# FootsiesGym: A Fighting Game Benchmark for Two-Player Zero-Sum Imperfect-Information Games

**Chase McDonald**[1], **Nathan Tsang**[2], **Wesley N. Kerr**[2]
chase@comoresearch.org, {ntsang, wkerr}@riotgames.com

[1]**Como Research**
[2]**Riot Games**

## Abstract

We present **FootsiesGym**, an open-source environment for learning in a non-trivial two-player, zero-sum, imperfect-information game. Built on HiFight's minimalist 2D fighting game *Footsies*, it isolates the cyclic, non-transitive strategic interactions of fighting game neutral play while remaining simple enough for efficient analysis. We provide a vectorized simulator that enables high-throughput training on standard hardware, making the environment accessible and reproducible. We describe the design of the environment, benchmark several reinforcement learning algorithms, and discuss open research directions it enables. The code is available at https://github.com/como-research/FootsiesGym.

## 1 Introduction

Progress in multi-agent and game-theoretic reinforcement learning depends on benchmarks that capture the right structure at a tractable cost. Existing environments tend toward two extremes. On one side are clean game-theoretic benchmarks such as matrix games and poker variants, which expose cyclic, mixed-strategy structure but typically have short horizons, simple dynamics, and limited exploration requirements. On the other are large, real-time environments (e.g., StarCraft II and Dota 2), which are deeply complex with long time horizons, but require extensive resources for training.

Fighting games can occupy a productive middle ground: they are real-time and spatial, yet keep the strategic structure front and center. They rely on fundamentally non-transitive dynamics based on the interaction between two opposing players. The basic strategy in fighting games is often described as a game of *rock-paper-scissors*, as players must learn mixed strategies over the primitives of movement and the different kinds of attacks that counter each other and form strategy cycles. Because no single option dominates, a player who commits to a pure strategy can be easily exploited.

In this work, we provide a benchmark environment to study these interactions in Footsies (HiFight, 2018), an open-source 2D fighter by HiFight, shown in Figure 1. Footsies is built around the *neutral game*: the phase in which neither player has a clear advantage, and both players maneuver, adjust their spacing and timing, and attempt to create an opening to attack. This is precisely where the cyclic, mixed-strategy structure described above emerges.

## 2 Related Work

**Benchmark Environments.** Multi-agent and game-theoretic reinforcement learning research spans a wide range of environment complexity. On one end, there are the small-scale environments that permit exact exploitability calculations (Lanctot et al., 2019; Rudolph et al., 2025), such

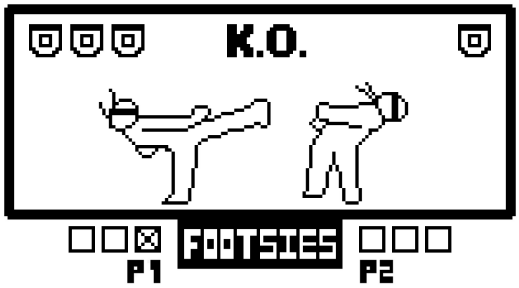



Figure 1: A screenshot of the Footsies game, from HiFight (2018). Two characters fight one another using a combination of movement, quick attacks, and special attacks. A player wins when they K.O. the other.

as Kuhn Poker, Goofspiel, Leduc Poker, Phantom Tic-Tac-Toe, and Dark Hex. While these environments are valuable for exact exploitability calculations and equilibrium analysis, they capture little of the complexity of richer games. On the other end, real-time strategy and MOBA environments such as StarCraft II (Vinyals et al., 2019) and Honor of Kings (Ye et al., 2020) have rich, long-horizon dynamics but demand substantial compute and time to train. Between these extremes, environments such as Slime Volleyball (Tang et al., 2022) and simplified Stratego variants (e.g., McAleer et al., 2020) offer intermediate complexity at moderate cost. FootsiesGym likewise occupies this middle ground: it is real-time, spatial, imperfect-information, and non-transitive, yet remains cheap enough to train on a single workstation in a reasonable amount of time.

**Fighting games as RL environments.** There are several instances of fighting games being used as RL environments. The FightingICE platform (Lu et al., 2013; Khan et al., 2022) has hosted a long-running fighting-game AI competition, while DIAMBRA Arena (Palmas, 2022) and Stable-Retro (Farama Foundation, 2023) expose classic arcade and console fighting games through emulation. These existing platforms typically wrap proprietary games that require user-supplied ROMs or closed-source engines, adding licensing and reproducibility friction. FootsiesGym instead builds on a fully open-source game with a purpose-built, vectorized simulator, prioritizing accessibility and training speed on standard hardware.

## 3 FootsiesGym Environment

Compared to existing fighting game environments, Footsies is intentionally minimalist, allowing for easier analysis and a focus on the genre's non-transitive structure. Typical fighting games feature more complex moves and mechanics, most notably combos: long sequences of chained attacks. These mechanics separate fighting games into two strategic regimes: the cyclic, mixed-strategy interaction that arises during the neutral game, and the largely transitive contest of execution that begins once a player lands an opening and starts a combo. By foregoing the latter, Footsies isolates the neutral game while preserving its cyclic strategic interactions. The key mechanics, restated from the Footsies documentation,[1] are:

- There is no health bar; the round is lost when a player is hit by a special attack.
- Opponent attacks can be blocked up to three times, indicated by the guard bar on the top right and left of the screen. After three blocks, every attack causes a guard break that leaves the player open to a follow-up attack.
- Players can dash forward and backward by pressing the direction button twice in quick succession.
- Special attacks are performed by holding the attack button for 60 frames and then releasing it, from neutral or with a forward/backward direction.

[1] https://github.com/hifight/Footsies

**Actions.** Each agent's per-step action is drawn from a discrete space of size $|\mathcal{A}| = 6$ by default, comprising NONE, BACK, FORWARD, ATTACK, BACK+ATTACK, and FORWARD+ATTACK. Special attacks in Footsies are executed by holding the attack button for a fixed duration and then releasing it. Because discovering when to begin charging and how long to hold the attack button through random exploration can be difficult, FootsiesGym optionally provides a special-charge action mode. When enabled, the action space expands to $|\mathcal{A}| = 9$ by adding three charge actions: BACK-CHARGE, NEUTRAL-CHARGE, and FORWARD-CHARGE. Each charge action toggles the held-attack "charge" state required to fire a special move: the first invocation begins holding the attack button, while the second releases it and executes the corresponding special move (back, neutral, or forward). Appendix E provides further details and illustrates the effect of enabling this option.

**Observations.** Each agent receives an 85-dimensional featurized observation (88-dimensional with the special-charge option enabled) comprising the features required to fully describe the state of both players. Each agent additionally observes a small block of privileged self-features, which includes its own dash availability, special-attack charging progress, and previous action. These are not exposed in the opponent block, mirroring the information asymmetry present in human play. The complete per-feature breakdown is given in Appendix B.

**Rewards.** Rewards are zero-sum between agents ($r^i = -r^{-i}$), where $i$ represents self, and $-i$ represents the opponent. A terminal win/loss reward of $\pm w$ (default $w = 10$) is issued when either agent is defeated. In the event of a timeout (default 4,000 steps), both agents receive 0 reward. Optional dense rewards can be set in the environment configuration to guide learning (see Appendix A).

### 3.1 Action Delay

Action delay is the number of game frames between action selection and in-game execution. It must be an integer multiple of the frame skip, such that an action chosen at step $t$ is executed at step $t + \frac{\text{action delay}}{\text{frame skip}}$. The delay determines which attacks are *reactable* versus which must be *predicted*. At an action delay of 0, the policy has frame-perfect reactions. In this setting, we observe that gameplay collapses into degenerate, purely reactive strategies. As the delay increases, moves become increasingly unreactable, reducing the neutral game to pure prediction over the opponent's intent and, at high values of action delay, producing near-random play. We provide additional results on the effect of manipulating action delay in Appendix D.

### 3.2 Implementation

Footsies is built in the Unity game engine, with all game logic written in C#. For training, we decouple the simulator from Unity's rendering loop and run it headlessly. Vectorization is handled entirely on the C# side: a single headless process steps $K$ independent game instances in parallel and exposes them behind one gRPC port. Python clients instantiate the environment following the PettingZoo (Terry et al., 2021) API. The number of in-process environments $K$ is passed in the config. Standard headless-mode optimizations (e.g., uncapped frame rate, disabled audio, no rendering) are applied automatically, and each vectorized instance is auto-reset on termination.

Figure 2 characterizes two complementary layers of parallelism. Within a single game process ($P$=1), aggregate throughput plateaus at roughly 19,000 environment steps per second. Running $P$ independent game processes in concurrent threads, each with its own gRPC port and $K$ in-process instances, raises this ceiling. At $P$=4 and $K$=128, aggregate throughput reaches roughly 52,500 environment steps per second, roughly 2.7× the single-process peak.

In addition to the headless training server, the same Unity project compiles to a windowed build with full rendering, used for visual inspection of trained agents and local play from a keyboard.

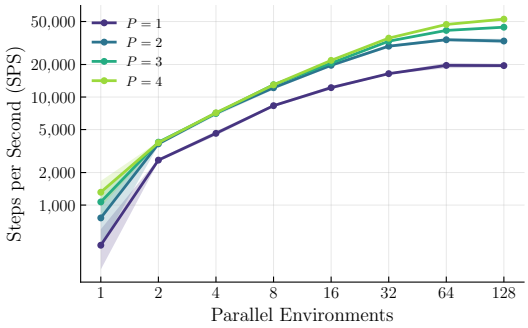
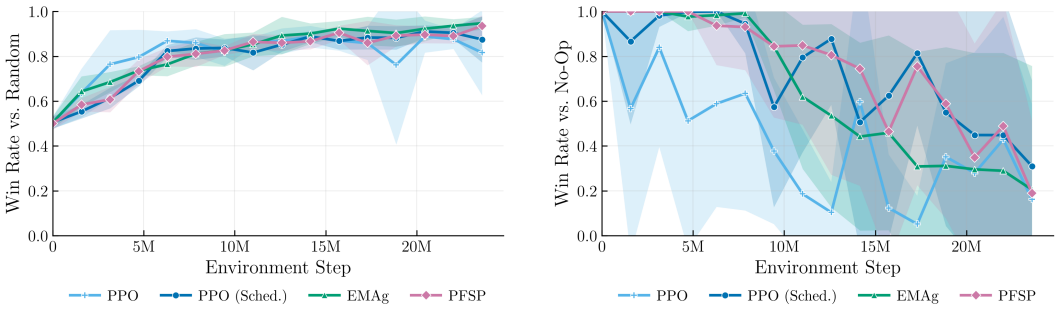



Figure 2: Aggregate environment steps per second as a function of the number of in-process parallel environments $K$, on a 24-core / 48-thread workstation. Each curve corresponds to a different number of independent game-server processes $P$ stepped concurrently from Python threads. Shaded band is the 95% confidence interval.

## 4 Experimental Baselines

We train a set of baseline algorithms to illustrate the learning dynamics in Footsies and highlight opportunities for future research. Importantly, the reported algorithms are only naïvely tuned through several trial-and-error iterations; they are not fully tuned implementations of these algorithms, and we make no definitive claim about their relative quality.

**Algorithms.** We use two variations of Proximal Policy Optimization (PPO; Schulman et al., 2017): one with a large entropy coefficient that anneals over the course of training and another with a fixed coefficient. We refer to these as PPO (Sched.) and PPO, respectively. This selection was motivated by the results of Rudolph et al. (2025), who illustrated that appropriately tuned entropy schedules in standard policy gradient methods can perform comparably or better than specialized game-theoretic training algorithms in cyclic two-player zero-sum imperfect-information games.




(a) vs. random opponent.

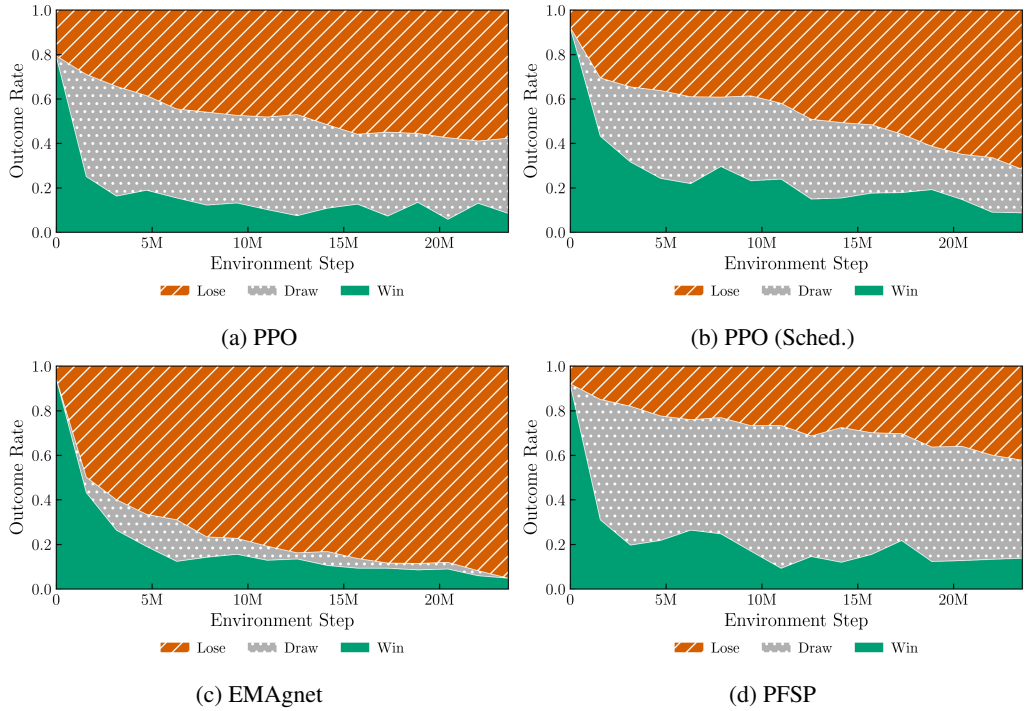
(b) vs. no-op opponent.

Figure 3: PPO, PPO (Sched.), EMAgnet, and PFSP baselines evaluated against two heuristic opponents: a (a) uniform-random policy and a (b) no-op policy. Curves are aggregated over 5 seeds; the shaded region represents 95% confidence intervals.

In addition, we illustrate training with EMAgnet (Maidment et al., 2026) and a population-based approach using the sampling rule of Prioritized Fictitious Self-Play (PFSP; Vinyals et al., 2019). The former is a regularization method for policy gradient algorithms that regularizes the policy towards an exponential moving average of its own weights, and it has been shown to outperform entropy regularization in standard benchmarks. The latter maintains a reservoir of past checkpoints and preferentially samples opponents that the current policy loses to. Full details on both algorithms are provided in Appendix C.

For all algorithms, we use an identical network structure and training regime: a 2-layer MLP over the 85-dimensional observation space. Each algorithm is trained for $2.5 \times 10^7$ environment steps with five random seeds. The complete set of parameters and environment configuration settings is provided in Appendix C.

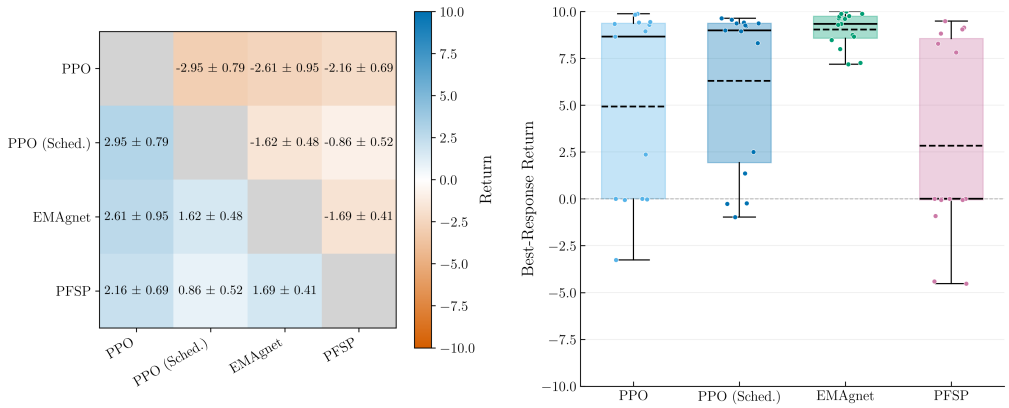
**Performance Against Heuristic Opponents.** In Figure 3, we show the win rate against uniform random and no-op opponents over the course of training. All policies achieve comparable performance against both opponents: their win rate increases to roughly 85–95% against the uniform random opponent. Because the no-op opponent never attacks, all non-winning matches end in timeouts, so the tie rate is simply $1 - p_{\text{win}}$. The results illustrate one of many areas for investigation that can be addressed in this environment. As the policies improve, they become increasingly reactive and less likely to initiate an engagement with an opponent who stands still. Although this behavior may be less exploitable, it may be undesirable. For example, it may go against the expectations of a game designer if their in-game AI fails to engage with players.




(a) PPO (b) PPO (Sched.) (c) EMAgnet (d) PFSP

Figure 4: Results of training approximate best responses for all (a) PPO, (b) PPO (Sched.), (c) EMAgnet, and (d) PFSP policies. Results are presented from the perspective of the trained policy (i.e., the loss rate is how often the best response policies win). PPO and PPO (Sched.) perform approximately equivalently. EMAgnet fares the worst, with the highest loss rate and the fewest ties. PFSP has a relatively high tie rate. The win rate is uniformly low across all algorithms. Three random seeds are used for each best response, resulting in 15 total for each algorithm (3 best response seeds $\times$ 5 algorithm seeds).

**Approximate Exploitability.** While the heuristic opponents provide a coarse view of skill, they say little about how robust each policy is to a dedicated adversary. We therefore also report the approximate exploitability of each set of policies by training approximate best responses to each. We train best responses using PPO against the fixed final checkpoint of each policy. This approach has become a common approximation of exploitability in games where it is intractable to calculate exact exploitability (e.g., Lanctot et al., 2019; Timbers et al., 2020; Sokota et al., 2022). The parameters used are identical to those of the fixed-entropy PPO policies (described in Appendix C). The results for each algorithm, in terms of the game outcome rates, are shown in Figure 4. We observe relatively




(a) Head-to-Head Returns

(b) Best Response Returns

Figure 5: (a) Head-to-head and (b) best response returns for all policies. PFSP has a positive return against all other algorithms. EMAgnet has a positive return against both PPO variants, but the best responses trained against it consistently attain the highest returns. In (a), each cell shows the expected return to the row policies when playing against the column policies (mean $\pm$ SEM). Each seed of the row algorithm plays each seed of the column algorithm for 96 games, for a total of $5 \times 5 \times 96 = 2{,}400$ games per cell. The return of the final best response checkpoints trained against each algorithm is represented in (b), with each point representing $\sim$140 games for each individual best response policy. Boxes span the interquartile range; the solid line is the median and the dashed line is the mean.

consistent results, with statistically significant differences between EMAgnet and PFSP in terms of final win rate and draw rate, with the latter performing more strongly against the best response (Holm-corrected Mann–Whitney $U$ test, $p = 0.048$ and $p = 0.045$, respectively). There are no significant differences in any corrected pairwise comparisons with either PPO variant.

To examine the relationship between skill and exploitability, Figure 5 shows both the head-to-head returns of all policies, as a proxy for skill, and the final returns of the best responses. PFSP has a positive return against all other policies, despite no statistically significant difference in exploitability relative to the PPO variants. Similarly, EMAgnet has a positive return against both PPO variants, but is more consistently exploited.

**Discovering Special Attacks.** Lastly, we show a distinguishing feature of FootsiesGym relative to other small-scale benchmark environments. As discussed in Section 3 (and further in Appendix E), the special attack actions are difficult to discover and learn to use effectively through random exploration. In particular, the directional special attack (`B_SPECIAL`) requires a player to select an attack action for 15 consecutive steps (at a frame skip of 4), releasing into either the FORWARD or BACKWARD action. There is a roughly 0.001% chance of this sequence occurring under a uniform random policy. The difficulty of learning this move is exacerbated by the fact that successful execution is dependent on the opponent's behavior. Figure 6 illustrates the usage of this move throughout training for all algorithms. It shows that throughout training, the average number of `B_SPECIAL` executions is near zero, with the one exception of the fixed-entropy PPO policy learning it for a brief period before it leaves the strategy space. This illustrates the complexity of the environment, as well as a potential incompatibility with standard game-theoretic training methods. Indeed, increasing entropy regularization, as is effective in small-scale benchmarks, may make it increasingly difficult for a policy to learn to utilize this move effectively. FootsiesGym also includes an option to augment the action space to make the use of special moves easier, as illustrated in Appendix E.

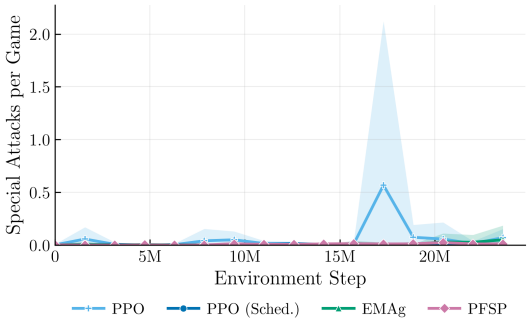



Figure 6: Usage of the directional special attack (`B_SPECIAL`) across algorithms. In the standard environment configuration, the move requires selecting any attack action (ATTACK, FORWARD+ATTACK, or BACK+ATTACK) for 15 contiguous steps (60 frames), followed by either the FORWARD or BACK action.

## 5 Discussion and Future Work

**Game Theoretic Training.** As previously discussed, FootsiesGym offers meaningful strategic complexity while remaining resource-efficient. This makes it a natural testbed for comparing, validating, and developing game-theoretic training regimes such as population-based training (Vinyals et al., 2019), PSRO (Lanctot et al., 2017), or regularization-based methods (Perolat et al., 2022; Sokota et al., 2022; Rudolph et al., 2025; Maidment et al., 2026). Our results also surface a concrete challenge for such methods: strategies that are valuable but hard to discover, such as the charged special attacks, may be driven out of the strategy space by the same regularization that these approaches rely on (Section 4). Understanding how game-theoretic training can retain such strategies is an open question that FootsiesGym is well positioned to study.

**Beyond Skill.** Beyond raw skill or exploitability minimization, an open question is what makes a policy *feel* like a compelling opponent rather than merely a strong one. In Section 4, we demonstrated that all baseline policies increase in skill but become increasingly reactive over time, failing to maintain a high win rate against a stationary opponent. We also showed that the learned policies almost never discover or consistently use special attacks, despite these being a core mechanic of the game. While high-skill policies are desirable in some settings, others may instead value policies that proactively engage opponents and make effective use of all core game mechanics, including special attacks. A growing body of work pursues this direction: Zhao et al. (2020) argue that "winning isn't everything," surveying agents designed for human-like behavior, playtesting, and game balancing rather than pure competitive optimality. Our environment illustrates a failure case for engaging play and prompts the question of how to develop algorithms that meet both game-theoretic and play-style requirements.

**Exploitability at Scale.** In game-theoretic environments, exploitability quickly becomes intractable to calculate directly. Indeed, sophisticated approaches are required for even relatively simple games (Rudolph et al., 2025), while more complex games require rough approximations, such as directly training a best response (Sokota et al., 2022). FootsiesGym, while being too complex for exact calculation, may provide a fruitful testbed for identifying approximate methods due to its speed and balanced complexity. Our baselines illustrate why such measures matter: EMAgnet attains positive head-to-head returns against both PPO variants yet is the most consistently exploited (Section 4), a disconnect that win-rate comparisons alone would not reveal.

**Human Evaluation and Imitation Learning.** Through the Unity game engine, the Footsies game can be exported to a WebGL build and deployed in the browser for human-subject experiments, enabling human-vs-AI evaluation and data collection for imitation learning. Such data would also

ground the play-style questions raised above: evaluation versus human participants affords the ability to directly assess whether a policy's behavior is perceived as engaging or fun.

**Limitations.** Footsies is a complex, real-time, imperfect-information game relative to many RL benchmarks. Nevertheless, there remains a substantial complexity gap between it and modern commercial fighting games. In addition, our reported exploitability is estimated with approximate best responses, which only lower-bound true exploitability (Timbers et al., 2020), and our baselines are only naïvely tuned; both caveats should be kept in mind when interpreting the comparisons in Section 4.

## A Environment configuration keys

Table 1 displays the environment configurations that are accessible to users. In all training runs, we use the default environment parameters. The exceptions are our demonstrations of the effects of manipulating `action_delay` and `use_special_charge_action`, which are illustrated in Appendix D and Appendix E, respectively.

| Key | Default | Description |
|---|---|---|
| `num_envs` | 1 | Parallel environments per process |
| `frame_skip` | 4 | Game frames per environment step (60 Hz native) |
| `action_delay` | 8 | Number of frames between action selection and execution |
| `max_t` | 4000 | Truncation horizon (environment steps) |
| `use_special_charge_action` | `False` | Expand action space to include charge toggles |
| `win_reward_scaling_coeff` | 10.0 | Magnitude of the terminal win/loss reward |
| `guard_break_reward` | 0.0 | Dense per-guard-break shaped reward |
| `use_reward_budget` | `False` | Deduct shaping from the win-reward budget |

Table 1: Selected FootsiesEnv config keys. `action_delay` must be a multiple of `frame_skip`.

## B Observation Features

Table 2 lists the features that make up each agent's observation. The Common block (distance between players) appears once. The Shared block appears twice, once for self and once for opponent. The Privileged block appears only for self, modeling the information asymmetry between two humans seated at separate controllers (e.g., a player does not directly see their opponent's special-attack charge timer). In total, the observation comprises 85 dimensions with the default action space and 88 dimensions with the special-charge option enabled.

## C Algorithm Details

In the following sections, we detail the parameters used for each of the three learning algorithms.

### C.1 Proximal Policy Optimization (PPO)

Table 3 lists the algorithm, optimization, and architecture hyperparameters used for both PPO self-play baselines in Section 4. The two variants differ only in the entropy coefficient: PPO (Sched.) linearly anneals it from 0.1 to 0, while PPO holds it fixed at 0.025.

### C.2 EMAgnet

EMAgnet was introduced by Maidment et al. (2026) and represents an alternative regularization scheme for policy gradient methods. We apply it to PPO, augmenting the loss function with a KL

| Block | Feature | Dim | Encoding |
|---|---|---|---|
| Common | inter-player distance ($\lvert x_{p_1} - x_{p_2} \rvert$) | 1 | scalar, normalized by stage width |
| Shared | `player_position_x` | 1 | scalar, normalized |
| | `velocity_x` | 1 | scalar, normalized |
| | `is_dead` | 1 | binary |
| | `vital_health` | 1 | scalar |
| | `guard_health` | 4 | one-hot over $\{0, 1, 2, 3\}$ |
| | `current_action_id` | 17 | one-hot over player script IDs |
| | `current_action_frame` | 1 | scalar, normalized |
| | `current_action_frame_count` | 1 | scalar, normalized |
| | `current_action_remaining` | 1 | scalar, normalized |
| | `is_action_end` | 1 | binary |
| | `is_always_cancelable` | 1 | binary |
| | `current_action_hit_count` | 1 | scalar |
| | `current_hit_stun_frame` | 1 | scalar, normalized |
| | `is_in_hit_stun` | 1 | binary |
| | `sprite_shake_position` | 1 | scalar |
| | `max_sprite_shake_frame` | 1 | scalar, normalized |
| | `is_face_right` | 1 | binary |
| | `current_frame_advantage` | 1 | scalar, normalized |
| Privileged | `would_next_forward_input_dash` | 1 | binary |
| | `would_next_backward_input_dash` | 1 | binary |
| | `special_attack_progress` | 1 | scalar, clipped to $[0, 1]$ |
| | `previous_action` | $\lvert\mathcal{A}\rvert$ | one-hot over agent actions |
| | `is_holding_special_charge` | 1 | binary |

Table 2: Per-feature breakdown of each agent's observation. Each observation contains the Common block (distance), two copies of the Shared block (self and opponent), and one copy of the Privileged block (self only). $|\mathcal{A}|$ is the size of the agent action space (6 by default, 9 with the special-charge action option enabled).

| Hyperparameter | Value |
|---|---|
| Total timesteps | $2.5 \times 10^7$ |
| Parallel game instances | 48 |
| Rollout length | 64 |
| Update epochs | 8 |
| Minibatches per epoch | 8 |
| Discount $\gamma$ | 0.995 |
| GAE $\lambda$ | 0.95 |
| PPO clip $\epsilon$ | 0.3 |
| Value-loss coefficient | 1.0 |
| Entropy coefficient | $0.1 \to 0$ (Sched.); 0.025 (fixed) |
| Max gradient norm | 0.5 |
| Learning rate | $1 \times 10^{-4} \to 0$ |
| Optimizer | Adam |
| Architecture | MLP, 2 hidden layers of 256 + ReLU |

Table 3: PPO hyperparameters used for the baselines in Section 4. The learning rate and the PPO (Sched.) entropy coefficient follow linear schedules.

divergence term towards a policy with the exponential moving average of the weights, described in Equation 1.

$$\mathcal{L}_{\text{PPO-EMAg}}(\theta) = \mathcal{L}_{\text{PPO}}(\theta) + \lambda_{\text{KL}}\, \mathbb{E}_{z\sim\mathcal{T}}\big[D_{\text{KL}}\big(\pi_{\theta_{\text{mag}}}(\cdot \mid z) \,\|\, \pi_\theta(\cdot \mid z)\big)\big], \tag{1}$$

where $\theta_{\text{mag}}$ denotes the magnet parameters and $\lambda_{\text{KL}} > 0$ controls the regularization strength. After each PPO update, the magnet parameters are updated as

$$\theta_{\text{mag}} \leftarrow (1-\tau)\,\theta_{\text{mag}} + \tau\,\theta, \tag{2}$$

with step size $\tau \in (0, 1]$. The parameters we used are provided in Table 4.

| Hyperparameter | Value |
|---|---|
| Total timesteps | $2.5 \times 10^7$ |
| Parallel game instances | 48 |
| Rollout length | 64 |
| Update epochs | 8 |
| Minibatches per epoch | 8 |
| Discount $\gamma$ | 0.995 |
| GAE $\lambda$ | 0.95 |
| PPO clip $\epsilon$ | 0.3 |
| Value-loss coefficient | 1.0 |
| Entropy coefficient | $3 \times 10^{-3}$ |
| Max gradient norm | 0.5 |
| Learning rate | $1 \times 10^{-3} \rightarrow 0$ |
| Magnet update rate $\tau$ | $1 \times 10^{-4}$ |
| Magnet KL coefficient $\lambda_{\text{KL}}$ | 0.5 |
| Optimizer | Adam |
| Architecture | MLP, 2 hidden layers of 256 + ReLU |

Table 4: EMAgnet hyperparameters used for the baseline in Section 4. Learning rate utilizes a linear schedule.

### C.3 Prioritized Fictitious Self-Play (PFSP)

We train PFSP agents using the prioritized fictitious self-play opponent-sampling rule of Vinyals et al. (2019), applied to a bounded reservoir of the training agent's own past checkpoints rather than a full multi-agent league. At fixed snapshot intervals, we snapshot the current policy into a reservoir of $N$ slots; once the reservoir is full, new snapshots replace existing ones via reservoir sampling. During training, each game is played against the current policy with probability $p_{\text{sp}} = 0.8$ (mirror self-play) and against a stored snapshot otherwise. The remaining $1 - p_{\text{sp}}$ of the opponent budget is distributed over the snapshots in proportion to a priority that favors opponents the agent is currently losing to.

Following the weighting of Vinyals et al. (2019), snapshot $i$ is sampled with weight

$$w_i \;\propto\; \left(1 - \hat{p}_i\right)^{1/2}, \tag{3}$$

where $\hat{p}_i$ is the estimated win rate of the current policy against snapshot $i$; snapshots the agent loses to more often are thus played more frequently. The win rates $\hat{p}_i$ are refreshed on the same cadence as snapshotting through dedicated evaluation games and smoothed with an exponential moving average, and a slot is reset to a neutral $\hat{p}_i = 0.5$ whenever it is overwritten. Unlike the original formulation, we construct the opponent pool from checkpoints of a single continuously trained policy rather than maintaining a league of independently trained best-response agents. The complete hyperparameters are listed in Table 5.

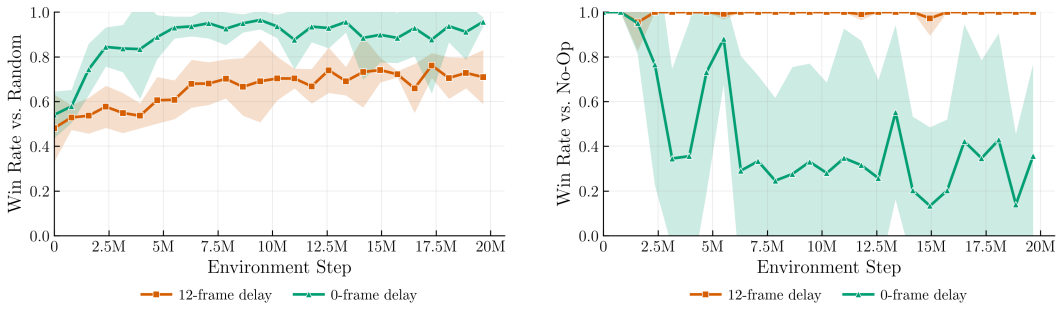
## D Effect of Action Delay on Policy Quality

Figure 7 compares PPO self-play policies trained with action delays of 0 and 12. All other hyperparameters and environment settings are identical to those in Appendix C.1. Although the 0-delay

| Hyperparameter | Value |
|---|---|
| Total timesteps | $2.5 \times 10^7$ |
| Parallel game instances | 48 |
| Rollout length | 64 |
| Update epochs | 8 |
| Minibatches per epoch | 8 |
| Discount $\gamma$ | 0.995 |
| GAE $\lambda$ | 0.95 |
| PPO clip $\epsilon$ | 0.3 |
| Value-loss coefficient | 1.0 |
| Entropy coefficient | $0.1 \to 0$ |
| Max gradient norm | 0.5 |
| Learning rate | $1 \times 10^{-4} \to 0$ |
| Reservoir size $N$ | 16 |
| Snapshot interval (env. steps) | 614,400 |
| Self-play proportion $p_{\text{sp}}$ | 0.8 |
| Optimizer | Adam |
| Architecture | MLP, 2 hidden layers of 256 + ReLU |

Table 5: PFSP hyperparameters used for the baseline in Section 4. Learning rate and entropy coefficient use linear schedules.

policy attains a higher win rate against the random opponent, it is completely degenerate against the no-op opponent. Consistent with visual inspection, the 0-delay policy is brittle in general despite its strong numerical performance against random. On the other hand, the 12-delay policy has worse performance against random, but has a near 100% win rate against no-op.




(a) vs. random opponent.

(b) vs. no-op opponent.

Figure 7: PPO self-play policies trained with `action_delay` $\in \{0, 12\}$, evaluated against a random opponent (a) and a no-op opponent (b). Curves are aggregated over 5 seeds; the line is the mean and the shaded band is the 95% confidence interval.

The impact of action delay manipulations is inherently tied to the frame data in Footsies. Whether a move is reactable or must be predicted is determined by the length of its startup, active, and recovery frames. Manipulations of the action delay lever (and frame skip) should be informed by the game's frame data. The frame data of the attacks is shown in Table 6.

The frame data allows us to calculate whether or not an attack is reactable at a given action delay. We illustrate this in Figure 8.

| Attack | Startup | Active | Recovery |
|---|---|---|---|
| N_ATTACK | 4 | 2 | 16 |
| B_ATTACK | 3 | 3 | 15 |
| N_SPECIAL | 11 | 4 | 29 |
| B_SPECIAL | 2 | 6 | 47 |

Table 6: Frame data for the four attacks in Footsies. Startup is the window to block on reaction; recovery is the window to whiff-punish on reaction. Whiff-punishing refers to attacking a player that missed an attack.

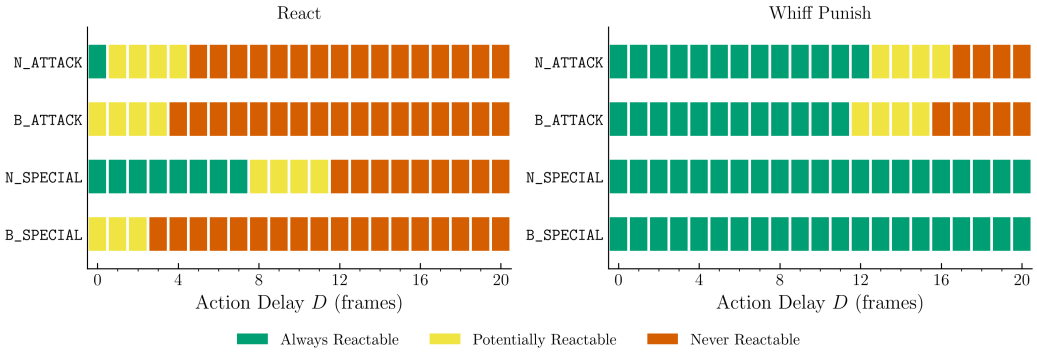



Figure 8: Reactability of each attack across action delays $D$. A stimulus can land anywhere in the skip window, adding 0 frames in the best case and the full frame skip (4 in this figure) in the worst case. As a result, each delay is one of three states: Always Reactable, Potentially Reactable, or Never Reactable. The reaction window is dictated by the attack's startup or recovery frames. At zero delay every attack is at least potentially reactable, but only N_ATTACK and N_SPECIAL are reactable in the worst case; both special attacks have long recoveries that permit punishment at any reasonable delay.

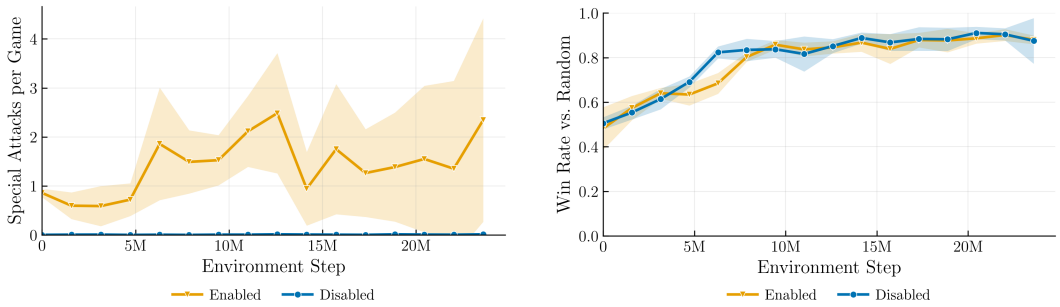
## E Effect of Enabling Special Charge Actions

There are two distinct special attacks in Footsies (scripts N_SPECIAL and B_SPECIAL). Both, if landed on a vulnerable opponent, win the round. The special attacks can be executed in several different ways. First, both can be executed by "charging" the attack action. If the attack action is held for at least 60 frames (or 15 steps at a frame skip of 4) then released, a special attack executes. If either the forward or backward directional actions are pressed when attack is released, the B_SPECIAL move is executed. If no directional modification is applied when the attack is released, N_SPECIAL is executed. Further, N_SPECIAL can also be triggered in a combination attack: if a regular attack is already being executed and the attack action is selected again, a combination is triggered that executes N_SPECIAL as a follow-up to the regular attack.

As discussed in Section 4, discovering the effective use of special attacks is difficult under random exploration due to the extended "charge" duration and strategic timing requirement. The use_special_charge_action option in FootsiesGym is designed to alleviate this difficulty by adding additional actions that toggle the charge state of the special attack actions. All three of the added actions enable a charge state when first pressed: any action that is executed afterwards will result in the composite action with ATTACK added to it (for example, FORWARD becomes FORWARD+ATTACK). When pressed again, the charge state is disabled and a corresponding action is executed: FORWARD-CHARGE becomes FORWARD, BACK-CHARGE becomes BACK, and NEUTRAL-CHARGE becomes NO-OP.

To illustrate the effect of this option, we train PPO (Sched.) with and without it enabled. Figure 9 shows move usage rates over training and performance against a random opponent. We see a

substantial increase in usage of the move, along with comparable performance against the uniform random opponent.




(a) Usage of `B_SPECIAL` action.

(b) Win rate vs. random opponent with and without special charge actions.

Figure 9: Comparison of PPO (Sched.) training with and without special charge actions enabled. The usage of `B_SPECIAL` is shown in (a), and the win rate is shown in (b).

## References

Farama Foundation. Stable-Retro: A maintained fork of openai's retro library. https://stable-retro.farama.org/, 2023. Source code at https://github.com/Farama-Foundation/stable-retro.

HiFight. FOOTSIES: A 2d fighting game. https://github.com/hifight/Footsies, 2018. Open-source game; accessed 2026.

Ibrahim Khan, Thai Van Nguyen, Xincheng Dai, and Ruck Thawonmas. Darefightingice competition: A fighting game sound design and ai competition. In *2022 IEEE Conference on Games (CoG)*, pp. 478–485. IEEE, 2022.

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.

Marc Lanctot, Edward Lockhart, Jean-Baptiste Lespiau, Vinicius Zambaldi, Satyaki Upadhyay, Julien Pérolat, Sriram Srinivasan, Finbarr Timbers, Karl Tuyls, Shayegan Omidshafiei, et al. OpenSpiel: A framework for reinforcement learning in games. *arXiv preprint arXiv:1908.09453*, 2019.

Feiyu Lu, Kaito Yamamoto, Luis H Nomura, Syunsuke Mizuno, YoungMin Lee, and Ruck Thawonmas. Fighting game artificial intelligence competition platform. In *2013 IEEE 2nd Global Conference on Consumer Electronics (GCCE)*, pp. 320–323. IEEE, 2013.

Tristan Maidment, JB Lanier, Chase McDonald, Nathan Tsang, Eugene Vinitsky, Roy Fox, Albert Wang, and Wesley N Kerr. Emagnet: Parameter-space ema regularization for policy gradient self-play in large games. *arXiv preprint arXiv:2606.23995*, 2026.

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.

Alessandro Palmas. DIAMBRA Arena: A new reinforcement learning platform for research and experimentation. *arXiv preprint arXiv:2210.10595*, 2022.

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.

Max Rudolph, Nathan Lichtle, Sobhan Mohammadpour, Alexandre Bayen, J Zico Kolter, Amy Zhang, Gabriele Farina, Eugene Vinitsky, and Samuel Sokota. Reevaluating policy gradient methods for imperfect-information games. *arXiv preprint arXiv:2502.08938*, 2025.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Samuel Sokota, Ryan D'Orazio, J Zico Kolter, Nicolas Loizou, Marc Lanctot, Ioannis Mitliagkas, Noam Brown, and Christian Kroer. A unified approach to reinforcement learning, quantal response equilibria, and two-player zero-sum games. *arXiv preprint arXiv:2206.05825*, 2022.

Yujin Tang, Yingtao Tian, and David Ha. Evojax: Hardware-accelerated neuroevolution. *arXiv preprint arXiv:2202.05008*, 2022.

J. Terry, Benjamin Black, Nathaniel Grammel, Mario Jayakumar, Ananth Hari, Ryan Sullivan, Luis S. Santos, Clemens Dieffendahl, Caroline Horsch, Rodrigo Perez-Vicente, Niall L. Williams, Yashas Lokesh, and Praveen Ravi. PettingZoo: Gym for multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 34:15032–15043, 2021.

Finbarr Timbers, Nolan Bard, Edward Lockhart, Marc Lanctot, Martin Schmid, Neil Burch, Julian Schrittwieser, Thomas Hubert, and Michael Bowling. Approximate exploitability: Learning a best response in large games. *arXiv preprint arXiv:2004.09677*, 2020.

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.

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, 2020.

Yunqi Zhao, Igor Borovikov, Fernando de Mesentier Silva, Ahmad Beirami, Jason Rupert, Caedmon Somers, Jesse Harder, John Kolen, Jervis Pinto, Reza Pourabolghasem, et al. Winning is not everything: Enhancing game development with intelligent agents. *IEEE Transactions on Games*, 12(2):199–212, 2020.