OptimizingMARL: Developing Cooperative Game Environments Based on Multi-agent Reinforcement Learning

Thaís Ferreira[1](✉), Esteban Clua[1], Troy Costa Kohwalter[1], and Rodrigo Santos[2]

[1] Universidade Federal Fluminense, Niteroi, Brazil
thais_ferreira@id.uff.br, {esteban,troy}@ic.uff.br
[2] Universidade Federal do Estado do Rio de Janeiro, Rio de Janeiro, Brazil
rps@uniriotec.br

Abstract. Intelligent agents are critical components of the current game development state of the art. With advances in hardware, many games can simulate cities and ecosystems full of agents. These environments are known as multi-agent environments. In this domain, reinforcement learning has been explored to develop artificial agents in games. In reinforcement learning, the agent must discover which actions lead to greater rewards by experimenting with these actions and defining a search by trial and error. Specifying when to reward agents is not a simple task and requires knowledge about the environment and the problem to be solved. Furthermore, defining the elements of multi-agent reinforcement learning required for the learning environment can be challenging for developers who are not domain experts. This paper proposes a framework for developing multi-agent cooperative game environments to facilitate the process and improve agent performance during reinforcement learning. The framework consists of steps for modeling the learning environment and designing rewards and knowledge distribution, trying to achieve the best environment configuration for training. The framework was applied to the development of three multi-agent environments, and tests were conducted to analyze the techniques used in reward design. The results show that the use of frequent rewards favors the emergence of essential behaviors (necessary for the resolution of tasks), improving the learning of agents. Although the knowledge distribution can reduce task complexity, dependency between groups is a decisive factor in its implementation.

Keywords: Multi-agent reinforcement learning · Cooperative environments · Games

This work is supported by CAPES and FAPERJ.

1 Introduction

The use of Artificial Intelligence (AI) techniques in electronic games has been notably explored in recent years [15,19]. This trend can be explained by the constant advances in research, the evolution in hardware and processing, and the consumer market demands. Reinforcement Learning (RL) [10] is an AI technique that has been largely applied and tested in games [20]. In RL, agents learn through rewards. If the agent receives a positive reward, it begins to understand that this behavior is beneficial. Similarly, if it receives a negative reward, it understands that something is wrong and looks to avoid this behavior. Through these rewards, the agent can learn, so the reward design must be carried out effectively, especially in multi-agent environments, where the distribution of rewards should favor collaboration among agents.

Understanding the whole process of multi-agent reinforcement learning is not a simple task. Although many works explore the Multi-Agent Reinforcement Learning (MARL) domain [2,4,5,9,16,18,19,21], the focus is not on game development but rather on creating new environments or algorithms. These environments are scaled for testing in specific domains with a focus on the performance of the algorithms used [1,8,11,12]. Exploring the MARL domain in game development is extremely important and can bring new perspectives and approaches. Moreover, designing rewards for the best agent learning is not an easy task, and it is necessary to explore and present methods around this challenge.

In this context, we propose the *OptimizingMARL*, a framework for cooperative game environments development based on MARL. The framework brings practices that guide the development and configuration of cooperative multi-agent environments looking for the best performance in agent training. This performance is related to the use of dense rewards (frequent rewards) to encourage the emergence of essential behaviors - those necessary to solve the tasks and the problem. It is possible to identify how the reward design influences the agents' training performance and which strategies are best for each environment developed through the experiments performed. Furthermore, the framework consists of steps for modeling the learning environment, enabling macro understanding of RL and game elements.

This paper is organized as follows: Sect. 2 outlines related work. Section 3 presents the *OptimizingMARL*. Section 4 describes the conducted experiments, presenting the development of the environments applying the framework and the analysis of the results. Finally, Sect. 5 concludes this work, listing contributions, limitations, and future work.

2 Related Work

In the context of MARL, many works explore the use of new algorithms to improve agent learning performance [1,8,11,12]. Foerster et al. [5] present the COMA algorithm that uses the same centralized network for all agents, with the agents' shared observations and actions as input. COMA uses the centralized network to estimate how much each agent influenced the expected value. Each agent learns via a modeled reward that compares the overall reward to the received when that agent's action is replaced by a default action [5].

Cohen et al. [3] introduce MA-POCA (*Multi-Agent Posthumous Credit Assignment*), a new algorithm that is very similar to COMA, but is capable of training groups of agents to solve tasks in which individual agents can be removed or introduced during the episode. Iqbal and Sha [7] present the MAAC algorithm that trains decentralized policies in multi-agent settings using a centralized network capable of sharing and selecting relevant information for each agent at every instant. While these works test the algorithms in environments that simulate games, the focus is not on game development. This works does not specify how the reward design was performed, as the focus is on the results and performance of each algorithm.

Through the literature review, we found only one study by Zhao et al. [22] that most closely resembles our proposal. Zhao et al. [22] present a hierarchical approach for training agents by breaking complex problems into a hierarchy of easy learning tasks to achieve near-human behavior and a high level of skill in team sports games. The authors first train the agents against other easy-level agents and then make the level more difficult once the agent has learned the basics. For the training, they use a proprietary tool they call *Simple Team Sports Simulator (STS2)*. It simulates a 2D soccer environment, in which the state space consists of the coordinates of the players, their speeds, and an indicator of who has the ball.

The authors run tests with agents in 1 vs 1 and 2 vs 2 matches using the PPO [14], DQN [13], and Rainbow [6] algorithms. Through the analysis of the results, the authors concluded that sparse score-based rewards are not sufficient to train the agents even at a high level, which requires the application of a more refined reward design.

Following the proposal of Zhao et al. [22], we seek to improve agent performance in MARL. However, our focus is on modeling the elements of MARL and on reward design. We propose an easy and more practical way to create multi-agent environments that enables a macro understanding of RL elements in the context of a cooperative multi-agent game environment.

3 The Framework OptimizingMARL

Training intelligent agents in cooperative game environments may not be a simple task. The process requires a series of steps and procedures such as specifying the agents, designing rewards, and developing the environment. Moreover, designing rewards that enable faster learning is not trivial, making the process of creating a multi-agent environment optimized for training difficult. To simplify and optimize the process of building and training intelligent agents in cooperative game environments, we propose the *OptimizingMARL*, which comprises three macro-steps: (1) conceptual modeling of the learning environment;

(2) reward design and knowledge distribution; and (3) the creation of the environment. Figure 1 illustrates the macro-steps of *OptimizingMARL* and the next subsections explain them in detail.

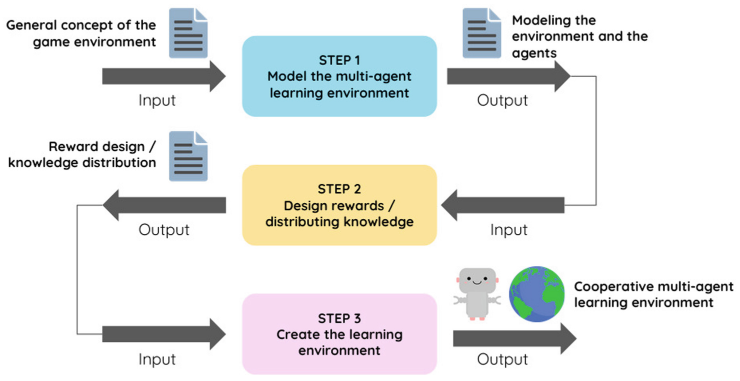
General concept of the game environment
Input
STEP 1
Model the multi-agent learning environment
Modeling the environment and the agents
Output
Reward design / knowledge distribution
Output
STEP 2
Design rewards / distributing knowledge
Input
STEP 3
Create the learning environment
Input
Cooperative multi-agent learning environment
Output

Fig. 1. Overview of the macro steps of the *OptimizingMARL*.

3.1 Model the Multi-agent Learning Environment

In this step, we conceptually specify game and RL elements (e.g., agents, environment). The **concept of the game/environment** provides critical information about the goals and tasks that the agents must accomplish and about the objects that will be present in the environment (e.g., objects that the agents can interact with, obstacles, enemies). The **environment** consists of the place where agents will interact and make decisions. The environment must have the necessary characteristics and objects for the agents to interact and solve their tasks. We can create a document specifying all the objects (and their properties) that must be present in the environment. We define the **scenario objects** - those that are part of the game, such as items, enemies, and obstacles, that the agents will interact with - and the **objects' characteristics** - each object of the scenario has its properties such as colliders, texture, physics, among others.

The **agent** is the actor that observes and performs actions in the environment. An agent must have a **observation space** - the agent collects its observations about the state of the world before making decisions - and **actions**. Finally, there is the reward signal. The reward signal should not be given all the time, but only when the agent performs an action that is good or bad for solving the problem. The reward signal is how the goals are communicated to the agent, so it needs to be configured so that maximizing the reward generates the desired optimal behavior. The next subsection presents the *OptimizingMARL* step related to reward design and the possibility of knowledge distribution based on this design.

3.2 Design Rewards and Distributing Knowledge

The reward is primal in RL and it works as an incentive mechanism, telling the agent what is correct or not. The RL agent's goal is to maximize the total reward received during an execution episode. The reward is immediate and defines the characteristics of the problem to be faced by the agent [17]. To know at what times to reward agents, it is essential that developers have mastery of the problem to be solved in the environment. It is common to have the necessary knowledge about the environment in games. However, depending on the complexity of the environment and the tasks, the projection of rewards may not be trivial or easy to visualize. Therefore, the *OptimizingMARL* comprises a set of activities for understanding the problem to be solved in the environment and for designing rewards. The steps consist of understanding the agents' goals as well as the tasks required to solve the problem. Figure 2 presents the flow of activities.

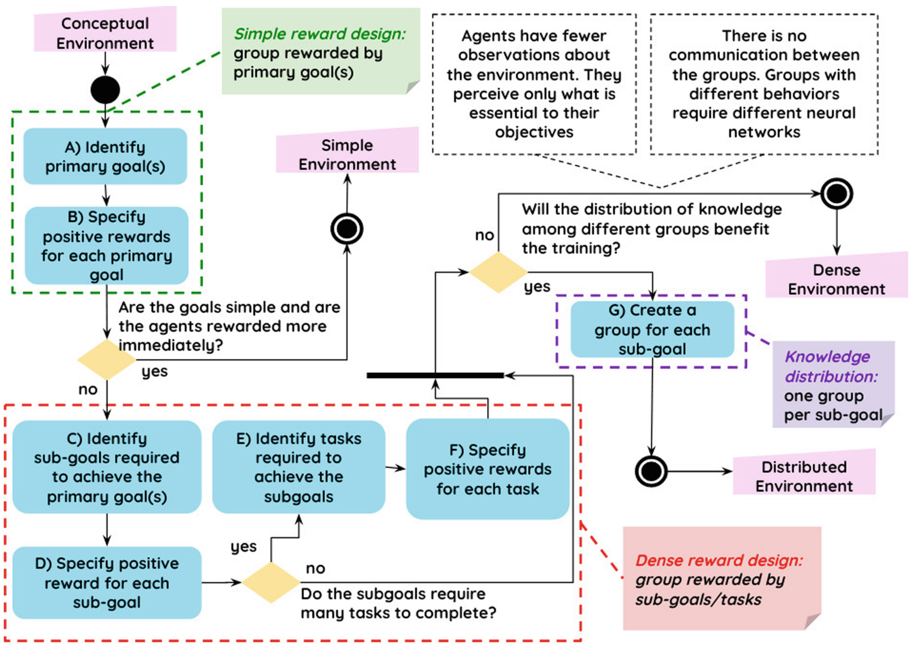
Conceptual Environment
Simple reward design: group rewarded by primary goal(s)
Agents have fewer observations about the environment. They perceive only what is essential to their objectives
There is no communication between the groups. Groups with different behaviors require different neural networks
A) Identify primary goal(s)
B) Specify positive rewards for each primary goal
Simple Environment
Will the distribution of knowledge among different groups benefit the training?
no
yes
Dense Environment
Are the goals simple and are the agents rewarded more immediately?
yes
no
G) Create a group for each sub-goal
Knowledge distribution: one group per sub-goal
C) Identify sub-goals required to achieve the primary goal(s)
E) Identify tasks required to achieve the subgoals
F) Specify positive rewards for each task
Distributed Environment
D) Specify positive reward for each sub-goal
yes
no
Do the subgoals require many tasks to complete?
Dense reward design: group rewarded by sub-goals/tasks

Fig. 2. Activity flow of the reward project/knowledge distribution of the *Optimizing-MARL*.

Simple Reward Design. It starts by identifying the primary goals (activity A) through the conceptual modeling of the environment. The primary goals are directly related to the main problem that the group of agents needs to solve. After their identification it specifies a reward for each of these primary goals (activity B). The reward design is complete, and the environment can begin to be modeled if the group gains rewards so that they can learn the behaviors essential to solving the problem. However, if the goals are still complex and the rewards are still sparse, this can hinder the agents' learning, causing training to take longer. Therefore, it is necessary to design dense rewards so that the group gains rewards more frequently to encourage the emergence of essential behaviors. We define these behaviors as those indispensable to solving the task.

Dense Reward Design. To define dense rewards, it is first necessary to identify the sub-goals derived from the primary goals (activity C). A reward is specified for each of these sub-goals (activity D). Therefore, the group gains rewards more frequently, learning essential behaviors more immediately. If the agents still need to do many tasks to achieve these sub-goals, it is interesting to identify these tasks (activity E) and reward the group for each task performed (activity F). Rewards for primary goals, sub-goals, and tasks should follow a value hierarchy. The group should be rewarded discretely for completing tasks while completing sub-goals will be rewarded more heavily until they complete the primary goal (higher reward value). Therefore, it prevents agents from getting "stuck" just completing tasks instead of progressing and solving the main problem of the environment. After step D (or F), if the knowledge distribution is not suitable for the situation, we end the process and generate a conceptual environment with dense rewards. Otherwise, we proceed to activity (G) and specify a group for each sub-goal following the dense reward design.

Knowledge Distribution. It is necessary to analyze whether the distribution of knowledge will favor training in the environment after specifying the rewards for sub-goals (and optionally the tasks). This step consists in creating a group for each sub-goal. Since each group has a specific goal, the agents may have fewer insights into the environment. After creating a group for each sub-goal, we generate the distributed conceptual environment.

3.3 Create the Learning Environment

The environment development step is the last step of the process. The development consists of creating the necessary elements in the scenario, the agents, programming their behaviors, indicating what to do when an agent is restarted, specifying its sensors, configuring the algorithms and the neural network parameters, and providing the rewards when necessary, among other activities. Unlike the previous two stages, the environment creation step is more specific, as it depends on the platform being used for agent development and training.

4 Experiments

The experiments were developed using the Unity ML-Agents Toolkit [20]. The flexibility of Unity enables the creation of tasks to complex 3D strategy games, physics-based puzzles, or multi-agent competitive games. Unlike other research platforms, Unity is not restricted to any specific gameplay genre or simulation, making it a general platform. We use the MA-POCA [3], a new algorithm designed to train groups of agents to solve tasks.

We performed tests for three configurations of the same environment: simple environment, based on a simple reward design (1), dense environment, which has a dense reward design (2), and a distributed environment, which has a knowledge division among different groups (3). In all experiments the agents perceive the environment through ray cast sensors. These sensors can perceive an element through a tag every time a ray intersects an object. If the tag of that object is specified in the sensor, the agent can perceive this object. ML-Agents Toolkit provides a fully connected neural network model to learn from those observations.

Figure 3 shows the three environments developed and used in the tests. The first environment (left) was not created entirely by us. This environment comes with Unity-ML agents, but we made changes to the environment and followed the steps in the framework to get the three possible configurations. The following subsections describes each environment in more details and our respective findings.

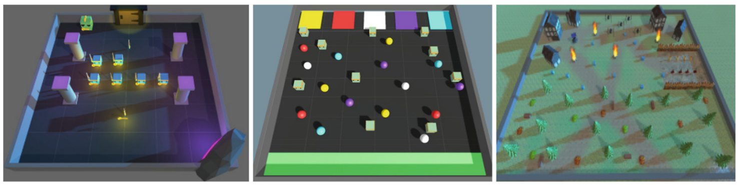

Fig. 3. Dungeon Escape environment (left); Color Balls (center); Wild World (right).

4.1 Dungeon Escape Environment

In the Dungeon Escape environment, the agents cooperate to escape from a dungeon before the dragon escapes through the portal. The agents need to defeat the dragon, get a key that was dropped, and open the door. The agents must collect the sword and hit the dragon to defeat the dragon. From this concept of the Dungeon Escape environment, we use the framework to model the learning environment (step 1). It defines the scenario objects (e.g., the agents, the dragon, the door, the key, the sword) and their properties (e.g., colliders, texture). After that, we applied step 2 of the *OptimizingMARL* following the activities to obtain the three configurations for the environment (simple, dense, and distributed). Figure 4 shows the designed reward and distributed knowledge activities. Finally, we create the learning environment (step 3).

In step 2, we first identify the primary goal, which is to escape from the dungeon. Then we reward the group when they achieve this goal. Since we applied

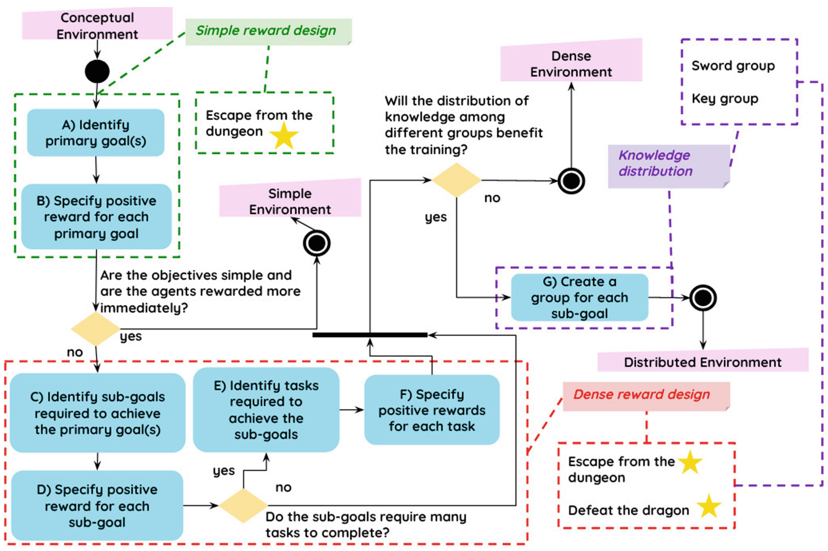
Conceptual Environment
Simple reward design
A) Identify primary goal(s)
B) Specify positive reward for each primary goal
Escape from the dungeon
Simple Environment
Are the objectives simple and are the agents rewarded more immediately?
yes
no
C) Identify sub-goals required to achieve the primary goal(s)
D) Specify positive reward for each sub-goal
E) Identify tasks required to achieve the sub-goals
F) Specify positive rewards for each task
yes
no
Do the sub-goals require many tasks to complete?
Will the distribution of knowledge among different groups benefit the training?
yes
no
Dense Environment
Sword group
Key group
Knowledge distribution
G) Create a group for each sub-goal
Distributed Environment
Dense reward design
Escape from the dungeon
Defeat the dragon

Fig. 4. Dungeon Escape environment rewards and knowledge distribution project.

only the steps related to the simple reward design, this environment is the simple reward environment (1). We continue following the activities to develop the dense reward environment (2). We identify the sub-goal needed to achieve the primary goal: defeat the dragon. Now the group gets rewarded when they defeat the dragon and when they escape from the dungeon. Since the environment is relatively simple, we decided not to divide the sub-goal into tasks and reward the group for them. But this is at the discretion of the development team.

We continue following the activities to develop the environment (3). Once we have identified the sub-goals, we can create an agent type responsible for each of these sub-goals. So we specify an agent type responsible for picking up the sword and defeating the enemy, and another agent type responsible for collecting the key and opening the door. So we will have two groups in this environment.

In both simple (1) and dense (2) environments, there is one group composed of six agents. All agents have the same behavior. The group gains a reward only when an agent escapes from the dungeon. In the distributed environment (3), the agents are divided into two groups, each composed of three agents. The *sword_group* is responsible for collecting the sword and defeating the dragon. This group gain a positively reward only when it defeats the dragon. The *key_group* is responsible for collecting the key that the dragon drops and for opening the door. This group gain a positively rewarded only when it effectively opens the door. The *sword_group* does not perceive the key, the door, or who has the key. The *key_group* does not perceive the sword, nor who has the sword.

Figure 5 presents the results for all environments: (1) simple reward design, (2) dense reward design, and (3) knowledge distribution. The maximum average reward that can be obtained by a group in all environments is 1.0. For analysis purposes, we perform knowledge distribution even though it is not suitable for this environment due to one group's dependence on the other.

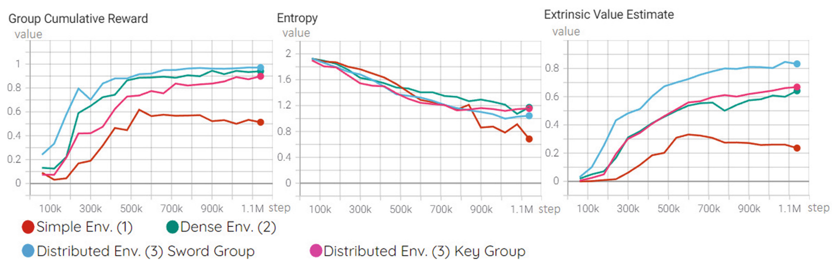

Fig. 5. Results for the three environments in Dungeon Escape experiment.

The cumulative reward group is the mean cumulative episode reward overall agents and should increase during a successful training session. Observing the cumulative reward graph, it is evident that the use of dense rewards (2, green line) improves learning performance if compared to the simple environment (1, red line). In the distributed environment (3), the *sword_group* (light blue line) obtained high rewards. From the graph, the dependency between the groups is clear. The *sword_group* was able to get more rewards because it depends only on itself. The *key_group* (pink line) got smaller rewards because to complete their goal they need the *sword_group* to complete theirs first. The key group gets more rewards as the *sword_group* learns.

Entropy is related to how random the decisions of the model are. It should slowly decrease in a successful training session. It occurred for all environments, meaning that the models make less random decisions during the sessions training. The extrinsic value estimate is the mean value approximation for all states visited by the agent. Once these values have converged to the optimal state values, then the optimal policy can be achieved, which should increase during a successful training session. The *sword_group* in environment 3 presented higher values when compared to the others environments. In this way, this group can obtain the optimal policy in fewer steps.

Through the analysis, we can conclude that dense sub-goal reward design is the best option for the Dungeon Escape environment. Although the *sword_group* in environment 3 shows better learning, the *key_group* was disadvantaged by the dependency between the groups. In addition, specifying the dense rewards for this environment takes less time than specifying two different agent types for each group.

4.2 Color Balls Environment

In the Color Balls environment, the agents need to get all fifteen balls to the goal (green area). To catch a ball of a specific color, the agent must wear the same color uniform. To put on the uniform, the agent needs to get on one of the colored squares. To change the color of the uniform, only climb the square of the desired color. When the agent owns a ball, it cannot change its uniform. We applied the framework to develop the three configurations of the same environment.

In the simple (1) and dense (2) environments, there is one group with ten agents. In environment 1, the group is positively rewarded when it completes the primary goal, which is when all fifteen balls are placed into the green area. In environments 2 and 3, the group is positively rewarded for each ball placed in the green area. In environment 3, the agents do not need to correlate the ball color with the uniform color they are wearing because each group has a specific color. The agents only need to understand that they need to be wearing their uniforms to collect the ball.

Figure 6 shows the results for the three environments. For better visualization, the graph shows the results of red group in environment 3. All groups in this environment have the same behavior, where the only difference is that each one perceives the ball and the square in the color of its group. For comparison, the maximum average reward that can be obtained by the groups is 5.0 in all environments. As expected, the simple environment (orange line) showed low group reward value because agents are failing to complete the tasks before time runs out. The entropy showed a slight decay but less when it is compared to the other environments. Since agents fail to accumulate rewards, the model continues to work on bad policies, and the estimated value for all states does not increase.

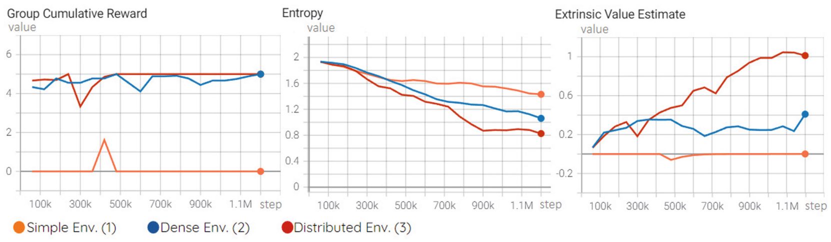

Fig. 6. Results for the three environments in color balls experiment.

This environment is challenging because agents need to understand that they can only pick up a ball if they are wearing the same color uniform. In this case, the vector of observations has an integer variable that corresponds to the color of the ball (1 = yellow, 2 = red, 3 = white, 4 = purple, and 5 = blue). Thus, it is necessary to associate the variable value with the tag in the raycast sensor, making learning more difficult. The group's performance improved greatly with the design of the dense rewards (blue line). In environment 2, the group is rewarded for each ball placed in the green area. More immediate reward facilitates the association between the uniform color (integer variable) and the ball tag (string in the raycast sensor).

In environment 3, we changed the integer variable to boolean, indicating whether the agent wore the uniform or not. In addition, the agents have decreased their perceptions and no longer need to associate the colors with the balls. Each group has a specific color and the agents need to put on their uniform, pick up the balls, and place them in the goal. This environment showed the best results, reaching the maximum reward starting at 480k steps. For the value estimate, the groups in environment 3 presented higher values when compared to environments 1 and 2. As such, the groups can obtain the optimal policy in fewer steps.

The analysis allowed us to conclude that distributing knowledge is the best option for the Color Balls environment. The groups in this environment have the same perceptions and behavior, making the knowledge distribution a simple task. The main change is in the vector of observations that starts working with a boolean variable instead of an integer.

4.3 Wild World Environment

In this environment, the agents of a village must survive for an "indeterminate" amount of time in a hostile environment. It means that while the agents keep reaching the goals, they can accumulate rewards until the maximum number of steps in the episode is reached. These goals consist of: making food to feed all the villagers, keeping the fireplaces lit to fight the cold and prevent the villagers from getting hypothermia, and fighting the ferocious wolves that reside in the forest near the village and can attack the villagers. We applied the *Optimizing-MARL* to develop the three configurations of the environment (simple, dense, and distributed).

In the simple (1) and dense (2) environments, there is one group consisting of fifteen agents. In environment 1, the group is positively rewarded when it completes the primary goals: prepare food, light fire, and defeat a wolf. In environment 2, the group is positively rewarded when it reaches the primary goals (higher reward) and sub-goals (small reward): plant vegetables, take wood, and equip the sword.

In environment 3, the agents are distributed into three groups. The *food_group* is responsible for collecting seed, planting the vegetables, harvesting the vegetables, and preparing the food (higher reward). This group is positively rewarded when it plants vegetables (small reward) and prepares the food. The *fire_group* is responsible for equipping ax, taking wood, and lighting the fire (higher reward). This group is positively rewarded when it takes the wood (small reward) and light the fire. The *wolf_group* is responsible for equipping the sword and defeating a wolf. This group is positively rewarded when it equips the sword (small reward) and defeats a wolf (higher reward).

Figure 7 shows the results for the three environments. The *wolf_group* in environment 3 is the only one that has a reward limit. This group gains rewards when an agent equips a sword and when an agent defeats a wolf. However, the number of swords and wolves is finite, making the maximum reward 36 per episode. Regarding the group cumulative reward, the simple environment (dark blue line) presents the lowest values. This can be easily explained because the group is rewarded only when it achieves its primary goals. The problem is that for an agent to achieve one of these goals, it needs to perform a series of tasks beforehand.

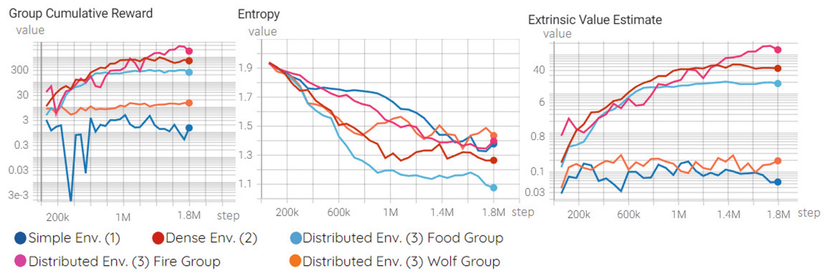

Fig. 7. Results for the three environments in wild world experiment.

The group in environment 2 shows a huge improvement over the previous group. The sub-goal reward encourages agents to complete the primary goals in stages. The *food_group* and *fire_group* groups in environment 3 were also able to achieve high rewards. However, even with fewer perceptions about the environment, the cumulative reward did not show much improvement compared to the group in environment 2. The *wolf_group* kept the accumulated reward around 15 (36 is the maximum value), showing no great improvement or worsening.

The entropy fell for all groups, although it fell more sharply in the *food_group* in environment 3, indicating less random decisions. For the value estimate, the group in environment 2 and the *food_group* and *fire_group* in environment 3 presented higher values when compared to the others. In this way, the groups can obtain the optimal policy in fewer steps.

From the analysis, we can conclude that dense sub-goal reward design is the best option for the Wild World environment. The *wolf_group* in environment 3 cannot perform well. Also, specifying the dense rewards for this environment takes less time than specifying three different agent types for each group.

5 Conclusion

In this paper, we introduced a framework to develop multi-agent environments for cooperative games. The proposed framework addresses the design of dense rewards for emergence of essential behaviors and the knowledge distribution. Through the framework it is possible to design environments aiming to improve the agents' training performance.

Based on the analysis of the results, we can conclude that: (i) dense rewards favor the emergence of essential behaviors, improving agent learning; (ii) knowledge distribution can decrease task complexity. The Color Balls environment showed better results with knowledge distribution. It happened because the task became less complex. Instead of associating the value of the integer variable with the ball tag, the agents now have to understand that if they are in uniform (boolean variable is true), they can pick up the ball; (iii) the dependency of the groups is an important factor in the knowledge distribution. In the Dungeon Escape environment, the key group showed lower results compared to the sword group due to their dependency. In the Survival, Color Balls and Wild World environments (*food_group* and *fire_group*), the groups showed very similar results, evidencing the importance of independence; and (iv) the framework allows the development of an environment in order to improve the learning performance of agents in multi-agent environments.

The main limitation of this work is the number of environments built and used for testing. At the time this paper was written, we had not yet developed environments that brought more elaborate challenges and incorporated different game mechanics. As future work, we intend to: (i) carry out new tests to analyze the impact of individual rewards (the agent is rewarded and not the group) on the performance of the group as a whole; (ii) build more complex environments; and (iii) seek a way to add communication between groups for environments with distributed knowledge.

Acknowledgments. The authors would like to thank NVIDIA, CAPES and FAPERJ for the financial support.

References

1. Bellemare, M.G., Naddaf, Y., Veness, J., Bowling, M.: The arcade learning environment: an evaluation platform for general agents. J. Artif. Intell. Res. **47**(1), 253–279 (2013)
2. Berner, C., et al.: Dota 2 with large scale deep reinforcement learning (2019)
3. Cohen, A., et al.: On the use and misuse of absorbing states in multi-agent reinforcement learning (2021)
4. Foerster, J., Assael, Y., Freitas, N., Whiteson, S.: Learning to communicate with deep multi-agent reinforcement learning. In: Proceedings of the 30th International Conference on Neural Information Processing Systems, pp. 2145–2153. NIPS'16, Curran Associates Inc. (2016)
5. Foerster, J., Farquhar, G., Afouras, T., Nardelli, N., Whiteson, S.: Counterfactual multi-agent policy gradients. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32, pp. 2974–2982. AAAI (2018)
6. Hessel, M., et al.: Rainbow: Combining improvements in deep reinforcement learning. In: Proceedings of the 32nd AAAI Conference on Artificial Intelligence, vol. 32, pp. 3215–3222. PKP Publishing Services Network (2018)
7. Iqbal, S., Sha, F.: Actor-attention-critic for multi-agent reinforcement learning. In: Proceedings of the 36th International Conference on Machine Learning, pp. 2961–2970. PMLR 97, Long Beach, California (2019)
8. Johnson, M., Hofmann, K., Hutton, T., Bignell, D.: The malmo platform for artificial intelligence experimentation. In: Proceedings of the 25th International Joint Conference on Artificial Intelligence, pp. 4246–4247. IJCAI'16, AAAI Press (2016)
9. Jorge, E., Kågebäck, M., Johansson, F., Gustavsson, E.: Learning to play guess who? and inventing a grounded language as a consequence (2016)

10. Kaelbling, L., Littman, M., Moore, A.: Reinforcement learning: a survey. J. Artif. Intell. Res. **4**, 237–285 (1996)
11. Kempka, M., Wydmuch, M., Runc, G., Toczek, J., Jaśkowski, W.: Vizdoom: A doom-based ai research platform for visual reinforcement learning. In: IEEE Conference on Computational Intelligence and Games (CIG), pp. 1–8 (2016)
12. Mnih, V., et al.: Playing atari with deep reinforcement learning (2013)
13. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**, 529–533 (2015)
14. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms (2017)
15. Summerville, A., et al.: Procedural content generation via machine learning (pcgml). IEEE Trans. Games **10**(3), 257–270 (2018)
16. Sunehag, P., et al.: Value-decomposition networks for cooperative multi-agent learning based on team reward. In: Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems. pp. 2085–2087. AAMAS '18, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2016)
17. Sutton, R., Barto, A.: Reinf. Learn.: Introduction. MIT Press, London, England (2018)
18. Vidhate, D., Kulkarni, P.: Enhanced cooperative multi-agent learning algorithms (ecmla) using reinforcement learning. In: 2016 International Conference on Computing, Analytics and Security Trends (CAST), pp. 556–561. IEEE (2016)
19. Vinyals, O., et al.: Grandmaster level in Starcraft II using multi-agent reinforcement learning. Nature **575**, 350–354 (2019)
20. Yannakakis, G., Togelius, J.: Artificial Intelligence and Games. Springer (2018). https://doi.org/10.1007/978-3-319-63519-4
21. Zhang, Q., Zhao, D., Lewis, F.: Model-free reinforcement learning for fully cooperative multi-agent graphical games. In: 2018 International Joint Conference on Neural Networks (IJCNN), pp. 1–6. IEEE (2018)
22. Zhao, Y., Borovikov, I., Rupert, J., Somers, C., Bierami, A.: On multi-agent learning in team sports games. In: Proceedings of the 36th International Conference on Machine Learning (ICML) (2019)