University of Bristol and University of West of England

Robotics

MSc Dissertation: Adversarial Learning in a Predator-Prey game

Author:

Nawid Keshtmand

Supervisor:

Professor Jonathan Lawry

Co-Supervisor:

Dr. Raul Santos-Rodriguez

September 12 2019

Contents

List of Figures

List of Tables

Acknowledgements

Here, the author would like to sincerely express his gratitude to his supervisors, Professor Jonathan Lawry and Dr. Raul Santos-Rodriguez. Thank you both for your support and guidance on areas related to and outside the scope this project. Moreover, the author wants to show his appreciation to Linton Singarayer for his friendship, fruitful discussions and time he takes to provide feedback on the author's work. In addition, the author would like to show his gratitude to Cheuk Chan for providing humour and motivation to the author in continuing to work on this project. Finally, the author is grateful for the support from his family and friends, in particular to his brothers, Ali and Bahram, as well as Ali's wife Roo. The author wishes you a lifetime full of happiness and thanks you all for the continuous support, guidance and faith they provide him.

Abstract

The use of machine learning models is becoming increasingly popular in a wide range of industries, however for high risk areas it is important to understand how agents behave. One high risk area is the area of autonomous vehicles such as self-driving cars or drones, where there can be competitive or adversarial agents present. In such cases, the reward or utility received by an agent is dependent on the actions taken by other agents in the population. This work involved modifying an existing predator-prey environment in order to examine how the policy of an agent behaves in an adversarial scenario. The predator and prey agents were trained simultaneously using the Deep Q-network algorithm. The policy of the agent was examined by quantifying the number of similar actions between two consecutive policies of an agent, as well as well using Decision trees as a proxy model of the policy from a neural network. From the Decision tree proxy models, the number of similar actions and the tree edit distance between two different Decision trees was calculated as a measure of similarity between different policies of a particular agent. This information was then used to find the optimum number of clusters required to cluster the different policies. The results suggest that policies change at a certain rate and it is possible for the policy of the predator to change even in the case where the prey was stationary by choosing degenerate actions which gave similar rewards. Different Decision trees had policies which were distinct to one another but behaved in a intuitive manner to maximse their expected reward. The optimum number of clusters for the different Decision trees was shown to be one but by performing clustering with a higher number of clusters, it was generally seen that consecutive policies were generally more similar to one another.

1 Introduction

1.1 Background

Reinforcement Learning is an active area of machine learning concerned with how software agents learn to perform actions through trial and error. Reinforcement Learning has a diverse range of applications from such as robotics, waste heat management and protein folding [6, 7, 8]. The focus of this project is on using Reinforcement Learning to train competing agents to perform different tasks, in order to see how the sequence of actions of an agent, which is known as a policy, change in distributed systems when there is feedback present. This is important due to the deployment of high risk AI technology, such as autonomous vehicles, where there can be an adversarial aspect present and there is a requirement in understanding how the agents behave for safety reason.

To obtain an indication of how the policy of an agent was changing, such as whether there was a pattern in the change in policy and whether the policy obtained made intuitive sense, several different approaches were used. The approaches used involved quantifying the number of actions which were identical for two consecutive policies of an agent and comparing Decision trees representing the different policies of an agent.

1.2 Aims and objectives

To achieve the aim of examining how the policy of the agent changed in a situation where the reward an agent received depended on the actions of other agents, several objectives were met:

- Developing a gridworld simulation environment for the different agents present
- Studying single and multi-agent Reinforcement Learning algorithms which can be used to train agents
- Investigating how an single-agent Reinforcement Learning algorithm known as the 'Deep Q-network' algorithm can be adapted for a competitive agent scenario
- Investigating proxy models of a a neural network to improve explainability
- Quantifying the similarity between different proxy models

1.3 Motivation

Multi-agent systems (MAS) are defined as a 'distributed systems of independent actors, called agents, that cooperate or compete to achieve a certain objective [9]'. Using a MAS is highly desirable as they can improve efficiency by having different agents specialised for certain tasks or dividing a task into several sub tasks which are completed simultaneously [10].

MAS have been used for several different applications and have been particularly effective for the purpose of energy management. Due to the global impact of climate change, there has been great effort by the government in minimising the use of harmful non-renewable fuels in favour of using renewable energy sources. One method of energy management which is based on a MAS approach is a smart grid. A smart grid refers to an electric grid where producers and consumers can be modelled as reactive agents, and this allows the distribution of energy to be varied depending on the needs of the consumers at a particular moment. In the study by Schatten et al 2016, a smart grid based approach to energy management was shown to lead to higher battery levels in an Eco-village in Crotia [11]. A potential model for a smart grid can be seen in Fig. 1.1 [1] where different agents are specialised for different tasks and co-ordinate with one another to optimise energy use for different locations.

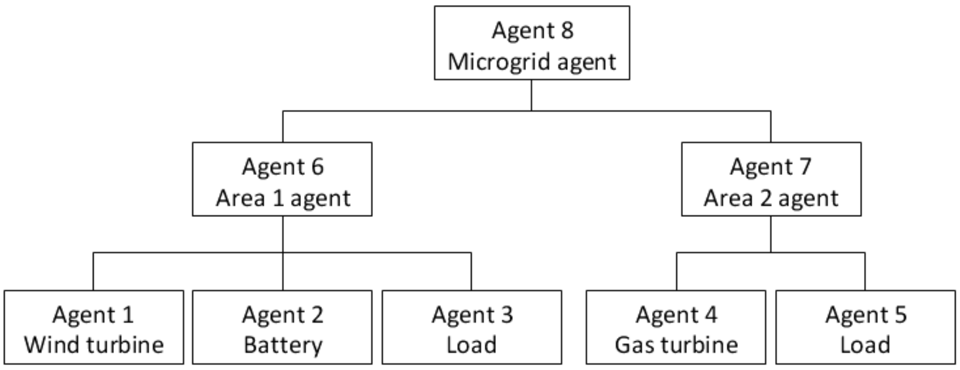


Figure 1.1: MAS system used for smart grid where different agents perform different roles. Each area is coordinated by an area agent, and these agents themselves are managed by a microgrid agent [1]

Another important application of a MAS is in controlling a swarm of drones for coverage, search and rescue or military action. Whilst, the movement of single drones can be automated effectively, the movement of several drones simultaneously is limited by the skills of a human operator and availability of qualified pilots. Therefore, being able train agents to perform these tasks autonomously could lead to a range of benefits including increasing the number of agents used simultaneously and optimizing drone trajectories to maximise coverage of an area. However, it can be difficult to use a MAS to perform a certain task as the behaviour of each agent in the MAS affects the behaviour of other agents. Hardcoding the behaviour of all the agents in MAS is one potential approach, however this can impractical due to the vast amount of rules required to take into account the interactions of each agent. An alternative strategy to solving this problem is by using a learning approach such as Reinforcement Learning, where the behaviour of the different agents can learn and adapt with time as they perform different actions in an environment.

There has been substantial progress in using Multi-Agent Reinforcement Learning (MARL) in recent years, but most of this work has been related to the case of fully cooperative settings. However, in a MAS, such as autonomous vehicles, there may be cases where agents are only partially cooperative with one another and contain elements of competition [12]. This could potentially lead to situations where individual autonomous vehicles would prioritise the transport of its own passenger which could lead to collisions occurring. It is important to understand how competition between agents arise in order to design MAS where agents behave in the way that is intended by the user. This is important in areas such as the emergency medical dispatch decisions after a natural disaster or a terrorist attack. If competition arises between the different agents in these situations, it can lead to movement of several different ambulances to a specific location and there would be diminishing returns for each agent as there would not be sufficient coverage of other locations [13]. Furthermore, in the case where drones are used for military action, this would likely lead to opposing groups using drones in retaliation, therefore understanding how an agent behaves in the presence of an competitive agent could enable advantageous military situations [14, 15].

To effectively implement an MAS using MARL, it is crucial to understand how an agent will behave in the presence of a competing agent. Therefore the focus of this project is to obtain an understanding of how the policy of the agent will change as a result of the actions of a competing adversary.

To study adversarial learning, a Predator-Prey game was chosen as it is a scenario which has already been studied in the literature for different areas such as biology and machine learning. Furthermore, it is a scenario which can vary significantly in complexity such as number of predators, number of prey, capabilities of each agent as well as the dynamics of the environment. This allowed the author to have significant flexibility in the different types of experiment possible such as having experiment related to the prey acting in different ways (stationary, predictable, intelligent behaviour patterns). Furthermore, the results of a predator-prey game can generalise to the real-life area of drone use, where a predator agent emulates a drone searching for a prey agent which would be target. Potential targets could be a lost traveller which could be modelled by a stationary or random moving prey, or a potential terrorist threat which can be represented as a intelligent prey agent.

2 Literature Review

2.1 Reinforcement Learning

Reinforcement Learning is a field where an agent learns to solve sequential decision problems through a trial and error learning process where no direct supervision is generally provided to the agent on what action to select. There are many different fields of Reinforcement Learning and has been applied to ranging from manipulation of robotic arms, strategies to optimise teaching and stock trading [6, 16, 17]. Reinforcement Learning can be thought as a combination of two main threads of research. One thread is the area of dynamic programming and optimal control which involves the problem of designing controller for a given system such that a certain optimality criterion is achieved [2]. The second thread of research is the area of learning through trial and error which has roots in behavioural psychology where the aspect of classical conditioning was seen in famous experiments such as Pavlov's dog as well other animals such as cats and rats [18, 19].

The two main entities in Reinforcement Learning are agents and an environment. An agent is a computer program that acts for a user. Agents can be solely software based or they can be embodied for the case where the software is paired with a robot body [20]. The history of an agent from the past till the current time step t can be summarised by a state s_t which is represented by features of the agent. The environment is an area where one or more agents can act upon [2]. Reinforcement Learning involves an agent being in state s_t at a discrete time step t and performing an action a_t which enables the agent to transition to a new state s_{t+1} and obtain a reward r_{t+1} from the environment. This interaction is formalized as a Markov Decision Process (MDP) which is a tuple $\langle \mathcal{S}, \mathcal{A}, T, r, \gamma \rangle$, where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a finite set of actions. $T(s, a, s')$ is a transition function which governs the transitions between states as shown by Eqn. 2.1. The reward function, r, relates the reward obtained in a particular state s after taking the action a, which is given by Eqn. 2.2 and γ is the discount factor which represents how much the agent cares about rewards at future time steps [21].

$$T(s, a, s') = P\left[S_{t+1} = s' | S_t = s, A_t = a\right] \tag{2.1}$$

$$r(s, a) = E\left[R_{t+1} | S_t = s, A_t = a\right] \tag{2.2}$$

A simplified version of the interactions between an agent and the environment is shown in Fig. 2.1 [2].

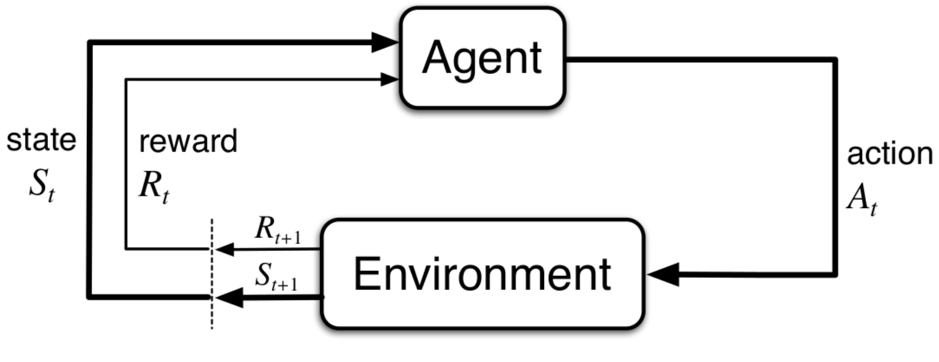


Figure 2.1: The agent–environment interaction in a Markov decision process [2]

The goal of the agent is to learn actions which will enable it to solve the sequential decision problem by maximising its long term scalar reward signal shown by Eqn. 2.3 [3]:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... + \gamma^{T-t-1} R_T = \sum_{i=0}^{T-t-1} \gamma^i R_{t+i+1} \tag{2.3}$$

Where R_{t+i} represents the reward at different time steps.

Action selection is governed by a policy, π, which is a function mapping state s to a probability distribution over actions a. Initially, an agent will adopt a random policy where it attempts all the different actions in a state with equal probability, but as it undergoes the trial and error learning process, it will learn an optimal policy π^*, which maximises its expected return. There are 2 main ways to learn a policy. One method is known as a policy based method where the policy for all the states is directly parameterised by a single parameter vector θ [2, 22]. The second approach to learn a policy is a value based approach where the reward obtained from a policy is represented by a metric known as a value function. This approach is generally known as value based Reinforcement Learning. There are two types of value functions, the state value function and the state action value function. The state value function V$_\pi$(s), defined by Eqn. 2.4, quantifies the total reward obtained by following the policy π starting in state s [23].

$$v_\pi(s) = E_\pi \left[G_t | S_t = s, \pi\right] \tag{2.4}$$

The action value function Q$_\pi$(s,a) defined by Eqn.2.5 quantifies the total reward obtained by performing action a in state s and then following the policy at all subsequent states [2].

$$Q_\pi(s,a) = E_\pi \left[G_t | S_t = s, A_t = a, \pi\right] \tag{2.5}$$

The optimal action-value function $Q^*\pi(s,a)$ is the maximum expected return achievable over all policies after entering a state and taking an action a. The optimal value is then given by Eqn. 2.6 [24]:

$$Q^*(s,a) = max_\pi Q_\pi(s,a) \tag{2.6}$$

An optimal policy can be derived from finding $Q^*(s,a)$ at every state and selecting the action which gives the highest $Q^*(s,a)$. These $Q^*(s,a)$ are found by using an equation known as the Bellman equation to iteratively update the Q values, which is given by Eqn. 2.7 [25]:

$$Q_{i+1}(s,a) = E\left[r + \gamma max_{a'} Q_i(s',a') | s,a\right] \tag{2.7}$$

2.2 Deep Reinforcement Learning

The approach of using the Bellman equation as a iterative update for each different state-action pair is effective for small state spaces however it is impractical for larger state spaces where generalization is required. However, this problem with scaling can be overcome with the use of function approximators such as a neural network. Therefore, despite having a long history, research in Reinforcement Learning has exponentially increased in the last few years due to improvements in function approximators such as deep learning architectures, increases in computational power and the wide accessibility of available data used to train these neural networks. The essential milestones of Reinforcement Learning can be seen in Fig. 2.2. This has led to the advent of Deep Reinforcement Learning [25].

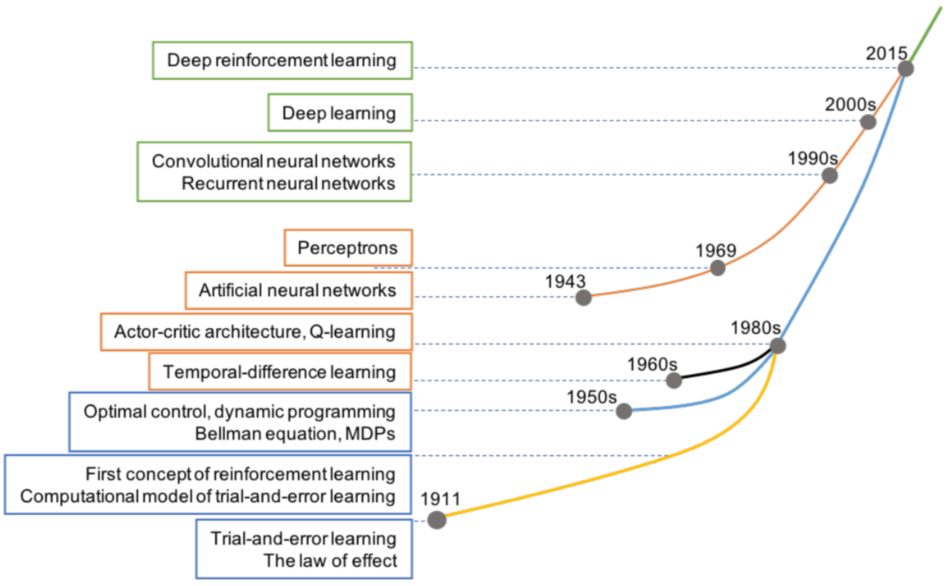

Deep reinforcement learning
2015
Deep learning
2000s
Convolutional neural networks
Recurrent neural networks
1990s
Perceptrons
1969
Artificial neural networks
1943
Actor-critic architecture, Q-learning
1980s
Temporal-difference learning
1960s
Optimal control, dynamic programming
Bellman equation, MDPs
1950s
First concept of reinforcement learning
Computational model of trial-and-error learning
1911
Trial-and-error learning
The law of effect

Figure 2.2: Emergence of Deep Reinforcement Learning through different essential milestones [3]

An early success story of Reinforcement Learning using a neural network as a function approximator is *TD-gammon*, which learnt the game backgammon using Reinforcement Learning by approximating the value function using a multi-layer perceptron with only a single hidden layer [23]. However, follow up work on *TD-gammon* was unsuccessful until the work done by Silver et Al in 2013 where the term 'Deep Reinforcement Learning' was first introduced [25]. In the past, Reinforcement Learning involved required features of a state to be carefully hand engineered by a human user which can be difficult and time consuming. This process is known as feature engineering [26]. However, deep neural networks are especially useful compared to other function approximators as they can generalise to large state spaces as well as automatically extract features from raw data, which is known as representation learning [27]. In the work done by Silver et Al in 2013, a deep Convolutional neural network enabled features to be extracted solely from sensory information in the form of pixel images. In this case, the different images corresponded to different states and based on the actions taken in the different states, the Q values could be approximated from the parameters of the neural network [25]. This was used to train an agent to learn control policies for playing seven different Atari games.

Similar to traditional Reinforcement Learning, after the use of using deep neural networks to approximate Q values, neural networks have also been used in policy based methods where the policy would be directly parameterised by the weights of a neural network. Policy based methods are especially effective in stochastic and continuous domains and have been used heavily in the field of robotics [28]. Other variations of Deep Reinforcement Learning include Actor-Critic methods which uses neural networks to parameterise both policy parameters and Q values. This is typically done using separate neural networks to parameterise the policy and the Q values however there has been single neural network architectures which can parameterise both the policy and the neural network [29].

Following the initial success of Deep Reinforcement Learning, it has been used for different applications such as training agents to play complex board games which have extremely large state spaces including Chess, Shogi and Go at the level of world champions [30, 31]. The main significance of these agents is that a single agent can learn to play all three of the board games for the case of the agent known as AlphaZero which means that Deep Reinforcement Learning does have the potential to generalise to different situations to certain extent [31]. As well as mastering both Atari and board games, Deep Reinforcement Learning has been effective in training agents to learn motion dynamics such as the movement of a cheetah, a cartpole and a walker [32]. Although all these applications used Deep Reinforcement Learning, there were subtle differences between how agents were trained for different applications, where some applications used different neural network architectures as well as a combination of supervised and Reinforcement Learning. The Atari games involved only using Deep Reinforcement Learning with a Convolutional network whilst the motion dynamics was learnt using a recurrent neural network. In contrast, the Go agent was trained using supervised learning from expert human games in conjunction with Reinforcement Learning from games of self-play which was simulated using a search algorithm known as Monte-Carlo tree search. Using Monte-Carlo tree search was possible as the rules of the game of Go are known, and so it was possible to build a model of the transition dynamics which enables agents to plan moves ahead which can be data and computational efficient. The subfield of Reinforcement Learning concerned with building models of the transition dynamics is known as Model-based Reinforcement Learning [33]. There are several other subfields of Reinforcement Learning such as Imitation learning which involves human providing examples to show the correct action an agent should choose in particular states. Furthermore, techniques from different subfields can be used in conjunction such as in the study by Kendall et Al 2019, where the actions based by an autonomous vehicles are planned in simulation using a model of the environment, and incorrect actions are corrected by a human in the loop taking control of the vehicle [12].

2.3 Multi-Agent Reinforcement Learning

Multi-Agent Reinforcement Learning (MARL) is a subfield of Reinforcement Learning which focuses on training a group of agents which are meant to cooperate or compete with one another, and have been already been used for games such as Starcraft [34]. MARL has received a lot of attention in recent years due to the benefits of utilising a MAS to solve a complex task through the cooperation of agents performing a single task on a large scale, or by different agents performing different subtasks [35, 36]. Furthermore, the use of MARL is likely to increase in the future in the light of industrial trends moving towards a network of internet connected devices known as the 'Internet Of Things' [37].

Despite all the potential benefits of MARL, controlling multiple agents can be more difficult than a single agent case. In a single agent Reinforcement Learning scenario, the state of the environment changes solely as a result of the actions of the agent. In contrast to the single agent case, in a MARL scenario the environment can change as a result of the actions of all the agents present and therefore the optimal policy of a particular agent depends on both the environment as well as the policy of the other agents as shown in Fig. 2.3 [4]. Therefore, when multiple agents are interacting with the environment, Reinforcement Learning no longer satisfies a Markov Decision Process (MDP) and it is necessary to adapt the MDP process to take into account multiple agents and multiple states. The adapted MDP process is known as Markov games [38].

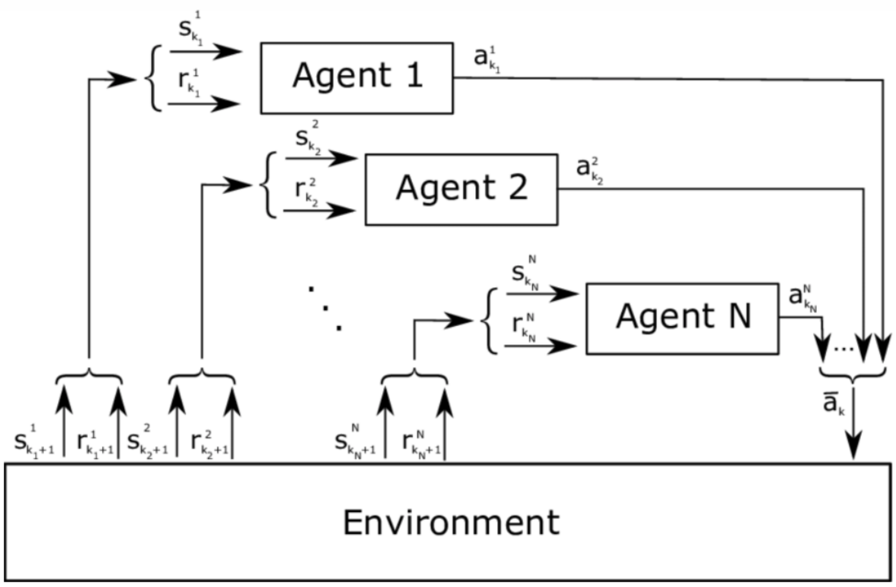


Figure 2.3: Interactions between agents and environment in a MARL setting [4]

Markov games involve the different agents in the environment taking actions simultaneously and receiving a reward based on their joint action. The interactions between the states and actions can be represented by the tuple (n, S, A, R, T) where n is a set of a finite number of agents agents, S is a set of states, A is a set of joint actions, $A = A_1$ x A_2 x ... x A_n, R is a reward function and T is a transition probability that gives the probability of transitions from state s to s' due to a joint action [39]. The objective of the n agents is to find a deterministic joint policy $\pi = (\pi_1, ..., \pi_n)$, where $\pi : S \to A$ and $\pi_i = S \to A_i$ is used to maximise the expected sum of the discounted common reward. Even though fully cooperative agents can be extremely useful, it is generally not the case in the real-world that agents are fully cooperative. Thus, it is also necessary to consider situations where agents are partially cooperative, competitive, adversarial or possibly a mix of cooperative and competitive such as in robot soccer [40]. The fundamental distinction between systems labeled as cooperative, competitive and adversarial is that for the cooperative case, agents goals is the maximization of a group utility. Competitive agents are solely focused on maximizing their own utility whilst adversarial agents are a type of competitive agents who are focused on maximising their own utility whilst minimising the utility of the other players. [36].

2.3.1 Competitive and Adversarial agents

An area where adversarial agents have been seen to be effective is at shaping the policy of the main agent known as the 'protagonist'. Typically, a protagonist used to control the motion of a robot is trained in simulation as this leads to faster training of the agent due to the use of parallel processing. However, when transferring the trained agent to real-life, the simulation to real transfer is ineffective due to modelling errors and differences between training and test scenarios. A solution to this problem can be to train the protagonist in the presence of an adversary. The adversary can destabilise the environment of the protagonist and encourage the main agent to develop a policy which is robust to external forces [41]. As well as making the protagonist robust to perturbation forces, adversaries have been used to train the protagonist to obtain a policy which can limit the risk of the protagonist. This is achieved by the protagonist choosing a policy also considering to minimise the variance of the expected future reward and low risk policies are important in the area of safe RL where risky actions can lead to potentially fatal consequences such as for the operation of an autonomous vehicle [42].

Another area where competitive and adversarial agents have been studied in is the context of games. Due to competition being prevalent in human activities such as economics, an understanding of how competitive agents behave is crucial for understanding how different entities behave. The interactions between the competitive agents can be seen in the context of games such as iterated prisoners dilemma or matching pennies ,which can be used to understand more relevant competitive behaviour such as bidding strategies in an auction [43, 44].

To examine the behaviour of the competitive and adversarial agents, it is first necessary to train the agents. In the case of cooperative MAS, it is possible to train a single centralised controller which controls the behaviour of the entire team of agents. However, in a competitive MAS, a approach known as concurrent learning is typically used which involves each agent learning individually to modify its behaviour and assuming that the other agents are part of the environment [3].

The main challenge for concurrent learning is that each learner is adapting its behaviour to other co-adapting agents in which it has no control over. Therefore, when there are multiple agents, as agents learn, each agent modifies their behaviour which can affect the learned behavior of the other agents which causes the non-stationarity in the training process. Concurrent learning has been successful in sequential environment where either the policy of the agent would be fixed or the agents take turns to act in the environment such as in the study done by Pan et Al 2019 where the protagonist agent would take m consecutive actions before an adversary would take n consecutive actions [42]. However, this non-stationarity can lead to issues in situations where agents takes actions simultaneously. The main approach to deal with this problem is opponent modelling, where an individual agent has a model or a way of taking into account the actions of the other agents in the environment.

Opponent modelling has been seen in the literature in several different ways. One variation of opponent modelling is to update the policy of a single agent in turn whilst keeping the policy of the other agents constant which allows the environment to be stationary momentarily [45]. The other main variation is for both agents to learn at the same time but make a model of how the other agents behave which is generally inferred from the actions of the other agent [46]. This could involve either making a single model of another agent or making several models for the behaviour of another agent. Having several ideas of the policy of the other agent can be beneficial as it lead to robustness in how the main agent acts [47]. However, having a single model can sometimes be more useful as it can lead to more sophisticated models such as in the work done by Foerster et Al 2018 where a model was made to take into account how the policy of the other agent will change at future time steps [44]. Whereas previous literature focused on building an internal model of the opponent whilst the agents learn, the work here will focus on obtaining an qualitative and quantitative understanding of how the policy changes in a simultaneous predator-prey game in order to to see if there is any pattern to how the policy of the agents changes, such as if the policy changes are cyclical in nature .

Various studies have been done on the area of predator-prey game where several different aspects have been examined such as the use of different Reinforcement Learning algorithms as well as different function approximators. An example of this would be the study done by Schrum et Al 2008, where a single prey agent was trained using 10 different Reinforcement Learning algorithms in order to compare how tabular algorithms and linear function approximators [48]. In comparison to this study which involved linear function approximators, the study by Egorov 2016 involved using a non linear function approximator (Convolutional neural network) to approximate the Q values of the agents by using a image representation of the MAS as the input state [45]. Other areas in which the Predator-Prey game has been studied is by the modelling of other agents. This has been seen in both the context of modelling the teammates of a particular agent as in the studies by Nagayuki et Al 2000 and Zhou et Al 2011 [46, 49] as well modelling an opponent of an agent [44].

In both the case of modelling a teammate and an opponent, building a model generally relies on observing the actions of the agent which is modelled and leads to an improved performance of the agent performing the modelling. The aspect of communication between agents has has also been studied in a Predator-Prey game. There are various different forms of communication and different information which can be communicated with one another. The study by Tan et Al 1993 examined Predator agents communicating information such as the position of the Prey agent, experience of an agent as well as the policy of an agent [50]. In contrast to this, the study by Partalas et Al 2007 involved Predator agents communicating with each other and voting regarding the joint action the group should take. [51].

3 Simulation Model

3.1 Predator-Prey Environment

An environment for a predator-prey game was developed in Python and is based on an existing environment found on Github for a multi-agent Predator Prey scenario [52]. For the purpose of this project, several factors were changed between the initial Github environment and the authors environment model such as the state space, action space and reward functions for the agent as well as additional modularities such as the presence of food at designated locations.

The environment is based on the format of an OpenAI gym environment where there are agents in an environment and the initial state of an agent would be based on an initial state distribution. The agents are able to take actions in the environment by using a step function which provides an agent with a reward and transition the agent to a new state. In typical OpenAI gym environments, only a single agent takes an action at a time. Whereas in this environment, the actions of both agents take place simultaneously and the resulting state and reward of each agent is dependent on the actions taken by both agents. An episode is a consists of the sequence of actions agents take before the game resets [2]. To conduct experiments efficiently, an episode would end automatically after a certain number of time steps. Timeout is necessary in practice, otherwise an episode can be arbitrarily long in time. These episodes terminate after 50 time steps and during the simulation, the agents cannot escape the environment. Any actions taken by the agent which would make it leave the boundaries of the environments results in the agent being unable to move or the agent moving to the other side of the gridworld (depending on the starting conditions).

The environment consisted of a discrete 7x7 (2D) gridworld occupied by two agents, a Predator and a Prey. The task of the predator was to follow and capture the prey, which occurred when the predator occupied the same grid cell as the prey. The aim of the prey agent was to flee from the predator and move to the food locations where it would be able to eat the food. The predator-prey game was designed to be a very simple model which would only focus on how the behaviour of the two agents present would affect one another. Therefore, the predator and prey were given full observability, the only dynamic elements present were the agents and the environment was designed to be stationary, which was why the location and amount of food present did not vary in time. Also, a more sophisticated model would take into account a hierarchy of different organisms in a food chain rather than having a simple top predator and bottom prey case. The gridworld can be seen in Fig. 3.1.

3.2 Agent state

The state of an agent at each time step is defined by a tuple of 5 features consisting of s = (Predator X position, Predator Y position, Prey X position, Prey Y position, individual agent stamina). The value of each feature was bounded between 0-1 to ensure that each feature had the same scale and equal importance when placed in a neural network model. To simplify the problem, the state space was discretised to enable only a fixed number of 2D positions for the Predator and Prey to occupy.

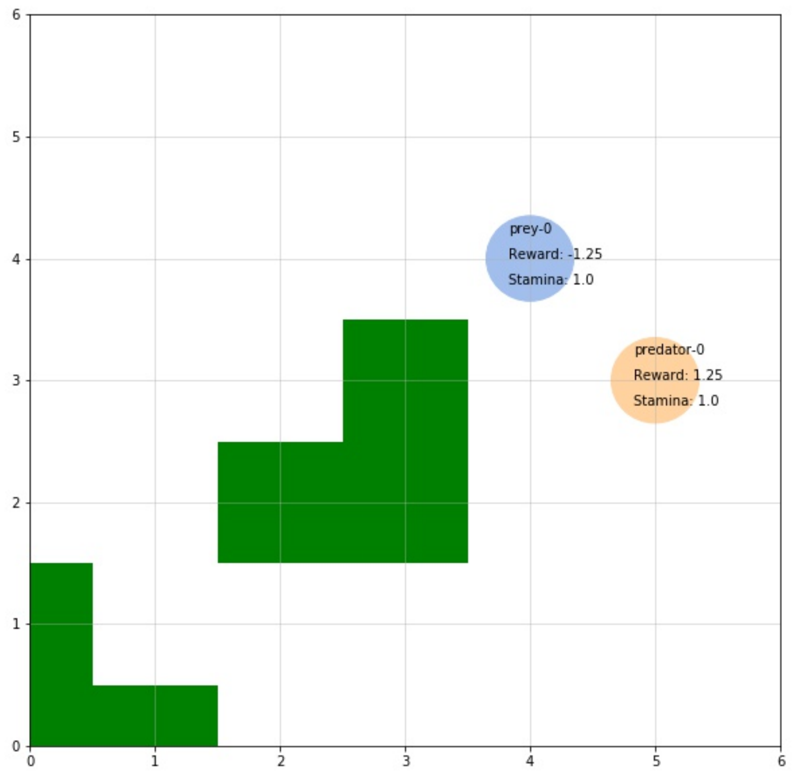


Figure 3.1: Predator (Orange) aims to to follow and capture the Prey (Blue) in the gridworld environment. The Prey aims to flee from the Predator and eat the food (Green)

3.3 Action space

Each agent had its own distinct set of actions. Both agents are able to move in the four cardinal directions or stay immobile. The action space of the Prey was to move one step in the four cardinal directions or to remain stationary which is represented by the set $A_{Prey} = \{Stationary, x+1, x-1, y+1, y-1\}$. The stationary action was used to allow an agent to replenish their stamina as well as allowing the Prey to eat food when the Prey was at the food locations. The action space of the Predator was to move one or two steps in the four cardinal directions or to stay stationary which enabled the Predator to replenish its stamina. The action space of the Predator was represented by the set $A_{Predator} = \{Stationary, x+1, x-1, y+1, y-1, x+2, x-2, y+2, y-2\}$. Both agents start with 1.0 units of stamina and the stationary action replenishes 0.05 units of stamina for an agent. The actions of moving one step in a particular direction uses 0.01 units of stamina for the Prey and 0.03 units of stamina for the Predator whilst the action of moving two steps in a particular direction uses 0.12 units of stamina for the Predator. The action space of the agents was designed to simulate a real pursuit predation environment where the Predator is able to move faster than a Prey, but likely to run out of stamina earlier [53].

3.4 Reward function

The aim of the learning is to have the predator and prey behave in way that the predator aims to catch the prey whilst the prey would try to escape but would also take the risk of staying stationary to eat the food when the predator was sufficiently far away. To enable the agents to learn these behaviours, it was necessary to carefully design a reward function for each of the agents, which is a process known as reward shaping [54]. To encourage the predator to capture the prey, one possible reward function which could have been used was to provide the predator with a reward solely when the predator occupying the same square, also referred to as colliding, with the prey. However, due to the size of the gridworld and the fact that the prey would be intending to maximising its distance away from the predator, this would lead to what is known as a 'sparse reward' situation where rewards would not occur frequently such as a reward only being obtained once during an episode [55]. This would result in the Predator taking a long time to learn which actions gives it a reward and as a result, it would take a long time for the predator to adopt the desired policy of colliding with the prey. Therefore, one way to alleviate the sparse reward problem would be to change the reward function into what is known as a 'dense' reward function where the predator can obtain a reward at every time step [2]. One example of a dense reward function could be rewarding the Predator based on its proximity to the Prey based on a Manhattan distance, which is a distance of two points measured along axes at right angles, which corresponds to the number of grid cells between the two agents [56]. This type of reward function is shown by Eqn. 3.1:

$$r_{predator}(X_{Predator}, X_{Prey}) = e^{-c\|X_{Predator}-X_{Prey}\|^2} \tag{3.1}$$

Where $X_{Predator} = (x_{Predator}, y_{Predator})$ and $X_{Prey} = (x_{Prey}, y_{Prey})$ are the positions of the predator and prey respectively and c is a constant.

However, a potential problem with using this type of reward function is that it instigates the Predator obtain a strategy where the Predator remains in close proximity of the Prey by using single step movements, but the Predator does not try to colide the Prey. The approach taken to solve this problem was to use a dense reward function based on the proximity to the Prey but also provide an additional reward when the Predator would collid with the Prey. This additional reward for colliding with the Prey encouraged the Predator to move two steps in pursuit of colliding with the Prey rather than simply stalking the Prey. The reward function of the Predator can be seen in Fig.3.2. For the case of the Prey, to train the Prey to move away from the Predator, the Prey was penalised (negative reward) based on its proximity to the Predator in the same way that the Predator was rewarded. This can be seen in Fig. 3.3. However, to encourage the Prey to eat the food when the Predator was far away, the Prey was given a positive reward of + 2 for every timestep it ate the food. The reward for eating the food was chosen to be +2 as a value which was too low would mean that the Prey would always aim to maximise its distance from the Predator and so the Prey would never remain stationary to eat the food. If the reward for eating the food was too high, then the Prey would adopt a policy where it remains at the food location even when the Predator is in close proximity. Through trial and error testing, a value of +2 was high enough for the Prey to eat the food when the Predator was sufficiently far away but also low enough such that the Prey would move away from the food to flee from a Predator in close proximity.

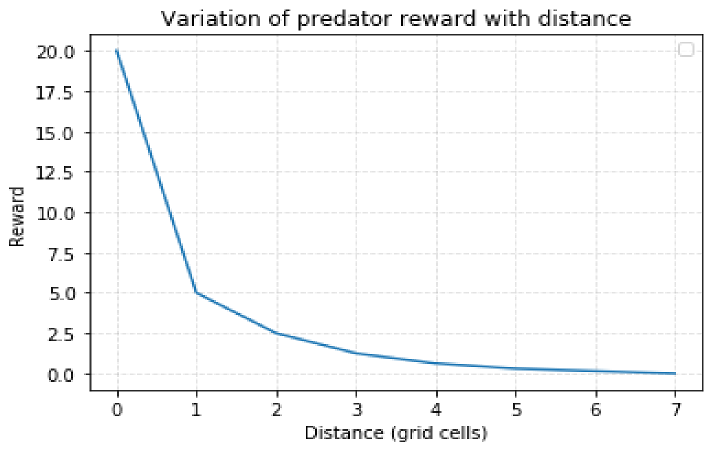


Figure 3.2: Variation of the predator reward as a function of Manhattan distance

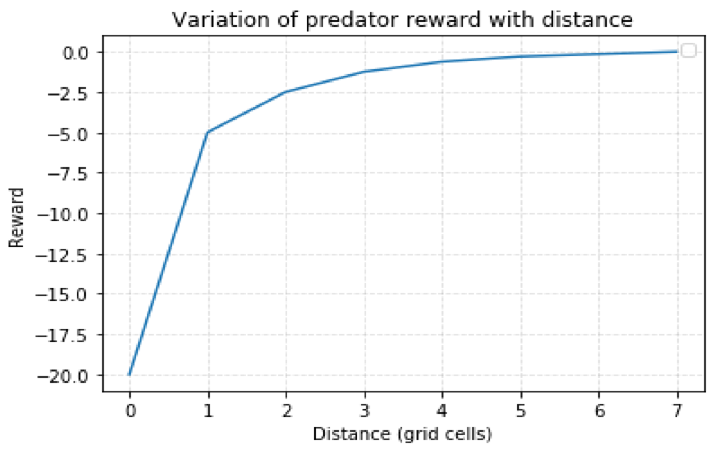


Figure 3.3: Variation of the prey reward as a function of Manhattan distance

3.5 Training algorithm

The algorithm that was used to train the agents was the Deep Q-Network (DQN) algorithm. DQN involves using a multi-layer neural network with weights θ as a non-linear function approximator to estimate the optimal action value function Q(s,a;θ) $\approx$ Q*(s,a).

Reinforcement Learning has been shown to be unstable when a non-linear function approximator such as a neural network is used to represent Q-values. This instability has been due to correlations in the input states used in the training of the network, as well as correlations between the Q-values predicted by the neural network and the target Q-values given by Eqn: 3.2

$$y = r + \gamma max_{a'} Q(s', a'; \theta) \tag{3.2}$$

The DQN algorithm solves these issues by using 2 key ideas. The first idea is to use experience replay, which involves storing an agents experience $e_t = (s_t, a_t, r_t, s_{t+1})$ at each time step in a dataset $D_t = (e_1, e_2, ..., e_t)$. During learning, Q-learning is applied on minibatches of experience drawn randomly from the dataset and this removes the correlation in the input states.

The second idea is to use an additioal neural network known as the target neural network where the value of θ_{target} is only updated with the Q-network parameters at every C steps [25]. A Q-network is then trained on minibatches of experience sampled from the dataset D_t in order to minimise a mean-squared loss function given by Eqn. 3.3 [24]:

$$\mathcal{L}(\theta_t) = E_{s,a,r,s'}\left[y_t - Q(s,a,\theta_t))^2\right] \tag{3.3}$$

where y_t is the target Q-value from the target network and it is given by Eqn. 3.4

$$y_t = E\left[r + \gamma max_{a'} Q^*(s', a'; \theta_{target}|s,a)\right] \tag{3.4}$$

DQN is an off-policy algorithm as it learns choose actions according to a greedy strategy $a = max_a Q(s,a,\theta)$ whilst following a behavioural distribution which ensures adequate exploration of the state space. The behavioural strategy which was used was an ϵ-greedy strategy which follows the greedy strategy with probability 1- ϵ and selects a random action with probability ϵ.

Each agent would be trained using separate neural networks. The initial neural network from the Github used 2 hidden layers with 512 neurons however the number of neurons was changed to 64 in this project [52]. This was due to 2 reasons:

- To reduce computational complexity
- To reduce overfitting

The Github project used a large multi-agent system where there were several predator and several prey agents, as well as a larger number of features to represent the state of each agent. Therefore, it was beneficial to use a larger neural network to train the agents in that case. However, for this project where there was only a single predator and prey with a state representation consisting of 5 features, a smaller neural network with 64 neurons in each layer was effectively able to train the agents.

After each episode, both agents were trained using gradient descent on a batch of 200 data points sampled from the experience replay buffer and the target network was updated after 20 episodes each time and the learning rate of the network was 0.01. The initial ϵ value was chosen to be 0.9 and would decay at a rate of 0.998 per episode until it reached a threshold value of 0.1. The experience replay buffer would store data from the different episodes. The buffer had a fixed size and would store only the last N experience tuples and remove older experience as new experience would come in. It has been seen generally that a default buffer size of 10^6 has been effective for the case of single-agent RL. However, the non-stationarity in a MAS can mean that the dynamics that generated the data in the agents replay memory no longer represents the current dynamics in which the agent is learning. One approach to avoid this problem was to limit the use of experience replay to short, recent buffers. It was necessary to test several different buffer sizes as smaller buffer sizes are more effective in terms of learning speed, however the stability decreases. This was due to a trade-off where smaller recent buffer leads to data which better represents the current dynamics but are higher correlated, whereas using a larger replay buffer has data which tends to be uncorrelated but more outdated. In the case of the predator-prey game in this project, a replay buffer size of 7500 data point was used for both agents as it was effective in allowing both agents to learn effectively.

4 Simulation Experiments

Each episode involved the Predator and Prey starting in random locations in the environment and would last for 50 time steps. After n episodes of training where the agents would behave with a ϵ-greedy policy, the current greedy policy of the agents would be tested for 100 episodes. The values of n chosen were 10, 100, 500 and 1000. During these 100 test episodes, several quantities of interest were calculated for each agent and these values would be plotted for the last episode of the test. The X axis in each of the plots represents the number of training episodes. The quantities calculated were:

- Return (discounted sum of rewards)
- Number of collisions
- Similarity
- Similarity centre

Recall from Section 2.1 that the return of an agent is the discounted sum of rewards during an episode and can be calculated using Eqn. 2.3. In the simulation, the γ value of for each agent was 0.9. The graph of the returns were nearly symmetrical as the prey was penalised based on the reward the predator obtains however there was slight asymmetry due to the additional reward the prey would obtain from eating the food.

The plot representing the number of collisions was based on the mean number of collisions that occurred at each time step. For the case where there are only two agents present, the maximum number of collisions at each time step can only be one, and the minimum number of collisions at each time step is zero. Therefore, each value on the plot would be between 0 and 1 for a single Predator and a single Prey scenario.

During each of the episodes, it was necessary to obtain a measure of how the policy of the agent was changing in order to understand how the actions of other agents was affecting the learning of a particular agent. One approach to evaluate this was by fixing the position of one of the agents and looking at whether the actions performed by the other agent was the same before and after a single training period of n episodes. Due to computational complexity, the similarity was measured at two different locations. The first fixed location was the location (0,0) which was an area of interest for two reasons. Firstly, it was a corner location which the Prey would go towards in order to maximise its distance from the Predator. Secondly, it was one of the locations of the food. Due to these reasons, this location was the location where the Prey would move towards most frequently and as a result, a location where the predator would move to. As this was the location where both agents spend most of their time, most of the learning of the agents would occur when the position of the agents is near the (0,0) position and so the actions taken in these states were likely to vary the most. The second fixed location for the agents was chosen to be at the centre of the gridworld at (3,3) position. This position was chosen as this was a position of the food and a position which would likely have an effect on the action taken by an agent at any position of the gridworld. The actions for an agent was seen for every possible location in the grid as well as 10 different stamina values of the agent. The similarity measure can be represented by Eqn. 4.1

$$Similarity = \sum_{i=0}^{490} Sim_i^k \tag{4.1}$$

$$Sim_i^k = \begin{cases} 1, & \text{if } \pi^k(s_i) = \pi^{k-1}(s_i) \\ 0, & \text{otherwise} \end{cases} \tag{4.2}$$

Where $\pi^k(s_i)$ represents the action taken by the kth policy of an agent when in state i.

The Similarity value based on the actions that an agent would take when the other agent is at the first fixed location is referred to as 'Similarity'. Whereas, the Similarity value based on the actions that an agent would take when the other agent is at the second fixed location is referred to as 'Similarity centre'.

Several experiment were conducted to examine how the policy of an agent would change during the learning process. In the first two experiments, the Predator would be learning after each episode whilst the policy of the Prey remained constat. During the last experiment, both the Predator and Prey would be learning after each episode. These experiment were:

- Experiment 1: Intelligent Predator vs stationary Prey
- Experiment 2: Intelligent Predator vs Prey moving in a particular direction
- Experiment 3: Intelligent Predator vs intelligent Prey

The first 2 experiments were conducted to see how the Predator learns in a case where they was no feedback from the Prey agent as the Prey would have a fixed policy. These experiments were performed to enable comparison between the learning of the Predator when the Prey policy fixed and when the Prey policy is changing. In experiments 1 and 2 where the Prey policy was fixed, it was hypothesised that the policy of the Predator would converge at some point after learning that the policy of the Prey was fixed.

4.1 Intelligent Predator vs stationary Prey

For the case of the intelligent Predator against the stationary Prey, the Predator was quickly able to find out the Prey was stationary which lead to the predator quickly moving to the location of the prey in the most efficient and direct manner which was the two step movements. This can be seen as the return of the Predator in Fig. 4.1 quickly maximised which minimized the return of the Prey. Furthermore, the number of collisions shifted to a value close to 1 which showed that the Predator occupied the same grid location as the Prey for every time step besides the initial time steps in which the Predator had to move to the location of the Prey.

From looking at the plots of the similarity in Fig. 4.1, it can be seen that the similarity of the Predator was oscillating near a value of 400. This reveals that the policy of the Predator was always changing even in a situation where the policy of the Prey was constant and there was only 10 episodes of training between the two consecutive policies in which the similarity was measured.

Furthermore, by looking at Fig 4.1 - 4.4, it can be seen that as the number of training episodes between testing consecutive policies increased, the similarity and similarity centre values decreased from from 400, 350, 250 to 200. This suggests that the policy of the Predator was always changing but changed in gradual manner. This is a peculiar result as the Prey was stationary in this test and therefore the Predator similarity should have converged in the same way as the return and number of collisions converged. However, since the Predator was obtaining the same return using different actions, it could suggest that there are different actions which enable it to obtain the same reward, such as actions which lead to the same change in the Manhattan distance of the agents.

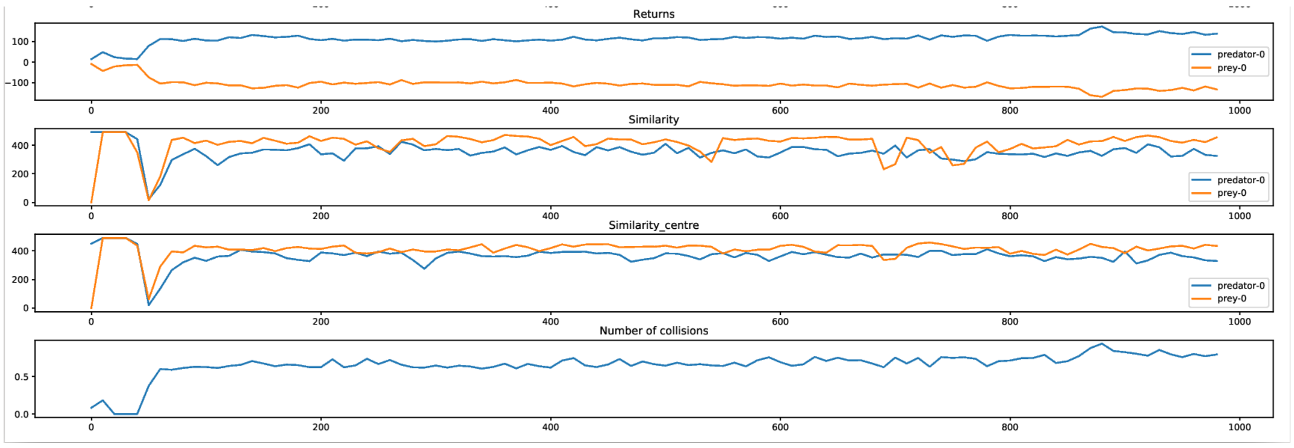


Figure 4.1: Data from experiment 1 where consecutive policies are tested every 10 episodes

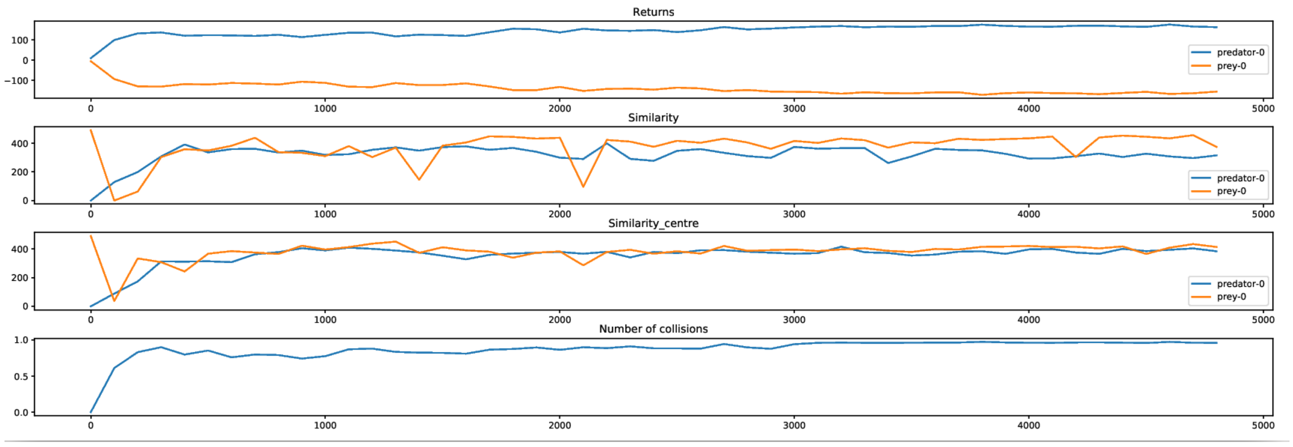


Figure 4.2: Data from experiment 1 where consecutive policies are tested every 100 episodes

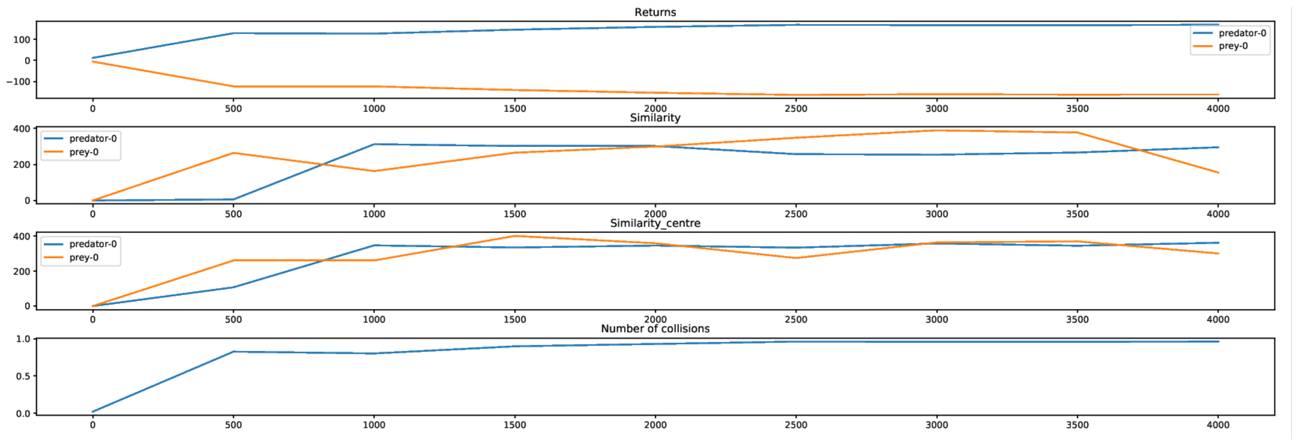


Figure 4.3: Data from experiment 1 where consecutive policies are tested every 500 episodes

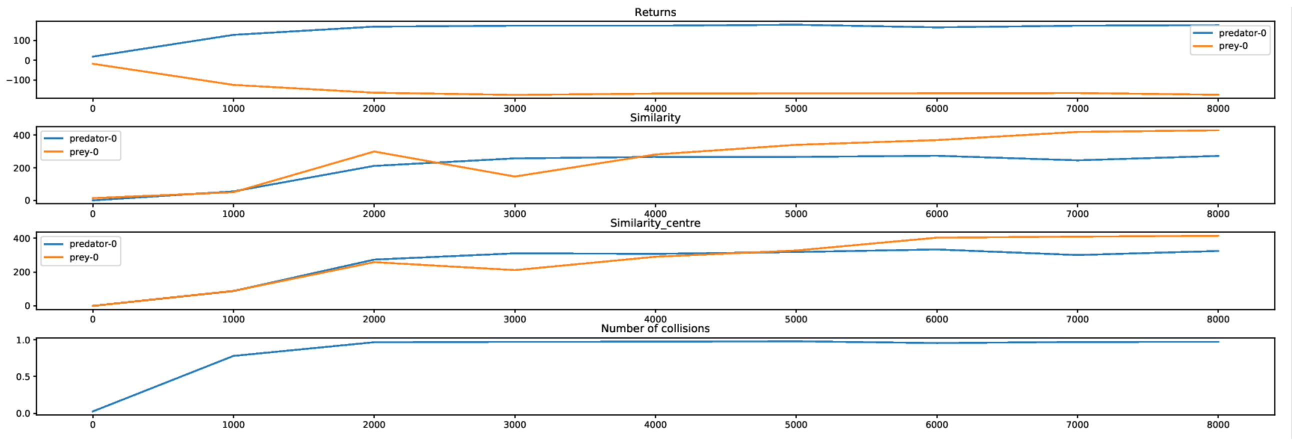


Figure 4.4: Data from experiment 1 where consecutive policies are tested every 1000 episodes

4.2 Intelligent Predator vs Prey moving in a particular direction

For this test, it was necessary to adopt the convention that movements out of the borders of the gridworld lead to an agent appearing on the other side of the gridworld, otherwise the Prey agent would end up being stationary when it reached a border. Similar to the test 1, the Predator was quickly able to figure out that the Prey had a distinct pattern in its movement. Due to Prey moving in a fixed direction being a more complex movement pattern than the case where the Prey was completely stationary , the Predator did take a longer time to figure out the movement pattern of the Prey which is shown by the plateauing of the return occurring at a later episode compared to the stationary case. This can be observed by comparing when the returns reach a maximum in Fig. 4.1 and Fig. 4.5. The number of collisions was slightly lower than the stationary test which was due to the limitation in the stamina of the Predator. The limitation in the stamina lead to the Predator resting at certain points and prevented it from keeping up with the Prey for all time steps of an episode. However, there does not seem to be any noticeable changes in the magnitude of the return of the Predator. This was due to the Predator being able to keep up with the Prey during the earlier times steps where the contribution to the reward was more significant due to the discounting at later time steps of the episode.

After the predator has learnt the correct strategy which was shown by the maximum in the return, the Predator displays the same pattern in the similarity measure where it oscillates between a certain number depending on the number of episodes between testing as shown in Fig. 4.6 - 4.8. Also, it can be noted that even though it does take longer for the predator to learn the optimal strategy in this scenario, the policy still does seem to change at the same rate as the stationary case as shown by the near identical values in which the similarity measure oscillates in the stationary (Fig. 4.1 - 4.4) and moving experiment (Fig. 4.5 - 4.8).

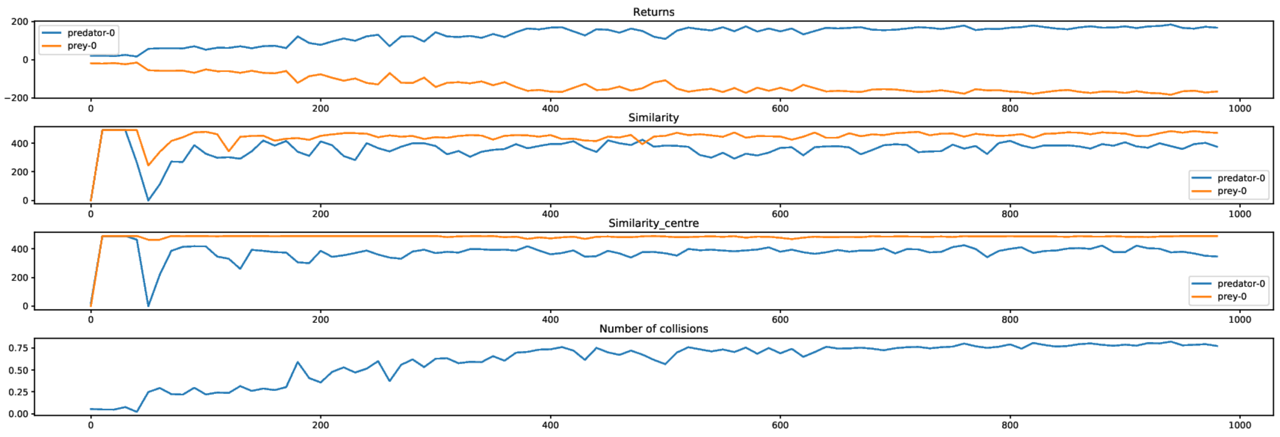


Figure 4.5: Data from experiment 2 where consecutive policies are tested every 10 episodes

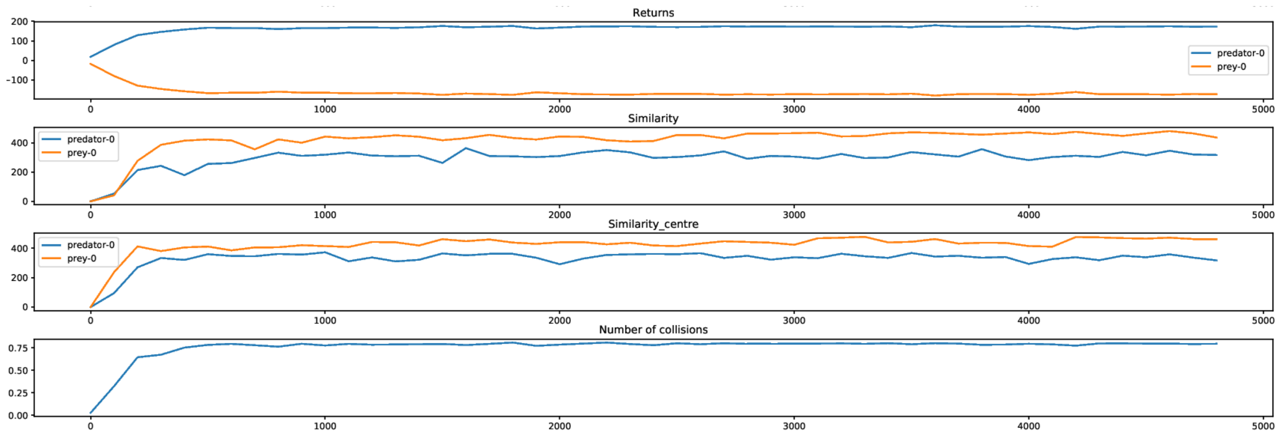


Figure 4.6: Data from experiment 2 where consecutive policies are tested every 100 episodes

4.3 Intelligent predator vs intelligent prey

For the case where both agents were learning simultaneously, two observations can be made by comparing Fig. 4.9 with 4.3 and 4.7 regarding the reward obtained. Firstly, the amount of reward obtained from the Predator was significantly lower. Secondly, there was significant oscillation in the returns of each of the agents. This behaviour was expected as the Prey was now adjusting its policy with respect to the presence of the Predator. This causes the Prey to successfully flee from the Predator which lead to the Predator being unable to maintain in close proximity to the Prey which decreased the overall return of the Predator. Furthermore, due to the Predator and Prey adapting

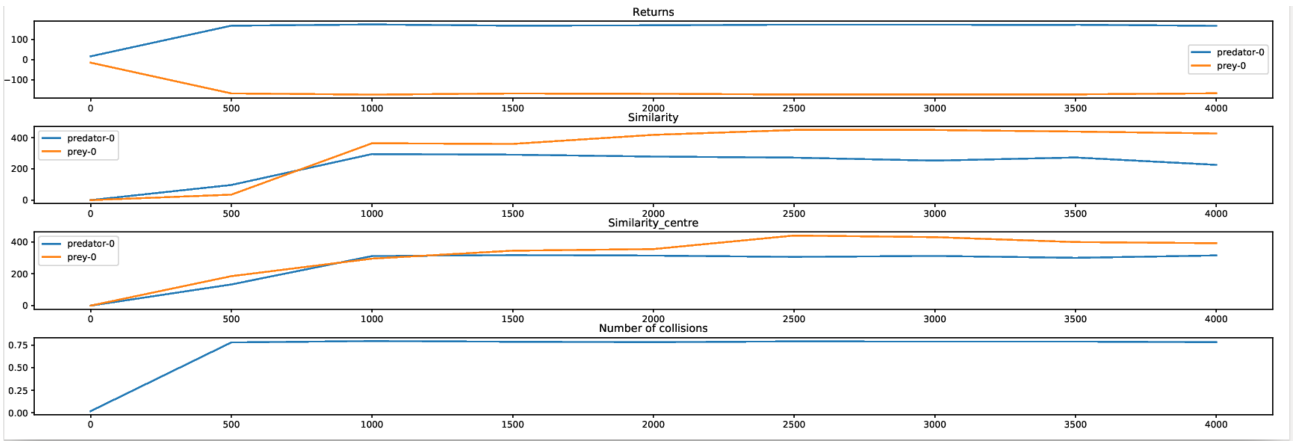


Figure 4.7: Data from experiment 2 where consecutive policies are tested every 500 episodes

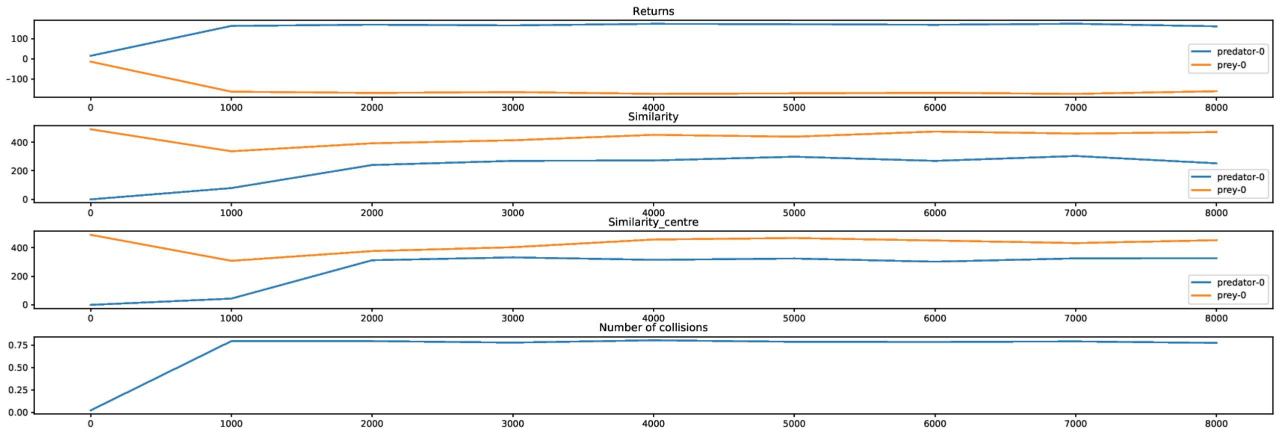


Figure 4.8: Data from experiment 2 where consecutive policies are tested every 1000 episodes

to one another, this lead to the oscillation in the returns of each of the agents.

The degree of oscillation in the returns of the agents may suggest that the agents were consistently well matched to one another. This could mean that the agents are not changing their policy significantly or that both agents are learning and adapting to one another so that the policy of both agents are equally matched. However, the small amount of oscillation could be due to the return being representative of the performance of the agents at earlier time steps of an episode. Therefore, the return was likely not a good representation of the performance of the agents over the entire episode especially since the predator has an advantage during the earlier time steps due to having a high stamina and being able to move faster than the prey. Furthermore, the prey typically only utilises the food at a later time steps of the episode where it can create some distance from the predator due to the stamina of the predator becoming low. One possible way to get a idea of how the policy changed would be to examine the returns of the agent for the case where the discount factor is equal to 1.

A better indication of how the performance of the agents match up to one another could be the large variation in the number of collisions as the number of collisions represented how the agents behaved during a whole episode. From the minima in the number of collisions at episode 500 and 1500 in Fig. 4.9, it reveals that even if the return of the Predator is high, the number of collisions can be low as the Prey may remain in close proximity of the Predator but adapt to successfully avoid colliding.

From Fig. 4.12, it can be seen that in the earlier episodes between 1000 - 3000 there was higher amount of oscillation in the similarity, and the similarity values can be seen to go below 200 for the Predator and the Prey agents. This could show that the policy of the agents were changing more significantly at the earlier episodes and then slows to a constant rate later on. This is further supported by Fig. 4.10 , where the similarity is changing significantly more than Fig. 4.1 and 4.5 (plot of similarity for the stationary case and moving case).This suggests that the policy of an agent changes faster in an adversarial situation compared to the case where the other agent is static or behaving in a repetitive manner.

From Fig. 4.11, there did not seem to be a clear pattern in the similarity as it was oscillating significantly for both Predator and Prey. Also, it can be seen that there are points in the plot where the similarity of both the prey and predator are fluctuating together showing that the agents are adapting to one another which is to be expected. Also as predicted, the oscillation of the similarity plot was larger than the similarity centre plot due to the preference of the prey moving to the corner (0,0) which allows the agents to more regularly update how to act when the other agent was in the corner.

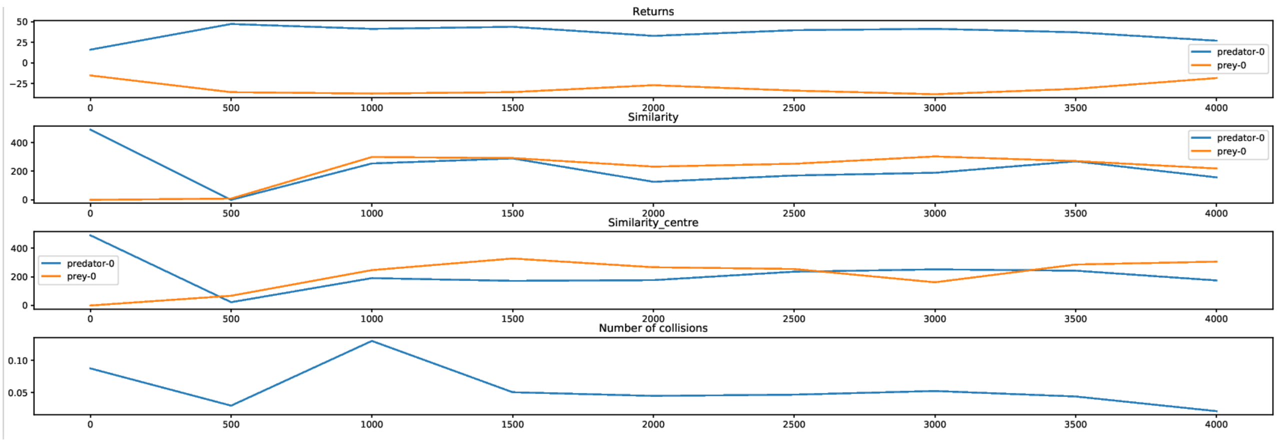


Figure 4.9: Data from experiment 3 where consecutive policies are tested every 500 episodes

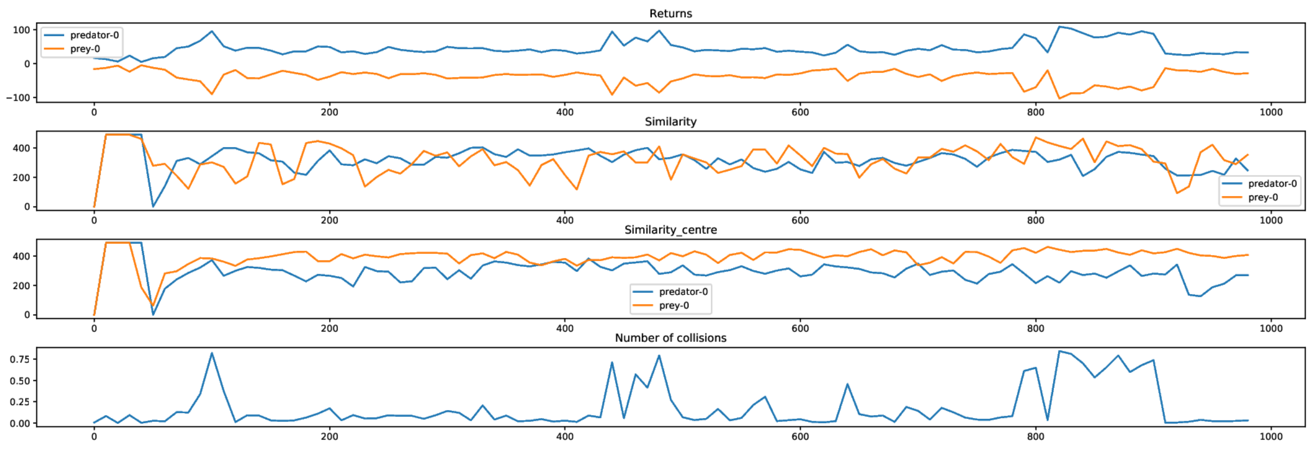


Figure 4.10: Data from experiment 3 where consecutive policies are tested every 10 episodes

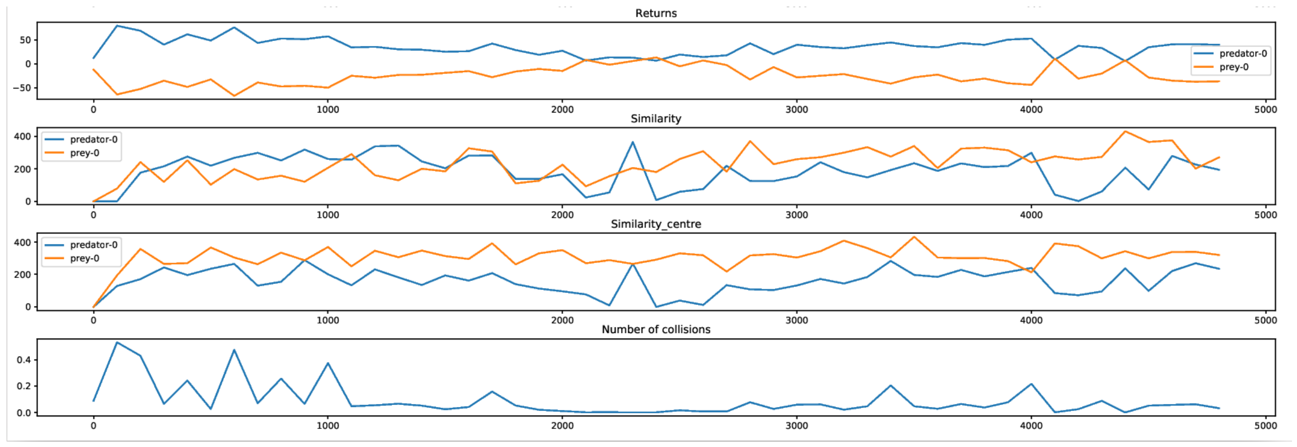


Figure 4.11: Data from experiment 3 where consecutive policies are tested every 100 episodes

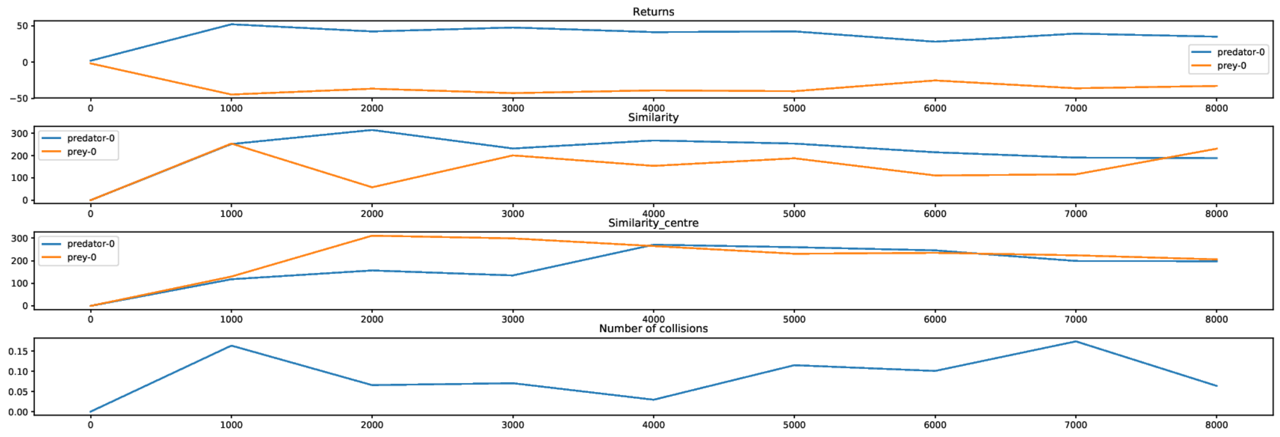


Figure 4.12: Data from experiment 3 where consecutive policies are tested every 1000 episodes

5 Evaluation

This section is related to the importance of explainability of machine learning models and describes the approach of using the rules of a Decision tree proxy model to qualitatively explain the policy of the Predator and Prey agents. Furthermore, a quantitative analysis of the different Decision tree proxy model is performed through the process of Spectral clustering.

5.1 Importance of explainability

Despite widespread adoption in academia and industry, many machine learning models remain black boxes. A machine learning model is a mathematical representation of a real-world process which is trained on training data with known labels and then is used to make predictions based on an unlabelled input. The concept of a 'black box' in machine learning is AI which offers little insight to how an specific output is reached. In contrast to this, Explainable AI (XAI) refers to methods and techniques in the application of AI technology such that the results can be understood by human experts [57].

XAI aims to achieve this by improving the explainabilty and interpretability of AI. Interpretability is related to the extent to which an outcome can be predicted based on a change to the input, which does not explicitly require an understanding of the operations which are occurring in the model. Whereas, Explainability is the extent to which the internal mechanics of a machine learning model can be explained in human terms and provides an understanding of why a change in the input instigates a change in the output [58].

As domains such as healthcare where questions of accountability and transparency are particularly important, deployment of AI and deep learning systems depends largely on the trust of a human in a model. Trust in a model can be difficult to obtain as even if a model has a high accuracy for a specified task, the model can still be untrustworthy. This is apparent in situations where there could be bias in a model such as in the case of image classification where a classification decision can be based on the background of an image which is an artifact of the training data used to train the classifier. This has been seen for the case of distinguishing between a wolf or a husky based solely on a snowy background [59]. Therefore, understanding the reasons behind a prediction is important if one plans to take an action based on a prediction. As a result, explainability is a required goal before using a machine learning model in a high-risk situation. Furthermore, there are also several other benefits of providing explainations such aiding in identifying problems in the training data and enabling users to have more feedback on the tuning of a machine learning model if the explanations are illogical. Also, it can ensure that machine learning models work effectively if the explanations are intelligible. However providing explanations can be difficult, especially for the case of a deep learning system. Modern deep learning systems use millions of parameters to transform an input feature space into a different representation space and then search for a decision boundary to separate the classes in the representation space. This enables neural networks to solve complex non-linear problems, however due to the number of parameters required, neural network are generally incomprehensible but using simpler models would lead to a trade-off in performance for explainability which is not desired.

Explanations of the operation of deep neural networks have focused on either explaining the processing of the data by the network, or by analysing the higher level features present in the network. This can be done via 3 main approaches [58]:

- Generated explanations
- Creating a saliency map
- Creating a proxy model

The optimal approach to explain how a neural network is performing a task would be for the neural network to generate their own explanations. This would involve the neural network providing textual or visual information which is meaningful to the user in order to provide a qualitative understanding of what the neural network is doing. This can be achieved by designing a deep generative models to generate their own human-interpretable explanation as part of the training of the model.

Saliency maps can be used to highlight a small portion of a computation which is the most relevant. In the context of image classification, saliency mapping involves repeatedly testing an image with a portion of the image occluded. This will enable a map to be created which displays the part of the data that has an influence on the output of the neural network.

A proxy model is a model which behaves similarly to the original model but behaves in a way that is easier to explain and interpret. One commonly used technique to build a proxy model of a neural network is locally interpretable model-agnostic explanations (LIME) where the behaviour of a black model is explained by probing the local area of an specific input. The outputs of the neural network around the local region are then used to construct a linear model that serves as a simplified proxy for the full model in the neighbourhood of the input [59]. Explainability is important in this project as the behaviour of a MAS can be more complex than the behaviour of a single agent case due to emergent behaviour obtained by the actions of the different agents in the environment. Therefore, it is necessary to understand how the agents in a MAS learn as time passes in order to prevent any unpredictable and potentially harmful behaviour being learnt.

5.2 Decision Trees

Another type of proxy model which can be used to explain the behaviour of neural network are Decision trees. Decision trees are non-linear algorithms which create as series of if-then production rules which can be used for classification and regression tasks.

A Decision tree is a flowchart like structure made from the recursive partitioning of the training data into different classes. A Decision tree is a directed tree that consists of internal nodes, leaf nodes and branches. The top most node in the tree (known as the tree node) has no incoming branches, the leaf nodes has no outgoing branches and the internal nodes are nodes between the tree node and the leaf nodes. Each internal node represents a test, and each outgoing branch corresponds to a possible outcome to the test. The test can involve comparing a numerical attribute against a threshold value or comparing whether a sample fits within a certain category. An example of a simple Decision tree can be seen in Fig. 5.1 .

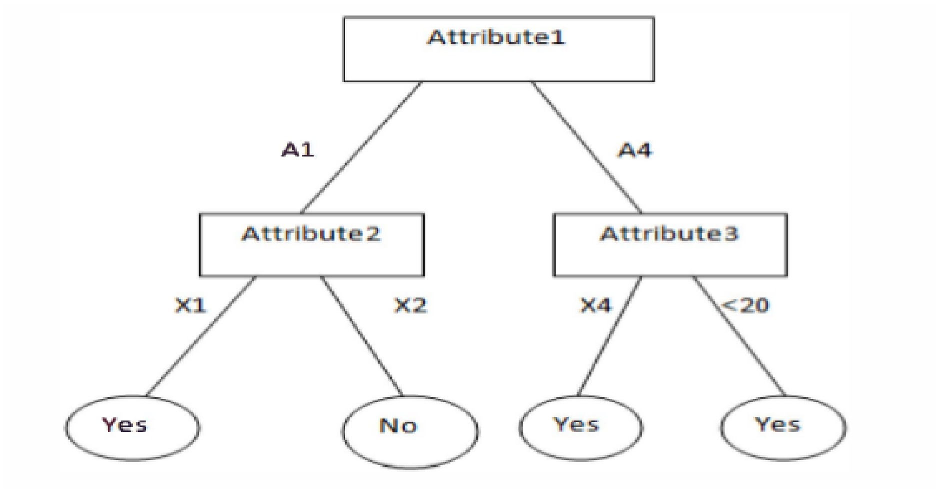


Figure 5.1: Simple example of a Decision tree [5]

The attribute chosen at each internal node depends on the discriminative power of each attribute that best split the data using the concept of Entropy or Gini index at a node t, which is given by Eqn. 5.1 and 5.2 respectively [60]:

$$\text{Entropy(t)} = -\sum_j [p(j|t)logp(j|t)] \tag{5.1}$$

$$\text{Gini(t)} = 1 - \sum_j [p(j|t)]^2 \tag{5.2}$$

Decision trees are able to perform splitting in one of two ways. One method is by using a single attribute at each node, which is known as univariate splitting. The second method involves using a linear combination of attributes for the splitting criteria and this is known as multivariate splitting [5].

A Decision tree makes a model of the decision function by recursively dividing the training data until a leaf node comprises completely or predominantly from a single class. In order to classify an unlabelled sample, a path is traced from the root to a leaf node which holds the class prediction for that sample. Decision trees can improve explainability as the if-then production rules generated by a Decision tree and the path traced from the root to a leaf node can be analysed by a user to get an understanding of the sequential process taken to make a prediction. However, building a tree can be memory intensive and the tree that is built can be quite complex which would make interpreting the path traced from the root to a leaf difficult to interpret. Therefore, an approach known as pruning can be used to obtain trees of an optimum size. Pruning is effective in reducing the size of the tree in such as way that the tree becomes less complex and simpler to interpret. Furthermore, pruning a tree can potentially improve the generalisation of the tree to make it more accurate on unseen samples. There are two main types of pruning:

- Pre-pruning (also known as stop-splitting)
- Post-pruning

Pre-pruning involves placing constraints on the tree before it has been built by fixing parameters such as the maximum depth of the tree and minimum number of samples within a given leaf node. Post-pruning involves first assembling a complete tree in which increasing the number of leaf nodes and the depth of the tree will no further enhance the precision on the training set, and then systematically removing subtrees which are not contributing to the generalisability of the tree.

For this project, Decision trees were used as proxy models for the decisions made by the neural network of the Predator and the neural network of the Prey at different episodes in the simulation which corresponded to the agents having different policies. A Decision tree was made for each of the different policies by saving the data points (state of the agent and the action taken) of a particular policy in a replay buffer of size 7500. Then, from these 7500 data points, 1000 data points would be sampled and fed into the Sci-kit Learn Decision tree algorithm which uses a univaritate splitting and Gini impurtiy splitting criteria to build a Decision tree [61]. To ensure that the Decision trees could be interpreted from its if-then rules, pre-pruning was used to ensure that a Decision tree would not become too large or complex. All the Decision trees were pre-pruned to have a maximum depth of 6 and a minimum number of 7 data points required to split an internal node. All the different Decision trees as well as the data used to make the Decision trees can be seen on the Authors Github [62].

5.2.1 Prey Decision tree comparison

Although the state of the Prey was represented by 5 features in the simulation, the stamina was redundant for the Prey. The stamina was redundant for the Prey as the number of time steps in an episode were short enough such that the actions of the Prey were unhindered by the stamina as the stamina of the Prey always exceeded the amount required to perform an action. As a result of this, the stamina feature was removed as an input of the Prey Decision tree classifiers in order to prevent building Decision trees which would overfit on a redundant feature. Therefore the Decision trees representing the different policies of the Prey were made using 4 features.

From the Decision trees made from the different policies of the Prey, there did not seem to be any consistency between the most important feature as there was no feature which was always present at the top node of all or a majority of the Decision trees. This could mean that all the features input into the Decision tree classifiers had the same importance and that only the relative difference between the position of the agents decided the action taken by Prey. However, instead of believing that all features have the same importance to a policy due to the lack of consistency between the different policies, an alternative view could be that different features have different importance in relation to a particular policy, and that the each of the policies are sufficiently different from one another.

Despite the differences between the different Decision trees, there were still some similarities present between them. The main aspect that remained the same across all the Decision trees was that a splitting value of 0.071 was present for the features of the Predator and Prey positions. This shows that the Decision trees were agreeing with the intuition that the Prey agent looks at how close it is or how close the Predator is from the location of the food, due to the Prey's tendency to move towards the food in the corner in order to maximise its reward.

Another observation is that even though there was food in another location closer to the centre of the gridworld, the focus of the Prey was to move towards the food in the corner of the map as shown by the splitting value of 0.071 being higher up in the Decision trees showing it had more importance than other splitting values. This also agrees with the intuition that moving towards the corner enables the prey to maximise its distance from the predator which was the top priority for the Prey and was more important than moving towards the centre of the map to obtain food. Therefore, from looking at the if-then rules of the Decision trees for the different policies of the prey, it seems that a majority of the Decisions trees agree that the Prey was moving in a logical manner in such a way to maximise its distance from the Predator whilst simultaneously maximizing the food that it eats. The other logical ways in which the Prey agent was behaving can be seen in the annotated Prey Decision tree in the Github [62].

To get an idea of how accurate the Decision trees were as a proxy model to the neural networks which characterised a particular policy, the classification accuracy of each Decision tree was checked by comparing the outputs of a Decision tree against the output of the neural network corresponding to the same policy. This was performed by training each Decision tree on 900 data points from the replay buffer of a particular policy and then testing on another 100 data points from the same policy. The Decision tree classifiers were tested using k-fold cross validation. K-fold cross validation enables all the data to be used for the training of the classifier as well as in testing, and it is a widely used technique to estimate the generalisation error of a classifier. K-fold cross validation consists of [63]:

- Dividing the sample data into k equal sized subsets. Each subset is referred to as a fold. Let the folds be named as f_1,f_2,. . . .,f_k.
- For i = 1:k, perform the classification using the fold f_i as a test set and keep the remaining k-1 folds as a training set.

The overall classification accuracy was calculated by the average of the k predicted accuracies. The results of the Prey Decision trees when using 10 fold cross validation can be seen in Table. 5.1.
From the results, it can be seen that the Prey Decision trees were effective proxy models for the neural networks as the mean classification accuracy across all the Decision trees were 76 %. However, there was a large amount of variation in the 10 different tests of the cross validation which suggests that the Decision trees were not stable as they would change significantly based on the training data used.

Table 5.1: Prey Decision tree accuracy on corresponding Prey policy from neural network

Tree	Test 1 (%)	Test 2 (%)	Test 3 (%)	Test 4 (%)	Test 5 (%)	Test 6 (%)	Test 7 (%)	Test 8 (%)	Test 9 (%)	Test 10 (%)	Mean (%)
1	93.2	92.2	91.3	93.1	96.0	92.9	95.9	94.9	98.0	99.0	94.6
2	52.9	69.3	84.0	81.0	72.0	78.0	65.0	75.0	80.0	84.7	74.2
3	75.5	68.3	66.3	74.3	69.3	72.0	71.7	72.7	73.7	73.5	71.7
4	65.7	78.4	70.6	69.3	71.3	77.0	62.6	60.2	71.4	69.4	69.6
5	68.0	69.9	80.2	60.0	76.0	70.0	88.9	82.8	71.4	87.8	75.5
6	74.8	64.7	59.8	80.4	71.3	62.6	58.2	69.4	59.2	62.2	66.3
7	75.5	88.2	87.1	73.3	79.2	70.3	83.0	80.8	83.5	81.4	80.2
8	76.5	73.5	71.3	75.2	75.0	63.0	76.0	58.6	76.5	67.3	71.3
9	74.8	95.1	86.0	91.0	85.0	75.0	86.0	88.0	87.8	86.7	85.5

5.2.2 Predator Decision Tree comparison

Similar to what was done for the policy of the Prey agent, Decision trees were made to represent the different policies of the Predator. Although in this case, all 5 features including Prey and Predator position as well as Predator stamina were used as inputs of the Decision tree. The Predator stamina was required as an input feature as the Predator stamina varied significantly across an episode and was limiting the actions taken by the predator. Furthermore, due to the action space of the predator being larger than the prey, a larger number of features were required for the more complex Decision trees of the Predator policies.

From the Predator Decision trees present in the Author's Github, it can be seen that the most important feature for the Predator was the Predator stamina as this features was generally at the root node of a Decision tree. This was especially apparent for all the policies which occurred after 4000 episodes of training as the stamina remained at the top for all these situations. The splitting value for the stamina which was generally observed the most was a value of 0.115. This makes intuitive sense as this was related to the minimum amount of stamina required to perform an action which enabled the Predator to move two steps at once, which was crucial in allowing the predator to collide with the prey, as well as catch up to the Prey when the Predator was far away. Furthermore, the importance of the stamina was further alluded to by the presence of the stamina attribute generally being several present in several different nodes in a single Decision tree.

Besides the stamina, there does not seem to a clear indication of the order of importance of the other features as there was not any consistency between which features was seen at nodes higher up in the tree. Therefore, the same conclusions which was made about the importance of features of the Prey can also be made about the remaining 4 features of the Predator. However, the splitting value of 0.071 was also present in all the different Decision trees which reveals that the position of the food was important to the Predator due to the frequency in which the Prey moves to that location.

Similarly to the Prey Decision trees, the classification accuracy of the Predator Decision trees were seen by comparing the outputs of a Decision tree against the output of the neural network corresponding to the same policy. The results of the classification can be seen in Table. 5.2. The results of the Predator classification was lower than the Prey classification accuracy which was to be expected due to the Predator action space being nearly twice as large as the action space of the Prey but still had an accuracy of but the accuracy was still approximately 68 %.

Table 5.2: Predator Decision tree accuracy on corresponding Predator policy from neural network

Tree	Test 1 (%)	Test 2 (%)	Test 3 (%)	Test 4 (%)	Test 5 (%)	Test 6 (%)	Test 7 (%)	Test 8 (%)	Test 9 (%)	Test 10 (%)	Mean (%)
1	76.9	73.1	83.3	82.4	88.0	85.9	65.7	82.7	87.6	88.5	81.4
2	56.3	72.8	66.7	72.5	58.4	58.0	60.2	61.2	60.2	77.1	64.3
3	61.5	66.0	45.1	47.5	65.3	59.0	55.6	54.1	64.9	61.5	58.1
4	68.0	74.8	67.0	72.5	67.6	76.2	69.7	68.8	65.6	76.0	70.6
5	44.2	65.7	74.5	82.4	65.3	60.6	66.7	68.4	60.8	73.2	66.2
6	60.2	58.3	63.7	76.2	63.4	57.0	55.1	65.3	60.2	53.6	61.3
7	67.6	81.6	78.4	62.7	68.6	69.3	67.3	69.8	81.2	79.2	72.6
8	59.6	69.2	74.8	62.0	69.0	58.6	62.2	62.2	63.3	62.9	64.4
9	65.0	77.7	71.8	80.2	67.7	77.8	82.8	83.8	82.7	83.5	77.3

5.3 Spectral Clustering

To obtain an quantitative understanding of how the policy of an agent changes as it learns, Spectral clustering was performed on the Decision trees. Spectral clustering was used to cluster the different Decision trees representing the different policies of the Predator and Prey agents in order to see if certain policies were more similart to one another and if there was any pattern in how a policy of an agent would change, e.g in a linear or a cyclical manner.

Clustering is one of the main tasks in machine learning where unlabeled data points are assigned to different clusters where similar data points are assigned to the same cluster. This is generally done by minimising the intra-cluster distance between points in a cluster whilst maximising the inter-cluster distance. This is generally possible if the different points in a cluster possess similar underlying characteristics. Although there may be underlying similarities between data points, the data points may have a non-linear relationship with one another which can be difficult to cluster using a traditional clustering method such as K-means clustering [63]. Therefore, this non-linearity needs to be taken into account when clustering the data. This can be achieved by using Spectral clustering [64, 65]. Spectral clustering consists of many different steps. Firstly, the data needs to be represented as a similarity graph between the N data points which are to be clustered. A graph is a set of vertices with a corresponding set of edges which connect the vertices which are similar to one another. A graph is obtained by computing a similarity measure between each pair of data points where a vertex

v_i represents a data point x_i.

There are several different measures of similarity or distance, between two data points p and q such as the Euclidean distance or Gaussian distance which can be seen in Eqn. 5.3 and 5.4:

$$\text{Euclidean(p,q)} = \sqrt{\sum_{i=1}^{n}(p_i - q_i)} \tag{5.3}$$

$$\text{Gaussian(p,q)} = e^{\frac{(p-q)^2}{2\sigma^2}} \tag{5.4}$$

By computing the similarity/distance between data points, this is used to produce a matrix known as the Adjacency matrix, A [65] . From the Adjacency matrix, it is possible to find out about the underlying structure of the data. This can be done by finding the Degree matrix, D, which is a diagonal matrix where the value at the entry (i,i) is equal to the number of edges connected to vertex i. By obtaining both the Adjacency matrix and the Degree matrix, the Laplacian Matrix L can be calculated according to Eqn. 5.5 [64, 65]

$$L = A - D \tag{5.5}$$

The significance of the Laplacian matrix is that it is a positive semi-definite matrix which enables it to be decomposed into its eigenvectors and eigenvalues. By finding the eigenvectors of the Laplacian matrix, it enables the original data points to be represented in an alternative way which is more indicative of the latent structure of the data. After the data is remapped into the eigenvectors of the Laplacian matrix, the data can be clustered by using a K-means clustering algorithm on the eigenvectors [65].

The process of K-means clustering consists of initialisation and then repetition of two steps:

- Assignment
- Update

K-random points are initialized to be cluster centroids, where k is the number of different clusters the data is categorized into. In the assignment step, each data point is assigned to the closest centroid. In the update step, the centroid is repositioned to the centre of the data points assigned to it. The assignment and update steps are repeated until the centroid stops moving [65].

5.3.1 Spectral clustering pairwise similarity results

To perform the Spectral clustering, it was first necessary to devise a measure of similarity between the different Decision trees. This was achieved by combining testing data from 6500 data points from a pair of policies and then measuring the fraction of the total number of data points where there were identical predictions from the pair of Decision trees related to those policies.

The fraction of identical predictions was used to form the Adjacency matrix of the Prey Decision trees which can be seen in Table. 5.3.
Following the creation of the Adjacency matrix, it was then necessary to find the optimum number of clusters required to cluster together similar Prey Decision trees. This was done by calculating the

Table 5.3: Results of the prediction similarity of pairs of Prey Decision trees

		Decision tree								
		1	2	3	4	5	6	7	8	9
Decision Tree	1	1.00	0.47	0.35	0.47	0.31	0.42	0.30	0.27	0.39
	2	0.47	1.00	0.46	0.54	0.38	0.45	0.62	0.36	0.59
	3	0.35	0.46	1.00	0.59	0.29	0.27	0.32	0.26	0.31
	4	0.47	0.54	0.59	1.00	0.35	0.37	0.44	0.35	0.38
	5	0.31	0.38	0.29	0.35	1.00	0.33	0.48	0.39	0.40
	6	0.42	0.45	0.27	0.37	0.33	1.00	0.50	0.39	0.52
	7	0.30	0.62	0.32	0.44	0.48	0.50	1.00	0.60	0.65
	8	0.27	0.36	0.26	0.35	0.39	0.39	0.60	1.00	0.50
	9	0.39	0.59	0.31	0.38	0.40	0.52	0.65	0.50	1.00

Laplacian matrix and then extracting the eigenvalues of the different eigenvectors of the Laplacian. From the graph of the eigenvalues seen in Fig. 5.2, the number of eigenvalues with a value of 0 represents how many connected components there should be present in a similarity graph.

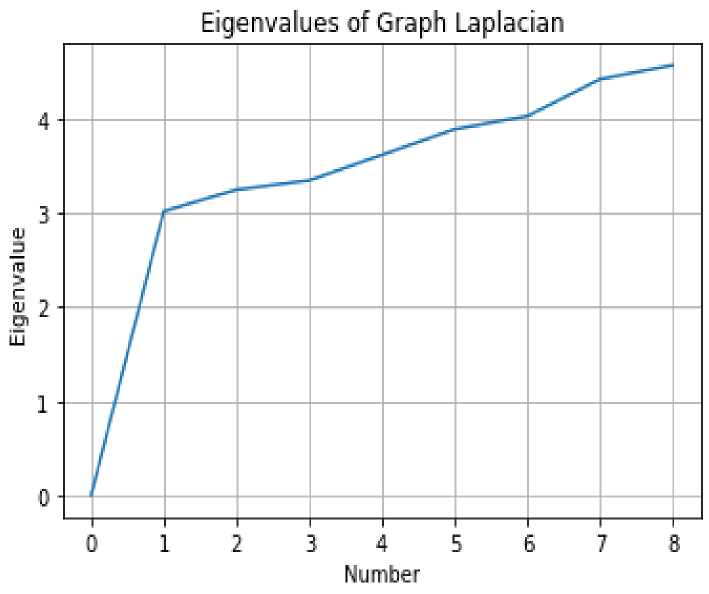


Figure 5.2: Plot of eigenvalues of the Laplacian matrix for the prey pairwise predictions

In this case, there was only a single eigenvalue at 0 which indicated that there was only a single connected component. In general, the magnitude of the eigenvalues with a non-zero value represent how connected that connected components is and how many broken links would be required in the similarity graph in order to increase the number of connected components [64]. Therefore, the number of eigenvalues before the first significant change in the magnitude between 2 consecutive eigenvalues was the optimum number of clusters to use, which in this case was a cluster number of 1 since the largest gap was the initial gap. This was further supported by attempting to perform the clustering despite evidence suggesting that there should only be 1 connected component.

From looking at the clustering results seen in Table. 5.4, there does not seem to be any clear indication of a consistent pattern in the clusters beside a slight preference for values of a same cluster being next to one another, which could show that the policy was changing at certain rate. Even though there does seem the reappearance of particular clusters at different policies which are far away from one another, there does not seem to be a clear pattern by which this occurs which suggests that the policy was not changing in a cyclical manner but instead in a random way.

Table 5.4: Prey Decision tree clustering results based on pairwise predictions similarity

Number of clusters	Clusters
2	0, 0, 0, 0, 1, 1, 1, 1, 1
3	0, 0, 2, 2, 1, 0, 1, 1, 0
4	2, 0, 3, 3, 1, 2, 1, 1, 0

The Spectral clustering for the Predator Decision trees was also performed in the same way as the Prey case which was by measuring the fraction of identical predictions from the pair of Decision trees. The Adjacency matrix for the Predator Decision trees can be seen in Table. 5.5 .

Table 5.5: Results of the prediction similarity of pairs of Predator Decision trees

		Decision tree								
		1	2	3	4	5	6	7	8	9
Decision Tree	1	1.00	0.36	0.24	0.27	0.23	0.14	0.09	0.05	0.09
	2	0.18	1.00	0.17	0.18	0.08	0.10	0.20	0.09	0.08
	3	0.11	0.19	1.00	0.15	0.06	0.07	0.09	0.09	0.09
	4	0.18	0.17	0.13	1.00	0.09	0.37	0.49	0.13	0.10
	5	0.21	0.11	0.13	0.12	1.00	0.29	0.09	0.21	0.33
	6	0.12	0.10	0.12	0.41	0.25	1.00	0.32	0.26	0.28
	7	0.17	0.22	0.05	0.49	0.06	0.23	1.00	0.20	0.03
	8	0.07	0.17	0.14	0.15	0.34	0.22	0.17	1.00	0.19
	9	0.13	0.05	0.06	0.20	0.34	0.21	0.09	0.36	1.00

The eigenvalues of the Laplacian matrix for the Predator Decision trees can be seen in Fig. 5.3. From the eigenvalues, it can be seen that the largest gap in the magnitude of the eigenvalues was also the initial gap which suggests that there should only be a single cluster for the Predator Decision trees. However by comparing Fig. 5.2 and 5.3, it can be seen that the magnitude of the gap between eigenvalues for the Predator Decision trees were lower than the Prey case, which indicates the lower similarity between the Predator Decision trees compared to the different prey Decision trees, which is to be expected due to the larger action space for the predator.

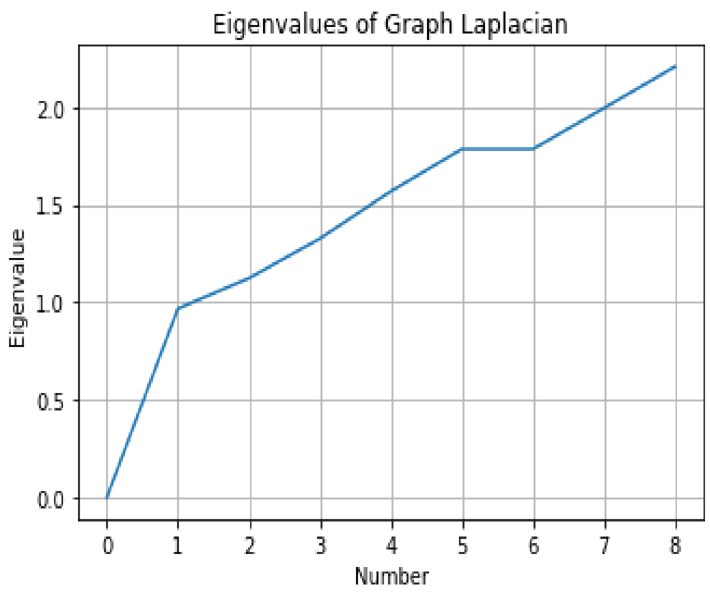


Figure 5.3: Plot of eigenvalues of the Laplacian matrix for the predator pairwise predictions

Despite one cluster being the most appropriate, clustering was performed with 2, 3 and 4 clusters and the results of the clustering can be seen in Table. 5.6. It was shown that policies in the same cluster are usually near one another supporting that the policy changes in a gradual way. Furthermore, the clusters in this case seem more stable than the Prey case as there was less variation in group a Decision tree would be clustered into when changing the cluster number.

Table 5.6: Predator Decision tree clustering results based on pairwise predictions similarity

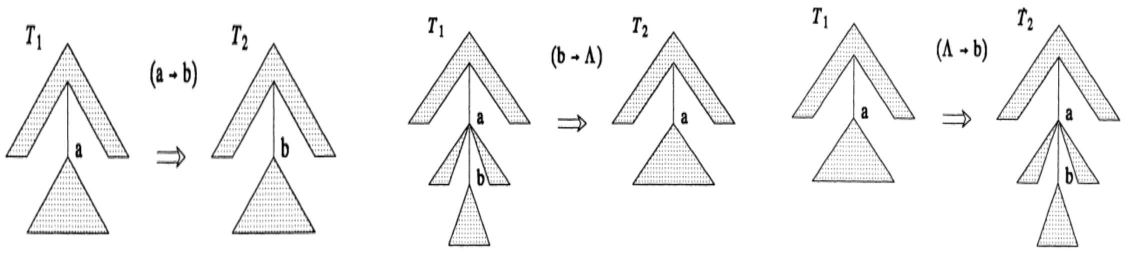
Number of clusters	Clusters
2	0, 0, 0, 0, 1, 1, 0, 1, 1
3	0, 0, 0, 2, 1, 2, 2, 1, 1
4	0, 0, 0, 2, 1, 2, 2, 3, 1

5.3.2 Spectral clustering tree edit distance results

To further analyse the Decision trees of a particular agent and to add robustness to the results of the Spectral clustering, the Spectral clustering was performed using another measure of similarity which was the tree edit distance between each pair of trees.

The tree edit distance is a measure of the difference in the structure of two different decision trees. The tree edit distance between two trees $\hat{x}$ and $\hat{y}$ is defined as the minimum number of node deletions, insertions and replacements that need to occur in $\hat{x}$ to obtain $\hat{y}$. The action of replacement results in changing one node label to another. Deleting a node n means making the children of n become the the children of the parent of n and then removing n (all children of the deleted node b become children of the parent a). Insertion of a node n as the child of the node n' will result in n becoming the parent of the current children of node n' [66, 67].

These different operations are illustrated in Fig. 5.4.



(a) Replacement (b) Deletion (c) Insertion

Figure 5.4: Tree edit operations [66]

There are several ways to define the tree edit distance such as having certain edits cost more than others which would involve the cost differing between different operations or operations occurring higher up the tree incurring a larger cost than operations lower down the tree. In this project, the tree edit distance was used as a measure of the difference between the Decision trees from different policies of a particular agent. Furthermore, the cost of each operation was equivalent regardless of the type of operation and the position of the tree which the operation occurred. Therefore, the tree edit distance was solely based on the length of the script required to transform $\hat{x}$ into $\hat{y}$.

The tree edit distances for the Prey Decision trees can be seen in Table. 5.7. This table was then used as an Adjacency matrix and Spectral clustering was performed after obtaining the Laplacian matrix from these values.

Table 5.7: Results of the tree edit distance between pairs of Prey Decision trees

		Decision tree								
		1	2	3	4	5	6	7	8	9
Decision Tree	1	0	42	28	58	33	47	38	34	45
	2	42	0	40	49	38	47	47	43	37
	3	28	40	0	57	35	43	36	37	43
	4	58	49	57	0	45	52	48	50	49
	5	33	38	35	45	0	39	36	30	40
	6	47	47	43	52	39	0	38	38	41
	7	38	47	36	48	36	38	0	34	44
	8	34	43	37	50	30	38	34	0	40
	9	45	37	43	49	40	41	44	40	0

Similar to the previous cases, the number of connected components was equal to 1 as shown from Fig.5.5. However, in this case the order of the clusters differ as shown by Table. 5.8, although it does seem to still retain the behaviour in which policies of a particular cluster occur close to one another.

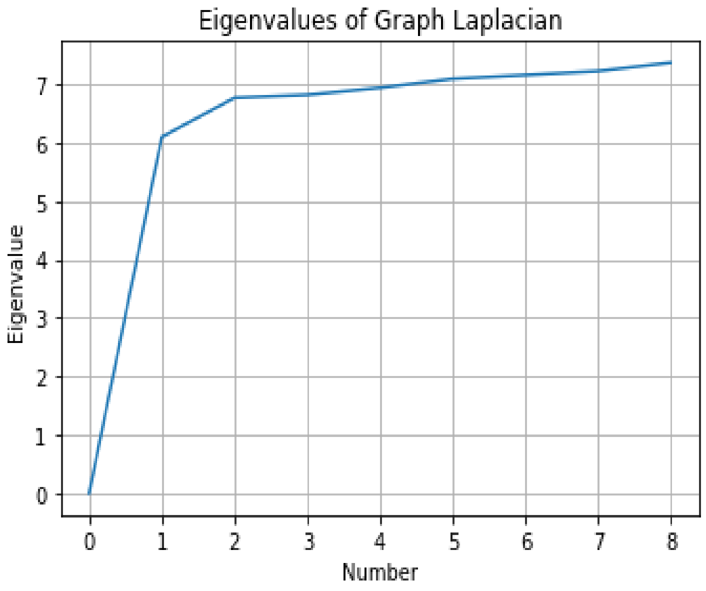


Figure 5.5: Plot of eigenvalues of the Laplacian matrix of the Prey policy based on tree edit distance

Table 5.8: Prey clustering results using based on tree edit distance

Number of clusters	Clusters
2	1, 0, 1, 0, 0, 0, 0, 1, 0
3	1, 0, 1, 2, 1, 1, 1, 1, 0
4	3, 0, 3, 2, 2, 1, 1, 1, 0

The tree edit distances for the Predator Decision trees can also be seen in Table. 5.9 and as before, the optimum number of connected components was 1 as seen from Fig.5.6. The Spectral clustering results for cluster numbers of 2,3 and 4 can be seen in Table. 5.10.

Similarly to the Prey case, the behaviour in which policies of a particular cluster occurred close to one another was present. Also, it seems that the clusters formed from the Predator Decision trees were more stable as there were less variation in the clustering as the number of clusters changed. From the results of the Spectral clustering using both distance measures, it was seen that the policies for Predator and Prey change in a gradual way as policies of a particular cluster appeared next to one another. The optimum number of clusters seemed to be 1 for both the Predator and the Prey when using both the classification accuracy and the tree edit distance as similarity measures. This was further supported by the fact that there did not seem to be any clear pattern in the occurrence of different clusters and that the appearance of a particular cluster appeared in a random way.

Table 5.9: Results of the tree edit distance between pairs of Predator Decision trees

		Decision tree								
		1	2	3	4	5	6	7	8	9
Decision Tree	1	0	63	46	52	44	55	46	54	43
	2	63	0	76	66	65	62	70	70	76
	3	46	76	0	51	44	57	44	59	47
	4	52	66	51	0	48	55	49	57	54
	5	44	65	44	48	0	47	45	55	48
	6	55	62	57	55	47	0	54	53	51
	7	46	70	44	49	45	54	0	52	34
	8	54	70	59	57	55	53	52	0	58
	9	43	76	47	54	48	51	34	58	0

Table 5.10: Predator clustering results using based on tree edit distance

Number of clusters	Clusters
2	1, 0, 1, 1, 1, 0, 1, 1, 1
3	2, 0, 2, 2, 2, 1, 2, 1, 2
4	3, 0, 1, 1, 1, 2, 3, 2, 3

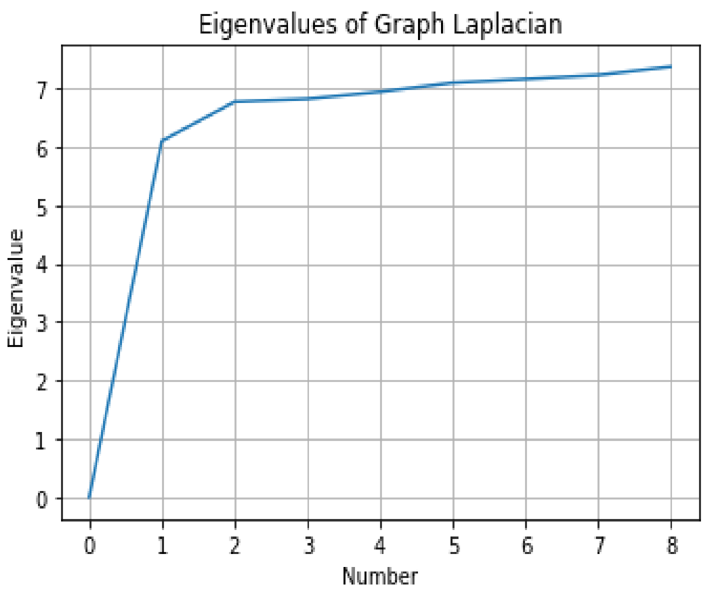


Figure 5.6: Plot of eigenvalues of the Laplacian matrix of the Predator policy based on tree edit distance

The results from the Spectral clustering using the tree edit distance were limited as the tree edit distance neglects to include the splitting value at the nodes and only includes the presence of the feature at each internal node. Furthermore, the tree edit distance only takes into account whether a node was a leaf node and does not take into account the predicted output at a leaf node.

Due to these limitations, the similarity from the common predictions of two trees were likely more accurate representation of the similarity between Decision trees. Also, the clustering used only 9 Decision trees which may not have been a sufficient number to use to find a clustering pattern.

6 Conclusion and Future work

The explainability of machine learning models are becoming crucial in society due to the use of machine learning in high risk multi-agent areas such as the use of autonomous vehicles. In order to use machine learning in a multi-agent domain, it is important to understand how the agents learn in a competitive or adversarial situation. This was done by examining a simplified Predator-Prey game where the only dynamic elements present in the environment were the agents.

Various methods were used to look at the different policies of an agent such as looking at the similar actions of an agent between two consecutive policies. A peculiar result found from this was that the Predator policy was varying even in a situation where the Prey was remaining stationary at one grid cell, which suggests the presence of different actions which lead to the Predator obtaining the same reward. Furthermore, by increasing the number of episodes between the evaluation of consecutive policies, consecutive policies were less similar to one another and would oscillate near a fixed value depending on the episode gap between consecutive policies. This shows that the process of changing a policy takes a significant amount of episodes to occur and the policy was changing at a somewhat fixed rate.

From comparing Decision trees representing the policies of a particular agent, the different Decision trees were sufficiently different from one another. This was shown by the low accuracy when examining actions in common between a pair of Decision trees as well the lack of consistency of a top feature for the Prey Decision trees. Nonetheless, the Decision trees showed that the agents behave in an intuitive manner where the Prey prioritized maximizing the distance from the Predator and moving to a food location by moving to the bottom left hand corner, which was shown by the splitting value of 0.071 being prevalent at nodes high up in the tree. Also, the Decision trees for the Predator policies prioritized the stamina of the Predator, which was crucial in enabling the Predator to move fast enough to catch the Prey. The Predator Decision trees also prioritized the splitting value of 0.071 due to the frequency in which the Prey was present in that location. Spectral clustering of the Decision trees of an agent was performed using two different measures of similarity. From looking at the Spectral clustering results using the similar action classification and the tree edit distance measures of similarity, the eigenvalue plots of the Laplacian matrix both showed that one cluster was the optimum number of clusters for the different Decision trees. This suggests that there was no distinct way to group together the different policies of an agent. However, despite one cluster being the optimum, by performing clustering on the Decision trees, potential insights were obtained such as policies of a particular cluster appearing close to one another. This supports the idea that the policy was changing in a gradual manner at a fixed rate and so consecutive policies were more likely to be similar to one another. Furthermore, besides the policies of an agent following the intuition discussed before, there does not seem to be any consistent pattern in how a policy changes.

This could be as a result of the stochastic nature of the neural network in which the Decision trees were modelling. As the Decision trees are only proxy models of a neural network, there are limitations in describing how the policy obtained from a neural network will behave especially since the accuracy of the Decision trees compared to the neural networks were only 76 and 68 % for the Prey and Predator Decision trees respectively. Furthermore, a more sophisticated calculation of the tree edit distance could be used to take into account the finer details of the Decision trees in order to obtain a more accurate result. To confirm whether the results from using a Decision tree proxy model were conclusive, an alternative proxy model such as LIME could be used to represent the policy of an agent.

Overall, the policies of the agents showed that they behaved in an intuitive manner and unexpected behaviour such as simply stalking the Prey did not occur after the reward shaping process. Therefore if a MAS is utilised, it is important to effectively configure the reward function of an agent. Also, due to the changes in the policy likely occurring in a random way, an approach to prevent unexpected behaviour from manifesting could be to formulate the changes in a policy as a constrained optimisation problem such as in the case of Trust-region Policy Optimization (TRPO) or Proximal Policy Optimization (PPO), where the policy of an agent cannot change too significantly at any point in time [68].

There are several different ways in which this work can be continued. Firstly, the simulation could be run for a longer number of episodes in order to see if the policy of the agents do converge at some point as well as obtaining a larger number of Decision trees for clustering. Also, as the Predator-Prey game was made to be quite simple with only 2 agents present in a fixed environment, further work could focus on investigating the learning of the agents in a more complex situation. The Predator-Prey game can be made more complex by:

- Giving the agents partial observability of the environment
- Adding additional properties to the agents
- Adding a third agent which could be hunted by the current Prey agent
- Food appearing in different locations as time progresses
- Food running out
- The reward from eating the food being based on how hungry the Prey agent is
- Adding obstacles which could prevent the Predator from passing but would not block the Prey

In this project, the agents learnt using a value based Reinforcement Learning algorithm. Therefore, it may be desirable to examine how a policy changes in the case where agents are trained using policy gradient based algorithms. As this project focused on how an agent learns only in the presence of a single adversary, the work can be extended by looking at how an agent learns in the presence of multiple adversaries as well as team mates. If the approach of learning in the presence of team mates and adversaries is pursued, then several different aspects could be varied. This includes examining how the learning of an agent changes when there is communications between the teammates of an agent or by changing the reward function so agents are rewarded based on the performance of whole team rather than based on their own performance. Another potential area of future work could be looking at the change in the policy if one of the agent was augmented by utilising a model of the opponent or if an agent was trained used hierarchical Reinforcement Learning.

References

[1] R. Roche, B. Blunier, A. Miraoui, V. Hilaire, and A. Koukam, "Multi-agent systems for grid energy management: A short review," in *IECON 2010-36th Annual Conference on IEEE Industrial Electronics Society*. IEEE, 2010, pp. 3341–3346.

[2] R. S. Sutton, A. G. Barto *et al.*, *Introduction to reinforcement learning.* MIT press Cambridge, 1998, vol. 135.

[3] T. T. Nguyen, N. D. Nguyen, and S. Nahavandi, "Deep reinforcement learning for multi-agent systems: A review of challenges, solutions and applications," *arXiv preprint arXiv:1812.11794*, 2018.

[4] "Multi-agent reinforcement learning," https://medium.com/@RemiStudios/multi-agent-reinforcement-learning-3f00b561f5f0, accessed: 2019-03-30.

[5] G. Gupta *et al.*, "A self explanatory review of decision tree classifiers," in *International conference on recent advances and innovations in engineering (ICRAIE-2014)*. IEEE, 2014, pp. 1–7.

[6] D. Quillen, E. Jang, O. Nachum, C. Finn, J. Ibarz, and S. Levine, "Deep reinforcement learning for vision-based robotic grasping: A simulated comparative evaluation of off-policy methods," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 6284–6291.

[7] Y. Li, Y. Wen, D. Tao, and K. Guan, "Transforming cooling optimization for green data center via deep reinforcement learning," *IEEE transactions on cybernetics*, 2019.

[8] R. Evans, J. Jumper, J. Kirkpatrick, L. Sifre, T. Green, C. Qin, A. Zidek, A. Nelson, A. Bridgland, H. Penedones *et al.*, "De novo structure prediction with deeplearning based scoring," *Annu Rev Biochem*, vol. 77, pp. 363–382, 2018.

[9] K. Tuyls and G. Weiss, "Multiagent learning: Basics, challenges, and prospects," *Ai Magazine*, vol. 33, no. 3, pp. 41–41, 2012.

[10] C. Kray, "The benefits of multi-agent systems in spatial reasoning." in *FLAIRS Conference*, 2001, pp. 552–556.

[11] I. Tomičić and M. Schatten, "A case study on renewable energy management in an eco-village community in croatia–an agent based approach," *International journal of renewable energy research*, vol. 6, no. 4, pp. 1307–1317, 2016.

[12] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J.-M. Allen, V.-D. Lam, A. Bewley, and A. Shah, "Learning to drive in a day," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8248–8254.

[13] S. Ji, Y. Zheng, Z. Wang, and T. Li, "A deep reinforcement learning-enabled dynamic redeployment system for mobile ambulances," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 3, no. 1, p. 15, 2019.

[14] I. Lachow, "The upside and downside of swarming drones," *Bulletin of the Atomic Scientists*, vol. 73, no. 2, pp. 96–101, 2017.

[15] M. J. Boyle, "The legal and ethical implications of drone warfare," 2015.

[16] A. Iglesias, P. Martínez, R. Aler, and F. Fernández, "Learning teaching strategies in an adaptive and intelligent educational system through reinforcement learning," *Applied Intelligence*, vol. 31, no. 1, pp. 89–106, 2009.

[17] Z. Xiong, X.-Y. Liu, S. Zhong, A. Walid *et al.*, "Practical deep reinforcement learning approach for stock trading," *arXiv preprint arXiv:1811.07522*, 2018.

[18] W. F. Hill, *Learning: A survey of psychological interpretations.* Thomas Y. Crowell, 1977.

[19] M. L. Minsky, *Theory of neural-analog reinforcement systems and its application to the brain model problem.* Princeton University., 1954.

[20] J. M. Bradshaw, "An introduction to software agents," *Software agents*, vol. 5, pp. 3–46, 1997.

[21] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[22] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, 2000, pp. 1057–1063.

[23] G. Tesauro, "Temporal difference learning and td-gammon," *Communications of the ACM*, vol. 38, no. 3, pp. 58–68, 1995.

[24] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Thirtieth AAAI conference on artificial intelligence*, 2016.

[25] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[26] A. Géron, *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems.* " O'Reilly Media, Inc.", 2017.

[27] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning.* MIT press, 2016.

[28] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[29] A. Levy, R. Platt, and K. Saenko, "Hierarchical actor-critic," *arXiv preprint arXiv:1712.00948*, 2017.

[30] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, 2016.

[31] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez *et al.*, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm. 2017," *arXiv preprint arXiv:1712.01815.*

[32] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson, “Learning latent dynamics for planning from pixels,” *arXiv preprint arXiv:1811.04551*, 2018.

[33] T. Weber, S. Racanière, D. P. Reichert, L. Buesing, A. Guez, D. J. Rezende, A. P. Badia, O. Vinyals, N. Heess, Y. Li *et al.*, “Imagination-augmented agents for deep reinforcement learning,” *arXiv preprint arXiv:1707.06203*, 2017.

[34] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser *et al.*, “Starcraft ii: A new challenge for reinforcement learning,” *arXiv preprint arXiv:1708.04782*, 2017.

[35] L. Panait and S. Luke, “Cooperative multi-agent learning: The state of the art,” *Autonomous agents and multi-agent systems*, vol. 11, no. 3, pp. 387–434, 2005.

[36] P. J. Hoen, K. Tuyls, L. Panait, S. Luke, and J. A. La Poutre, “An overview of cooperative and competitive multiagent learning,” in *Proceedings of the First international conference on Learning and Adaption in Multi-Agent Systems.* Springer-Verlag, 2005, pp. 1–46.

[37] M. Schatten, I. Tomičić, and B. O. Đurić, “A review on application domains of large-scale multiagent systems,” in *Central European Conference on Information and Intelligent Systems*, 2017.

[38] M. L. Littman, “Markov games as a framework for multi-agent reinforcement learning,” in *Machine learning proceedings 1994.* Elsevier, 1994, pp. 157–163.

[39] E. A. O. Diallo, A. Sugiyama, and T. Sugawara, “Learning to coordinate with deep reinforcement learning in doubles pong game,” in *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA).* IEEE, 2017, pp. 14–19.

[40] S. Yamada and T. Yamaguchi, “Training aibo like a dog,” in *the 13th International Workshop on Robot and Human Interactive Communication*, 2004, pp. 431–436.

[41] L. Pinto, J. Davidson, R. Sukthankar, and A. Gupta, “Robust adversarial reinforcement learning,” in *Proceedings of the 34th International Conference on Machine Learning-Volume 70.* JMLR. org, 2017, pp. 2817–2826.

[42] X. Pan, D. Seita, Y. Gao, and J. Canny, “Risk averse robust adversarial reinforcement learning,” *arXiv preprint arXiv:1904.00511*, 2019.

[43] J. Z. Leibo, V. Zambaldi, M. Lanctot, J. Marecki, and T. Graepel, “Multi-agent reinforcement learning in sequential social dilemmas,” in *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems.* International Foundation for Autonomous Agents and Multiagent Systems, 2017, pp. 464–473.

[44] J. Foerster, R. Y. Chen, M. Al-Shedivat, S. Whiteson, P. Abbeel, and I. Mordatch, “Learning with opponent-learning awareness,” in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems.* International Foundation for Autonomous Agents and Multiagent Systems, 2018, pp. 122–130.

[45] M. Egorov, “Multi-agent deep reinforcement learning,” *CS231n: Convolutional Neural Networks for Visual Recognition*, 2016.

[46] Y. Nagayuki, S. Ishii, and K. Doya, “Multi-agent reinforcement learning: An approach based on the other agent’s internal model,” in *Proceedings Fourth International Conference on MultiAgent Systems*. IEEE, 2000, pp. 215–221.

[47] R. Lowe, Y. Wu, A. Tamar, J. Harb, O. P. Abbeel, and I. Mordatch, “Multi-agent actor-critic for mixed cooperative-competitive environments,” in *Advances in Neural Information Processing Systems*, 2017, pp. 6379–6390.

[48] J. Schrum, *Competition Between Reinforcement Learning Methods in a Predator-Prey Grid World*. Computer Science Department, University of Texas at Austin, 2008.

[49] P. Zhou and H. Shen, “Multi-agent cooperation by reinforcement learning with teammate modeling and reward allotment,” in *2011 Eighth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, vol. 2. IEEE, 2011, pp. 1316–1319.

[50] M. Tan, “Multi-agent reinforcement learning: Independent vs. cooperative agents,” in *Proceedings of the tenth international conference on machine learning*, 1993, pp. 330–337.

[51] I. Partalas, I. Feneris, and I. Vlahavas, “Multi-agent reinforcement learning using strategies and voting,” in *19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2007)*, vol. 2. IEEE, 2007, pp. 318–324.

[52] “Marl patrolling agents,” https://github.com/bdvllrs/marl-patrolling-agents, accessed: 2019-08-28.

[53] T. Y. Moore and A. A. Biewener, “Outrun or outmaneuver: Predator–prey interactions as a model system for integrating biomechanical studies in a broader ecological and evolutionary context,” *Integrative and Comparative Biology*, vol. 55, no. 6, pp. 1188–1197, 2015.

[54] A. Y. Ng, D. Harada, and S. Russell, “Policy invariance under reward transformations: Theory and application to reward shaping,” in *ICML*, vol. 99, 1999, pp. 278–287.

[55] M. Riedmiller, R. Hafner, T. Lampe, M. Neunert, J. Degrave, T. Van de Wiele, V. Mnih, N. Heess, and J. T. Springenberg, “Learning by playing-solving sparse reward tasks from scratch,” *arXiv preprint arXiv:1802.10567*, 2018.

[56] Z. Li, Q. Ding, and W. Zhang, “A comparative study of different distances for similarity estimation,” in *International Conference on Intelligent Computing and Information Science*. Springer, 2011, pp. 483–488.

[57] C. Rudin, “Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead,” *Nature Machine Intelligence*, vol. 1, no. 5, p. 206, 2019.

[58] L. H. Gilpin, D. Bau, B. Z. Yuan, A. Bajwa, M. Specter, and L. Kagal, “Explaining explanations: An overview of interpretability of machine learning,” in *2018 IEEE 5th International Conference on data science and advanced analytics (DSAA)*. IEEE, 2018, pp. 80–89.

[59] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why should i trust you?: Explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining.* ACM, 2016, pp. 1135–1144.

[60] K. Kaur and D. Bhutani, "A review on classification using decision tree," *IJCAT-International Journal of Computing and Technology*, 2015.

[61] "Sci-kit learn decision tree classifier," https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html, accessed: 2019-08-15.

[62] "Decision-tree github," https://github.com/nerdk312/Decision-Trees, accessed: 2019-09-10.

[63] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning.* Springer, 2013, vol. 112.

[64] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *Advances in neural information processing systems*, 2002, pp. 849–856.

[65] U. Von Luxburg, "A tutorial on spectral clustering," *Statistics and computing*, vol. 17, no. 4, pp. 395–416, 2007.

[66] K. Zhang and D. Shasha, "Simple fast algorithms for the editing distance between trees and related problems," *SIAM journal on computing*, vol. 18, no. 6, pp. 1245–1262, 1989.

[67] B. Paaßen, "Revisiting the tree edit distance and its backtracing: A tutorial," *arXiv preprint arXiv:1805.06869*, 2018.

[68] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.