Human-Like Goalkeeping in a Realistic Football Simulation: a Sample-Efficient Reinforcement Learning Approach

Alessandro Sestini[1*] **Joakim Bergdahl**[1] **Jean-Philippe Barrette-LaPierre**[1]
Florian Fuchs[1] **Brady Chen**[2] **Micheal Jones**[2*] **Linus Gisslén**[1]
[1] SEED - Electronic Arts, Stockholm, Sweden
[2] EA Sports, Vancouver, Canada
{asestini, michaelj}@ea.com

Abstract

While several high profile video games have served as testbeds for Deep Reinforcement Learning (DRL), this technique has rarely been employed by the game industry for crafting authentic AI behaviors. Previous research focuses on training super-human agents with large models, which is impractical for game studios with limited resources aiming for human-like agents. This paper proposes a sample-efficient DRL method tailored for training and fine-tuning agents in industrial settings such as the video game industry. Our method improves sample efficiency of value-based DRL by leveraging pre-collected data and increasing network plasticity. We evaluate our method training a goalkeeper agent in *EA SPORTS FC 25*, one of the best-selling football simulations today. Our agent outperforms the game's built-in AI by 10% in ball saving rate. Ablation studies show that our method trains agents 50% faster compared to standard DRL methods. Finally, qualitative evaluation from domain experts indicates that our approach creates more human-like gameplay compared to hand-crafted agents. As a testimony of the impact of the approach, the method is intended to replace the hand-crafted counterpart in next iterations of the series.

1 Introduction

Deep Reinforcement Learning (DRL) research has demonstrated significant potential in areas such as robotics [Schulman et al., 2015, Sferrazza et al., 2024], control of nuclear fusion plasma in a tokamak [Degrave et al., 2022], and design of faster sorting algorithms [Mankowitz et al., 2023]. At the same time, the video game industry experienced significant technological progress – in areas such as computer graphics, physics, and design – that led to the development of more complex and immersive game experiences. Despite this progress, designing Artificial Intelligence (AI) systems to manage Non-Player Characters (NPCs) remains a complex element of the creative process that significantly influences the quality of games [Jacob et al., 2020]. Games provide a natural testbed for DRL research, and the application of DRL in this context has shown great promise. Notable examples such as OpenAI Five [Berner et al., 2019] and AlphaStar [Vinyals et al., 2019] show how DRL can outperform professional players in complex video games. These successes have accelerated the deployment of this technique in commercial video games, with examples such as GT Sophy [Wurman et al., 2022], Naruto Mobile [Zhang et al., 2024a], and Arena Breakout [Zhang et al., 2024b], while still not yet widely employed in the industry.

The success of the aforementioned approaches depends on extensive training, requiring a massive amount of online environment interactions and large neural networks. Although DRL can find

*Corresponding authors

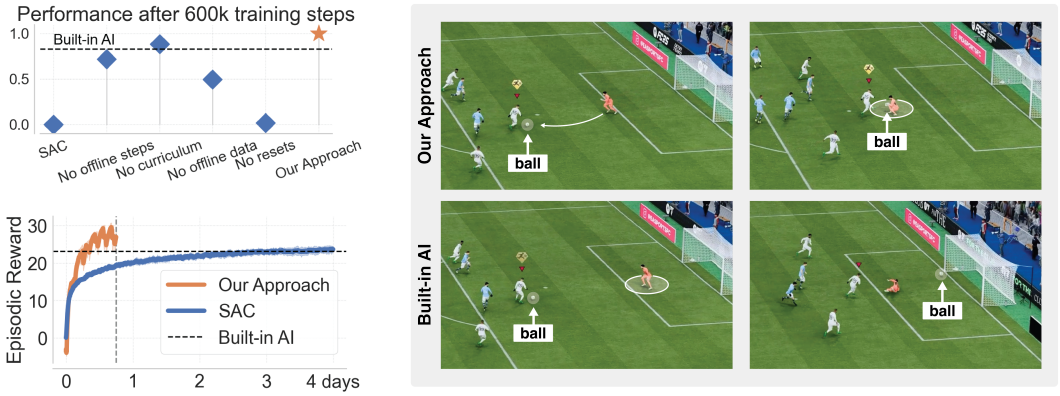

Figure 1: **Main results of our approach**. **Top-left**: our approach compared to removing the variations we add on top of the standard SAC algorithm. **Bottom-left**: training time compared to standard SAC algorithm. Our method outperforms built-in AI in less than one day. The standard SAC algorithm is able to match the performance of the built-in AI, but only after 4 days of training. **Right**: an example showcasing the behavioral differences between our agent (top) and the built-in AI (bottom) in the same situation. Our agent is more proactive and better understands the current situation, anticipating the shot. Quoting a professional goalkeeper: *"the goalkeeper plays it really well! The keeper looks for opportunities to steal ground as the striker enters the box."*

well-performing policies with enough interactions, applying it in real-world scenarios is complex. For instance, the constantly evolving nature of games makes it difficult for developers to deploy solutions that require days or weeks to produce results [Gillberg et al., 2023]. Therefore, such application requires a method that: (1) outperforms classical hand-crafted AI systems both in terms of numerical performance as well as perceived human-likeness; (2) trains quickly, both in terms of sample efficiency and wall-clock time; and (3) is easy to adjust without retraining agents from scratch.

This paper proposes a sample-efficient method for training human-like AI in video games that is computationally efficient enough to be practically applied in production. Moreover, the method allows for easy modifications to the agent without restarting the training procedure from scratch. We analyze recent research on sample-efficient and offline RL [Schwarzer et al., 2023, Ball et al., 2023, Wang et al., 2024] and showcase the techniques we combined and extended for developing our method. We evaluate our approach on *EA SPORTS FC 25*, a AAA football simulation game. In particular, we train the goalkeeper's positioning system. This choice stems from the need to improve the hand-crafted, non-realistic, and complex-to-maintain AI. Determining where to position the goalkeeper to better save a goal or anticipate opponents is complex, and manually crafting the decision-making process is challenging. Not to mention, developing a system that mimics real human players is usually a time-consuming and difficult task.

In summary, the key contributions are: (1) we propose a new sample-efficient DRL method that marks a step forward in the practical application of DRL in real-world settings; (2) we propose a framework for improving an agent's behavior through expert feedback; (3) we show how the method is able to train a human-like agent that consistently outperforms the built-in AI; and (4) we conduct extensive ablations to show our contributions to the sample-efficient DRL landscape. A summary of the results is shown in Figure 1. Although the main focus of this paper is on game development, the findings transfer to other areas where sample efficiency and human-likeness are fundamental challenges, as shown in additional benchmark domains (see Appendix G).

2 Related Work

The challenge of improving sample efficiency in DRL has been gaining interest from the research community, highlighting its importance for successful applications. Additionally, an increasing number of game studios are attempting to deploy DRL agents in games.

Sample-efficient DRL. DRL requires extensive interaction with the environment to reach the desired performance, which becomes impractical in real-world applications where such interactions

are expensive, time-consuming, or risky. Yarats et al. [2021] use data augmentation to design a sample-efficient DRL method, while Schwarzer et al. [2020] use a self-supervised temporal consistency loss with data augmentation; EfficientZero is a model-based RL algorithm with self-supervised learning to learn a temporally consistent environment model and use it to correct off-policy value targets [Wang et al., 2024]. However, most of these approaches focus on learning with limited data, often at the cost of increasing computational resource needs, resulting in time-consuming training.

Recent results suggest that scaling the replay ratio factor – the number of optimization steps over environment steps – is a simple but effective approach for enhancing sample efficiency when combined with periodic network resets [D'Oro et al., 2022, Schwarzer et al., 2023]. Moreover, recent literature in offline RL such as RLPD [Ball et al., 2023] and SDBG [Macaluso et al., 2024] shows great promise in using offline data for boosting online policies. In this work, we combine and extend several of the latest advances in sample efficiency to address the challenges defined in Section 1.

DRL for video games. Recent advancements in DRL have led to impressive results in complex games such as StarCraft II, Dota 2, and Gran Turismo 7 [Vinyals et al., 2019, Berner et al., 2019, Wurman et al., 2022]. However, these approaches aim to create super-human agents with extraordinary resources. For instance, the work by Berner et al. [2019] took months to train an agent that could beat professional players in the game Dota 2.

As stated by Jacob et al. [2020], the video game industry does not need agents built to "beat the game", but rather to produce human-like behaviors. Similarly, many other fields such as robotics or autonomous driving do not require super-human agents, but rather agents that can better fit the context. In video games, some notable examples include: DeepCrawl, an effective DRL system that is able to create a variety of NPC behaviors for a published roguelike game [Sestini et al., 2020]; the work by Zhang et al. [2024b] that trained a DRL agent for the game Arena Breakout; and Shūkai, a practical DRL algorithm specifically tailored for commercial fighting games [Zhang et al., 2024a]. Although the goal of the cited works is to develop an agent suitable for integration into a real product rather than solely outperforming human performance, none of the papers address the practical implications of training a sample-inefficient DRL agent in a sample-restricted setting.

3 Methodology

In here, we first define preliminaries useful for understanding the subsequent sections. Second, we describe the algorithm we use and our evaluation framework. Finally, we show how we enable human-in-the-loop for easy fine-tuning of an under-performing agent in specific scenarios.

3.1 Preliminaries

An RL setting is commonly formalized as a Markov Decision Process (MDP) which consists of a tuple $\langle S, A, R, P, \gamma \rangle$, where S is the space state, A the action space, $P : S \times A \to S$ the transition function, $R : S \times A \to \mathbb{R}$ the reward function and $\gamma \in [0, 1)$ the discount factor. A policy π is formalized as a function that maps states to a distribution of actions. The goal is to find an optimal policy that maximizes the sum of expected discounted reward. We represent this objective with an action-value function $Q^\pi(s_t, a_t) = \mathbb{E}_{\pi,P}[\sum_{k=1}^{N} \gamma^k r_{t+k+1} \mid s_t \in S, a_t \in A]$, where N is the time horizon. In the case of offline RL, the agent does not interact with the environment, but learns from a fixed offline dataset [Levine et al., 2020]. Offline RL typically assumes access to N previously-collected transitions $D = \{s_t^i, a_t^i, r_t^i, s_{t+1}^i\}_{i=1}^N$, which are gathered using a policy π_o. The goal remains to find the optimal policy that maximizes the expected discounted reward.

In this work, we build our method on top of the value-based Soft Actor-Critic (SAC) [Haarnoja et al., 2018] algorithm. We give a thorough description of SAC in Appendix A. Most modern DRL value function-based approaches store the agent's experience with the environment in a replay buffer, potentially keeping that data for the entire training period. In this regard, the replay ratio – the ratio between the number of gradient updates and the number of environment steps– plays a crucial role. For instance, the original DQN algorithm [Mnih et al., 2015] uses a replay ratio of 0.25, while recent sample-efficient algorithms such as BBF [Schwarzer et al., 2023] or SR-SPR [D'Oro et al., 2022] use 8 and 16.

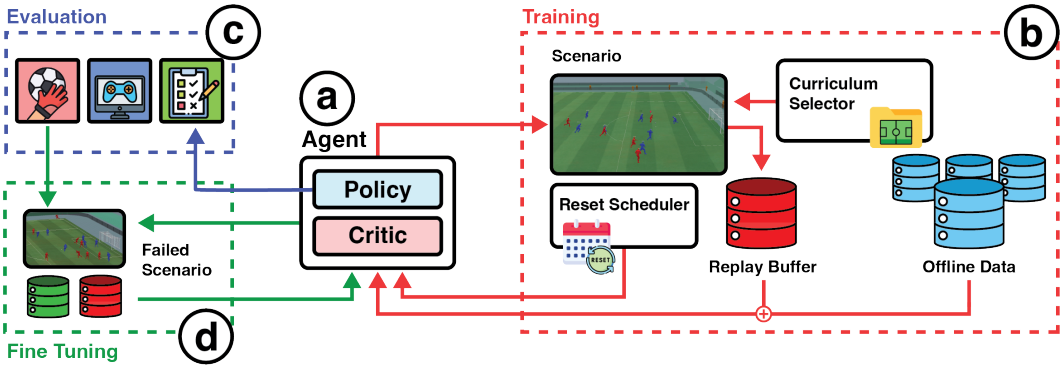

Figure 2: **Overview of the proposed method**. (a) shows the agent composed of the policy and action-value functions as employed by SAC. (b) shows the main training framework, composed of the elements delineated in Section 3.3: *curriculum learning*, *offline data*, and *network resets*. (c) shows the three components of the evaluation framework: *automatic quantitative evaluation*, *human qualitative evaluation*, and *expert-authored test suite*, described in Section 3.6. Finally, (d) shows the fine tuning component. In this, we use only the failed scenario to generate new data but we combine it with the previously collected replay buffer. More details in Section 3.7.

3.2 EA SPORTS FC 25

We evaluate our method on the game *EA SPORTS FC 25*, developed by Electronic Arts (EA). The game is part of the *EA SPORTS FC* series, with new entries released every year. *EA SPORTS FC 24* reached a peak of 21 million weekly active users in fiscal year 2024. Figure 1 shows screenshots of the game. The game is a fully physics-based football simulation, where players play against other humans and in-game AI systems. While the outfield players in the game have advanced, hand-crafted AI, the goalkeeper system suffers from suboptimal behavior. We aim to improve upon this system by leveraging DRL, while respecting the requirements listed in Section 1. We train our agent using a low-resolution version of the game. This version removes all graphical enhancements unnecessary for training, and it allows for unlocked frame rates, speeding up the simulation by a factor of three.

3.3 Algorithm

Our algorithm extends the base SAC approach by incorporating modifications aimed at improving sample efficiency. Here, we list all the changes applied to the base algorithm.

Replay ratio and hard reset. Following the SR-SPR [D'Oro et al., 2022] and BBF [Schwarzer et al., 2023] algorithms, and in contrast to classical approaches [Mnih et al., 2015, Haarnoja et al., 2018], in our experiments we use a replay ratio of 1. In comparison with SR-SPR and BBF, which employ soft resets of the value function, we perform hard resets of both the policy and value function networks every 10^5 steps. Soft resets involve the reinitialization of a subset of weights, such as the last layers, while hard resets completely reinitialize the network parameters. Moreover, unlike the other two approaches, the moment we apply a hard reset, we increase the replay ratio to 6.4×10^3 and train using the current buffer, without allowing other online environment interactions. This is equivalent to performing a vast number of offline updates during the reset. After the offline steps, we resume online training with a replay ratio of 1. In Section 4.3, we show how these modifications help improve the sample efficiency of the algorithm while keeping the computational cost low.

Scenario-based learning. A full football match provides only a few salient situations for the goalkeeper. Hence, it is more efficient to train using specific situations. With the help of a domain expert, we define scenarios mimicking real training challenges faced by human goalkeepers. A scenario includes one or more situations. These situations vary in the starting positions of the players; in the particular situation played, e.g. a corner kick; and in the behavior of the other players, e.g. the skill level of the built-in AI for the opponents and teammates. For training the agent using scenarios, we employ curriculum learning [Bengio et al., 2009], dividing the scenarios into multiple phases. When we move from one phase to the next, to mitigate the risk of catastrophic forgetting, we reuse

a subset of scenarios encountered in previous phases. Example of scenarios and more details on curriculum learning are described in Appendix E.2.

Learning with offline data. The game already features a built-in AI solution for the goalkeeper. We decided to leverage the behavior of the built-in AI to bootstrap the phases of the curriculum. For this goal, we first collect a dataset of N transitions running the built-in AI for each curriculum phase: $D_{\pi_o} = \{s_t^i, a_t^i, r_t^i, s_{t+1}^i\}_{i=1}^N$, where π_o represents the built-in AI, and r_t is generated from the reward function. Similarly to RLPD [Ball et al., 2023], we then use the symmetric sampling technique, where for each training batch we sample 50% of the data from the agent's replay buffer D_π, and the remaining 50% from D_{π_o}. We use layer normalization to mitigate catastrophic overestimation. However, unlike RLPD, we do not employ an ensemble of Q-networks. Using multiple Q-networks can improve sample efficiency but requires significant computational resources. In our preliminary experiments, we observed that we could achieve substantial improvements in sample efficiency using only the offline dataset and layer normalization. Furthermore, D_{π_o} is sub-optimal. Hence, in order to outperform the built-in AI, we remove D_{π_o} after the first reset in each of the curriculum phases.

3.4 Action and state spaces

The action space of the agent consists of three continuous values: two values for the relative target position, and one value for the intensity of the movement. We use the same state space for both the agent's policy and action-value functions. These features are directly available from the game engine. We use a total of 110 features, each normalized to the range $[-1, 1]$. The state space is divided into three main components: *goalkeeper features*, a set of 33 ego-centric values such as relative positions of the ball, relative positions of the goal, and velocities of the goalkeeper; *opponent features*, a set of 65 values about the 5 closest opponents to the goal, 13 values for each opponent including relative position to the goalkeeper and velocities; and *teammate features*, a set of 12 values about the 4 closest teammates to the goal, 3 values for each teammate including relative position to the goalkeeper. Our preliminary experiments showed that adding more than 5 opponents and 4 teammates to the input of the agent does not significantly influence the overall performance. Appendix C provides a detailed description of the action and state spaces.

3.5 Reward function

In this section, we describe the methodology used for finding a suitable reward function from exchanges with domain experts. The mathematical notation of the resulting function is reported in Appendix D. With the help of a professional goalkeeper, we first defined situations that real human goalkeepers face during training. We translated these situations into training scenarios. From these scenarios, we derived the reward function by asking the expert how a real goalkeeper would act. The simplicity of the scenarios allowed us to quickly train and test the agent's behavior in this first iteration. During this phase, we focused on the qualitative behavior rather than the agent's overall performance. After extensive experimentation, we defined the reward function with three main components: a sparse component for saving goals, a dense component for encouraging play near the middle line, and penalty components for reducing noisy movements. The main motivation for the latter two components is that human players need to conserve energy, whereas the agent has infinite stamina and can exploit this to move more and save more shots. The scenarios used in this first iteration are part of the first phase in the curriculum, while new and more complex scenarios are added in subsequent phases. However, the same reward function applies across all training phases.

3.6 Evaluation framework

We define an evaluation framework composed of three steps. We run the evaluation framework after reward convergence during the main training procedure.

Automatic quantitative evaluation. For evaluating the performance of the agent, we use a specific scenario built similarly to the training ones. This scenario covers multiple situations where the agent faces an opponent that shoots towards the goal for 2,000 shots, using different types of shots with varying levels of difficulty. The metric we use is the percentage of saves – the higher, the better. This specific scenario is not used for training the agent, but only for evaluating it. Appendix F.1 shows details of the scenario.

Method	Training evaluation	Quantitative evaluation	Expert-authored test suite
	Success Rate		Completion Ratio
Our method	$\mathbf{90.0\% \pm 1.22}$	$\mathbf{73.46\% \pm 1.04}$	$91.5\% \pm 0.3$
Built-in AI	$82.6\% \pm 1.67$	$65.58\% \pm 1.18$	$\mathbf{94.0\% \pm 0.0}^*$

Method	Goal Conceding Rate ↓	Ball Saving Rate ↑
Our method	$\mathbf{25.25\%}$	$\mathbf{54.12\%}$
Built-in AI	29.10%	48.27%

Table 1: **Results compared to the main baseline, the built-in AI in EA SPORTS FC 25. Left**: the performance over three benchmark tests, reported over 5 different seeds. The table shows how our method achieves better performance than the built-in AI, while at the same time passing most of the expert-authored tests. **Right**: our agent and built-in AI facing an experienced human player. The player plays against either our agent or the built-in AI for 400 games. * The built-in AI and the tests in the expert-authored suite are deterministic.

	Main Agent	Tuned Agent 1	Tuned Agent 2	Tuned Agent 3
		Success Rate		
Last curr phase	$90.0\% \pm 1.22$	$88.0\% \pm 2.45$	$88.4\% \pm 2.40$	$88.6\% \pm 1.67$
Failed scenario 1	$28.4\% \pm 2.88$	$77.6\% \pm 1.51$	$70.4\% \pm 3.65$	$69.0\% \pm 2.24$
Failed scenario 2	$13.8\% \pm 1.64$	$00.1\% \pm 0.16$	$60.4\% \pm 2.07$	$49.2\% \pm 0.83$
Failed scenario 3	$00.0\% \pm 0.00$	$00.0\% \pm 0.00$	$00.1\% \pm 0.18$	$14.0\% \pm 1.00$
Average	$33.0\% \pm 1.15$	$41.4\% \pm 1.03$	$54.8\% \pm 2.07$	$\mathbf{55.2\% \pm 1.43}$

Table 2: **Results of fine-tuning.** We report the success rate of the Main Agent, which is trained from scratch using the main method, and Tuned Agents, which have been fine-tuned sequentially on one failed scenario at a time, evaluated over the last curriculum phase and failed scenarios. Failed scenarios are those identified by domain experts where the main agent under-performed. The last row shows the average result over all the evaluation tests. For each fine-tuning iteration, we perform 200,000 training steps compared to the 600,000 steps in the main procedure. More details in Section 4.1.

Human qualitative evaluation. Integrating the agent into the game while keeping the experience enjoyable requires a subjective evaluation from human players, making their feedback crucial for evaluation. To achieve this, we leverage the Quality Verification (QV) team within the game studio, and ask professional players to play the game and provide feedback on how to improve the agent. This is important in scenarios where it performs poorly due to a lack of experience during training.

Expert-authored test suite. To control specific cases – such as critical, edge or challenging cases – we deploy a set of qualitative tests. Each test includes a situation (including starting positions, velocities, and rotation of both players and the ball) and hand-crafted conditions to assess whether the behavior of the model meets the designers' expectations. This allows us to keep track of each critical situation automatically. The initial test suite was made for the built-in AI. During manual evaluation, each time a new situation is identified as problematic or failing, a test case is created. Keeping the test in the evaluation suite ensures that the behavior stays good even after re-training. A total of 344 tests are created to cover all critical situations. We give examples of such test cases in Appendix F.2.

3.7 Improving Behavior Through Expert Feedback

We propose a new framework for incorporating domain experts' feedback to improve performance. After the main training converges, domain experts such as professional goalkeepers and QV testers evaluate the agent's behavior using the evaluation framework described in Section 3.6. Whenever they identify a situation in which the agent does not perform as intended, such as a scenario not covered in the initial training or a discovered exploit in the agent's behavior, they create new scenarios and fine-tune the agent specifically on those. For our fine-tuning process, we re-purpose Replay across Experiment (RaE), a symmetric sampling scheme to leverage offline data from previous experiments to improve exploration and bootstrap learning [Tirumala et al., 2023]. While the paper shows that this can improve performance when mixed with a majority of online data, we show that this approach can be used with a minority of online data in the sense of a fine-tuning approach. We define D_{π_0} as the replay buffer collected during the main training process. We then follow these steps:

1. Create a training scenario based *on only one failed scenario* identified during evaluation;

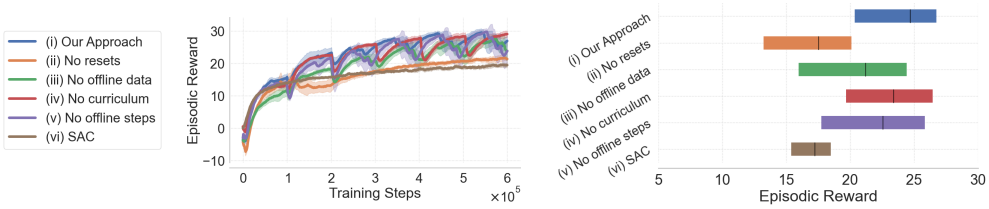

Figure 3: **The impact of all the components of our method**. **Left**: training curves comparing our approach removing different components. The curve represents the mean while the shaded areas represent the standard deviation of 5 different seeds. The drops in the plot correspond to network resets. **Right**: interquantile mean of the reward of all the ablations measured at the end of the training. As the plot shows, our agent achieves the highest performance in fewer training steps than the ablations, being more stable.

2. Run a training process using new policy and action-value function networks, and only the new training scenario rather than those defined in the curriculum;
3. During network updates, for each training batch, sample 50% of the data from the previous buffer D_{π_i} and 50% from the new online buffer $D_{\pi_{i+1}}$. Here, π_i is the policy trained at previous iteration i, while π_{i+1} is the policy trained during fine tuning.
4. Combine the two buffers after training, so that $D_{\pi_{i+1}} = D_{\pi_i} \cup D_{\pi_{i+1}}$. Return to step 1.

This process can be repeated for each failed scenario that testers find. Using this technique, we can leverage the knowledge learned from the previous buffer while acquiring new skills to solve the failed scenarios in a sample-efficient manner, as we will show in Section 4. Figure 2 shows an overview of all the components of the algorithm.

4 Experiments

In this section, we present the experimental results of our method using *EA SPORTS FC 25* as a training and evaluation platform, and in particular the goalkeeper positioning system as control problem (described in Section 3.2). Additionally, we provide results using the MuJoCo suite in Appendix G, demonstrating how our method generalizes to other environments as well. We conduct three types of studies: a quantitative study, where we test the performance of our agent against the built-in AI; a qualitative study, where we compare the qualitative behavior of our agent to that of the built-in AI; and an efficiency study, where we define ablated versions of our method to assess the contribution of each component to performance efficiency. We train each of our main agents for 600,000 training steps, while for the fine-tuning experiments we retrain the agent for 200,000 steps. On average, it takes between 18 and 24 hours to train the agent using all curriculum phases, plus an additional 3 to 6 hours for each fine-tuning iteration. More details about the computational resources, network architectures, and all hyper-parameters are provided in Appendix E.

4.1 Quantitative study

First, we evaluate the performance of an agent trained from scratch using the approach detailed in Section 3.3. Then, we evaluate the performance of our fine-tuning approach, detailed in Section 3.7. Our primary baseline is the built-in AI. The main metrics we use are: the mean success rate over 500 episodes using training scenarios; the results of the automatic quantitative evaluation; and the completion ratio of the expert test suite. Each experiment was repeated with 5 different seeds.

Main training. We run three benchmarks after training has converged, and we compute: (i) the success rate of a training evaluation benchmark where we test the agent using scenarios in the last phase of curriculum; (ii) the success rate of the automatic quantitative evaluation; and (iii) the completion ratio of the expert-authored test suite. As Table 1 (left) shows, our agent outperforms or is on par with the built-in AI in all tests. Our agent achieves higher success rates in the training scenarios. Additionally, our agent surpasses the built-in AI by almost 10 percentage points in success rate during the quantitative evaluation. In the expert test suite, our agent has similar but lower performance than

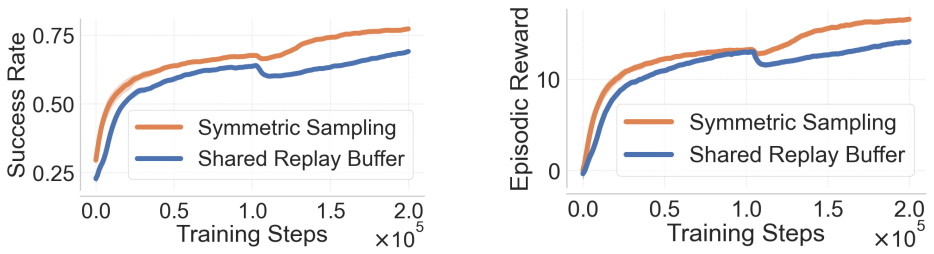

Figure 4: **The impact of symmetric sampling during fine-tuning**. We compare fine-tuning using the symmetric sampling outlined in Section 3.7 to a standard approach adding new data to the original buffer. **Left**: the success rate achieved by both agents in the failed scenario. **Right**: the episodic reward. Without symmetric sampling, the agent is not able to reach the same performance of our method in the same number of training steps.

the built-in AI. This is expected, as most cases in the benchmark were hand-designed with the built-in AI system in mind. However, it is not trivial to tune a machine learning agent to pass these tests. These experiments demonstrate that our reward function produces agents with similar qualitative behavior to the built-in AI but with higher performance. This meets the requirements of developing a game AI that outperforms traditional methods while exhibiting human-like behavior. We want to emphasize that (ii) is the *most meaningful evaluation* because (i) is biased towards our agent, as it is trained to maximize the reward in those scenarios, and (iii) is biased towards the built-in AI, since the tests were originally made for the built-in AI. In contrast, evaluation (ii) is completely independent of both the training and the development of the built-in AI.

Fine-tuning. We deploy our fine-tuning approach on three failed scenarios. Among the failed scenarios, we select three that were especially challenging for the goalkeeper. We first identify the best seed in terms of success rate of the quantitative evaluation resulted from main training process. Then, we evaluate it in the last phase of curriculum and the three failed scenarios. We then sequentially fine-tune the agent on these failed situations. After each step, we evaluate the agent on each of the three failed scenarios as well as the last curriculum phase, collecting the success rate over 500 episodes. As Table 2 shows, our approach improves the overall performance in the failed scenarios with fewer resources than the main training. The last agent performs better on average over all scenarios than all other agents. For each iteration, we perform 200,000 training steps compared to the 600,000 steps in the main training. However, after each iteration, the performance in previously failed scenarios degrades slightly. In fact, there is a risk the fine-tuned agent overspecializes to the scenario being used. If we, in one of the next iterations, use a failed scenario where the best performance conflicts with the behavior developed in previous iterations, we may observe a performance degradation. The degradation is not significant in our use case, as the average performance still improves.

4.2 Qualitative study

We gather feedback from playtesters and expert goalkeepers who evaluate the agent. Moreover, we qualitatively compare our agent's behavior to that of the built-in AI.

Human evaluation. We let an experienced human player compete against our agent and the built-in AI. In this evaluation, the human controls one team in a 7v7 scenario. The human player plays 400 games against either our agent or the built-in AI, with the agent type randomly selected at the beginning of each game, which is kept hidden from the player. Each game concludes when the ball goes out of bounds, the goalkeeper catches the ball, or a goal is scored. Table 1 (right) shows the agents' goal conceding rate and the percentage of saves.

Our agent demonstrates higher performance than the built-in AI, even against an experienced human player. An interesting finding is the percentage of saves: our agent is significantly more capable of saving and catching shots from the human player. The human playtester states: *"The movement of the goalkeeper is much more realistic and, from an end-user point of view, is much more enjoyable to play against/with. The goalkeeper is much more reliable, and when one manages to score, it is a very rewarding feeling."* Furthermore, their personal favorite aspect of the agent is during *"one versus one*

situations or passes across the box, where these chances are no longer guaranteed goals." Finally, they note, *"How the goalkeeper approaches these scenarios is perfect, only jumping on the ball when it is loose from the attacker's feet."*

Behavior analysis. Figure 1 compares the behavior of our agent and the built-in AI in one scenario. The figure shows how our agent better understands the situation, being more proactive and trying to anticipate the strikers before they shoot. This is a typical human-like behavior one can see in real football matches. In contrast, the built-in AI is more passive and waits for strikers, allowing them to score goals. We present an extended version in Appendix B.

4.3 Ablation studies

To evaluate the efficiency of our method compared to standard approaches, we define several ablated versions.

Main training. We analyze the impact of each component detailed in Section 3 by removing them from our method. Figure 3 shows the results, highlighting the performance of: (i) our approach, (ii) our approach without high replay ratio and resets, (iii) without leveraging offline data, (iv) without curriculum learning using only the final phase of the curriculum, (v) without offline steps during resets, and (vi) using standard SAC. The results demonstrate that the component removed in (ii) is fundamental to the agent's performance. Incorporating resets increases the plasticity of the networks and prevents them from becoming stuck in local optima [D'Oro et al., 2022]. Additionally, as shown in the training curves in Figure 3, offline data (iii) significantly improves efficiency, particularly in the initial phases of training. While (iii) achieves similar performance asymptotically, the performance gap at the beginning is evident. The figure shows that variation (iv) performs similarly to our agent. However, our method reaches the highest performance peak more quickly and is more stable: the standard deviation of (iv) is higher than that of (i). This can be attributed to the inclusion of all easier scenarios from previous phases in the final phase of the curriculum used by (iv). Without proper complexity scaling, training becomes noisy, increasing result variability and reducing stability. The same applies for (v): while the average performance is similar to our agent, without offline steps the training is more unstable. The difference can be seen in the figure: while (v) can sometimes outperform (i), the standard deviation of (v) is a sign of reduced training stability. To improve stability, it therefore makes sense to introduce offline steps.

Fine-tuning. We analyze the impact of the fine-tuning approach described in Section 3.7 by removing the symmetric sampling. For this ablation we repeat the fine-tuning experiment described in Section 4.1, but only for the failed scenario 1. We continue training the main agent using only the failed scenario, but adding the experience to the same buffer created during the main training D_{π_0}. We do not use symmetric sampling, but we sample randomly from the shared buffer. We start with randomly initialized policy and action-value function networks, and we keep the same reset scheduler. We train the agents for 200,000 steps, as described in Section 4.1. Figure 4 shows that the agent with symmetric sampling outperforms the ablated agent in both performance and training speed.

5 Limitations and future work

Although the proposed fine-tuning method enables simple adjustments to the policy, our evaluation shows that repeated fine-tuning leads to a loss of performance on earlier scenarios. This could be caused not only by the loss of plasticity, but also by catastrophic forgetting [Dohare et al., 2024].

In DRL for games, it is common to train agents against an already existing built-in AI or with self-play. However, to cover the range of situations that can occur in matches with humans, we believe training with human data could bring many advantages. However, it is unclear how to separate and leverage data coming from different types of players, and how to successfully learn from this data.

In settings such as robotics, autonomous driving, and video games, the qualitative behavior is often more important than quantitative performance. Our evaluations show that our agent exhibits noisier behavior compared to the built-in AI. Although this issue does not affect the final performance of our agent, we believe there is room for improvement to train smoother policies.

6 Conclusions

We presented a sample-efficient DRL method able to train human-like AI. Our method trains agents more than 50% faster than using standard DRL algorithms. Our ablations show the contribution of our method to the general sample efficiency landscape in DRL. To prove the practicality of the method, we tested it in *EA SPORTS FC 25*, a commercial football video game. By leveraging and improving on the latest advancements in sample-efficient DRL, our method can train well-performing agents in less than one day. Moreover, our approach allows game developers to adjust the policy's performance in case of bad behaviors without retraining it from scratch. Our agent outperforms the existing built-in AI in both quantitative and qualitative terms.

References

Philip J Ball, Laura Smith, Ilya Kostrikov, and Sergey Levine. Efficient online reinforcement learning with offline data. In *International Conference on Machine Learning*, pages 1577–1594. PMLR, 2023.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Debiak, Christy Dennison, David Farhi, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint 1912.06680*, 2019.

Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de Las Casas, et al. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897):414–419, 2022.

Shibhansh Dohare, J Fernando Hernandez-Garcia, Qingfeng Lan, Parash Rahman, A Rupam Mahmood, and Richard S Sutton. Loss of plasticity in deep continual learning. *Nature*, 632(8026): 768–774, 2024.

Pierluca D'Oro, Max Schwarzer, Evgenii Nikishin, Pierre-Luc Bacon, Marc G Bellemare, and Aaron Courville. Sample-efficient reinforcement learning by breaking the replay ratio barrier. In *Deep Reinforcement Learning Workshop NeurIPS 2022*, 2022.

Scott Fujimoto, Wei-Di Chang, Edward Smith, Shixiang Shane Gu, Doina Precup, and David Meger. For sale: State-action representation learning for deep reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024a.

Scott Fujimoto, Wei-Di Chang, Edward Smith, Shixiang Shane Gu, Doina Precup, and David Meger. For sale: State-action representation learning for deep reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024b.

Jonas Gillberg, Joakim Bergdahl, Alessandro Sestini, Andrew Eakins, and Linus Gisslén. Technical challenges of deploying reinforcement learning agents for game testing in AAA games. In *2023 IEEE Conference on Games (CoG)*, pages 1–8. IEEE, 2023.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

Mikhail Jacob, Sam Devlin, and Katja Hofmann. "it's unwieldy and it takes a lot of time"—challenges and opportunities for creating agents in commercial games. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, pages 88–94, 2020.

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

Girolamo Macaluso, Alessandro Sestini, and Andrew D Bagdanov. Small dataset, big gains: Enhancing reinforcement learning by offline pre-training with model-based augmentation. In *Computer Sciences & Mathematics Forum*, page 4. MDPI, 2024.

Daniel J Mankowitz, Andrea Michi, Anton Zhernov, Marco Gelmi, Marco Selvi, Cosmin Paduraru, Edouard Leurent, Shariq Iqbal, Jean-Baptiste Lespiau, Alex Ahern, et al. Faster sorting algorithms discovered using deep reinforcement learning. *Nature*, 618(7964):257–263, 2023.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

Max Schwarzer, Ankesh Anand, Rishab Goel, R Devon Hjelm, Aaron Courville, and Philip Bachman. Data-efficient reinforcement learning with self-predictive representations. *arXiv preprint arXiv:2007.05929*, 2020.

Max Schwarzer, Johan Samir Obando Ceron, Aaron Courville, Marc G Bellemare, Rishabh Agarwal, and Pablo Samuel Castro. Bigger, better, faster: Human-level atari with human-level efficiency. In *International Conference on Machine Learning*, pages 30365–30380. PMLR, 2023.

Alessandro Sestini, Alexander Kuhnle, and Andrew D Bagdanov. Deepcrawl: Deep reinforcement learning for turn-based strategy games. *arXiv preprint arXiv:2012.01914*, 2020.

Carmelo Sferrazza, Younggyo Seo, Hao Liu, Youngwoon Lee, and Pieter Abbeel. The power of the senses: Generalizable manipulation from vision and touch through masked multimodal learning. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9698–9705. IEEE, 2024.

Dhruva Tirumala, Thomas Lampe, Jose Enrique Chen, Tuomas Haarnoja, Sandy Huang, Guy Lever, Ben Moran, Tim Hertweck, Leonard Hasenclever, Martin Riedmiller, et al. Replay across experiments: A natural extension of off-policy rl. *arXiv preprint arXiv:2311.15951*, 2023.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 2019.

Shengjie Wang, Shaohuai Liu, Weirui Ye, Jiacheng You, and Yang Gao. Efficientzero v2: Mastering discrete and continuous control with limited data. *arXiv preprint arXiv:2403.00564*, 2024.

Peter R Wurman, Samuel Barrett, Kenta Kawamoto, James MacGlashan, Kaushik Subramanian, Thomas J Walsh, Roberto Capobianco, et al. Outracing champion gran turismo drivers with deep reinforcement learning. *Nature*, 2022.

Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto. Mastering visual continuous control: Improved data-augmented reinforcement learning. *arXiv preprint arXiv:2107.09645*, 2021.

Chen Zhang, Qiang He, Zhou Yuan, Elvis S Liu, Hong Wang, Jian Zhao, and Yang Wang. Advancing DRL agents in commercial fighting games: Training, integration, and agent-human alignment. *arXiv preprint arXiv:2406.01103*, 2024a.

Chen Zhang, Huan Hu, Yuan Zhou, Qiyang Cao, Ruochen Liu, Wenya Wei, and Elvis S. Liu. Training interactive agent in large FPS game map with rule-enhanced reinforcement learning. In *2024 IEEE Conference on Games (CoG)*. IEEE, 2024b.

A Soft Actor-Critic

We consider a discrete-time Markov Decision Process (MDP), which consists of a tuple $\langle S, A, R, P, \gamma \rangle$. All the elements in the MDP are defined in Section 3.1. Soft Actor-Critic (SAC) focuses on the maximum entropy reinforcement learning setting, where the agent's objective is to find the optimal policy π^* which maximizes the expected cumulative reward while keeping the entropy $\mathcal{H}$ of the policy distribution high:

$$J = \arg\max_{\pi^*} \mathbb{E}_{s_0 \sim P} \left[\sum_{t=0}^{\infty} \gamma^t (r_t - \alpha \mathcal{H}(\pi(\cdot\,, s_t))) \right], \tag{1}$$

where α is an hyper-parameter. The action value function is defined by:

$$Q(s, a) = \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t (r_t - \alpha \log(\pi(a_t|s_t))) \mid s_0 = s, a_0 = a \right]. \tag{2}$$

$Q(s, a)$ describes the expected future discounted reward gained by taking action a_t at a particular state s_t at timestep t. In particular, SAC parametrizes the action value function and policy as neural networks and trains two independent versions of the Q function, using the minimum of their estimates to compute the regression targets for Temporal Difference (TD) learning. The optimization objective for the Q functions is:

$$L_Q = \mathbb{E}_{(s,a,r,s',d) \sim D}[(Q_i(s, a) - y(r, s', d))^2], \tag{3}$$

where D is the dataset, d is a value indicating whether the episode is terminated, Q_i is the action value function with $i = 1, 2$, and y is defined as the target value:

$$y(r, s', d) = r + \gamma(1 - d)(\min_{i=1,2} Q_{\text{target}_i}(s', a) - \alpha \log \pi(\hat{a}', s')), \quad \hat{a}' \sim \pi(\cdot\,, s'), \tag{4}$$

where Q_{target_i} is the target action value function, a copy of Q_i initialized with the same initial weights as Q_i. Finally, the policy is optimized accordingly to:

$$L_\pi = \max \mathbb{E}[\min_{i=1,2} Q_i(s, \hat{a}) - \alpha \log \pi(\hat{a}, s)], \quad \hat{a} \sim \pi(\cdot, s). \tag{5}$$

After each iteration, SAC updates the weights of the target value functions through a soft update:

$$\phi_{\text{target}_i} \leftarrow \rho \phi_{\text{target}_i} - (1 - \rho)\phi_i, \tag{6}$$

where ϕ_{target_i} is the set of weights for Q_{target_i}, ϕ_i is the set of weights for Q_i, and ρ is a hyper-parameter.

B Additional Qualitative Behavior Analysis

In this section we expand the qualitative behavior analysis outlined in Section 4.2. Figure 5, Figure 6, Figure 7, and Figure 8 show different scenarios comparing the qualitative behavior of our agent compared to the built-in AI. The results demonstrate, across all the scenarios shown, that our agent performs better than the built-in AI mainly because it exhibits a more human-like behavior than the hand-crafted AI. In Figure 5 and Figure 7, the agent exhibits a more proactive behavior, similar to that of a human goalkeeper. According to a professional goalkeeper, in order to increase the probability of catching the ball in a 1v1 situation, a real human goalkeeper should "close the space" of the striker. Figure 6 shows the agent achieve better positioning by covering the middle-line – the line between the ball and the center of the goal – successfully saving a shot from the distance. Instead, the built-in AI leaves space for the striker to shoot to the "far post." In Figure 8, the starting position of the agent is very similar to the one of the built-in AI, but it is different enough so that our agent can catch the ball, while the built-in AI can not.

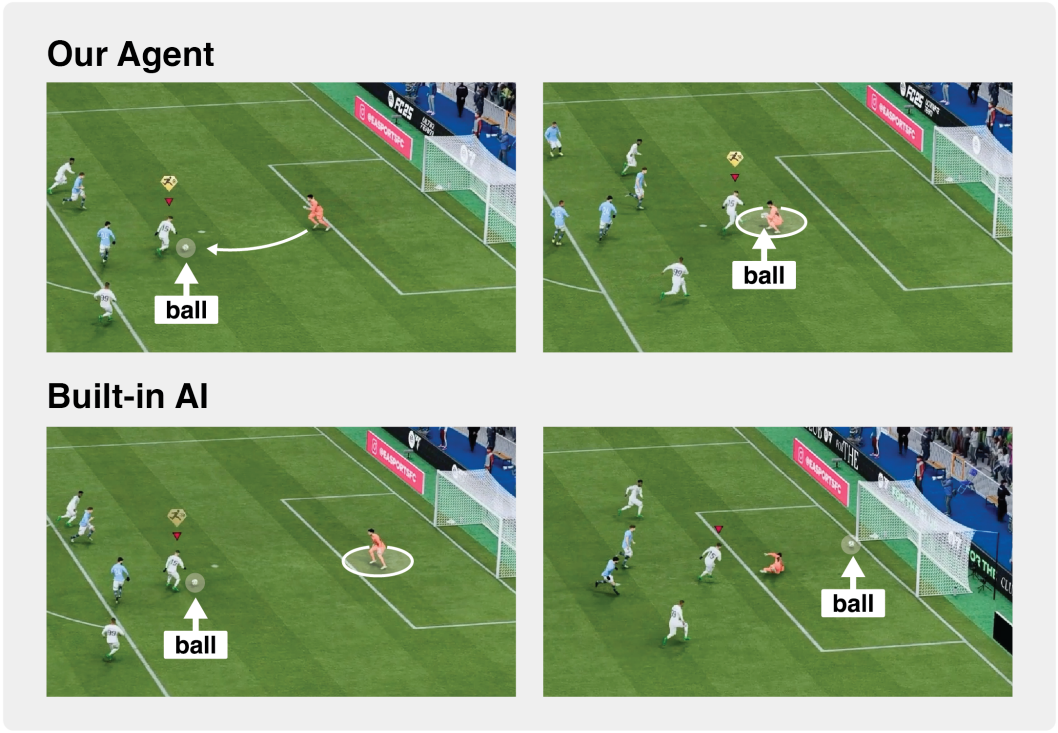

Figure 5: **An example showcasing the behavioral differences** between our agent (top) and the built-in AI (bottom) in the same situation. Our agent is more proactive and better understands the current situation, anticipating the shot and moving forward early. In contrast, the built-in AI is more passive, allowing the striker to score a goal by leaving the goal cage open. We report a quote from a professional goalkeeper: *"the goalkeeper plays it really well! Keeper looks for opportunities to steal ground as the striker enters the box."*

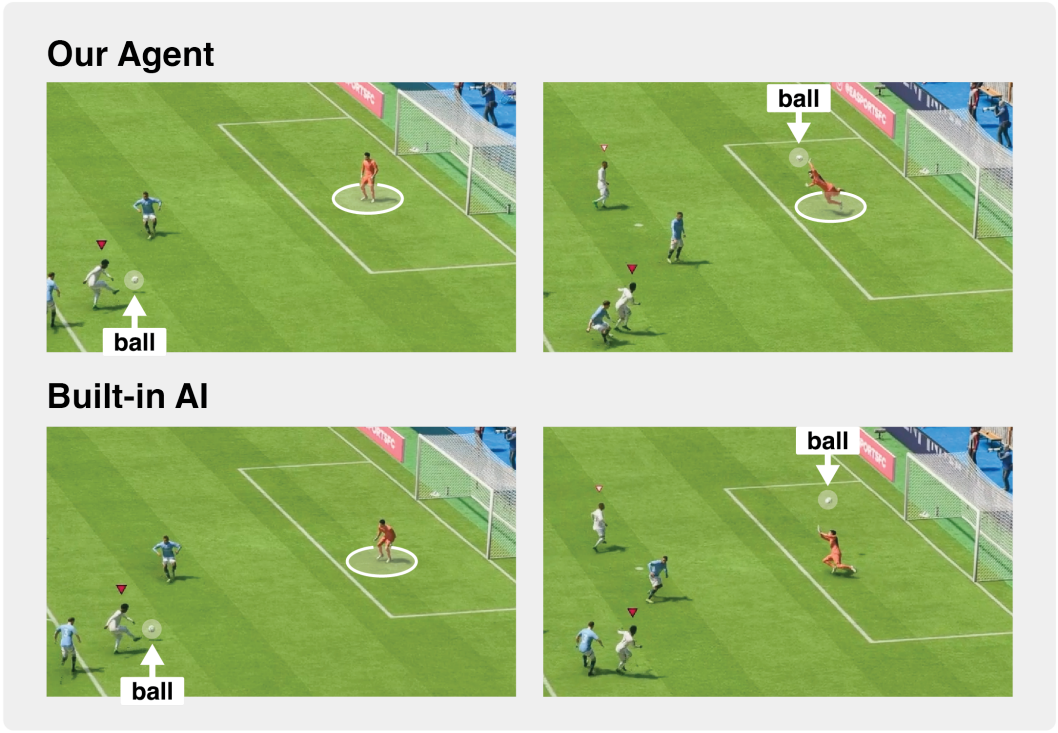

Figure 6: **An example showcasing the behavioral differences** between our agent (top) and the built-in AI (bottom) in the same situation. Our agent better covers the middle-line – the line between the ball and the center of the goal – being able to save the distant shot from the striker. The built-in AI is less patient, covering the first post – the post closer to the ball – leaving enough space for the striker to score. We report a quote from a professional goalkeeper: "*Good positioning by the goalkeeper, and the depth is good*."

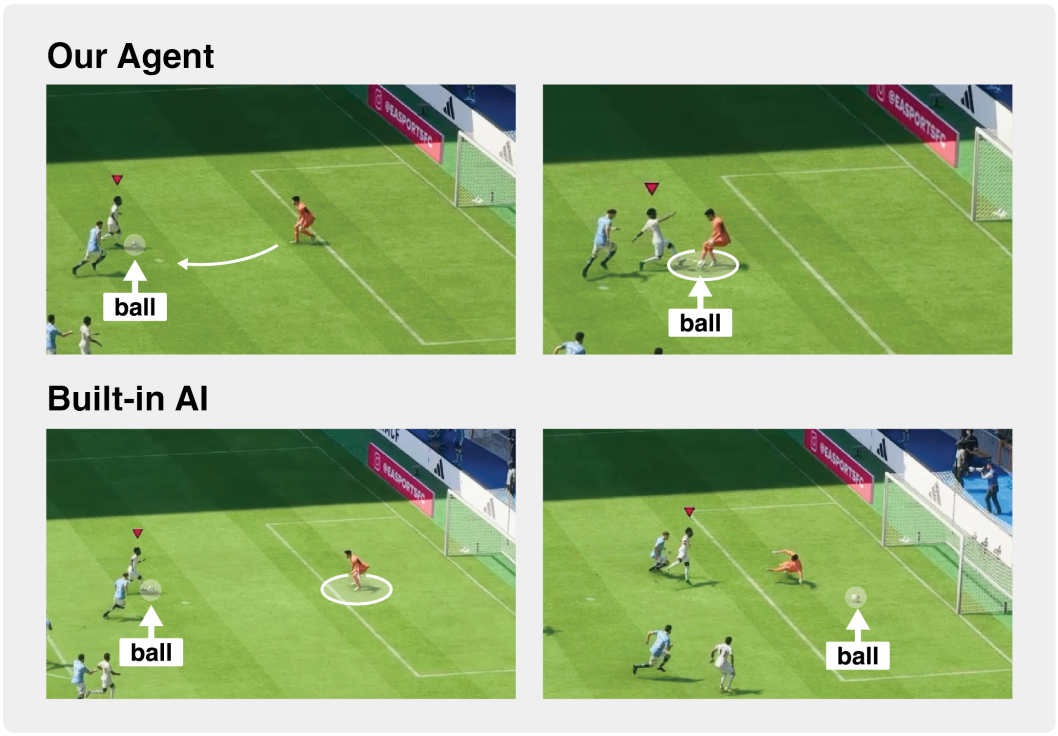

Figure 7: **An example showcasing the behavioral differences** between our agent (top) and the built-in AI (bottom) in the same situation. Our agent understands that in particular 1v1 situations, in order to increase the probability of saving the ball, it needs to "close the space" of the striker. Instead, the built-in AI always use the same passive behavior, giving more chances to the striker.

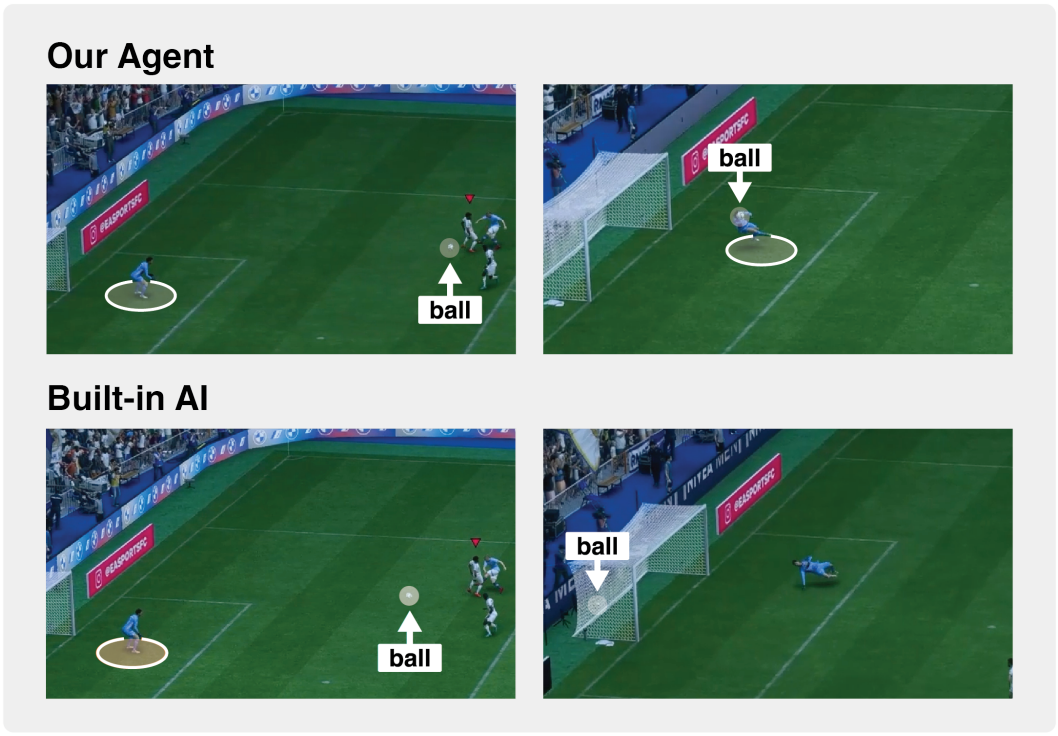

Figure 8: **An example showcasing the behavioral differences** between our agent (top) and the built-in AI (bottom) in the same situation. The starting position of the agent is very similar to the one of the built-in AI, but it is different enough so that our agent can catch the ball, while the built-in AI can not.

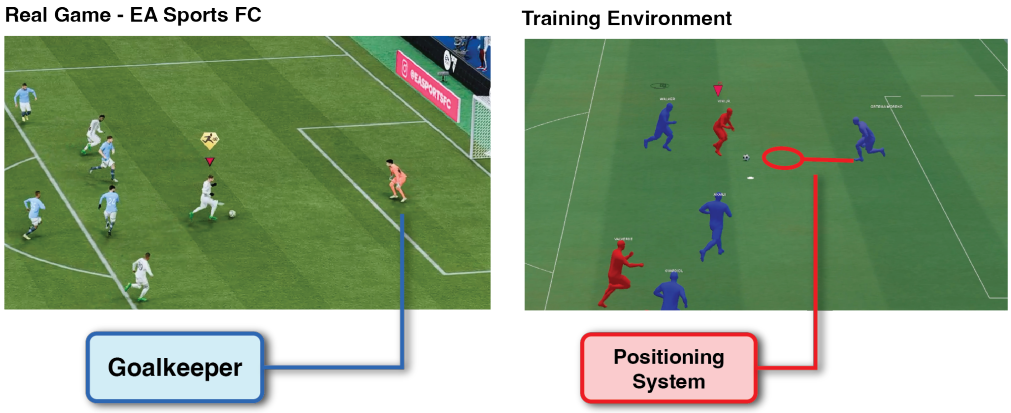

Figure 9: **Overview of EA SPORTS FC 25 and the goalkeeper positioning system**. **Left**: screenshot of the real game, **Right**: screenshot of the training environment. The training environment has the exact same gameplay mechanics (logic, physics, etc.) as the real game, but it lacks all the graphical enhancements. The goalkeeper positioning system outputs a target position, shown by the red line and circle in the right image, and the desired intensity of the movement.

C Action and state spaces

The action space of the agent consists of three continuous values: two values for the relative XZ components of the movement vector, and one value for the intensity of the movement. The latter controls how fast the agent should reach the position defined by the movement vector.

A variety of state features are input to the neural networks. We use the same state space for both the agent and action-value functions. These features are directly available from the game engine. Each of the features is normalized to the $[-1, 1]$ range. We divide the state space into three main components, for a total of 109 features. The first component comprises the set of goalkeeper egocentric information, such as position and velocities; the second component includes a set of opponent features; and the third includes a set of features for the goalkeeper's teammates.

D Reward Function

The reward function is hand-crafted for achieving a human-like behavior. Several aspects of the reward function were directly influenced by feedback from a professional goalkeeper who evaluated the agent during initial development. Extra attention was given to complex scenarios such as lobs and many-vs-many situations. It is composed of three reward components: a sparse reward for catching or deflecting the ball, a dense reward for encouraging the player to cover most of the area delimited by the goal, and penalties to incentivize smooth movements.

We argue that leveraging domain expertise is fundamental for developing a DRL agent that exhibits good qualitative behavior suitable for complex practical settings, such as video games. For instance, in the initial iterations, it was relatively easy to train an agent capable of beating the built-in AI, but it exhibited noisy behaviors that would detract from its credibility and the ability to be fun to play against. With the help of domain expertise, we realized that smoother behavior is more important than pure performance.

E Training Setup and Hyper-parameters

All training was performed deploying 4 parallel environments on the same machine with an NVIDIA A6000 GPU with 48GB RAM and a AMD Ryzen Threadripper PRO 7975WX 32-Core CPU. As mentioned in Section 3.2, we train our agent using a low-resolution version of the game, in which we can unlock the frame rate allowing us to speed-up the simulation by a factor of 3. Figure 9 shows the difference between the real game and the low-resolution version.

E.1 Learning Architecture Details

As previously mentioned, we use Soft Actor-Critic (SAC) [Haarnoja et al., 2018] as the optimization algorithm. Standard SAC requires a policy network and two identical action-value function networks. We use the same architecture for both the policy and action-value function networks, but we initialize them independently. In order to satisfy the runtime requirements for running the policy network in a game at 60 FPS, we constrain the network's size. Both the policy and the action-value functions are represented by 5-layer MLPs, all layers with a size of 256, ReLU activations, and layer normalization. The last layer of the policy outputs the mean and standard deviation of a diagonal Gaussian distribution for each of the three actions. The action-value functions have a last layer of size 1 without activation, returning the estimated future discounted reward.

E.2 Curriculum Learning

We let C^i represent the collection of scenarios in phase i. To mitigate the risk of catastrophic forgetting, we reuse scenarios encountered in previous phases such that $C^i \cap C^{i-1} \neq \emptyset, \ \forall \ 2 \leq i \leq N$, where N is the number of phases in total. We move from C^i to C^{i+1} when the agent has reached an average success rate in all scenarios of C^i. The scenarios retained from the previous phase and their number, as well as the success rate threshold for each phase, are manually determined.

Figures from 10 to 15 show examples of scenarios used during curriculum learning. Figures 10 and 11 illustrate scenarios from phase 1, Figures 12 and 13 from phase 2, and Figures 14 and 15 from phase 3. More details of the individual scenarios are provided in the captions. Figures 10 and 12 show examples of scenarios that are kept from phase 1 to phase 2 and from phase 2 to phase 3.

Figure 10: **Example of scenario within phase 1.** In this scenario, the agent (blue) is tasked to catch a ball that is slowly rolling away from the goal cage. There is also a single striker (red) who does not act. This simple situation helps the agent learn what it means to catch the ball at the beginning of the training.

Figure 11: **Example of scenario within phase 1.** In this scenario, we task the agent (blue) to learn a common situation in football: playing 1v1 with a striker (red). Although it may seem simple, this scenario contains hidden complexities depending on the expertise of the striker and the starting positions of players. For this phase, we use a naive striker who moves toward the goal cage and shoots when it sees an opportunity. We maintain this scenario in subsequent phases of curriculum learning. Additionally, more complex versions of this scenario are added in later phases.

Figure 12: **Example of scenario within phase 2.** In this scenario, we show an evolution of the situation shown in Figure 10: a 2v1 scenario. Two strikers (red) face off against the agent (blue). This situation is more complex than the 1v1 scenario because the agent needs to decide whether to rush out towards the strikers if it perceives shooting intentions, or to wait for a pass between the strikers.

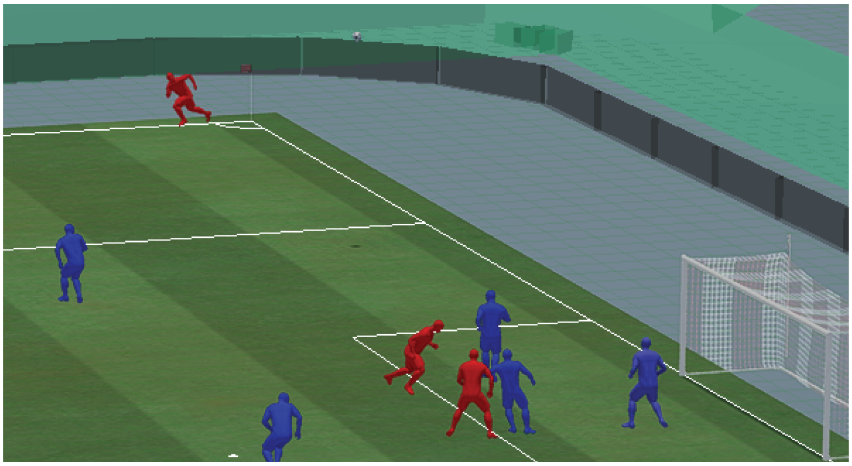

Figure 13: **Example of scenario within phase 2.** In this scenario, we present a common situation in football: a corner kick. The red players are the agent's opponents, while the blue players are the agent's teammates. In this situation, the agent must decide whether to rush out and reach the ball before the other strikers, risking a shot; or to wait and allow the striker to shoot, aiming for a safe catch. This situation shows an example scenario that we keep in phase 3.

Figure 14: **Example of scenario within phase 3.** The figure shows a general situation: a 7v7 match. In this scenario, all players, including strikers and defenders, do not follow any specific behaviors. We can see a striker (in red), while the blue team represents the agent's teammates. The scenario starts in the agent's half of the pitch and ends if the ball goes out of bounds, the goalkeeper catches the ball, or after a maximum number of steps. We have different versions of this scenario that vary the difficulty level of the strikers AI.

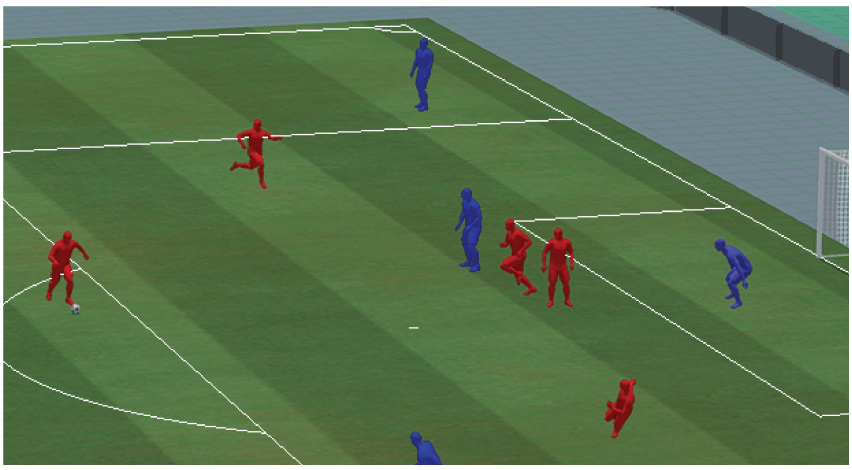

Figure 15: **Example of scenario within phase 3.** The figure shows one of the most complex scenarios in all of the curriculum phases. In this situation, 7 strikers (red) with the highest skill-level of AI face 7 defenders (blue) with the lowest skill-level of AI. The goalkeeper cannot rely on its teammates and must correctly understand the situation to find the best position to decrease the probability of a goal.

E.3 Hyper-parameters

Table 3 describes the set of hyper-parameters used in our experiments.

Parameter	**Value**
Online batch size	512
Offline batch size	512
Buffer size	10^7
Action repetitions	5
Max timesteps	300
Discount γ	0.997
Optimizer	Adam
Learning rate	5×10^{-4}
β_1	0.9
β_2	0.999
Replay ratio	1
Number of Curriculum Phases N	3
Reset interval (gradient steps)	$100,000$
Offline steps	$6,400$
Random initial actions	$25,000$
Episode of demonstrations for each curriculum phase	$1,000$
Success rate threshold for each curriculum phase	$[0.90, 0.90]$
Scenarios in each curriculum phase	$[11, 18, 25]$
Q networks depth	5 layers
Q networks width	256
Q networks layer normalization	True
π network depth	5 layers
π network width	256
π network layer normalization	True

Table 3: **Hyper-parameters.** The most important hyper-parameters of our approach and their respective values.

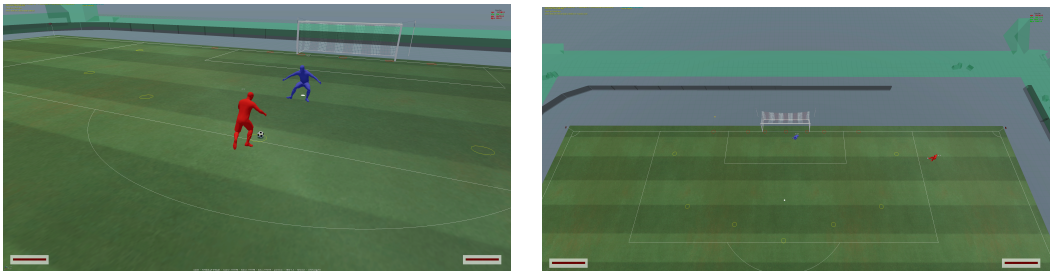

Figure 16: **Screenshots of the automatic quantitative evaluation**. In the automatic quantitative evaluation, the agent (blue) faces a striker (red) for 2,000 shots. **Left**: a screenshot of the test running. **Right**: a top-down view of the test. The figure shows the possible starting position of the striker in yellow, and the possible shooting target in red. For every shot, the striker will randomize the power, the type, and the target of the shot.

F Evaluation Framework

In this section we describe more details regarding the framework detailed in Section 3.6. Moreover, we show some screenshots of the scenarios used in the benchmarks.

F.1 Automatic Quantitative Evaluation

Figure 16 shows the scenario used for the automatic quantitative evaluation. It is a simple scenario where the agent faces an opponent that shoots towards the goal for 2,000 steps, using different types of shots with varying levels of difficulty. The possible starting position of the striker is highlighted in yellow, while the possible shooting targets are highlighted in red.

F.2 Expert-authored Test Suite

Figures 17 to 20 show examples of tests and their completion condition used in the expert-authored test suite. We use a total of 344 tests for evaluating the agent, with different completion conditions.

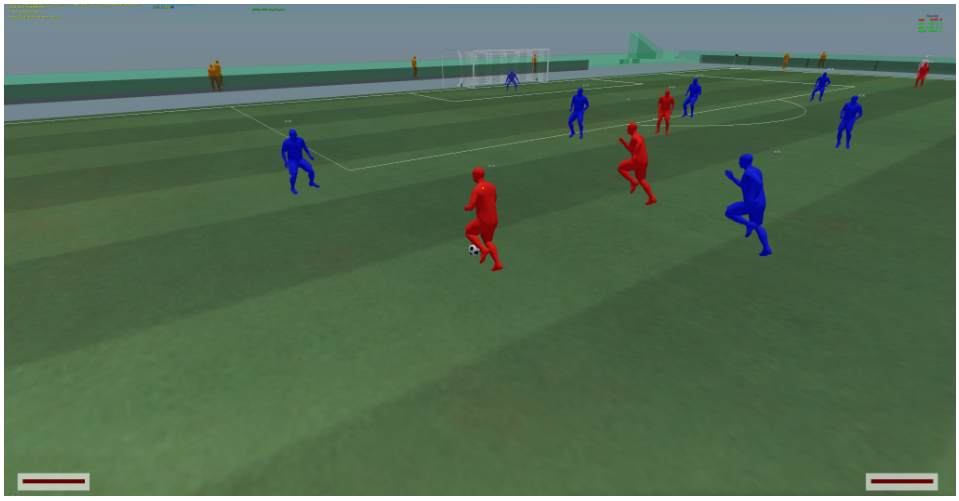

Figure 17: **Example of a test in the expert-authored test suite.** In this test, the striker (in red, in the foreground) is threatening the right post, and human goalkeepers should cover it as fast as they can. For this reason, in this situation the test checks whether the agent is moving towards the right post.

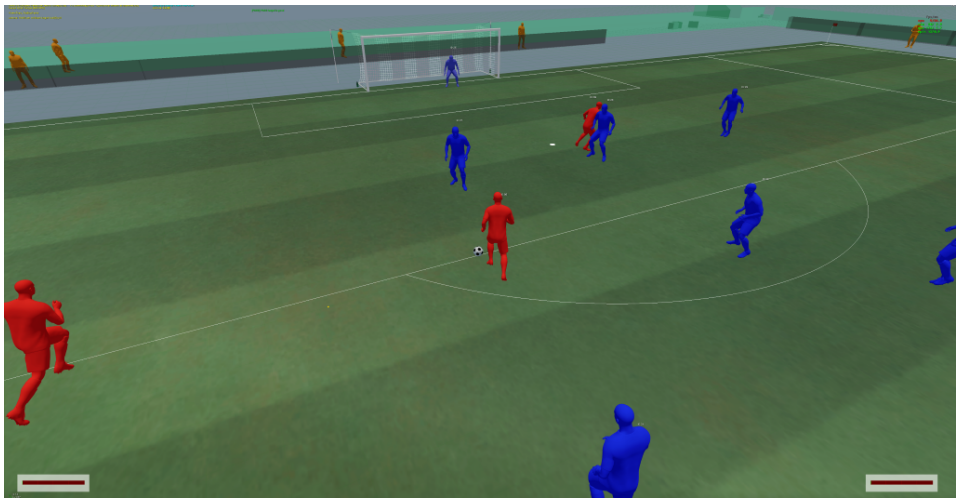

Figure 18: **Example of a test in the automatic expert-authored test suite.** In this test, the striker (in red, in the foreground) is threatening the center position of the goal net, and goalkeeper is very close to the goal line. In this type of situations, the goalkeeper should move forward. For this reason, in this situation the test checks whether the agent is moving towards the red striker.

Figure 19: **Example of a test in the automatic expert-authored test suite.** In this test, the striker (in red) and the goalkeeper (in blue) are far from the ball, that is moving towards the goal line. It is possible for the human goalkeeper to catch the ball before the striker, thus this test checks if the agent anticipates the striker.

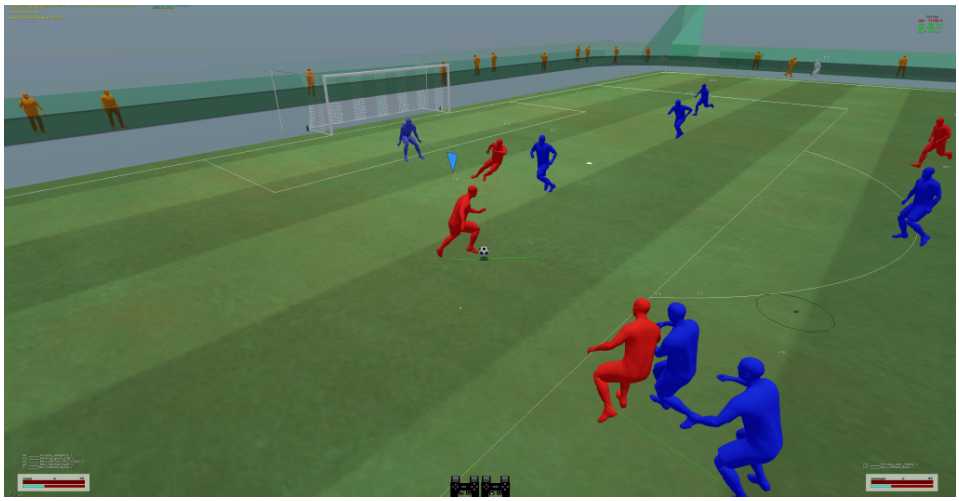

Figure 20: **Example of a test in the expert-authored test suite.** In this test, the goalkeeper is far from the goal net and out of standard position, leaving time and space for the striker to score. Therefore, this test checks if the goalkeeper is going towards the goal net repositioning itself in the standard position.

G Additional Results in MuJoCo

We compare our method with the standard SAC algorithm in the MuJoCo suite. To demonstrate the sample efficiency of our approach, we limit our budget to 100K environment steps (referred to as MuJoCo 100K, similar to Fujimoto et al. [2024b]). We use D4RL *medium* datasets [Fujimoto et al., 2024a] to simulate sub-optimal samples as our offline datasets and we do not use curriculum learning. The tasks we consider are: *Hopper*, *HalfCheetah*, *Ant*, and *Humanoid*. Table 4 shows the results. Our approach achieves better results than the standard SAC in all tasks except the Humanoid task. The latter is a complex task that likely requires more samples, both offline and online, to achieve good results.

Task	**SAC**	**Our Method**
Hopper	1830.02 ± 479.80	$\mathbf{1911.12 \pm 409.40}$
HalfCheetah	1401.39 ± 140.15	$\mathbf{4552.16 \pm 129.32}$
Ant	1253.02 ± 36.27	$\mathbf{1767.12 \pm 425.99}$
Humanoid	$\mathbf{2059.02 \pm 555.02}$	604.83 ± 48.03

Table 4: **Additional Results on MuJoCo 100K.** For this experiment, we limit our budget to 100K samples, similar to Fujimoto et al. [2024b]. We compare our method with the standard SAC algorithm. Our method achieves better results in all tasks except Humanoid.