plotted against the winning rate over matched policies (left figures), training steps (middle figures) and the number of policies (right figures).

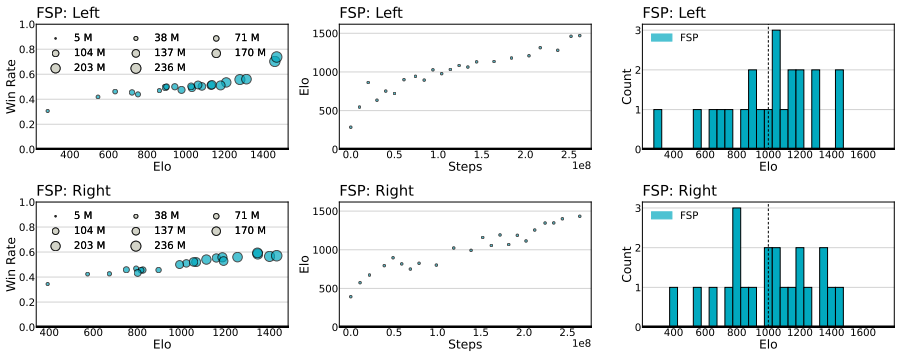

Figure 12: The Elo rating for the population of agents trained with FSP algorithm. The upper three plots are for left-side player and the bottom three are for the right-side player. The Elo rating is plotted against the winning rate over matched policies (left figures), training steps (middle figures) and the number of policies (right figures).

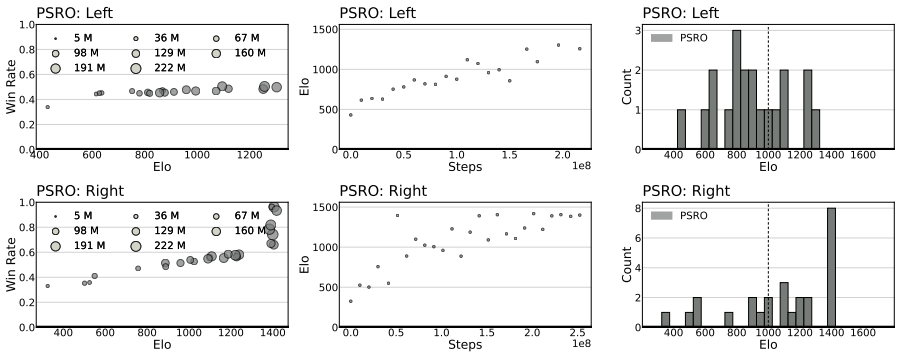

Figure 13: The Elo rating for the population of agents trained with PSRO algorithm. The upper three plots are for left-side player and the bottom three are for the right-side player. The Elo rating is plotted against the winning rate over matched policies (left figures), training steps (middle figures) and the number of policies (right figures).

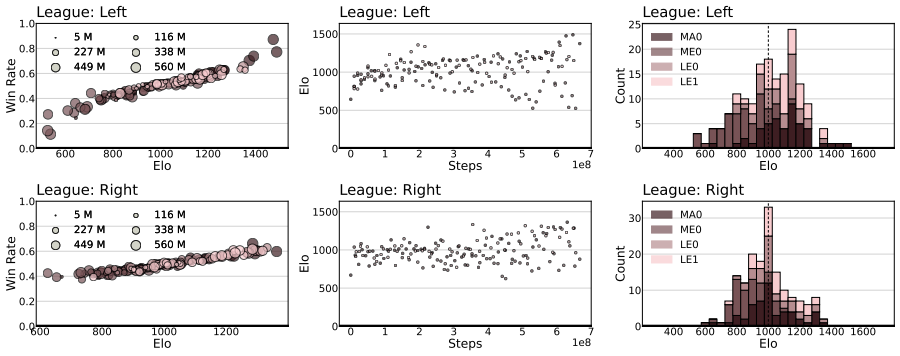

Figure 14: The Elo rating for the population of agents trained with League training. The upper three plots are for left-side player and the bottom three are for the right-side player. The Elo rating is plotted against the winning rate over matched policies (left figures), training steps (middle figures) and the number of policies (right figures).

Figure 15: Demonstration of the exploiting strategy of one human player. The human player (Ryu on the right in white) defends when the AI opponent (Ryu on the left in gray) attacks, and inflicts damage with low kicks.