May 2022

Testing Reinforcement Learning Explainability Methods in a Multi-agent Cooperative Environment

Marc DOMÈNECH I VILA [a] Dmitry GNATYSHAK [b] and Sergio ALVAREZ-NAPAGAO [b]

[a] *Universitat Politècnica de Catalunya, Barcelona, Spain*

[b] *Barcelona Supercomputing Center, Barcelona, Spain*

Abstract. Even though with each passing day the AI gains popularity thanks to its successful application in many domains, the truth is that it also receives a lot of criticism. In particular, people ask themselves if its decisions are well-informed and if they can rely on its decisions. The answers to these questions become crucial in cooperative environments to be understandable to humans and can cooperate with them. In this work, we will apply an approach for explainability based on the creation of a Policy Graph (PG) that represents the agent's behaviour. This work has two main contributions: the first is a way to measure the similarity between the explanations and the agent's behaviour, by building another agent that follows a policy based on the explainability method and comparing the behaviour of both agents. The second manages to explain an RL agent in a multi-agent cooperative environment.

Keywords. Explainable AI, Reinforcement Learning, Policy Graphs, Multi-agent Reinforcement Learning, Cooperative Environments

1. Introduction and Motivation

Humans and complex algorithms for controlling agents do not usually share a common language. In the context of many AI methods, including those based on Machine Learning (ML), when evaluating a specific problem, we can get metrics like accuracy, reward or loss with respect to an objective function, etc. These metrics can give us an idea about whether an algorithm controlling an agent, such as a robot, is learning to perform a certain task correctly or not. However, these metrics are frequently not enough to understand the agent's behaviour or why the agent is taking a certain action. This is known as the explainability problem.

For this reason, an increasing amount of research is being done on how we can develop intelligent agents that are more understandable to humans. This is one of the many objectives of a field known as Explainable Artificial Intelligence (XAI)[1]. This concept has been gaining awareness in the last years since the more complex a black-box model is, the more difficult it becomes to understand how it makes decisions, especially when we talk about ML methods like RL, where the system learns autonomously. From these scenarios, it becomes even more necessary to understand the reasoning behind their decisions.

This paper aims to continue the line of research opened in [2]. In this work, it was possible to explain the behavior of an RL agent in an environment such as Cartpole, following the work made in [3]. In our case, we will focus on a sub-field of Reinforcement Learning called Multi-Agent Reinforcement Learning (MARL). We intend to go a step further and see if it is possible to apply the same methodology used in Cartpole, to explain the behavior of an agent in a more complex environment that requires cooperation. We believe that these scenarios are very interesting because the agent has to consider the other agents to cooperate. This means that we could explain things like: what role plays each in the game? why does the agent decide to cooperate? etc.

Currently, there are several approaches to explain RL agents. In this work, we briefly overview some of them in **Section 2**, we choose one of them that builds a graph that represents the agent's behaviour and we apply it to a MARL environment in **Section 4**. Once we apply the method, we will build a new agent using the graph as a policy in order to compare both agents in **Section 5**. Next, we will talk about giving explanations in **Section 6** and finally, we end with a summary of the main conclusions and contributions from the work done in **Section 7**.

2. State of the art

Over time, different techniques have been proposed to face this problem. In this section, we will provide a short overview of some of them. In addition, we will talk in more depth about the method that we will use in this work. A more detailed study of the explainability methods in RL can be found in [4,5].

There are different ways to classify these methods. It is quite common to classify them into three large groups: those that use another method to generate explanations while keeping the original model (Post-Hoc), those that introduce a new interpretable learning model (Intrinsic), or those that combine both methods by altering the original model to improve the interpretability instead of replacing it entirely. Furthermore, it is also usual to subdivide these approaches by the scope of their explanations into global and local ones. The former shows the global strategy used by the agent, while the latter can explain the policy's actions locally on a case-by-case basis.

Before going deeper, it should be noted that in all these classifications, multiple metrics can be produced during and after the agent construction. For example, in [6] the authors proposed to collect 3 levels of performance data: environment analysis, interaction analysis and meta-analysis. For each level, they have outlined a wide range of statistics and metrics that can provide useful information about the agent's performance and the influence of the environment on it.

One of the most classical examples of a global intrinsic approach is the Decision Tree model even with the new methods and their variations [7,8]. Here, the agent only has to answer the questions from the root to the bottom of the tree in order to decide which action to take. Although these models are meant to be understandable by design, the fact is that using these models in a complex problem can lead to an enormous tree that is too complex to be analysed as a whole. This forces us to maintain a balance between precision and explainability. There are multiple approaches to deal with these issues. For example, [7] proposes to generate decision trees using NN models. Another proposal talks about using differentiable decision trees [8], whose training can be updated incrementally, using some algorithm such as gradient descent.

Another example of a global intrinsic approach is building an agent policy with some high-level domain-specific programming language [9]. Such programmatic policies have the benefits of being more easily interpreted than NN and being amenable to verification by symbolic methods. This method is based on the idea of learning policies that are represented in a human-readable language.

If we talk about environments with visual observations, we can find a large number of saliency map-based methods. These methods can represent both local intrinsic and local post-hoc explainability approaches. For instance, a method proposed in [10] in addition to selecting an action, also generates intrinsic importance maps for the pixels of the image. Alternatively, the classical saliency maps described in [11] use post-hoc back-propagation to find the maps. On the other hand, [12] proposes a perturbation-based approach similar to the well-known LIME algorithm for deep NN in order to generate saliency maps. It is also true that you have to be careful with these types of algorithms since recent studies have demonstrated their propensity for adversarial attacks [13].

In order to get global post-hoc explanations, we can analyze the behaviour of the trained policy. By doing this analysis, we can identify the most interesting states or situations showing the behaviour of the agent. Different metrics and approaches can be used to select these execution traces. For example, [14] defines a state importance measure as the difference between the discounted reward values for the best and the worst action choice in this state. Thanks to this definition, we can reduce the number of shown traces. On the other hand, [15] proposes looking for critical states which are defined as states for which choosing a random action is significantly worse than choosing a specific one in terms of reward. Further analysis of this family of methods can be found in [16].

Finally, there are some methods that create simplified representations of the policy or the environment in order to use them to generate local explanations. These methods are on the border between global post-hoc and local post-hoc. For example, in [17,18] authors use NN to generate a graph representation of scenes (images, for instance) that can be later used by a reasoning engine. Another approach consists in building a Markov Decision Process and follow the graph from the input state to the main reward state [19]. This allows us to ask simple questions about the chosen actions. As an optional step, we can simplify the state representation (discretizing it if needed). This step becomes crucial when we are talking about more complex environments [4]. The latter approach is the one we will use in our work. It consists of building a policy graph by mapping the original state to a set of predicates (discretization step) and then repeatedly running the agent policy, recording its interactions with the environment. This graph of states and actions can then be used for answering simple questions about the agent's execution which is shown in Section 6.

3. Training agents in a cooperative environment: Overcooked-AI

In this paper, we have used PantheonRL[21] package for training and testing an agent in Overcooked-AI[20]. Overcooked-AI is a benchmark environment for fully cooperative human-AI task performance, based on the wildly popular video game Overcooked[1]. The goal of the game is to deliver soups as fast as possible. Each soup requires placing up to 3 ingredients in a pot, waiting for the soup to cook, and then having an agent pick up the soup and delivering it. The agents should split up tasks on the fly and coordinate effectively in order to achieve high reward. The environment has the following reward function: 3 points if the agent places an onion in a pot or if takes a dish, and 5 points if it takes a soup. Here in this work, we have worked with five different layouts: *simple*, *unident_s*, *random0*, *random1* and *random3* (**Figure 1**).

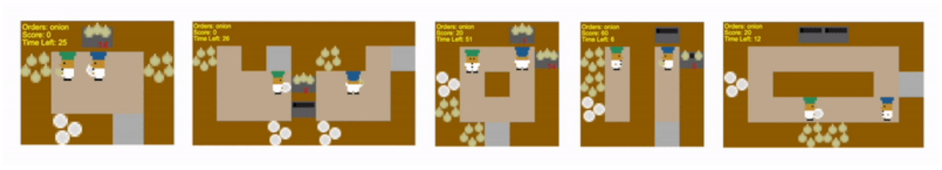

Figure 1. Overcooked layouts (*simple, unident_s, random1, random0, random3)*

The aim of our work is not to solve the Overcooked game but rather to analyze the potential of explainability in this cooperative setting. Therefore, we do not really care about what method is used to train our agent. However, it is important that the agent performs reasonably well in order to verify that we are explaining an agent with a reasonable policy. Therefore, we have used the PPO method because is one of the methods that has achieved the best results in [20]. Indeed, if we get good results with PPO, we should get good results with other methods since this explainability method is independent of the RL method used. In our case, we have trained five different agents (one for each layout) for 1M total timesteps and with a episode length of 400. The results are the following:

- *simple*: Mean Episode Reward = 387.87 and Standard Deviation = 25.33
- *unident_s*: Mean Episode Reward = 757.71 and Standard Deviation = 53.03
- *random0*: Mean Episode Reward = 395.01 and Standard Deviation = 54.43
- *random1*: Mean Episode Reward = 266.01 and Standard Deviation = 48.11
- *random3*: Mean Episode Reward = 62.5 and Standard Deviation = 5

4. Building a policy graph for the trained agent

As we mentioned in **Section 2**, we have chosen a method based on the creation of a policy graph. This method consists of building a directed graph where each node represents a state, and each edge represents the transition going from one node to another taking a specific action. But before starting building the graph, we have to formalize the necessary elements.

[1]http://www.ghosttowngames.com/overcooked

4.1. State representation

At each timestep, the environment returns: a list with the objects not owned by the agent present in the layout and, for each player, the position, orientation, and object that it is holding and its information. Moreover, we can also get at the start of the game the location of the basic object sources (dishes, onions, etc.).

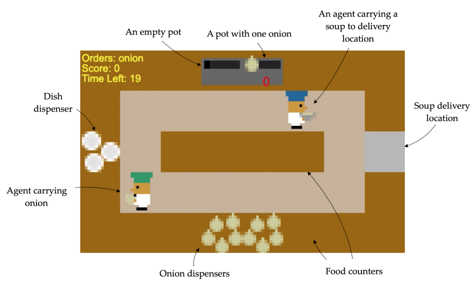

Figure 2. Overcooked-AI game dynamics.

For example, the agent would receive the following data from **Figure 2**.

- **Player 1**: Position (5, 1) - Facing (1, 0) - Holding Soup
- **Player 2**: Position (1, 3) - Facing (-1, 0) - Holding Onion
- **Not owned objects**: Soup at (4, 0) - with 1 onion and 0 cooking time.

4.2. Discretization

In order to build the policy graph, we need to discretize the state representation. To do it, we have created a set of predicates to represent each state, this step is crucial since we need to map each state to a node in our PG. At first, we proposed 2 basic predicates: the first one is *held* that has 5 possible values depending on which object the agent is holding (Example: "O" for "Onion" object or "*" if is not holding anything). The second one is *pot_state* that has 4 possible values depending on the state of each pot:

- "Of", when [pot.onions $= 0$].
- "Fi" , when [pot.onions $= 3 \wedge$ pot.timer $= 20$].
- "Co" , when [pot.onions $= 3 \wedge$ pot.timer < 20].
- "Wa" , when [pot.onions < 3].

Later, we realized that this proposal led to a drawback. In this representation, we are not storing any information related to the position of the objects on the map. This causes that we cannot relate the actions we execute with the relative position of the objects. For this reason, we decided to add 6 new predicates: *onion_pos(X)*, *tomato_pos(X)*, *dish_pos(X)*, *pot_pos(X)*, *service_pos(X)* and *soup_pos(X)*. All of them have the same possible values, the only change is the object to which it refers. Then, each of this predicates can have 6 possible values depending on the next action to perform to reach the object quickly (Example: "T" for "Top" action).

Finally, we decided to add two more predicates to represent in some way information that can help the agent to cooperate. The first one is *partner_zone* that has 8 possible values depending on which direction (cardinal point) the partner is located (Example: "NE" for "North East"). The second one is *held_partner* that is the same predicate as *held* but about the partner.

4.3. Algorithm for building the policy graph

As we mentioned in **Section 2**, the aim of the PG algorithm is to record all the interactions of the original agent with the environment an build graph like in **Figure 3**. This work has two approaches regarding this algorithm.

- **Partial Policy Graph**: This algorithm builds a directed graph. For each state, it takes the most probable action. Therefore, we do not add all the agent interactions to our graph, we only pay attention to the interactions that belong to the most used action by the agent. This means that for each node, we only have one possible action (the most probable in this case). We think this could be an interesting approach since we are building a deterministic agent. Probably this algorithm can work well in simpler layouts where decision making is easier.
- **Complete Policy Graph**: This algorithm builds a multi-directed graph. For each state, it takes all the agent interactions. Therefore, we add them all to our graph. This means that for each node, we have multiple possible actions. We think this could be an interesting approach since we are maintaining stochasticity. Probably this algorithm can work better in more complex layouts where decision making is more relative.

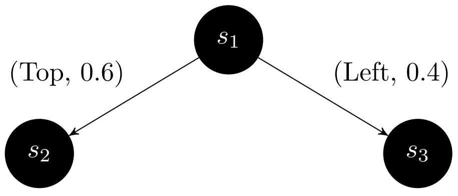

Figure 3. PG Intuition. Nodes S_i represent states and edges represent a transition between 2 states taking an action with some probability.

5. Validating the policy graph

In this section, we will study how to validate our Policy Graph. To do it, we built another agent that follows a policy based on the PG we obtained as explained in **Section 4**. What we will do is run the new agent multiple times in the environment. At each timestep, the agent will receive a state, and he has to be capable of taking an action following his PG. At his point, if, for example we use a discretizer with all the mentioned predicates in *simple*, this means that we have nine predicates. These nine predicates mean that there are potentially 7,464,960 different states.

Due to the large number of possible states, it is very likely that the new agent will find states never seen before. For this reason, is very important to set up a strategy to deal with these situations.

5.0.1. State similarity

In this work, we have introduced state similarity to deal with unknown states. We consider that two states (S_1, S_2) are similar when $\textit{diff}(S_1, S_2) <= 1$, where $\textit{diff}(S_1, S_2)$ computes the number of different predicate values one each other. For example, if we have the states $S_1 =$ O-Co-S and $S_2 =$ O-Co-N, then $\textit{diff}(S_1, S_2) = 1$. We have considered that when there is a difference greater than 1, the two states become totally different.

5.1. Partial vs Complete PG

As we saw in **Section 4**, we are testing two different algorithms. For this reason, we will build two new agents (for each layout). Now, we will see how each of these algorithms acts when making decisions. To do so, imagine we are in state S, we can differentiate 3 cases:

1. $S \in PG$: Picks an action using weights as a probability distribution.
2. $S \notin PG$, but similar state found: Same as case 1 but using similar state.
3. $S \notin PG$ and similar state not found: Pick a random action.

5.2. Results

Now, we will see the results we have obtained. All the agents has been trained using batches of 25 seeds. Altogether, the training consisted in 500 seeds and 3 episodes per seed. In order to validate the new agent, we have generated 3 metrics. The first one is the Transferred Learning (TL) that is the division of the RL agent average episode reward (AER) by the new agent AER. The second one is the Standard Deviation (STD) that is the division of the RL agent average standard deviation (ASTD) by the new agent ASTD. Finally, the third one is New States (NS) that is the division of the number of unknown visited states by the total number of visited ones. In order to do comparisons, we have tested both PG algorithms using multiple discretizers (D11, D12, D13 and D14). D11 has the predicates *held*, *pot_state* and *predicate_pos*, D12 has D11 predicate plus *held_partner*, D13 has D11 predicate plus *partner_zone* and D14 has all the predicates.

Figure 4 shows the results we have obtained from the different agents. First, we will look at the *simple* and *unident_s* scenarios. We can see that in both cases the Partial agents manage to outperform the original ones while the Complete agents do not manage to reach 100% of TL. On the other hand, it seems that in this case it is not necessary to introduce cooperative predicates to explain their behavior, unlike the *random0* and *random3* scenarios. For instance, in *random0* the Partial agent needs the predicate *partner_zone* and in *random3* the predicate *held_partner* to get good results. Secondly, we can see how in *random1*, the Partial agent is not even able to score even though the Complete algorithm scores quite well. Regarding the standard deviation of these agents, it is usually higher than what we had originally. We cannot lose sight of the fact that the PG we have

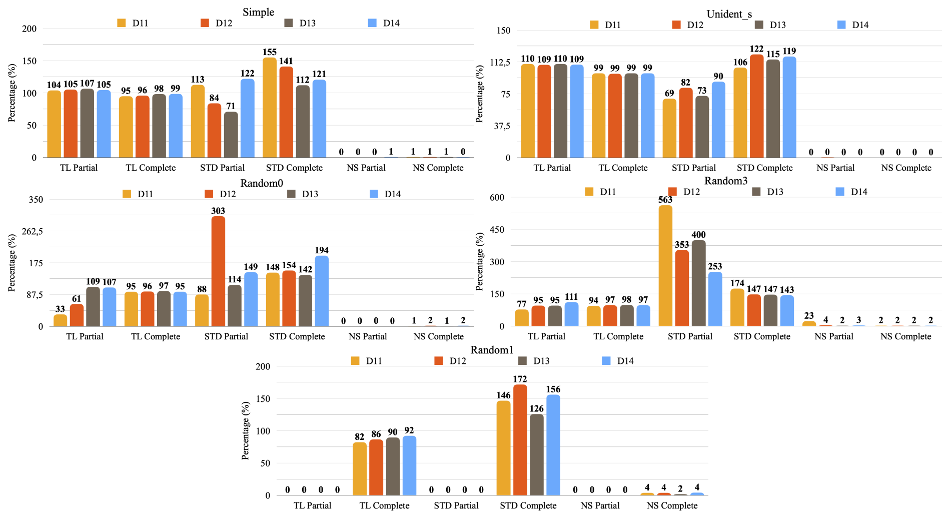

Figure 4. Results after testing both PG algorithms with discretizers D11, D12, D13, D14 for layouts *simple, unident_s, random0, random1, random3*

.

built is based on the simplification of states and actions, so the policy also ends up being a simplification. Finally, we can see how the vast majority of agents achieve a really low NS percentage, which means that the agent knows exactly what action to take most of the time.

To summarise, according to the results we have seen in **Figure 4**, we can conclude that the Complete algorithm is more stable than the Partial one. The latter is capable of obtaining even better results than the original ones and at the same time, it is capable of getting zero points depending on the layout. This behaviour is probably due to the deterministic nature of the algorithm. On the other hand, we have also seen that although there are scenarios where it is not necessary to introduce cooperative predicates to explain its behaviour, there are others where this information is determinant to be able to give explanations.

6. Explaining Overcooked

Once we have built our PG, we have access to the generation of natural language explanations as shown in [4], in where the authors validate the PG by comparing the sentences generated by the algorithm against sentences written by human experts. While this is valid as a qualitative approach for validation, it becomes obvious that this method heavily depends on experts and on the nature of the specific domain.

In this work, we have followed [4]. In it, the authors propose three questions to answer in order to explain the environment. These questions and answers are:

1. **What will you do when you are in state X?**: Look for the possible actions in the policy graph from the input state that the user wants to check.

2. **When do you perform action X?**: Look for these states where the action X is the most probable action.
3. **Why did not you perform action X in state Y?**: Look at which state I would reach if I had chosen action X in state Y. Once we have those nearby (similar) states, then we compute the difference between state Y and the nearby state.

7. Conclusions

XAI is a research area that is growing by leaps and bounds in recent years, especially in the field of RL. This research can be key when studying the quality of an agent's reasoning or even help in the interaction between humans and this type of models. In fact, there are already some works that attempt to answer to some of these aspects. However, it is necessary to test these methods to see how effective they are in practice.

In this work, we have used an explainability method based on the construction of a PG by discretizing the state representation into predicates for later apply it to a cooperative MARL environment (Overcooked). We have proposed 2 different algorithms to generate the PG and we have managed to give some explanations that according to [4] should be validated by human experts with domain-specific knowledge. In order to validate the PG, we have proposed a new method to test automatically different polcies by building a new agent that follows a policy based on the PG. We also have seen that the introduction of cooperative predicates, could be crucial in order to explain Multi-Agent cooperative environments.

References

[1] Longo L, Goebel R, Lecue F, Kieseberg P, Holzinger A. Explainable Artificial Intelligence: Concepts, Applications, Research Challenges and Visions. Lect

[2] Climent A, Gnatyshak D, Alvarez-Napagao S. Applying and Verifying an Explainability Method Based on Policy Graphs in the Context of Reinforcement Learning. Front Artif Intell Appl [Internet]. 2021 Oct 14;339:455–64. Available from: https://ebooks.iospress.nl/doi/10.3233/FAIA210166

[3] Hayes B, Shah JA. Improving Robot Controller Transparency Through Autonomous Policy Explanation. In: 2017 12th ACM/IEEE International Conference on Human-Robot Interaction (HRI. 2017. p. 303–12.

[4] Alharin A, Doan TN, Sartipi M. Reinforcement learning interpretation methods: A survey. IEEE Access. 2020;8:171058–77.

[5] Puiutta E, Veith EMSP. Explainable Reinforcement Learning: A Survey. Lect Notes Comput Sci (including Subser Lect Notes Artif Intell Lect Notes Bioinformatics). 2020 May 13;12279 LNCS:77–95.

[6] Sequeira P, Gervasio M. Interestingness elements for explainable reinforcement learning: Understanding agents' capabilities and limitations. Artif Intell. 2020 Nov 1;288.

[7] Bastani O, Pu Y, Solar-Lezama A. Verifiable Reinforcement Learning via Policy Extraction. Adv Neural Inf Process Syst. 2018 May 22;2018-December:2494–504.

[8] Silva A, Killian T, Rodriguez IDJ, Son S-H, Gombolay M. Optimization Methods for Interpretable Differentiable Decision Trees in Reinforcement Learning. 2019 Mar 22 [cited 2022 May 12]; Available from: http://arxiv.org/abs/1903.09338

[9] Verma A, Murali V, Singh R, Kohli P, Chaudhuri S. Programmatically Interpretable Reinforcement Learning. 35th Int Conf Mach Learn ICML 2018 [Internet]. 2018 Apr 6;11:8024–33. Available from: https://arxiv.org/abs/1804.02477v3

[10] Nikulin D, Ianina A, Aliev V, Nikolenko S. Free-Lunch Saliency via Attention in Atari Agents. Proc - 2019 Int Conf Comput Vis Work ICCVW 2019. 2019 Aug 7;4240–9.

[11] Simonyan K, Vedaldi A, Zisserman A. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. 2nd Int Conf Learn Represent ICLR 2014 - Work Track Proc [Internet]. 2013 Dec 20 [cited 2022 May 12]; Available from: https://arxiv.org/abs/1312.6034v2

[12] Greydanus S, Koul A, Dodge J, Fern A. Visualizing and Understanding Atari Agents. 35th Int Conf Mach Learn ICML 2018 [Internet]. 2017 Oct 31 [cited 2022 May 12];4:2877–86. Available from: https://arxiv.org/abs/1711.00138v3

[13] Slack D, Hilgard S, Jia E, Singh S, Lakkaraju H. Fooling LIME and SHAP: Adversarial Attacks on Post hoc Explanation Methods. AIES 2020 - Proc AAAI/ACM Conf AI, Ethics, Soc [Internet]. 2019 Nov 6 [cited 2022 May 12];180–6. Available from: https://arxiv.org/abs/1911.02508v2

[14] Amir D, Amir O. HIGHLIGHTS: Summarizing Agent Behavior to People. In: Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems; 2018. p. 1168–1176. (AAMAS '18).

[15] Huang SH, Bhatia K, Abbeel P, Dragan AD. Establishing Appropriate Trust via Critical States. IEEE Int Conf Intell Robot Syst. 2018 Oct 18;3929–36.

[16] Amir O, Doshi-Velez F, Sarne D. Summarizing Agent Strategies. Auton Agent Multi Agent Syst [Internet]. 2019 Sep;33(5):628–644. Available from: https://doi.org/10.1007/s10458-019-09418-w

[17] Klawonn M, Heim E. Generating Triples with Adversarial Networks for Scene Graph Construction. 32nd AAAI Conf Artif Intell AAAI 2018 [Internet]. 2018 Feb 7 [cited 2022 May 12];6992–9. Available from: www.aaai.org

[18] Klawonn M, Heim E, Hendler J. Exploiting Class Learnability in Noisy Data. 33rd AAAI Conf Artif Intell AAAI 2019, 31st Innov Appl Artif Intell Conf IAAI 2019 9th AAAI Symp Educ Adv Artif Intell EAAI 2019 [Internet]. 2018 Nov 15 [cited 2022 May 12];4082–9. Available from: www.aaai.org

[19] Madumal P, Miller T, Sonenberg L, Vetere F. Explainable Reinforcement Learning Through a Causal Lens. AAAI 2020 - 34th AAAI Conf Artif Intell [Internet]. 2019 May 27 [cited 2022 May 12];2493–500. Available from: www.aaai.org

[20] Carroll M, Shah R, Ho MK, Griffiths TL, Seshia SA, Abbeel P, et al. On the Utility of Learning about Humans for Human-AI Coordination. Adv Neural Inf Process Syst [Internet]. 2019 Oct 13 [cited 2022 May 14];32. Available from: https://arxiv.org/abs/1910.05789v2

[21] Sarkar B, Talati A, Shih A, Dorsa S. PantheonRL: A MARL Library for Dynamic Training Interactions. In: Proceedings of the 36th AAAI Conference on Artificial Intelligence (Demo Track). 2022.