A framework for multi-agent algorithmic recourse in normal form games

Kevin O'Brien and Patrick Mannion

July 2023

Abstract Complete abstract at the end

1 Introduction - edit

As deployments of systems using artificial intelligence (AI) become more commonplace in the real world, there has been a corresponding growth in interest in the field of explainable artificial intelligence (XAI). AI systems are now used for a variety of roles previously performed by humans. From simple applications like recommendation systems that recommend content to watch or products to consider buying, to more critical systems like health diagnosis systems or transport navigation systems, the role of AI has increased significantly in recent times with many impacts, both good and bad [DR20].

Many prominent AI systems make use of machine learning (ML), a process which allows a system to improve its performance at a task with experience. For example, a training dataset with examples of past decisions by humans could be used to train a ML model. The output of the training process is a ML model/hypothesis that generalises over the examples in the dataset; this trained model could then be used to make future decisions automatically [Mah20].

While AI systems are now being more commonly adopted for decision making applications, there is still a lot of concern among policymakers and the general public about using these systems and the impact they are having or will have on human lives PM: Find a nice example reference for this one. KOB: For example there are concerns relating to the role of AI in healthcare and with the impact AI can have on academic integrity, and some work is underway to address these concerns [KA22; Uzu23].

For some AI applications, the consequences of a poor decision by an AI system are not very severe. For example, when recommending a TV show to watch, if the recommendation given by the system is not interesting for the user, the only negative impacts for the user are more time spent finding something to watch, and perhaps an increasing level of dissatisfaction with the service if the recommendations are consistently poor. The impacts of poor recommendations are likely to be felt more keenly by the provider of the service, as consistently poor recommendations will lead to users abandoning the service. In this case it is useful for the provider to understand why its systems got it wrong, a situation where XAI can play a role [Web+23].

For AI applications which have more serious potential impacts on human lives, XAI has a more important role to play, helping to foster trust in the AI system, and to ensure the system is acting in a fair way. For example, for a system which handles a health diagnosis and recommends a treatment, it is important decisions taken about the diagnosis and treatment are correct and can be trusted [MKR21]. Another example is a automated system used for assessing loan applications. Users and regulators will rightly want to know if the reasons for rejecting loan applications are loan fair and ethical, and that loans are not rejected based on sensitive attributes such as race or gender [Bar+20]. XAI has a role to play in developing such systems, enabling developers to use explanations to ensure the systems they are creating are fair, ethical, trustworthy and safe

[BP21; Rud19; YL22].

A popular sub-field of XAI is counterfactual explanations (CFE). For example, if a loan application is rejected, CFE will show examples of approved loan applications. This at least allows a user to compare their application to see what changes it would need to make. This is often taken a step further to inform the user what actions it needs to take to achieve a different outcome. Using the loan application example, the CFE may recommend the user needs to be earning more money. These types of recommendations by CFE are known as recourses. Both CFE and recourses are commonly used to refer to the same thing, and essentially they are similar, but with slightly different aims as already outlined here. As both aim to provide a user with information as to what is required for a different decision, just the term CFE will mainly be used for the remainder of this article [Sla+21].

It is important CFE provided are fair, trustworthy, and ethical. They provide a useful technique for developers to ensure systems do not recommend actions to take which are not robust. For example, informing a user they need to improve salary by a certain percentage, but also need to inform user the length of time with current employer must be a certain amount of years. Recommended actions need to be robust and also achievable. A system must not recommend something which is unfair or could cause offence. For example, change marital status to divorced, change gender, or change race [VDH20].

In summary CFE are used to determine the reasons for a decision and to provide a course of actions for achieving a different outcome. The course of actions can be in stages and the CFE can highlight the changes at different stages and show what actions will be made at the next stage for a chosen action. This helps provide a good understanding of the system [VDH20]. There is of course a trade off here, as how much should be revealed by the system, so as to not compromise Intellectual Capital [Bar+20].

PM: bring in multi-agent link here - real world decision making systems are rarely completely isolated, interactions with outer users/agent are very common etc.

KOB: Results from AI systems are often determined from actions involving more than one agent (users or or other systems). These kind of systems are referred to as Multi Agent systems. The role these other agents contribute, through their role and actions, to results produced by AI systems, need to be understood when generating a CFE for one agent [OK22].

XAI, or more specifically CFE, does have a role to play in the learning process to ensure better results, in terms of thrust, fairness, ethics, safety, privacy and also strategy. With Reinforcement Learning (RL), CFE has a role, to inform the RL process during the learning stage in order to develop a good strategy [Ngu+21].

RL is the process of learning through feedback a strategy for interacting with an environment. The feedback is represented by rewards and punishments. Learning is based on figuring out which actions lead to better rewards. Once this process is complete, the learner has a policy which it can use for interacting with its environment at any future point [RN16].

CFE could be used in this feedback process. Learner gets reward/punishment for an action, but can also get some direction as to what adjustment to an action should be made to achieve a better result. The main purpose of this article is to describe such a process, and provide implementation and experiments demonstrating this process.

As suggested by the title of this article, and mentioned in the first paragraph of this section, Normal Form Games (NFG), will be used to demonstrate using CFE as part of the RL process.

NFG are the environment of the learner in this article. For the purposes of this article the focus will be on the most basic examples of NFG. Examples of such NFG are the games prisoner dilemma, stag hunt, matching pennies, and rock, paper, scissors. These games involve two players. Each player can select one action from a list of two or more actions. When a player is choosing an action, it does not necessarily know what action the other player has chosen or will chose. Based on the actions chosen by both players, a reward or score is awarded to each player. The objective of the game can be two fold. What is the best action to choose to maximise an individual players payoff, or what action will amount to the best action for the group. In some cases the best action for an individual will also be the best action for the group. The chosen action is referred to as the strategy deployed [RN16; OK22].

Using RL, a player can learn the best strategy playing its opponent based on the objective. An RL agent can be used, to select an action, and receive feedback in the form of a signal, to indicate direction of change to the action to achieve a better result. The feedback can include details of the opponents own actions. For training purposes, the opponents actions can be known. The aim of this process is for an RL agent to learn the best strategy to improve its probability of meeting its objective. Then the RL agent settles on a probability, it can then store values for future references as part of its game playing policy.

This process involves multiple agents, so it is a multi agent environment [RN16]. One of the agents in training can be used as a teacher agent, another agent can be used as a pupil (the learner), and another agent used as the opponent. The pupil agent is the RL agent in this case.

In the subsequent sections, this process is discussed in more detail. Section 2 provides more background on XAI, NFG, RL, optimisation, and presents algorithms for optimisation for calculating strategies along with worked examples. Section 2 also includes a subsection on related work. Section 3 focuses on implementation of solution. Section 4 presents experiments carried out using the solution. Section 5 introduces other solutions which offer optimisation techniques and includes experiments using these solutions to confirm the validity of optimisation solution proposed in this article. Section 6 concludes the articles and recommends some future direction in this area.

2 Introduction

As the title suggests, this article looks at the role Explainable Artificial Intelligence (XAI) can play in Reinforcement Learning (RL). The article focuses on Normal Form Games (NFG) with two players and any number of actions, and how XAI can assist players to learn a good game playing strategy using RL.

XAI has grown in popularity in recent times and is now the focus of a lot of research in the area of Artificial Intelligence (AI). AI systems are now used for a variety of roles previously performed by humans. From simple applications like recommendations systems, what to watch on television or what products to consider buying, to more critical systems like health diagnosis systems or transport navigation systems, the role of AI has increased significantly in recent times with many impacts, both good and bad [DR20].

Decisions by such AI systems are made based on how the system was trained. Training these kind of systems is known as Machine Learning (ML). Using data which includes actual decisions, a hypothesis is generated, and this hypothesis is then used for making future decisions [Mah20].

While AI systems are now being more commonly adapted, there is still a lot of fear about using these systems and the impact they are having or will have on human lives. For applications like recommending what TV show to watch, there is not really any issue. There is no real cost involved if a recommending system suggest a TV show which turns out not to be interesting. The impact maybe more felt by the provider of the recommendation than the consumer. However it maybe useful for the provider to understand why its systems got it wrong. This is an example of where XAI can play a role [Web+23].

For systems which have a potential impact on human lives, XAI has a more important role to play. An explanation of decisions would help to foster thrust in the system, and ensure the system is acting in a fair way. For example, a system which handles a health diagnosis and recommends a treatment, it is important decisions taken about the diagnosis and treatment are correct and can be trusted. XAI can help with this [MKR21].

Another example is a system used for assessing loan applications. Are the reasons for rejecting the loan fair and ethical? Was the loan rejected based on race or gender for example [Bar+20]?

XAI has a role to play in developing such systems, to enable developers to use explanations to ensure the systems they are creating are fair, ethical, trustworthy and safe [BP21], [Rud19], [YL22].

A popular sub-field of XAI is Counter Factual Explanations (CFE). For example, if a loan application is rejected, CFE will show examples of approved loan applications. This at least allows a user to compare their application to see what changes it would need to make. This is often taken a step further to inform the user what actions it needs to take to achieve a different outcome. Using the loan application example, the CFE may recommend the user needs to be earning more money. These types of recommendations by CFE are known as recourses. Both CFE and Recourse's are commonly used to refer to the same thing, and essentially they are similar, but with slightly different aims as already outlined here. As both aim to provide a user with information as to what is required for a different decision, just the term CFE will mainly be used for the remainder of this article [Sla+21].

It is important CFE provided are fair, trustworthy, and ethical. They provide a useful technique for developers to ensure systems do not recommend actions to take which are not robust. For example, informing a user they need to improve salary by a certain percentage, but also need to inform user the length of time with current employer must be a certain amount of years. Recommended actions need to be robust and also achievable. A system must not recommend something which is unfair or could cause offence. For example, change marital status to divorced, change gender, or change race [VDH20].

In summary CFE are used to determine the reasons for a decision and to provide a course of actions for achieving a different outcome. The course of actions can be in stages and the CFE can highlight the changes at different stages and show what actions will be made at the next stage for a chosen action. This helps provide a good understanding of the system [VDH20]. There is of course a trade off here, as how much should be revealed by the system, so as to not compromise Intellectual Capital [Bar+20].

XAI, or more specifically CFE, does have a role to play in the learning process to ensure better results, in terms of thrust, fairness, ethics, safety, privacy and also strategy. With Reinforcement Learning (RL), CFE has a role, to inform the RL process during the learning stage in order to develop a good strategy [Ngu+21].

RL is the process of learning through feedback a strategy for interacting with an environment. The feedback is represented by rewards and punishments. Learning is based on figuring out which actions lead to better rewards. Once this process is complete, the learner has a policy which it can use for interacting with its environment at any future point [RN16].

CFE could be used in this feedback process. Learner gets reward/punishment for an action, but can also get some direction as to what adjustment to an action should be made to achieve a better result. The main purpose of this article is to describe such a process, and provide implementation and experiments demonstrating this process.

As suggested by the title of this article, and mentioned in the first paragraph of this section, Normal Form Games (NFG), will be used to demonstrate using

CFE as part of the RL process.

NFG are the environment of the learner in this article. For the purposes of this article the focus will be on the most basic examples of NFG. Examples of such NFG are the games prisoner dilemma, stag hunt, matching pennies, and rock, paper, scissors. These games involve two players. Each player can select one action from a list of two or more actions. When a player is choosing an action, it does not necessarily know what action the other player has chosen or will chose. Based on the actions chosen by both players, a reward or score is awarded to each player. The objective of the game can be two fold. What is the best action to choose to maximise an individual players payoff, or what action will amount to the best action for the group. In some cases the best action for an individual will also be the best action for the group. The chosen action is referred to as the strategy deployed [RN16; OK22].

Using RL, a player can learn the best strategy playing its opponent based on the objective. An RL agent can be used, to select an action, and receive feedback in the form of a signal, to indicate direction of change to the action to achieve a better result. The feedback can include details of the opponents own actions. For training purposes, the opponents actions can be known. The aim of this process is for an RL agent to learn the best strategy to improve its probability of meeting its objective. Then the RL agent settles on a probability, it can then store values for future references as part of its game playing policy.

This process involves multiple agents, so it is a multi agent environment [RN16]. One of the agents in training can be used as a teacher agent, another agent can be used as a pupil (the learner), and another agent used as the opponent. The pupil agent is the RL agent in this case.

In the subsequent sections, this process is discussed in more detail. Section 2 provides more background on XAI, NFG, RL, optimisation, and presents algorithms for optimisation for calculating strategies along with worked examples. Section 2 also includes a subsection on related work. Section 3 focuses on implementation of solution. Section 4 presents experiments carried out using the solution. Section 5 introduces other solutions which offer optimisation techniques and includes experiments using these solutions to confirm the validity of optimisation solution proposed in this article. Section 6 concludes the articles and recommends some future direction in this area.

3 Background

3.1 XAI

There are many different types of explanations and techniques available for deriving them in AI. In Machine Learning, a hypothesis is created which represents a model of data. The hypothesis is created using data which describes the model. Once the hypothesis is created from the training process, it is used to make predictions with unseen data [Oqu+12]. There are many different models available in Machine Learning, and which model to use largely depends on the task and/or developers experiences of certain types of models. Models are either opaque or transparent [BP21].

The architecture of transparent models is obvious to end users, maybe not immediately, but certainly after a period of familiarisation with the model. These models as a result of their transparency provide what is commonly known as global explanations [Dan+20]. As the model is transparent, it is relatively straightforward to come up with a global explanation from the model to explain any decision made. Such explanations are not specific to a particular decision, but provide enough information to at least partially explain a decision and give a user some thrust in the outcome. A user may not like the decision, but the global explanation at least provides some context to the decision.

Opaque models are more difficult to understand than transparent models. An example of an opaque model is a neural network. The architecture and data structure are hidden, so it is more difficult to understand how decisions were made. Such models are used in certain circumstances for performance reasons in the belief that opaque models provide more accurate decisions than transparent models. In some cases this is true. Another reason for using such models, is to protect propriety information. Often providers of systems which use opaque models are protecting their model from public knowledge so as to avoid competitors gaining any advantage on them.

There is an argument that all models should be transparent, as performance differences between such models is not significant enough to warrant using opaque models in cases where a transparent model would adequately suffice [Rud19].

As it is more difficult to understand opaque models, it is more difficult to provide explanations. Usually some extra work is required in order to derive explanations from these types of models. One way of doing it, is to build a partial global model which mimics some of the opaque model and is transparent [BP21]. This allows for deriving global explanations. Another way, is to create software, to extract explanations from the model. The explanations could be extracted at a global level, or extracted for the purpose of a single decision from the system. The explanations for a single decision are known as local explanations. These types of explanations are derived either while the decision is being generated by the model, or by running a query on the model after the decision is made to extract an explanation for that decision [Dan+20]. These explanations tend to be more accurate than the global explanations from transparent model, as global explanations tend to be approximated [Cyr+21].

To enhance explanations and make them more useful, some recommendations can be included. These recommendations are basically advice on what steps are required to achieve a different outcome from the model. These recommendations are referred to as Counter Factual Explanations (CFE). CFE are generated by comparing the actual result with an alternative result and providing an explanation as to what updates are required to achieve this alternative result [DVC23]. For example, update an action probability in a Normal Form Game to achieve a better outcome.

For this article, the focus is on local and accurate explanations which are produced along with the result as CFE. Reinforcement Learning will use these explanations to form a strategy for playing a Normal Form Game.

3.2 Normal Form Games

Normal Form Games (NFG) involve players, actions and a payoff function. The payoff function represents the utility for each player [Röp+22]. For this article the focus will be on two player games where a player can choose more than one action. For each player, the actions chosen will result in a payoff, but the payoff is calculated based on what actions the other player has chosen. The payoff function can be represented by a matrix which contains the payoff value for each player based on the actions they have chosen and the actions the other player has chosen. The actions are a numeric value which is referred to as an action probability. This type of representation of the payoff function is known as the normal form, thus the name of these types of games, NFG [RN16].

The idea behind these games is for a player to pick a suitable strategy to maximise their payoff. As there are other players involved, the strategy is based on action probabilities the other players will chose. Players can chose a fixed strategy, also referred to as pure strategy, or a mixed strategy. Based on the game and payoffs involved, which type of strategy to use for best results can be determined [Kre89].

A fixed strategy involves a player always choosing the same action. Other players can determine from this the best strategy for them. A mixed strategy involves players choosing actions based on a different approach [Kre89]. For example, player may chose action 1, 20 percent of the time, and action 2, 80 percent of the time. Playing against this kind of strategy will involve different calculations and could result in a fixed strategy in response to this or a mixed strategy.

When playing such games, the actions of the other players can be kept hidden until all players have chosen an action, in which case the action to choose is based on a number of factors which are determined by the payoffs available. From analysing the payoffs it can be determined which actions offer the most likely payoff. If one player is similar to another player, then both will likely chose the same strategy. Both players will analyze the payoff and determine the opponent will make the same assumptions for its strategy. In this case this is known as the dominant payoff [RN16].

Players may also communicate which each other when developing strategies. This helps to determine what actions other players should choose in response to maximise their payoff.

In NFG, equilibrium exists in the payoff function. This helps to develop strategies for players to determine the best action to choose. The mathematician John Nash proved that at least one equilibrium exists in each NFG [Kre89].

A classic example of a NFG is the prisoner dilemma [OK22]. This is a two player game, and each player can choose one of two actions - defect or cooperate. The payoff function is based on prison sentence reduction. If player 1 cooperates and player 2 defects, then player 1 gets no reduction in sentence, while player 2 reduces their sentence by 3 years. Likewise, if player 1 defects, and player 2 cooperates, then player 1 reduces their sentence by 3 years and player 2 gets no reductions in sentence. If both players cooperate, then both players get a 2 year reduction in sentence, while if both players defect, then they both get a one year reduction in sentence. This payoff function can be represented using the matrix presented in Table 1.

		Player 2	
		Cooperate	*Defect*
Player 1	*Cooperate*	$(2, 2)$	$(0, 3)$
	Defect	$(3, 0)$	$(1, 1)$

Table 1: Prisoner Dilemma Payoff Matrix.

From this payoff function, equilibrium is achieved if both players cooperate or both defect. If both defect, then the sum of payoffs is 2 (1 + 1), whereas if both corporate, the sum of payoffs is 4 (2 + 2). If both cooperate, then this is the best payoff for the group. If one defects and one cooperates, then the individual payoff for the player who defects is the maximum value available for an individual, but for the group, it is not, as the sum of payoffs in this scenario is 3 (0 + 3). This presents many options to the players.

Other examples of NFG, which are similar to Prisoner Dilemma, include stag hunt, matching pennies and rock, paper scissors. These games are used in the experiment presented in this article, and are described in more detail in section 4.

This article will use these games to demonstrate the role of XAI, or more specifically CFE, when an RL agent is learning a strategy to play these games. A CFE will be provided as feedback to the RL agent using a signal to indicate what adjustment it needs to make to achieve a better result.

To inform the RL agent using feedback by way of a CFE, an optimization technique is used to aid the generation of the CFE for this feedback.

3.3 Reinforcement Learning

Reinforcement Learning (RL) is a process of training using feedback to indicate if actions performed are correct or otherwise. Feedback usually indicates a level of confidence a learner can have in actions performed. Feedback is based on a reward structure. For example, a high reward indicates to the learner that the action performed is good. RL can be described as a trial and error approach to finding a good solution [PV20].

RL can be used with NFG by leveraging the payoff function, as the payoffs indicate result obtained and checks can be done to compare payoffs for all actions available to the learner. RL is ideal in a scenario like an NFG when playing against another player. RL can help a player devise a strategy in game playing. This strategy can then be used in the future to play these games. The strategy in RL is commonly referred to as a policy.

Using XAI with CFE, the RL process can be advised on what reward is achieved with the current action, and if necessary how the reward can be improved by adjusting the action taken. Advice on action adjustment is an example of a CFE.

3.4 Optimisation

As part of the experiment which this article is based on, an optimisation technique was created in order to help generate the CFE for the RL process.

Optimisation in the context of this article is a process to compute better solutions for an agent. With NFG there are constraints involved. Such constraints as number of players involved, number of actions, probability values of those actions, rewards, and in some cases equilibria. The solution is formulated based on these constraints and objectives associated with these constraints. An example of an objective is achieving the maximum payoff for a single player [OK22].

A worked example is presented in the next sub section, describing a process which derives a solution, and uses this solution to advise the player of result and if any adjustment is required to achieve a better outcome.

3.5 Worked Example

For the worked example, the NFG Prisoner Dilemma, which was described in section 2.2, will be used. This worked example will demonstrate and describe how results are generated, how an RL agent will assume the role of a player and learn the best strategy to create a policy.

3.5.1 Payoff Function Calculations

This section presents the formulas to calculate the payoff values for each action and the total payoffs.

Referring to Table 1, Prisoner Dilemma Payoff Matrix, there are two players - Player 1 and Player 2. There are two actions available - Cooperate and Defect. A binary choice. The payoff function includes the reward value available for each player, and the reward for one player depends on the choice of the other player.

Rewards are determined as follows:

- Player 1 Cooperates, Player 2 Cooperates - Player 1 gets reward of 2 (**P1X1**), while player 2 gets reward of 2 (**P2X1**).
- Player 1 Cooperates, Player 2 Defects - Player 1 gets reward of 0 (**P1X2**), while player 2 gets reward of 3 (**P2Y1**).
- Player 1 Defects, Player 2 Cooperates - Player 1 gets reward of 3 (**P1Y1**), while player 2 gets reward of 0 (**P2X2**).
- Player 1 Defects, Player 2 Defects - Player 1 gets reward of 1 (**P1Y2**), while player 2 gets reward of 1 (**P2Y2**).

This payoff function can be represented using the matrix presented in Table 2.

		Player 2	
		X	Y
Player 1	X	$(P1X1, P2X1)$	$(P1X2, P2Y1)$
	Y	$(P1Y1, P2X2)$	$(P1Y2, P2Y2)$

Table 2: Two Player, 2 Action NFG Payoff Matrix.

Note that this table is the same as Table 1, but just using different syntax to represent the values, and present a solution which can be reused for other NFG for 2 players with 2 actions.

As both players in what is described so far select either one action or the other action, it is possible to apply probabilities to the actions when playing an NFG. For example a probability of 30% on first action and probability of 70% on second action. The same calculations will apply with the payoff matrix to calculate payoffs for each player.

Using the notation from Table 2, what follows are the calculations for Player 1 and Player 2 based on values selected for actions X and Y. Note that the values for X and Y for each player must sum up to 1, so that:

Y = 1 - X

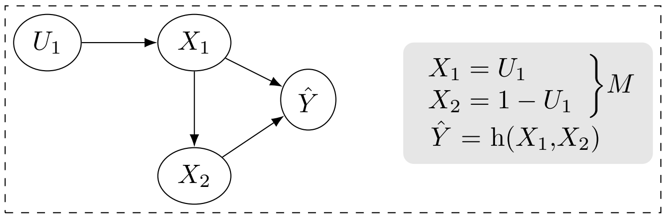

Figure 1: Causal Graph [KSV21; OK22] representing Prisoner Dilemma. U_1 represents a value from 0 to 1 chosen by a player for cooperate. M is the set of equations for the model. $\hat{Y}$ represents the function to calculate the total payoff for each player - see below - equations (1) to (4) for player 1, and (5) to (8) for player 2. Refer to table 2 for variables used in these equations.

Total Payoff for Player 1:

$$P1PX = P2X * P1X1 + P2Y * P1X2 \tag{1}$$

$$P1PY = P2X * P1Y1 + P2Y * P1Y2 \tag{2}$$

$$TPP1 = P1PX * P1X + P1PY * P1Y \tag{3}$$

$$\hat{Y} = TPP1 \tag{4}$$

Total Payoff for Player 2:

$$P2PX = P1X * P2X1 + P1Y * P2X2 \tag{5}$$

$$P2PY = P1X * P2Y1 + P1Y * P2Y2 \tag{6}$$

$$TPP2 = P2PX * P2X + P2PY * P2Y \tag{7}$$

$$\hat{Y} = TPP2 \tag{8}$$

Example:

Player 1, P1, sets value for X to 0, and Player 2, P2, sets value for X to 1.

P1: Y = 1 - X = 1 - 0 = 1

P2: Y = 1 - X = 1 - 1 = 0

P1(X, Y) = P1(0, 1)

P2(X, Y) = P2(1, 0)

Note: X represents X_1 and U_1, while Y represents X_2 in this example

P1 Payoff for X. Equation (1):

P1PX = P2X * P1X1 + P2Y * P1X2

1 * 2 + 0 * 0 = 2

P1 Payoff for Y. Equation (2):

P1PY = P2X * P1Y1 + P2Y * P1Y2

1 * 3 + 0 * 1 = 3

Total Payoff for P1. Equations (3) and (4):

TPP1 = P1PX * P1X + P1PY * P1Y = 2 * 0 + 3 * 1 = 3

P2 Payoff for X. Equation (5):

P2PX = P1X * P2X1 + P1Y * P2X2

0 * 2 + 1 * 0 = 0

P2 Payoff for Y. Equation (6):

P2PY = P1X * P2Y1 + P1Y * P2Y2

0 * 3 + 1 * 1 = 1

Total Payoff for P2. Equations (7) and (8):

TPP2 = P2PX * P2X + P2PY * P2Y = 0 * 1 + 1 * 0 = 0

In this example, the total payoff for player 1 (TPP1) is 3, while for player 2 the total payoff (TPP2) is 0, when player 1 sets action probability for X (P1X) to 0, and player 2 sets action probability for X (P2X) to 1. Player 1 achieves the best payoff in this example.

3.5.2 Optimisation

In the example just given, player one achieves the maximum reward, whereas the other player achieved the minimum reward.

An optimisation technique can be used to improve the other players reward. This technique can indicate what value it can use for X, in order to improve it rewards.

In the example given, Player 1 sets X to 0, and Player 2 sets X to 1. If player 2 set X to 0, it would increase its reward to 1, but also reduce the reward for player 1 from 3 to 1. For player 2 to increase its reward to the maximum available, then player 1 would need to adjust its value for X to 1.

For the experiment which is presented in section 4 an optimisation technique is used to calculate all rewards for each player based on all possible values chosen by all the players. Having all possible values available provides a reference for determining the best game playing strategy in NFG to maximise individual payoff, group payoff, or group and individual payoff, depending on the objective. This reference to payoffs can be used when opponents in an NFG are using a fixed strategy or a mixed strategy.

The optimisation process used in this experiment involves the following:

1. Generate all possible combinations of action probabilities for NFG
2. Calculate payoffs for each player playing all these different combination of action probabilities and determine the maximum payoff for each.

Generate Action Probabilities

In an NFG, with two actions for example, the sum of both actions adds up to 1. Action probabilities can be generated using different intervals, so there is not just 1 and 0. There are other probabilities which can be used. Using intervals of 0.25, the following combinations of action probabilities are available for a NFG with 2 actions:

X	**Y**
0	1
0.25	0.75
0.5	0.5
0.75	0.25
1	0

Table 3: Action probabilities in increments of 0.25

Note: For experimental purposes, smaller interval values will be used. Instead of 0.25, a value like 0.01 would be used instead. This would improve the accuracy of optimisation.

Determine Max Payoffs Once all the possible combinations of action probabilities are generated, the payoffs for using each of these action probabilities for each player against an opponent using each of these action probabilities is determined. From the payoffs generated on each play, the maximum payoff for each player is then determined.

This process is described as follows:

Player 1 (P1) is fixed with X at 0.25, and Y at 0.75. To determine the best payoff for Player 2 (P2) playing against these fixed values, payoffs for player 2 are calculated as follows:

		X	**Y**
Game 1	P1	0.25	0.75
	P2	0	1
Game 2	P1	0.25	0.75
	P2	0.25	0.75
Game 3	P1	0.25	0.75
	P2	0.5	0.5
Game 4	P1	0.25	0.75
	P2	0.75	0.25
Game 5	P1	0.25	0.75
	P2	1	0

Table 4: Action Probabilities for Player 2 in each game

In each of the games, the payoff for player 2 is calculated. At the end, the game which produces the maximum payoff for player 2 is selected, when playing against a player using a fixed strategy - just like player 1 in this example is.

This process can be expanded to determine the payoff for the group as well as the payoff for player 2, and enables optimizing a strategy for picking individual and/or group strategy for playing the game.

This process is repeated, with player 1 fixed for (X, Y) at (0.5, 0.5), then (0.75, 0.25), then (0, 1) and lastly (1, 0).

The whole process is then repeated but instead the values for player 2 are fixed to determine a strategy for player 1.

The algorithm, to calculate best responses, for this process is represented as follows:

Algorithm 1: Probability Actions Rewards

```
Data: List (X, Y), Payoff Function
Result: List Payoffs
1 for each player do
2     for For each action probability X, Y as i do
3         for For each action probability X, Y as j do
              // Calculate payoff - formulas in section 2.5.1
4         Select max payoff for i
```

Applying this algorithm produces all possible payoffs. Below is table 5, which shows the payoffs calculated for each game for player 2, using the payoff function from table 1 for prisoner dilemma:

		X	**Y**	Payoff
Game 1	P1	0.25	0.75	
	P2	0	1	1.5
Game 2	P1	0.25	0.75	
	P2	0.25	0.75	1.25
Game 3	P1	0.25	0.75	
	P2	0.5	0.5	1
Game 4	P1	0.25	0.75	
	P2	0.75	0.25	0.75
Game 5	P1	0.25	0.75	
	P2	1	0	0.5

Table 5: Payoffs for Player 2 in each game using Prisoner Dilemma Payoff function

From the payoffs produced, game 1 produces the maximum payoff. If player 1 is playing action probabilities of (X, Y) as (0.25, 0.75), the best response for player 2 is to play action probabilities of (0, 1), as this produces the maximum payoff.

3.5.3 Normal Form Games Strategies

In the previous subsection on optimisation, a process is described to find the best payoff for an opponent playing against an opponent which uses a fixed strategy. This process leads to the best payoff achievable using a predefined incremental value. For example, try playing the opponent using an action probability of 0, then 0.25, then 0.5, then 0.75, and finally 1. From each of these, determine which action probability gives the best payoff. The best payoff from this approach may not be the actual best payoff which can be achieved. In this example, the increments used were 0.25, but if a smaller increment was used, then the closer to the best action probability to play to get the actual best payoff will be achieved. In some cases it is possible to achieve the actual best payoff, but in some cases it is possible just to get close. Using smaller increments will increase performance execution cost, so a trade off maybe required if this is an issue. In general though, this approach will at least provide guidance as to the direction (increase or decrease) of the action probability to improve the payoff. This must be considered as part of any game playing strategy when learning to play a Normal Form Game (NFG). In a Normal Form Game like prisoner dilemma or stag hunt, the payoffs for both players, will end up the same. What is meant by this in this context is if player 1 uses an action probability of 1 for one action, then if the best payoff for player 2 is achieved by it setting its action probability for the same action to 1, then the inverse will also be true - If player 2 uses an action probability of 1 for one action, then the best payoff for player

1 will be achieved by it setting its action probability for the same action to 1. Realistically in this type of game, only the calculations to determine the best strategy for one player is required when playing against an opponent at fixed interval values. The process does not need to be repeated for the other player, as the results will be the same. While this is generally understood, calculations for both players are performed in the experiment presented in this paper, just to confirm that.

Some Normal Form Games though do require calculations for both players, as results will be different. An example of such a game is matching pennies which is discussed in section 4 as it is one of the games which is used as part of the experiment presented in this paper. Matching pennies requires one player to match the other players choice to gain a reward, while the other player gets a reward if their choice does not match the other players reward. The rewards in this scenario present different incentives for each of the players, so a strategy for both needs to be calculated. Games like stag hunt or prisoner dilemma use a reward structure where rewards are based on players cooperating or defecting. If both cooperate the the highest reward is achieved for the group and maybe also for the individual, whereas if both defect, you get a lower reward for the group. With this type of game the best strategy for one player can be worked out, and this will also be the best strategy for the other player.

In NFG it is possible to achieve Nash Equilibrium. When both players achieve a reward from chosen actions and no incentive exists to deviate on these actions in order to achieve a higher reward, then Nash Equilibrium is achieved [Röp+22]. In some NFG a pure strategy can exist to achieve Nash Equilibrium. A pure strategy involves each player choosing a particular action. For example in stag hunt a player can choose to cooperate or defect. This is a pure strategy. If both players choose cooperate then Nash Equilibrium is achieved as neither player can achieve a higher reward by deviating from cooperate. Other NFG like matching pennies, achieving Nash Equilibrium using a pure strategy is not possible, as the payoff function does not readily support this. Matching pennies is referred to as a zero sum game. A zero sum game is where the rewards for both players on each selecting actions will add up to 0. A zero sum game like matching pennies requires the use of what is known as a mixed strategy to achieve Nash Equilibrium, as one player needs a match to get the reward, while the other player needs no match to get the reward. The mixed strategy is based on selecting probabilities for each action. For example, in a two player game with two possible actions, the probabilities of the actions add up to 1. If action 1 is set to 1, then action 2 is set to 0. Adjusting the probability values of the actions, so they still add up to 1, to values between 0 and 1, it may be possible to achieve Nash Equilibrium this way. This approach is known as a mixed strategy. For example player 1 chooses a probability on action 1 of 0.6, and action 2 of 0.4, while player 2 chooses a probability on action 1 of 0.3, and action 2 of 0.7, to obtain Nash Equilibrium [RN16].

3.5.4 Reinforcement Learning

An approach to determine a best strategy for a playing playing a NFG can be used as part of reinforcement learning (RL).

An RL agent can be set up to interact with a service which provides advise on best actions for the RL agent to take. The service can act as a teaching agent. This service uses the optimisation process described in the last sub section 2.5.2 before any interaction with the RL agent. The service generates a lookup from this for payoffs.

On commencement of the learning process, the RL agent is informed by the service of the probability values on the actions being used by its opponent. The RL agent then randomly sets probability values for its own actions, and informs the teaching service of these values. The teaching service then checks these probabilities referencing its payoff lookup. This is done to check if the probability values chosen by the RL agent produce the maximum payoff, as available in the lookup.

If they do produce the maximum payoff, the RL agent is informed of this via a signal from the teaching service. The RL agent can then add this to its policy for future lookup to use again if playing an opponent with the same probability values set on its own actions.

If the action probabilities chosen by the RL agent do not produce a payoff which could be better, the teaching service can advise the RL agent using a signal to increase or decrease actions probabilities, and indicate by what percentage. The RL agent can choose to implement this advise in whatever way it is configured to do so. If the advise is acted upon by the RL agent, it will send the updated action probabilities to the teaching service and await for further feedback from the teaching service, which will confirm by way of signal whether the action probabilities are good, or further action (increase or decrease action probabilities) by the RL agent is required. This process continues until the RL agent receives a signal from the teaching service which it is configured to interpret as the best payoff available, or RL agent decides on its own, using a threshold value for example, that it has achieved an acceptable payoff. The action probabilities used by an RL agent maybe a slight adjustment from actual action probability value advised by the teaching service. The RL agent maybe configured to not fully thrust the service (adversarial), or due to other constraints maybe configured to just get within a certain range of advised solution from teaching service.

The process described is now presented for both the Teaching Service (TS) and the Reinforcement Learning (RL) Agent as algorithms 2 and 3:

Algorithm 2: Teaching Service

```
Data: None
Result: Explanation, CFE
1  Start TS;
2  Generate All Action Probabilities at Intervals;
3  Generate Payoff Function;
4  Run Algorithm 1;
5  Store output from Algorithm 1 as payoff list;
6  Fix TS player action probabilities;
7  while Wait for RL Agent requests do
8      if First request from RL agent then
9          Signal action probabilities;
10     else if action probabilities received from RL agent then
11         Look up payoff list for maximum payoff;
12         if RL actions achieve maximum payoff then
13             Signal to RL agent no change required;
14         else
15             Explanation: Signal adjust probabilities to RL agent;
16             CFE: Signal direction and percentage to adjust by to RL agent;
17 End TS;
```

Algorithm 3: Reinforcement Learning Agent

```
Data: None
Result: Action Probabilities
1  Start RL Agent;
2  Signal to TS ready to begin;
3  while Wait for signal from TS do
4      if Signal action probabilities then
5          Signal random action probabilities to TS;
6      else if Signal CFE then
7          Amend RL agent player action probabilities;
8          Signal updated action probabilities to TS;
9      else
10         Store actions as policy;
11 End RL Agent;
```

Note: Algorithms presented here uses a fixed strategy. An RL agent learns how to play an opponent with fixed action probabilities.

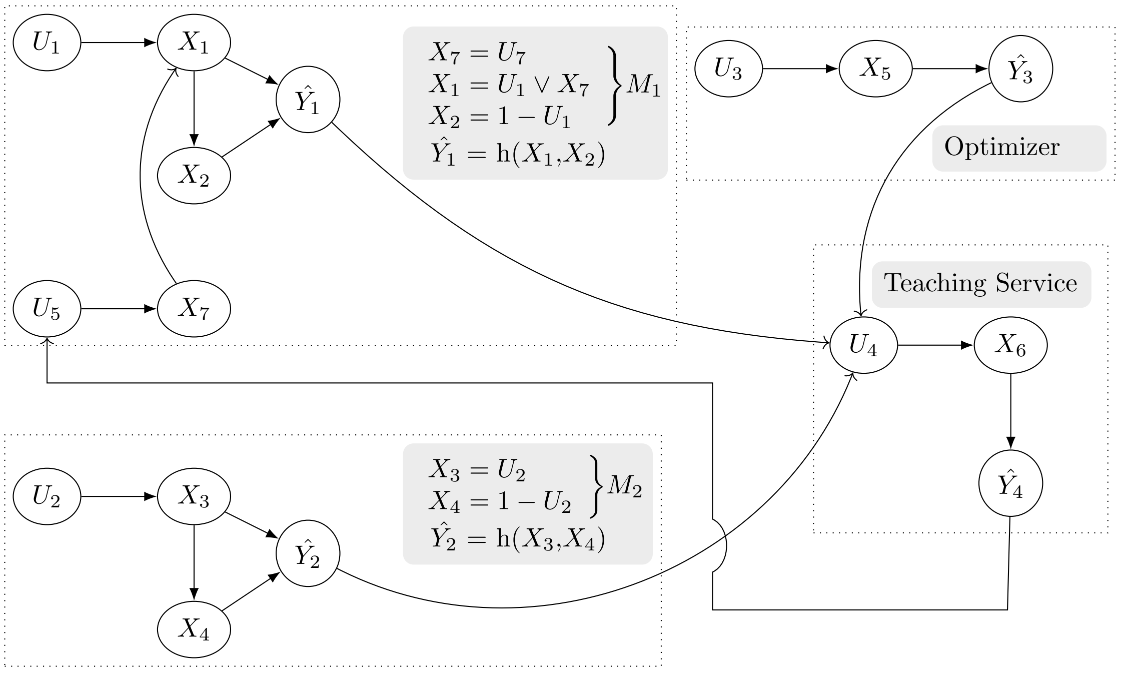

Figure 2: Causal Graph [KSV21; OK22] representing framework described in this section.

3.5.5 Framework

The framework described in this section using the worked example is represented as a causal graph in figure 2.

The Opponent is similar to Figure 1, except $\hat{Y_2}$ does not include any calculations. The values for X_3 and X_4 are passed to the Teaching Service, U_4. The RL Agent is similar to figure 1, except $\hat{Y_1}$ does not include any calculations. The values for X_1 and X_2 are passed to the Teaching Service, U_4. The Optimizer, U_3, represents the increments for generating action probabilities - see Table 3 for example. Any value from 0 to 1 can be used for U_3. $\hat{Y_3}$ calculates and produces a list of all possible payoffs for players - see Algorithm 1 and an example list in Table 5. U_3 represents $\hat{Y_1}$, $\hat{Y_2}$ and $\hat{Y_3}$. $\hat{Y_4}$ determines the payoff for the RL Agent, and checks the list from $\hat{Y_3}$ to see if a better payoff is possible. See Algorithm 2. A CFE is constructed based on this - $\hat{Y_4}$. The result from $\hat{Y_4}$ is passed to the RL agent - U_5 and based on this, a value for X_1 is determined, if $\hat{Y_4}$ indicates an adjustment to the X_1 value. If $\hat{Y_4}$ does not indicate an adjustment, then value for X_1 is added to a policy - see Algorithm 2.

Note that the framework for Reinforcement Learning with CFE, uses 3 inputs - U_1, U_2 and U_3. In the example of Prisoner Dilemma. U_1 represents the action probability for cooperate selected by the RL agent - a random value from 0 to 1. U_2 represents the action probability for cooperate selected by the opponent - a random value from 0 to 1. U_3 represents the incremental value used by the optimizer when calculating payoffs - see section 3.5.2, tables 3, 4 and 5.

3.6 Related Work

The ideas presented in this article were formed from doing a literature review on XAI, CFE, RL and NFG. Some of the articles reviewed focus on XRL (Explainable Reinforcement Learning). XRL is primarily concerned with providing explanations of actions chosen which lead to a policy. While this paper is focused on using XAI with RL, it is not a paper on XRL. The distinction here is that XAI, or more specifically, CFE are used to to aid the learning process. Of course, the explanations generated during learning, can be reviewed after learning to understand decisions made by the agent, as advised by the teaching service, so there is an XRL element to it [GD22], [PV20].

An RL architecture is present in a number of papers which uses an actor/critic process for the RL. This process is adapted in a number of ways. One way is the critic is responsible for the rewards which aids the learning process. Another way, is the critic is part of the RL agent and reviews the rewards, which informs the actor process, which is also part of the RL agent, in making decisions on actions to choose [PV20], [Ngu+21].

Other articles discuss generating and improving CFE provided by using Multi Agent Multi Objective RL and Deep RL [DVC23].

One interesting article reviewed, [Ngu+21], uses a Multi Agent RL architecture for generating CFE. Each agent, has an actor and critic. The critic advises the actor. The critic uses a process to optimize the generation of CFE. It focuses on the smallest changes possible to produce a different output. The critic shares this information with the actor, which uses this information for generating the CFE. The CFE is human interpretable and optimizes what changes are required to produce a different output.

This approach has similarities to the approach considered here. CFE's are used to provide an explanation but also to aid the learning process. The teaching service is similar to the critic in that it generates the CFE. While the RL agent acts on this similar to how the actor acts on it. The difference is that, with the actor/critic approach, the main aim is to generate the CFE and show what would happen if applied, whereas for this article, the CFE is acted upon to enhance learning. Also, there is similarities in terms of Multi Agent, but in [Ngu+21], the agents are both RL agents, whereas in this article the two agents are a teaching service and an RL agent.

Counterfactual Regret (CFR) is a useful technique for determining better outcomes. This technique generates CFE. It is an iterative process whereby an outcome is obtained, then CFEs are generated. Each CFE is then assessed, and the one which produces the best outcome is selected. If the outcome from this CFE produces a better outcome than the original outcome, then this CFE is used as the solution. In game playing, a player uses a strategy, so if the CFE chosen in this process produces a better result than the players strategy, then the players strategy is updated to reflect this [Lan+19], [He+22]. This process helps to improve the policy and demonstrates how CFEs can be used for this. This is similar to what is proposed in this article, except in this article an optimization approach is used to determine a better outcome. The process described in this article leads to one CFE produced from optimization, which is used to advise the RL agent on any improvements it can make, if any are available. The optimization process described in this article generates a list of CFE for this purpose, so one can be chosen from the list each time the RL agent performs an action. The CFE is not generated at each stage. Of course, in different circumstances, for other types of games, CFR would be more suitable when considering cost of performing optimization, or it just may not be realistic to use an optimization process similar to the one used in this paper.

Another article [OK22], discusses CFE and NFG in a Multi Agent scenario. This article looks at the role of recourse in NFG to determine if recommended changes would lead to a positive result for the individual (one agent) but have a negative impact on the group (other agents). RL is not used here, but the ideas for determining recourse, and assessing the impact of implementing the recourse will be used as part of the RL process in this paper. [OK22] references [KSV21] which looks at the impact of actions by an individual agent, how CFE are generated and how steps are provided for the individual agent to act upon. In this paper, the individual agent is the RL agent, and the RL process uses CFE to advise the RL agent how it must adjust its actions to achieve a more favourable outcome. The RL process includes the RL agent and other agents, a teaching service, and opponent, so it is a multi agent process.

Another approach in literature discusses using policies to replay a game, and pausing the game at certain points. At these points, CFEs are generated, which help determine if the policy has the the correct action or actions at that point. This appears to be a quality assurance process though not stated as such [Ols+21].

RL can be used to create policies for generating CFEs. The policy can then be used to generate CFEs along with the result produced by the action or actions. This approach reduces cost of generating the CFE after the result using a post hoc method. Post hoc approaches to generating CFEs involves generating CFEs after the result is determined and assessing the best CFE(s) to present with the result. If RL is used to create a policy for generating CFE, then the CFE will be quicker to generate and more likely to be realistic, fair, actionable, and more robust, once the RL process is configured to achieve this [SVK21].

RL is used to create policies from environments. CFE can be used to generate slightly adjusted environments, and policies can then be generated from these adjusted environments. The RL agent can then generate a generic policy from all these policies [Fro+22].

In [ACJ19], an RL agent performs an action at a certain stage. This is reflected in the policy generated from the RL process. Explanations as to why the RL agent performed that action at that stage can be generated using counterfactuals. At a certain stage it can be determined what other actions could be chosen at that stage using CFE, and this should help determine why the RL agent went with a particular action for its policy. This article seems to focus on using counterfactuals to understand why the RL agent choose the action it did. It is like generating a list of options, and from that list choose an action. This can lead to understanding the behaviour of the RL agent. It chose this action from all these, and you can ascertain why. This gives the explanation. The counterfactuals help generate this list of options. The experiment in [ACJ19] uses saliency maps - check policy at certain state, present a saliency map, then present other saliency maps from counterfactuals and compare. This is a post RL process. CFE are not used as part of the learning process.

There are approaches in related work which have similar themes to what is discussed in this paper, mainly CFR and actor/critic. This paper differs as it aims to demonstrate using optimization to generate a single CFE and to use this process to inform the RL process for generating policies. CFEs are available post hoc so can be used to explain policy. The value of this approach is that more clarity is provided so the policy is trusted, and assurance is provided by using an optimization approach. This will hopefully lead to new approaches to adapting CFE for providing explanations, thrust in decisions made, and transparency where it does not exist. The approach could be applied to other tasks than playing NFG. The framework proposed in this paper could be evolved for other scenarios so it is application and / or model agnostic. This would make it a state of the art framework based on the approach used and provide some quality assurance to any learning process.

4 Implementation

4.1 Introduction

For implementing the ideas discussed in this article, a jupyter notebook was used to:

- Calculate payoffs
- Generate All Action Probabilities at Intervals
- Implement Optimization Technique to calculate best responses to fixed and mixed strategies for all action probabilities at intervals - Algorithm 1.
- Create graphs to show results
- Perform some testing
- Provide commentary on the process

A client / server architecture was created for the Reinforcement Learning Process to demonstrate the implementation of Algorithms 2 and 3. This implementation is used to:

- Demonstrate interactions between agents in a NFG environment.
- Calculate best responses
- Generate Explanations and Counterfactual Explanations.
- Act on Counterfactual Explanations
- Demonstrate use and creation of a policy for Reinforcement Learning.

The notebook and the code for the client / server architecture is available at:

URL FOR GITHUB HERE

4.2 Optimisation and Responses

The notebook is used to develop the ideas for optimisation and responses described in this article. The purpose of this is to ensure a valid process to obtain the best responses for NFGs using probability values at fixed intervals is possible. The notebook includes commentary which describes the development of these ideas, and includes tests to ensure the code used works as expected.

A summary of tasks included in the notebook, and the order in which they are completed is as follows:

- Implement Payoff function as a matrix for the NFG stag hunt.
- Implement a function to calculate rewards for players in an NFG using the payoff function.

- Test a single game using this function to ensure payoffs produced are as expected.
- Implement a function to calculate responses for a two player, two action NFG, where one player has a fixed action probability, and the action probability for the other player is set at different values in increments and the payoff at each increment is calculated.
- Test this function and produce a graph from the results of this function showing the payoff at each increment for the other player. From the results and graph it is possible to determine what action probability for the other player produces the best payoff playing against a fixed action probability.
- Implement a best responses function to calculate action probabilities at intervals which produce the best payoff against an opponent with fixed action probability at each interval.
- Implement a framework for a two player, two or three action NFG, which in future will be extendable for more than three actions, and potentially for more than two players.
- Generate all action probabilities for an NFG at configurable interval values.
- Using all action probabilities generated, calculate payoffs, and determine the action probabilities that give the best payoff for a player playing an opponent whose action probabilities are fixed at each interval.
- Test generating all action probabilities and using the framework for the NFG stag hunt, produce a graph showing the action probability which produces best payoff for each of the two players against an opponent at each fixed interval. This demonstrates the strategy is the same for both players as the payoff function supports this as equilibrium is possible and a pure strategy can be used for this payoff function.
- Test again using this framework, but this time for a NFG with three actions - rock, paper, scissor and print out action probabilities which produce the best payoff for each of the two players against an opponent at each interval. This also demonstrates the strategy is the same for both players as the payoff function supports this as equilibrium is possible and a pure strategy can be used for this payoff function. Playing rock, paper, scissors, also demonstrates using a zero sum payoff function.
- Test again using this framework, but this time for a NFG with two actions, matching pennies. This demonstrates a NFG with a payoff function that requires using a mixed strategy for equilibrium, and is also a zero sum payoff function.
- Produce a graph showing action probabilities which produce the best payoff for each of the two players against an opponent at each interval for matching pennies, as determined by the payoff function, equilibria exists

but only when using a mixed strategy instead of a pure strategy. The strategy required is not the same for both players in this game of matching pennies.

- Note: Mixed and Pure Strategies, and Equilibrium are discussed in sub section 2.5.3.

4.3 Reinforcement Learning

For reinforcement learning, a client server approach is used. There is a server which acts as the Teaching Service (TS), and two clients. One client acts as an opponent, while the other acts as the Reinforcement Learning (RL) Agent. This framework is currently implemented for 2 players, 2 actions Normal Form Game. An example of such a game is stag hunt.

The TS is started. Then the first client is started and connected to the TS. It is assigned the role of RL Agent by the TS.

Then the second client is started and is connected to the TS and assigned the role of opponent. The opponent client is prompted to enter an action probability for first action. This is a fixed action probability which the Reinforcement Learning agent will learn to play against. When the opponent provides an action probability to the TS, the TS calculates what the best response is to this action probability. This best response is then held in memory by the Teaching Service, and is used for advising the RL agent during the learning process.

When the TS is calculating the best response to the opponents entered action probability, it calculates the best responses to all possible opponents action probabilities, and stores these responses in memory. The TS does this only once, after the opponent has entered its first action probability. When the opponent enters any subsequent action probabilities, the TS can check its memory for all the best responses to select the best one to use against the opponents currently used action probability. All these action probabilities held in memory are set at intervals. So if using intervals of 0.2 for example, then in memory, the TS will have the best response for an RL agent to an opponent using 0, or 0.2, or 0.4, or 0.6, or 0.8 or 1.

The RL agent is then advised by the TS what action probability its opponent is using. The RL agent then checks its policy, which is a json file, to see if it has an action probability from previous games available to play against its opponents action probability value to obtain the best payoff. If the RL agent has an action probability available in its policy, then it will send this value to the TS. Upon receiving this value from the RL agent, the TS checks if this value matches the value it has for best response held in memory. If it matches, it advises the RL agent, the best action probability was used by the RL agent. If the action probability sent by the RL agent does not match the best response, then the TS determines if the action probability sent needs to increase or decrease, and by what percentage. The RL agent is then advised of this outcome. Upon receiving this advise, the RL agent adjusts its action probability by the recommended percentage, and sends the updated action probability to the TS. The TS then confirms if this is the best action probability to use. It will confirm this as long as there is no issue with the update of the action probability. Of course, if there is an issue, it will advise once again whether to increase or decrease and by what percentage, based on the most recent action probability received. This process will continue until the RL agent sends an action probability which produces the best response.

Once the TS confirms to the RL agent that it has used the best action probability in response to its opponents action probability to obtain the best payoff, then the RL agent needs to update its policy to store this response so it is available for future use.

When the RL agent is initially informed by the TS of its opponents action probability value, it checks its policy, and if a response is not available in the policy, then the RL agent, randomly selects a response value and sends it to the TS.

The advise provided by the TS is an explanation, and this explanation includes recourse for action for the RL agent to take - the CFE - if recourse is required. The recourse is to increase or decrease by a certain percentage.

With the client/server model approach used here, once an RL agent has learned the best action probability to play against an opponents fixed action probability and updates its policy, the opponent is prompted to enter another action probability value. The RL agent is then informed of this, and a similar process is used again to determine best response for the RL agent. By similar, it is more or less the same process. The one small difference is that the TS does not need to calculate the best response to advise the RL agent, as after the first action probability provided by the opponent, all the best responses are available in memory, so just a look up is now required by the TS.

This is a manual process but perfectly demonstrates the concept here of using XAI and CFE to aid the learning process. For example, if the intervals used to teach an RL agent is set to 0.25, then the opponent when prompted would enter 0, then the RL agent would learn the best response to 0. Then the opponent is prompted again once the agent has learned the best response to 0, and 0.25 is entered. This repeats for 0.5, 0.75 and then 1. At the end, the RL agent has a policy in place for the five values it is played against, with the best responses for each. This policy can then be used for playing games in the future.

This process could be automated, so that the Teaching Service does not require an opponent client, and can generate the action probabilities for the opponent automatically. In terms of understanding the concept here, the manual approach with the server acting as a Teaching Service, and two clients, opponent and RL agent, helps to gain a better understanding of the concepts of the approach described in this article. Automating this process could be used as part of a future work possibility with Reinforcement Learning.

4.3.1 Example

An example of a game demonstrating the interactions from this client server model is now presented. In this example the RL agent is learning to play a fixed action probability using the NFG stag hunt at intervals of 0.2:

Teaching Service

The server is started. The server acts as the Teaching Service. Once the server is started, it waits for client connections.

RL Agent

Client is started

Teaching Service

Request from RL agent client recieved and connection established.

RL Agent

Type 'hello' at prompt to indicate to Teaching Service that RL Agent client is now ready to start. By doing this, it indicates to the Teaching Service, that this client is taking on the role of RL Agent.

Wait for response from Teaching Service.

Opponent

Client is started

Teaching Service

Request from Opponent client recieved and connection established.

Opponent

Type 'hello' at prompt to indicate to Teaching Service that Opponent client is now ready to start. By doing this, it indicates to the Teaching Service, that this client is taking on the role of Opponent.

Wait for response from Teaching Service.

Teaching Service

Send message to Opponent

Opponent

Opponent receives message from Teaching Service - "Enter a number from 0 to 1 inclusive."

Enter 0 at the Opponent prompt

Teaching Service

Teaching Service receives a value of 0. This is the action probability on the first action the opponent is using. In the case of stag hunt, and based on the payoff function, this is the action probability for stag that the row player (opponent) is using, so the action probability value for hare is set to 1 (1 - 0).

The Teaching Service calculates the best responses (action probability values) for the RL agent at intervals to achieve the best payoff it can against this opponent, and stores them in memory. The intervals in this example are 0.2.

The Teaching Service then checks its memory for the best response for the RL agent to an opponent using the action probability of 0 and stores this value separately in memory. The value it stores in memory in this example is 0.0.

The Teaching Service sends a message to the RL agent.

RL Agent

RL Agent receives message from Teaching Service:

You are playing against a fixed strategy with action 0.0. Enter a number from 0 to 1 inclusive.

RL agents looks up if response exists in policy. If action exists, the prompt indicates this. In this example, one does exist, so the following is printed to the RL Agents screen:

action exists: 0.0

Based on this advise, enter 0.0 at the RL Agent prompt.

Teaching Service

Teaching Service receives action probability value of 0.0 for stag action from the RL Agent and compares it to the best response value it has stored in memory.

In this example, it matches, so it sends a message to the RL Agent, and also the Opponent.

RL Agent

RL Agent receives message from Teaching Service:

You have picked the best value 0.0.

Your opponent is playing 0.0.

No further input required from you for now. Thank you.

Add to policy

The RL agent then updates its policy, to ensure it stores a response for 0.0, as 0.0. However in this example it already exists, so an update on the policy is performed to ensure the correct response is captured.

Opponent

Opponent receives message from Teaching Service:

Enter another number from 0 to 1

Enter 0.2 at the prompt.

Teaching Service

Teaching Service receives a value of 0.2. This is the action probability on the first action the opponent is using. In the case of stag hunt, and based on the payoff function, this is the action probability for stag that the row player (opponent) is using, so the action probability value for hare is set to 0.8 (1 - 0.2).

The Teaching Service then checks its memory for the best response for the RL agent to an opponent using the action probability of 0.2 and stores this value separately in memory. The value it stores in memory in this example is 0.0.

The Teaching Service sends a message to the RL agent.

RL Agent

RL Agent receives message from Teaching Service:

You are playing against a fixed strategy with action 0.2. Enter a number from 0 to 1 inclusive.

RL agents looks up if response exists in policy. If action exists, the prompt indicates this. In this example, one does not exist, so a value for RL agent is entered manually:

A value of 0.1 is entered at the RL Agent prompt. The value 0.1 is randomly chosen by the user.

Teaching Service

Teaching Service receives action probability value of 0.1 for stag action from the RL Agent and compares it to the best response value it has stored in memory.

In this example, it does not matches, so it sends a message to the RL Agent indicating this.

RL Agent

RL Agent receives message from Teaching Service:

Decrease probability of action 0.1 by 10.0

Based on this advise, the agent updates action probability from 0.1 to 0.0, send sends this to to Teaching Service.

Teaching Service

Teaching Service receives action probability value of 0.0 for stag action from the RL Agent and compares it to the best response value it has stored in memory.

In this example, it matches, so it sends a message to the RL Agent, and also the Opponent.

RL Agent

RL Agent receives message from Teaching Service:

You have picked the best value 0.0.

Your opponent is playing 0.2.

No further input required from you for now. Thank you.

Add to policy

The RL agent then adds it to its policy, to ensure it stores a response for 0.2, as 0.0.

Opponent Opponent receives message from Teaching Service:

Enter another number from 0 to 1

Enter 0.4 at the prompt.

Teaching Service

Teaching Service receives a value of 0.4. This is the action probability on the first action the opponent is using. In the case of stag hunt, and based on the payoff function, this is the action probability for stag that the row player (opponent) is using, so the action probability value for hare is set to 0.6 (1 - 0.4).

The Teaching Service then checks its memory for the best response for the RL agent to an opponent using the action probability of 0.4 and stores this value separately in memory. The value it stores in memory in this example is 1.0.

The Teaching Service sends a message to the RL agent.

RL Agent

RL Agent receives message from Teaching Service:

You are playing against a fixed strategy with action 0.4. Enter a number from 0 to 1 inclusive.

RL agents looks up if response exists in policy. If action exists, the prompt indicates this. In this example, one does not exist, so a value for RL agent is entered manually:

A value of 1.0 is entered at the RL Agent prompt. The value 1.0 is randomly chosen by the user.

Teaching Service

Teaching Service receives action probability value of 1.0 for stag action from the RL Agent and compares it to the best response value it has stored in memory.

In this example, it matches, so it sends a message to the RL Agent, and also the Opponent

RL Agent RL Agent receives message from Teaching Service:

You have picked the best value 1.0.

Your opponent is playing 0.4.

No further input required from you for now. Thank you.

Add to policy

The RL agent then adds it to its policy, to ensure it stores a response for 0.4, as 1.0.

Opponent

Opponent receives message from Teaching Service:

Enter another number from 0 to 1

Enter 0.6 at the prompt.

Teaching Service

Teaching Service receives a value of 0.6. This is the action probability on the first action the opponent is using. In the case of stag hunt, and based on the payoff function, this is the action probability for stag that the row player (opponent) is using, so the action probability value for hare is set to 0.4 (1 - 0.6).

The Teaching Service then checks its memory for the best response for the RL agent to an opponent using the action probability of 0.6 and stores this value separately in memory. The value it stores in memory in this example is 1.0.

The Teaching Service sends a message to the RL agent.

RL Agent

RL Agent receives message from Teaching Service:

You are playing against a fixed strategy with action 0.6. Enter a number from 0 to 1 inclusive.

RL agents looks up if response exists in policy. If action exists, the prompt indicates this. In this example, one does not exist, so a value for RL agent is entered manually:

A value of 0.5 is entered at the RL Agent prompt. The value 0.5 is randomly chosen by the user.

Teaching Service

Teaching Service receives action probability value of 0.5 for stag action from the RL Agent and compares it to the best response value it has stored in memory.

In this example, it does not matches, so it sends a message to the RL Agent indicating this.

RL Agent

RL Agent receives message from Teaching Service:

Increase probability of action 0.5 by 50.0

Based on this advise, the agent updates action probability from 0.5 to 1.0, send sends this to to Teaching Service.

Teaching Service

Teaching Service receives action probability value of 1.0 for stag action from the RL Agent and compares it to the best response value it has stored in memory.

In this example, it matches, so it sends a message to the RL Agent, and also the Opponent

RL Agent

RL Agent receives message from Teaching Service:

You have picked the best value 1.0.

Your opponent is playing 0.6.

No further input required from you for now. Thank you.

Add to policy

The RL agent then adds it to its policy, to ensure it stores a response for 0.6, as 1.0.

Opponent

Opponent receives message from Teaching Service:

Enter another number from 0 to 1

Enter 0.8 at the prompt.

Teaching Service

Teaching Service receives a value of 0.8. This is the action probability on the first action the opponent is using. In the case of stag hunt, and based on the payoff function, this is the action probability for stag that the row player (opponent) is using, so the action probability value for hare is set to 0.2 (1 - 0.8).

The Teaching Service then checks its memory for the best response for the RL agent to an opponent using the action probability of 0.8 and stores this value separately in memory. The value it stores in memory in this example is 1.0.

The Teaching Service sends a message to the RL agent.

RL Agent

RL Agent receives message from Teaching Service:

You are playing against a fixed strategy with action 0.8. Enter a number from 0 to 1 inclusive.

RL agents looks up if response exists in policy. If action exists, the prompt indicates this. In this example, one does not exist, so a value for RL agent is entered manually:

A value of 1.0 is entered at the RL Agent prompt. The value 1.0 is randomly chosen by the user.

Teaching Service

Teaching Service receives action probability value of 1.0 for stag action from the RL Agent and compares it to the best response value it has stored in memory.

In this example, it matches, so it sends a message to the RL Agent, and also the Opponent

RL Agent

RL Agent receives message from Teaching Service:

You have picked the best value 1.0.

Your opponent is playing 0.8.

No further input required from you for now. Thank you.

Add to policy

The RL agent then adds it to its policy, to ensure it stores a response for 0.8, as 1.0.

Opponent

Opponent receives message from Teaching Service:

Enter another number from 0 to 1

Enter 1.0 at the prompt.

Teaching Service

Teaching Service receives a value of 1.0. This is the action probability on the first action the opponent is using. In the case of stag hunt, and based on the payoff function, this is the action probability for stag that the row player (opponent) is using, so the action probability value for hare is set to 0.0 (1 - 1.0).

The Teaching Service then checks its memory for the best response for the RL agent to an opponent using the action probability of 1.0 and stores this value separately in memory. The value it stores in memory in this example is 1.0.

The Teaching Service sends a message to the RL agent.

RL Agent

RL Agent receives message from Teaching Service:

You are playing against a fixed strategy with action 1.0. Enter a number from 0 to 1 inclusive.

RL agents looks up if response exists in policy. If action exists, the prompt indicates this. In this example, one does not exist, so a value for RL agent is entered manually:

A value of 0.8 is entered at the RL Agent prompt. The value 0.8 is randomly chosen by the user.

Teaching Service

Teaching Service receives action probability value of 0.8 for stag action from the RL Agent and compares it to the best response value it has stored in memory.

In this example, it does not matches, so it sends a message to the RL Agent indicating this.

RL Agent

RL Agent receives message from Teaching Service:

Increase probability of action 0.8 by 20.0

Based on this advise, the agent updates action probability from 0.8 to 1.0, send sends this to to Teaching Service.

Teaching Service

Teaching Service receives action probability value of 1.0 for stag action from the RL Agent and compares it to the best response value it has stored in memory.

In this example, it matches, so it sends a message to the RL Agent, and also the Opponent

RL Agent

RL Agent receives message from Teaching Service:

You have picked the best value 1.0.

Your opponent is playing 1.0.

No further input required from you for now. Thank you.

Add to policy

The RL agent then adds it to its policy, to ensure it stores a response for 1.0, as 1.0.

Opponent

Opponent receives message from Teaching Service:

Enter another number from 0 to 1

4.3.2 Policy

A policy is created as a result of the learning process, as described in the example. The policy is stored in a JSON file, policy.json. On completion of the example, the policy.json file is as follows:

Listing 1: policy created from example

```
{
    "0.0": 0.0,
    "0.2": 0.0,
    "0.4": 1.0,
    "0.6": 1.0,
    "0.8": 1.0,
    "1.0": 1.0
}
```

From this policy, if a player is playing an opponent in stag hunt with a fixed strategy action probability of 0.4 on stag for example, then the best response for the player is to use an action probability on stag of 1.0 as this response will achieve the best payoff for the player.

5 Experiments

In the previous section, implementation of solutions was described for optimisation strategies, and using CFE for reinforcement learning. Using the implementation, various experiments were carried out to obtain results. These experiments are detailed in the sub sections which follow. Some of these experiments are available in the notebook and the others require using the client server framework to confirm results from the experiments. In each subsection which follows, this is noted.

5.1 Experiment 1

The first experiment is to determine which action probability will produce the best payoff for a player playing against another playing using a fixed action probability. The best payoff is determined from calculating the payoffs achieved using action probabilities at incremental values from 0 to 1. It is the best payoff as determined from the increments and not necessarily the actual best payoff, as the action probabilities to achieve this may fall in between the increments.

For this experiment, the NFG stag hunt was used and player 1 had its action probability for stag set to 0. Player 2 had its action probability for stag set to 0, and the total payoff for player 2 was calculated. This process was repeated where the action probability for player 2 was updated to 0.25, then updated to 0.5, then updated to 0.75 and finally updated to 1. After each update for player 2, the total payoff was captured, and from this the action probability which produces the highest payoff for player 2 is determined.

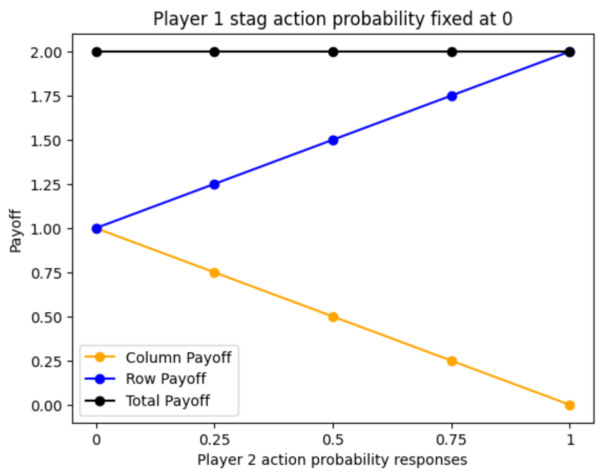

Figure 3: Results from Experiment 1

Figure 2 shows the results from this experiment. It includes the payoffs for player 1, player 2, and the sum of these payoffs achieved for each of the action probabilities used by player 2.

From the results, the best payoff player 2 can achieve when player 1 is using an action probability on stag of 0, is a payoff of 1. To achieve this payoff of 1, player 2 needs to use an action probability on stag of 0. This is the expected result.

This experiment was repeated but using a different incremental value of 0.01 instead of 0.25. The results produced were the same. The graph shows the same trend as that demonstrated in figure 1. This was the expected behaviour.

The code and results for this experiment is available in the notebook.

The experiment was repeated a number of times, but instead of fixing action probability for stag for player 1 to 0, it was fixed at different values. Each times it was fixed, the experiment to determine best action probability on stag for player 2 to achieve the best payoff for player 2 was carried out. Player 1 action probabilities for stag were fixed using incremental values of 0.25. For each value it was fixed at, the expected results for player 2 were obtained. This was done just to confirm the code executed as expected.

5.2 Experiment 2

The second experiment determines the action probability which will produce the best payoff for a player against an opponent at different intervals. The intervals are the action probabilities for the opponent. For example, if player 1 has a fixed action probability on stag, in the NFG stag hunt, of 0.2, then the action probability on stag for player 2 to use against player 1 to achieve the best payoff is determined. This is done for player 2, when player 1 is action probability for stag is fixed at 0, 0.2, 0.4, 0.6, 0.8 and 1, if using increments of 0.2. It is then repeated for player 1, so the best action probability for player 1 to use to achieve the best payoff when player 2 has its action probability fixed at 0, then at 0.2, then at 0.4, then at 0.6, then at 0.8 and finally 1.

The action probabilities for stag are set in this experiment, and the action probabilities for hare are determined by subtracting the value for stag action probability from 1.

The results from this experiment are shown in figure 3 for stag.

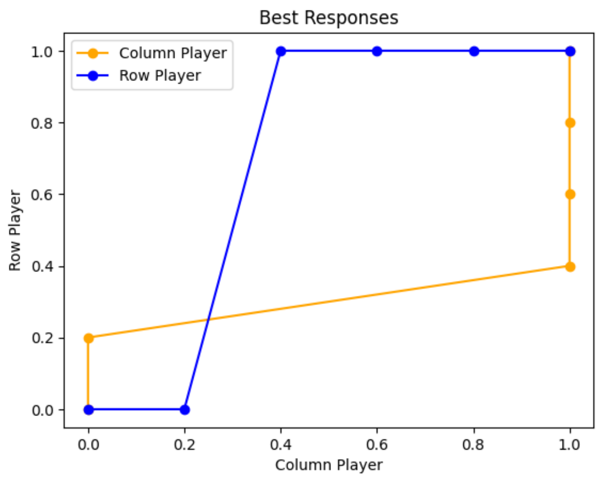

Figure 4: Results from Experiment 2 using increments of 0.2

The results in figure 3 are from an experiment which uses increments of 0.2. This experiment was repeated but using increments of 0.01. Results for this are shown in figure 4.

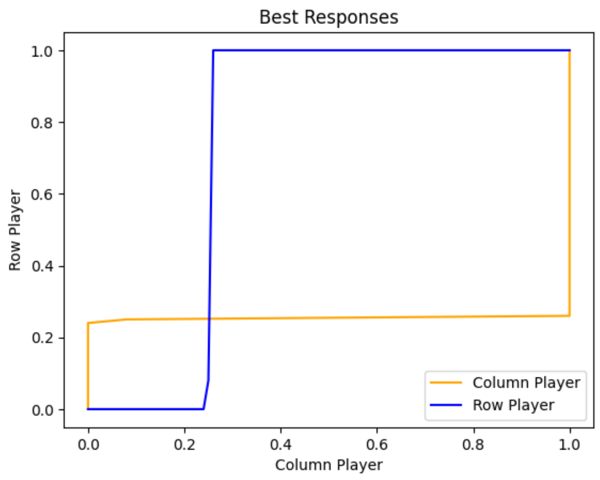

Figure 5: Results from Experiment 2 using increments of 0.01

A similar output is achieved when using smaller intervals. Results provide better accuracy showing responses between 0.2 and 0.4, where the best response changes to 1. When using increments of 0.2, when using 1 for best response, it indicates to use this response when opponent is at 0.4, but when using increments of 0.01, it indicates to use a response of 1 just above 0.2. So it is actually closer to 0.2 than 0.4. This provides a more accurate indicator.

The code and results for this experiment is available in the notebook.

5.3 Experiment 3

This is a similar experiment to the previous one, except it is for a NFG with three actions instead of two action. The NFG used for this experiment is Rock, Paper, and Scissors. Probability action values are set for each of rock, paper and scissors. The values for each are from 0 to 1, and the total of these probability values is 1.

All the action probabilities are generated using intervals of 0.25 and then intervals of 0.01.

When the action probability values are generated using intervals of 0.25, the following values are created:

Listing 2: Action Probabilities in increments of 0.25

```
[(0.0, 0.0, 1.0), (0.0, 1.0, 0.0), (0.0, 0.75, 0.25),
(0.0, 0.25, 0.75), (0.0, 0.5, 0.5), (0.25, 0.0, 0.75),
(0.25, 0.75, 0.0), (0.25, 0.5, 0.25), (0.25, 0.25, 0.5),
(0.5, 0.5, 0.0), (0.5, 0.0, 0.5), (0.5, 0.25, 0.25),
(0.75, 0.0, 0.25), (0.75, 0.25, 0.0), (1.0, 0.0, 0.0)]
```

Each value is a list that contains three values. The first value is for paper, the second value is for scissors, and the third value is for rock.

The process used to generate this list of values was also used in experiment 2, but each list contains 2 values, one for stag and one for hare. Similar process is also used in the next experiment, experiment 4.

The values from each list are played against each of the values in the other lists, and the best payoff for each is determined.

For example, if playing against (0.0, 0.0, 1.0), which represents rock, then use (1.0, 0.0, 0.0), which is paper. Paper covers rock, so this wins and gives the best payoff. A sample of other values from the output was checked to determine if action probabilities recommended do produce the best payoff against a fixed list of action probabilities. Expected results were achieved.

A similar process was performed using intervals of 0.01 and expected results were also achieved.

5.4 Experiment 4

This experiment is similar to experiment 2, except the NFG Matching Pennies is used. In matching pennies, there are two actions, tails and heads. The action probabilities for tails are set in this experiment, and the action probabilities for head are determined by subtracting the value for tails action probability from 1.

The results obtained when using intervals of 0.2 for tails, are shown in figure 5.

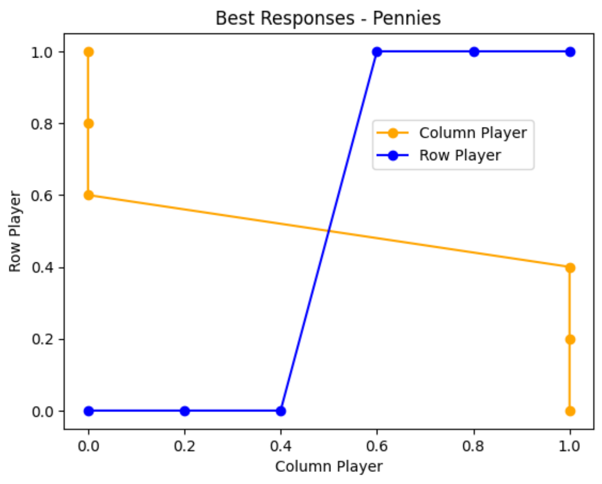

Figure 6: Results from Experiment 4 using increments of 0.2

Results are as expected. For example, column players gets best payoff using an action probability of 1 when row player has their action probability set to 0. Row player gets best payoff using action probability of 1 when column players has their action probability also set to 1. So different strategies required depending on the player and the graph output demonstrates this.

The results obtained when using intervals of 0.01, are shown in figure 6.

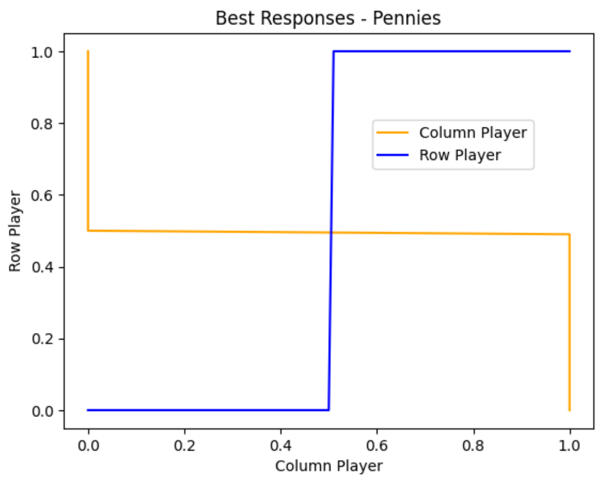

Figure 7: Results from Experiment 4 using increments of 0.01

As was seen in experiment 2, smaller increments produce a more accurate result. This is the case in experiment 4 as well.

From this experiment the following can be concluded:

- Row player uses an action probability of 1 when column player uses any action probability value greater than 0.5.
- Row player uses an action probability of 0 when column player uses any action probability value less than 0.5.
- Column player uses an action probability of 1 when row player uses any action probability value less than 0.5.
- Column player uses an action probability of 0 when row player uses any action probability value greater than 0.5.
- For both column and row player the best action probability to use is 0, if their opponent is using an action probability of 0.5

This result shows the optimisation technique as applied here works as expected.

5.5 Experiment 5

In experiment 2, the NFG stag hunt was used. In this experiment, a similar NFG is used, Prisoner Dilemma.

The results from this experiment are shown in figure 7 for stag.

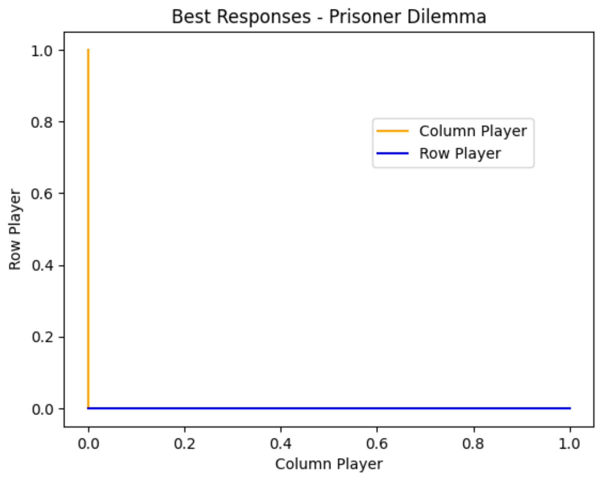

Figure 8: Results from Experiment 6 using increments of 0.01

The experiment used increments of 0.01 and results achieved were as expected. The best response for row player to any fixed probability for column player is always to set the probability for cooperate to 0. Also, the best response for column player to any fixed probability for row player is always to set the probability for cooperate to 0.

The code and results for this experiment is available in the notebook.

5.6 Experiment 6

This experiment demonstrates using the client server framework for CFE in Reinforcement Learning. The process involved here is outlined in detail in section 3.3.

For this experiment the NFG stag hunt is used. The opponent sets the action probability manually for stag in intervals of 0.1, starting with 0, and ending with 1. At each interval, the Reinforcement Learning (RL) Agent provides an action probability for stag, any value from 0 to 1. This value is set manually at a prompt. The Teaching Service then advises the RL Agent, on updates it needs to make to achieve a better payoff, if an update is required. The Teaching Service, records this in a log file which can be consulted at the end of the RL process. The RL agent, once it has learned an action probability value to use, updates its policy.

The policy which exists at the end of this process is as follows:

Listing 3: policy created from stag hunt experiment

```
{
    "0.0": 0.0,
    "0.1": 0.0,
    "0.2": 0.0,
    "0.3": 1.0,
    "0.4": 1.0,
    "0.5": 1.0,
    "0.6": 1.0,
    "0.7": 1.0,
    "0.8": 1.0,
    "0.9": 1.0,
    "1.0": 1.0
}
```

Note that the values on the left, "0.5" for example, are the action probability values for the Teaching Service, and the value on the right is the action probability value the RL agent learned to achieve the best payoff.

The CFE at the end of this process demonstrates explanations and advise given for RL agent at each interval is as follows:

Listing 4: CFE at end of experiment

```
Decrease probability of action 0.1 by 10%
You have picked the best value 0.0.
Your opponent is playing 0.0.

Decrease probability of action 0.5 by 50%
You have picked the best value 0.0.
Your opponent is playing 0.1.

Decrease probability of action 0.8 by 80%
You have picked the best value 0.0.
Your opponent is playing 0.2.

You have picked the best value 1.0.
Your opponent is playing 0.3.

Increase probability of action 0.7 by 30%
You have picked the best value 1.0.
Your opponent is playing 0.4.

Increase probability of action 0.5 by 50%
You have picked the best value 1.0.
```

```
Your opponent is playing 0.5.

Increase probability of action 0.9 by 10%
You have picked the best value 1.0.
Your opponent is playing 0.6.

Increase probability of action 0.2 by 80%
You have picked the best value 1.0.
Your opponent is playing 0.7.

Increase probability of action 0.6 by 40%
You have picked the best value 1.0.
Your opponent is playing 0.8.

Increase probability of action 0.6 by 40%
You have picked the best value 1.0.
Your opponent is playing 0.9.

Increase probability of action 0.0 by 100%
You have picked the best value 1.0.
Your opponent is playing 1.0.
```

Further tests were carried out as part of this experiment to check if an action probability is available for the RL Agent to use in its policy, that the RL agent does use it. The experiment showed that it does in fact use the policy as expected, and requires no updates from the Teaching Service to update its own action probability value as it has learned the correct value to use previously.

5.7 Experiment 7

This experiment is similar to experiment 6. For this experiment the NFG prisoner dilemma is used. The reinforcement learning agent learns the best responses against an opponent playing fixed action probabilities on cooperate at intervals of 0.2 from 0 to 1.

The policy which exists at the end of this process is as follows:

Listing 5: policy created from prisoner dilemma experiment

```
{
    "0.0": 0.0,
    "0.2": 0.0,
    "0.4": 0.0,
    "0.6": 0.0,
    "0.8": 0.0,
    "1.0": 0.0
}
```

Note that the values on the left, "0.2" for example, are the action probability values for the Teaching Service, and the value on the right is the action probability value the RL agent learned to achieve the best payoff.

The CFE at the end of this process demonstrates explanations and advise given for RL agent at each interval is as follows:

Listing 6: CFE at end of experiment

```
Decrease probability of action 0.2 by 20%
You have picked the best value 0.0.
Your opponent is playing 0.0.

Decrease probability of action 0.2 by 20%
You have picked the best value 0.0.
Your opponent is playing 0.2.

Decrease probability of action 0.6 by 60%
You have picked the best value 0.0.
Your opponent is playing 0.4.

You have picked the best value 0.0.
Your opponent is playing 0.6.

Decrease probability of action 0.1 by 10%
You have picked the best value 0.0.
Your opponent is playing 0.8.

Decrease probability of action 0.9 by 90%
You have picked the best value 0.0.
Your opponent is playing 1.0.
```

Results from both experiment 6 and 7 were as expected. The RL agent in both experiments learned expected strategy.

6 Other Solutions

In this section, other solutions which are available for determining the best responses in NFG are assessed. The primary purpose of this is to confirm the implementation used in the experiments for calculating the best responses is valid.

Two technologies were chosen for assessment and comparison purposes. These are:

- Ramo
- Nashpy

An introduction to each is given in the subsections which follow. These subsection include details of experiments, and results from these experiments along with a discussion highlighting similarities and differences, where applicable, to the implementation used for the experiments described in the previous section.

6.1 Ramo

Ramo is an abbreviation for Rational Agents with Multiple Objectives. It is an algorithmic game theory framework offering a collection of algorithms and utilities for computing or learning (approximate) equilibria in multi-objective games. [Röp22]

Ramo offers a number of functions which can be used for determining equilibria in such games. The NFG used for this article are for single objective games. However the functions provided by Ramo can be used for determining the best responses in these single objective games with some amendments.

Ramo offers a number of utility functions for multi-objective games which can be reused but the framework does support custom utility functions if none of the utility functions offered by Ramo are suitable for the desired task. Ramo does not have a utility function for a single-objective game, so one was created in order to calculate the payoffs for such games.

Once the utility function is implemented and payoffs calculated, a Ramo function is used to calculate the best response for each player against a fixed value. As part of the experiment to use Ramo, calculations are performed invoking the function to calculate the best responses in steps of 0.01, from 0 to 1.

This experiment with Ramo was performed on the NFG stag hunt. The results obtained are similar to those obtained from experiments on stag hunt described in the previous section.

6.1.1 Stag Hunt

The results for best responses for each player in stag hunt are shown in graphical form in figure 8 below.

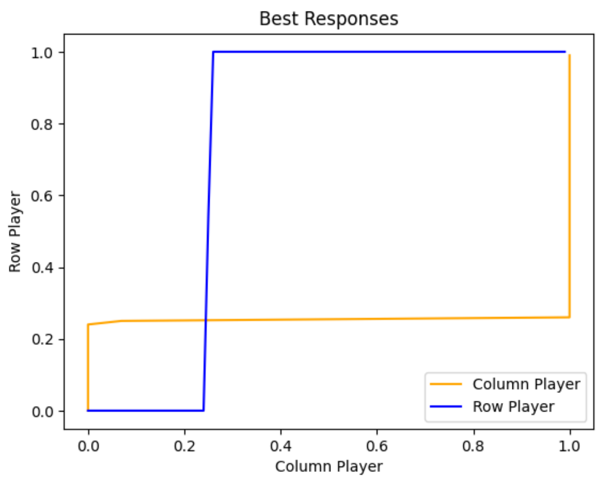

Figure 9: Results from Stag Hunt Experiment with Ramo using increments of 0.01

The results are similar to those shown in figure 4.

6.2 Nashpy

Nashpy is a python library for two player games. It provides functionality for playing various types of games, including NFG, and developing strategies for playing these games.

Nashpy contains functionality for determining the best response for a player against an opponent. [Kni23]

As part of the experiment to use Nashpy, calculations are performed invoking the Nashpy function to calculate the best responses in steps of 0.01, from 0 to 1.

This experiment with Nashpy was performed on the NFG matching pennies for comparison purposes. The results obtained are similar to those obtained from experiments on matching pennies described in the previous section.

6.2.1 Matching Pennies

The results for best responses for each player in matching pennies are shown in graphical form in figure 9 below.

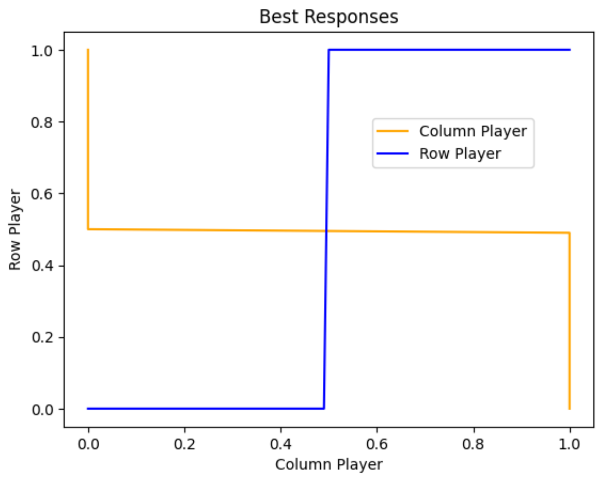

Figure 10: Results from Matching Pennies Experiment with Nashpy using increments of 0.01

The results are similar to those shown in figure 6.

6.3 Conclusion

The results seen from Ramo and Nashpy confirm that the approach used for calculating best responses in this article are valid. Experiments with the other NFG using Ramo and Nashpy were also carried out and results for best response were similar to those obtained using the approach outlined in this article. The implementation and results for these experiments using Ramo and Nashpy are available at [GITHUB REF HERE]

The solver tool available in excel was also used to check best responses to some randomly selected values, and in each case the expected results as recorded in this article were obtained.

7 Discussion

The results from this experiment do show XAI, and CFE in particular could have role to play to in reinforcement learning. The CFE enable the RL agent to learn action probability values to enable it develop a policy which will give the desired results. In sub section 2.6, it was identified that this approach, while sharing other ideas from other work, does provide a possible new approach to include with current RL methods. Further experiments would need to be carried out to determine how useful it actually is. However it can be concluded that using XAI as part of the RL process, explanations can be extracted to understand decisions of the RL agent during the learning process, and thrust that solutions learned by the agent are the desired solutions or at least close to the desired solutions.

The optimisation approach used in this experiment to find a good solution is not unique to this experiment. There exists many other optimisation techniques. As part of this experiment, the optimisation approach used, was compared to other optimisation approaches to sanity check, that results obtained were similar. This was done solely to ensure the optimisation approach adapted in this experiment is a valid approach. See section 5.

While the work presented in this article does demonstrate a role for XAI in RL, it is just a start. The work in this article could be used as a basis for further work exploring the role the role XAI in RL. The experiments used in this article focused on NFG. These games were the environment. Future work could explore different environments. For example, in route planning or diagnostics, using existing optimisation techniques along with XAI, to develop policies. Confirming XAI has a role in other environments with developing policies would at least demonstrate its role has greater scope beyond NFG. The NFG used in the experiments, are single objective two player. There is scope to explore environments where more that one objective and/or more than two players exist. This would tie into any work relating to using XAI for RL on other environments.

The process for developing policies in the experiment described in this article uses a bespoke technique involving adding/updating action values to/in a policy file. Other techniques exist for creating policies in RL, and these could be incorporated as part of any future work and could lead to other future possibilities with XAI and policy creation in RL.

As mentioned in sub section 2.6, CFE have been used in environments with an Actor/Critic framework, though this work did not use CFE as part of the learning process. Using CFE as part of the learning process in an environment with Actor/Critic, could be explored to determine if any benefits from this could be obtained.

In conclusion, the work for this article does show a role for XAI and CFE in RL, and provides a platform for future work in this area.

A Code

Code used for experiment is available on GitHub - give URL.

References

[Kre89] David M. Kreps. “Nash Equilibrium”. In: *Game Theory*. Ed. by John Eatwell, Murray Milgate, and Peter Newman. London: Palgrave Macmillan UK, 1989, pp. 167–177. ISBN: 978-1-349-20181-5. DOI: 10.1007/978-1-349-20181-5_19. URL: https://doi.org/10.1007/978-1-349-20181-5_19.

[Oqu+12] MA Oquendo et al. “Machine learning and data mining: strategies for hypothesis generation”. In: *Molecular psychiatry* 17.10 (2012), pp. 956–959.

[RN16] S. Russell and P. Norvig. “Artificial Intelligence: A Modern Approach, Global Edition”. In: Pearson Education, 2016, pp. 666–673. ISBN: 9781292153964.

[ACJ19] Akanksha Atrey, Kaleigh Clary, and David Jensen. “Exploratory not explanatory: Counterfactual analysis of saliency maps for deep reinforcement learning”. In: *arXiv preprint arXiv:1912.05743* (2019).

[Lan+19] Marc Lanctot et al. “OpenSpiel: A framework for reinforcement learning in games”. In: *arXiv preprint arXiv:1908.09453* (2019).

[Rud19] Cynthia Rudin. “Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead”. In: *Nature Machine Intelligence* 1.5 (2019), pp. 206–215. URL: https://arxiv.org/pdf/1811.10154.pdf.

[Bar+20] Alejandro Barredo Arrieta et al. “Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI”. In: *Information Fusion* 58 (2020), pp. 82–115. ISSN: 1566-2535. DOI: https://doi.org/10.1016/j.inffus.2019.12.012. URL: https://www.sciencedirect.com/science/article/pii/S1566253519308103.

[Dan+20] Marina Danilevsky et al. “A Survey of the State of Explainable AI for Natural Language Processing”. In: (2020). DOI: 10.48550/ARXIV.2010.00711. URL: https://arxiv.org/abs/2010.00711.

[DR20] Arun Das and Paul Rad. “Opportunities and Challenges in Explainable Artificial Intelligence (XAI): A Survey”. In: (2020). DOI: 10.48550/ARXIV.2006.11371. URL: https://arxiv.org/abs/2006.11371.

[Mah20] Batta Mahesh. “Machine learning algorithms-a review”. In: *International Journal of Science and Research (IJSR).[Internet]* 9.1 (2020), pp. 381–386.

[PV20] Erika Puiutta and Eric MSP Veith. “Explainable reinforcement learning: A survey”. In: *International cross-domain conference for machine learning and knowledge extraction*. Springer. 2020, pp. 77–95.

[VDH20] Sahil Verma, John P. Dickerson, and Keegan Hines. “Counterfactual Explanations for Machine Learning: A Review”. In: *CoRR* abs/2010.10596 (2020). arXiv: 2010.10596. URL: https://arxiv.org/abs/2010.10596.

[BP21] Vaishak Belle and Ioannis Papantonis. “Principles and Practice of Explainable Machine Learning”. In: *Frontiers in Big Data* 4 (2021). ISSN: 2624-909X. DOI: 10.3389/fdata.2021.688969. URL: https://www.frontiersin.org/article/10.3389/fdata.2021.688969.

[Cyr+21] Kristijonas Cyras et al. “Argumentative XAI: A Survey”. In: *CoRR* abs/2105.11266 (2021). arXiv: 2105.11266. URL: https://arxiv.org/abs/2105.11266.

[KSV21] Amir-Hossein Karimi, Bernhard Schölkopf, and Isabel Valera. “Algorithmic recourse: from counterfactual explanations to interventions”. In: *Proceedings of the 2021 ACM conference on fairness, accountability, and transparency*. 2021, pp. 353–362.

[MKR21] Aniek F. Markus, Jan A. Kors, and Peter R. Rijnbeek. “The role of explainability in creating trustworthy artificial intelligence for health care: A comprehensive survey of the terminology, design choices, and evaluation strategies”. In: *Journal of Biomedical Informatics* 113 (2021), p. 103655. ISSN: 1532-0464. DOI: https://doi.org/10.1016/j.jbi.2020.103655. URL: https://www.sciencedirect.com/science/article/pii/S1532046420302835.

[Ngu+21] Tri Minh Nguyen et al. “Counterfactual explanation with multi-agent reinforcement learning for drug target prediction”. In: *arXiv preprint arXiv:2103.12983* (2021).

[Ols+21] Matthew L Olson et al. “Counterfactual state explanations for reinforcement learning agents via generative deep learning”. In: *Artificial Intelligence* 295 (2021), p. 103455.

[SVK21] Robert-Florian Samoilescu, Arnaud Van Looveren, and Janis Klaise. “Model-agnostic and scalable counterfactual explanations via reinforcement learning”. In: *arXiv preprint arXiv:2106.02597* (2021).

[Sla+21] Dylan Slack et al. *Counterfactual Explanations Can Be Manipulated*. 2021. DOI: 10.48550/ARXIV.2106.02666. URL: https://arxiv.org/abs/2106.02666.

[Fro+22] Julius Frost et al. “Explaining reinforcement learning policies through counterfactual trajectories”. In: *arXiv preprint arXiv:2201.12462* (2022).

[GD22] Jasmina Gajcin and Ivana Dusparic. “Counterfactual Explanations for Reinforcement Learning”. In: *arXiv preprint arXiv:2210.11846* (2022).

[He+22] Kangxin He et al. “Finding nash equilibrium for imperfect information games via fictitious play based on local regret minimization”. In: *International Journal of Intelligent Systems* 37.9 (2022), pp. 6152–6167.

[KA22] Chokri Kooli and Hend Al Muftah. “Artificial intelligence in healthcare: a comprehensive review of its ethical concerns”. In: *Technological Sustainability* 1.2 (2022), pp. 121–131.

[OK22] Andrew O’Brien and Edward Kim. “Toward Multi-Agent Algorithmic Recourse: Challenges From a Game-Theoretic Perspective”. In: *The International FLAIRS Conference Proceedings*. Vol. 35. 2022.

[Röp22] Willem Röpke. *Ramo: Rational Agents with Multiple Objectives.* https://github.com/wilrop/mo-game-theory. 2022.

[Röp+22] Willem Röpke et al. "On nash equilibria in normal-form games with vectorial payoffs". In: *Autonomous Agents and Multi-Agent Systems* 36.2 (2022), p. 53.

[YL22] Liangru Yu and Yi Li. "Artificial Intelligence Decision-Making Transparency and Employees Trust: The Parallel Multiple Mediating Effect of Effectiveness and Discomfort". In: *Behavioral Sciences* 12.5 (2022). ISSN: 2076-328X. DOI: 10.3390/bs12050127. URL: https://www.mdpi.com/2076-328X/12/5/127.

[DVC23] Richard Dazeley, Peter Vamplew, and Francisco Cruz. "Explainable reinforcement learning for broad-xai: a conceptual framework and survey". In: *Neural Computing and Applications* (2023), pp. 1–24. URL: https://doi.org/10.1007/s00521-023-08423-1.

[Kni23] Vince Knight. *Nashpy: a python library for 2 player games.* https://github.com/drvinceknight/Nashpy. 2023.

[Uzu23] Levent Uzun. "ChatGPT and academic integrity concerns: Detecting artificial intelligence generated content". In: *Language Education and Technology* 3.1 (2023).

[Web+23] Leander Weber et al. "Beyond explaining: Opportunities and challenges of XAI-based model improvement". In: *Information Fusion* 92 (2023), pp. 154–176. ISSN: 1566-2535. DOI: https://doi.org/10.1016/j.inffus.2022.11.013. URL: https://www.sciencedirect.com/science/article/pii/S1566253522002238.