UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Testing Reinforcement Learning explainability methods in a Multi-agent cooperative environment

Marc Domènech i Vila

Bachelor's thesis
Specialization in Computing

High Performance Artificial Intelligence
Facultat d'Informàtica de Barcelona
Universitat Politècnica de Catalunya

Director: Sergio Álvarez-Napagao
Codirector: Dmitry Gnatyshak
GEP Tutor: Joan Sardà

Saturday 2nd July, 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 cases, it is not clear whether their decisions are well-informed or whether we can trust them. Having an answer to these concerns is crucial in many domains, such as those in were humans and intelligent agents must cooperate in a shared environment. In this paper, we introduce an application of an explainability method based on the creation of a Policy Graph (PG). This graph is built using discrete predicates that represent and explain the behaviour of a trained agent in a cooperative environment with multiple agents. We also present a method to measure the similarity between the explanations obtained and the behaviour of the agent. This method consists of building a new agent with a policy based on the PG, in order to compare its behaviour with that of the original agent.

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

Resum

L'adopció d'algorismes basats en Intel·ligència Artificial (IA) ha augmentat ràpidament durant els darrers anys. Tanmateix, alguns aspectes d'aquestes tècniques estan sota un escrutini intens. Per exemple, en molts casos, no és clar si les seves decisions estan ben fonamentades ni tampoc si hi podem confiar. Tenir una resposta a aquestes preocupacions és crucial en molts dominis, com aquells en què els humans i els agents intel·ligents han de cooperar en un entorn compartit. En aquest article us presentem una aplicació d'un mètode d'explicabilitat basat en la creació d'un Policy Graph. Aquest graf es construeix fent servir predicats discrets que representen i expliquen el comportament d'un agent entrenat en un entorn cooperatiu amb múltiples agents. També presentem un mètode per mesurar la semblança entre les explicacions obtingudes i el comportament de l'agent. Aquest mètode consisteix a construir un nou agent amb una política basada en el PG, per després poder comparar el seu comportament amb el de l'agent original.

Paraules clau. IA Explicable, Aprenentatge per Reforç, Grafs de Polítiques, Aprenentatge per Reforç Multi-agent, Entorns Cooperatius

Resumen

La adopción de algoritmos basados en Inteligencia Artificial (IA) ha aumentado rápidamente durante los últimos años. Sin embargo, algunos aspectos de estas técnicas, están bajo un intenso escrutinio. Por ejemplo, en muchos casos, no está claro si sus decisiones están bien fundamentadas ni tampoco si podemos confiar en ellas. Tener una respuesta a estas preocupaciones es crucial en muchos dominios, como aquellos en los que los humanos y los agentes inteligentes deben cooperar en un entorno compartido. En este artículo, presentamos una aplicación de un método de explicabilidad basado en la creación de un Policy Graph. Este grafo, se construye usando a predicados discretos que representan y explican el comportamiento de un agente entrenado en un entorno cooperativo con múltiples agentes. También presentamos un método para medir la similitud entre las explicaciones obtenidas y el comportamiento del agente. Este metodo, consiste en construi un nuevo agente con una política basada en el PG, para luego poder comparar su comportamiento con el del agente original.

Palabras clave. IA Explicable, Aprendizaje por Refuerzo, Grafos de Políticas, Aprendizaje por Refuerzo Multi-agente, Entornos Cooperativos

Contents

List of Figures

List of Tables

Abbreviations

AI Artificial Intelligence.

BSC Barcelona Supercomputing Center.

CCIA International Conference of the Catalan Association for Artificial Intelligence.

GDPR European Union's General Data Protection Regulation.

IDE Integrated Development Environment.

MARL Multi-Agent Reinforcement Learning.

MDP Markov Decision Process.

ML Machine Learning.

NN Neural Networks.

NS New States.

PG Policy Graph.

PPO Proximal Policy Optimization.

RL Reinforcement Learning.

STD Standard Deviation.

TL Transferred Learning.

XAI Explainable Artificial Intelligence.

XRL Explainable Reinforcement Learning.

Chapter 1

Introduction

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 and natural language processing. It is reasonable to assume that the range of potential applications will keep growing bigger in future years. However, there are still many concerns about the transparency, understandability and trustworthiness [1] of systems built using these methods, esp. when they are based on so-called *blackbox* models [2]. For example, there is still a need for proper explanations of the behaviour of autonomous vehicles and this could be a risk for their real-world applicability and regulation [3].

Since Artificial Intelligence (AI) has an increasing impact on people's everyday lives, it becomes urgent to keep progressing in the field of Explainable Artificial Intelligence (XAI) [4]. In fact, there are already regulations 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 an explanation of an algorithmic decision that was made about them. Therefore, XAI is not only desirable but also a frequent requirement. This is also the case for virtual or physical agents trained via RL. Moreover, Explainable Reinforcement Learning (XRL) is starting to gain momentum as a different field of XAI.

This work aims to continue the line of research opened in [6, 7], which consists in producing explanations from predicate-based PG generated from the observation of RL-trained agents in the Cartpole environment. Here, we present an application of the same methodology in order to generate explanations for agents trained in a cooperative environment using Multi-Agent Reinforcement Learning (MARL) methods. In the physical world, cooperation between humans and AIs will gradually become more common [8], and thus we believe that it is crucial to be able to explain the behaviour of cooperative agents so that their actions are understandable and can be trusted by humans.

Currently, there are several approaches to explain RL agents. In this work, we briefly overview some of them in **Chapter 3**, we choose one that builds a graph that represents the agent's behaviour and we apply it to a MARL environment in **Chapter 5**, also giving insight in how to generate explanations. Once we apply the method, we build a new agent using the graph as a policy in order to compare both agents in **Chapter 6** and finally, we end with a summary of the main conclusions and contributions from the work done in **Chapter 8**.

On the other hand, we also provide the project planning (see **Appendix A**), an analysis of the budget (see **Appendix B**) and a sustainability report (see **Appendix C**).

1.1 Context

Within the Degree in Computer Engineering, taught by Facultat d'Informàtica de Barcelona, there are different mentions. This final degree project belongs to the mention of computing, specifically, the field of Artificial Intelligence. The project is carried out in the HPAI research group which is specialised among other topics, in RL.

This project aims to be the continuation of another work [7] carried out by the HPAI research group, which consists of applying explainability methods to an RL agent in a simple environment such as CartPole. In this environment, the agent played alone and had no interaction with other agents.

This work aims to investigate if the same XRL method can be used in a more complex environment that involves cooperation between agents. To do so, we will apply an approach for explainability based on the creation of a 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.

1.2 Stakeholders

Many parties are involved in this project, in a direct or indirect way, depending on the type of benefit or influence drawn from the work.

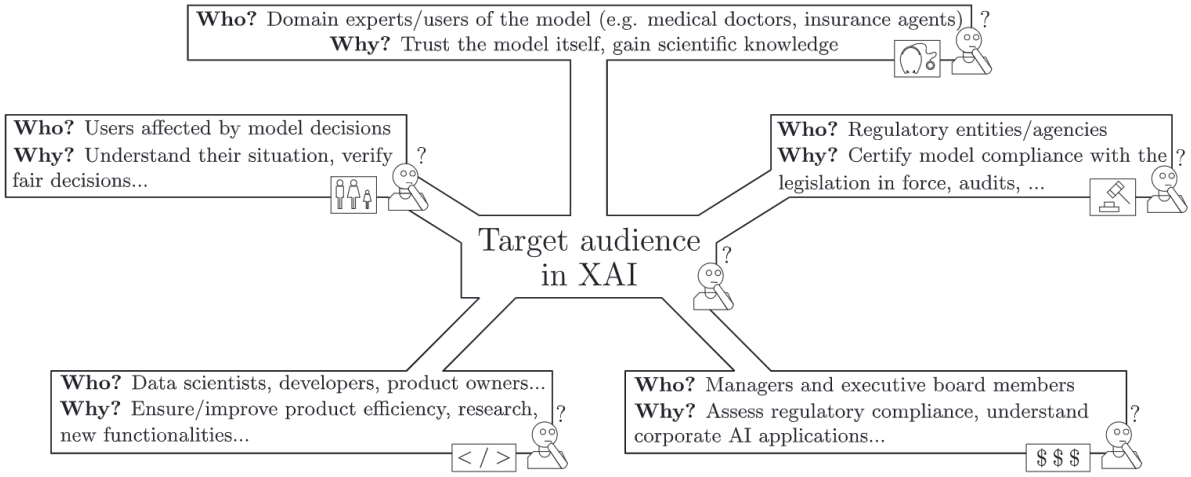


Figure 1.1: Diagram showing the different purposes of explainability in ML models sought by different audience profiles. Image partly inspired by the one presented in [9].

According to [9], in **Figure 1.1**, we can see the great variety of XAI stakeholders. For instance, a possible stakeholder could be a user affected by an AI decision in a medical context. On the other hand, regulatory entities/agencies need explainable AIs in order to certify the models' compliance with the legislation.

In this project, the director Sergio Álvarez Napagao and the co-director Dmitry Gnatyshak are stakeholders that have a direct implication on the thesis since it is one of their areas of study.

On the other hand, we have the stakeholders that do not directly interact with the project

but receive direct benefits. These stakeholders can be divided into three groups: companies, the scientific community and users.

Companies need to trust the models that they are using to offer the best experience to their users, also to certify model compliance with the legislation in force, etc. The scientific community would have access to this study and could use these results for other studies in this area. Finally, users that are affected by a model's decisions, need to understand the behaviour behind the model.

1.3 Scope

Once the basic concepts had been introduced, we can move forward to the objectives. Moreover, we will break down the objectives into sub-objectives and we also will explain both the project requirements and its possible obstacles and risks.

1.3.1 Objectives

As mentioned before, this project aims to do our bit to solve the problem of explainability in AI, especially in multi-agent cooperative environments. Although XAI is not new, there is still a lot of work to do, from testing state-of-the-art techniques in different environments in order to verify their performance to designing new methods. In this project, we are getting inside a very new field of XAI called XRL. Therefore, there is much to investigate.

The main objective of this project is to develop, implement and evaluate the use of an explainability method based on the construction of a Policy Graph that represents the agent's behaviour in a more complex environment that requires cooperation. To do so, we used the Overcooked game (**Section 1.4**).

The objective can be divided into the following sub-objectives:

1. **[O.1] Study state-of-the-art**: This phase consisted of doing research on the current XRL state-of-the-art. To do so, it was needed to read books and previous studies.
2. **[O.2] Choose XRL method**: Once we had studied the current state-of-the-art, it was needed to choose a method. In our case, we decided to choose an already existent method that had not been tested in a MARL cooperative environment yet.
3. **[O.3] Replicate the chosen paper**: After choosing the XRL method, we replicated the method and extended it in order to test it in a more complex MARL cooperative environment.
4. **[O.4] Validate the method**: Once we implemented the method, we had to validate its results and extract some conclusions.
5. **[O.5] Propose extensions and implement them**: This phase consisted of proposing and implementing possible improvements in the XRL method. Of course, we also tested them in order to see their contributions.

1.3.2 Requirements

After defining the objectives and sub-objectives of the project, now we are going to see the requirements.

1.3.2.1 Functional

First of all, we will talk about those functionalities that we want our program to have.

- We need a code that allows us to run an RL agent into the Overcooked environment multiple times.
- Later, we need a program that for each environment timestep it was capable of building a Markov Decision Process (MDP) graph, using the agent's interactions with the environment.
- Finally, we need to extend the method in some way that allows us to see if the explanations we are giving are related to the behaviour of the RL agent.

1.3.2.2 Non-Functional

Now, we will talk about the expectations of the program.

- We want the project to be open source so that anyone can use and extend it. For this reason, it is necessary to use free software.
- Use good programming practices, with a readable style and as less complex as possible.
- Document all the code and the experimentation well so that they can be used or easily consulted by other people.

1.3.3 Obstacles and Risks

In this section, we will try to anticipate the obstacles and risks that we may find throughout the project in order to minimise their impact. The main obstacles that we may find are the following:

- **Limited time**: In this project, there is a lot of work to do, therefore, it will require good planning in order to achieve all the objectives while meeting the deadline (see **Appendix A**).
- **Inexperience in the field**: Due to my inexperience in the field of RL and XAI, it could happen that I had to spend more time than expected.
- **Computational power**: Reinforcement learning is a computationally expensive field, especially in complex simulation environments. In our case, it is likely that we do not have this problem since our environment is in 2D and it is not very expensive computationally. However, there is a risk that the training will be too expensive for my computer.

1.4 Justification

This work focuses on the investigation of explaining cooperative environments. This is an area that is really interesting from the point of view that we are no longer talking about agents that act alone, but we are talking about agents capable of interacting with other agents or even humans. As we will mention in **Section 2.6**, this type of environment adds more complexity to the decision-making process since for example there is more uncertainty about the effect of the

agent's own actions. Research in this field may be crucial not only because of the ability to raise the quality of these agents' answers but also because it would facilitate their interaction with humans.

As we commented in **Section 1.1**, we implemented the explainability method used in [6, 7], but in our case, we applied it in a cooperative MARL environment. We decided to use this method for 3 main reasons.

The first one is because we think that natural language explanations are one of the easiest communication methods to understand. This method is easy to understand even for people who are not experts. This makes the explanations understandable to everyone. Moreover, the translation of predicates to natural language is a solved problem.

Secondly, the logic behind this method is not difficult to understand. Thanks to this, it was easier to do experiments and test different extensions of the method.

Finally, it is a method that had not been tested in a cooperative environment. For this reason, we decided to use it in order to know whether it was useful in these scenarios.

Regarding the RL agents, there are two possible approaches. On the one hand, we can take an agent who has already been previously trained and on the other hand, we can start training one from scratch. If we choose the first option, we have an advantage since we will not waste time training it. On the other hand, training an agent from the beginning gives us the advantage of having more freedom when choosing the game that we find most interesting. Eventually, we decided to use an already trained agent due to the limited time to do the thesis.

1.5 Methodology and Rigour

The methodology used for this project is the Kanban methodology, which its main objective is to manage how the tasks are completed in a general way.

Kanban is a Japanese word that means 'visual cards', hence this methodology uses visual notes where each one represents a task to be done. The cards will be on a board with four different columns:

- **To-do**: In this column, we can find all the tasks that have been defined but have not started yet.
- **Progress**: We will find all the tasks that we are doing at the moment.
- **Testing**: We will find all the tasks that have already been completed but it has not yet been verified that everything works correctly.
- **Completed**: Here we will find all those tasks that have been completed and tested successfully.

1.5.1 Resources

Now, we are going to talk about the different resources we used throughout the thesis.

Human resources

The main human resource of the thesis was the researchers Sergio Álvarez Napagao and Dmitry Gnatyshak. Both have mentored the research, so they were fundamental. Finally, Joan Sardà was the tutor in charge of correcting the project management part, so he had a big influence on the scope and organisation of the work.

Material resources

This project aims to expand another thesis [7], for this reason, some books and papers were needed as material resources. On the other hand, it was also needed some software and hardware resources to make the practical part.

- **Overleaf**: It is a useful cooperative tool to write documents with Latex. It was used to write all the thesis documentation.
- **Trello**: To keep track of work and manage all the tasks, we used Trello. Trello is an application that allows you to set up tasks using cards, lists and boards. This methodology is quite common since it is very easy to use as well as being very visual.
- **Atenea/Racó**: It was used to communicate with the GEP professor and for delivering the thesis when it was required.
- **Rocket.Chat**: It was used to communicate with Sergio and Dmitry. We used this tool because is the one they already used in HPAI research team.
- **PyCharm IDE**: PyCharm is a useful tool for coding python projects since it facilitates coding. For this reason, it was used for the practical work.
- **GitLab**: It was indispensable to use any tool to maintain version control of the code. In this project, we used GitLab because the HPAI research group has its own server and they opened a repository for the project.
- **Keynote**: It was used to create the presentation for the oral defence.
- **GanttPRO**: It was used to create the Gantt diagram of the project so we could have good project planning.
- **Computers**: It was used to perform all the practical parts. In this project, it was used a MacBook Pro with 16GB of RAM and an Intel Core i7 CPU

1.5.2 Validation

On the one hand, the quality of the explanations must be validated. As the authors comment in [6], the explanations should be validated by human experts with domain-specific knowledge. However, we will take the explanations and we will see if they make sense from a logical point of view.

On the other hand, we have to guarantee that the answers given by the explainability method, are based on the behaviour of the RL agents. To do so, we introduced a method for automatically validating it by building a new agent that follows a policy based on the PG. Once we have this new agent, we will compare its performance against the original one (RL).

1.6 Summary

In this first chapter, we have explained the motivation behind our research, describing the importance of explainability in the field of Artificial Intelligence and its different applications. We have seen that currently there is a lack of transparency, understandability and trustworthiness in these types of models. For this reason, this project aims to contribute our grain of sand to the construction of explainable AI agents.

In **Section 1.1** we have introduced the context of the project. In there, we have seen that we decided to replicate and extend another work [7] carried out by the HPAI research group, which consists of applying an explainability method to an RL agent in a simple environment such as CartPole. In our case, we extended the work in order to test whether the same method can be applied to a cooperative multi-agent environment. This method consists in producing explanations from predicate-based PG generated from the observation of RL-trained agents. Also, we have defined the structure of the document.

In **Section 1.2**, we have seen the large number of stakeholders involved. Later, in **Chapter 1.3** we have defined the scope of the work by introducing both the objectives and the requirements in addition to the obstacles and risks of the work.

Besides, in **Section 1.4**, we have justified some of the decisions made throughout the project. For instance, we have discussed the reasons for using the chosen XRL method and why we used pre-trained RL agents.

Eventually, in **Chapter 1.5**, we have seen the methodology used for carrying out the project and also the method to validate the results.

Now, it is time to introduce the basic concepts needed to fully understand our work.

Chapter 2

Background

As we mentioned in **Chapter 1**, in this chapter we are going to introduce some background knowledge in order to understand correctly the methodology followed in this work. All the content from this chapter is a gentle description of the different concepts that will be discussed throughout the project. Hence, we encourage revising references to get more detailed explanations about the concepts.

Thus, in this chapter, we are going to explain in more depth Reinforcement Learning (see **Section 2.1**) and the different paradigms in this field. Later, we will talk about important concepts in RL like agents, environment, and rewards (see **Section 2.2**) and policies (see **Section 2.3**). Later, we will explain what a Markov Decision Process is (see **Section 2.4**), as well as see an example to understand its logic. Finally, we will introduce the Explainability problem in the context of RL (see **Section 2.5**) and explain what Cooperative AI is (see **Section 2.6**), describing its challenges and its importance to integrate AI well in our society.

Finally, we will introduce some of the previous studies that have been important throughout the project.

2.1 Reinforcement Learning

All living beings exhibit some type of behaviour, in the sense that they perform some action in response to the signals they receive from the environment in which they live. Over time, some of them change their behaviour, so when receiving the same signal at two different times, can lead to different behaviours. In these cases, we can say that these living beings have been learning over time regardless of whether this learning has given them an advantage or not. This variation in behaviour is what we take advantage of when we try to teach a dog to sit when we ask them.

Precisely, the field of machine learning is in charge of studying the creation of algorithms capable of learning from their environment. The most popular types of machine learning are supervised[1] and unsupervised[2] learning. Although these two types are the best known today, there is a third class called Reinforcement Learning.

Unlike supervised learning, in RL, when the agent makes a mistake and does not take the correct action, it is not informed of what the correct action would have been. Instead, it is only informed that the action was not the appropriate one or how much error has been made. If we think about it, it seems that RL is the one that best suits what happens in nature when

[1]Type of algorithm that learns to map an input to an output based on example input-output pairs.
[2]Type of algorithm that learns patterns from untagged data.

analyzing physical agents. A very simple example would be when we want to teach a dog to sit. When it sits, he receives a cookie but if it does not, it may receive some kind of punishment (or simply not receive any reward). Repeating this action, the dog learns which behaviours are more rewarding than others.

There are two paradigms in the field of RL regarding the number of agents to be trained in the environment:

- **Vanilla RL**: There is only one agent in the environment. In it, the agent tries to maximise the total rewards of the environment. An example of this could be a robot trying to learn to walk.
- **MARL**: There are multiple agents in the environment. In it, we study how agents interact with each other (cooperation or competition). An example of this could be training a set of agents to play hide-and-seek[3] [10].

2.2 Agent, environment and rewards

First of all, we can define an agent as an entity capable of perceiving its environment, processing such perceptions and responding or acting in its environment in a rational way, that is, in a correct way and maximising an expected result.

Let us imagine an agent learning to play Super Mario Bros as a working example. The RL process can be modelled as a loop that works like this:

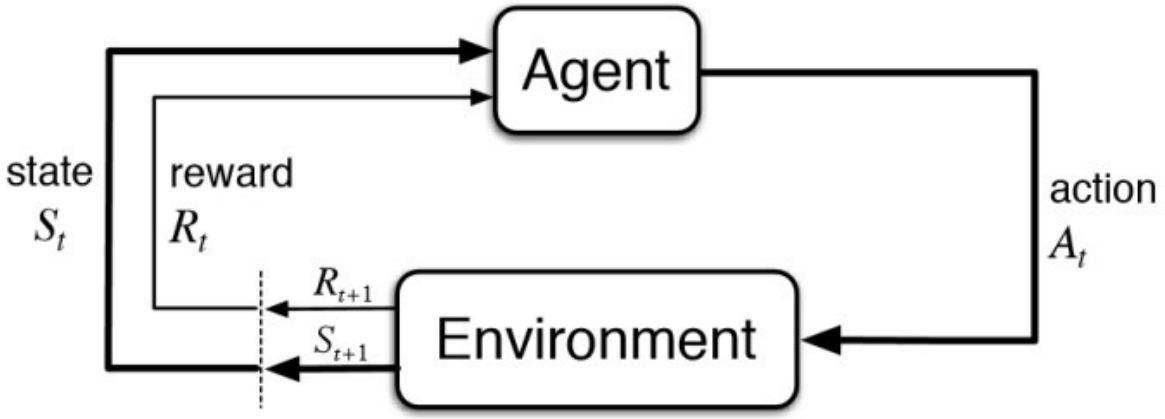


Figure 2.1: Agent-environment interaction loop. Image taken from [here]

1. Our agent receives a state S from the environment. In our case, we receive the first frame of our game (state) from Super Mario Bros (environment).
2. Based on that state S, the agent takes an action a, for example, moving to the right.
3. Environment transitions to a new state S' (new frame).
4. Environment gives some reward R to the agent, for example, not dead: +1 point.

This RL loop outputs a sequence of state, action and reward. The goal of the agent is to maximise the expected cumulative reward.

[3]Game where there are two teams. One team is in charge of hiding and the other tries to find them.

2.3 Policy

A policy π is what defines the agent's behaviour at a given time, therefore, the agent has to learn a policy function. This lets us map each state to the best corresponding action.

We have two types of policy:

- **Deterministic**: A policy that given a state, will always return the same action.
- **Stochastic**: A policy that given a state, will output a distribution probability over actions.

2.4 Markov Decision Process

The MDP is an outcome prediction model. The model attempts to predict an outcome given only the information provided by the current state. However, the MDP incorporates the characteristics of actions and motivations. At each step during the process, the decision-maker can choose to perform an action available in the current state, which moves the model to the next step and offers a reward to the decision-maker. We can represent an MDP as a graph where each node represents states and weighted edges represent probabilities of going from one state to another.

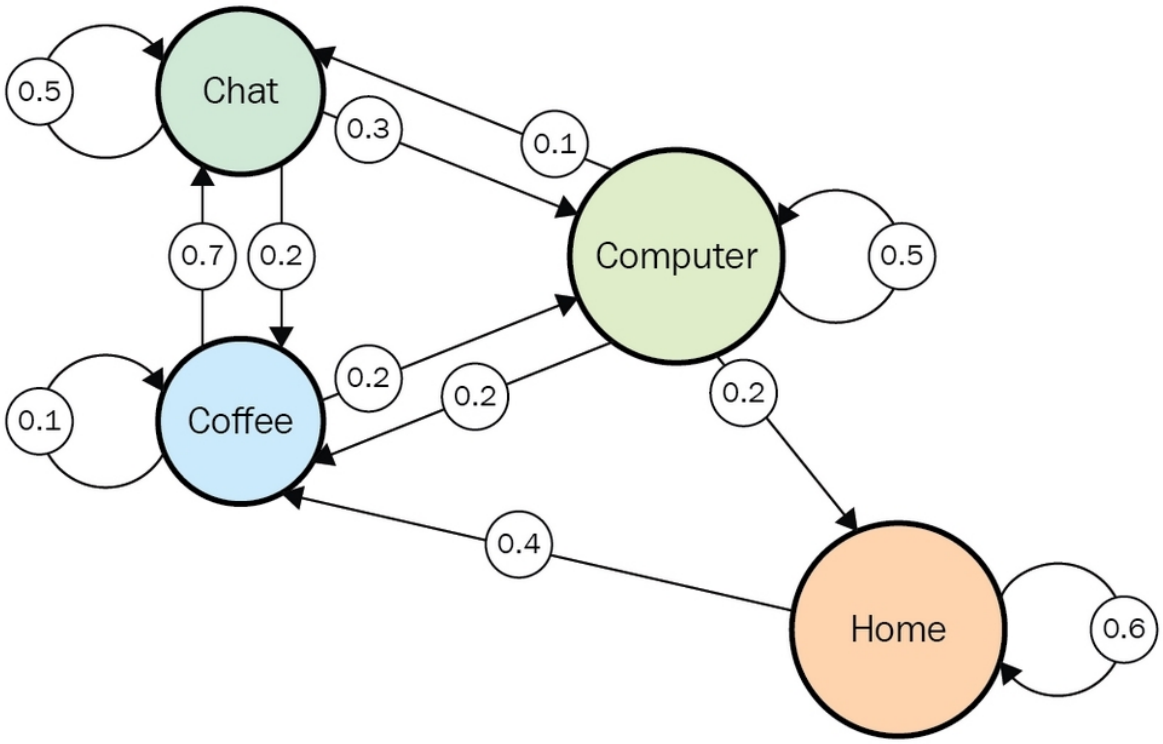


Figure 2.2: State transition graph with transition probabilities. Image taken from [11]

In **Figure 2.2** we can see an example of an MDP where for example, we have a 20% of probability of going from state *Computer* to state *Home*.

2.5 Explainability problem

In the context of Machine Learning (ML), we can use a number of different metrics to try to evaluate whether, for example, a robot is learning to perform its task correctly or not. Some of these metrics can be its accuracy, its reward with respect to an objective function, etc. Unfortunately, many times all this information is not enough to understand the behaviour or the reasoning behind its decisions.

When we train an RL agent, we try to find an optimal 'Policy' that allows the agent to decide which is the most appropriate action each time. To find such an optimal policy, currently, the use of Neural Networks (NN) is quite common. By using these types of techniques, we realise that policy is not decided by a program programmed by a human. This means that it tries to solve the problem in a logical way using procedural or rule-based methods. Unlike that policy is taken by an algorithm that bases its result on a set of training data. This type of agent has a disadvantage and that is that the more complex the model is, the more complicated it becomes to try to understand them. This causes that in many cases, the agent learns to solve her task, but humans are not able to explain her reasoning, thus becoming a kind of 'blackbox'.

So, -How can we explain the behaviour of an agent and its reasoning?- This question is part of the field of XAI [9].

According to [9], "In order to avoid limiting the effectiveness of the current generation of AI systems, eXplainable AI (XAI) proposes creating a suite of ML techniques that 1) produce more explainable models while maintaining a high level of learning performance (e.g., prediction accuracy), and 2) enable humans to understand, appropriately trust, and effectively manage the emerging generation of artificially intelligent partners. [...]".

Some of the utilities of making research in the field of XAI, could be:

- **Justifications**: By justifying the decisions of a model, people can understand and rely better on these models.
- **Monitoring and Debugging**: Understanding these models can help to find errors or unwanted decisions. Thus, we could modify the agent's behaviour in these situations.
- **Improvements**: Understanding these models also can help us to improve their behaviour and make them more intelligent.
- **Discoveries**: AI has achieved to beat humans in a lot of aspects, for example in chess. If we could understand their reasoning, humans would be able to learn and discover new strategies.

2.6 Cooperative AI

When we talk about AI, the first thing that comes to our mind is that of a solitary machine confronting a non-social environment. In other words, as a baby, we first want them to have a basic understanding of their environment and how to interact with it. On the other hand, there are also AIs that focus on multi-agent environments, but it is quite common for them to focus on two-player zero-sum games[4] such as chess or Go, games based on competition. Unfortunately, these scenarios are not the usual ones in the real world. For this reason, AI needs a social understanding and cooperative intelligence to integrate well into society.

[4]A zero-sum game is a situation where one person's loss in a transaction is equivalent to another person's gain.

According to [8], the authors claim that it is time to prioritise the development of cooperative AIs that are capable of agreeing on mutually beneficial joint actions, even when incentives are not fully aligned. Just as psychologists studying humans discovered that the infant's brain does not fully develop without social interaction, progress toward socially valuable AI requires placing the problem of cooperation at the centre of our research.

Cooperative AI can help us solve many of the cooperation problems we find in our daily lives. For example, most of the value of autonomous cars will come not from driving on empty roads, but from vehicles that coordinate seamlessly with the flow of pedestrians, cyclists, and human-driven vehicles. Thus, cooperative AI is not an alternative to autonomous intelligence but goes further.

However, the cooperative intelligence needed to achieve these goals requires among others, dealing with the following aspects:

- **Understanding**: The agents should take into account the consequences of their actions, they should also be capable of predicting another's behaviour as well as the implications of another's beliefs and preferences.
- **Communication**: The agents should have the ability to explicitly and credibly share information with others relevant to understanding behaviour, intentions and preferences.
- **Commitments**: The agents should make reliable promises when needed for cooperation.
- **Norms and institutions**: There is also needed some social infrastructure that helps understanding, communication and commitment between agents. For instance, shared beliefs or rules could be used.

2.7 Previous studies

In this section, we are going to see some of the previous studies we used in order to face our project.

In 2020, a paper that talked about XAI [9] aimed at providing newcomers to the field with a thorough taxonomy that can serve as reference material for doing research. One year later, another paper appeared talking about applying explainability methods in the field of Reinforcement Learning, among them, the Policy Graphs method [12]. Moreover, in 2022 appeared a third XRL survey [13] trying to give a systematic overview of existing methods in the explainable RL area and propose a novel unified taxonomy, building and expanding on the existing ones. More detailed information about these surveys will be explained in **Chapter 3**.

Here in this work, we have used, as a baseline, a bachelor's UPC thesis with the objective to apply the PG explainability method to an agent trained to play CartPole [7]. Fortunately, it was possible to explain the agent's behaviour with good results. It is at this time that the HPAI team questioned whether it was possible to apply the same method to an agent trained in another environment where cooperation with other agents (humans, AI, etc.) is required.

2.8 Summary

Altogether, in this chapter, we have introduced the basic necessary concepts to understand this work. We have explained the fundamentals of Reinforcement Learning and Markov Decision

Processes. Moreover, we have talked about the Explainability problem in the context of Artificial Intelligence and especially in Reinforcement Learning in addition to introducing Cooperative AI as a necessary line of research for the coming years.

Finally, we have briefly presented some of the previous studies that have had an important weight in the project. For instance, we emphasise that we have used a bachelor's UPC thesis [7] as a baseline for our work.

Next chapter, we are going to introduce some of the state-of-the-art explainability methods in the field of Reinforcement Learning.

Chapter 3

State of the art

The area of explainability in reinforcement learning is still relatively new, especially when dealing with policies as blackbox models. In this section, we will provide a brief overview of some state-of-the-art XRL methods as well as discuss in more depth the method chosen in this work. A more detailed study of the explainability methods in RL can be found in [13].

According to [13], XRL methods can be classified by their:

- **Scope of explanation** (global/local): Global models show the global strategy used by the agent while local models offer explanations for a specific decision/prediction or feature and help answer the question "Why did the model make this decision in this example".
- **Timing of explanation** (post-hoc/intrinsic): Post-hoc models use another method to generate explanations while keeping the original model while intrinsic ones introduce a new interpretable learning model.
- **Time horizon of explanation** (reactive/proactive): In reactive models the explanations are focused on the immediate moment, considering only a very short horizon and momentary information while proactive models focus on longer-term consequences.
- **Type of the environment** (deterministic/stochastic): We say that an environment is deterministic if an action performed in the environment causes a definite effect, if not, it is stochastic.
- **Type of policy** (deterministic/stochastic): What differentiates a stochastic policy and a deterministic policy, is that in a stochastic policy, it is possible to have more the one action to choose from in a certain situation.
- **Agent cardinality** (single-agent/multi-agent): A single-agent environment only has one agent interacting with the environment while in a multi-agent there is more than one.

Here, we will divide the XRL methods into reactive and proactive models.

3.1 Reactive methods

Reactive methods are those that try to answer the question "what happened?". For example, in a self-driving car context, a reactive answer to the question "Why did the car stop?" would be: "A pedestrian appeared in front of the car".

In this section, we are going to see different methods that use reactive explanations.

3.1.1 Policy simplification

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 take. For instance, Coppens et al. [14] 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.

3.1.2 Reward decomposition

Another family is reward decomposition, which tries to decompose the reward into meaningful components. In [15], Juozapaitis et al. decompose the Q-function into reward types to try to explain why an action is preferred over another. With this, they can know whether the agent is taking an action to be closer to the objective or to avoid penalties.

3.1.3 Feature contribution and visual methods

Another approach is feature contribution and visual methods, like LIME [16] or SHAP [17], 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. differentiate between gradient-based and perturbation-based saliency methods [18]. The first ones 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.

3.2 Proactive methods

According to [13], proactive methods are those that try to answer the question "why has something happened?". Once again, in a self-driving car context, a reactive answer to the question "Why did the car stop?" would be: "If the car had not stopped, it would have run over a pedestrian. The pedestrian would suffer injuries that might have been fatal. The car did not want to injure the pedestrian and so it stopped."

In this section, we are going to see different methods that use proactive explanations.

3.2.1 Structural causal model

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 [19]. According to [20], 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. generate explanations of behaviour based on a counterfactual analysis of the structural causal model that is learned during RL [21].

3.2.2 Hierarchical policy

Another approach tries to break down one task into multiple sub-tasks in order to represent different abstraction levels [22]. Therefore, each task can only be carried out if its predecessor tasks have been finished.

According to [23], in order to achieve interoperability, it is important that the tasks had been described by humans. For instance, [22] 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.

3.2.3 Relational reinforcement learning

In addition, there is another approach that combines relational learning or inductive logic programming with RL. The idea behind these methods [24] is to represent states, actions and policies using first order (or relational) language. Thanks to this, it is easier to generalise over goals, states and actions, exploiting knowledge learnt during an earlier learning phase.

3.2.4 Explanation in terms of consequences

Finally, another approach consists in building a MDP and following the graph from the input state to the main reward state [21]. This allows us to ask simple questions about the chosen actions. As an optional step, we can simplify the state representation (discretising it if needed). This step becomes crucial when we are talking about more complex environments [6]. 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 (discretisation 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 **Chapter 5**.

3.3 Summary

In summary, in this chapter, we have classified some of the state-of-the-art XRL methods into families following the work done in [13]. For each family of methods, we have introduced their basics in addition to some examples.

On the other hand, we have also emphasised the method used in this project. In our case, we have used a method that consists of building a policy graph by mapping the original state to a set of predicates (discretisation 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 **Chapter 5** in order to get explanations about its behaviour. At this point, we are ready to start applying the chosen XRL method. But first, in the next chapter, we will get a trained RL agent.

Chapter 4

Training

According to **Chapter 1**, this project aims to continue the line of research opened [7], but now, trying to use the same explainability method in a cooperative environment.

In this section, we will introduce Overcooked (see **Section 4.1**), the multi-agent cooperative environment that we will use to test the chosen XRL method. We will describe the game dynamics, its reward function and the different layouts we will use to do the experiments. Also, we will overview the state representation (see **Section 4.2**) in addition to some examples to clearly see the information available to the agent at each timestep. Eventually, we will explain how we have trained the RL agents (see **Section 4.3**) besides, their performance.

4.1 Overcooked environment

In this paper, we have used PantheonRL [25] package for training and testing an agent in Overcooked-AI. Overcooked-AI is a benchmark environment for fully cooperative human-AI task performance, based on the wildly popular video game Overcooked. 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 a 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. We have decided to use this game not only because it is 100% cooperative but also because it was easy to control the game execution through PantheonRL.

This game has infinite different layouts distributions, however, we have preferred to use several layouts instead of using just one. The reason why we have decided this is basically to be able to see if the layout we choose is an important factor to take into account when trying to explain the behaviour of one of its agents. Here in this work, we have worked with five different layouts: *simple*, *unident_s*, *random0*, *random1* and *random3* to test different situations.

According to [26], in **Figure 4.1**, "Cramped Room (*simple*) presents low-level coordination challenges: in this shared, confined space it is very easy for the agents to collide. Asymmetric Advantages (*unident_s*) test whether players can choose high-level strategies that play to their strengths. In Coordination Ring (*random1*), players must coordinate to travel between the bottom left and top right corners of the layout. Forced Coordination (*random0*) instead removes collision coordination problems and forces players to develop a high-level joint strategy, since neither player can serve a dish by themselves. Counter Circuit (*random3*) involves a non-obvious coordination strategy, where onions are passed over the counter to the pot, rather than being

Figure 4.1: Experiment layouts. From left to right: *simple*, *unident_s*, *random1*, *random0* and *random3*). Image taken from [26]

carried around. [...]"

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

Figure 4.2 shows two random states of the game. Now, we will see how the environment would represent these states and what information each agent receives at each timestep.

For example, in **Figure 4.2a**, each agent would receive the following data:

- **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.

On the other hand, in **Figure 4.2b**, the agents can observe the following information:

- **Player 1**: Position (2, 3) - Facing (0, 1) - Holding Soup
- **Player 2**: Position (1, 1) - Facing (-1, 0) - Holding None
- **Not owned objects**:
  - Soup at (3, 0) - with 1 onion and 0 cooking time.
  - Soup at (4, 1) - with 1 onion and 0 cooking time.

Later, in **Chapter 5**, we will see how we have discretised the state representation in order to build the policy graph.

4.3 Agent Training

The aim of this work is not to solve the Overcooked game but rather to analyse 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 Proximal Policy Optimization (PPO) [27] algorithm because it is one of the methods that has achieved the best results in [26]. Indeed, if we get good results with PPO, we should also get good results with other methods since this explainability method is independent from the RL method 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. The results are the ones in **Table 4.1**:

Layout	**Mean Episode Reward (MER)**	**Standard Deviation (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

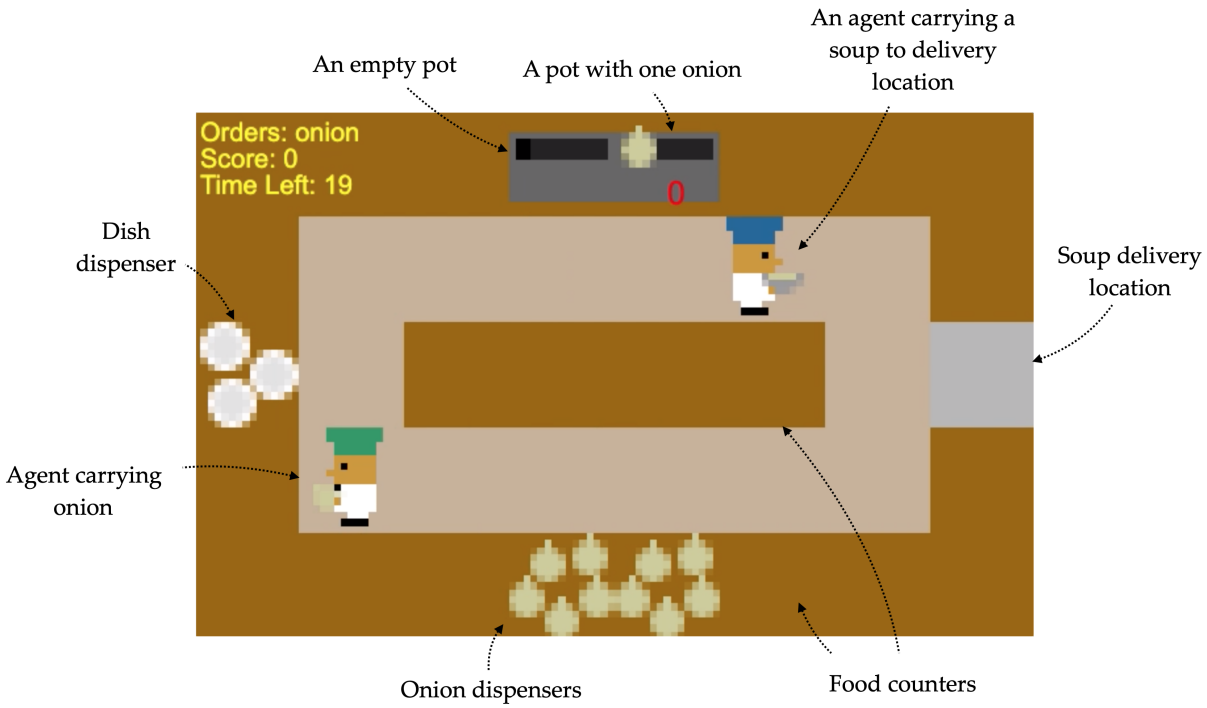
Table 4.1: RL agents results. [Own Creation]

4.4 Summary

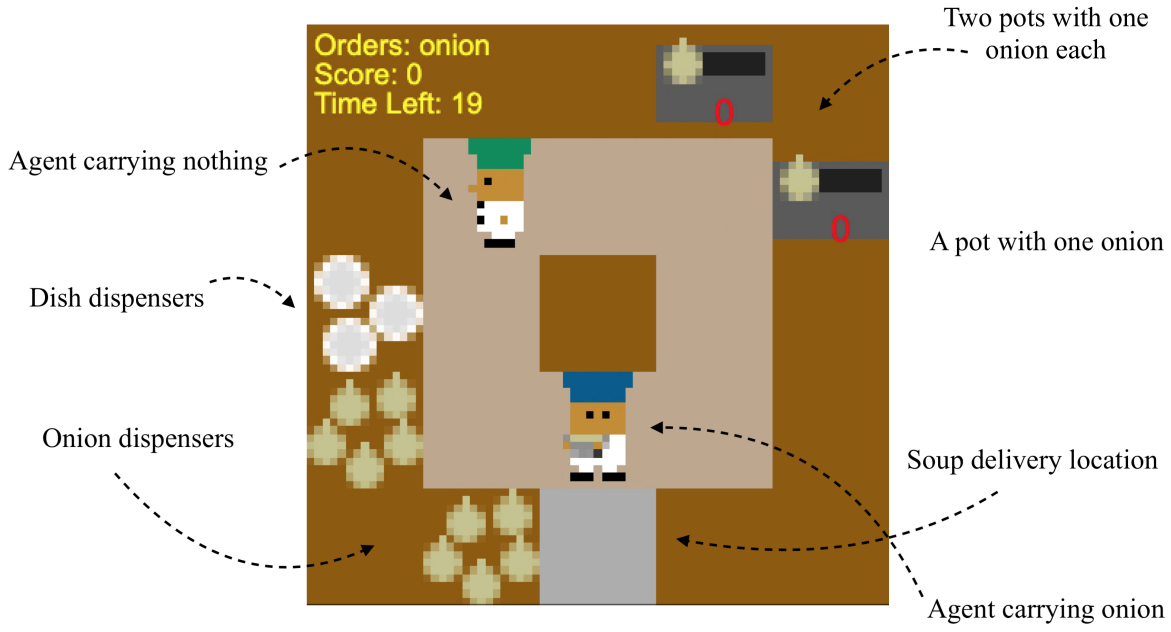
To sum up, in this chapter, we have introduced Overcooked, a multi-agent cooperative environment. In **Section 4.1**, we have explained its dynamics, its reward function and the different layouts used in this work. Besides, in **Section 4.2**, we have seen which information gets the agent from each observation of the environment in addition to seeing some examples.

On the other hand, in section **Section 4.3**, we have seen that our objective was not to solve the game but rather to get an agent that performs reasonably well in order to analyse their behaviour. Finally, we have trained 5 different agents (in different layouts) and we also have seen the results they achieved.

Now that we already have the RL agents, we are ready to analyse their behaviour using the chosen explainability method.



(a) Example of Overcooked-AI state representation in *random3* layout. Image taken from [here]



(b) Example of Overcooked-AI state representation in *random1* layout. [Own Creation]

Figure 4.2: Examples of Overcooked-AI game dynamics.

Chapter 5

Building a policy graph

In this chapter, we will introduce a set of predicates (see **Section 5.1**) in order to discretise the state representation seen on **Section 4.2**. Later, we will define some discretisers in order to test different sets of predicates and so, to compare which combination of predicates model's better the agent behaviour. Moreover, we will revisit the examples seen in **Section 4.2**, and we will see how they would look like if we discretise them.

In **Section 5.2** we will explain the logic behind the Policy Graph algorithm in addition to introducing two new approaches for this algorithm. For each of them, we will explain its implementation as well as an example to clearly understand it. Once we visited the PG algorithms, we will propose a state similarity function (see **Section 5.3**) in order to deal with unknown states when asking to the graph.

Finally, in **Section 5.4**, we will see how we extracted explanations from the PG, answering 3 questions proposed in [6]. For each of them, we will explain how to answer it helping us with pseudocode besides showing some examples from the built PGs.

5.1 State discretisation

In order to build the policy graph, we need to discretise the state representation. This step is crucial since we need to map each state to a node in our PG. We have created a total of 10 predicates to represent each state.

The first two are *held* and *held_partner*, which have 5 possible values depending on which object the agent and its partner, respectively, are holding (e.g., "O" for "Onion" object or "*" if is not holding anything). The third is *pot_state*, which 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].

To relate the actions of the agent with the relative position of the objects, we introduce another 6 predicates: *onion_pos*, *tomato_pos*, *dish_pos*, *pot_pos*, *service_pos* and *soup_pos*. All

of them with the same 6 possible values depending on the next action to perform to reach the object quickly (e.g., "T" for "Top" action). In order to simplify the writing of the document, from now on we will refer to these 6 predicates as *predicate_pos*.

The last one is *partner_zone*, that intends to help the agents cooperate along with *held_partner*. It has 8 possible values depending on which cardinal point the partner is located (e.g., "NE" for "North East").

On the other hand, in **Chapter 7** we will see that we implemented and tested 4 different discretisers (*D11*, *D12*, *D13*, *D14*) in order to test which one modelled better the agent behaviour. Each of them has the following predicates:

- **D11**: Has the predicates *held*, *pot_state* and *predicate_pos*.
- **D12**: Has D11 predicates plus *held_partner*
- **D13**: D13 has D11 predicates plus *partner_zone*
- **D14**: D14 has all predicates.

From now on, we will represent a discretized state writing each predicate value separated with dashes using the following order: *held*, *held_partner*, *pot_state*, *onion_pos*, *tomato_pos*, *dish_pos*, *pot_pos*, *service_pos*, *soup_pos* and *partner_zone*.

For instance, going back to the states shown in **Figure 4.2**, the resulting discretisation from the point of view of player 1 (blue chef) using *D14* would be:

- **Figure 4.2a**: *S-O-Of-Wa-R-S-L-L-L-R-L-SW*. This state means that the chef is holding a soup while his partner is holding an onion. One of the pots is turned off while the other has one or two onions. Moreover, he has to go to the right to reach an onion or the service delivery location, nothing to get a tomato, and left for reaching a dish, a soup or both pots. Finally, he knows that his partner is located in SW.
- **Figure 4.2b**: *S-*-Wa-Wa-L-S-L-R-R-I-R-NW*. This state means that the chef is holding a soup while his partner is not holding anything. Both pots have one or two onions. Moreover, he has to go to the left to reach an onion or a dish, nothing to get a tomato, right for reaching both pots or a soup and interact for reaching the service delivery location. Finally, he knows that his partner is located in NW.

Remember in this case, there are two pots for each layout, therefore, it is needed to have as many *pot_predicates* as pots in it.

5.2 Policy graph algorithms

As we mentioned in **Chapter 3**, the aim of the PG algorithm is to apply the same method as in **Figure 5.6**: record all the interactions of the original trained agent by executing it in a large set of random environments and build a graph relating predicate-based states and actions. To do so, we managed to modify the PantheonRL package to be able to control the execution of the game and therefore, be able to build our MDP. Now, we will see the code we used to build this PG.

```
1  def play_epoch(self, num_episodes, verbose=False):
2      """
3      Plays the agent one epoch.
4
5      Play an Epoch in current env in order to feed the PG.
6      An epoch it is formed by a number of episodes.
7
8      :param int num_episodes: Number of episodes to execute per seed
9      :param verbose: Prints additional information
10     """
11     rewards = []
12     for episode in range(num_episodes):
13         obs = self.env.reset()
14         done = False
15         reward = 0
16         info = None
17         while not done:
18             # We get the action
19             action = self.ego.get_action(obs, False)
20
21             if info is not None:
22                 obs_state = info['Obs_str']
23             else:
24                 obs_state = None
25
26             # Run step and save the observation
27             obs, new_reward, done, info = self.env.step(action)
28             obs_state_next = info['Obs_str']
29
30             # Update global reward
31             reward += new_reward
32
33             # Update PG with current obs and action
34             self.update_frequencies(obs_state, action, obs_state_next,
   verbose=verbose)
35
36         rewards.append(reward)
37
38     self.env.close()
39
40     # Compute the average reward and std
41     average_reward = sum(rewards) / num_episodes
42     std = np.std(rewards)
```

Figure 5.1: Code that uses a RL agent to play an epoch. [Own Creation]

In **Code 5.1**, we can see the function in charge of playing an epoch. Namely, it runs the agent for a number of episodes[1]. As we can see in the code, basically we have modified the *get_action()* method from PantheonRL in order to return in *info* variable the observation with the shape mentioned in **Section 4.2**. Note that since the RL agents are trained with PPO algorithm, the variable *obs* has a featurized observation so that the NN could be trained. Another important modification is the addition of line 34. This line of code calls the function *update_frequencies()* that is in charge of recording all the agent's interactions with the environment (see **Code 5.2**).

This function, first of all, discretise both the actual and the final state. Finally, adds 1 to the

[1]An entire game. In this project each game consists of 400 seconds

```python
def update_frequencies(self, obs: OvercookedState, act, next_state: OvercookedState, verbose=False):
    """
    Updates the attribute 'frequencies'.
    In this dictionary we save how many times the agent took act from obs to next_state.
    Given an observation, an action and the resulting observation, update frequencies correctly.

    :param obs: Actual state
    :param act: Action
    :param next_state: Resulting state
    :param verbose: Prints additional information
    """
    # Compute both predicates
    _, predicate = self.discretizer.get_predicates(obs)
    _, predicate_next = self.discretizer.get_predicates(next_state)

    self.frequencies[predicate][act][predicate_next] += 1
```

Figure 5.2: Code that records all the interactions of the agent with the environment and save them to the *self.frequencies* attribute. [Own Creation]

self.frequencies dictionary. This attribute saves for each discretised state all the actions that the agent has performed as well as the discretised state in which it has ended up.

At this point, we are only recording the agent's interactions with the environment but we have not built the policy graph yet. Now, we will see how we build it.

In **Code 5.3**, we can see how we implemented the feeding process (recording all the interactions in *self.frequencies* attribute). For each seed, the code generates a new game and runs as many episodes as we want. Finally, when the feeding process finishes, then we build the PG.

Our work has followed two approaches to building this graph:

- **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, only those 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. Regarding its implementation, in **Code 5.4** we can see how, indeed, it only considers the most probable action.

- **Complete Policy Graph**: This algorithm builds a multi-directed graph[2]. For each state, it takes all the agent interactions, adding them to our graph. This means that for each node, we have multiple possible actions with an associated probability. We think this could be an interesting approach since we are maintaining stochasticity. Regarding its implementation, in **Code 5.5** we can see how, indeed, it considers all the agent's interactions.

[2]A directed graph where can be multiple edges between two nodes.

```
def feed(self, seeds, num_episodes, verbose=False):
    """
    Runs the agent and builds the MDP

    :param seeds: List of seeds
    :type seeds: list or range
    :param verbose: Prints additional information
    :param int num_episodes: Number of episodes to execute per seed
    """

    for seed in seeds:
        self.params.seed = seed
        self.params.total_episodes = num_episodes

        # Generates the environment, agents...
        self.env, self.altenv, self.ego, self.alt = generate_game(self.
   params, verbose=verbose)

        # Play an epoch
        self.play_epoch(num_episodes=num_episodes, verbose=verbose)

    # Once we finished the feeding, then we build the Graph
    self.build_mdp(verbose=verbose)
```

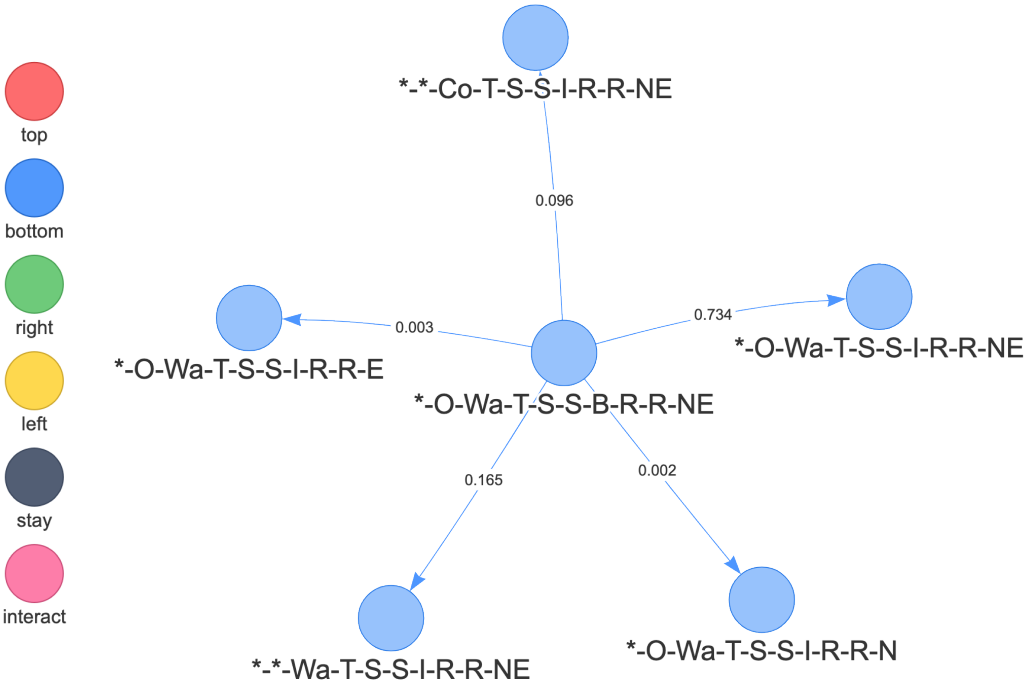
Figure 5.3: Code that runs an RL agent several epochs with different seeds. [Own Creation]

```python
def build_mdp(self, verbose=False):
    """
    Takes frequencies and builds the Markov Decision Graph.

    For each (node) we take the most frequent action.
    Then we add to the graph all the edges (node, best_action, next_node)
    """
    self.pg = nx.DiGraph(name='MDP')
    # For each state, we take into account the most probable action
    for state, actions in self.frequencies.items():
        action_keys = np.array(list(actions.keys()))

        freq_actions = np.array(
            [sum(list(actions[a].values())) for a in action_keys],
            dtype=np.int
        )
        freq_actions_sum = np.sum(freq_actions)

        prob_actions = freq_actions / freq_actions_sum
        prob_actions_sum = np.sum(prob_actions)

        action_more_prob = action_keys[np.argmax(prob_actions)]

        next_states = actions[action_more_prob]

        freq_next = np.array(
            [freq for _, freq in list(next_states.items())],
            dtype=np.int
        )
        freq_next_sum = np.sum(freq_next)

        prob_next_s = freq_next / freq_next_sum
        prob_next_s_sum = np.sum(prob_next_s)

        # Add new_edges to the Graph
        new_edges = list(
            zip(
                [state] * len(list(next_states.keys())),
                list(next_states.keys()),
                prob_next_s
            )
        )
        for _, next_state, prob in new_edges:

            # label: In order to show the edge label with pyvis
            color = get_assigned_color(action_more_prob)
            self.update_edge(
                state,
                next_state,
                int(action_more_prob),
                prob,
                color
            )
    nx.set_node_attributes(self.pg,
        {node: self.discretizer.get_predicate_label(node)
        for node, freq in self.frequencies.items()},
        name='label')
```

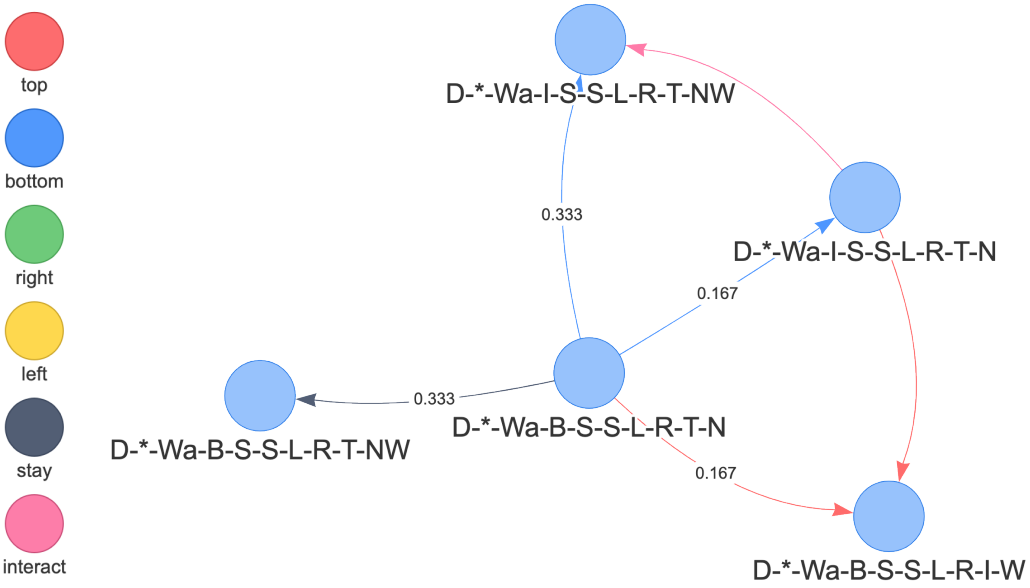
Figure 5.4: Code that builds a PG using the Partial PG algorithm. [Own Creation]

```python
def build_mdp(self, verbose=False):
    """
    Takes frequencies and builds the Markov Decision Graph.

    In this case, we add all the possible edges.
    Then we add to the graph all the edges (node, action, next_node)

    In this algorithm, each node can have multiple edges with different
    actions
    """
    self.pg = nx.MultiDiGraph(name='MDP')

    for state, actions in self.frequencies.items():
        # Times per action
        sum_partial = {
            action: sum(list(next_states.values()))
            for action, next_states in
                self.frequencies[state].items()}
        # Total times
        sum_total = sum(list(sum_partial.values()))

        # Percentage per action
        prob_sub_partial = self.frequencies[state].copy()
        prob_sub_partial = {
            action: {next_s: freq / sum_total
                for next_s, freq in prob_sub_partial[action].items()
            }
            for action, _ in prob_sub_partial.items()
        }

        new_edges = []
        for action, next_states in prob_sub_partial.items():
            new_edge = list(zip([state] * len(next_states.values()),
                                [action] * len(next_states.values()),
                                list(next_states.keys()),
                                list(next_states.values()),
                                )
                            )
            new_edges.append(new_edge)

        for edge_list_by_actions in new_edges:
            for u, a, v, p in edge_list_by_actions:
                # label: In order to show the edge label with pyvis
                color = get_assigned_color(a)
                self.update_edge(u, v, int(a), p, color)

        self.normalize_node_weights(state)

    nx.set_node_attributes(self.pg,
        {node: self.discretizer.get_predicate_label(node)
        for node, freq in self.frequencies.items()},
        name='label')
```

Figure 5.5: Code that builds a PG using the Complete PG algorithm. [Own Creation]



(a) Partial policy graph algorithm



(b) Complete policy graph algorithm

Figure 5.6: Examples of policy graph algorithms. Each edge has a weight (probability of choosing an action and ending up) and a colour (action). [Own Creation]

5.3 Nearby/Similar states

If we discretise the space as explained in **Section 5.1**, for instance in *simple* layout, we have 10 predicates and 37,324,800 potential states. During the construction of the PG, the algorithm will visit a large number of states but, probably not all of them. At this point, we asked ourselves: what happens if we ask for a valid but unknown state for the PG?

In order to solve so, we have introduced state similarity to deal with unknown states. We consider that two states (S_1, S_2) are similar when $diff\,(S_1, S_2) <= 1$, where $diff\,(S_1, S_2)$ 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 $diff\,(S_1, S_2) = 1$. We have considered that when there is a difference greater than 1, the two states become totally different.

5.4 Explainability algorithm

Once we have built our PG, we have access to the generation of natural language explanations as in [6], 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. However, the authors propose three questions to help explain the agent's behaviour. These questions are:

1. **What will you do when you are in state X?**
2. **When do you perform action X?**
3. **Why did you not perform action X in state Y?**

Our system is capable to answer any of these three questions. To do so, we use a script that allows us to ask for explanations. Before running the program, we need to define: which PG want to use, in which layout we want to play and which PG algorithm we have used for building it. Once we have defined all the previous parameters, then the program shows a menu like the one in **Figure 5.7**. Thus, the user can select the question he wants to ask the PG.

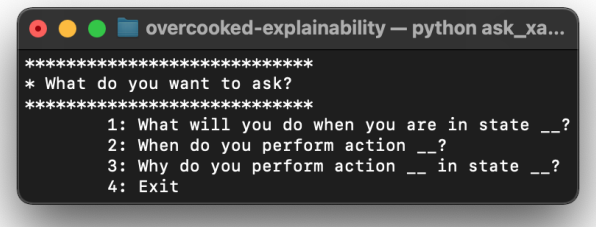


Figure 5.7: Explainability menu. [Own creation]

Now, we will go into detail in each of these questions, besides seeing some examples.

5.4.1 What will you do when you are in state X?

Described in **Figure 5.8**. The answer to this question consists of looking for the possible actions in the PG from the input state that the user wants to check. Here, we can consider 3 cases:

1. $X \in PG$: Returns the probability of taking each of the actions from X.
2. $X \notin PG$, but a similar state is found: Same as case 1 but using the similar state.
3. $X \notin PG$ and a similar state is not found: Returns a probability distribution over the actions where all of them has the same probability.

Algorithm 4: Characterize Situational Behavior

Input: Behavioral Model $G = \{V, E\}$, Communicable Predicate Library C, State region description d, Max action threshold $cluster\_max$
Output: Explanation of policy behavior in d per action and its accompanying state region

$S \leftarrow dict()$;
$descriptions \leftarrow dict()$;
DNF_description ← convert_to_DNF_formula(d, C);
foreach $s \in \{v \in V \mid$ *test_dnf(v,DNF_description) is True* $\}$ **do**
  $S[\pi(s)] \leftarrow S[\pi(s)] \cup s$;
  if $|S| > cluster\_max$ **then**
    return too_many_actions_error
  end
end
foreach $a \in S$ **do**
  descriptions[a] ← describe($S[a], V \setminus S[a], C$);
end
return descriptions;

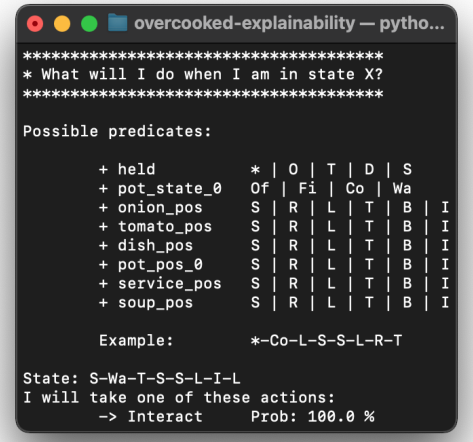
Figure 5.8: What will you do when you are in state X? Image taken from [6]

Here, the system needs to know which state you are asking for. Therefore, the system shows all the predicates that make up a state so that the user can enter a valid state. Once the user enters it, then the program executes the algorithm shown in **Figure 5.8**. When the computation finishes, the user can see the probability of choosing each of its possible actions.

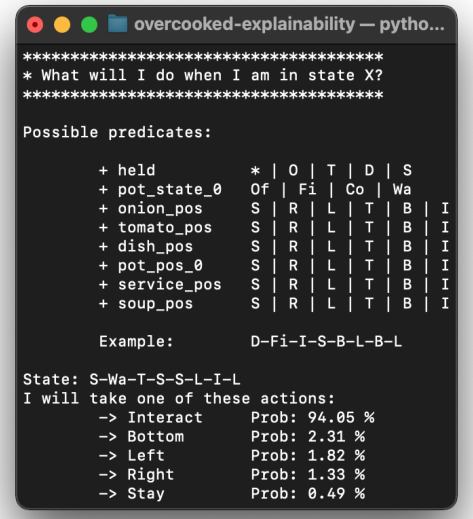
For example, in **Figure 5.9**, the user asks for the state *S-Wa-T-S-S-L-I-L*. From a logical point of view, make sense that in both cases the most probable action was interacting since the state shows us that the agent holds a soup and it is in front of the service.

5.4.2 When do you perform action X?

Described in **Figure 5.10**. The answer to this question consists of looking for those states where the action X is the most probable action.



(a) Partial PG. [Own creation]



(b) Complete PG. [Own creation]

Figure 5.9: Question 1. Using *D11* in *simple* layout. [Own Creation]

Here, the system needs to know which action you are asking for. Thus, the system shows all the available actions the agent can take. After that, the system executes the algorithm shown in **Figure 5.10**. Once the computation finishes, the user can see the list of states where action X is the most probable one.

For instance, in **Figure 5.11**, we can see that the user asks for *interact* action. On the one hand, in **Figure 5.11a** we can see the state **-Co-L-S-I-L-R-T*. According to it, the agent is not holding anything, there is a pot cooking onions and it is just in front of a dish. Taking this into account seems reasonable to think that the agent will take the dish to later go for the soup.

On the other hand, in **Figure 5.11b**, we can see the state **-Co-B-S-S-L-R-I*. In this case, we can deduce that the agent is interacting since it has a soup just in front, so it could serve it.

5.4.3 Why did you not perform action X in state Y?

Described in **Figure 5.12**. The answer to this question is to find out which predicates are responsible for deciding whether to take action X or not. To do so, we have decided to find the nearby (similar) states to Y and then, check if any of them performs action X. In that case, we compute the difference between both states in order to see what would be the difference. Here, we can consider 3 cases:

1. $Y \in PG$: Returns the difference between Y and each of the nearby states whose most probable action is X.

2. $Y \notin PG$, but a similar state is found: Same as case 1 but using the similar state.

3. $Y \notin PG$ and a similar state is not found: It returns that it doesn't know why it hasn't made that decision.

Algorithm 2: Identify Dominant-action State Region

Input: Behavioral Model $G = \{V, E\}$, Target Action a
Output: Set of target states S_{π^a} where a_t is the dominant action, Set of non-target states $S_{\pi^{*\setminus a}}$

$S_{\pi^a} \leftarrow \{\}$;
$S_{\pi^{*\setminus a}} \leftarrow \{\}$;
foreach $s \in V$ **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
return $S_{\pi^a}, S_{\pi^{*\setminus a}}$;

Figure 5.10: When do you perform action X? Image taken from [6]

If, on the contrary, the algorithm does not find any nearby state whose most probable action is X, then the algorithm responds that it has no explanation for not choosing the action X. This means that we can only provide an explanation as long as X is the most likely action of one of the neighbouring states.

In this case, the system needs to know which action and state you are asking for. Thus, the system shows all the available actions the agent can take and also all the predicates that make up a state. After that, the system executes the algorithm shown in **Figure 5.12**. Once the computation finishes, the user can see the list of states. For instance, in **Figure 5.13**, we can see that the user asks why the agent did not perform *top* action in state *D-Co-L-S-I-T-B-L*. On the one hand, in **Figure 5.13a** we can see that the agent returns a set of states where it would choose the action *top* instead of going to the left. It would choose to go up if the nearest dish, pot and service sources were on the right and the nearest soup was on top. Taking this into account seems reasonable to think that the agent would go up in order to get the soup.

On the other hand, in **Figure 5.13b**, we can see the state the input state does not exist in the MDP. Nevertheless, the explainability method finds a similar state *D-Co-L-S-S-T-B-L*. Similarly to **Figure 5.13a**, we can deduce that the agent would go up instead of going to the left if the nearest pot and service were at the right and the nearest soup was on top. Also in this case seems that the agent intends to get the soup.

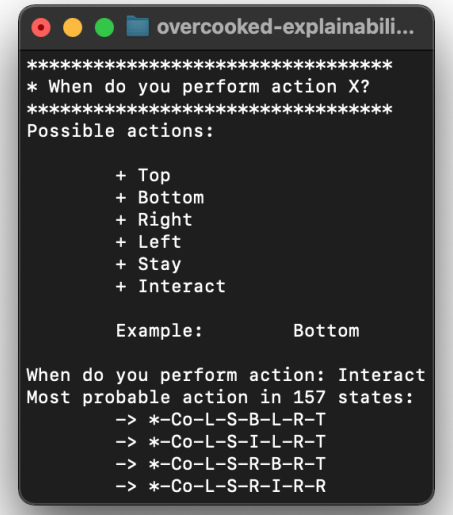
Although many of the answers to this question are difficult to evaluate from a logical point of view, the truth is that they can help us understand what strategy each agent follows to cooperate with its partner.

5.4.4 Explainability conclusions

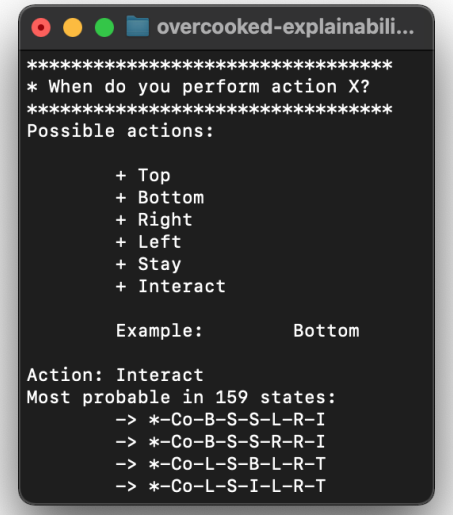
As we mentioned in **Chapter 1**, one of the objectives of this project is to test if we can get valid explanations from an RL agent behaviour using the explainability method used in [7] in a multi-agent cooperative environment. In this section, we have seen some examples of the explanations given by our PGs.

We would like to emphasise the fact that not all the explanations given by the PG are so easy to interpret or understand at first sight.

For instance, in **Figure 5.13a**, we have seen that we asked the PG why did not perform *top* action in the state "D-Co-L-S-I-T-B-L". In this case, the graph answered that he would have



(a) Partial PG. [Own creation]



(b) Complete PG. [Own creation]

Figure 5.11: Question 2. Using *D11* in *simple* layout. [Own Creation]

chosen to go to the left, but here is not so clear the reason behind this decision, at least at a first sight. We could make a hypothesis about the strategy that each agent is playing and maybe we could understand its decision. There is also the possibility that in this state the agent is not taking the more appropriate action. Due to the limitations of time, we could not analyse in more depth these aspects.

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 valid explanations from the PG.

5.5 Summary

In this chapter, we have applied the chosen explainability method to the agents we got in **Chapter 4**. To do so, in **Section 5.1**, we have introduced a set of predicates in order to map each state to a node in our PG. Eventually, we have proposed 10 predicates (*held*, *held_partner*, *pot_state*, *onion_pos*, *tomato_pos*, *dish_pos*, *pot_pos*, *service_pos*, *soup_pos* and *partner_zone*). Besides, we introduced 4 different discretisers (*D11*, *D12*, *D13*, *D14*) to test which subset of predicates models better the agent's behaviour. We have also seen two examples of discretised states.

In **Section 5.2**, we have seen the logic behind the Policy Graph algorithm in addition to proposing two different implementations of this algorithm (Partial and Complete). Also, we have seen examples of both algorithms. Furthermore, we have also shown some code implementations in order to make it easier to understand the logic behind these algorithms.

Algorithm 3: Identify Behavioral Divergences

Input: Behavioral Model $G = \{V, E\}$, 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.

$S_{\pi^a} \leftarrow \{\}$;
$S_{\pi^{*\setminus a}} \leftarrow \{\}$;
foreach $D \in \{1, ..., D_{const}\}$ **do**
  foreach $s \in \{v \in V \mid distance(v, s_p) \leq D\}$ **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
end
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;

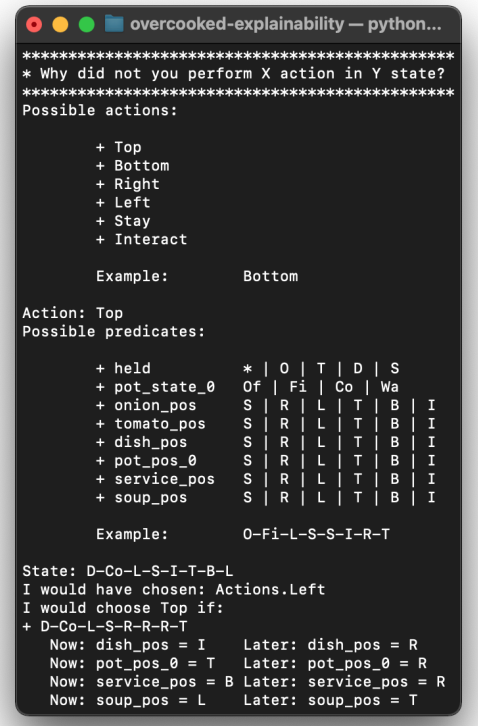
Figure 5.12: Why did you not perform action X in state Y? Image taken from [6]

Later, in **Section 5.3**, we have introduced state similarity in order to deal with unknown states. Eventually, in **Section 5.4**, we explained how this PG can help us to give explanations. To do so, we used three algorithms proposed in [6] to answer three questions. For each of these questions, we overviewed the algorithm that gives the answer and also we showed real Overcooked examples.

Finally, we concluded that we have 2 types of explanations. On the one hand, we have seen that those explanations that can be analysed from a human perspective make sense and therefore, are very useful to understand the reasons behind the agent's behaviour. On the other hand, we have those explanations that require more attention since we would need to analyse in more depth the strategy of each agent. However, we have met the goal of extracting valid explanations from the PG.

However, we have seen that even though some explanations are not as clear as we wanted (at least at first glance), we have explanations about why the agent is taking each action. Therefore, we can use this method to better understand under what circumstances the agent makes certain decisions.

Once we have implemented the explainability method and we have overviewed its explanations, it is time to validate that the answers given by the PG are based on the behaviour of the RL agents.



(a) Partial PG. [Own creation]

(b) Complete PG. [Own creation]

Figure 5.13: Question 3. Using *D11* in *simple* layout. [Own Creation]

Chapter 6

Policy Graphs

Once we built the MDPs and had all the explainability algorithms, we were ready to check the results. But, how do we guarantee that the answers given by the PG are based on the behaviour of the RL agents? The fact is that we could analyse whether the answers had more or less sense from a human perspective but it was not enough to answer the previous question. For this reason, let us introduce our novel contributions in order to validate this method.

In this chapter, we will introduce a new method for automatically validating whether the answers given by the PG are based on the behaviour of the RL agents (see **Section 6.1**). Moreover, we will talk about the different discretisers we have used throughout the project in order to model the agent's behaviour.

6.1 Building Policy Graph-based agents

In order to validate the explainability method, we decided to build another agent that follows a policy based on the PG we obtained as explained in **Chapter 5**: at each step, the agent receives the current state and decides its next action by querying the PG for the most probable action from the most similar state to the current one. In order to test this new agent, we run it multiple times in random environments. Therefore, assuming the agent is in state S, we can distinguish 3 cases:

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: Picks a random action.

Now, we will see how we implemented the code in order to validate the PGs.

As we can see in **Code 6.1**, the function is very similar to the function *play_epoch* seen in **Code 5.1**. In this case, before choosing an action it is necessary to discretise the observation. After this step, the code calls to the *select_action_using_mdp()* function in order to ask the PG which action has to perform. How is normal, each of the PG algorithms has a different implementation of this function. While in **Code 6.2** the implementation asks for the unique possible action, in **Code 6.3** the function manages to select an action following the probability distribution.

Regarding the state discretisation, at first we proposed to use two basic predicates: *held*, and *pot_state*. Later, we realised 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 we cannot relate the actions we execute with the relative position of the objects. For this reason, we introduced another 6 predicates: *onion_pos*, *tomato_pos*, *dish_pos*, *pot_pos*, *service_pos* and *soup_pos*. Using these 8 predicates led us to achieve good results in most of the layouts.

Nonetheless, we ask ourselves whether the introduction of cooperative predicates could improve results since the cooperative nature of the environment. For this reason, we decided to add the last two predicates to help the agents cooperate with each other: *partner_zone* and *held_partner*. As we will see in section **Section 7.1**, the introduction of these predicates notably improved the results in some layouts.

6.2 Summary

In this chapter, we have managed to answer the question: how do we guarantee that the answers given by the PG are based on the behaviour of the RL agents? To do so, in **Section 6.1** we have introduced a new method that consists in building another agent that follows a policy based on the PG. The idea behind this new method is to compare both agents' performance (RL and PG). In the end, if the PG agent gets similar results to the original RL agent, then we can conclude that both have similar behaviour. We have also shown some implementations to make it easier to understand the logic behind the validation algorithm.

On the other hand, we overviewed the different discretisers we used throughout the project. Starting with *D11*, that only has the predicates *held*, *pot_state* and *predicate_pos* and ending with *D14*, that has all the predicates.

Now, in **Chapter 7**, we will compare the performance of the original agents against the PG agents.

```python
def test_epoch(self, num_episodes, verbose=False):
    """
    Plays the MDP agent for one epoch.
    """
    rewards = []
    for episode in range(num_episodes):
        self.env.reset()
        done = False
        reward = 0
        actual_state = None
        next_state = None

        while not done:
            # Get the predicate (actual state) and take an action
            predicate_dict, predicate = self.discretizer.get_predicates(actual_state)
            action = self.select_action_using_mdp(predicate, verbose=verbose)
            _, new_reward, done, info = self.env.step(action.value)

            if info is not None:
                next_state = info['Obs_str']

            # Update global reward
            reward += new_reward

            actual_state = next_state
            self.pg_metrics['Visited States'] += 1

        rewards.append(reward)

    self.env.close()

    # Compute the average reward and std
    average_reward = np.sum(rewards) / num_episodes
    std = np.std(rewards)

    self.pg_metrics['AER'].append(average_reward)
    self.pg_metrics['STD'].append(std)
```

Figure 6.1: Code that runs an epoch using a PG to take decisions. [Own Creation]

```
def select_action_using_mdp(self, predicate, verbose):
    """
    Given a predicate, goes to the MDP and selects the corresponding
    action.

    :param predicate: Existent or non-existent state
    :param verbose: Prints additional information
    :return: Action
    """

    # Predicate does not exist
    if predicate not in self.pg.nodes:
        self.pg_metrics['new_state'] += 1

    nearest_predicate = self.get_nearest_predicate(predicate, verbose=
    verbose)
    possible_actions = self.get_possible_actions(nearest_predicate)

    # Possible actions always will have 1 element since  for each state
    we only save the best action
    return possible_actions[0][0]
```

Figure 6.2: Code that asks to the PG which action to perform in a specific state using Partial PG algorithm. [Own Creation]

```python
def select_action_using_mdp(self, predicate, verbose):
    """
    Given a predicate, goes to the MDP and selects the corresponding action.

    :param predicate: Existent or non-existent state
    :param verbose: Prints additional information
    :return: Action
    """
    # Predicate does not exist
    if predicate not in self.pg.nodes:
        self.pg_metrics['new_state'] += 1

    nearest_predicate = self.get_nearest_predicate(predicate, verbose=verbose)
    possible_actions = self.get_possible_actions(nearest_predicate)

    # Probability distribution
    p = [data[1] for data in possible_actions]
    a = [data[0].value for data in possible_actions]

    sum_prob = sum(p)
    assert (sum_prob > 0.999 and sum_prob < 1.001), f"{sum(p)} - {p}"

    # Take one action with a given Probability distribution
    p = np.array(p)
    p /= p.sum()
    rand_action = np.random.choice(a, p=p)
    rand_action = Actions(rand_action)

    return rand_action
```

Figure 6.3: Code that asks to the PG which action to perform in a specific state using Complete PG algorithm. [Own Creation]

Chapter 7

Experimental results

Once we have explained the validation method, we will validate all the PG built in **Chapter 5**. To do so, in this chapter, we compare the RL agents against PG agents using 3 metrics (see **Section 7.1**). Moreover, we explain how we have performed these tests.

In order to make the experiments, we have used the 5 different layouts mentioned in **Section 4.1** (*simple*, *unident_s*, *random0*, *random1* and *random3*). For each layout, we have used the 4 different discretisers introduced in **Section 5.1**. Additionally, for each discretiser, we have tested both Partial and Complete algorithms explained in **Section 5.2**.

In summary, we have created 2 PG algorithms $\times$ 4 discretizers $\times$ 5 layouts $=$ 40 different PG agents. From now on, we will use the notation *simple_P_D11* to refer to a PG agent fed in *simple* layout, using Partial algorithm and discretiser D14.

Finally, we will draw the conclusions of the presented results in **Section 7.2**.

7.1 Policy Graph agents

Now, we are going to see whether the PG agents have similar behaviour to the original RL agents. To do it, we have used 3 metrics to compare RL agents against PG agents:

- **Transferred Learning (TL)**: The ratio between the RL agent's average episode reward and the new agent's average episode reward. Namely, having 100 % TL, means that the PG agent has achieved the same average episode reward as the original one.

- **Standard Deviation (STD)**: The ratio between the RL agent average standard deviation of reward and the new agent's standard deviation of reward.

- **New States (NS) visited**: The proportion between previously unknown and total visited states.

All the PG agents have been fed[1] using batches of 25 seeds. Altogether, the training consisted of 500 seeds and 3 episodes per seed.

[1]Run the agent in order to record all its interactions with the environment to later build the PG

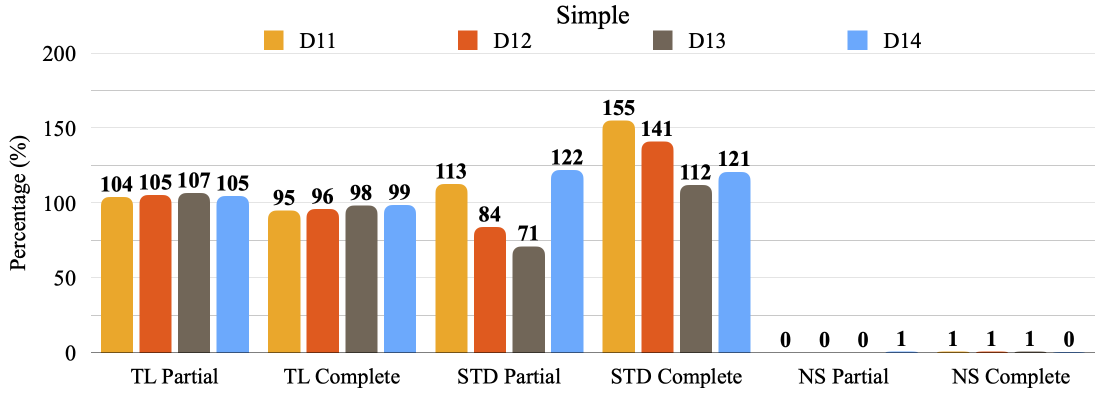


Figure 7.1: Average results of the last 250 seeds from layout *simple*. [Own creation]

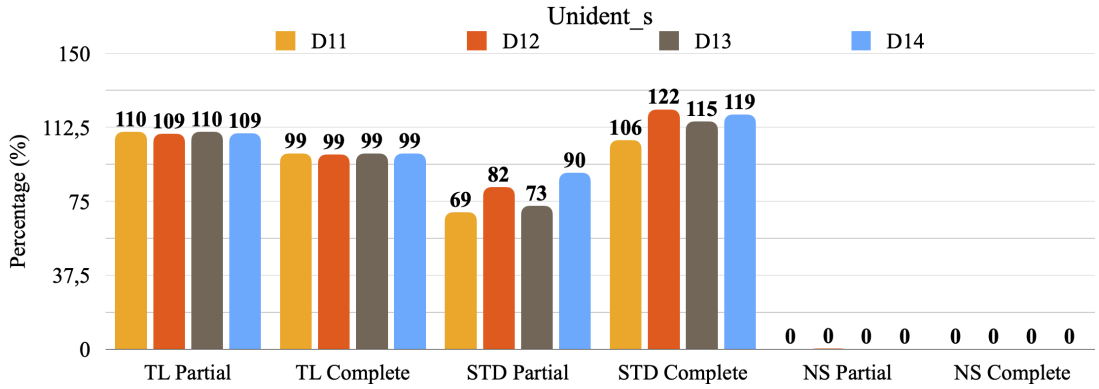


Figure 7.2: Average results of the last 250 seeds from layout *unident_s*. [Own creation]

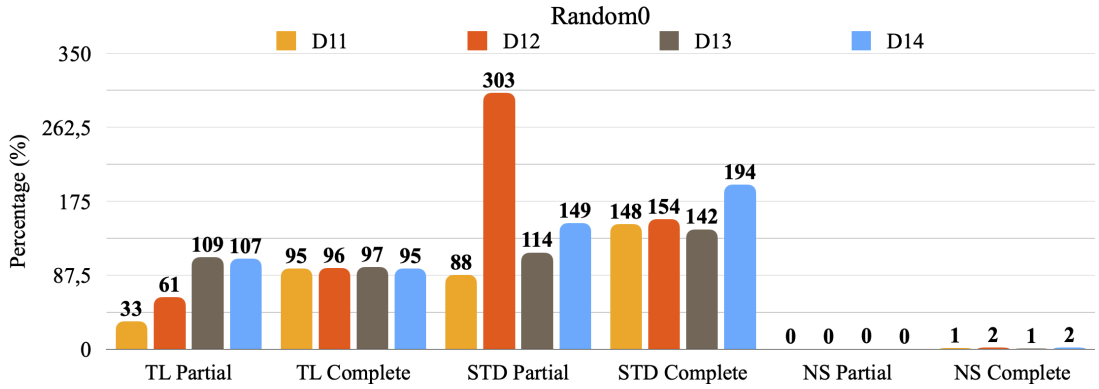


Figure 7.3: Average results of the last 250 seeds from layout *random0*. [Own creation]

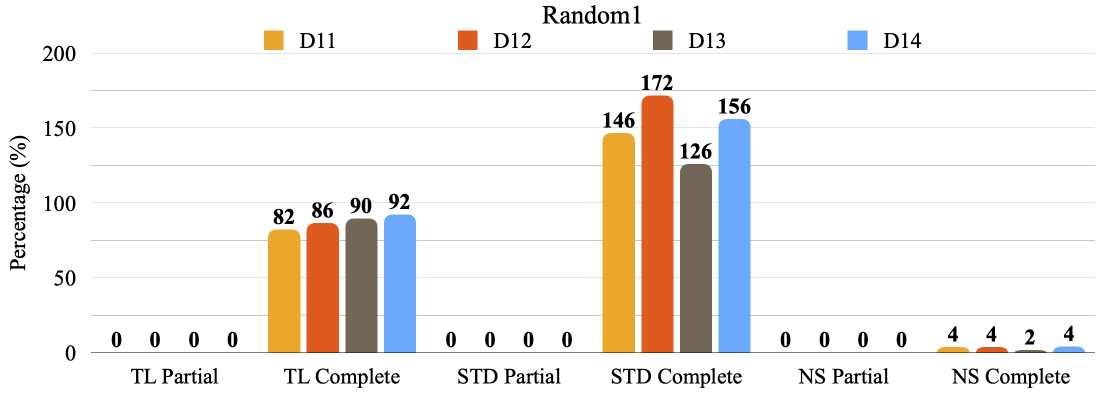


Figure 7.4: Average results of the last 250 seeds from layout *random1*. [Own creation]

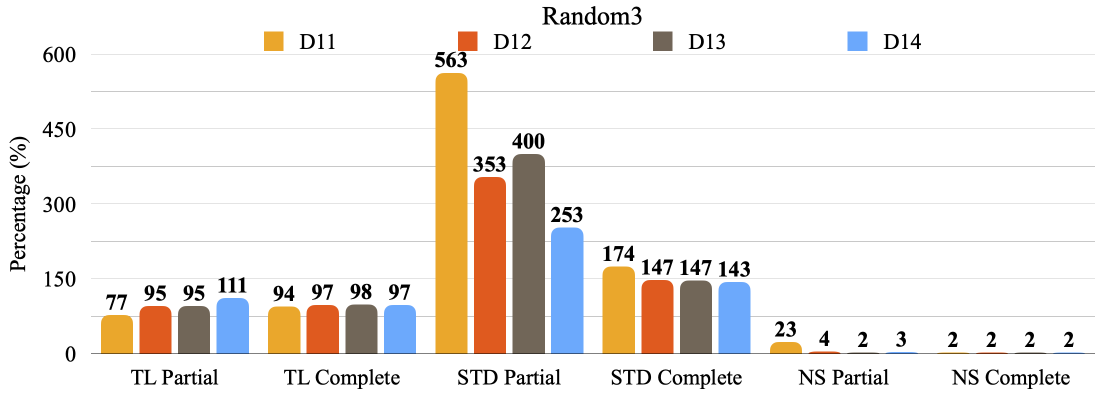


Figure 7.5: Average results of the last 250 seeds from layout *random3*. [Own creation]

Figures [7.1-7.5] show the results we have obtained from the different agents. We can see that in the *simple* and *unident_s* scenarios, the Partial agents manage to outperform the original ones while the Complete agents do not. On the other hand, it seems that in these cases it is not necessary to introduce cooperative predicates to explain their behaviour, unlike in 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. We can also 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 the original agent's, likely because the PG we have 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.

7.2 Conclusion

To summarise, according to the results we obtained in **Figures** [7.1-7.5], we have managed to build new agents capable to play Overcooked, only asking the PG what action to perform at each state. Indeed, there are PGs that have managed to outperform the original ones since there are graphs that achieve TL rates greater than 100%.

Analysing all the new agents, we have seen that the Complete algorithm is more stable regarding the 3 metrics than the Partial one, as the latter can score zero points or perform better

than the original agent depending on the layout, probably due to its deterministic nature.

On the other hand, 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 where this information is crucial, which makes sense due to the fact that the layout influences the need for cooperation.

7.3 Summary

In this chapter, we have tested 40 different agents using different discretisers (*D11*, *D12*, *D13*, *D14*), policy algorithms (Partial and Complete) and layouts (*simple*, *unident_s*, *random0*, *random1* and *random3*). We have also introduced three different metrics to compare each of these agents against their corresponding RL agent. The first one was TL, the ratio between the RL agent's average episode reward and the new agent's average episode reward. The second one was STD, the ratio between the RL agent average standard deviation of reward and the new agent's standard deviation of reward. And lastly, NS visited, the proportion between previously unknown and total visited states.

After analysing the results, we have seen that the Complete algorithm is more stable than the Partial one, as the latter can score zero points or perform better than the original agent depending on the layout.

On the other hand, regarding the used predicates, 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 where this information is crucial.

Now we have checked that this XRL can be used to explain a multi-agent cooperative environment, we can move forward to the conclusions chapter.

Chapter 8

Conclusions and Future Work

Once we have seen all the results, it is time to draw conclusions. In this chapter, we are going go to present the main highlights of the work (see **Section 8.1**), besides reviewing the initial objectives (see **Section 8.2**). Moreover, we have reviewed the technical competences (see **Section 8.3**) and also the contributions to my career (see **Section 8.4**). Finally, we are going to describe some interesting future lines of research (see **Section 8.5**).

8.1 Main Highlights

XAI is a research area that is growing by leaps and bounds 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 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. For this reason, 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.

This work aims to continue the line of research opened in [6, 7], which consists in producing explanations from a predicate-based Policy Graph generated from the observation of RL-trained agents in the Cartpole environment. Our contribution is to develop, implement and evaluate the use of the same method in a multi-agent cooperative environment.

To do so, we first had to study the most basic concepts such as RL, XAI and MDP (see **Chapter 2**). Once these concepts were learnt, then we did research on the current XRL methods (see **Chapter 3**). Although most of this research was done at the very beginning, we have been reading and learning about the topic throughout all the thesis.

Once we had the background needed, in **Chapter 4**, we trained 5 different RL agents that were capable to play reasonably well in Overcooked.

Thus, in **Chapter 5** we started implementing, applying and extending the chosen explainability method. In our case, the method consists of building a policy graph by mapping the original state to a set of predicates (discretisation step) and then repeatedly running the agent policy, recording its interactions with the environment. First of all, we had to propose a set of predicates in order to discretise the state representation (see **Section 5.1**). Moreover, we proposed a set of discretisers to test which subset of predicates modelled better the agent's behaviour. Later, we implemented and extended the PG algorithm. In our case, we proposed two different approaches for extending this algorithm: Partial (deterministic) and Complete (stochastic) PG algorithm.

Once we have built the policy graphs, then we had access to the generation of natural language explanations. In **Section 5.4**, we explained how these PGs could help us to give explanations. To do so, we used three algorithms proposed in [6] to answer three questions. For each of these questions, we overviewed the algorithm that gives the answer and also we showed real Overcooked examples.

Regarding the quality of the explanations, in **Section 5.4.4**, we are thrilled to say that we have managed to get very useful explanations. However, in this section, we also mention that even though some explanations are not as clear as we wanted (at least at first glance), we have explanations about why the agent is taking each action. Therefore, we can use this method to better understand under what circumstances the agent makes certain decisions.

After validating that in fact, we achieved to get valid explanations, in **Chapter 6**, we asked ourselves how do we guarantee that the answers given by the PG are based on the behaviour of the RL agents? To answer this question, we decided to propose a new method for validating it. This new method consists in building another agent that follows a policy based on the PG. Therefore, in **Chapter 7** we built 40 different PG agents (testing different PG configurations) in order to compare them against the RL agents. To do it, we proposed 3 metrics, TL, STD and NS.

After comparing the agents' results, in **Section 7.2**, we were surprised by the good results the new agents achieved. Indeed, there were PGs that managed to outperform the original ones since there were graphs that achieved TL rates greater than 100%. These good results mean that we have managed to build a policy that can successfully play Overcooked based only on the answers from the explainability method. This shows that the answers provided by the PG are more than enough to understand the logic behind the agents' decisions.

To sum up, the main contribution of this work is to have managed to implement, extend and validate the chosen explainability method in order to give explanations in a multi-agent cooperative environment.

8.2 Reviewing the objectives

In this section, we are going to review the objectives proposed in **Section 1.3.1**.

- **[O.1] Study state-of-the-art**: At the beginning of the thesis, we first studied the most basic concepts such as RL, XAI and MDP (**Chapter 2**). Once these concepts were learnt, then we did research on the current XRL methods (**Chapter 3**). Although most of this research was done at the very beginning, we have been reading and learning about the topic throughout all the thesis.

- **[O.2] Choose XRL method**: Then, we chose an explainability method in order to continue the line of research opened in [6, 7], which consists in producing explanations from predicate-based PG generated from the observation of RL-trained agents in the Cartpole environment.

- **[O.3] Replicate the chosen paper**: After choosing the XRL method, we replicated the method and extended it in order to test it in Overcooked. We implemented the algorithms needed and we were able to build a PG from a RL-trained agent. (**Chapter 5**). Moreover, we managed to give explanations that according to [6] should be validated by human experts with domain-specific knowledge.

- **[O.4] Validate the method**: At this point, we introduced a way to automatically validate whether the PG explanations could be extrapolated to the original RL agent (**Chapter 6**). To do so, we decided to build another agent that followed a policy based on the PG.

- **[O.5] Propose extensions and implement them**: After validating the method, we went one step further and proposed some modifications in order to improve its performance (**Chapter 5**). For this reason, we proposed two different PG algorithms (deterministic and stochastic) in addition to the different discretisers.

8.3 Reviewing the technical competences

Now, we will revise the technical competencies related to this project, and whether they have been fulfilled.

- **CCO1.1**: The project required modifying and extending the XRL method we chose since it was only tested in single-agent environments. We proposed different algorithm modifications taking into account the cooperative nature of the environment. Also, we proposed a method for automatically validating that the PG explanations could be extrapolated to the original RL-trained agents. To do it, we built a new agent with a policy based on the PG.

- **CCO1.3**: It was also required to analyse different MARL environments and platforms. Firstly, we decided to use Overcooked-AI due to is a game that requires cooperation. Besides, we decided to use the PantheonRL package due to its facility to control the execution of the program.

- **CCO2.2**: This was the base of this work. We had to extract the appropriate information from a RL model, process it and represent it into a compact form using predicates. Namely, we have been capable of discretizing each of the agent's observations, to later build a PG that represents the agent's behaviour. Once we built the PG, we implemented the algorithms for creating a new agent using this PG as a policy.

- **CCO2.4**: Even though finally, we decided to not implement our RL algorithm, we had to different RL agents using a PPO algorithm implemented in the Overcooked-AI package. On the other hand, we had to implement an algorithm that was capable of recording all the agent's interactions with the environment in order to build the PG. To do it, we had to manage a large data volume.

- **CCO3.1**: Related to CCO2.4, it was crucial to implement an efficient code due to the large data volume. If we had not paid attention to efficiency, it would have been impossible to run all the desired experiments.

8.4 Contributions to my career

Throughout the thesis, I had to search for a lot of information since it XAI is a relatively very new field. However, thanks to the Degree in Computer Engineering, there were a lot of concepts that were not new to me. For instance, in the Machine Learning subject, I had the opportunity to take my first steps in NNs and to know how they were implemented. This knowledge has been crucial for understanding faster the basics of Reinforcement Learning, especially in the field of Deep RL.

On the other hand, the subject of Distributed Intelligent Systems (SID) introduced me to multi-agent environments and their complexities. Moreover, SID helped me to understand better cooperation in multi-agent environments, which helped me to study the possible state discretisations of the environment.

This work allowed me to explore new fields in the area of AI. For example, I have been able to study more in-depth Reinforcement Learning and some of its techniques. Besides, I have been able to learn about XAI which allowed me to be more aware of the current limitations of AI.

Eventually, I think one of the most interesting things I have learnt during the project has been to plan and implement a project of this magnitude.

8.5 Future work

However, there is a lot of work to do. For this reason, we want to propose possible ways to expand this thesis.

- According to [6], the user can ask 3 types of questions. In this thesis, we focus on answering only one of these. Therefore, a possible extension could be to try to test the other types using the same technique used in this project.
- Of course, another possibility is to try different state discretisations, with different predicates and combinations. The better the discretisation of the agent's observation, the better answers we will obtain from the model. A possible modification could be modifying the *soup_pos* predicate so that only takes into account finished soups.
- According to **Section 5.4.4**, another interesting extension of the work can be to analyse and study in more depth those explanations that require more attention. This line of research could study each player's strategy from the explanations given by the PG.
- Another possibility is to propose different PG algorithms as well as test other similarity functions between states in order to see if they can achieve better results.
- In addition, it would also be interesting to test this method with other games.
- On the other hand, this project opens the doors to possible publications. In fact, this work is currently under review at the International Conference of the Catalan Association for Artificial Intelligence (CCIA).

Bibliography

[1] B. O. Li *et al.*, "Trustworthy AI: From Principles to Practices," Oct. 2021. DOI: 10.48550/arxiv.2110.01167. arXiv: 2110.01167. [Online]. Available: https://arxiv.org/abs/2110.01167v2.

[2] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, and D. Pedreschi, "A Survey Of Methods For Explaining Black Box Models," *ACM Computing Surveys*, vol. 51, no. 5, Feb. 2018, ISSN: 15577341. DOI: 10.48550/arxiv.1802.01933. arXiv: 1802.01933. [Online]. Available: https://arxiv.org/abs/1802.01933v3.

[3] D. Omeiza, H. Webb, M. Jirotka, and L. Kunze, "Explanations in Autonomous Driving: A Survey," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–21, Mar. 2021. DOI: 10.1109/TITS.2021.3122865. arXiv: 2103.05154v4. [Online]. Available: http://arxiv.org/abs/2103.05154%20http://dx.doi.org/10.1109/TITS.2021.3122865.

[4] L. Longo, R. Goebel, F. Lecue, P. Kieseberg, and A. Holzinger, "Explainable Artificial Intelligence: Concepts, Applications, Research Challenges and Visions," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 12279 LNCS, pp. 1–16, 2020, ISSN: 16113349. DOI: 10.1007/978-3-030-57321-8_1. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-57321-8_1.

[5] B. Goodman and S. Flaxman, "European Union regulations on algorithmic decision-making and a "right to explanation"," *AI Magazine*, vol. 38, no. 3, pp. 50–57, Jun. 2016, ISSN: 07384602. DOI: 10.1609/aimag.v38i3.2741. arXiv: 1606.08813. [Online]. Available: https://arxiv.org/abs/1606.08813v3.

[6] B. Hayes and J. A. Shah, "Improving Robot Controller Transparency Through Autonomous Policy Explanation," in *ACM/IEEE International Conference on Human-Robot Interaction*, vol. Part F1271, 2017, pp. 303–312, ISBN: 9781450343367. DOI: 10.1145/2909824.3020233.

[7] A. Climent, D. Gnatyshak, and S. Alvarez-Napagao, "Applying and Verifying an Explainability Method Based on Policy Graphs in the Context of Reinforcement Learning," *Frontiers in Artificial Intelligence and Applications*, vol. 339, pp. 455–464, Oct. 2021, ISSN: 09226389. DOI: 10.3233/FAIA210166. [Online]. Available: https://ebooks.iospress.nl/doi/10.3233/FAIA210166.

[8] A. Dafoe *et al.*, "Open Problems in Cooperative AI," pp. 2020–2032, Dec. 2020. DOI: 10.48550/arxiv.2012.08630. arXiv: 2012.08630. [Online]. Available: https://arxiv.org/abs/2012.08630v1.

[9] A. Barredo Arrieta *et al.*, "Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI," *Information Fusion*, vol. 58, pp. 82–115, Oct. 2020, ISSN: 15662535. DOI: 10.1016/j.inffus.2019.12.012. arXiv: 1910.10045. [Online]. Available: https://arxiv.org/abs/1910.10045v2.

[10] B. Baker *et al.*, "Emergent Tool Use From Multi-Agent Autocurricula," Sep. 2019. DOI: 10.48550/arxiv.1909.07528. arXiv: 1909.07528. [Online]. Available: https://arxiv.org/abs/1909.07528v2.

[11] M. Lapan, "Deep Reinforcement Learning Hands-On. Apply modern RL methods, with deep Q-networks, value iteration, policy gradients, TRPO, AlphaGo zero and more," *Packt Publishing Ltd.*, p. 547, 2018.

[12] A. Heuillet, F. Couthouis, and N. Díaz-Rodríguez, "Explainability in deep reinforcement learning," *Knowledge-Based Systems*, vol. 214, Aug. 2021, ISSN: 09507051. DOI: 10.1016/j.knosys.2020.106685. arXiv: 2008.06693. [Online]. Available: https://arxiv.org/abs/2008.06693v4.

[13] A. Krajna, M. Brcic, T. Lipic, and J. Doncevic, "Explainability in reinforcement learning: perspective and position," *JOURNAL OF XXXX FILES*, vol. 18, no. 9, Mar. 2022. DOI: 10.48550/arxiv.2203.11547. arXiv: 2203.11547. [Online]. Available: https://arxiv.org/abs/2203.11547v1.

[14] Y. Efthymiadis, K. Lenaerts, and T. Nowe, "Distilling Deep Reinforcement Learning Policies in Soft Decision Trees," pp. 1–6, 2019.

[15] Z. Juozapaitis, A. Koul, A. Fern, M. Erwig, and F. Doshi-Velez, "Explainable Reinforcement Learning via Reward Decomposition," 2019.

[16] M. T. Ribeiro, S. Singh, and C. Guestrin, ""Why Should I Trust You?" Explaining the Predictions of Any Classifier," *NAACL-HLT 2016 - 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Proceedings of the Demonstrations Session*, pp. 97–101, Feb. 2016. DOI: 10.18653/v1/n16-3020. arXiv: 1602.04938. [Online]. Available: https://arxiv.org/abs/1602.04938v3.

[17] S. M. Lundberg and S. I. Lee, "A Unified Approach to Interpreting Model Predictions," *Advances in Neural Information Processing Systems*, vol. 2017-Decem, pp. 4766–4775, May 2017, ISSN: 10495258. DOI: 10.48550/arxiv.1705.07874. arXiv: 1705.07874. [Online]. Available: https://arxiv.org/abs/1705.07874v2.

[18] S. Greydanus, A. Koul, J. Dodge, and A. Fern, "Visualizing and Understanding Atari Agents," in *Proceedings of the 35th International Conference on Machine Learning*, J. Dy and A. Krause, Eds., ser. Proceedings of Machine Learning Research, vol. 80, PMLR, 2018, pp. 1792–1801. [Online]. Available: https://proceedings.mlr.press/v80/greydanus18a.html.

[19] S. A. Sloman, "Causal Models: How People Think about the World and Its Alternatives," 2005.

[20] J. Y. Halpern and J. Pearl, "Causes and Explanations: A Structural-Model Approach — Part 1: Causes," Jan. 2013. DOI: 10.48550/arxiv.1301.2275. arXiv: 1301.2275. [Online]. Available: https://arxiv.org/abs/1301.2275v1.

[21] P. Madumal, T. Miller, L. Sonenberg, and F. Vetere, "Explainable reinforcement learning through a causal lens," *AAAI 2020 - 34th AAAI Conference on Artificial Intelligence*, pp. 2493–2500, May 2020, ISSN: 2159-5399. DOI: 10.1609/aaai.v34i03.5631. arXiv: 1905.10958. [Online]. Available: www.aaai.org.

[22] T. D. Kulkarni, K. R. Narasimhan, A. Saeedi, and J. B. Tenenbaum, "Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation," *Advances in Neural Information Processing Systems*, pp. 3682–3690, Apr. 2016, ISSN: 10495258. DOI: 10.48550/arxiv.1604.06057. arXiv: 1604.06057. [Online]. Available: https://arxiv.org/abs/1604.06057v2.

[23] T. Shu, C. Xiong, and R. Socher, "Hierarchical and Interpretable Skill Acquisition in Multi-task Reinforcement Learning," *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, Dec. 2017. DOI: 10.48550/arxiv.1712.07294. arXiv: 1712.07294. [Online]. Available: https://arxiv.org/abs/1712.07294v1.

[24] V. Zambaldi *et al.*, "Relational Deep Reinforcement Learning," Jun. 2018. DOI: 10.48550/arxiv.1806.01830. arXiv: 1806.01830. [Online]. Available: https://arxiv.org/abs/1806.01830v2.

[25] B. Sarkar, A. Talati, A. Shih, and D. Sadigh, "PantheonRL: A MARL Library for Dynamic Training Interactions," in *Proceedings of the 36th AAAI Conference on Artificial Intelligence (Demo Track)*, 2021. arXiv: 2112.07013. [Online]. Available: http://arxiv.org/abs/2112.07013.

[26] M. Carroll *et al.*, "On the utility of learning about humans for Human-AI coordination," *Advances in Neural Information Processing Systems*, vol. 32, Oct. 2019, ISSN: 10495258. DOI: 10.48550/arxiv.1910.05789. arXiv: 1910.05789. [Online]. Available: https://arxiv.org/abs/1910.05789v2.

[27] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. K. Openai, "Proximal Policy Optimization Algorithms," Jul. 2017. DOI: 10.48550/arxiv.1707.06347. arXiv: 1707.06347. [Online]. Available: https://arxiv.org/abs/1707.06347v2.

[28] PayScale, *Project manager salary*, 2018. [Online]. Available: https://www.payscale.com/research/ES/Job=Project_Manager%2C_Information_Technology_(IT)/Salary.

[29] PayScale, *Research Scientist Salary*, 2021. [Online]. Available: https://www.payscale.com/research/US/Job=Research_Scientist/Salary.

[30] PayScale, *Machine Learning Engineer salary*. [Online]. Available: https://www.salary.com/tools/salary-calculator/machine-learning-engineer-hourly.

[31] PayScale, *Software Tester Salary*, 2018. [Online]. Available: https://www.payscale.com/research/US/Job=Software_Tester/Salary.

[32] PayScale, *Software Analyst Salary*. [Online]. Available: https://www.payscale.com/research/US/Job=Software_Analyst/Salary.

Appendix A

Project Planning

In any project, it is very important to have a plan to be able to achieve the objectives. The project began on September 13th and ends on June 13th to be able to present it on June 27th (287 days). Taking into account that on average I will work 4 hours a day on the project this adds up to a total of 1.148 hours. Having this information, in this section we will focus on the tasks, their distribution throughout these 1.148 hours (287 days) and the risk management.

A.1 Task definition

In this section, we described and organised the different tasks that were carried out taking into account the possible dependencies that they could have with other tasks. We ordered them in chronological order.

A.1.1 Project management (PM)

This task consisted in planning a good organisation for the project.This task only required a computer. This task was the first one we started on September 13, together with task SS.

- **[PM.1] Contextualisation and project scope**: Definition of the project scope in the context of its study. This task consisted in present the objectives of the project, the relevance of the area and how to do it.
- **[PM.2] Planning**: Plan the tasks to be carried out, describing the different phases of the project, resources and the requirements needed.
- **[PM.3] Economic management**: Analyse the economic dimension of the project.
- **[PM.4] Sustainability report**: Do a sustainability analysis of the project.
- **[PM.5] Integration in final document**: Once the previous tasks and the conclusions of the work were completed, all the documentation had to be gathered in the same document, correcting possible errors.
- **[PM.6] Meetings**: Meetings were scheduled as needed to check that the work was getting on properly. Preferably the meetings were held in person at the university, but sometimes the situation forced us to use Google Meet.

A.1.2 Study state-of-the-art (SS)

Considering that one of the main purposes of the thesis is to research the state-of-the-art XRL methods in a more complex cooperation environment, it was fundamental to have a solid theoretical part to ensure the consistency of the experiments. To do this, this task consisted in study the theoretical bases to understand Reinforcement Learning and the XRL methods. This task started on September 13th and it overlapped with the tasks related to PM. It required reading papers, books, internet, etc.

- **[SS.1] Study RL**: Study what is RL and what are the actual techniques (Q-Learning, Deep Q-Learning, etc.)
- **[SS.2] Study XRL**: Study what is XRL and what are the actual techniques. Specifically, study more in-depth Policy Graphs and Markov Decision Processes.

A.1.3 Getting RL agents (GRLA)

In order to explain the behaviour of an RL agent, we first had to train one. In our case, we decided to train an agent to learn how to play Overcooked, a cooperative multi-agent game. It required reading papers, books and internet in addition to a computer for training the RL agents. This task was divided into the following sub-tasks.

- **[GRLA.1] Train RL agents**: Train multiple RL agents with multiple parameters in order to take the best one. To do it, we used PantheonRL library [25] for training the agents in Overcooked-AI environment.
- **[GRLA.2] Test RL Agents**: Test RL agents to choose the best ones and then be able to use them to explain their behaviour.

A.1.4 Building PG (BPG)

Once the GRLA task was completed, we had to implement the algorithms needed in order to build the RL. In this case, it was required to read papers, books and internet in addition to a computer. for implementing these algorithms. This task can be divided into the following sub-tasks.

- **[BPG.1] Develop Discretisers**: Develop multiple discretisers that define different state representations in order to compare which one fits better to our scenario.
- **[BPG.2] Develop PG algorithms**: Develop multiple PG algorithms that take a discretiser as a parameter and build a PG based on the selected state representation running multiple times the original agent.

A.1.5 Validating PG (VPG)

This task consisted in answering the question: how do we guarantee that the answers given by the PG are based on the behaviour of the RL agents? To do so, we decided to build another agent that follows a policy based on the PGs obtained in task BPG. Both BPG and VPG tasks overlapped for a period of time because it was possible to build new PGs while the previous were

being tested. This task required reading books, papers and internet in addition to a computer to build and test the agents.

- **[VPG.1] Build PG agents**: Using the Policy Graphs built in former tasks, create a new agent that follows a policy based on its PG.
- **[VPG.2] Test PG agents**: Test if the PG agents perform similar to the original ones.

A.1.6 Explaining Overcooked scenario (EOS)

Once the first PG was built, it was possible to start developing the explainability algorithms. To do so, we decided to answer the 3 questions proposed on [6]. In order to carry out the task, it was necessary to read papers and internet and a computer to develop the algorithms. In parallel to this task, it was possible to continue improving the implementation of the previous task BPG.

A.1.7 Analysis and conclusions (AC)

One of the most important tasks of the thesis was to provide a good analysis and a coherent conclusion of the results. Once all the PG agents were built and validated, it was possible to start providing the thesis results and conclusions. Moreover, we revisited the initial work objectives seen in **Section 1.3.1**. This task only required a computer with Overleaf and Numbers app in order to plot charts.

This task only requires a computer. It can be divided into the following sub-tasks.

- **[AC.1] Compare both agents:** Pair each PG to its corresponding RL agent and run both in the same environment (separately) over and over again to see if their behaviour is similar.
- **[AC.2] Conclusions:** Evaluate whether the PG agents manage to achieve similar results to the original RL agents and test if the explanations given from the explainability method make sense from a logical point of view.

A.1.8 Documentation (D)

All the work done has been reported in a final document throughout the project in order to avoid haste. To do this task was only needed Overleaf, a Latex editor. This task was carried out throughout the project, for this reason, it has no dependencies and it re

A.1.9 Presentation (P)

At the end of the project, an oral defence was carried out. For this reason, this task consisted in considering which were the parts that require more attention in order to explain them. This task required a computer and Keynote app and it was indispensable to have finished task D.

A.2 Summary of the tasks

Below, we can see a summary of the tasks with its approximate needed time to carry it out.

Id.	Task	Time (h)	Dependencies
PM	**Project Management**	**161**	
PM.1	- Contextualization and project scope	25	
PM.2	- Planning	20	PM.1
PM.3	- Economic management	20	PM.2
PM.4	- Sustainability report	20	PM.3
PM.5	- Integration in final document	56	AC.2
PM.6	- Meetings	20	
SS	**Study state-of-the-art**	**168**	
SS.1	- Study RL	56	
SS.2	- Study XRL	112	SS.1
GRLA	**Getting RL Agents**	**60**	
GRLA.1	- Train RL agents	56	
GRLA.2	- Test RL agents	8	
BPG	**Building PG**	**484**	**GRLA**
BPG.1	- Develop Discretizers	200	
BPG.2	- Develop PG algorithms	284	
VPG	**Validating PG**	**88**	
VPG.1	- Build PG agents	78	BPG.2
VPG.2	- Test PG agents	10	
EOS	**Explaining Overcooked scenario**	**50**	**BPG.2**
AC	**Analysis and Conclusions**	**16**	**VPG**
AC.1	- Compare results	12	
AC.2	- Conclusions	4	
D	**Documentation**	**30**	
P	**Presentation**	**36**	**D**

Table A.1: Summary of the tasks. [Own Creation]

A.3 Risk management

In this section, we will talk about the different risks to which we submit carrying out this project, how they could affect us and how we could solve them.

A.3.1 Deadline of the project

Perhaps the most obvious risk is the project deadline. The project has a lot of tasks and sub-tasks and it is very likely that some of them require more or less time as initially expected. For this reason, it is necessary to make a plan to know how to react to these events.

- **Impact**: Medium
- **Proposed solution**: Recalculate the timings of each task. With these modifications, the planning will have to be updated. Once we updated the planning, if the problem persists, it will be necessary to dedicate more hours each day to meet deadlines.

A.3.2 Inexperience in the field

Reinforcement Learning and explainability methods are a totally new field for me. For this reason, it is likely that I would have to spend more time than I expected on this task.

- **Impact**: Medium
- **Proposed solution**: If I dedicate more time than I had planned, I will have to recalculate the timings of the rest of the tasks to meet the deadline. In case it were impossible to meet the deadline, it will be needed to dedicate more hours each day to be able to meet the deadline. On the other hand, In case it were necessary to reduce the number of hours of any task, it would be the BPG task.

A.3.3 Computational power

Reinforcement learning is a computationally expensive field, especially in complex simulation environments. In our case, it is likely that we do not have this problem since our environment is in 2D and it is not very expensive computationally. However, there is a risk that the training might be too expensive.

- **Impact**: High
- **Proposed solution**: The solution would be to explore alternatives for cloud computing, such as the use of Google Collaborative. In case the problem got worse and it were not possible to solve it through cloud computing, we would reduce the number of hours spend on BPG tasks. In that case, it would be necessary to recalculate the times of all the tasks and update the planning.

A.3.4 Covid-19

In case that due to Covid-19 we had to confine ourselves all the work would be done online.

- **Impact**: Low
- **Proposed solution**: All scheduled meetings with the tutors would be using Google Meet. The rest of the tasks would not be affected.

A.4 Gantt diagram

In this section, we show how we are going to carry out all the tasks mentioned above, how they will be distributed over time and the dependencies between them. The thesis begins on September 13th and ends on June 27th. The meetings with tutors will be scheduled in function of our necessities.

In **Figure A.1**, we can see the Gantt diagram.

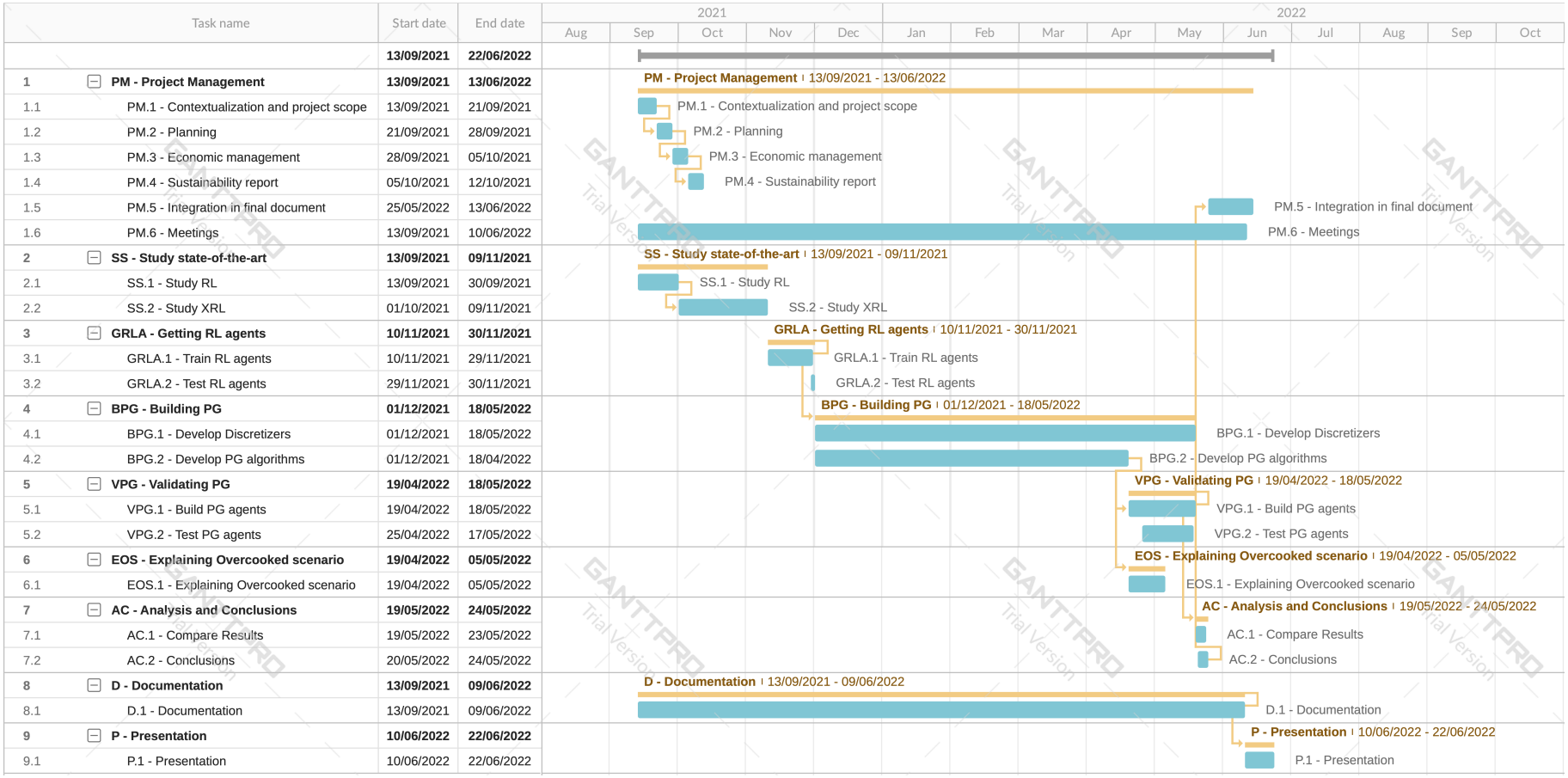


Figure A.1: Gantt Diagram [Own Creation]

Appendix B

Budget

In this section, we are going to talk about the economic cost of the project. To do so, we will describe the personal cost, generic costs and other costs in addition to the mechanisms for controlling potential budget deviations, including the numerical indicators.

B.1 Staff costs

In this section we are going to compute the total cost of each task defined in **Section A.2**. To calculate the cost of each task, we first need to know the cost per hour of each worker involved. Since we know the cost per hour of each worker, we only have to multiply his cost by the total hours of his tasks.

Despite all the roles have been performed by me and my tutors, we prefer to divide the project into roles to understand better the cost of it.

In this project, there are 5 types of staff. To begin with, the **Project Manager** is responsible for the planning and correct development of the project. In this case, the GEP tutor, the thesis supervisors and I, have played this role. In addition to this role, we have **Junior Researcher** that has studied the RL techniques to train agents aside from having studied XRL methods. Together with **Junior Research** we also needed a **Junior Developer** and a **Tester** that implemented RL, XRL and checked that everything worked well. Finally, we need a **Analyst** who could write the conclusions of our work.

In this thesis the Project Management role involves the tutors and me, the rest of the roles are going to be played by myself. Therefore, we show in the following tables the total cost of the thesis.

Firstly, as we can see in **Table B.1**, we need to know the average salary of our staff.

Role	**Cost(€)/h**
Project Manager	35[28]
Junior Researcher	21[29]
Junior Developer	20[30]
Tester	10[31]
Analyst	19[32]

Table B.1: Cost per hour of the different roles. [Own Creation]

Once we know the salary (cost per hour) of each role, we could know how much money would we need to carry out the project (**Table B.2**). Moreover, we also show in **Table B.3** the cost of each task.

Task	Hours	Project Manager	Junior Researcher	Junior Developer	Tester	Analyst
Project Management	161	161	0	0	0	0
Study-state-of-the-art	168	0	168	0	0	0
Getting RL agents	60	0	0	56	8	0
- Train RL Agents	56	0	0	56	0	0
- Test RL Agents	8	0	0	0	8	0
Building PG	484	0	0	484	0	0
Validating PG	88	0	0	78	10	0
- Build PG Agents	78	0	0	78	0	0
- Test PG Agents	10	0	0	0	10	0
Explaining Overcooked scenario	50	0	0	50	0	0
Analysis and Conclusions	16	0	0	0	0	16
Documentation	30	30	0	0	0	0
Presentation	36	36	0	0	0	0

Table B.2: Estimated time per task. [Own Creation]

Task	**Cost (€)**
Project Management	5.635
Study-state-of-the-art	3.528
Getting RL agents	1.200
- Train RL Agents	1.120
- Test RL Agents	80
Building PG	9.680
Validating PG	1.660
- Build PG Agents	1.560
- Test PG Agents	100
Explaining Overcooked scenario	1.000
Analysis and conclusions	304
Documentation	1.050
Presentation	1.260

Table B.3: Cost per task. [Own Creation]

Finally, we get that the global cost of the project is the sum of all the tasks, that is to say, **25.317€**. Additionally, in **Table B.4** we attach the cost of recruitment.

Role	**Cost(€)**	**Cost SS(€)**
Project Manager	7.945	9.820
Junior Researcher	3.528	4.361
Junior Developer	13.360	16.513
Tester	180	222
Analyst	304	376

Table B.4: Cost of recruitment taking into account Social Security. [Own Creation]

If we take into account the Social Charge, then the total cost of recruitment is **31.292€**

B.2 Generic Costs

In this section, we are going to see the general costs of the project such as the amortisations and the indirect costs.

B.2.1 Amortisation

Another aspect to take into account is the amortisation of material resources. In this project, I only take into account the hardware resources because the software is open source. I will work approximately 4 hours per day for 126 days, in a MacBook Pro. The formula to compute the amortisation is the following.

$$A = ResourcePrice \times \frac{1}{YearsOfUse} \times \frac{1}{DaysOfWork} \times \frac{1}{HoursPerDay} \times HoursUsed$$

Since we only need a laptop, if we apply the previous formula we get the following amortisation (**Table B.5**).

Hardware	Price (€)	Life Expectancy (Years)	Days of Work	Hours per day	Hours used	Amortization (€)
MacBook Pro	2.900	9	287	4	1.148	322,22

Table B.5: Amortization of the Hardware. [Own Creation]

B.3 Indirect costs

We can't forget that in a real project we also have indirect costs that have to be contemplated. Things like electricity and the internet are costs that we often forget. For this reason, in this section, we will talk about this type of indirect cost.

- **Internet cost:** The internet cost is around 100€ per month. Therefore the consumption is the following:

  (10 months) * (100€/month) * (4h/24h) = 166,6€

- **Electricity cost:** The electricity cost is around 0.201€/kWh per month. Additionally, we know that MacBook Pro uses 120kWh. Therefore the consumption is the following:

  (10 months) * (120 kWh) * (0.201€/kWh) * (4h/24h) = 40,02€

- **Travel cost:** I have to use public transport to meet with the tutor once every two weeks, to do so I have to use a T-Casual for only one zone. It costs 11,35€ per 10 journeys. Then the total cost is:

  (11,35€/10 journeys) * (40 journeys) = 454€

- **Work space cost:** This project will mainly be developed in my house, located in Barcelona. Actually, I could rent my house for 1350€/month. However, I will not use all the house, for this reason, we could divide the rent by 3 since it could be two more students living here. Therefore, my space in the house would cost 450€/month. Consequently, I get the following cost:

  (450€/month) * (10 months) = 4.500€

- **Water cost:** The water cost is around 35,3€ per person. Therefore the consumption is the following:

(10 months) * (35,3€/month) * (4h/24h) = 58,82€

B.4 The general cost of the project

To sum up, In **Table B.6** we show all the generic costs of the project:

Concept	Cost (€)
Amortization	322,22
Internet	166,6
Electricity	40,02
Travel	454
Work space	4.500
Water cost	58,82
Total	**5.541,72**

Table B.6: Generic costs of the project. [Own Creation]

B.5 Deviations in the budget

Once the total cost of the project is defined, it is very important to take into account possible deviations in the budget.

B.5.1 Contingency

During the development of the project, it can appear unexpected events, which take part of our budget. For this reason, it is always necessary to prepare a fund of contingency to be prepared to face these events. Since this research project tries to use very new technologies, the probability of unforeseen events is high. For this reason, we will add a 15% contingency margin.

B.5.2 Incidental costs

In **Section A.3** we defined all the possible risks that may appear. Now, we are going to quantify them in order to conclude the cost of applying alternative plans to the final budget (**Table B.7**).

Incident	Estimated cost (€)	Risk (%)	Cost (€)
Deadline of the project	2.100	20	420
Inexperience in the field	8.600	30	2.580
Computational power	60	20	12
Covid-19	0	50	0
Total	**10.760**		**3.012**

Table B.7: Incident cost. [Own Creation]

B.5.3 Final cost

Finally, we only have to put together all the previous sections. As a result, we show in the following table the total cost of the project (**Table B.8**).

Concept	Cost (€)
PCA	31.292
CG	5.541,72
Contingency margin	5.525,05
Incidental cost	3.012
Total	**45.370,77**

Table B.8: Total cost of the project. [Own Creation]

B.6 Management control

In this section, we will describe how we are going to model the potential budget deviation. We will calculate the individual deviation of each task from the estimated cost. To do so, we will use the following formulas:

- **Real Cost**: Means the real cost of the task. Consequently, we should recalculate the PCA, CG, contingency and incidents for each task to check any incidences. This process helps me to identify which part of the task has been miss-estimated in order to re-plan it.
- **Estimated cost**: Means the estimated cost of a task.
- **Deviation**: Means the difference between the real cost and the estimated cost of a task.

In conclusion, the deviation indicates how much the real cost varies from its estimation. In addition, we can see that if the deviation is negative, we must assign part of the contingency fund to the task in order to cover the deviation. However, if the deviation is positive, we may reallocate the extra money for other incidences.

Appendix C

Sustainability

In every project, it is important to carry out a sustainability analysis taking into account three dimensions: economic, environmental and social. For this reason, I will perform a self-assessment on the domain of sustainability competence, and then, based on some questions, the three dimensions are analysed within the framework of the project.

C.1 Self-assessment

During my career, I have heard many times the concept of sustainability. It is also true that despite having heard it a lot, I had never had to think of a project in key sustainability. As I have been answering the form, I have realised things that until now did not have in mind. An example of this is to know that there was an economic dimension. It is true that if I stop to think about it, it makes a lot of sense that there is an economic dimension, and also very important.

Answer the form has allowed me to reflect on the causes, consequences and solutions to current social, economic and environmental problems. Evaluating the sustainability of this work allows students to remind them that we live in a world where not any project is worth it. The simple fact of responding to this form obliges students to reflect on their own projects and gives them an opportunity to improve them in this regard.

Honestly, I had no knowledge of the existence of so many indicators to evaluate the sustainability of the project. I had to refresh some concepts such as amortisation to understand the impact of work in all areas. When I think of sustainability, the first thing that comes to my head is social impact, such as equality, the environmental footprint ... and now after the survey, I also think of the economic area, a fundamental area to verify our project is economically viable.

C.2 Economic dimension

C.2.1 Regarding the Project put into production (PPP): Reflection on the cost you estimated for the realisation of the project?

In this project we have seen the economic impact of this project (see **Appendix B**). There we can find all the information about the human and material costs and the potential deviations that may appear during the project.

The cost of the staff has been obtained by searching through the Internet. In addition, I have also measured other costs such as contingencies, incidental costs and their effects on the budget. In my opinion, I think the budget is reasonable and could be carried out in a real project. It is also true that I could try to use a co-working space in order to try to reduce the necessary workspace.

C.2.2 Regarding the Useful Life: How are the cost aspects of the problem you want to address currently being addressed (state of the art)?

Currently, the XRL field is a field that is still developing. Companies that usually have some type of RL agent to do tasks, work with probabilities, that is, they buy robots that do their task correctly on a very high percentage of times. That remaining percentage of times where the robot does not correctly perform your task is usually converted into a loss of money for the company.

C.2.3 How will your solution improve economically (costs ...) compared to the existing ones?

What this project poses is to try to transfer the knowledge of an RL agent (unintelligible for humans) to a Rule-Based Agent where a human can understand how the agent is making decisions. Any contribution that I can do in this field, contributes to the companies that use this type of technology to not depend on an agent that makes inexplicable decisions but that all their behaviour can be explained. This advance allows us to understand under what conditions an agent takes certain decisions, which allows us to modify those behaviours at our whim. Being able to understand how you think allows you to foresee the behaviour of our agents to avoid losing money.

C.3 Social dimension

C.3.1 Regarding the Project Put into Production (PPP): What do you think the realization of this project will bring to you on a personal level?

First, this project will allow me to enter the world of Machine Learning and will allow me to apply many of the knowledge acquired in the career. Second, this project will help me to introduce myself in the world of research and know what are the steps necessary to achieve research of this type, being the main responsible for a large project will help me organize myself better and plan the projects correctly.

C.3.2 Regarding Useful Life: How do you currently solve the problem you want to address (state of the art)? i. How will yours improve socially (quality of life) solution with respect to existing ones?

Currently the more complex the model is, the more complicated it becomes to try to explain it. For this reason, when the model is sufficiently complex we simply accept that we cannot understand it.

The main objective of this project is that a person can understand why an agent is making certain decisions. Understanding how these models think will help developers improve them and help people trust these types of technologies.

C.3.3 Regarding Useful Life: Is there a real need for the project?

From my point of view, I think it is very necessary. AI is a field that has been growing a lot in the last few years and very fast. Every time we generate more complex models that are capable of doing more and more complex things, but at the same time I think that we understand less and less how this type of technology works. Also, as I mentioned in the previous question, humans, as with other people, need to understand how these types of algorithms 'think' in order to trust them.

C.4 Environmental dimension

C.4.1 Regarding the Project put into production (PPP): Have you estimated the environmental impact that the realisation of the project will have?

I have not estimated the environmental impact of the project. Nevertheless, this project does not waste material but rather has a high electricity consumption for the experiments. The training of our agent will be the task that will consume the most electricity. Hence, there will be a high waste of electricity.

C.4.2 Regarding the Project in Production (PPP): Have you considered minimising its impact, for example, by reusing resources?

In this project, the only part I can reuse would be to find a trained agent who could play Overcooked. With the agent already trained I should only focus on the part of applying explainability methods.

C.4.3 Regarding the Useful Life: How do you currently solve the problem you want to address (state of the art) ?, i. How will your solution improve environmentally with respect to the existing ones?

Currently there is no way to know which methodology is the best to explain the behaviour of an agent. For this, what is done right now is to test different methodologies and see which one best suits the problem.

Right now, the solution proposed by this project continues through investigating various explanatory methods to see which one best suits the problem I want to solve (in this case, playing Overcooked).