
Reinforcement Learning for Multi-Agent Soccer

Vincent Hu

MSc Visual Computing
Simon Fraser University
Canada
jiaqingh@sfu.ca

Zhuo Ning

MSc Visual Computing
Simon Fraser University
Canada
zna3@sfu.ca

Adam Brykajlo

MSc Visual Computing
Simon Fraser University
Canada
abrykajl@sfu.ca

Murali Krishtna J

MSc Visual Computing
Simon Fraser University
Canada
mkj11@sfu.ca

1 Introduction

Multi-agent reinforcement learning (MARL) presents unique challenges in competitive environments such as soccer, where agents must learn complex coordination strategies while competing against opponents [5, 1]. This paper presents a comparative study of three reinforcement learning algorithms—Proximal Policy Optimization (PPO) [9], Soft Actor-Critic (SAC) [4], and Policy Optimization with Competitive Agents (POCA) [2]—applied to a two-versus-two soccer environment. We analyze the reward structures and training dynamics of each policy, providing detailed formulations of the reward functions that guide agent behavior. Notably, PPO and POCA use a minimal reward structure consisting only of goal rewards and existential penalties, while SAC employs an extended reward function with continuous shaped rewards to facilitate off-policy learning. Our experiments are conducted using the Unity ML-Agents framework [6], which provides a robust platform for training intelligent agents in complex multi-agent environments. Related work in multi-agent football has demonstrated the effectiveness of curriculum learning and self-play [7], techniques that we also employ in our training methodology.

2 Related Work

The application of Deep Reinforcement Learning to competitive multi-agent environments, such as robotic soccer, necessitates navigating complex challenges regarding coordination, credit assignment, and sample efficiency. Recent literature has focused on adapting foundational single-agent algorithms for multi-agent contexts and developing specialized architectures to handle team-based dynamics.

Foundational Algorithms in Multi-Agent Settings Proximal Policy Optimization (PPO) has established itself as a dominant baseline in Multi-Agent Reinforcement Learning (MARL), despite originating as a single-agent algorithm [9]. Yu et al. demonstrated that Multi-Agent PPO (MAPPO) achieves strong performance across cooperative benchmarks like StarCraft, suggesting that implementation details such as value normalization often outweigh the need for specialized multi-agent architectures [10]. PPO stabilizes training via a clipped surrogate objective, which constrains policy updates to prevent destructive large steps, a feature critical for the high variance of multi-agent environments.

Soft Actor-Critic (SAC) represents the state-of-the-art in off-policy learning for continuous control. Operating within a maximum entropy framework, SAC optimizes a policy to maximize both expected return and entropy, encouraging diverse exploration and robustness to noise [4]. While SAC typically offers superior sample efficiency compared to on-policy methods by reusing past experiences, it can

be brittle with respect to hyperparameters in complex multi-agent scenarios. Implementations of SAC in soccer environments often require extended, dense reward shaping to facilitate effective training.

Multi-Agent Coordination and Credit Assignment A persistent challenge in cooperative MARL is the "credit assignment problem" (determining which agent actions contributed to a team's success). The Centralized Training Decentralized Execution (CTDE) paradigm addresses this by utilizing global information during training [8]. While approaches like COMA introduced counterfactual baselines to isolate agent contributions [3], they often struggle in environments where agents may terminate early or spawn mid-episode [2]. Traditional methods handle this via "absorbing states," where inactive agents process dummy inputs, wasting computational resources.

Cohen et al. introduced Multi-Agent POsthumous Credit Assignment (POCA). This architecture utilizes a self-attention mechanism in the centralized critic, allowing it to process a variable number of agents without requiring absorbing states [2]. POCA enables "posthumous credit assignment," allowing agents to learn from rewards achieved by the team after the agent's own termination, which is essential for learning cooperative behaviors like self-sacrifice.

Self-Play and Evaluation Metrics Mastering competitive games relies heavily on self-play, where agents improve by competing against past versions of themselves. This allows agents to learn without human demonstrations, with opponent difficulty naturally scaling alongside agent skill. To address non-transitivity in win rates (A beats B, B beats C, but C beats A), the ELO rating system maintains relative skill ratings based on match outcomes and opponent strength. Unity ML-Agents supports ELO-based matchmaking, allowing consistent performance tracking during training.[6].

3 Methods

3.1 Environment Setup

The soccer environment consists of two teams (Blue and Purple), each with two agents: a striker and a goalie. The objective is to score goals by moving the ball into the opposing team's goal while preventing the opponent from scoring. Agents receive individual rewards based on their actions and the game state, enabling decentralized learning.

3.2 Reward Formulation

The reward structure differs between on-policy (PPO, POCA) and off-policy (SAC) algorithms. PPO and POCA use a minimal reward structure with only goal rewards and existential penalties, while SAC employs an extended reward function with continuous shaped rewards to provide denser learning signals for off-policy training.

3.2.1 Base Reward Structure (PPO and POCA)

For PPO and POCA policies, the reward function consists of only two components:

$$R_t^{\text{PPO/POCA}} = R_{\text{existential}} + R_{\text{goal}} \quad (1)$$

3.2.2 Extended Reward Structure (SAC)

For SAC training, we augment the base rewards with continuous shaped rewards to provide denser learning signals:

$$R_t^{\text{SAC}} = R_{\text{existential}} + R_{\text{ball_approach}} + R_{\text{ball_progress}} + R_{\text{ball_touch}} + R_{\text{shoot}} + R_{\text{goal}} \quad (2)$$

The rationale for this difference is that off-policy algorithms like SAC benefit from denser reward signals, while on-policy methods (PPO, POCA) can learn effectively from sparse rewards. Each component is detailed below.

3.2.3 Existential Reward

The existential reward provides a baseline signal that varies by agent position:

$$R_{\text{existential}} = \begin{cases} \frac{1}{T_{\text{max}}} & \text{if position} = \text{Goalie} \\ -0.1 \times \frac{1}{T_{\text{max}}} & \text{if position} = \text{Striker} \end{cases} \quad (3)$$

where T_{max} is the maximum number of environment steps (25,000). This reward encourages goalies to maintain their position while slightly penalizing strikers to promote active engagement.

3.2.4 Ball Approach Reward (SAC)

To encourage agents to approach and control the ball, we provide shaped rewards based on distance reduction and orientation. This component is only used for SAC training:

$$R_{\text{ball_approach}} = R_{\text{distance}} + R_{\text{facing}} + R_{\text{proximity}} \quad (4)$$

The distance reward encourages getting closer to the ball:

$$R_{\text{distance}} = 0.05 \times (d_{t-1} - d_t) \quad (5)$$

where d_t is the distance between the agent and the ball at time t . The facing reward encourages the agent to orient toward the ball:

$$R_{\text{facing}} = 0.005 \times \max(0, \mathbf{v}_{\text{forward}} \cdot \mathbf{v}_{\text{to_ball}}) \quad (6)$$

where $\mathbf{v}_{\text{forward}}$ is the agent's forward direction vector and $\mathbf{v}_{\text{to_ball}}$ is the normalized vector from the agent to the ball. Additionally, a proximity bonus is given when the agent is within 3 units of the ball:

$$R_{\text{proximity}} = \begin{cases} 0.01 \times \frac{5-d_t}{5} & \text{if } d_t < 3 \\ 0 & \text{otherwise} \end{cases} \quad (7)$$

3.2.5 Ball Progress Reward (SAC)

This component rewards agents for pushing the ball toward the opposing goal while penalizing regression toward their own goal. This component is only used for SAC training:

$$R_{\text{ball_progress}} = R_{\text{push}} + R_{\text{control_bonus}} - R_{\text{regression}} + R_{\text{shoot_zone}} \quad (8)$$

The push reward is based on the ball's progress toward the opposing goal:

$$R_{\text{push}} = \begin{cases} 0.05 \times (d_{\text{opp},t-1} - d_{\text{opp},t}) & \text{if } d_{\text{opp},t-1} > d_{\text{opp},t} \\ 0 & \text{otherwise} \end{cases} \quad (9)$$

where $d_{\text{opp},t}$ is the distance from the ball to the opposing goal at time t . An additional control bonus is given when the agent is controlling the ball (within 3 units) while pushing:

$$R_{\text{control_bonus}} = \begin{cases} 1.5 \times R_{\text{push}} & \text{if } d_{\text{agent_ball}} < 3 \text{ and } R_{\text{push}} > 0 \\ 0 & \text{otherwise} \end{cases} \quad (10)$$

The regression penalty discourages the ball from moving toward the agent's own goal:

$$R_{\text{regression}} = \begin{cases} 0.03 \times (d_{\text{own},t-1} - d_{\text{own},t}) & \text{if } d_{\text{own},t-1} > d_{\text{own},t} \\ 0 & \text{otherwise} \end{cases} \quad (11)$$

where $d_{\text{own},t}$ is the distance from the ball to the agent's own goal. A shooting zone bonus encourages play near the opposing goal:

$$R_{\text{shoot_zone}} = \begin{cases} 0.01 \times \frac{5-d_{\text{opp},t}}{5} & \text{if } d_{\text{opp},t} < 5 \\ 0 & \text{otherwise} \end{cases} \quad (12)$$

3.2.6 Ball Touch Reward (SAC)

When an agent collides with the ball, it receives a touch reward. This component is only used for SAC training:

$$R_{\text{ball_touch}} = 0.2 \times c_{\text{touch}} \quad (13)$$

where c_{touch} is a curriculum parameter (default 1.0) that can be adjusted during training.

3.2.7 Shooting Reward (SAC)

When an agent kicks the ball within the shooting distance threshold (5 units) of the opposing goal, additional rewards are provided. This component is only used for SAC training:

$$R_{\text{shoot}} = R_{\text{shoot_base}} + R_{\text{shoot_direction}} \quad (14)$$

The base shooting reward is:

$$R_{\text{shoot_base}} = 0.5 \times \frac{5 - d_{\text{opp}}}{5} \quad (15)$$

An additional direction reward is given if the kick direction is aligned with the goal (alignment > 0.5):

$$R_{\text{shoot_direction}} = \begin{cases} 0.5 \times R_{\text{shoot_base}} & \text{if } \mathbf{v}_{\text{kick}} \cdot \mathbf{v}_{\text{to_goal}} > 0.5 \\ 0 & \text{otherwise} \end{cases} \quad (16)$$

3.2.8 Goal Reward

When a goal is scored, the winning team receives a time-dependent reward, while the losing team receives a penalty:

$$R_{\text{goal}} = \begin{cases} \max(1, 2 - \frac{t}{T_{\text{max}}}) & \text{if scoring team} \\ -1 & \text{if opposing team} \end{cases} \quad (17)$$

where t is the current time step. This encourages faster goal scoring.

3.3 Policy Algorithms

3.3.1 Proximal Policy Optimization (PPO)

PPO is an on-policy algorithm that uses clipped surrogate objectives to prevent large policy updates [9]. The objective function is:

$$J(\pi) = \mathbb{E}_{s \sim \rho^{\pi_{\text{old}}}, a \sim \pi_{\text{old}}} \left[\min \left(r(\theta) \hat{A}, \text{clip}(r(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A} \right) \right] \quad (18)$$

where $r(\theta) = \frac{\pi_{\theta}(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}$ is the probability ratio, \hat{A} is the estimated advantage, and ϵ is the clipping parameter. PPO uses only the base reward structure (existential penalty and goal reward) without continuous shaped rewards.

3.3.2 Soft Actor-Critic (SAC)

SAC is an off-policy actor-critic algorithm that uses a maximum entropy framework [4]. The objective function maximizes both expected return and entropy:

$$J(\pi) = \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^T \gamma^t (r(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot|s_t))) \right] \quad (19)$$

where α is the temperature parameter controlling the trade-off between exploration and exploitation, and \mathcal{H} is the entropy of the policy. SAC benefits from the extended reward structure with continuous shaped rewards, as off-policy algorithms require denser reward signals for effective learning. The reward signal uses $\gamma = 0.99$ and strength = 1.0.

3.3.3 POsthumous Credit Assignment (POCA)

POCA (POsthumous Credit Assignment) is designed for competitive multi-agent environments and uses a centralized training, decentralized execution (CTDE) approach [2]. We used a variant of the algorithm that supports multiple agents (MAPoCA), developed by Unity Technologies, effectively handles agent spawning and despawning without requiring absorbing states. The policy optimization follows:

$$J(\pi_i) = \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^T \gamma^t A_i(s_t, a_t) \right] \quad (20)$$

where A_i is the advantage function for agent i , computed using Generalized Advantage Estimation (GAE) with $\lambda = 0.95$. The algorithm uses PPO-style clipping with $\epsilon = 0.2$ and $\beta = 0.005$ for entropy regularization. Like PPO, POCA uses only the base reward structure (existential penalty and goal reward).

4 Experimental Setup

All experiments were conducted using the Unity ML-Agents framework [6], which provides a comprehensive toolkit for training intelligent agents in Unity environments. All policies were trained for 50 million steps using the Adam optimizer. The hyperparameters for each policy are as follows:

Table 1: Hyperparameter configurations for PPO, SAC, and POCA agents.

Parameter	PPO	SAC	POCA
<i>General Settings</i>			
Network Architecture	2 hidden layers, 512 units each		
Discount Factor (γ)	0.99	0.99	0.99
Time Horizon (steps)	1000	1000	1000
Reward Structure	Base ^a	Extended ^b	Base ^a
<i>Training Parameters</i>			
Learning Rate	3×10^{-4}	1×10^{-4}	3×10^{-4}
LR Schedule	Constant	Linear	Constant
Batch Size	2048	512	2048
Buffer Size	20,480	500,000	20,480
Buffer Init. Steps	—	50,000	—
Epochs per Update	3	—	3
<i>Algorithm Specifics</i>			
Clipping (ϵ)	0.2	—	0.2
GAE (λ)	0.95	—	0.95
Entropy Coeff.	$\beta = 0.01$	Init. = 0.5	$\beta = 0.005$
Soft Update (τ)	—	0.005	—

Note: Dashes (—) indicate the parameter is not applicable to the algorithm.

^a Existential penalty + Goal reward.

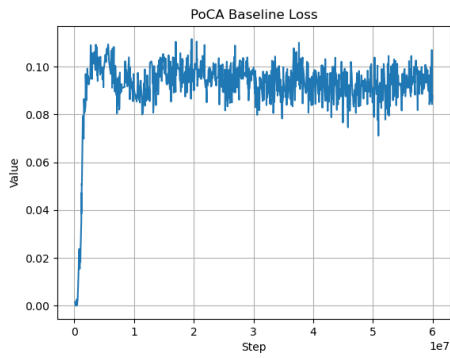
^b Existential penalty + Continuous shaped rewards + Goal reward.

All policies employed self-play mechanisms to improve training stability and agent competitiveness. The SAC policy used a self-play window of 20 models with team changes every 500,000 steps, while POCA used a window of 10 models with team changes every 200,000 steps. PPO typically uses similar self-play configurations for competitive environments.

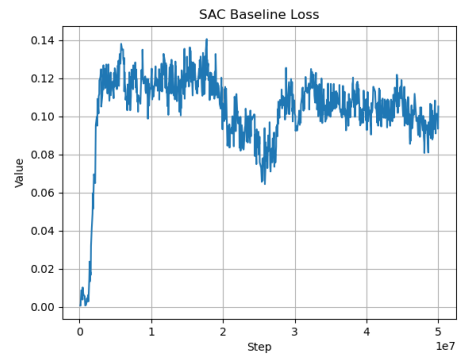
5 Results



Figure 1: Value loss comparison between three policies (POCA, PPO, and SAC)



(a) POCA Baseline Loss



(b) SAC Baseline Loss

Figure 2: Baseline loss comparison between POCA and SAC policies

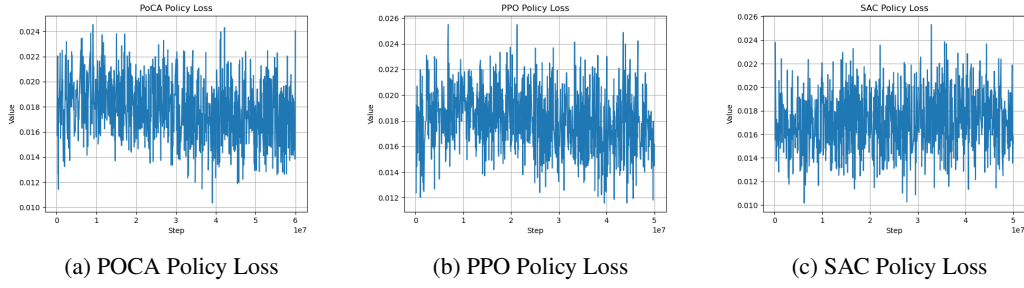


Figure 3: Policy loss comparison between three policies (POCA, PPO, and SAC)

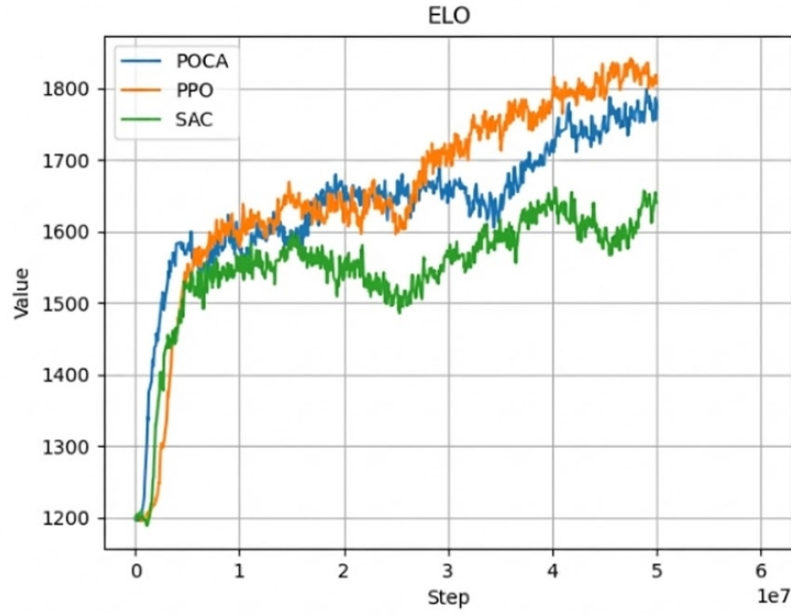


Figure 4: ELO rating progression during training

Table 2: Performance comparison between PPO, PoCA, and SAC policies

	PPO	PoCA	SAC
PPO	-	41%	70%
PoCA	59%	-	73%
SAC	30%	27%	-

*Win rates are computed from 100 simulated games between each policy pair.
Cell values represent win rate of row policy against column policy.*

6 Conclusion

This study presents a comprehensive comparison of three reinforcement learning algorithms—PPO, SAC, and POCA—in a competitive multi-agent soccer environment. Our experimental results reveal significant performance differences among the algorithms, with POCA emerging as the most effective policy for this multi-agent competitive setting.

6.1 Key Findings

The experimental results demonstrate that POCA (POsthumous Credit Assignment) achieves superior performance in head-to-head competitions, winning 59% of matches against PPO and 73% against SAC. This strong performance can be attributed to POCA’s specialized design for competitive multi-agent environments, which effectively handles agent interactions and credit assignment in team-based scenarios. The algorithm’s centralized training, decentralized execution (CTDE) approach, combined with its ability to handle agent spawning and despawning without absorbing states, makes it particularly well-suited for dynamic multi-agent environments.

PPO shows competitive performance against SAC (70% win rate) but struggles against POCA (41% win rate). Despite this, PPO demonstrates strong generalization capabilities, as evidenced by its ability to adapt to different opponent strategies. Notably, during training, PPO agents gradually developed role specialization, with distinct defender and striker behaviors emerging in the later stages of training. This emergent role differentiation highlights PPO’s generalization power and its capacity to discover strategic team formations through self-play.

Multi-agent POCA, with its inherent support for multi-agent coordination, exhibited faster emergence of role specialization compared to PPO. The algorithm’s natural multi-agent collaborative properties enabled it to more rapidly converge to a strategy where agents differentiated into defender and striker roles, contributing to its superior overall performance. This faster role emergence can be attributed to POCA’s centralized training approach, which allows for better coordination and credit assignment among team members.

SAC, despite benefiting from extended reward shaping and off-policy learning advantages, performs poorly in this environment, achieving only 27% and 30% win rates against PoCA and PPO respectively. This suggests that while SAC’s reward structure includes continuous shaped rewards, the reward shaping may require more fine-tuning to effectively guide learning in competitive multi-agent scenarios. The current reward design, while beneficial for off-policy learning, may not adequately capture the complex strategic interactions and coordination requirements inherent in competitive team-based environments. More sophisticated reward engineering that explicitly accounts for multi-agent dynamics, team coordination, and opponent behavior could potentially improve SAC’s performance in this setting.

6.2 Limitations

Several limitations should be acknowledged. First, the experiments were conducted in a specific environment configuration, and the results may not generalize to other multi-agent competitive settings. Second, the training was limited to 50 million steps, and longer training might reveal different performance characteristics. Third, the hyperparameter configurations were based on standard practices but were not extensively tuned for this specific environment, which may have affected the relative performance of the algorithms.

7 Future Work

Several promising directions for future research emerge from this study:

Reward Engineering: While SAC used extended reward shaping, further investigation into reward structures specifically designed for competitive multi-agent settings could improve all algorithms’ performance. Curriculum learning approaches that gradually increase environment complexity might also enhance training efficiency.

Architecture Improvements: Exploring more sophisticated network architectures, such as attention mechanisms or graph neural networks that explicitly model agent relationships, could better capture the multi-agent dynamics in competitive scenarios.

Self-Play Enhancements: The self-play mechanisms used in this study could be further optimized. Investigating adaptive opponent sampling strategies, population-based training, or more sophisticated ELO-based matchmaking could improve training stability and final performance.

Transfer Learning: Investigating whether policies trained in this environment can transfer to related multi-agent competitive tasks, or whether pre-training on simpler environments can accelerate learning, represents an interesting research direction.

Real-World Applications: Extending these findings to real-world multi-agent competitive scenarios, such as robotic soccer, autonomous vehicle coordination, or strategic game playing, would validate the practical applicability of these algorithms.

Theoretical Analysis: A deeper theoretical understanding of why POCA outperforms other algorithms in this setting, particularly regarding credit assignment in competitive environments, could inform the design of future multi-agent reinforcement learning algorithms.

8 Appendix

8.1 Code and Environment Availability

We built a standalone environment that can be run with Unity. The complete source code, training scripts, and documentation are publicly available to facilitate reproducibility and further research.

Repository: <https://github.com/kyon317/Reinforcement-Learning-for-Multi-Agent-Soccer>

To run our codes and scripts, please follow the instructions provided in the README file. The repository includes:

- Complete Unity project setup with ML-Agents integration
- Training configuration files for PPO, SAC, and POCA algorithms
- Pre-trained model files for all three policies
- Detailed installation and usage instructions

Pre-trained Models: All trained models can be found in `/Assets/SoccerTwo/Models/`, organized by algorithm type (PPO, SAC, and POCA). These models can be directly loaded in Unity for inference and evaluation.

References

- [1] T. Bansal, J. Pachocki, S. Sidor, I. Sutskever, and I. Mordatch. Emergent complexity via multi-agent competition. *arXiv preprint arXiv:1710.03748*, 2018.
- [2] A. Cohen, E. Teng, V.-P. Berges, R.-P. Dong, H. Henry, M. Mattar, A. Zook, and S. Ganguly. On the use and misuse of absorbing states in multi-agent reinforcement learning. In *RL in Games Workshop at AAAI*, 2022.
- [3] J. N. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson. Counterfactual multi-agent policy gradients. *arXiv preprint arXiv:1705.08926*, 2018.
- [4] T. Haarnoja, A. Zhou, P. Abbeel, and S. 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.
- [5] Z.-W. Hong, S.-Y. Su, T.-Y. Shann, Y.-H. Chang, and C.-Y. Lee. A deep policy inference q-network for multi-agent systems. *arXiv preprint arXiv:1712.07893*, 2018.
- [6] A. Juliani, V.-P. Berges, E. Vckay, Y. Gao, H. Henry, M. Mattar, and D. Lange. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*, 2018.
- [7] F. Lin, Z. Wang, B. Li, R. Wang, Z. Zheng, Y. Wang, J. Yu, W. Wang, and J. Wang. Tizero: Mastering multi-agent football with curriculum learning and self-play. *arXiv preprint arXiv:2302.07515*, 2023.
- [8] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *arXiv preprint arXiv:1706.02275*, 2017.
- [9] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- [10] C. Yu, A. Velu, E. Vinitisky, J. Gao, Y. Wang, A. Bayen, and Y. Wu. The surprising effectiveness of ppo in cooperative multi-agent games. *arXiv preprint arXiv:2103.01955*, 2022.