DRF: Dense Reward Reinforcement Learning Framework based on Large Language Models

Jin Ziyuan
ShanghaiTech University
jinzy2024@shanghaitech.edu.cn

Cui Hanjia
ShanghaiTech University
cuihj2024@shanghaitech.edu.cn

Abstract

Sparse reward signals in multi-agent reinforcement learning (MARL) often impede effective learning and coordination, as agents receive feedback only for rare outcomes like scoring a goal in soccer environment. We propose DRF, an approach that leverages large language model (LLM) to automatically generate dense reward shaping function, providing rich intermediate feedback without extensive manual engineering. DRF iteratively refines the reward function through human-in-the-loop feedback, allowing high-level human guidance while minimizing low-level reward tuning. In a 3v3 soccer environment, DRF-enabled agents trained with self-play achieve significantly higher ELO ratings and exhibit more coordinated team behaviors than those using sparse or manually shaped rewards. These results demonstrate that LLM-generated dense rewards can effectively overcome sparse reward bottlenecks, improving learning efficiency and multi-agent coordination.

1 Introduction

Reinforcement learning (RL) agents often struggle when feedback signals are spare and delayed. With such sparse rewards, agent receive little guidance on how intermediate actions contribute to success. This challenge is amplified in multi-agent domains, such as soccer or team-based games where a reward might only be given for scoring a goal at the end of episode. The lack of frequent feedback complicates credit assignment, making it difficult for each agent to learn which behaviors lead to the team's win. Past work [1] has shown that without additional guidance, agents may require prohibitively many trials to acquire high-level teamwork from low-level interactions.

Reward shaping [3] has long been studied as a remedy for sparse rewards in reinforcement learning. By adding an auxiliary reward to the environments base reward, one can provide the agent with hints that guide it toward desired behavior. However, designing an effective dense reward function by hand is time-consuming and demands significant expertise in both the task domain and RL tuning.

Introducing dense rewards in a competitive multi-agent setting can easily destabilize training if those rewards misalign with the true objective. By rewarding intermediate behaviors (e.g. chasing or kicking the ball) too much, the agent may start optimizing for those proxy rewards rather than actual goals. For example, in a soccer scenario, giving a reward for simply touching the ball may leads agents to endlessly dribbling the ball without scoring. This behavior yields high cumulative reward from the agent's perspective but does not translate into wins. Balancing multiple reward terms is tricky in practice, and poor tuning can alter the optimal policy. Prior research has established conditions like potential-based reward shaping, proving that if the shaping reward is derived from a potential function on states, it preserves the optimal policy of the original. This ensures the agents optimal solution under the shaped reward is still optimal for the original task. However, crafting such potential functions for complex multi-agent tasks is non-trivial.

Another issue is credit assignment noise in multi-agent rewards. If agent gets incremental rewards that depend on global states rather than its own actions, the learning signal becomes noisy or misleading.

For example, one striker might receive a reward because the ball moved toward the opponent's goal, even if a teammate was the one who kicked it. This make it hard for the policy to figure out which actions were truly beneficial. Some approaches [7] propose difference rewards or individualized shaping to tackle multi-agent credit assignment where each agent gets a shaped reward that subtracts the team reward without its contribution, to explicitly measure individual impact. While effective in principle, these methods still require knowledge of each agent's individual contribution and require manual specification of the reward decomposition.

Designing rewards for competitive multi-agent environments requires care to avoid the pitfalls. To overcome the difficulty of manual reward engineering, we explore the use of large language models to automate the design of dense reward functions. Recent advances [2, 6] demonstrate that LLMs can generate structured code or descriptions from high-level instructions, suggesting they can be harnessed to write reward functions given a description of the task. The generated reward functions were interpretable and effective, in some cases matching or exceeding expert-designed rewards in single-agent domains. These developments inspire our approach to use LLMs for automatic reward shaping in multi-agent environments. By having an LLM generate a dense reward function from an environment description, we aim to provide agents with informative feedback at fine time granularity, without manually encoding domain-specific reward heuristics.

2 Background

PPO Proximal Policy Optimization (PPO) [5] is a policy-gradient reinforcement learning algorithm introduced by Schulman et al. as an efficient, stable improvements on Trust Region Policy Optimization (TRPO) [4]. Let $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$. TRPO maximizes a "surrogate" objective:

$$L^{CPI}(\theta) = \hat{E}_t\big[r_t(\theta)\hat{A}_t\big]$$

Without a constraint, maximization of L^{CPI} would lead to an excessively large policy update. PPO modify the objective to penalize changes to the policy that move $r_t(\theta)$ away from 1:

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

Intuitively, PPO's clipping acts as a soft trust region: it prevents big policy jumps could collapse performance, while it allow more flexibility than hard constraints like in TRPO.

Self-Play Self-play mirrors how humans improve by playing against opponents of similar skill and gradually increasing competition level. Self-play pits agents against copies of themselves or past versions, creating a curriculum of progressively harder opponents. In a symmetric game such as soccer, one can train a single neural network policy and use it to control both teams during training, periodically freezing one teams policy as a fixed opponent while the other team learns, and vice versa. Using self-play, the difficulty for each team automatically adjusts, each team is always competing against a recent version of itself, providing a moving target that prevents overfitting to a static opponent.

3 Method

DRF use an LLM to automatically create a detailed reward function that provides feedback for intermediate behaviors in a multi-agent game. The DRF approach consists of an iterative four-step cycle, illustrated below:

Prompting and Code Generation A crucial aspect of using LLMs for reward design is how we prompt the model. The prompt must provide enough context for the LLM to understand the environment and the desired behaviors, while avoiding ambiguity that could lead to incorrect or irrelevant reward terms. Key sources of information for the prompt include the environment description, observation and action space, and the desired reward logic. The prompt should also mention any safety or practicality constraints (e.g., do not use information not given, keep magnitudes small and balanced). Finally, when prompting the LLM, it may be useful to adopt chain-of-thought prompting, encourage the model to reason about why certain behaviors should be rewarded.

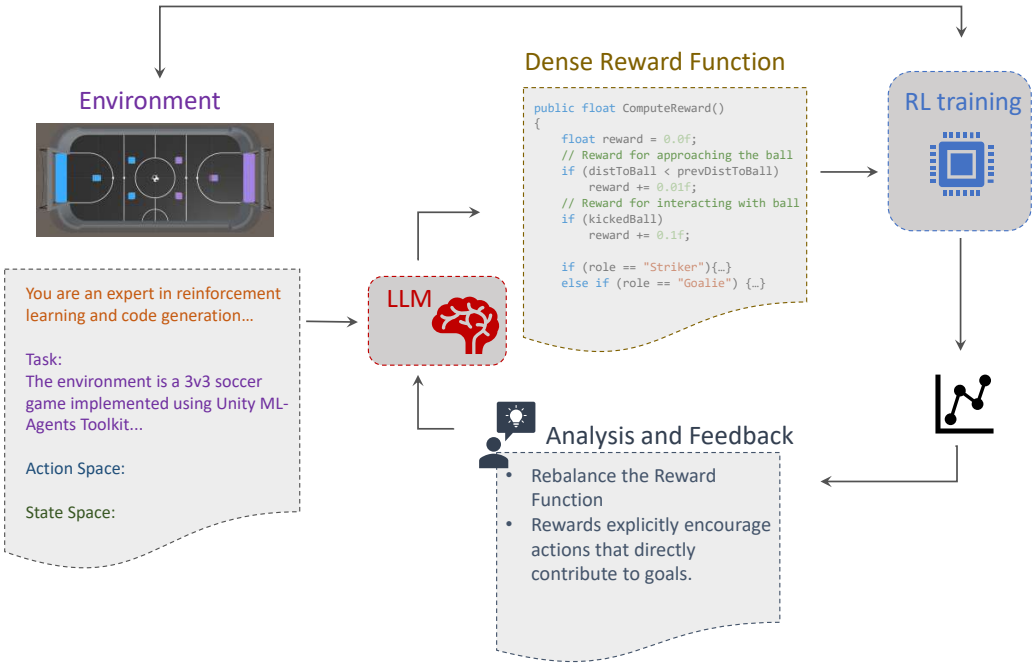

Figure 1: LLM-driven reward design loop. The Reward Designer LLM generates an initial dense reward function from environment and task descriptions. The policy is trained with this reward, and a Analyzer evaluates the agents behavior. The LLM incorporates the suggestion into the reward code and the cycle repeats, progressively refining the reward function.

One of the advantages of using an LLM to generate this code is interpretability. The output is a human-readable script where each reward term is explicitly coded. This is far more interpretable than a learned neural network reward model. It allowed us to inspect the code and understand why the agents might be behaving a certain way since we can see what is being rewarded. In our experiments, we indeed cross-verified the LLMs reward code to ensure it aligned with our intentions before deploying it for training.

Integration and Training The generated reward code is integrated into the environment as the reward function for the agents. Before running training, we verify the LLMs reward code. This includes basic syntax check and logical sanity checks. We then train the agents using the dense reward function. We run the training for a fixed number of timesteps and monitor performance metrics like ELO rating over time.

Upon integrating an LLM-generated reward function into training, it is unlikely that optimal performance will be achieved on the first deployment. Iterative refinement, particularly through the incorporation of human insights, can significantly enhance the effectiveness of the reward function.

Behavior Analysis Following an initial training iteration using the LLM-generated reward, we systematically analyze the resulting agent behaviors. This analysis involves reviewing episodes of agents engaged in simulated soccer matches, either through direct observation of simulations or through detailed examination of logged trajectories and performance statistics. The primary objective is to detect undesirable behaviors or overlooked incentives within the agents' learned strategies. For instance, analysts may identify cases where agents disproportionately exploit a particular dense reward (e.g., they pass excessively to farm passing rewards but then never shoot). Alternatively, analysts might note insufficient reinforcement of critical teamwork behaviors by the existing reward function. While human expertise is integral to this analytical process, analysts do not manually

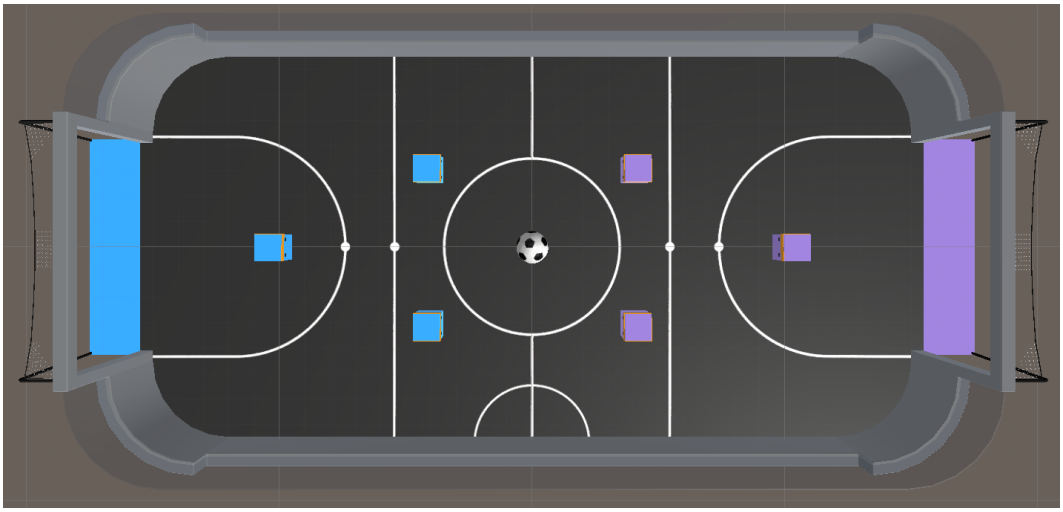

Figure 2: Unity ML-Agents 3v3 Soccer environment.

modify the reward function code. Instead, they summarize observed issues and suggest adjustments to facilitate further automated refinement by the LLM.

Refinement via Re-Prompting Using the observations from the last step, we construct a new prompt for LLM to refine the reward function. We feed this feedback to the LLM, possibly along with the previous reward code, and ask it to modify or regenerate the reward function accordingly. This iterative loop can be repeated multiple times until the reward function is deemed satisfactory. In practice, we found that only a few iterations were needed in our experiments to reach a well-shaped reward function.

Human feedback To further enhance the integration of human feedback, we conducted an additional experimental attempt. During the game execution, we sampled a batch of performance scenarios that align more closely with human expectations. For instance, in a soccer game, reinforcement learning strategies might encourage agents to remain in a stalemate to maintain higher reward returns, whereas humans prefer to see two forwards cooperate and dribble past opponents. To enable the LLM to better reflect such reward design tendencies, we collected relevant samples and performed supervised fine-tuning (SFT) on a locally deployable model (QWen2.5-VL-7B), thereby more effectively incorporating human preferences into the LLM.

4 Experiment

We evaluate the DRF approach on a multi-agent soccer environment using the Unity ML-Agents toolkit. The 3v3 Soccer environment is a physics-based multi-agent simulation in which two teams compete in a soccer match. Each team consists of 2 offense-oriented strikers and 1 defense-oriented goalie. An episode consists of a timed match where play resets after a goal or when a maximum number of simulation steps is reached. In this environment, each teams objective is to score goals in the opponents net while preventing goals in their own net. The environment is thus a mix of competitive and cooperative dynamics. Agents on the same team must cooperate while both teams are in competition.

Observation Space The environment is partially observable. Each agent does not have access to the full game but rather perceives the world through a set of ray sensors detecting nearby objects (ball, teammates, opponents, walls, goals) within a certain angle and distance.

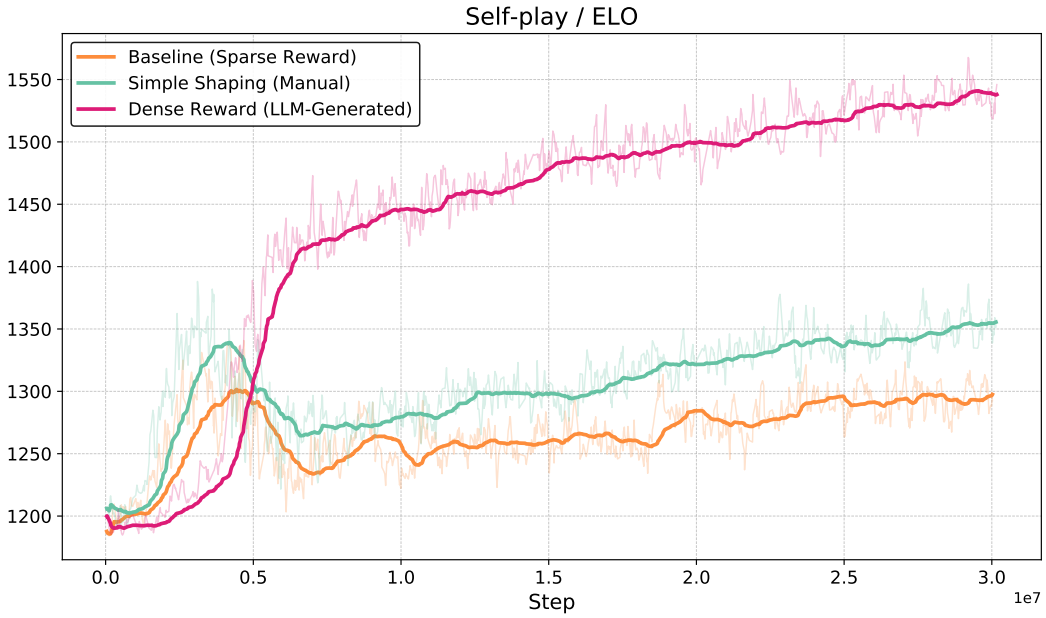

Figure 3: Self-play ELO performance curves over training steps for the three reward strategies. The dense reward approach clearly produces the fastest and highest rise in ELO, indicating superior policy performance.

Action Space Each agent's action is specified in a branched discrete format. The agent can choose a movement direction (move forward, backward, move sideways) and a rotation (rotate left, right). The action space is same for all agents, and each agent chooses its action independently at each step.

Rewards The default reward scheme in the soccer environment is designed to encourage scoring and winning while discouraging passive play. The rewards are extremely sparse and team-based. A team receive a +1 reward (minus a small time penalty factor) only when scoring a goal against the opponent and a -1 reward if a goal is scored against their own team. Aside from the goal events, there are no other rewards. This sparse reward structure reflects the true objective to win the game, but provides very little learning signal. An agent might take thousands of steps with no reward, then suddenly get a positive reward at a goal, making it hard to assign credit to the actions that led to that success. The difficulty is compounded in multi-agent environment because a single agent's reward also depends on the teamwork and opposing team's behavior.

For experimental comparison, we define three reward settings for training the agents:

Baseline Sparse Reward Agents are trained with the environments sparse reward only. No additional shaping is provided. This represents the default case with minimal feedback.

Manual Dense Reward We designed a simple dense reward function with a few shaping terms, informed by basic soccer heuristics. In addition to the same goal rewards as baseline, the manual shaping reward included. For example, similar to previous work, we award a small positive reward whenever an agent touches or kicks the ball, encouraging active play with the ball since possessing the ball is prerequisite to scoring in soccer. The shaping magnitudes are tuned lightly so as not to overshadow the main goal reward.

LLM-Generated Dense Reward This is the complex reward function generated by the language model after iterative refinement. It includes a richer set of terms that densely cover various aspects of good play. For instance, the LLM-generated reward function might incorporate offensive incentives, defensive incentives and teamwork incentives. This reward function provides dense feedback at every time step.

To ensure a fair comparison, all training hyperparameters and network architectures were kept identical across the three reward settings, only the reward function differed. We use GPT-4.5 as our LLM, given its ability to understand natural language descriptions and generate syntactically correct code. PPO algorithm is selected as the basic algorithm for agent model training. We do not use any pre-trained model, instead training via self-play. Each training run consists of 30 million environment steps. We ran 3 independent training runs for each method to account for stochasticity, and we report the average trends.

Evaluation Metrics The primary metric for performance is the ELO rating achieved by the trained agents in self-play matches. ELO is a standard rating system for competitive games, which we adapt here. Agents start around 1200 ELO, and as training progresses, their self-play framework adjusts their ELO based on wins or losses against past versions. A higher ELO indicates a stronger policy. We track the ELO of the latest policy model over training time.

As shown in Figure 3, the DRF method leads to a dramatic improvement in learning speed and final performance. Agents trained with the dense reward function exhibit a steep, sustained rise in ELO, surpassing 1400 within the first 6 million steps. By contrast, agents trained under the sparse-reward baseline improve only gradually, reflecting the well-known challenges posed by sparse feedback. The simple shaped reward yields an intermediate trajectory. While it speeds up learning relative to the baseline, its gains plateau earlier and never catches up to the dense reward.

After 30 million training steps, DRF-trained agents achieve an ELO of approximately 1550, whereas agents using the simple shaped reward level off near 1350 and the baseline remains around 1300. Notably, most of the final performance disparity emerges early in training, indicating that the rich intermediate feedback encoded by the LLM-generated dense reward enables agents to uncover effective strategies far more rapidly than the alternative schemes. In essence, Figure 3 highlights that a well-shaped, information-rich reward signal can dramatically accelerate learning and yield stronger policies in multi-agent competitive settings.

5 Future Work

The success of the LLM-generated reward highlights the richness of information that modern language models have about tasks. The model effectively drew on general knowledge of soccer to formulate the reward. However, we also note some limitations. The quality of the reward function is highly dependent on the prompt given to the LLM and the models capability. A flawed prompt could lead to a poor reward that might misguide the agent. We had to remain in the loop to verify and tweak the LLMs proposals. The present findings are contingent on the capacity of the underlying model. Replicating comparable performance on smaller-scale models remains an open objective that we intend to pursue in future work.

Our experiments in the selection of foundational models reveal that models within the GPT4o series and above, including GPT4.5 and o3, demonstrate significant capabilities in generating reasonable and efficient reward function designs. In contrast, models with performance metrics below GPT4o, such as QWen2.5-VL-7B and InternVL, exhibit notable deficiencies in generating semantically coherent reward structures, even after supervised fine-tuning.

We have designed a set of reasonable and general system prompts for the DRF. To further enhance its generalization capabilities, our next step is to delegate the generation of both the action space and state space within the game environment to the DRF. Additionally, to enable the analyzer within the DRF to better align with human preferences, we plan to incorporate a memory storage module (RAG) and a reflection mechanism. This will allow the DRF to fully absorb human feedback and summarize experiences across a wide range of gaming environments, ultimately forming a fully automated dense reinforcement learning reward design methodology.

6 Conclusion

In this project, we demonstrated that using a language model to generate dense reward functions can substantially enhance the training of agents even in a complex multi-agent environment. The fusion of LLMs and reinforcement learning opens up exciting possibilities to inject knowledge and preferences into the traditionally trial-and-error learning process. By generating dense reward functions, LLMs

act as a bridge between high-level intent and low-level training signals. Our project focused on a concrete soccer application but the principles extend to any multi-agent system with sparse rewards. With careful prompt design and iterative feedback, we can harness this approach to train agents that learn faster and align better with the expected behaviors.

Acknowledgments

We are grateful to the instructor and TAs in the reinforcement learning course for their helpful guidance and insightful feedback.

References

[1] ABE, T., ORIHARA, R., SEI, Y., TAHARA, Y., AND OHSUGA, A. Step-by-step acquisition of cooperative behavior in soccer task. *Journal of Advances in Information Technology 13*, 2 (4 2022), 147–154.

[2] MA, Y. J., LIANG, W., WANG, G., HUANG, D., BASTANI, O., JAYARAMAN, D., ZHU, Y., FAN, L., AND ANANDKUMAR, A. Eureka: Human-level reward design via coding large language models. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024* (2024).

[3] NG, A. Y., HARADA, D., AND RUSSELL, S. J. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning* (San Francisco, CA, USA, 1999), ICML '99, Morgan Kaufmann Publishers Inc., p. 278287.

[4] SCHULMAN, J., LEVINE, S., ABBEEL, P., JORDAN, M. I., AND MORITZ, P. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015* (2015), F. R. Bach and D. M. Blei, Eds., vol. 37 of *JMLR Workshop and Conference Proceedings*, JMLR.org, pp. 1889–1897.

[5] SCHULMAN, J., WOLSKI, F., DHARIWAL, P., RADFORD, A., AND KLIMOV, O. Proximal policy optimization algorithms. *CoRR abs/1707.06347* (2017).

[6] XIE, T., ZHAO, S., WU, C. H., LIU, Y., LUO, Q., ZHONG, V., YANG, Y., AND YU, T. Text2reward: Reward shaping with language models for reinforcement learning. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024* (2024).

[7] YANG, C., YANG, G., AND ZHANG, J. Learning individual difference rewards in multi-agent reinforcement learning. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems* (Richland, SC, 2023), AAMAS '23, International Foundation for Autonomous Agents and Multiagent Systems, p. 24182420.

A Appendix / supplemental material

Here is the example of our prompt:

```
You are an expert in reinforcement learning and code generation. You excel at
↪  understanding the objectives of tasks and analyzing potential effects observable
↪  from states and actions. Your goal is to write a detailed reward function that
↪  efficiently guides agents to learn optimal strategies.

Task:
The environment is a 3v3 soccer game implemented using Unity ML-Agents Toolkit.
↪  There are two teams (blue vs. purple), each consisting of two Strikers and one
↪  Goalie. The primary goal of each team is to score by getting the ball into the
↪  opponent's goal while defending their own goal.

Action Space:
{
  "forward": "Forward", // "Forward" or "Backward"
  "strafe": "Left", //  "Left" or "Right"
  "rotate": "Left" //  "Left" or "Right"
}

State Space:
{
  "ball": {
    "position": [x, y, z],
    "rotation": [qx, qy, qz, qw]
  },
  "goals": {
    "GoalBlue": {"position": [x, y, z]},
    "GoalPurple": {"position": [x, y, z]}
  },
  "agents": [
    {
      "team": "Blue", // "Blue" or "Purple"
      "role": "Striker", // "Striker" or "Goalie"
      "position": [x, y, z],
      "rotation": [qx, qy, qz, qw]
    },
    { /* other agents */ }
  ]
}

Key Considerations:
1. We already include a goal group reward. A staged reward could make the training
↪  more stable.
2. You may define clear, observable metrics that encourage cooperation.
3. You may define role-specific rewards for goalie and striker accordingly.
4. Do not use information not given and focus on most relevant factors.

Present your thought process step-by-step:
1. Analyze and discuss the potential impact of actions and state information on team
↪  performance.
2. Suggest detailed reward design ideas.
3. Write the reward function clearly.
```