Stimulating Soccer with Multi-Agent Reinforcement Learning: A 2 vs 2 Approach

Shravan Conjeevaram
Texas A&M University
shravan10@tamu.edu

Phanender Chalasani
Texas A&M University
phanenderchalasani@tamu.edu

Video Link **GitHub Link**

***Abstract*—Multi-agent reinforcement learning(MARL) presents unique challenges towards the development of autonomous systems that can cooperate as well as compete. We present an implementation for a 2-vs-2 soccer environment where agents will have to learn complex behaviors through reinforcement learning. We use Proximal Policy Optimization with centralized training and decentralized execution to develop agents that can show effective team coordination and compete against opponents. Our implementation focuses on realistic physics simulation, sophisticated reward engineering, and balanced learning objectives for creating intelligent soccer-playing agents.**

Keywords: **Multi Agent Reinforcement Learning, Autonomous Systems, Proximal Policy Optimization, Physics Simulation, Reward Engineering**

I. Introduction

Setting up multi-agent reinforcement learning in sports simulation environments has been an important challenge in the field of artificial intelligence. Soccer, in particular, presents a complex domain requiring agents to master individual skills and team coordination in a continuous state space with partially observable information. Our work implements a 2-vs-2 soccer environment in which autonomous agents should learn to coordinate within their own team and compete with opponents.

Our implementation utilizes a Markov Decision Process framework incorporating:

- Continuous state space representing player positions, ball dynamics, and game state
- Discrete action space for movement and ball control
- Physics-based transition functions
- Structured reward system balancing immediate and long-term objectives

We employ Proximal Policy Optimization (PPO) as our primary learning algorithm, enhanced with multi-agent coordination mechanisms. The agents are trained by Centralized Training with Decentralized Execution (CTDE), where agents learn from others in a centralized fashion but perform as an independent agent during gameplay.

The key contributions of this work include:

- Implementation of a scalable multi-agent soccer environment
- Development of a logical reward structures promoting both individual and team performance
- Integration of physics-based simulation with reinforcement learning
- Analysis of emergent cooperative and competitive behaviors

This problem is sequential because the agents' actions depend on both the current state and past actions; each of these steps influences the future state and rewards. A striker moving towards the ball or a goalie's positioning in relation to a shot creates the temporal evolution in a soccer environment. The agents make decisions in which the rewards are delayed, spread over several time steps, and the agents need to learn what happened due to past decisions. This temporal dependency is modeled as a Markov Decision Process (MDP), where future actions depend on past experiences, making the problem inherently sequential. Techniques such as Proximal Policy Optimization (PPO) take this into consideration by calculating the advantage of actions over time with respect to both immediate and far-reaching consequences-a key aspect in the development of strategies and optimization for long-term performance.

Unity, with its ML-Agents Toolkit, is used to simulate the environment. Its physics engine supports realistic ball and player dynamics, while its multi-agent capabilities make it ideal for reinforcement learning experiments. Unity's flexibility and scalability allow for efficient training and testing of the agents.

II. Motivation

The development of intelligent autonomous agents that can exhibit coordinated behavior in complex environments remains one of the biggest challenges in AI research. Our project tries to address this challenge using a 2-versus-2 soccer simulation,

providing an ideal balance between complexity and computational feasibility. A soccer environment provides unique opportunities to explore both cooperative and competitive behaviors simultaneously, making it an excellent testbed for multi-agent reinforcement learning algorithms.

We implement a centralized training and decentralized execution to develop agents capable of learning sophisticated team strategy while maintaining independent operations in gameplay. In this manner, we further not only the basic understanding of multi-agent systems but also the practical insights necessary for coordinated decision-making under uncertainty. Moreover, the project provides a foothold into scaling to more complex scenarios and may contribute to broader applications in robotics and autonomous systems where the agents have to balance between keeping individual and team objectives.

III. Literature Review

This section covers ten papers we referred to while working on this project. These papers focus on topics like multi-agent reinforcement learning, reward engineering, and the use of Proximal Policy Optimization (PPO) in similar setups. Some ideas from these works were directly implemented, while others guided us in adapting techniques to suit the requirements of our 2v2 soccer environment. We discuss the key aspects of these papers, highlighting the methods we followed, modified, or avoided based on our project goals.

[1] Schulman et al. (2017) introduced Proximal Policy Optimization, a stable and efficient policy gradient method. PPO uses a clipped probability ratio that balances exploration and stability, avoiding large policy updates. It is computationally efficient and effective in dynamic environments, making it suitable for our 2v2 soccer simulation to enable coordinated and strategic decision-making by agents.

[2] Egorov (2016) showed how MADQN could be combined with CNNs for processing complex state representations in multi-agent setups. CNNs help agents to comprehend spatial data such as position and interaction, which aids decision-making. This idea has been adapted in our project to enhance state representation such that agents can dynamically position themselves for better coordination.

[3] Ning and Xie (2024) presented a centralized training and decentralized execution (CTDE) framework for multi-agent environments. This approach allows agents to learn together during training while executing strategies individually during gameplay. We applied this approach in our soccer simulation to balance team coordination and competitive strategies effectively.

[4] Duan et al. (2012) used probabilistic neural networks to advance the action prediction mechanism for enhancing coordination within a team of robot soccer players. Joint-state and joint-action representation were also explored. We didn't implement action prediction; however, the focus on the joint-state representation helped to get cooperative strategies refined within teams in this project.

[5] Nakahara et al. (2023) proposed a method for calculating Q-values for on-ball and off-ball actions in soccer using MARL. Their approach considers player contributions based on spatial and temporal positioning. While not directly implemented, this idea inspired the structuring of our MARL environment to improve team strategies and positioning.

[6] Wu et al. (2021) proposed CoPPO, extending PPO with coordinated policy updates through dynamic step-size adaptation. We borrowed ideas from CoPPO regarding how to balance individual learning with team coordination, especially for the cooperative striker-goalie pairs in our soccer simulation.

[7] Li et al. (2024) used a decentralized MARL framework with GPU-accelerated simulations to train agents efficiently. Their curriculum learning approach inspired our training process, while their work on sparse reward settings provided insights into developing effective team strategies in our simplified 2v2 scenario.

[8] Stone et al. (2005) applied reinforcement learning to Soccer Keepaway, focusing on multi-agent coordination and sparse rewards. Their work highlighted ways to handle large state spaces and break down complex soccer tasks, which influenced our approach to designing a more tractable 2v2 scenario.

[9] Song et al. (2024) standardized environment settings and benchmarked MARL algorithms in the Google Research Football environment. Their findings, especially regarding the effectiveness of policy-based methods, guided our algorithm selection and training process.

[10] Smit et al. (2023) show for the first time the scaling of MARL towards full 11v11 soccer. While we focus on a much simpler 2v2 scenario, we still used their improvements to PPO, as well as their focus on adequate environment design regarding multi-agent coordination.

IV. Methodology

A. Unity Environment

Our implementation utilizes Unity ML-Agents[11] to create a 2-versus-2 soccer environment where four agents compete in teams to score goals while defending their own goal. The environment provides physics-based simulation with customizable parameters for realistic gameplay dynamics, as shown in Figure 1.

1) Environment Setup: The simulation environment consists of: - Four agents divided into two teams of two players each - A spherical ball with adjustable scale (default: 7.5) - Physics engine with configurable gravity (default: 9.81) - Bounded playing field with goals at opposite ends

2) State Representation: The state space comprises 336 dimensions derived from: - 11 forward ray-casts distributed over 120 degrees (264 dimensions) - 3 backward ray-casts over 90 degrees (72 dimensions) - Each ray-cast detects 6 object types and their distances - Three observation stacks for temporal information

3) Action Space: Agents utilize a discrete branched action space with three components: - Forward/backward movement - Sideways movement - Rotational control

4) Reward Structure: The reward function implements a temporal-dependent scoring system with various factors taken into account.

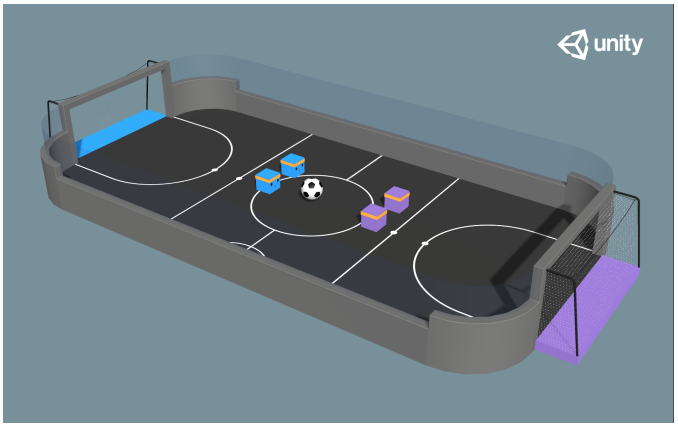

Fig. 1. Unity ML-Agents 2v2 Soccer Environment showing the playing field with two teams (blue and purple) and a soccer ball in the center. The environment includes goals at both ends and clearly marked field boundaries.

B. Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) is a policy gradient method that aims to optimize an agent's policy while maintaining stability by limiting the size of policy updates. The PPO objective function is defined as:

$$\mathcal{L}^{\text{PPO}} = \mathbb{E}_t\Big[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\Big] \quad (1)$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio, ϵ is the clipping parameter, and $\hat{A}_t$ is the advantage estimate.

PPO uses Generalized Advantage Estimation (GAE) to compute $\hat{A}_t$, which balances bias and variance:

$$\hat{A}_t = \sum_{l=0}^{\infty}(\gamma\lambda)^l\delta_{t+l} \quad (2)$$

where $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$ is the temporal difference error, γ is the discount factor, and λ controls the bias-variance tradeoff.

1) Agent Architecture: The architecture for the PPO agent consists of a shared network with separate actor and critic outputs. The network processes the input state through three fully connected layers with LayerNorm and ReLU activations. The actor output produces action probabilities using a softmax layer, and the critic output predicts the value function. The network is defined as:

$$\begin{aligned} x_1 &= \text{ReLU}(\text{LayerNorm}(\text{FC}_1(s))),\\ x_2 &= \text{ReLU}(\text{LayerNorm}(\text{FC}_2(x_1))),\\ x_3 &= \text{ReLU}(\text{FC}_3(x_2)),\\ \pi(a|s) &= \text{softmax}(\text{Actor}(x_3)),\\ V(s) &= \text{Critic}(x_3), \end{aligned}$$

where s is the input state, $\pi(a|s)$ is the action probability distribution, and $V(s)$ is the state value estimate.

2) Training Setup: The Unity ML-Agents toolkit was used to simulate the soccer environment. The state space includes positional, velocity, and angular information for agents and the ball, while the action space consists of actions such as movement and kicking.

The training process consists of three distinct phases: - Phase 1: Single striker training against randomly acting opponents - Phase 2: Striker-goalie pair training against stationary team - Phase 3: Striker-goalie pair against randomly acting opponents

The striker-goalie pair learns through PPO, while the opponent team provides a competitive baseline for the agents to train. Training is conducted for 3000 episodes, with each episode consisting of up to 1000 steps.

3) Reward Design: The reward function is carefully designed to encourage cooperative and competitive behaviors.

We used different rewards for Phase 1 and Phase 2,3

Reward Design for Phase-1

$$Reward = \begin{cases} +1.0 & \text{if a goal is scored}\\ -1.0 & \text{if a goal is conceded}\\ +0.5 & \text{if a successful defense is made}\\ +0.1 & \text{if a successful pass is completed}\\ -0.1 & \text{if a failed pass occurs (ball intercepted)}\\ 0 & \text{otherwise.} \end{cases}$$

Reward Design for Phase-2,3

- **Ball Control:** Agents are rewarded for maintaining proximity to the ball, with a penalty for being far from the ball:

$$r_{\text{control}} = \begin{cases} 2.0 & \text{if } d_{\text{agent,ball}} < 1.0,\\ -0.2 \cdot d_{\text{agent,ball}} & \text{otherwise.} \end{cases} \quad (3)$$

- **Goal Alignment:** Reward is proportional to the alignment of the agent-ball vector with the ball-goal direction:

$$r_{\text{align}} = 0.5 \cdot \frac{(\vec{p}_{\text{ball}} - \vec{p}_{\text{agent}}) \cdot (\vec{p}_{\text{goal}} - \vec{p}_{\text{ball}})}{\|\vec{p}_{\text{goal}} - \vec{p}_{\text{ball}}\|} \quad (4)$$

- **Scoring Goals:** Large rewards are given for scoring goals, with additional rewards for reducing the distance between the ball and the goal:

$$r_{\text{goal}} = \begin{cases} 100.0 & \text{if } d_{\text{ball,goal}} < 1.5, \\ 2.0 \cdot (5.0 - d_{\text{ball,goal}}) & \text{if } d_{\text{ball,goal}} < 5.0. \end{cases} \quad (5)$$

where $d_{\text{x,y}}$ = Distance between x and y and p_{x} = Position of x

4) Hyperparameters: The PPO hyperparameters were tuned and tested before arriving at the final values. A grid-search was done to get the best values. The ones that were used are:

- Learning rate: 3×10^{-4}
- Clipping parameter: $\epsilon = 0.2$
- Discount factor: $\gamma = 0.99$
- GAE parameter: $\lambda = 0.95$
- Value loss coefficient: 0.5
- Entropy coefficient: 0.01
- Maximum gradient norm: 0.5
- Mini-batch size: 64
- Epochs per update: 10

C. Multi-Agent Posthumous Credit Assignment

Multi-Agent Posthumous Credit Assignment (MA-POCA)[12],[13] is a multi-agent reinforcement learning technique that retrospectively analyzes entire episodes to more accurately assign credit to individual agents actions. MA-POCA introduces a centralized evaluation mechanism that retrospectively analyzes team performance, enabling precise determination of individual agent contributions beyond traditional uniform reward distribution. By utilizing a strategic computational critic that deeply analyzes the actions of each agent and its contextual impact, MAPOCA enables more subtle policy learning in complex collaborative scenarios. This method keeps decision-making decentralized while introducing a centralized performance evaluation; this improves the capability of agents to devise refined cooperative strategies. With this approach, MA-POCA tries to overcome fundamental limitations in the current multi-agent learning frameworks, especially within team-based domains that demand sophisticated inter-agent coordination.

1) Core Parameters in POCA: The core parameters used for training agents:

- Trainer type: `poca` - Specifies the Multi-Agent Posthumous Credit Assignment algorithm as the training method.
- Max steps: `30000000` - Sets a 30 million step training limit for both Striker and Goalie agents.
- Keep checkpoints: `5` - Preserves the 5 most recent model checkpoints during training.

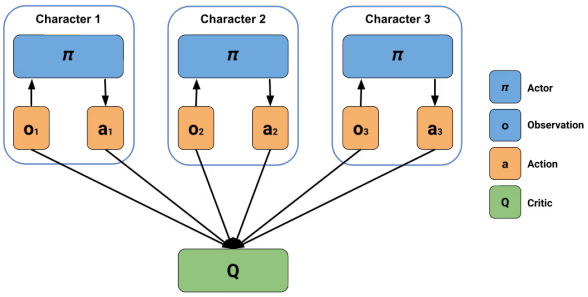

Fig. 2. Working of POCA Algorithm [source]

2) Hyperparameters used for training agents: The hyperparameters used are:

- Batch size: `2048` - Defines the experience batch size used in each training iteration.
- Buffer size: `20480` - Sets the experience replay buffer capacity for storing agent interactions.
- Learning rate: `0.0003` - Determines the gradient update step size with constant schedule.
- Beta: `0.005` - Controls entropy regularization for exploration balance.
- Epsilon: `0.2` - Sets the PPO clipping parameter for policy updates.
- Lambda: `0.95` - Specifies the GAE (Generalized Advantage Estimation) lambda parameter.
- Number of epochs: `3` - Number of training passes through the experience buffer.

3) Network Architecture used for training agents: The network architecture parameters used are:

- Hidden units: `512` - Specifies 512 neurons in each hidden layer.
- Number of layers: `2` - Implements a two-hidden-layer neural network structure.
- Normalize: `false` - Disables input normalization.
- Visual encoding type: `simple` - Uses basic visual encoding for processing observations.

4) Reward Configuration used for training agents: The reward configuration parameters used are:

- Gamma: `0.99` - Sets the future reward discount factor.
- Strength: `1.0` - Applies a unit scaling to extrinsic rewards.
- Time horizon: `1000` - Defines the sequence length for training.
- Summary frequency: `10000` - Sets training summary generation interval.

5) Self-Play Settings: The self-play settings used are:

- Save steps: `50000` - Frequency for saving model snapshots.
- Team change: `200000` - Interval between team composition changes.

- Swap steps: `1000` for Goalie, `4000` for Striker - Different role swap frequencies.
- Window: `10` - Maintains 10 past policies for training.
- Play against latest model ratio: `0.5` - 50% chance of playing against the most recent policy.

V. Results

After training the models with different reward functions, the results we obtained were not satisfactory. We tried different reward functions, checking for convergence, but each reward function has its own disadvantages. We tried using negative reward when agents scored own goal, which made agents not to defend the goals. We tried using positive reward when agents move the ball, but this resulted in agents just moving in random and not scoring any goals. So, after these initial experiments, we tried adding another reward function where agents are rewarded with one point when a goal is scored and one point is deducted when a goal is conceded. This reward function resulted in positive rewards but the agents didn't learn how to score goals.

The main issue from graph shown in Figure 3 is that the

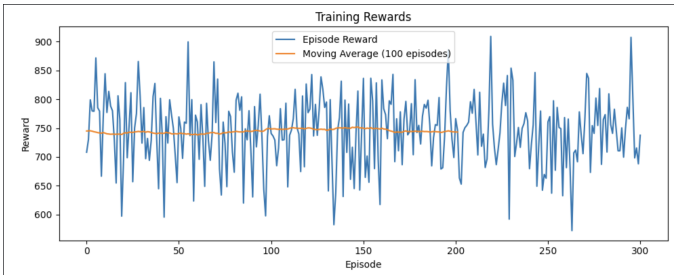

Fig. 3. Episodes vs Rewards Plot at the end of 300 episodes

rewards the agents receive are varying a lot from episode to episode, as can be seen by ups and downs in the blue line. That reflects that the agents sometimes do well but not always, which means their performance is not reliable. The fluctuations suggest that though the agents can achieve high rewards, they struggle to maintain this performance consistently. We tried giving a reward for having control of the ball and penalized when agents move away from the ball but this resulted in very high negative rewards which did not show

We tried training the agents using the mlagents toolkit[12] using both MA-POCA and PPO. We trained the agents for 5 million steps. We saved the model in ONNX(Open neural networks exchange) format and pushed the model to HuggingFace to visualize the performance of Agents[POCA Model], [PPO Model]. HuggingFace has inbuilt visualisation to watch the agents play in Soccer environment [source]. We can watch the agents play in the environment by selecting the model we pushed. `Phani0404/poca-SoccerTwos1` and `Phani0404/poca-SoccerTwos_ppo`.

While watching the agents play in the environment we noticed that agents trained with POCA performed better than agents trained on PPO. While POCA uses a centralized critic with self-attention mechanism that can better handle team dynamics to individual agents, PPO lacks POCA's specialized ability to perform credit assignment determining each agent's specific contribution to the team's success, which is crucial in cooperative environments where agents share a reward function.

VI. Conclusion

Our implementation of 2-vs-2 soccer simulation with multi-agent reinforcement learning highlighted several important challenges and insights. A key challenge in the process was the difficulty in designing an effective reward structure that could steer agent behavior consistently towards desired outcomes. Various reward structures are experimented with such as goal-based rewards, rewards over ball control, and defensive positioning; yet performance patterns proved to be highly volatile. Reward fluctuations across training episodes suggest that while agents do achieve occasional episodes with achieved high performance, they struggled to maintain con- istent behavior. Several key limitations emerged from this implementation:

- The reward engineering proved particularly challenging, as different reward structures led to unintended behaviors, such as agents avoiding defensive responsibilities or engaging in random movement patterns
- Even with positive rewards for scoring and negative rewards for conceding goals, agents failed to develop reliable goal-scoring strategies
- The high variance in episode rewards suggests funda-Mental problems either with the learning process or the environment design.

These results demonstrate the difficulty of training multi- agent systems in competitive environments and point to the development of more sophisticated approaches to reward design and credit assignment in multi-agent reinforcement learning settings. Future work is required for developing more stable mechanisms of reward and investigation of alternative training architectures that can handle the complexities of team-based competitive scenarios.// After several tries with dif- With the different hyperparameters and reward functions, we moved to mlagents toolkit inbuilt training method called mlagentslearn; we trained two models based on POCA and PPO algorithms. We watched the models play against each other and noticed that POCA outperforms PPO. For our future plans, we will try to discover a better reward function and try to implement the environment for eleven players in each team.

References

[1] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[2] M. Egorov, "Multi-agent deep reinforcement learning," *CS231n: Convolutional Neural Networks for Visual Recognition*, Stanford University, 2016.

[3] Z. Ning and L. Xie, "A survey on multi-agent reinforcement learning and its application," *Journal of Automation and Intelligence*, 2024.

[4] Y. Duan, B. X. Cui, and X. H. Xu, "A multi-agent reinforcement learning approach to robot soccer," *Artificial Intelligence Review*, vol. 38, no. 3, pp. 193–211, 2012.

[5] H. Nakahara, K. Tsutsui, K. Takeda, and K. Fujii, "Action valuation of on-and off-ball soccer players based on multi-agent deep reinforcement learning," 2023.
[6] Z. Wu, C. Yu, D. Ye, J. Zhang, H. Piao, and H. H. Zhuo, "Coordinated Proximal Policy Optimization," in *Advances in Neural Information Processing Systems (NeurIPS 2021)*, 2021.
[7] Z. Li, F. Bjelonic, V. Klemm, and M. Hutter, "MARLadona - Towards Cooperative Team Play Using Multi-Agent Reinforcement Learning," 2024.
[8] P. Stone, R. S. Sutton, and G. Kuhlmann, "Reinforcement Learning for RoboCup-Soccer Keepaway," *Adaptive Behavior*, vol. 13, no. 3, pp. 165–188, 2005.
[9] Y. Song, H. Jiang, H. Zhang, Z. Tian, W. Zhang, and J. Wang, "Boosting Studies of Multi-Agent Reinforcement Learning on Google Research Football Environment: The Past, Present, and Future," in *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024)*, pp. 1772, 2024.
[10] A. Smit, H. A. Engelbrecht, W. Brink, and A. Pretorius, "Scaling multi-agent reinforcement learning to full 11 versus 11 simulated robotic football," *Autonomous Agents and Multi-Agent Systems*, vol. 37, no. 2, pp. 1–34, 2023.
[11] https://github.com/Unity-Technologies/ml-agents/blob/develop/docs/Learning-Environment-Examples.mdsoccer-twos
[12] https://unity-technologies.github.io/ml-agents/Training-ML-Agents/
[13] A. Cohen, E. Teng, V. P. Berges, and R. P. Dong, "On the Use and Misuse of Absorbing States in Multi-agent Reinforcement Learning," *arXiv:2111.05992v2 [cs.LG] 7 Jun 2022*, 2022.