Explaining the Behaviour of Reinforcement Learning Agents in a Multi-Agent Cooperative Environment Using Policy Graphs

Marc Domènech i Vila , Dmitry Gnatyshak , Adrian Tormos * , Victor Gimenez-Abalos , Sergio Alvarez-Napagao

Posted Date: 19 January 2024

doi: 10.20944/preprints202401.1421.v1

Keywords: Explainable AI; Reinforcement Learning; Policy Graphs; Multi-agent Reinforcement Learning; Cooperative Environments

Article

Explaining the Behaviour of Reinforcement Learning Agents in a Multi-Agent Cooperative Environment Using Policy Graphs †

Marc Domenech i Vila [1], Dmitry Gnatyshak [2], Adrian Tormos [2,*], Victor Gimenez-Abalos [2] and Sergio Alvarez-Napagao [1,2]

[1] Universitat Politecnica de Catalunya; marc.domenech.vila@estudiantat.upc.edu
[2] Barcelona Supercomputing Center; dmitry.gnatyshak@bsc.es,adrian.tormos@bsc.es,victor.gimenez@bsc.es,sergio.alvarez@bsc.es
* Correspondence: adrian.tormos@bsc.es

† This paper is an extended version of our paper published at the 24th International Conference of the Catalan Association for Artificial Intelligence (CCIA), held in Sitges, Spain, from 19th to 21th October 2022.

Abstract: The adoption of algorithms based on Artificial Intelligence (AI) has been rapidly increasing during the last years. However, some aspects of AI techniques are under heavy scrutiny. For instance, in many use cases, it is not clear whether the decisions of an algorithm are well-informed and conforming to human understanding. Having ways to address these concerns is crucial in many domains, especially whenever humans and intelligent (physical or virtual) agents must cooperate in a shared environment. In this paper, we apply an explainability method based on the creation of a Policy Graph (PG) based on discrete predicates that represent and explain a trained agent's behaviour in a multi-agent cooperative environment. We show that from these policy graphs, policies for surrogate interpretable agents can be automatically generated. These policies can be used to measure the reliability of the explanations enabled by the PGs, through a fair behavioural comparison between the original opaque agent and the surrogate one. The contributions of this paper represent the first use case of policy graphs in the context of explaining agent behaviour in cooperative multi-agent scenarios and presents experimental results that sets this kind of scenario apart from previous implementations in single-agent scenarios: when requiring cooperative behaviour, predicates that allow representing observations about the other agents are crucial to replicate the opaque agent's behaviour and increase the reliability of explanations.

Keywords: explainable AI; reinforcement learning; policy graphs; multi-agent reinforcement learning; cooperative environments

1. Introduction and Motivation

Over the last decade, methods based on machine learning have achieved remarkable performance in many seemingly complex tasks such as image processing and generation, speech recognition or natural language processing. It is reasonable to assume that the range of potential applications will keep growing in the forthcoming years. However, there are still many concerns about the transparency, understandability and trustworthiness of systems built using these methods, especially when they are based on so-called *opaque* models [1]. For example, there is still a need for proper explanations of the behaviour of agents where their behaviour could be a risk for their real-world applicability and regulation, in domains such as autonomous driving, robots, chatbots, personal assistants, or recommendation, planning or tutoring systems [2,3].

Since AI has an increasing impact on people's everyday lives, it becomes urgent to keep progressing on the field of Explainable Artificial Intelligence (XAI) [4]. In fact, there are already regulations in place that require AI model creators to enable mechanisms that can produce explanations for them, such as the European Union's General Data Protection Regulation (GDPR) that went into effect on May 25, 2018 [5]. This law creates a "Right to Explanation" whereby a user can ask for the

 doi:10.20944/preprints202401.1421.v1

explanation of an algorithmic decision that was made about them. Therefore, explainability is not only desirable but is also a frequent requirement and, in cases where personal data is involved, it is mandatory. Furthermore, the European AI Act [6] prescribes that AI systems presenting risks, such as those that interact with humans and therefore may impact safety, must be transparent. Cooperation between humans and AIs will gradually become more common [7], and thus it is crucial to be able to explain the behavior of cooperative agents so that their actions are understandable and can be trusted by humans. However, in many cases agents trained to operate and cooperate in physical or virtual environments use Reinforcement Learning (RL) complex models such as deep neural networks that are opaque by nature. Due to this need for being able to make such systems more transparent, explainability in RL (XRL) is starting to gain momentum as a distinct field of explainability.

This paper aims to contribute to XRL by building upon the line of research opened in [8,9], which has consisted in producing explanations from predicate-based Policy Graphs (PG) generated from the observation of RL-trained agents in single-agent environments. This paper is an extension of the conference paper [10][1], in which we present a reformulation of the same methodology to generate explanations for agents trained with Multi-Agent Reinforcement Learning (MARL) methods in a cooperative environment. The contributions of this paper are focused on analysing what kind of explanations can be produced and whether there can be explanations about the relationship between the agents, i.e. about cooperation.

Currently, there are several approaches to explain agents trained with reinforcement learning, mainly from a single-agent perspective. In this work, we briefly overview some of them in Section 2 and we introduce the approach followed in this paper, based on the creation of policy graphs. We briefly summarise our initial results of applying such approach to a single-agent environment in Section 2.1. In Section 3 we introduce a multi-agent cooperative environment, which we use in Section 4 to apply our explainability method, giving some insights about the required methodology to generate explanations in a new domain. In Section 5, we introduce three algorithms that query policy graphs. In Section 6, we build new agents using the graph as a policy to compare them with the originals, in order to have a measure of reliability for the explanations. Finally, we end with a summary of the main conclusions and contributions from the work done in Section 7.

2. Background

The area of explainability in reinforcement learning is still relatively new, especially when dealing with policies as opaque models. In this section, we will provide a brief overview of some state-of-the-art XRL methods and discuss, in more depth, the method chosen for our work. A more detailed study of the explainability methods in RL can be found in [11]: XRL methods can be classified by their time horizon of explanation (reactive/proactive), scope of explanation (global/local), timing of explanation (*post hoc*/intrinsic), type of the environment (deterministic/stochastic), type of policy (deterministic/stochastic) and their agent cardinality (single-agent/multi-agent system).

Reactive explanations are those that are focused on the immediate moment. A family of reactive methods is policy simplification, which finds solutions based on tree structures. In these, the agent answers the questions from the root to the bottom of the tree in order to decide which action to execute. For instance, Coppens et al. [12] use Soft Decision Trees (SFT), structures that work similarly to binary trees but where each decision node works as a single perceptron that returns, for a given input x, the probability of going right or left. This allows the model to learn a hierarchy of filters in its decision nodes. Another family is reward decomposition, which tries to decompose the reward into meaningful components. In [13], Juozapaitis et al. decompose the Q-function into reward types to try to explain

[1] Vila, M., Gnatyshak, D., Tormos, A. & Alvarez-Napagao, S.: Testing Reinforcement Learning Explainability Methods in a Multi-agent Cooperative Environment. Published in: Artificial Intelligence Research and Development 355 A. Cortés et al. (Eds.) © 2022 The authors and IOS Press. This article is published online with Open Access by IOS Press and distributed under the terms of the Creative Commons Attribution Non-Commercial License 4.0 (CC BY-NC 4.0). doi:10.3233/FAIA220358

 doi:10.20944/preprints202401.1421.v1

why an action is preferred over another. With this, they can know whether the agent is choosing an action to be closer to the objective or to avoid penalties. Another approach is feature contribution and visual methods, like LIME [14] or SHAP [15], which try to find which of the model features are the most relevant in order to make decisions. On the other hand, Greydanus et al. [16] differentiate between gradient-based and perturbation-based saliency methods. The former try to answer the question "Which features are the most relevant to decide the output?" while the latter are based on the idea of perturbing the input of the model in order to analyse how its predictions changes.

Proactive models are those that focus on longer-term consequences. One possible approach is to analyse the relationships between variables. This family of techniques give explanations that are very close to humans because we see the world through a causal lens [17]. According to [18], the causal model tries to describe the world using random variables. Each of these variables has a causal influence on the others. This influence is modelled through a set of structural equations. Madumal et al. [19] generate explanations of behaviour based on a counterfactual analysis of the structural causal model that is learned during RL. Another approach tries to break down one task into multiple subtasks in order to represent different abstraction levels [20]. Therefore, each task can only be carried out if its predecessor tasks have been finished.

According to [21], in order to achieve interoperability, it is important that the tasks are described by humans beforehand. For instance, [20] defines two different policies in hierarchical RL, local and global policies. The first one uses atomic actions in order to achieve the sub-objectives while the second one uses the local policies in order to achieve the final goal.

In addition, there is another approach that combines relational learning or inductive logic programming with RL. The idea behind these methods [22] is to represent states, actions and policies using first order (or relational) language. Thanks to this, it is easier to generalize over goals, states and actions, exploiting knowledge learnt during an earlier learning phase.

Finally, 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 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 [8]. In this work, we will use this last approach: we will use a method that consists in 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 at the end of Section 4. This is a *post hoc* and proactive method, with a global scope of explanation, which works with both stochastic environments and policies and has until now only been tested in single-agent environments (Table 1).

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 19 January 2024 doi:10.20944/preprints202401.1421.v1

Table 1. Summarized comparison of XRL methods with policy graphs according to the taxonomy presented in Krajna et al. [11]. (*i.e.* respectively: time horizon, scope, timing, types of environment and policy and agent cardinality). A more complete description of each method can be found in the cited paper. Env. stands for type of environment (stochastic/deterministic), while P-H stands for *post hoc* and Intr. stands for intrinsic.

Method	Horiz.	Scope	Timing	Env.	Policy	Agents	Description
Coppens et al. [12]	Reac.	Global	P-H	Stoch.	Stoch.	Single	Binary decision trees, value heatmap images
Juozapaitis et al. [13]	Reac.	Local	P-H	Stoch.	Deter.	Single	Decomposed reward diagrams and images
Greydanus et al. [16]	Reac.	Global	P-H	Deter.	Deter.	Multi-	Attention saliency maps
Madumal et al. [19]	Proac.	Local	P-H	Stoch.	Stoch.	Multi-	Counterfactual text explanations
Kulkarni et al. [20]	Proac.	Local	Intr.	Stoch.	Stoch.	Multi-	Attention saliency maps
Zambaldi et al. [22]	Proac.	Local	Intr.	Stoch.	Stoch.	Multi-	Counterfactual text explanations
Policy graphs	Proac.	Global	P-H	Stoch.	Stoch.	Sing./Mult.	Behaviour graphs, text explanations, transparent agent version

Thus, a policy graph is built by sampling the behaviour of a trained agent in the environment in the form of a labelled directed graph $PG = \langle V, E, A \rangle$ where each node $v_i \in V$ represents a discretized state s_i, and each edge $e = (v_i, a, v_j) \in E$ represents the transition (s_i, a, s_j), where $a \in A$ an action from the available actions of the environment. Note that for the sake of brevity, hereinafter we equate discretized states and their node counterparts in a policy graph: $s_i = v_i \wedge V \subseteq S$, where S is the set of all discretized states.

There may exist more than one transition between the same pairs of nodes as long as the labels are different, as performing different actions in a certain state may still lead to the same resulting state. Conversely, there may exist more than one transition between a particular state and many different states labelled with the same action, as the result of applying an action on the environment may have a stochastic effect (*e.g.*, the action could occasionally fail).

Each node $v \in V$ stores the well-formed formula in propositional logic that represents its state and its probability $P(v)$. Each edge $e = (v_i, a, v_j)$ stores the action a that causes the transition it represents, and its probability $P(v_j, a|v_i)$, such that:

$$\forall v \in V, \exists e = (v, a, v') \in E \Rightarrow \sum_{(v,a_i,v_j) \in E} P(v_j, a_i|v) = 1 \quad (1)$$

Essentially, for every node in the policy graph, either the sum of probabilities for every edge originating from the node is exactly 1, or the node has no such edges.

2.1. Explaining the Cartpole Scenario

Before detailing the use of policy graphs for explaining agents' behaviour in a multi-agent cooperative setting, it may be worth describing first how this methodology can be applied to a single-agent setting.

In [9], we use Cartpole as the scenario for this work. Cartpole is an environment for reinforcement learning provided by the OpenAI Gym library[2], in which a pole is attached to a cart through an unactuated joint, and both move along a friction-less track, as depicted in Figure 1. The agent controls

[2] https://gym.openai.com

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 19 January 2024 doi:10.20944/preprints202401.1421.v1

the cart by moving it left or right, and the challenge is to prevent the pole from falling over due to gravity. To represent the state of the system, the agent has access to observations containing four variables: cart position, cart velocity, pole angle, and pole angular velocity. A fall is determined when the pole angle exceeds 12 degrees from the vertical or when the cart moves out of bounds.

It was possible to generate valid and accurate explanations for this environment by using the following set of predicates as the discretiser for the policy graphs:

- $pole\_falling(X)$, where $X \in \{left, right\}$.
- $pole\_stabilizing(X)$, where $X \in \{left, right\}$.
- $pole\_standing\_up()$.
- $cart\_moving(X)$, where $X \in \{left, right\}$.
- $cart\_pos(X)$, where $X \in \{far\_left, far\_right, left, right\}$.
- $cart\_near\_middle()$.
- $stuck(X)$, where $X \in \{left, right\}$.

This set of predicates, combined with the algorithms introduced in [8], allow for generating explanations for the behaviour of any agent operating in the Cartpole environment. This explanations can be validated via qualitative methods, *e.g.* by human expert validation. In order to validate these explanations from a quantitative perspective, our previous work [9] presented a novel technique based on the creation of a new agent policy inferred from the structure of the policy graph, trying to imitate the behaviour developed by the original trained agent's policy.

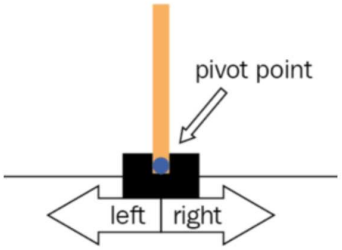

Figure 1. Cartpole environment.

One concern related to this proposal is that the policy graph is based on a simplification of the states and the actions (using predicates), and therefore such a policy could also be an over-simplification of a policy that is backed by a complex, opaque model. For this reason, our aim was to confirm whether the behaviour of the two policies entail a similar performance and, therefore, that the critical decisions that influence performance are comparable. If that is the case, the explanations generated would be able to reflect the trained agent in terms of human interpretation.

Figure 2 provides a histogram illustrating the final steps achieved by various policies. These final steps could be a consequence of either reaching success (200 steps) or encountering failure. Among these policies, the ones involving random agents (HRD 5.54% success rate and RND 0%) are severely outmatched by those considering either the original agent (DQN 78.65%), the policy graph (PGR 78.16%), or their hybrid (HEX 72.95%), the three of them having comparable success rates. Interestingly, the success rate of the PGR agent displays significant deviations in the initial steps, particularly around steps 25 to 50, as evidenced by non-marginal frequencies in the histogram. These deviations can be attributed to the previously mentioned state simplification and discretization, affecting the agent's ability to stabilize in challenging scenarios, such as when the pole is far from the center. In contrast, DQN and HEX, the latter being 50% based on PGR, showcase remarkably similar histograms, indicating that the PGR policy has a stable behaviour after 50 steps, aligning well with the original behavior.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 19 January 2024 doi:10.20944/preprints202401.1421.v1

Figure 2. Histogram of frequencies indicating up to which step each policy was able to keep the pole standing up. Policies are, in this order (left to right, top to bottom): DQN (all actions are chosen by the original policy), PGR (all actions are chosen by the policy graph), HEX (each step, the action is chosen at random between the original policy and the policy graph), HRD (each step, the action is chosen at random between the original policy and purely random), and RND (each step, the action is chosen at random from all valid actions). A successful policy reaches step 200. Please note that the random agent (RND) cannot keep the pole straight and almost never reaches step 50, while the agents trained using reinforcement learning (DQN) and the simplified agent (PGR) or combined (HEX, which is 50% DQN and 50% PGR) almost always succeed (they reach step 200), esp. when avoiding failing during the first steps.

Additionally, we conducted a correlation analysis (Spearman) involving three domain-specific metrics: average cart movement, average pole velocity, and average pole rotation (as shown in Figure 3). The most significant correlations regarding cart movement are observed between DQN and HEX (0.60, $p<0.001$) and between PGR and HEX (0.53, $p<0.001$), which is logical given HEX being combination of both policies. The correlation between DQN and PGR (0.26, $p<0.001$) is statistically significant but relatively low. However, when examining the impact of actions on the pole, including velocity and rotation, higher correlations are identified: DQN and PGR (0.55, $p<0.001$), DQN and HEX (0.72, $p<0.001$), and PGR and HEX (0.67, $p<0.001$).

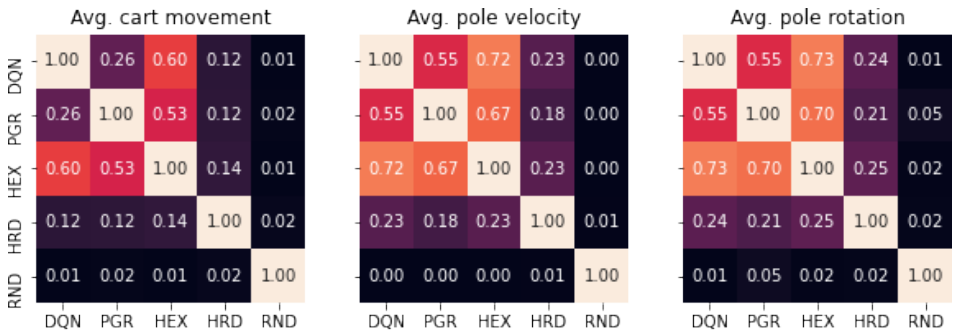

Figure 3. Cross-correlations between cart movement, pole velocity and pole rotation, averaged by step

Our analysis suggested that the behaviors of the two agents yield similar outcomes, both in terms of performance and their influence on the pole's behavior, in an environment where there is only one agent and therefore there are no actions from other agents to take into account. For further details on the methodology and the results of the quantitative analysis for the Cartpole environment, please refer to [9]. From now on in this paper we focus on the application of policy graphs to a multi-agent scenario, in where we analyse the feasibility of using policy graphs to explain the behaviour of agents in environments where the actions of other agents are relevant for the performance.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 19 January 2024 doi:10.20944/preprints202401.1421.v1

3. Overcooked-AI: A Multi-Agent Cooperative Environment

In this paper, we have used the PantheonRL [23] package for training and testing an agent in Overcooked-AI [24]. Overcooked-AI is a benchmark environment for fully cooperative human-AI task performance, based on the popular video game Overcooked[3]. 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 rewards.

The environment has a sparse reward function. On the step that certain tasks of interest are performed, the following rewards are given: 3 points if an agent places an onion in a pot or if it takes a dish, 5 points if it takes a soup and 20 points if the soup is placed in a delivery area. The same reward is delivered to both agents, without regard to which agent performed the task itself.

Here in this work, we have worked with five different layouts: *simple*, *unident_s*, *random0*, *random1* and *random3* (Figure 4).

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. We can also get the location of the basic objects (dispensers, etc.) at the start of the game.

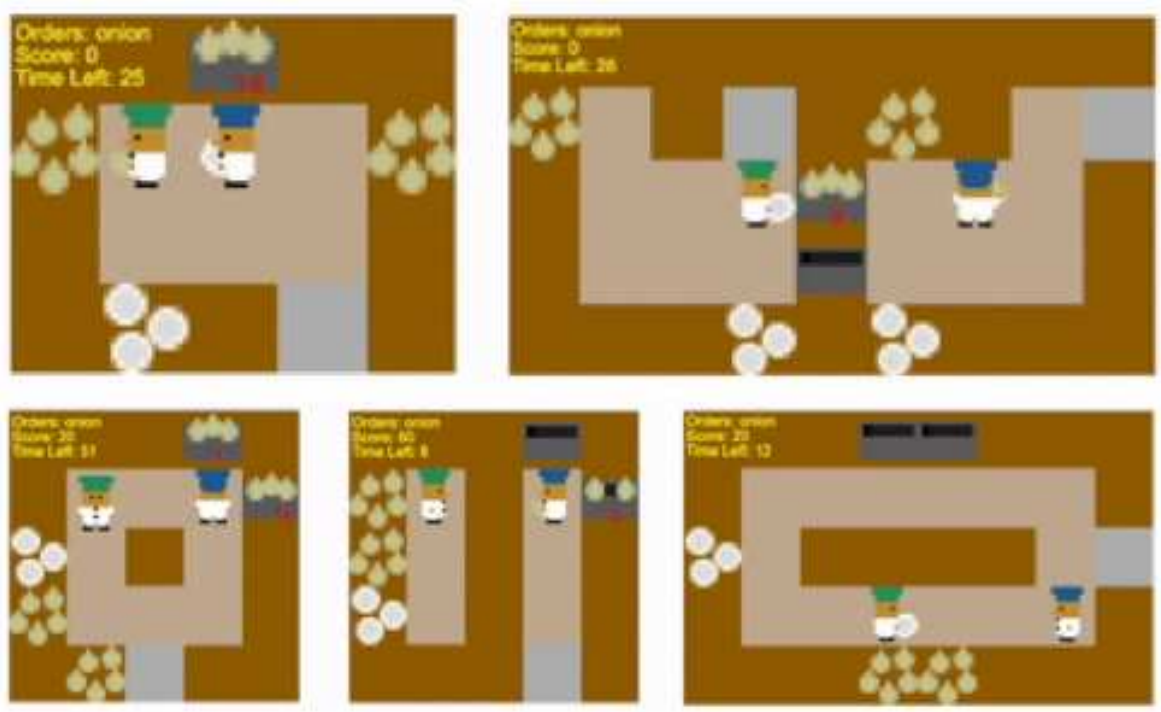

Figure 4. Overcooked layouts: *simple* (top left), *unident_s,* (top right), *random1* (bottom left), *random0* (bottom center), *random3* (bottom right)

For example, the agent would receive the following data from the situation depicted in Figure 5:

- **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.

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

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 19 January 2024 doi:10.20944/preprints202401.1421.v1

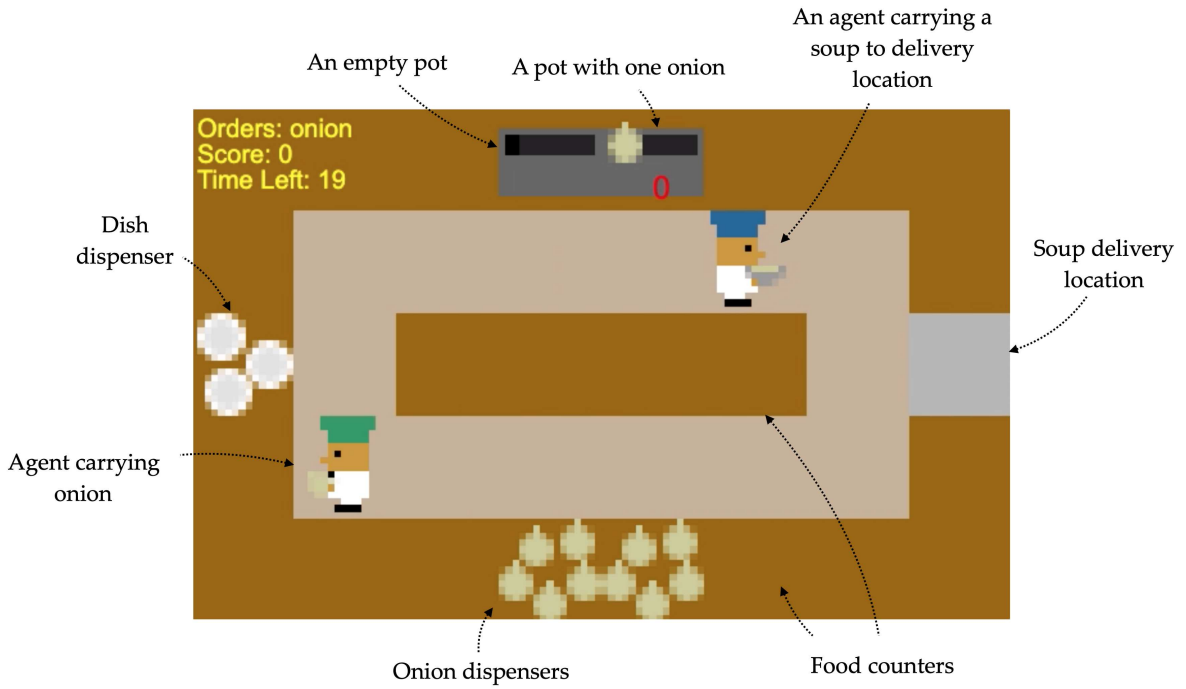

Figure 5. Overcooked-AI game dynamics.

The environment layout can have a strong influence on the degree of cooperation and the strategy agents must follow in order to win the game or to have an optimal behaviour. For instance, *unindent_s* has enough elements at each side of the kitchen for each agent to serve soups by themselves without the need of the other, so cooperation is not mandatory for winning, but desirable for optimality. On the other hand, *random0* has different elements at each side of the layout division so cooperation is mandatory for winning. *random1* and *random3* require that agents are capable of not blocking each other.

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. Thus, we train our agents using Proximal Policy Optimization (PPO) [25] because it has achieved great results in Overcooked previously [24]. We use the Stable Baselines 3 [26] Python package to train all agents. Indeed, if we get good results with PPO, we should also get good results with other methods since our explainability method is independent from the training algorithms used. In our case, we have trained five different agents (one for each layout) for 1M total timesteps and with an episode length of 400 steps. Training results can be found in Table 2.

Table 2. Performance metrics of the trained agent pairs.

Layout	Mean reward	Std.
simple	387.87	25.33
unident_s	757.71	53.03
random0	395.01	54.43
random1	266.01	48.11
random3	62.5	5.00

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 19 January 2024 doi:10.20944/preprints202401.1421.v1

4. Building a Policy Graph for the Trained Agent

We have created a total of 10 predicates to represent each state. The first two predicates are *held* and *held_partner*, which have 5 possible values depending on which object the agent and its partner, respectively, are holding: *O(nion), T(omato), D(ish), S(ervice) or * for nothing*.

The third predicate is *pot_state*, which has 4 possible values depending on the state of each pot:

- "*Of*" (Off), when $[\text{pot.onions} = 0]$.
- "*Fi*" (Finished), when $[\text{pot.onions} = 3 \wedge \text{pot.timer} \geq 20]$.
- "*Co*" (Cooking), when $[\text{pot.onions} = 3 \wedge \text{pot.timer} < 20]$.
- "*Wa*" (Waiting), when $[\text{pot.onions} < 3]$.

To relate the actions of the agent with the relative position of the objects in the environment, we introduce 6 more predicates: *onion_pos(X)*, *tomato_pos(X)*, *dish_pos(X)*, *pot_pos(X)*, *service_pos(X)* and *soup_pos(X)*. All of them can have the same 6 possible values depending on the next action to perform to reach the corresponding object as quickly as possible: *S(tay)*, *R(ight)*, *L(eft)*, *T(op)*, *B(ottom)* or *I(nteract)*. The last predicate is *partner_zone*, intended to represent agents' cooperation along with *held_partner*. It has 8 possible values depending on which cardinal point the partner is located at (e.g., "NE" for "North East").

As mentioned in Section 2, the aim of the policy graph algorithm is to apply the following method: to record all the interactions of the original trained agent by executing it in a large set of random environments and to build a graph relating predicate-based states seen in the environment with the actions executed by the agents after each of those states.

An example can be found in Figure 6. In this graph, the state on the left side represents the state {*held(Dish), held_partner(Onion), pot_state(Finished), onion_pos(Interact), tomato_pos(Stay), dish_pos(Stay), pot_pos(Top), service_pos(Right), soup_pos(Right), partner-_zone(South)*}. The state on the right side represents the state {*held(Dish), held_partner(Onion), pot_state(Finished), onion_pos(Left), tomato_pos(Stay), dish_pos(Stay), pot_pos(Left), service-_pos(Right), soup_pos(Top), partner_zone(South West)*}. This policy graph shows that, from the left state, there is a 20% probability for the agent to interact with the object in front of them, in this case the onion (due to *onion_pos* having value *Interact*). In that case, the state does not change due to the action not being effective: there are infinite onions available so it is still possible to interact with them, but the agent is already holding a dish so there is no real effect. There is a 80% probability for the agent to move right, which will cause a change of some values in the state: it is not possible to interact with the onions in the new location, as the onions are now to the left of the agent; the closest soup is now to the top instead of to the right, and the partner is to the southwest of the agent instead of to the south.

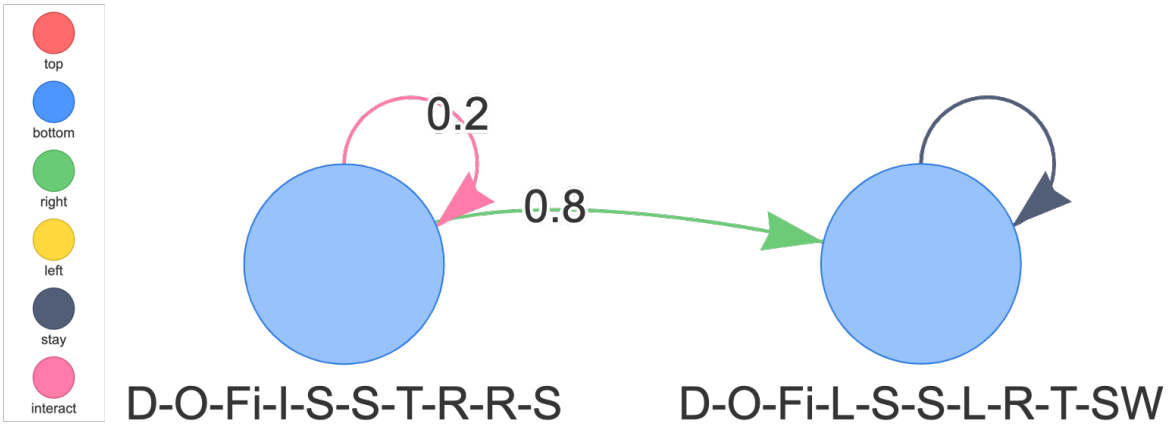

Figure 6. Extract of two states from a Stochastic policy graph generated from Overcooked.

Our work has followed two distinct approaches for building this graph:

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 19 January 2024 doi:10.20944/preprints202401.1421.v1

- **Greedy Policy Graph**: The output of the algorithm is a directed graph. For each state, the agent takes the most probable action. Therefore, not all the agent interactions are present in the graph, only the most probable action from each node. The determinism of this agent could be an interesting approach from a explainability perspective, as it is intuitively more interpretable to analyse a single action than a probability distribution.
- **Stochastic Policy Graph**: The output of the algorithm is a multi-directed graph. For each state, the agent records the action probability for all actions. As such, each state has multiple possible actions, each with its associated probability, as well as a different probability distributions for future states, one for each action. This representation is much more representative of the original agent, since the original agent may have been stochastic, or its behaviour may not be fully translated to the 'most probable action' whenever the discretiser does not capture all information of the original state.

From the policy graphs we have built, we can create surrogate agents that base their behaviour on sampling them: at each step, these agents receive the current state as an observation from the environment and decide their next action by querying their policy graph for the most probable action on the current state. This results in a transparent agent whose behaviour at any step can be directly examined via querying its policy graph.

There is a consideration to be made, though. With our proposed discretisation, there exist a total of 37,324,800 potential states, which means it is highly unlikely that the policy graph building algorithm will have observed state transitions involving all of them. In consequence, we introduce a state similarity metric $diff$: $S \times S \longmapsto \mathbb{R}$ to deal with previously unknown states, such that two states s_i, s_j are similar if $diff(s_i, s_j) \leq \varepsilon$, where $\varepsilon \in \mathbb{R}$ is a defined threshold. In this work, we define $diff$ as the amount of different predicate values between them[4], and set ε to 1.

Let $PG = \langle V, E, A \rangle$ a policy graph. Given a certain discretised state s, we can distinguish 3 cases:

1. $s \in V$: The surrogate agent picks an action using weights from the probability distribution of s in the PG.
2. $s \notin V \wedge \exists s' \in V, diff(s, s') \leq 1$: The agent picks an action using weights from the probability distribution of s_j in the PG.
3. $s \notin V \wedge \nexists s' \in V, diff(s, s') \leq 1$: The agent picks a random action with an uniform distribution.

Using these surrogate agents, we will analyse under which condition each of these approaches offers better explainability in Section 6.

5. Explainability Algorithm

The following questions can be asked of a policy graph, which are a starting point to obtaining explanations on agent behaviour:

1. **What will you do when you are in state region**[5] X**?**
2. **When do you perform action** a**?**
3. **Why did you not perform action** a **in state** s**?**

Each of these questions can be answered with custom algorithms that leverage the probability distributions learnt by the policy graph. For our work, we borrow from and evolve upon the original conceptualisation found in [8], with some changes described in the following subsections. In Section 5.4 we open a discussion regarding the validity and the limitations of this approach.

---

[4] *E.g.* if we have the states s_i = O-Co-S and s_j = O-Co-N, then $diff(s_i, s_j) = 1$.
[5] *E.g.* all states where *pot_state(Finished)*.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 19 January 2024 doi:10.20944/preprints202401.1421.v1

5.1. What Will You Do When You Are in State Region X?

The answer to this question consists in aggregating the probabilities of possible actions in the policy graph from all the input states that the user wants to check. Let $PG = \langle V, E, A \rangle$ be the computed policy graph, $X \in S$ the target state region, and $dist$: $\mathcal{P}(S) \times S \longmapsto \mathbb{R}$ a measure of distance between a discretized state and a state region in S [6]. Very similarly to the possible courses of action of a surrogate agent defined in Section 4, there are three possible cases regarding the information available about the state region X.

1. $|X \cap V| > 0$: The policy graph generation algorithm has seen one or more states $s \in X$ during training, and it is likely that we can extract the probability of choosing among each of the accessible actions from them.[7]
2. $X \cap V = \varnothing$, but one or more similar states are found ($\exists s \in V : dist(X, s) < \varepsilon$, $\varepsilon \in \mathbb{R}$): the policy graph has never seen any state in X so we rely on a measure of similarity to another state to extrapolate (as in case 1).
3. $X \cap V = \varnothing$ and no similar state is found: Returns a uniform probability distribution over all the actions.

Algorithm 1 What will you do when you are in state region X?

Input: Policy Graph $PG = \langle V, E, A \rangle$, Action Set A, Set of states X, Distance threshold ε
Output: Explanation of policy behavior in X per action

$X' \leftarrow X$
if $X \cap V = \varnothing$ **then**
  $X' \leftarrow \{v \in V \mid dist(X, v) = \min_{v' \in V} dist(X, v')\}$ ▷ The set of states in V closest to X
end if
if $dist(X, X') > \varepsilon$ **then**
  $\mathbf{P} \leftarrow Uniform(A)$;
else
  for all $a \in A$ **do**
    $\mathbf{P}[a] \leftarrow \frac{\sum_{s \in X'} P(a|s)P(s)}{\sum_{s \in X'} P(s)}$
  end for
end if
return P

A formal version of this procedure is shown in Algorithm 1.

For example, for the state region $\{\{held(Service), pot\_state(Waiting), onion\_pos(Top), tomato\_pos(Stay), dish\_pos(Stay), pot\_pos(Left), service\_pos(Interact), soup\_pos(Left)\}\}$ (see Figure 7), the most probable action is *to interact* since the state shows us that the agent holds a soup and it is in front of the service.

---

[6] This distance function can depend heavily on the environment and on the predicate set chosen. An in-depth analysis of possible functions is out of the scope of this paper, but it is part of future work. For the sake of proof-of-concept, the distance function we have chosen for the work presented in this paper consists in: let $s \in S$ and $X = \{s_1, ...\} \subseteq S$, we define $dist(X, s) = min_{s_i \in X}\, diff(s_i, s)$, where $diff$ is the function defined in Section 4. For example, this measure for the states in Figure 6 would be $dist(\{s_{left}\}, s_{right}) = 4$ as only four predicates change value between them.

[7] This is only likely since a state s may have been visited very few times, and the estimation of probability may be little informed, in which case we would consider the other options in the list.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 19 January 2024 doi:10.20944/preprints202401.1421.v1


```
* What will I do when I am in state X?

Possible predicates:

  + held          * | O | T | D | S
  + pot_state_0   Of | Fi | Co | Wa
  + onion_pos     S | R | L | T | B | I
  + tomato_pos    S | R | L | T | B | I
  + dish_pos      S | R | L | T | B | I
  + pot_pos_0     S | R | L | T | B | I
  + service_pos   S | R | L | T | B | I
  + soup_pos      S | R | L | T | B | I

State: S-Wa-T-S-S-L-I-L
I will take one of these actions:
  -> Interact     Prob: 94.05 %
  -> Bottom       Prob: 2.31 %
  -> Left         Prob: 1.82 %
  -> Right        Prob: 1.33 %
  -> Stay         Prob: 0.49 %
```

```
* What will I do when I am in state X?

Possible predicates:

  + held          * | O | T | D | S
  + pot_state_0   Of | Fi | Co | Wa
  + onion_pos     S | R | L | T | B | I
  + tomato_pos    S | R | L | T | B | I
  + dish_pos      S | R | L | T | B | I
  + pot_pos_0     S | R | L | T | B | I
  + service_pos   S | R | L | T | B | I
  + soup_pos      S | R | L | T | B | I

State: S-Wa-T-S-S-L-I-L
I will take one of these actions:
  -> Interact     Prob: 100 %
```

Figure 7. Example output of the algorithm *What will I do when I am in state region* $\{s\}$*?* from policy graphs generated for a PPO-trained agent. On the left, the result for the Stochastic Policy Graph; on the right, the result for the Greedy Policy Graph.

5.2. When Do You Perform Action a?

The answer to such a question can be generated from an extensive list of all states that perform such an action (*i.e.* when is it the most likely?), as formalised in Algorithm 2.

Algorithm 2 When do you perform action a?

Input: Policy Graph $PG = \langle V, E, A \rangle$, Target action a
Output: Set of target states S_{PG^a} where target a is the dominant action, Set of non-target states $S_{PG^{*\setminus a}}$

$S_{PG^a} \leftarrow \{\}$
$S_{PG^{*\setminus a}} \leftarrow \{\}$
for all $s \in V$ **do**
  $a^* \leftarrow arg\,max_{a \in A}\, P(a|s)$
  if $a^* = a$ **then**
    $S_{PG^a} \leftarrow S_{PG^a} \cup s$;
  else
    $S_{PG^{*\setminus a}} \leftarrow S_{PG^{*\setminus a}} \cup s$;
  end if
end for
return $S_{PG^a}, S_{PG^{*\setminus a}}$

For instance, when asking the policy graph of a trained agent for the *Interact* action, the list may include the state $\{$*held(Nothing), pot_state(Cooking), onion_pos(Left), tomato_pos(Stay), dish_pos(Interact), pot_pos(Left), service_pos(Right), soup_pos(Top)*$\}$ (see Figure 8). The main predicates to analyse here would be: (1) the agent is empty-handed, (2) the pot is cooking, and (3) we are in interact position with the dish pile. This results in picking up a plate, which seems reasonable given a plate is necessary to pick up soup from the pot once it finishes cooking.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 19 January 2024 doi:10.20944/preprints202401.1421.v1


```
* When do you perform action X?

Possible actions:

  + Top
  + Bottom
  + Right
  + Left
  + Stay
  + Interact

When do you perform action: Interact
Most probable action in 159 states:
  -> *-Co-B-S-S-L-R-I
  -> *-Co-B-S-S-R-R-I
  -> *-Co-L-S-B-L-R-T
  -> *-Co-L-S-I-L-R-T
```

```
* When do you perform action X?

Possible actions:

  + Top
  + Bottom
  + Right
  + Left
  + Stay
  + Interact

When do you perform action: Interact
Most probable action in 157 states:
  -> *-Co-L-S-B-L-R-T
  -> *-Co-L-S-I-L-R-T
  -> *-Co-L-S-R-B-R-T
  -> *-Co-L-S-R-I-R-R
```

Figure 8. Example output of the algorithm *When do you perform action a?* from policy graphs generated for a PPO-trained agent. On the left, the result for the Stochastic Policy Graph; on the right, the result for the Greedy Policy Graph.

An intuitive improvement over this version would be finding which subset of satisfied predicates are sufficient to cause the action being picked. This would greatly reduce the complexity of analysis, given that instead of outputting a large number of states, we would find a small set of partial predicates.

5.3. Why Did You Not Perform Action a in State s?

This question is a non-trivial counterfactual question, regarding a transition: doing action a in a state s which potentially has never been sampled (hence the value in answering the question). To do so, Algorithm 3 finds the neighbouring states (*i.e.* states within a distance threshold in the metric space proposed in Section 5.1), and lists the difference in predicate sets between the regions where action a is performed and where it is not. Much like before, we distinguish between 3 cases depending on the characteristics of s:

Algorithm 3 Why did you not perform action a in state s_p?

Input: Policy Graph $G = V, E$, Target Action a, Previous State s_p, Distance threshold D_{const}
Output: Explanation of difference between current state and state region where a_t is performed, explanation of where a_t is performed locally.

$S_{\pi^a} \leftarrow \{\}$
$S_{\pi^{*\setminus a}} \leftarrow \{\}$
for all $s \in \{v \in V \mid dist(\{v\}, s_p) \leq \varepsilon\}$ **do**
  $a^* \leftarrow most\_frequent\_action\_executed\_from(s)$;
  if $a^* == a$ **then**
    $S_{\pi^a} \leftarrow S_{\pi^a} \cup s$;
  else
    $S_{\pi^{*\setminus a}} \leftarrow S_{\pi^{*\setminus a}} \cup s$;
  end if
end for
$expected\_region \leftarrow describe(S_{\pi^a}, S_{\pi^{*\setminus a}}, C)$;
$current\_region \leftarrow describe(\{s_p\}, S_{\pi^a}, C)$;
return $diff(expected\_region, current\_region), expected\_region$;

1. $s \in V$: The states within a distance threshold δ to s are gathered, and filtered to those where action a is the most likely (S_{PG^a}). The output is the list of differences $\forall v \in S_{PG^a}, preds(s) - preds(v)$. If $S_{PG^a} = \varnothing$, no explanation is given and it is suggested to increase the threshold.
2. $s \notin V$ but $\exists s' \in V : dist(\{s\}, s') < \varepsilon$, $\varepsilon \in \mathbb{R}$: the state s' substitutes s in the algorithm above.
3. $s \notin V$ and no similar state is found: no explanation is given due to lack of information.

As an example, consider the answer to "why the agent did not take the *top* action in state {*held(Dish), pot_state(Cooking), onion_pos(Left), tomato_pos(Stay), dish_pos(Interact), pot_pos(Top), service_pos(Bottom), soup_pos(Left)*}?", as seen in Figure 9. This state is not part of the policy graph, whereas the one where *dish_pos(Stay)* indeed is (the difference being whether the dish pile is in the

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 19 January 2024 doi:10.20944/preprints202401.1421.v1

agent's interact position (s) or south (s'). The predominant action in state s' is going to the *left*. This algorithm's output is the list of nearby states where the chosen action is *top*. For example, {*held(Dish)*, *pot_state(Cooking)*, *onion_pos(Left)*, *tomato_pos(Stay)*, *dish_pos(Stay)*, *pot_pos(Right)*, *service_pos(Right)*, *soup_pos(Top)*}, where the nearest soup was on top. Seeing the difference a human interpreter may understand that the agent has taken the action which brings him closer to the soup. The output of the algorithm itself is arguably not an explanation, but it gives information which could be useful in providing one.

```
* Why did not you perform X in Y state?

State: D-Co-L-S-S-T-B-L
I would have chosen: Actions.Left
I would choose Top if:
+ D-Co-L-S-S-R-R-T
  pot_pos_0 = T   -> pot_pos_0 = R
  service_pos = B -> service_pos = R
  soup_pos = L    -> soup_pos = T
```

```
* Why did not you perform X in Y state?

State: D-Co-L-S-S-T-B-L
I would have chosen: Actions.Left
I would choose Top if:
+ D-Co-L-S-S-R-R-T
  dish_pos = I    -> dish_pos = R
  pot_pos_0 = T   -> pot_pos_0 = R
  service_pos = B -> service_pos = R
  soup_pos = L    -> soup_pos: T
```

Figure 9. Example output of the algorithm *Why did you not perform action a in state s?* from policy graphs generated for a PPO-trained agent. On the left, the result for the Stochastic Policy Graph; on the right, the result for the Greedy Policy Graph.

5.4. Can We Rely on These Explanations?

As we mentioned in Section 1, one of the objectives of this project is to test if we can get valid explanations from an agent behaviour using the explainability method used in [9] in a multi-agent cooperative environment. In this section, we have seen some examples of the explanations given by our policy graphs.

We would like to emphasise the fact that not all the explanations given by the policy graph are so easy to interpret or understand at first sight. For instance, in Figure 9, we have seen that we asked the policy graph why the agent did not perform the *top* action in the state {*held(Dish)*, *pot_state(Cooking)*, *onion_pos(Left)*, *tomato_pos(Stay)*, *dish_pos(Interact)*, *pot_pos(Top)*, *service_pos(Bottom)*, *soup_pos(Left)*}. In this case, the algorithm answered that they would have chosen to go to the left, but the reason behind this decision is not so clear here, at least at first sight. We could elaborate hypotheses about the strategy that each agent was playing and maybe they could understand their decision. There is also the possibility that, in this state, the agent is not choosing the more appropriate action. While analysing these aspects in more depth is crucial for the objective of achieving better explainability, it is out of the scope of this paper so we leave it for future work.

The policy graphs derived in the previous sections enable the generation of natural language explanations. In [8], the explanations are validated by comparing the sentences generated by the algorithm against sentences written by human experts. While this may be a valid qualitative approach for validation, it relies on expert availability and on the nature of the specific domain.

Even though sometimes the explanations given by the method are not as illustrative as we intended, at least we have an explanation. Namely, although at first glance we have not been able to draw too many conclusions, this method is explaining to us what has to happen for the agent to take said action. Therefore, it is one more tool to study and analyse the behaviour of these types of AI models. For all the reasons mentioned above, we can say we have met the goal of extracting useful explanations from the policy graph.

6. Validating the Policy Graph

To ensure that the policy graphs provide true explanations, we generate a policy graph for each Overcooked layout based on the trajectories of 1500 episodes on different seeds, and we build surrogate agents from them as explained in Section 4. In order to test these new surrogate agents, we run them through the environment for a total of 500 different fixed seeds and track the obtained rewards, the

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 19 January 2024 doi:10.20944/preprints202401.1421.v1

amount of previously unknown states encountered, the amount of served soups per episode and the mean log-likelihood $-\bar{l} = -\sum log\, P(s', a|s)$ of the original's trajectories for each policy graph.

In order to gauge the effect of cooperative predicates, we build 4 policy graphs per Overcooked layout, increasingly more expressive in relation to the state and actions of the agent's partner. For each of the 4 policy graphs, we use a different subset of the 10 predicates defined in Section 4. We give these subsets of predicates the names D11 to D14 (Table 3).

As we saw in Section 4, we are testing 2 different policy graph generation algorithms. Therefore, we have built 2 surrogate agents for each layout. We call an agent *Greedy* if its policy is generated from a **Greedy Policy Graph**; analogously, we call an agent *Stochastic* if its policy is generated from a **Stochastic Policy Graph**.

All experiment runs on the Overcooked environment have been performed using Python 3.7, PantheonRL v0.0.1 [23] and Overcooked-AI v0.0.1[8]. All PPO agents have been trained with Stable Baselines 3 v1.7.0 [26], while the policy graphs have been generated with NetworkX v2.6.3[9]

	Single-agent			Coop.	
	held	*pot_state*	*object_pos*	*held_partner*	*partner_zone*
D11	X	X	X		
D12	X	X	X	X	
D13	X	X	X		X
D14	X	X	X	X	X

Table 3. Sets of predicates used to generate policy graphs. Note that *held_partner* and *partner_zone* are the only cooperative predicates in the sets.

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

Figure 10 shows the rewards obtained by the different surrogate agents. We can see that in the *simple* and *unident_s* scenarios, the greedy surrogate agents manage to consistently outperform the original ones while the stochastic ones do not. We can also see how in *random1*, the greedy agents are not able to function properly while the stochastic agents almost score as well as the originals. Note how the addition of the *partner_zone* predicate to an agent clearly improves its median score or makes it perform better more consistently in several cases. Examples include the greedy agents for *simple*, *random3* and *random0* and the stochastic agents for *simple*, *random1* and *random3*. It also can be seen that, although they may improve performance, in most layouts it is not necessary to introduce cooperative predicates to explain the surrogate agents' behaviour. In the *random0* scenario, though, the agents need the predicate *partner_zone* to get good results.

[8] https://github.com/HumanCompatibleAI/overcooked_ai
[9] https://networkx.org/

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 19 January 2024 doi:10.20944/preprints202401.1421.v1

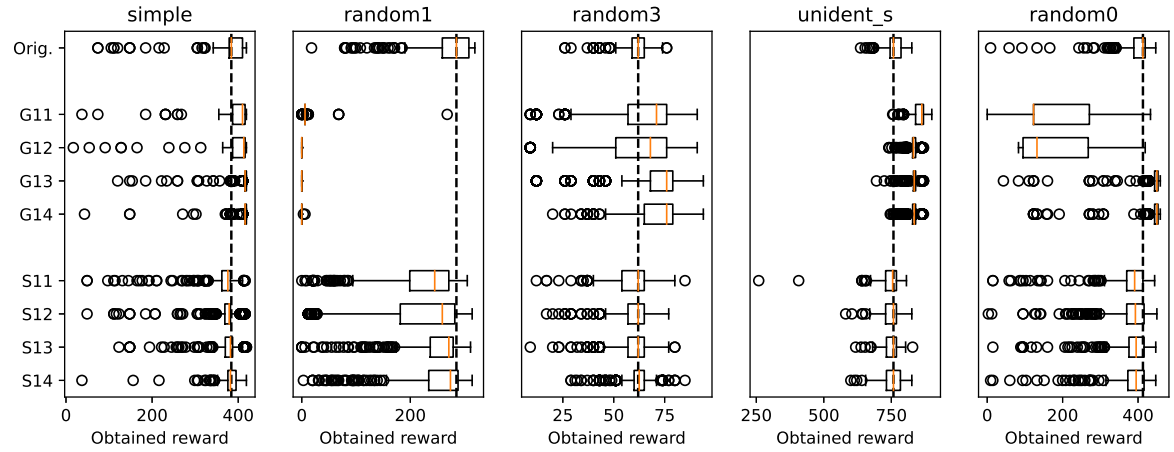

Figure 10. Obtained rewards by the original and surrogate agents in all layouts. Orig. stands for the original agent, and G and S stand for the greedy and stochastic surrogate agents. G11 to G14 and S11 to D14 respectively refer to the subsets of predicates D11 to D14.

Regarding the encounter of previously unknown states (Table 4), we can see that most of the greedy surrogate agents get to a very small amount of new states in relation to their policy graph's size (*i.e.* already seen states) and spend very little amount of steps in them in relation with the total evaluation time. An exception is *random_3*, for which it appears that there still was a notable amount of states to explore when building the policy graph. On the other hand, all the stochastic surrogate agents can consistently arrive to unexplored states to a higher degree than their greedy counterparts. This is to be expected, as the stochastic nature of the former inherently promotes exploration.

Table 4. Surrogate agent statistics. $|PG|$ represents the size of the agent's policy graph. NS means amount of new states encountered, $\%|PG|$ represents the amount of new states as a proportion of the size of the agent's policy graph, and %steps indicates the percentage of steps during the evaluation runs that the agent spent in new state. $-\bar{l}$ is the negative log-likelihood. Note that a previously unknown state transition would make the log-likelihood go to Infinity, so instead we penalize it by $min_{PG}\, log\, P(s', a|s)$.

				Greedy			Stochastic		
		$-\bar{l}$	$\|PG\|$	NS	$\%\|PG\|$	%steps	NS	$\%\|PG\|$	%steps
simple	D11	**880.1**	475	**0**	**0.0%**	**0.0%**	**70**	14.7%	1.3%
	D12	996.2	884	2	0.2%	**0.0%**	75	8.5%	0.6%
	D13	1036.2	1045	5	0.5%	**0.0%**	94	9.0%	0.5%
	D14	1042.0	1734	5	0.3%	**0.0%**	135	**7.8%**	**0.3%**
random_1	D11	**696.1**	2663	2	0.1%	**0.0%**	**197**	**7.4%**	**1.6%**
	D12	731.4	5016	**0**	**0.0%**	**0.0%**	625	12.5%	3.2%
	D13	828.2	5256	**0**	**0.0%**	**0.0%**	530	10.1%	2.3%
	D14	827.8	8566	29	0.3%	0.3%	1260	14.7%	3.2%
random_3	D11	**760.5**	2195	**31**	**1.4%**	3.2%	**177**	**8.1%**	**1.2%**
	D12	787.1	3613	54	1.5%	**1.4%**	419	11.6%	1.4%
	D13	859.3	3536	92	2.6%	4.3%	595	16.8%	1.6%
	D14	878.6	5436	161	3.0%	2.1%	827	15.2%	1.3%
unident_s	D11	**559.5**	202	**0**	**0.0%**	**0.0%**	**12**	5.9%	0.2%
	D12	673.8	464	**0**	**0.0%**	**0.0%**	49	10.6%	0.1%
	D13	743.0	422	1	0.2%	**0.0%**	19	**4.5%**	**0.0%**
	D14	777.4	900	5	0.6%	**0.0%**	70	7.8%	0.1%
random0	D11	**650.5**	1793	**3**	**0.2%**	0.3%	**133**	7.4%	1.6%
	D12	693.6	3089	37	1.2%	30.6%	268	8.7%	2.1%
	D13	710.9	2696	6	**0.2%**	**0.0%**	181	6.7%	**1.2%**
	D14	745.5	4392	36	0.8%	0.3%	285	**6.5%**	1.7%

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 19 January 2024 doi:10.20944/preprints202401.1421.v1

Also on Table 4, we can observe that the episode trajectories achieved by the original agents have a better mean log-likelihood with the D11 policy graphs than the others. This stems from the fact that the policy graphs built with D11 have a simpler world representation than those built with D12-14, as D11 is the smallest subset of predicates in use. As a result, these policy graphs have less entropy.

To summarise, the stochastic policy graphs provide stable surrogate agents that mirror the original agent's behaviour consistently. However, greedy policy graphs are a double edged sword: their surrogate agents can act like distilled versions of the original's behaviour that remove noise, or stop functioning because their deterministic nature did not capture fundamental parts of the original's behaviour. Thus, stochastic policy graphs are the more reliable choice from an XRL point of view and should be the one used for producing explanations.

We have also seen that although there are scenarios where it is not necessary to introduce cooperative predicates to explain the agent's behaviour, there are others – particularly those that promote the need for cooperation – where this information is crucial.

7. Conclusions

Explainability in artificial intelligence is a research area that has been rapidly growing in recent years, due to the need to understand and justify the decisions made by AIs, especially in the field of RL. All the research made in this area can be key not only to study the quality of an agent's decision but also to help people rely on AI, especially in situations where humans and machines have to cooperate, and it is becoming necessary to be able to give explanations about their decisions. There are already some proposals in the literature to provide them, and it is important to test their effectiveness in practice.

In this paper, we have presented the procedure and the experimental results of applying an explainability method, based on the construction of a policy graph by discretising the state representation into predicates, into a cooperative MARL environment (Overcooked). We have proposed two different algorithms to generate the policy graph and we have used them to generate explanations that, following [8], can be transformed into human language expressions. In principle, the quality of these explanations can be qualitatively validated by human experts with domain-specific knowledge. Our contribution to this respect is a quantitative validation of the generated policy graphs by applying a method previously employed into a single-agent environment (Cartpole) [9], automatically generating policies based on these explanations in order to build agents that represent their original behaviour. Finally, we have analysed the behaviour of agents following these new policies in terms of their explainability capabilities, depending on the environmental conditions and we have shown that relevant predicates for cooperation become also important for the explanations when the environmental conditions influence the need for such cooperation.

The contributions presented in this paper are part of ongoing research, by testing among different environments, types of agent policy, and explainability methods. Therefore, several important points can be explored to complement and advance our work. For example, policy graphs are rarely complete as seen in Section 6. It would be very interesting to be able to produce a domain-agnostic distance function that enables a reliable detection of similar states for different environments.

There should also be a more general and comprehensive analysis of the question-answering capabilities of policy graphs, as this can help quantifying the understandability of the explanations produced. One way can be via the exploration and creation of new algorithms, adding to the ones explored in this paper, in order to attend to other modalities that the receivers of explanations might find important. Additionally, a more thorough study of when and why the policy graphs adjust better to the original agent, regardless of the specific domain – in terms of, for instance, whether the policy is greedy or stochastic, or what the relationship between each subset of predicates and the performance is.

Finally, other environments should be explored. Specifically, Overcooked-AI requires cooperation in some layouts but this cooperation is not explicitly reflected in the policy but emerging from the

 doi:10.20944/preprints202401.1421.v1

behaviour. It would be interesting to explore an application of policy graphs in an environment where there are explicit mechanisms such as communication or cooperative planning. In competitive or mixed-motive environments, such as StarCraft II [27], Neural MMO [28], Google Research Football[10], or Multi-Agent Particle [29], there also exists potential in using policy graphs to create a model of the behaviour of opponent agents based on observations. The resulting graph could be leveraged by an agent using Graph Neural Networks (GNN) [30] to change its own behaviour with the purpose of increasing performance against specific strategies previously encountered.

Author Contributions: Conceptualization, V.G.-A., A.T., D.G. and S.A.-N.; methodology, M.D.-i-V., A.T. and S.A.-N.; software, M.D.-i-V. and A.T.; validation, M.D.-i-V., A.T. and V.G.-A.; formal analysis, M.D.-i-V. and V.G.-A.; investigation, M.D.-i-V., D.G. and V.G.-A.; writing—original draft preparation, M.D.-i-V.; writing—review and editing, A.T., V.G.-A. and S.A.-N.; supervision, D.G. and S.A.-N.; project administration, S.A.-N.; funding acquisition, S.A.-N.. All authors have read and agreed to the published version of the manuscript.

Funding: This work has been partially supported by the H2020 knowlEdge European project (Grant agreement ID: 957331).

Data Availability Statement: Data and code can be found at the public repository: https://github.com/MarcDV1999/overcooked-explainability

Acknowledgments: The authors want to thank Pablo A. Martin-Torres and Dario Garcia-Gasulla for their feedback and comments that helped improve the final version of the paper.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

XAI	Explainable Artificial Intelligence
GDPR	General Data Protection Regulation
RL	Reinforcement Learning
XRL	Explainable Reinforcement Learning
PG	Policy Graph
MARL	Multi-Agent Reinforcement Learning
SFT	Soft Decision Tree
LIME	Local Interpretable Model-agnostic Explanations
SHAP	SHapley Additive exPlanations
PPO	Proximal Policy Optimization
TL	Transferred Learning
STD	Standard Deviation
NS	New States

References

1. Li, B.O.; Qi, P.; Liu, B.O.; Di, S.; Liu, J.; Pei, J.; Yi, J.; Zhou, B. Trustworthy AI: From Principles to Practices **2021**. [2110.01167]. https://doi.org/10.48550/arxiv.2110.01167.
2. Omeiza, D.; Webb, H.; Jirotka, M.; Kunze, L. Explanations in Autonomous Driving: A Survey. *IEEE Transactions on Intelligent Transportation Systems* **2021**, pp. 1–21. arXiv:2103.05154 [cs], https://doi.org/10.1109/TITS.2021.3122865.
3. Rosenfeld, A.; Richardson, A. Explainability in human–agent systems. *Autonomous Agents and Multi-Agent Systems* **2019**, *33*, 673–705.
4. Longo, L.; Goebel, R.; Lecue, F.; Kieseberg, P.; Holzinger, A. Explainable artificial intelligence: Concepts, applications, research challenges and visions. In Proceedings of the International Cross-Domain Conference for Machine Learning and Knowledge Extraction. Springer, 2020, pp. 1–16.

[10] https://github.com/google-research/football

5. Goodman, B.; Flaxman, S. European Union regulations on algorithmic decision-making and a "right to explanation". *AI magazine* **2017**, *38*, 50–57.
6. Madiega, T. Artificial intelligence act. *European Parliament: European Parliamentary Research Service* **2021**.
7. Dafoe, A.; Hughes, E.; Bachrach, Y.; Collins, T.; McKee, K.R.; Leibo, J.Z.; Larson, K.; Graepel, T. Open Problems in Cooperative AI. *arXiv:2012.08630 [cs]* **2020**. arXiv: 2012.08630.
8. Hayes, B.; Shah, J.A. Improving robot controller transparency through autonomous policy explanation. In Proceedings of the 2017 12th ACM/IEEE International Conference on Human-Robot Interaction (HRI. IEEE, 2017, pp. 303–312.
9. Climent, A.; Gnatyshak, D.; Alvarez-Napagao, S. Applying and Verifying an Explainability Method Based on Policy Graphs in the Context of Reinforcement Learning. In *Artificial Intelligence Research and Development*; IOS Press, 2021; pp. 455–464.
10. Vila, M., Gnatyshak, D., Tormos, A. & Alvarez-Napagao, S. Testing Reinforcement Learning Explainability Methods in a Multi-agent Cooperative Environment. *Artificial Intelligence Research And Development*. **356** pp. 355-364 (2022).
11. Krajna, A.; Brcic, M.; Lipic, T.; Doncevic, J. Explainability in reinforcement learning: perspective and position. *arXiv preprint arXiv:2203.11547* **2022**.
12. Coppens, Y.; Efthymiadis, K.; Lenaerts, T.; Nowé, A.; Miller, T.; Weber, R.; Magazzeni, D. Distilling deep reinforcement learning policies in soft decision trees. In Proceedings of the Proceedings of the IJCAI 2019 workshop on explainable artificial intelligence, 2019, pp. 1–6.
13. Juozapaitis, Z.; Koul, A.; Fern, A.; Erwig, M.; Doshi-Velez, F. Explainable reinforcement learning via reward decomposition. In Proceedings of the IJCAI/ECAI Workshop on explainable artificial intelligence, 2019.
14. Ribeiro, M.T.; Singh, S.; Guestrin, C. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In Proceedings of the Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016, 2016, pp. 1135–1144.
15. Lundberg, S.M.; Lee, S.I. A Unified Approach to Interpreting Model Predictions. In *Advances in Neural Information Processing Systems 30*; Guyon, I.; Luxburg, U.V.; Bengio, S.; Wallach, H.; Fergus, R.; Vishwanathan, S.; Garnett, R., Eds.; Curran Associates, Inc., 2017; pp. 4765–4774.
16. Greydanus, S.; Koul, A.; Dodge, J.; Fern, A. Visualizing and understanding atari agents. In Proceedings of the International conference on machine learning. PMLR, 2018, pp. 1792–1801.
17. Sloman, S. *Causal models: How people think about the world and its alternatives*; Oxford University Press, 2005.
18. Halpern, J.Y.; Pearl, J. Causes and Explanations: A Structural-Model Approach — Part 1: Causes **2013**. [1301.2275]. https://doi.org/10.48550/arxiv.1301.2275.
19. Madumal, P.; Miller, T.; Sonenberg, L.; Vetere, F. Explainable reinforcement learning through a causal lens. In Proceedings of the Proceedings of the AAAI conference on artificial intelligence, 2020, Vol. 34, pp. 2493–2500. Issue: 03.
20. Kulkarni, T.D.; Narasimhan, K.; Saeedi, A.; Tenenbaum, J. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *Advances in neural information processing systems* **2016**, *29*.
21. Shu, T.; Xiong, C.; Socher, R. Hierarchical and interpretable skill acquisition in multi-task reinforcement learning. *arXiv preprint arXiv:1712.07294* **2017**.
22. Zambaldi, V.; Raposo, D.; Santoro, A.; Bapst, V.; Li, Y.; Babuschkin, I.; Tuyls, K.; Reichert, D.; Lillicrap, T.; Lockhart, E.; et al. Relational Deep Reinforcement Learning, 2018. Number: arXiv:1806.01830 arXiv:1806.01830.
23. Sarkar, B.; Talati, A.; Shih, A.; Sadigh, D. PantheonRL: A MARL Library for Dynamic Training Interactions, 2021. Number: arXiv:2112.07013 arXiv:2112.07013 [cs].
24. Carroll, M.; Shah, R.; Ho, M.K.; Griffiths, T.; Seshia, S.; Abbeel, P.; Dragan, A. On the utility of learning about humans for human-ai coordination. *Advances in neural information processing systems* **2019**, *32*.
25. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal Policy Optimization Algorithms, 2017. Number: arXiv:1707.06347 arXiv:1707.06347 [cs]
26. Raffin, A.; Hill, A.; Gleave, A.; Kanervisto, A.; Ernestus, M; Dormann, N. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal Of Machine Learning Research*. **22**, 1-8 (2021).
27. Vinyals, O.; Ewalds, T.; Bartunov, S.; Georgiev, P.; Vezhnevets, A.; Yeo, M.; Makhzani, A.; Küttler, H.; Agapiou, J.; Schrittwieser, J.; Quan, J.; Gaffney, S.; Petersen, S.; Simonyan, K.; Schaul, T.; Hasselt, H.; Silver,

D.; Lillicrap, T.; Calderone, K.; Keet, P.; Brunasso, A.; Lawrence, D.; Ekermo, A.; Repp, J.; Tsing, R. StarCraft II: A New Challenge for Reinforcement Learning, 2017. Number: arXiv:1708.04782 arXiv:1708.04782 [cs].

28. Suarez, J.; Du, Y.; Isola, P; Mordatch, I. Neural MMO: A Massively Multiagent Game Environment for Training and Evaluating Intelligent Agents, 2019. Number: arXiv:1903.00784 arXiv:1903.00784 [cs, stat]
29. Lowe, R.; Tamar, A.; Harb, J.; Pieter Abbeel, O.; Mordatch, I. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances In Neural Information Processing Systems*. **30** (2017).
30. Munikoti, S.; Agarwal, D.; Das, L.; Halappanavar, M.; Natarajan, B. Challenges and Opportunities in Deep Reinforcement Learning With Graph Neural Networks: A Comprehensive Review of Algorithms and Applications. *IEEE Transactions On Neural Networks And Learning Systems*. pp. 1-21 (2023).