Centralized Critic per Knowledge for Cooperative Multi-Agent Game Environments

Thaís Ferreira
Instituto de Computação
Universidade Federal Fluminense
Niteroi, Brazil
thais_ferreira@id.uff.br

Esteban Clua
Instituto de Computação
Universidade Federal Fluminense
Niteroi, Brazil
esteban@ic.uff.br

Troy Costa Kohwalter
Instituto de Computação
Universidade Federal Fluminense
Niteroi, Brazil
troy@ic.uff.br

2021 20th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames) | 978-1-6654-0189-0/21/$31.00 ©2021 IEEE | DOI: 10.1109/SBGAMES54170.2021.00015

***Abstract*—Cooperative multiplayer games are based on rules where players must collaborate to solve certain tasks. These games bring specific challenges when using Multi-Agent Reinforcement Learning (MARL), since they present requirements related to the training of these collaborative behaviors, such as partial observations, non-stationary, and the problem of credit assignment. One of the approaches used in MARL to solve these challenges is centralized training with decentralized execution. The idea is to use the available knowledge about the full state and information of the environment in the training phase, but policy learning takes place in a decentralized way, not depending on this knowledge. In this work, we study the approach of centralized training with decentralized execution. We seek to validate whether the division of knowledge about the environment (e.g. observations, perception of objects, enemies, obstacles) by different groups (different centralized critics) improves learning performance in multi-agent environments. Our results show that specifying a centralized critic per knowledge improves the training, but it also increases the time of the training process.**

***Index Terms*—Cooperative Multi-Agents, Reinforcement Learning, ML-Agents, MA-POCA**

I. Introduction

The usage of modern Artificial Intelligence (AI) techniques in games has evolved due to advances in hardware and research. This evolution follows the constant growth of the gaming industry and the availability of new tools and frameworks. Besides that, one of the game genres that have been gaining prominence in the last year is multiplayer games. In games like Among Us and Deceit, the players must collaborate to solve the tasks at hand. In these multi-agent environments, the agents can learn from other agents' expertise and from their own experience. In this paper we intend to present how centralized critic per knowledge can impact cooperative multi-agent based games.

In these environments, there are restrictions so that some agents, at a given time, may not know everything about the world that others know [1]. Typically, agents have a partial observation, not having complete information about the environment and other agents' behavior, requiring them to communicate with their neighbors and learn the ideal policies based on observed local information [1]. Furthermore it is possible to have different agents that learn and adapt from each other's context in the environments.

This work is supported by CAPES and FAPERJ

These characteristics configure a non-stationary environment, as the behavior of other agents constantly changes during the training process [2]. Minor modifications in learned behaviors can result in random changes of macro properties resulting from the communication of these agents.

Reinforcement Learning (RL) techniques have attracted many researchers in the investigation of multi-agent environments [1] [2] [3] [4]. It has been used to solve the problem of how an autonomous agent that feels and acts in a given environment can learn to choose optimal actions to achieve its objective [5]. More recently it is used for building intelligent agents to solve complex challenges with multiple agents in gaming environments. For example, AlphaStar [3] which achieved the best professional player level performance in Starcraft II and OpenAI Five [4] that defeated the world champion in Dota II. Due to the results, Multi-Agent Reinforcement Learning (MARL) offers new possibilities for the development of intelligent agents in games with multiple artificial agents.

In the MARL domain, there are several agents who usually have their own private observation and want to take an action based on that specific set of observations. These observations are usually is constrained to a local issue and aren't necessarily the same as the full state of the environment. However, in a simulation environment such as games, we usually have access to the full state and information in the training phase. Therefore, it is possible to use this knowledge combined with decentralized execution, since the agents cannot have access to the full state during the execution phase. In decentralized policies, each agent selects its own action conditioned only on its local action-observation history [6].

Another challenge in MARL is the problem of credit assignment [7]. This problem occurs when agents who did not contribute to the task's resolution are rewarded. This type of practice can encourage behaviors that do not help with the task resolution, as an agent who executed a sub-optimal policy is being rewarded. We assume that in multi-agent learning it is necessary to find a way to give credit (reward) only to agents who contribute to achieving the goal. In cooperative settings, joint actions typically generate only global rewards, making it difficult for each agent to deduce its own contribution to the team's success [6].

2159-6662/21/$31.00 ©2021 IEEE
DOI 10.1109/SBGames54170.2021.00015

To solve these problems, the approach of centralized training with decentralized execution has recently been developed by the deep RL community ([8], [9]). Many of these works propose new techniques or try to improve existing algorithms. However, it is still necessary to analyze the impact of agents' knowledge on the centralized critic's coordination, especially in environments that require agents to handle with many observations and perceptions about the environment. Depending on the environment and the complexity of the tasks, it might be interesting to divide the observations and perceptions among different groups of agents, each one coordinated by a specific critic.

In our work we propose to study the approach of centralized training with decentralized execution. We seek to validate whether the division of knowledge about the environment (e.g. observations, perception of objects, obstacles, enemies) by different groups (different centralized critics) improves learning performance in multi-agent environments. We seek to analyze how the division of knowledge influences the gain of group rewards, the episode length, losses and policy statistics, and how much the increase in the number of centralized critics influences the time of the training process. We evaluated our approach through experiments using the MA-POCA [10] algorithm that allows centralized training with independent policy learning. Our results show that specifying a centralized critic per knowledge, in general, improves the process, but it also increases the time of the training process.

Our paper is organized as follows: Section 2 presents the concepts and terms about reinforcement learning and cooperative multi-agent tasks. Section 3 outlines the related work for this work, introducing some techniques used to solve the most common problems in MARL. It also describes the algorithm used in our experiments. Section 4 presents approaches for extending policy gradients in multi-agent settings, detailing the approach adopted in this work. Section 5 describes the test experiments conducted, introducing the environments and the configurations for the agents and neural network hyperparameters. Section 6 presents the analysis of the results. Finally, Section 7 concludes this work, listing contributions, limitations, and future work.

II. Background

A. Reinforcement Learning

RL is a subfield of machine learning that teaches an agent how to choose an action from e predefined action space. It interacts with an environment, in order to maximize rewards over time. In RL we have: (i) *agents* that is the program we train, with the aim of complete a specified task; (ii) *environment*, which is the world where the agent performs actions; (iii) *action*, that corresponds to things the agents can do and may cause changes in the environment; (iv) *rewards*, that is a feedback to the agent and can be positive or negative; and (v) *states* that the agent observes.

Agents learn in an interactive environment by trial and error using feedback (reward/penalties) from their actions and experiences. An agent essentially tries different actions on the environment and learns from the feedback that it gets back. The goal is to find the optimal results. The choice of which action to take in each state in order to get optimal results is known as the policy, π [11]. The policy is a function that maps states to the actions and can be approximated using neural networks (with parameters θ). In general, policies may be stochastic, specifying probabilities for each action [12].

The RL process involves iteratively collecting data by interacting with the environment (experiences). Traditionally, the agent observes the state of the environment s and takes action a based on policy $\pi(a|s)$. Then agent gets a reward r and is mapped to the next state s'. Collection of these experiences ($\langle s, a, r, s' \rangle$) is the data that agent uses to train the policy (parametersθ). Typically the experiences are collected using the latest learned policy, and then uses that experience to improve the policy in an interactive way. The agent may read data from the environment in order to collect the samples. The policy π_k can be updated with data collected by π_k itself. We optimise the current policy π_k and use it to determine what spaces and actions to explore and sample next. This means trying to improve the same policy that the agent is already using for action selection. The policy used for data generation is called behavior policy. In general, the behavior policy is equal to the policy used for action selection.

B. Cooperative Multi-Agent Task

A fully cooperative multi-agent task can be described as a stochastic game G, defined by a tuple $G = \langle S, U, P, r, Z, O, n, \gamma \rangle$, in which n agents identified by $a \in A \equiv \{1, ..., n\}$ choose sequential actions. The environment has a true state $s \in S$. At each time step, each agent simultaneously chooses an action $u^a \in U$, forming a joint action $u \in U \equiv U^n$ which induces a transition in the environment according to the state transition function $P(s'|s, u) : S \times U \times S \rightarrow [0, 1]$. All the agents share the same reward function $r(s, u) : S \times U \rightarrow \mathbb{R}$ and $\gamma \in [0, 1)$ is a discount factor.

We consider a partially observable setting, in which agents draw observations $z \in Z$ according to the observation function $O(s, a) : S \times A \rightarrow Z$. Each agent has an action-observation history $\tau^a \in T \equiv (Z \times U)^*$, on which it conditions a stochastic policy $\pi^a(u^a|\tau^a) : T \times U \rightarrow [0, 1]$. We represent joint quantities over agents in bold, and joint quantities over agents other than a given agent a with the superscript $-a$.

In actor-critic approaches, the actor (policy) is trained by following a gradient that depends on a critic, which usually estimates a value function [6]. The gradient must be estimated from trajectories sampled from the environment, and the (action-)value functions must be estimated with function approximators. Consequently, the bias and variance of the gradient estimate depend strongly on the exact choice of estimator [13]. In this paper, we train critics on-policy adapted to be used with deep neural networks.

III. Related Works

A naive approach to solve multi-agent problems is to use single-agent RL algorithms for each agent and treat other agents as part of the environment. One popular method is the Independent Q-Learning (IQL) [14], where each agent has one separate action-value function that gets the agent's local observation to select its action based on it. Unfortunately, this does not perform well in practice because the dynamic of the environment is constantly changing, being hard to approximate the Q-function [15].

Another possible solution is to centralise training and decentralise execution. There are several works that try to follow this method and can be divided into two groups: value-based methods (e.g. Value Decomposition Networks (VDN) and QMIX); and actor-critic methods (e.g. MADDPG and COMA). The first one tries to propose a way to be able to use value-based methods (e.g. Q-learning), train them in a centralized way and use them for decentralized execution.

Sunehag *et al.* [16] propose the usage of separate action-value functions for multiple agents and learn them by just one shared team reward signal. The joint action-value function is a linear summation of all action-value functions of all agents. Actually, by using a single shared reward signal, it tries to learn decomposed value functions for each agent and use it for decentralized execution. They used two centralized agents as baselines and perform the set of experiments on two-dimensional maze environments.

QMIX [17] is an extension to VDN [16] but tries to mix the Q-value of different agents in a nonlinear way. They use global state s_t as input to hypernetworks to generate weights and biases of the mixing network. Rashid *et al.* [17] evaluate QMIX on a challenging set of StarCraft II micromanagement tasks and show that QMIX significantly outperforms existing value-based multi-agent reinforcement learning methods.

Actor-critic-based methods try to use actor-critic architecture [13] to do centralized training and decentralized execution. Usually, they use the full state and additional information which are available in the training phase in the critic network to generate a richer signal for the actor. Foester *et al.* [6] propose the usage of policy gradient methods with a centralized critic and test their approach on a StarCraft micromanagement task. They learn a single centralized critic for all agents, learning discrete policies and combine recurrent policies with feed-forward critics.

Lowe *et al.* [18] suggest an approach similar to counterfactual multi-agent policy gradients [6] but learn a centralized critic for each agent, allowing for agents with differing reward functions including competitive scenarios. They used feed-forward policies and learning continuous policies. They also focus on the problem of micromanagement in StarCraft.

In our work, we use MA-POCA (MultiAgent POsthumous Credit Assignment) [10], which is a novel multi-agent trainer that trains a centralized critic. We may give rewards to the team as a whole, and the agents will learn how best to contribute to achieving that reward. Agents can also be given rewards individually, and the team will work together to help the individual achieve those goals. MA-POCA builds on previous work in multi-agent cooperative learning those we discussed before ([6], [18]).

IV. Centralized Critic per Knowledge

Our goal is to study the approach of centralized training with decentralized execution. We seek to validate whether the division of knowledge about the environment (e.g. observations, perception of objects, enemies, obstacles) by different groups (i.e., different centralized critics) improves learning performance in multi-agent environments. We use two environments divided into two different scenarios to achieve our goal.

In the first scenario of each environment, there is only one group being coordinated by a centralized critic. All agents in this group need to complete the same complex task and have the same knowledge about the environment. In the second scenario, the complex task is split into two new simpler tasks. There are two groups, each having a different task and knowledge about the environment. Each group is coordinated by a different centralized critic. Fig. 1 illustrates our approach. For a better understanding of the approach, the next subsections present methods to apply policy gradients to multiple agents.

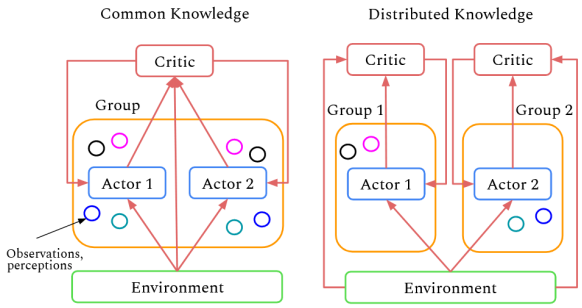

Fig. 1. Centralized critic with common knowledge and distributed knowledge.

A. Independent Actor-Critic (IAC)

The simplest way to apply policy gradients to multiple agents is to have each agent learn independently, with its actor and critic, from its action-observation history. Each agent has its policy network and is trained separately only using its experience [19]. IAC uses an actor-critic algorithm for each agent, treating other agents as part of the environment.

In this implementation, the learning process can be optimized by sharing parameters among the agents [6]. The agents can still behave differently because they receive different observations. The learning process remains independent because each agent's critic estimates only a local value function. Independent learning is one of the most straightforward approaches to MARL. However, the lack of information sharing at training time makes it difficult for an individual agent to estimate the contribution of its actions to the team's reward.

B. MultiAgent POsthumous Credit Assignment (MA-POCA)

The difficulties discussed above arise because, beyond the parameter sharing, IAC fails to exploit the fact that learning is centralized in our setting. For these reasons, we use MA-POCA, which overcomes this limitation and uses a centralized critic. In IAC, each actor $\pi(u^a|\tau^a)$ and each critic $Q(\tau^a, u^a)$ or $V(\tau^a)$ conditions only on the agent's own action-observation history τ^a. However, the critic is used only during learning, and only the actor is demanded during execution. Since learning is centralized, we can use a centralized critic that conditions the true global state s or the joint action-observation histories τ [6].

A naive way to use this centralized critic would be for each actor to follow a gradient based on the *temporal difference* (TD) error estimated from this critic. However, TD error considers only global rewards [12]. Thus, the gradient computed for each actor does not explicitly reason about how that particular agent's actions contribute to that global reward (the credit assignment problem). The gradient for that agent becomes very noisy since the other agents may be exploring [6].

Therefore, MA-POCA uses a baseline that marginalizes the action of the agent associated with its observations. The MA-POCA uses a network body that uses a self-attention layer to handle state and action input from a potentially variable number of agents that share the same observation and action space. This network body is used to compute the value and MA-POCA baseline for a variable number of agents in a group that all share the same observation and action space.

The POCA baseline calls a method in the network body that returns sampled actions. This method uses as parameters the agent observations. The POCA baseline uses parameters based on the observation of the agent, a tuple of observations and actions for all groupmates, and optional parameters about memory usage. The output is a tuple of reward stream to tensor and critic memories. Finally, a centralized value function calls the forward pass of the network body with only the states of all agents. It uses a list of observations for all agents in the group and returns a tuple of reward stream to tensor and critic memories.

V. Experiment

In this section, we present the analysis of the performance for multiple agents in two different environments through our approach of centralized critic per knowledge. The experiments are performed using the Unity ML-Agents Toolkit [20]. The flexibility of Unity enables the creation of tasks ranging from simple 2D grid-world problems to complex 3D strategy games, physics-based puzzles, or multi-agent competitive games. Unlike other research platforms, Unity is not restricted to any specific genre of gameplay or simulation, making it a general platform. Furthermore, the Unity Editor enables rapid prototyping and development of games and simulated environments.

The experiment was conducted in two 3D environments composed of multiple agents: Dungeon Escape environment and Survival environment. As in multiplayer games, agents need to work cooperatively to achieve a goal. This goal can be the same for all agents or not. In some games, it is necessary for one group to perform certain tasks while another group performs others. There are groups with different sub-goals but working cooperatively in the same environment.

Thus, we conducted our experiments in two environments with different scenarios. In the first scenario, there is only one group being coordinated by a centralized critic. All agents in this group need to complete the same complex task and have the same knowledge about the environment (they notice all objects, enemies, obstacles). This scenario is referred to as a *Common Knowledge* scenario. For the second scenario, the perceptions and observations of the group from the Common Knowledge approach were divided between two different groups. Each group has a specific behavior and is coordinated by a centralized critic. This scenario is referred to as a *Distributed Knowledge* scenario. In the following subsections, we introduce the two 3D environments and their scenarios.

A. Dungeon Escape Environment

In the Dungeon Escape environment, the episode completes successfully when the agents escape from the dungeon before the dragon escapes through the portal. The dragon starts at one end of the dungeon and must cross it to enter the portal and escape. It has specific behavior, programmed manually. Agents need to defeat the dragon, collect the key it drops and escape through a door that will appear once they picked the key. The agent needs to get one of the swords scattered around the map (there are two swords) to defeat the enemy. If an agent that doesn't have the sword collides with the dragon, that agent is removed from the episode and the enemy remains alive. If an agent holding the sword collides with the dragon, the agent is removed from the episode, but the enemy is defeated and drops the key. Since then, the other agents can collect the key and open the door that will appear, escaping the dungeon and completing the objective.

In both scenarios of this environment, agents have information about position, rotation, physics properties, and collider. Agents perceive the environment through ray cast sensors. These sensors can perceive an element through a tag every time a ray finds an object. If the tag of that object is specified in the sensor, the agent can perceive the object. The agents can perceive walls, the door, the dragon, other agents, the portal, sword, and key. Agents collect observations through an observation vector. ML-Agents Toolkit provides a fully connected neural network model to learn from those observations. These observations vary from the Common Knowledge scenario to Distributed Knowledge scenario. Fig. 2 shows the Dungeon Escape environment.

1) Common Knowledge Scenario Configuration: In the Common Knowledge scenario, which is composed of one group (*one_group*), all agents collect observations about who has the sword and who has the key. The agents' ray cast sensor contains both these object tags. In this scenario all four agents

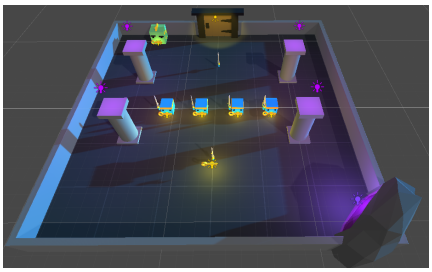

Fig. 2. Dungeon Escape environment.

can take the sword and the key. Once one sword is collected, the other disappears. The group is positively rewarded when the agent who holds the key collides with the door. It is the only time the agents are rewarded. Group rewards are meant to reinforce agents to act in the group's best interest instead of individual ones. The MA-POCA trainer rewards agents as a group and allows them to learn how their contributions helped achieve the goal. Even when an agent is removed in the middle of the episode, he is still able to learn how his actions affected the team's performance. This feature is extremely important for the development of intelligent agents in video games.

2) Distributed Knowledge Scenario Configuration: In the Distributed Knowledge scenario the agents were divided into two different groups. The *sword_group* is formed by two agents able to collect the sword, who have the same objective: to defeat the dragon. This group is positively rewarded only when it defeats the dragon. The *key_group* is formed by two agents able to collect the key and have the same objective: to open the door. This group is positively rewarded only when it opens the door. The *sword_group* ray cast sensor contains the sword tag but it does not contain the key tag. The *key_group* ray cast sensor contains the key tag but does not contain the sword tag. The *sword_group* observation vector collects information only about who has the sword. The *key_group* observation vector collects information only about who has the key. In this configuration, there is a centralized critic responsible for coordinating *sword_group* and another specific centralized critic for *key_group*.

B. Survival Environment

In the Survival environment, the episode completes successfully when the agents collect the necessary resources before the night arrives. This environment has twelve agents that need to collect water, food, and light two bonfires. To take the water, the agents need a bottle. To take the food the agents need a bag. Finally, to light the bonfire, the agents need to collect wood. If an agent tries to interact with a resource without possessing the item needed to collect it nothing will happen. To interact with the items/resources, agents need to collide with them. Thus, the initial goal of the agents is to move around and explore the map.

In both scenarios of this environment, agents have information about position, rotation, physics properties, and collisions obstacles. Agents perceive the environment through ray cast sensors. The agents can perceive walls, bags, bottles, woods, water, food, bonfire, trees, and other agents. Agents collect observations through an observation vector. These observations vary from the Common Knowledge scenario to Distributed Knowledge scenario. Fig. 3 shows the Survival environment.

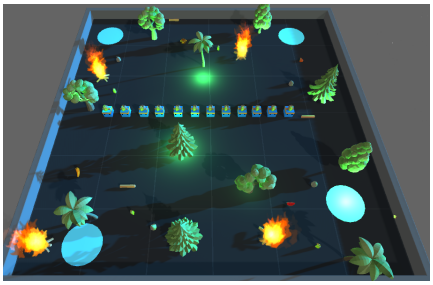

Fig. 3. Survival environment.

C. Common Knowledge Scenario Configuration

In the Common Knowledge scenario, which is composed of one group (*survival_one_group*), all agents collect observations about who has the bag, bottle, and wood. In this scenario, all twelve agents can take the bag, bottle, wood, water, food, and light the bonfires. The group is positively rewarded when the agents take two units of water, two foods, and light two bonfires. It is the only time the agents are positively rewarded. The group is negatively rewarded every step of the episode. The episode is restarted, and agents receive no rewards when the episode's duration ends before the agents complete the objective.

1) Distributed Knowledge Scenario Configuration: In the Distributed Knowledge scenario the agents were divided into three different groups. The *survival_food_group* is formed by four agents able to collect and perceive the bag and food. The *survival_food_group* observation vector collects information only about who has the bag. This group is positively rewarded only when it takes two foods. The *survival_water_group* is formed by four agents able to collect and perceive the bottle and water. The *survival_water_group* observation vector collects information only about who has the bottle. This group is positively rewarded only when it takes two units of water. The *survival_bonfire_group* is formed by four agents able to collect and perceive the wood and bonfire. The *survival_bonfire_group* observation vector collects information only about who has the wood. This group is positively rewarded only when it lights two bonfires.

Each group is negatively rewarded every step of the episode. The episode is restarted, and agents receive no rewards when the episode's duration ends before any group completes the objective. In the Distributed Knowledge scenario, there are three centralized critics responsible for coordinating each group.

D. Configuration File

We define the training hyperparameters for each behavior in the scene through a *configuration* file. In both environments

(Dungeon Escape and Survival), there is one complex behavior to be trained in the Common Knowledge scenario. In the Distributed Knowledge scenario, the complex behavior was divided into simple behaviors that will learn at the same time. For the Dungeon Escape environment, there are two behaviors, one for each group (*sword_group* and *key_group*). For the Survival environment, there are three behaviors, one for each group (*survival_food_group*, *survival_water_group* and *survival_bonfire_group*). In the *configuration* file, the only difference between the two scenarios (Common and Distributed Knowledge) is the number of neural networks that are trained. For each behavior, we have a different network.

The behavior parameters for training the neural network are composed of the observation space and the actions. As mentioned before, in both environments (Dungeon Escape and Survival), the observation space is formed by the raycast sensor capable of perceiving the objects through the specified tags and the observation vector (which agent has the sword/key/bag/bottle/wood). All agents have a raycast sensor, being able to perceive the environment according to the tags specified in the sensor.

The actions consist of one discrete action branch with seven possible actions: turn clockwise and counterclockwise, move along four different face directions (right, left, up, down), or do nothing. The agents learn through the cycle: collect observations, select an action using its policy, take an action, reset if it reaches the maximum number of steps or if it completes the task. For example, if an agent that has the key perceives the door (through the sensor) it must make the decision to move to the door. Once it collides with the door, its group is rewarded, and the agent learns that when it has the key it must search for the door and go to it.

The neural network simulates this behavior by learning about collected observations and then predicting outcomes. As the neural network learns, the weights of the connections between the neurons are "fine-tuned", allowing the network to come up with accurate predictions. All network hyperparameters are set to the same values as shown below.

The trainer type is set to POCA. In the common trainer configurations, the maximum number of model checkpoints to keep is set to 5. Checkpoints are saved after the number of steps specified by the `checkpoint_interval` option (the number of experiences collected between each checkpoint by the trainer). Once the maximum number of checkpoints has been reached, the oldest checkpoint is deleted when saving a new checkpoint.

The number of steps of experience to collect per agent before adding it to the experience buffer is set to 64; the number of experiences that need to be collected before generating and displaying training statistics is set to 60000. There are 2 hidden layers to the neural network. Each fully connected layer of the neural network has 256 units. We do not apply normalization to the vector observation inputs, because normalization can be harmful with simpler discrete control problems.

The extrinsic reward is configured as follows: 0.99 to gamma (discount factor for future rewards coming from the environment.) and 1.0 to strength (factor by which to multiply the reward given by the environment). The POCA hyperparameters are configured as follows:

- `batch_size` = 1024 (typical range: 512 - 5120). Corresponds to the number of experiences in each iteration of gradient descent;
- `buffer_size` = 10240 (typical range: 2048 - 409600). Corresponds to the number of experiences to collect before updating the policy model;
- `learning_rate` = 0.0003 (typical range: 1e-5 - 1e-3), which is the strength of each gradient descent update step;
- `beta = 0.01` (typical range: 1e-4 - 1e-2), which is the strength of the entropy regularization;
- epsilon = 0.2 (typical range: 0.1 - 0.3). This influences how rapidly the policy can evolve during training and corresponds to the acceptable threshold of divergence between the old and new policies during gradient descent updating;
- `lambd` = 0.95 (typical range: 0.9 - 0.95). Corresponds to the regularization parameter used when calculating the Generalized Advantage Estimate (GAE). This can be thought as how much the agent relies on its current value estimate when calculating an updated value estimate.
- `num_epoch` = 3 (typical range: 3 - 10). Corresponds to the number of passes to make through the experience buffer when performing gradient descent optimization.
- `learning_rate_schedule` = constant. Determines how learning rate changes over time. Constant learning rate keeps the learning rate constant for the entire training run. For POCA is recommend decaying learning rate until `max_steps` so learning converges more stably.

VI. Results

Our goal is to validate whether the division of knowledge about the environment by different groups (different centralized critics) improves learning performance in multi-agent environments or not. We want to analyze how the division of knowledge influences the (1) gain of group rewards, (2) the episode length, (3) the losses statistics, (4) the policy statistics, and (5) how much the increase in the number of agents coordinated by centralized critics influences the training process. Therefore, in the following subsections, we analyze some statistics saved throughout the training process related to each of those five factors.

A. Group Rewards

The ML-Agents Toolkit saves statistics during learning sessions that can be viewed with TensorBoard. *Group Cumulative Reward* is the mean cumulative episode reward overall groups and it is expected to increase during a successful training session. We divide the analysis by environments.

1) Dungeon Escape: Fig. 4 shows the resulting graph for the two training sessions (Common Knowledge and Distributed Knowledge scenarios) for the Dungeon Escape environment. The blue line represents the only group (*one_group*)

in the Common Knowledge scenario (all agents can take the sword, defeat the dragon, take the key and open the door). The orange line represents the *key_group* (which must collect the key and open the door) of Distributed Knowledge scenario, and the gray line represents *sword_group* (which must collect the sword and defeat the dragon) also from Distributed Knowledge scenario.

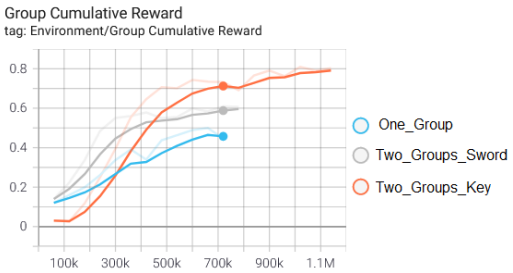

Fig. 4. The mean cumulative episode reward overall groups for Dungeon Escape environment.

It is possible to observe that until 360k steps, the group that accumulated the most rewards was the *sword_group* (rewarded for defeating the dragon). It occurs because this group is rewarded as soon as the dragon is defeated. Although (*one_group*) has four agents capable of performing this task, they are only rewarded after collecting the key and opening the door. Until 360k the *key_group* (rewarded by opening the door) is expected to perform less than the *sword_group*, as the *key_group* depends on the *sword_group* accomplishing its goal (defeating the dragon), in order to *key_group* accomplishing theirs (take the key that the dragon dropped and open the door).

Between 120k and 480k steps, the cumulative group reward grows a lot for the *key_group*. At this point, the *sword_group* reward begins to stabilize, indicating that this group is learning how to complete its objective, allowing the *key_group* to learn theirs. In general, the group cumulative reward for the *one_group* grows more slowly than the others. It occurs because this group needs to perform more tasks to achieve the goal and be rewarded.

2) Survival: Fig. 5 shows the resulting graph for the two training sessions (Common Knowledge and Distributed Knowledge scenarios) for the Survival environment. In this environment, we ran two tests for the Common Knowledge scenario, as we wanted to ascertain whether the group's accumulated reward would indeed be lower and more unstable. The orange and red lines refer to the tests in the Common Knowledge scenario. The blue, pink, and green lines refer to each of the three groups in the Distributed Knowledge scenario.

The tests of the Common Knowledge scenario showed lower mean cumulative rewards than the Distributed Knowledge scenario. It may be related to some factors such as the smaller number of tasks that the groups need to complete after the knowledge division, the more immediate reward due to this

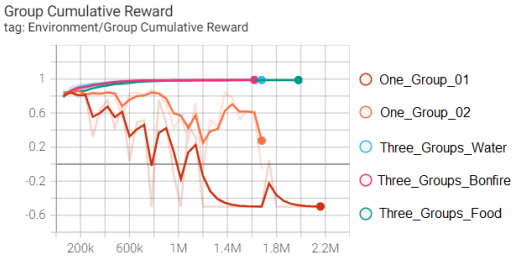

Fig. 5. The mean cumulative episode reward overall groups for Survival environment.

division, and the smaller number of agents coordinated by the same centralized critic. In the Distributed Knowledge scenario the groups maintain the reward gain stable and close to the maximum mean cumulative reward (1.0).

B. Episode Length

The episode length is the mean length of each episode in the environment for all agents. By analyzing the average episode duration, we can get a sense of whether or not the agents are taking longer to complete their tasks.

1) Dungeon Escape: Fig. 6 shows the *Episode Length* episode length for the Dungeon Escape environment. The episode length for the *sword_group* is always shorter compared to the other groups. It occurs because this group is rewarded more immediately, as the agents need to collect the sword and collide with the dragon. In *one_group* the agents need to collect the sword, defeat the dragon, get the key and open the door; and the *key_group* depends on the *sword_group* to complete its objective.

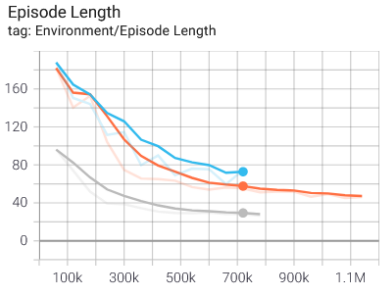

Fig. 6. Episode length for the Dungeon Escape environment.

2) Survival: Fig. 7 shows the *Episode Length* for the Survival environment. By analyzing it, we realized that the division of tasks and perceptions between the groups caused the agents to complete the tasks more quickly. In the tests of the Common Knowledge scenario the episodes evidenciates that the agents were taking more time to complete the tasks.

C. Losses

We analyzed the policy and value loss for the losses statistics. The policy loss is the mean magnitude of the policy loss function. It correlates to how much the policy (process for

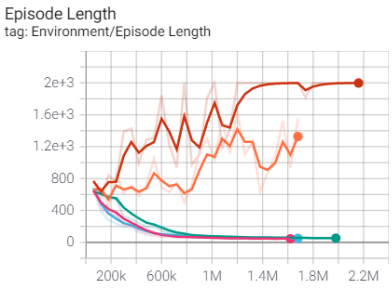

Fig. 7. Episode length for the Survival environment.

deciding actions) is changing. The policy loss measures how much the agent prediction was wrong. When the agent makes poor predictions, the policy loss will be high. It is expected that these values oscillate during training.

1) Dungeon Escape: Fig. 8 shows that the magnitude of the policy loss is decreasing during the training session for all groups in the Dungeon Escape environment. It means the agents' predictions (predict what the best action would be in the current situation) are getting better.

Fig. 8. The mean magnitude of the policy loss function for the Dungeon Escape environment.

The value loss is the mean loss of the value function update. It correlates to how well the model can predict the value of each state. The value indicates the long-term desirability of states after taking into account the states that are likely to follow and the rewards available in those states [17]. This should increase while the agent is learning, and then decrease once the reward stabilizes. Fig. 9 shows that the value loss for the *sword_group* is higher than the other groups. In the initial steps, agents in this group are learning faster, which corresponds to the group with the greatest reward in the initial steps.

2) Survival: Fig. 10 shows that the magnitude of the policy loss, in general, is decreasing during the training session for all groups in the Distributed Knowledge scenario. It means the agents' predictions are getting better. In the Common Knowledge scenario, we can perceive that the orange line started at 0.023 but is increasing over time. It means the agents' predictions are not getting better. This may be related to the number of agents coordinated by the centralized critic, where only one is coordinating all twelve agents.

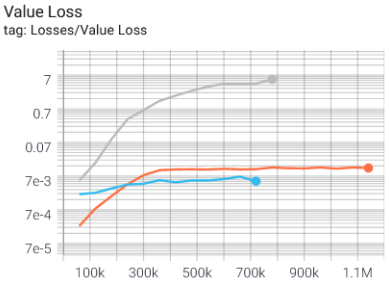

Fig. 9. The mean loss of the value function update for the Dungeon Escape environment.

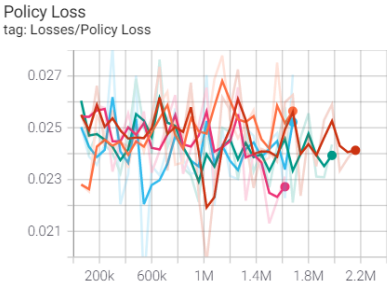

Fig. 10. The mean magnitude of the policy loss function for the Survival environment.

The value loss should increase while the agent is learning, and then decrease once the reward stabilizes. Fig. 11 shows that the value loss remains varying in the Common Knowledge environment. It happens because the agents remain to make random decisions and are not learning satisfactorily. On the other hand, in the Distributed Knowledge scenario, we notice that the values increase a little at the beginning because the agents of the three groups are learning. A little later, they manage to reach rewards close to the maximum value and we notice that the value loss starts to fall, indicating that the agents are learning to perform the tasks.

Fig. 11. The mean loss of the value function update for the Survival environment.

D. Policy

For the policy statistics, we analyze the entropy and the value estimate. The entropy means how random the decisions

of the model are. It is expected to slowly decrease during a successful training process. At the beginning of the training, the agents' decisions are more random, however, as the training progresses, they learn and their actions become fewer random.

1) Dungeon Escape: Fig. 12 shows the graph for entropy. All three groups take fewer random actions during the training session because they are learning. The value estimate is the mean value estimate for all states visited by the agents. These values should increase as the cumulative reward increases. They correspond to how much future reward the agent predicts itself receiving at any given point. In Fig. 13 the value estimate increase as the cumulative reward increases. In the 300k step, the *key_group* cumulative reward starts to increase more than the *one_group* cumulative reward.

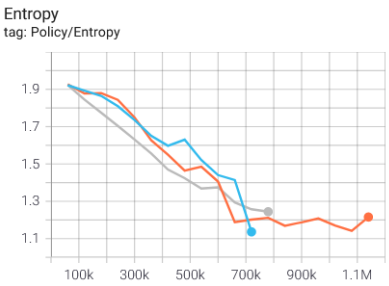

Fig. 12. Entropy for the Dungeon Escape environment.

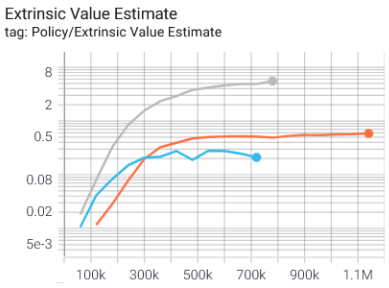

Fig. 13. Extrinsic value estimate for the Dungeon Escape environment.

2) Survival: Fig. 14 shows the graph for entropy. The entropy for the Common Knowledge scenario decreases slowly, because the model make more random decisions than in the Distributed Knowledge scenario.

Fig. 15 shows the value estimate for Survival environment. As seen in the cumulative reward graph, all three groups in the Distributed Knowledge scenario achieved high mean cumulative rewards. As the reward gain increasing, the extrinsic value estimate increasing too, since it corresponds to how much future reward the agent predicts itself receiving at any given point.

E. Considerations

In the Dungeon Escape environment, the cumulative reward group was better in Distributed Knowledge scenario, where

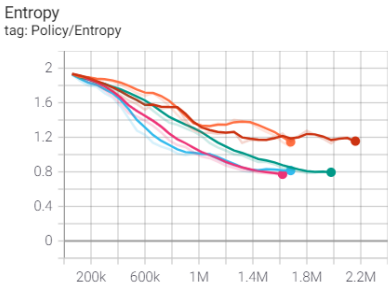

Fig. 14. Entropy for the Survival environment.

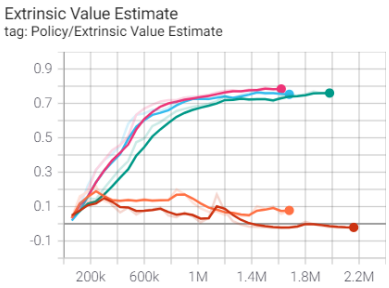

Fig. 15. Extrinsic value estimate for the Survival environment.

each group performs specific tasks and has different goals. Regarding entropy, both scenarios showed a gradual decrease in the randomness of the taken actions. Distributed Knowledge scenarios also had the highest values for model prediction.

In the Survival environment, the differences are more evident. The cumulative reward group was much better in Distributed Knowledge scenario, where each group performs specific tasks and has different goals. Distributed Knowledge scenario also shows better results for episode length, policy loss, value loss, entropy, and value estimate. Distributed Knowledge scenarios also had the highest values for model prediction.

It indicates that sharing knowledge among different groups in the same environment can bring better results. Agents learn to find rewards more efficiently, making less random decisions in fewer steps, making more accurate predictions, and decreasing the time it takes to complete tasks. This improved performance may be related to the smaller number of agents being coordinated by the centralized critics in the Distributed Knowledge scenario. Also, in this scenario, agents have fewer observations and perceptions about the environment, so the neural network does not have to deal with as many inputs.

We can conclude that specifying a centralized critic per knowledge, in general, improves the training process. However, the test execution time for the Distributed Knowledge scenarios was much higher. The execution time of the training process in the Dungeon Escape environment takes 1 hour and 13 minutes to reach 720k steps in the Common Knowledge scenario. In Distributed Knowledge scenario it takes 2 hours and 57 minutes. Both scenarios showed good results, with an

improvement in the Distributed Knowledge scenario.

The execution time of the training process in the Survival environment takes 2 hours and 6 minutes to reach 1.62M steps in the Common Knowledge scenario. In Distributed Knowledge scenario it takes about 11 hours. However, up to 1.62M steps, only the Distributed Knowledge scenario showed good results. The rewards and other factors remained very poor for the Common Knowledge scenario. The training runtime for the Distributed Knowledge scenario is due to the larger number of neural networks (each behavior has one neural network). In the Survival environment, there are three neural networks in the Distributed Knowledge scenario and only one in the Common Knowledge scenario. We can conclude that distributed knowledge is more appropriate when there are many agents in the same environment, which occurs in many games.

VII. Conclusion

In many cooperative multiplayer games, players must collaborate to solve certain tasks. However, these environments present challenges related to the training of these collaborative behaviors, such as partially observations, non-stationary, and the problem of credit assignment. One of the approaches used in MARL to solve these challenges is centralized training with decentralized execution. The idea is to use the available knowledge about full state and information of the environment in the training phase, but policy learning takes place in a decentralized way, not depending on this knowledge.

In this work, we proposed studying the framework of centralized training with decentralized execution in two different environments. We validated whether the division of knowledge about the environment (e.g. observations, perception of objects, obstacles, enemies) by different groups (different centralized critics) improves learning performance in multi-agent environments. We analyzed how the division of knowledge influenced the (1) gain of group rewards, (2) the episode length, (3) the losses statistics, (4) the policy statistics, and (5) how much the increase in the number of agents coordinated by centralized critics influences the training process.

For the experiments, we use the POCA trainer, an approach that allows training policies in a centralized way, but each policy acts independently during the inference. The experiments are performed using the Unity ML-Agents Toolkit. The results show that specifying a centralized critic per behavior improves the training process but also increasing the time execution. We can conclude that distributed knowledge is more appropriate when there are many agents in the same environment. Also, since each agent has its actor-critic with its local observations and perceptions, decreasing the number of observations makes it easier for the centralized critic to coordinate the independent actors.

Acknowledgment

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

References

[1] D. Vidhate and P. Kulkarni, "Enhanced cooperative multi-agent learning algorithms (ecmla) using reinforcement learning," in *International Conference on Computing, Analytics and Security Trends (CAST)*, pp. 556–561, IEEE, 2016.

[2] Q. Zhang, D. Zhao, and F. L. Lewis, "Model-free reinforcement learning for fully cooperative multi-agent graphical games," in *International Joint Conference on Neural Networks (IJCNN)*, pp. 1–6, IEEE, 2018.

[3] O. Vinyals, I. Babuschkin, W. Czarnecki, and et al., "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, pp. 350–354, 2019.

[4] OpenAI, C. Berner, G. Brockman, and B. C. et al., "Dota 2 with large scale deep reinforcement learning." arXiv:1912.06680, 2019.

[5] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237—-285, 1996.

[6] J. N. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," in *AAAI Conference on Artificial Intelligence (AAAI)*, pp. 2974–2982, AAAI, 2018.

[7] Y. Chang, T. Ho, and L. Kaelbling, "All learning is local: Multi-agent learning in global reward games," in *Advances in Neural Information Processing Systems*, pp. 807–814, MIT Press, 2004.

[8] J. Foerster, Y. M. Assael, N. de Freitas, and S. Whiteson, "Learning to communicate with deep multi-agent reinforcement learning," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, p. 2145–2153, Curran Associates Inc., 2016.

[9] E. Jorge, M. Kageback, and E. Gustavsson, "Learning to play guess who? and inventing a grounded language as a consequence." arXiv:1611.03218, 2016.

[10] Unity. https://github.com/Unity-Technologies/ml-agents/blob/main/ml-agents/mlagents/trainers/poca/trainer.py, 2021. Accessed: 2021-06-18.

[11] S. Marsland, *Machine Learning An Algorithmic Perspective*. Boca Raton, FL: Chapman & Hall/CRC, 2015.

[12] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[13] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Advances in Neural Information Processing Systems*, p. 1008–1014, 2000.

[14] M. Tan, *Multi-agent reinforcement learning: Independent vs. cooperative agents*, pp. 487–494. San Francisco, CA: Morgan Kaufmann Publishers Inc., 1997.

[15] L. Matignon, G. J. Laurent, and N. Le Fort-Piat, "Independent reinforcement learners in cooperative markov games: a survey regarding coordination problems," *Knowledge Engineering Review*, vol. 27, no. 1, pp. 1–31, 2012.

[16] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, and T. Graepel, "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*, AAMAS '18, (Richland, SC), pp. 2085–2087, International Foundation for Autonomous Agents and Multiagent Systems, 2016.

[17] T. Rashid, M. Samvelyana, C. S. Witt, G. Farquhar, J. Foerster, and S. Whiteson, "Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning," in *In Proceedings of the International Conference on Machine Learning*, pp. 4292–4301, 2018.

[18] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments." arXiv:1706.02275v4, 2020.

[19] F. Christianos, L. Schäfer, and S. V. Albrecht, "Shared experience actor-critic for multi-agent reinforcement learning," in *Proceedings of the 34th Conference on Neural Information Processing Systems (NeurIPS)*, pp. 10707–10717, 2020.

[20] A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar, and D. Lange, "Unity: A general platform for intelligent agents." arXiv:1809.02627v2, 2020.