ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

Dipartimento di Informatica - Scienza e Ingegneria - DISI

Corso di Laurea Magistrale in Ingegneria Informatica

TESI DI LAUREA

in

AUTONOMOUS AND ADAPTIVE SYSTEMS M

Opponent Modelling using Inverse Reinforcement Learning

CANDIDATO
Martina Rossi

RELATORE:
Chiar.mo Prof. Mirco Musolesi

Anno Accademico 2019/2020

Sessione III

A tutti coloro che
hanno fatto parte della mia vita
in questi due anni e mezzo,
da una parte all'altra dell'Oceano

Abstract

One of the research areas of artificial intelligence (AI) which is particularly active lately concerns the study of autonomous agents, which are pervasive even in daily life. The main goal is developing agents which interact efficiently with other agents or humans. Consequently, these relations would be greatly simplified by the ability of autonomously inferring the preferences of other entities and adapting the agent's strategy accordingly. Therefore, the aim of this work is implementing a learning agent which interacts with another entity in the same environment and uses this experience to extrapolate the opponent's values. This information can be applied to cooperate or exploit the adversary, depending on the agent's objective. Thus, central themes are Reinforcement Learning, Multiagent environments and Value Alignment. The agent presented applies Deep Q-Learning and receives a reward which is computed blending the environment feedbacks and the opponent's rewards. These values are obtained executing the Maximum Entropy Inverse Reinforcement Learning algorithm on the previous interactions. The behaviour of the proposed entity is tested on two different environments: the Centipede game and the Apple Picking game. The outcomes obtained are promising since they demonstrate that the agent can properly infer the opponent's preferences and use this knowledge to adapt its strategy. However, the final behaviour does not always satisfy the expectations; thus, limitations of the current approach and future works to improve the agent are analysed.

Abstract

Un’area di ricerca particolarmente attiva ultimamente nel campo dell'intelligenza artificiale (IA) riguarda lo studio di agenti autonomi, notevolmente diffusi anche nella vita quotidiana. L'obiettivo principale è sviluppare agenti che interagiscano in modo efficiente con altri agenti o esseri umani. Di conseguenza, queste relazioni potrebbero essere notevolmente semplificate grazie alla capacità di dedurre autonomamente le preferenze di altre entità e di adattare di conseguenza la strategia dell'agente. Pertanto, lo scopo di questa tesi è implementare un agente, in grado di apprendere, che interagisce con un'altra entità nello stesso ambiente e utilizza questa esperienza per estrapolare le preferenze dell'avversario. Queste informazioni possono essere impiegate per cooperare o sfruttare l'interlocutore, a seconda dell'obiettivo dell'agente. Pertanto, i temi centrali sono il Reinforcement Learning, gli ambienti multi-agente e il Value alignment. L'agente presentato apprende tramite Deep Q-Learning e riceve una ricompensa che viene calcolata combinando i feedback dell’ambiente e il reward dell'avversario. Questi valori sono ottenuti eseguendo l'algoritmo Maximum Entropy Inverse Reinforcement Learning sulle interazioni precedenti. Il comportamento dell’agente proposto viene testato in due diversi ambienti: il gioco Centipede e il gioco Apple Picking. I risultati ottenuti sono promettenti poiché dimostrano che l'agente può dedurre correttamente le preferenze dell'avversario e utilizzare questa conoscenza per adattare la sua strategia. Tuttavia, il comportamento finale non sempre corrisponde alle aspettative; sono quindi analizzati i limiti dell'approccio attuale e i gli sviluppi futuri per migliorare l'agente.

Contents

1 Introduction

Autonomous agents are increasingly widespread and integrated into everyday life; indeed, an important research area in modern artificial intelligence (AI) focuses on these entities. Teaching AI systems to behave consistently with human intentions and desires is challenging. There are many different possibilities to solve this problem such as collecting and tagging all the required examples; nevertheless, this method, named Supervised Learning, is often infeasible for autonomous agents. Therefore, the Reinforcement Learning (RL) paradigm, based on trial and error, might be preferred. However, RL algorithms require a reward function to determine the feedback for each action; correctly defining this function results difficult as well. As a matter of fact, the risk is not only obtaining an agent which does not learn the wanted behaviour, but also an entity which might be harmful.

The problem of AI safety is thoroughly discussed in multiple studies, given the high number of accidents that could or have occurred (Amodei et al., 2016). Therefore, a possible approach to avoid the misspecification of the reward function consists in allowing the intelligent agent to autonomously extract this information from a series of demonstrations. This method is also similar to the human way of learning some practical tasks such as driving. Furthermore, it also favours the learning of more abstract concepts like moral values, this results particularly interesting for the field of Machine Ethics.

Overall, being able to obtain an agent which successfully interacts with other agents or humans is particularly important. Specifically, this interaction could be more proficient if the agent is able to infer the preferences of its adversary from previous experiences. In this way, the autonomous agent could align with these values or use this information to exploit its opponent, depending on its goal. Therefore, cooperation or exploitation would be encouraged changing some parameters in the agent initialization. The resulting entity would be independent and very flexible since it adapts its behaviour on the opponent's values, depending on the initial settings.

Consequently, the goal of this work is studying how to develop the described agent, focusing on every complex aspect that characterizes it. More specifically, the main objective is implementing a learning agent which infers the opponent's rewards and use this information to find the strategy that best satisfies its goal. Several approaches, discussed in the literature, are thoroughly analysed and central topics are Reinforcement Learning, Multiagent environments and Value Alignment. The presented agent is trained using the Deep Q-Learning algorithm and it receives a reward which is obtained combining the environment feedbacks and the opponent's preferences. These last values are computed applying the Maximum Entropy Inverse Reinforcement Learning procedure to a set of demonstrations, collected during previous interactions. We conduct testing in two different environments, the Centipede game and the Apple Picking game. In particular, these settings are chosen among Social Dilemmas, since they exemplify concrete problems which result very common in real interactions.

Considering the outcomes of the simulations that have been carried out, it results clear that the agent is able to infer the opponent's rewards; moreover, this information can be properly used to modify the learning process. Cooperative and exploiting behaviours are observed, depending on the parameters value; however, it is worth noting that encouraging cooperation appears easier than favouring exploitation. Furthermore, learning in the Apple Picking environment is significantly more challenging, compared to the Centipede game; thus, the agent's performance is worse in this case. These observations open some considerations on limitations of the proposed agent and future work to improve and extend its behaviour.

This dissertation is organised as follows. In Section 2 we discuss related work and the state of art in relevant areas. Chapter 3 describes the approach followed to design the agent with intrinsic reward, while the experimental settings and results are presented in Chapter 4. Finally, Chapter 5 concludes the dissertation, highlighting limitations and future work.

2 Related work

The central objective of this work is implementing an agent that is able to learn in a multiagent environment using data on the opponent's strategy which are autonomously inferred from previous interactions. Social dilemmas are considered as an interesting setting for the agents' execution. Therefore, the key relevant areas of this project are: Reinforcement Learning, Multiagent Learning, techniques to discover di opponent's strategy, the Value Alignment problem, and Social Dilemmas. These topics are illustrated and analysed in the following sections.

2.1 Reinforcement Learning

Reinforcement Learning is a machine learning paradigm that is inspired by the instinctive way of learning of all living beings. More specifically, the learning agent interacts with the surrounding environment and receives a reward signal from it, as a consequence of the agent's actions. These mechanism of trial-and-error and reward signals are the learning basis of a Reinforcement Learning agent (Kaelbling et al., 1996; Sutton & Barto, 2018).

Reinforcement Learning (RL) differs from the other two main Machine Learning paradigms, namely Supervised Learning and Unsupervised Learning. The former consists in learning through a training set of examples that are previously labelled by an expert; the goal of the agent is to generalize from these samples and choose the correct response in situations not included in the training data. In RL, agents do not have any direct information on how to solve a specific task, they can only try different actions and learn from the effects that these actions have on the environment. On the other hand, Unsupervised Learning is mainly focused on finding how unlabelled data are organized and correctly classify new examples (Sutton & Barto, 2018).

A peculiar challenge that distinguishes RL from other learning methods is the fundamental trade-off between exploration and exploitation. In particular, the agent collects information on the environment through experience and selects the actions that entail the highest reward. This means choosing actions already experienced in past interactions; however, the agent must explore the environment to find the optimal strategy. Hence, the dilemma is whether exploiting known high-reward actions, which could be a sub-optimal strategy, or exploring the remaining possibilities, with higher risk of failure. Indeed, in both situations the agent will probably fail the task before finding the optimal policy.

In a Reinforcement Learning system, the main components are a policy, a reward function, a value function and a model; however, the latter is not necessary (Sutton & Barto, 2018). The policy (denoted by π) describes the agent's behaviour and consists in a mapping between states and actions, $\pi: S \rightarrow A$; more specifically, it expresses the probability of selecting each possible action given a certain state. The optimal policy associates each state to the action that produces the greatest long term reward (Kaelbling et al., 1996). The policy could be represented by a look up table or through a function of different complexity. The reward function, given the agent's current state and action, determines how good or bad the agent's new state is, as a consequence of its action. More specifically, the reward function returns a numerical value and the agent's sole purpose is to maximize the total reward on the long run, which might not correspond to increasing the immediate reward. Defining a proper reward function is not a trivial problem; if it is badly specified, the agent will not learn the expected behaviour. The value function expresses the total reward that an agent can obtain starting from a certain state (or a state-action pair); therefore, this function considers the long-term reward, while the reward function gives only the immediate reward associated to each state. The value function is updated during training, considering the frequency with which some states follow the current one and their immediate reward. Since the sequence of states that follows the current one depends on the agent's behaviour, the value function is defined with respect to a particular policy. Finally, the model describes the environment providing the probabilities of moving from one state to another if a certain action is chosen. Not all Reinforcement Learning algorithms require it, but only those defined as model-based.

A Reinforcement Learning problem can be formalized as a Markov Decision Process (MDP). A MDP is defined as a tuple of four elements $\{\mathrm{S}, \mathrm{A}, \mathrm{R}, \mathrm{T}\}$; S is the set of states, A is the collection of possible actions, $\mathrm{R}: \mathrm{S} \times A \rightarrow \mathcal{R} \subset \mathrm{R}$ is the reward function and $\mathrm{T}: \mathrm{S} \times A \rightarrow P(s'|s, a)$ is the transition probability function, which is a probability distribution over S and determines the system's dynamics. In particular, this transition probability function defines the probability of experiencing the future state s' starting from state s and choosing action a (Kaelbling et al., 1996; Littman, 1994). The two main entities are the agent and the environment, which interacts at discrete time steps $\mathrm{t} = 0, 1, \dots, \mathrm{T}$.

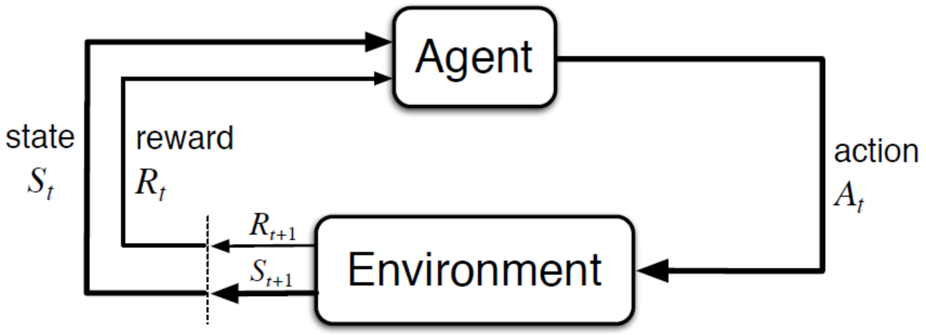


Figure 1 - Interaction between agent and environment in a MDP (Sutton & Barto, 2018).

At each time step t, the agent obtains, from the environment, the current state S_t and selects the action A_t. The following step, the agent receives a reward R_{t+1} and the updated state S_{t+1}. This sequence of states, actions and rewards produces a trajectory. If the MDP has finite state, action, and reward sets, then it is defined a finite MDP. The state is said to have the so called Markov Property if it contains all the information that influences the choice of the next action (Sutton & Barto, 2018).

The goal of the agent is maximizing the accumulated reward, which means maximizing the expected return from a certain state. This expected return can be formally expressed as

$$G_t \doteq R_{t+1} + R_{t+2} + \cdots + R_T$$

where R_T is the reward of the last step. This definition is meaningful when the considered problem has a final step, and the entire sequence of agent's interactions can be subdivided in series each ending with a final state; these series are defined episodes and these problems are called episodic tasks. On the other hand, in case of continuing tasks, there is not a clear definition of final state and the agent keep interacting with the environment infinitely ($T = \infty$). Hence, a more general definition of expected return is

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

where γ is a discount factor with values in [0,1]. The discount factor expresses the present value of future rewards and it is fundamental in case of continuing tasks to obtain a sum that converges to a finite value ($\gamma < 1$)(Sutton & Barto, 2018).

A Reinforcement Learning agent learns to succeed in his task by better estimating the value function. As already mentioned, the value function approximates the expected return from a given state following a chosen policy. The value function $v_{\pi}(s)$, which is more specifically called state-value function for policy π, can be formally defined as

$$v_{\pi}(s) \doteq \mathbb{E}_{\pi}[G_t|S_t = s] = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}|S_t = s\right]$$

It is worth noting that the value function is recursive; the relation between the value of the current state $v_{\pi}(s)$ and the value associated to the possible following states $v_{\pi}(s')$ is expressed by the Bellman equation.

$$\begin{aligned} v_{\pi}(s) &\doteq \mathbb{E}_{\pi}[G_t|S_t = s] \\ &= \mathbb{E}_{\pi}[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \sum_{a}\pi(a|s)\sum_{s'}\sum_{r}p(s',r|s,a)\big[r + \gamma\mathbb{E}_{\pi}[G_{t+1}|S_{t+1} = s']\big] \\ &= \sum_{a}\pi(a|s)\sum_{s',r}p(s',r|s,a)[r + \gamma v_{\pi}(s')] \end{aligned}$$

The policy that entails the highest cumulated reward, compared to all the possible policies, is called optimal policy, π_*. It always exists at least one optimal policy such that $v_{\pi_*}(s) \geq v_{\pi}(s), \forall s \in S$. The formal definition of the optimal state-value function is

$$v_*(s) \doteq \max_{\pi} v_{\pi}(s)$$

It is possible to re-write the Bellman equation for the optimal value function, defined Bellman optimality equation.

Moreover, it is possible to define an action-value function for policy π, $q_{\pi}(s,a)$, which expresses the expected return of choosing action a in state s and then following policy π.

$$q_{\pi}(s,a) \doteq \mathbb{E}_{\pi}[G_t|S_t = s, A_t = a] = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}|S_t = s, A_t = a\right]$$

The Bellman equation, the optimal action-value function and the Bellman optimality equation can be equivalently expressed considering $q_{\pi}(s,a)$ (Sutton & Barto, 2018).

2.1.1 Q-Learning

Solving a Reinforcement Learning problem means tackling the prediction problem and the control problem. In particular, the first corresponds to computing the value function of a given policy, while the second is finding the optimal policy. A class of methods suitable for these goals is represented by Temporal Difference Learning (TD Learning).

Temporal Difference methods use experience to compute the value function of a given policy, progressively updating estimated values using previous guesses. Indeed, the state value $V(S_t)$ is corrected after each time step; the update rule is:

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

The quantity between the square brackets $R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is called TD error (δ_t) since it can be interpreted as an error between the current estimate of the value function, $V(S_t)$, and a better evaluation of it $R_{t+1} + \gamma V(S_{t+1})$ (Sutton & Barto, 2018).

The update rule expressed above is the core of the Temporal Difference algorithm defined TD(0); this method is also called one-step TD, because it improves the state value estimates by looking ahead one step. Because of this behaviour, in which the algorithm updates a guess with a guess, TD learning is said to bootstrap. The complete TD(0) algorithm is described below.

Algorithm 1: TD(0) algorithm (Sutton & Barto, 2018)

```
$Given\ the\ policy\ \pi, the\ step\ size\ \alpha\ \in ]0,1]$:

$Arbitrarily\ initialize\ V(s)\ for\ all\ states, except\ V(terminal) = 0$
$Loop\ for\ each\ episode$:
    $Initialize\ S$
    $Loop\ for\ each\ step\ of\ episode$:
        $A \leftarrow action\ selected\ by\ \pi\ in\ S$
        $Take\ action\ A, observe\ R\ and\ S'$
        $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
        $S\ \leftarrow S'$
    $until\ S\ is\ terminal$
```

The main advantages of TD methods, compared to other algorithms such as Dynamic Programming and Monte Carlo methods, are the use of experience only, instead of requiring a model of the environment dynamics, and the possibility of immediately update the value function, instead of waiting until the end of the episode.

Regarding the control problem, a possible approach is the off-policy Temporal Difference control algorithm called Q-learning, which was firstly proposed by Watkins. In this case, the action-value function is computed and the Q-values are adjusted with the following rule (Watkins, 1989).

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q\,(S_{t+1}, a) - Q(S_t, A_t)\right]$$

In particular, this mechanism directly computes the optimal value q_*. Indeed, Q-learning is also defined off-policy TD control since it improves a policy which is not the one used to select actions. The complete algorithm is illustrated below.

Algorithm 2: Q-learning algorithm (Sutton & Barto, 2018)

```
$Given\ the\ step\ size\ \alpha \in ]0,1]\ and\ small\ \varepsilon > 0$:

$Arbitrarily\ initialize\ Q(s,a)\ for\ all\ states\ and\ actions, except\ Q(terminal,\cdot) = 0$
$Loop\ for\ each\ episode$:
    $Initialize\ S$
    $Loop\ for\ each\ step\ of\ episode$:
        $Choose\ A\ from\ S\ using\ policy\ derived\ from\ Q\ (for\ example\ \varepsilon - greedy)$
        $Take\ action\ A, observe\ R\ and\ S'$
        $Q(S,A) \leftarrow Q(S,A) + \alpha \left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$
        $S \leftarrow S'$
    $until\ S\ is\ terminal$
```

The Q-learning algorithm has been proven to converge if the state-action pairs are continually updated; moreover, the convergence is not affected by the degree of exploration. However, the time required to reach the optimal policy might be very high (Kaelbling et al., 1996).

The easiest approach for representing the state-value function (or the action-value function) is using tabular methods, which means storing in memory a table with an entry for any possible state (or state-action pair) and use this space to save and update the values. The Q-learning algorithm can be implemented using this technique in case of tasks with small state spaces.

However, tabular methods are not suitable for complicated problems both for the size of the state space and for the time required to compute the best policy; therefore, the value function must be approximated. Indeed, in many situations, each state is visited only once or few times; hence, some generalization is required. This can be obtained through function approximation, which is a form of supervised learning. A possible implementation involves a Neural Network with its weight vector $\mathbf{w}$ that can be updated to improve the value prediction.

Nevertheless, function approximation introduces issues that were not present in tabular methods. In particular, there are three elements, identified as "the deadly triad" (Sutton & Barto, 2018), that might produce instability and divergence: function approximation, bootstrapping and off-policy learning.

Despite the problems stated before, function approximation is often required and applied to Q-learning. In this case, the weights of the neural network can be updated using Stochastic Gradient-Descent (SGD), which modifies the weight vector by a small constant in the opposite direction of the gradient of the error:

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha[v_\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t)]\nabla\hat{v}(S_t, \mathbf{w}_t)$$

Where α is the step size ($\alpha > 0$), $v_\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t)$ is the difference between the expected and predicted state value and $\nabla\hat{v}(S_t, \mathbf{w}_t)$ is the gradient of the function with respect to the weights (Sutton & Barto, 2018).

This weight update can be similarly applied to semi-gradient Q-learning, obtaining the following expression.

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha\left[R_{t+1} + \gamma \max_a \hat{q}\,(S_{t+1}, a, \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t)\right]\nabla\hat{q}(S_t, A_t, \mathbf{w}_t)$$[1]

An additional technique that can be integrated in semi-gradient Q-learning is experience replay, which was introduced by (Lin, 1992). In particular, every time that the agent selects an action A_t in the current state S_t and receives a reward R_{t+1} and the following state representation S_{t+1}, the tuple $(S_t, A_t, R_{t+1}, S_{t+1})$ is saved in memory (Sutton & Barto, 2018). Specifically, each quadruple is an experience, and it is saved in the replay memory. At every time step, a mini-batch of past experiences is randomly sampled from the replay memory and it is used to apply multiple Q-learning updates. If the same experience is used too many times, an undesirable overtraining is obtained; therefore, it is important to apply some strategy to prevent this situation (Lin, 1992). For example, only the most recent N experiences should be saved in the replay memory, as implemented in (Mnih et al., 2015);

[1] (Mnih et al., 2015; Sutton & Barto, 2018)

this is particularly reasonable, since old experiences might be too different from the most recent ones and they could negatively affect the agent, which keeps learning.

Experience replay has multiple useful effects on the learning agent. First of all, it allows for an efficient use of experiences, which are presented to the agent multiple times instead of only once (Adam et al., 2012). In addition to it, with this mechanism, the agent learns considerably faster (Lin, 1992). Finally, the experiences are randomly sampled from the replay memory, therefore the agent learns from uncorrelated data; this removes one source of instability, favouring the Q-learning convergence (Sutton & Barto, 2018).

In addition to experience replay, another technique which improves the stability of the learning algorithm consists in using a second neural network, called target network, to compute the targets of the Q-learning updates. After a fixed number of training steps, the primary network, used to approximate Q, is cloned in the second network, which computes the targets for the same number of training steps. Then, the primary network is copied again and the process repeats. This delay, introduced between the Q value updates on the primary network and the moment in which these affect the computation of the Q learning targets, helps to avoid oscillations in the learning process (Mnih et al., 2015).

2.2 Multiagent learning

A multiagent system is a group of independent entities, called agents, which interact in the same environment. These agents may have different goals or a common objective; hence, they might compete or cooperate towards their target (Busoniu et al., 2008; Shoham & Leyton-Brown, 2008; K. Tuyls & Stone, 2018; Karl Tuyls & Weiss, 2012).

Considering the complexity and dynamism of this system, agents which learn their strategy are more robust, compared to entities which perform a deterministic behaviour. Multiagent learning combines machine learning methods with multiagent settings in order to obtain agents that learn their tasks dynamically (Karl Tuyls & Weiss, 2012).

The environment where these agents operate can be formally defined as an extension of the Markov Decision Process and it is called stochastic or Markov game. A Markov game is a tuple $\{\mathrm{P}, \mathrm{S}, \mathrm{A}, \mathrm{R}, \mathrm{T}\}$ where P is the collection of n agents, S is the set of k states, A is the series of joint actions ($A = A_1 \times A_2 \times \ldots \times A_n$, A_i is the set of actions available to agent i), R: $S \times A \rightarrow \mathbb{R}^n$ is the reward function and T: $S \times A \times S \rightarrow [0,1]$ is the transition probability function (Karl Tuyls & Weiss, 2012). It is worth noting that in a multiagent system, the reward received by each agent depends on the action chosen by all the entities. Hence, every agent influences the learning process of all the remaining actors that interact in the same environment (Shoham & Leyton-Brown, 2008).

A multiagent settings entails some new challenges, compared to the single learning agent situation. First of all, the curse of dimensionality is particularly clear in a multiagent system; indeed, complicated environments have enormous state spaces. Moreover, this is not the only reason: raising the number of agents exponentially increases the state-action combinations. In addition to it, the presence of several agents in the same environment produces non-stationarity; this significantly complicates learning. Finally, as mentioned before, each agent's action affects the learning of all the remaining entities; thus, the exploration of one agent can negatively influence the ability of other agents to find the optimal policy (Busoniu et al., 2008).

There are multiple approaches in multiagent learning that can be classified following different principles. For example, (Busoniu et al., 2008) subdivides them depending on the type of the task: fully cooperative tasks, fully competitive tasks and mixed tasks. In fully cooperative tasks, all the entities involved have the same goal and the same reward function; therefore, the objective is to maximize the common total return. There are different learning algorithms for this situation which might use explicit coordination methods, involve indirect coordination mechanism, or solve the task without considering coordination. On the other hand, fully competitive tasks involve agents with opposite objectives; thus, algorithms that are independent from the remaining agents are suitable in this situation. Finally, in mixed tasks, the agents' reward does not have any constraint; this might be the case of self-interested actors or cooperative agents which might compete with the other entities in some situations. Single agent Reinforcement Learning algorithms could be applied in this context, even if convergence is not guaranteed.

On the other hand, (Karl Tuyls & Weiss, 2012) classifies Multiagent Learning (MAL) in three groups: multiplied learning, divided learning, and interactive learning. In case of multiplied learning, each agent learns separately; therefore, even though agents can interact, the learning algorithm is not affected. In divided learning, the task is shared among the agents, which have the same objective. The overall learning is obtained through the agents' interaction, but the learning process of each agent is individual. Lastly, interactive learning is characterized by a joint learning process which involves all the agents.

Finally, (K. Tuyls & Stone, 2018) classifies the learning paradigms in individual learning, population learning and protocol learning. In case of individual learning, one agent or more learns independently in a common environment. In particular, agents can learn towards individual utility or social welfare. On the contrary, population learning is identified by group level learning, realized through local interactions. This is typically implemented with co-evolutionary learning or swarm intelligence paradigms. Ultimately, protocol learning is focused on learning the interaction mechanism among the agents; an example is the adaptive learning design paradigm.

Considering the features of the agents and environments presented in the following, the implemented tasks can be described as mixed tasks (Busoniu et al., 2008), while learning can be defined as multiplied learning (Karl Tuyls & Weiss, 2012) or individual learning (K. Tuyls & Stone, 2018), depending on the type of classification.

2.3 Discover the Opponent's Strategy

In the field of multiagent learning, there are multiple approaches to tackle this complex environment, as illustrated before. Specifically, the presence of other agents can be explicitly analysed or not. However, being able to infer how the other agents act is undoubtedly convenient (Albrecht & Stone, 2018). This ability is included in the broad category of understanding skills thoroughly analysed in (Dafoe et al., 2020), where the focus is Cooperative AI. Considering the simple case of two agents only, if one agent is able to understand the opponent's strategy, it can use this knowledge not only to cooperate with the other entity but also to exploit its policy, depending on the specific task. How to discover the opponent's strategy is a cutting-edge theme and very different solutions are offered.

The typical category of approaches to this problem is opponent modelling. In a multiagent environment, the ability of one agent to infer the beliefs and goals of the remaining entities helps to understand their behaviour. All this information can be included in a model of the agents, which returns specific predictions, given the required data (typically past interactions and observations). The capability of discovering the opponent's strategy is very powerful since it does not limit to the case of artificial agents only, but it can be applied to machine-human interactions. Being able to understand the intentions of the adversary can really improve the agent's performance in interactive tasks. Indeed, there are countless examples and situations in which knowing the goals and beliefs of the other entities help the agent to react in the proper way, depending on its specific goal; autonomous vehicles are a striking example. This is particularly important in situations where interaction occurs without any explicit coordination and communication system. Interesting categories of modelling methods are Policy Reconstruction, Type-based Reasoning, Classification and Plan Recognition (Albrecht & Stone, 2018). Policy Reconstruction approaches recreate the agent's process of selecting the proper behaviour and build a model that directly predict the agent's actions. Specifically, a possible implementation consists in Utility Reconstruction where the agent's preferences are modelled through a utility function associated to the actions. Using this approach, it is possible to learn any model and this is achieved during the agents' interactions; however, a high number of observations might be necessary to learn the proper model and this process could be complex in terms of time and space. Type-based Reasoning assumes a set of known types of agent models; these models are provided to the agent before it begins to interact and they can be generated in various ways. In particular, these data can be directly specified by a domain expert or could be derived from previous experiences, lived personally by the agent or contained in a database. Given this knowledge, the agent starts the interaction with an initial estimate of the opponent's model, depending on the occurrences of the prototypes; then, it refines this prediction thanks to the real-time experiences. This approach does not require building an agent's model from scratch for every execution; however, the pre-defined types must be provided in some way and this a priori knowledge is not always available. Moreover, an incorrect type-space produces absolutely wrong predictions. On the other hand, classification mainly focuses on assigning class labels to the agents, depending on specific properties. This can be obtained using various machine learning algorithms; however, complete models typically require a considerable amount of data and they are usually developed before agents' interactions. Finally, Plan Recognition has the objective of inferring the agent's goal and plan. This often involves a plan library, which contains the possible plans. Nonetheless, creating a complete plan library might be difficult (Albrecht & Stone, 2018).

Many concrete implementations use these ideas. The learning method "Learning with Opponent-Learning Awareness" (LOLA) (Foerster et al., 2018) accounts for the impact of an agent's policy on the other agent's learning. Specifically, a LOLA agent maximises the expected return after the opponent revises its strategy with one learning step. In terms of opponent's policy, two different approaches are proposed; in the first version, the agent can directly access the opponent's parameters. On the other hand, a more realistic implementation uses an opponent modelling strategy which resembles behavioural cloning.

Moreover, the research proposed by (Gallego et al., 2019) focuses on strategies to account for the presence of other agents in multiagent reinforcement learning. Indeed, single agent reinforcement learning approaches typically fail in multiagent settings due to the non-stationarity introduced by other agents that interact in the same environment. Specifically, a new framework is proposed; Threatened Markov Decision Processes extend the classical MDP setting in order to explicitly describe the presence of an opponent agent, which causes non-stationarity in the shared environment. Moreover, the Q-learning update rule is modified to handle this non-stationarity; indeed, the agent can successfully react to the opponent's move since the learning rule includes an average over the more likely actions of the competitor. Hence, the opponent's policy must be inferred in some way from previous observations. In particular, an approach similar to Fictious Play is introduced for this purpose; an alternative uses Type-Based Reasoning to solve uncertainties on the opponent's properties.

In the context of agents aware of the opponents, (He et al., 2016) adopts a neural network approach. In particular, the learning agent does not predict the adversary's actions directly, but it determines a hidden description of the opponent, training a Deep Q-Network using previous observations. This DQN is then used to choose the best next action from the actual state. The learning structure is composed by two different networks: the policy learning module computes the Q-values, while the opponent learning module derives the opponent strategy. Two different approaches are proposed to combine the two networks: DRON-concat simply concatenates the two modules, while DRON-MOE is based on a Mixture-of-Experts network.

An interesting approach to opponent modelling is described in (Markovitch & Reger, 2005), where the learning agent does not infer a complete model of its adversary, but it only deducts its weakness. In addition to it, this information is used only to bias the agent's choices. This solution aims to tackle the complexity and risk problems, which arises in opponent's modelling. Indeed, learning a complete and accurate model requires many examples and it is complicated. Moreover, if the model is not correct, the risk is that the predictions relating to it are harmful to the agent, rather than improving its behaviour. The opponent weakness model is elaborated before the agents' interaction starts. The concept of weakness is firstly defined, then a set of examples are collected and tagged, identifying the situations in which the adversary results weak; this process is obtained using a teacher. Finally, the examples are translated using a feature vector representation and an induction algorithm is applied to obtain a classifier. Once the classifier is ready, it can be used online to detect situations in which the opponent is at a disadvantage, influencing the agent's choices. This information can be integrated in the agent learning process in various ways such as a numerical value in the utility function. This method can be further improved adding a self-model of the learning agent's weakness. In this way, favourable actions are actions that are classified as strong for the subject and weak for the opponent.

Different approaches from explicit opponent modelling are proposed as well. The study described in (Anastassacos et al., 2020) introduces a probing system in order to gain experiences that include behavioural change of the opponent. This experience is then used to train the reinforcement learning agent with an adjusted reward. In addition to it, another possibility is offered by the Machine Theory of Mind (Rabinowitz et al., 2018). This method applies the human Theory of Mind to multiagent settings. As a matter of fact, the goal is to realize an agent which is able to infer the others mental states, that means understanding the opponent's intentions and future behaviour. This is obtained implementing a Theory of Mind neural network which assembles models of other agents using meta-learning. The architecture of the Theory of Mind network is formed by three different modules: the character net, the mental net, and the prediction net. Each part has a specific role in the complex task of determining an accurate characterization of the opponent and predicting the agent's behaviour. Indeed, the proposed system learns how to model other agents autonomously. This articulated structure allows to obtain detailed predictions but obviously requires training many more parameters.

Finally, a still different method to deduce the opponent's strategy may be to use the various types of imitation learning. The main idea is learning from a set of demonstrations; this approach is similar to the Policy Reconstruction technique mentioned before.

2.3.1 Imitation Learning

When the complexity of an agent's task increases, manually specifying its behaviour is very difficult. Moreover, even if the agent autonomously learns the proper policy, some information, such as a reward function, is needed. However, defining a correct reward function might not be easy as well and it often requires a thorough understanding of the agent's task and environment. Hence, the founding idea of Imitation Learning is learning directly from a set of demonstrations, typically provided by an expert (Hussein et al., 2017; Osa et al., 2018). The paradigm based on mimicking the given behaviour has biological foundations and it simplifies the teaching of complicated tasks. This approach can be used to learn the behaviour of anyone; therefore, it can also be applied in the context of learning the opponent's strategy.

Formally, Imitation Learning is a paradigm where the agent learns a policy which recreates the behaviour presented in the demonstrations. These examples can be provided by an expert or by another agent and they form the dataset $\mathcal{D} = (\tau_i, s_i, r_i)_{i=1}^{N}$. Each element is composed by a trajectory τ, which is a sequence of measurements linked to a specific demonstration (typically a sequence of state-action tuples). s_i specifies the context conditions of the demonstration and r_i is the reward signal that can be optionally provided. Solving an Imitation Learning problem means finding the policy which produces a behaviour that is the most similar to the one presented in the demonstrations dataset. The learning process can be executed both online or offline (Osa et al., 2018).

Imitation Learning methods can be subdivided into two broad categories: Behavioural Cloning and Inverse Reinforcement Learning (Ng & Russell, 2000; Osa et al., 2018). The first approach is based on directly learning the association between states, or contexts, and actions, without inferring the reward function. This can be solved using a supervised learning technique and the policy which is most compatible with the demonstrated behaviour is typically obtained through a regression problem. However, directly learning the state-action mapping has some limitations. Indeed, this method is not robust to significant state changes; if the demonstrations provided are not complete, the model is unable to generalize correctly when the agent is in states never seen before. Moreover, the clones generally fail in determining the underlying structure of the teacher's behaviour (Bratko & Šuc, 2002; Hussein et al., 2017).

On the other hand, Inverse Reinforcement Learning uses the examples provided to infer the reward function that the entity optimizes when acting (Hussein et al., 2017; Ng & Russell, 2000; Osa et al., 2018). However, there might be multiple equivalent reward functions associated to the same optimal policy; therefore, a supplementary objective function is required to find a unique solution to the problem.

2.3.2 Inverse Reinforcement Learning

Inverse Reinforcement Learning (IRL) was introduced by Russell, who provides a first definition of this computational problem (Russell, 1998).

Given:

1) measurements of an agent's behaviour over time, in a variety of circumstances,
2) measurements of the sensory inputs to that agent,
3) a model of the physical environment (including the agent's body)

Determine the reward function that the agent is optimizing

Deducting the reward from a set of demonstrations has multiple advantages. First of all, as mentioned for imitation learning algorithms in general, the agent can learn a certain behaviour without manually specifying the reward for each situation. This is particularly convenient if the task to be solved is complex and it is difficult to define an appropriate reward function, as in the case of self-driving cars (Abbeel & Ng, 2004; Arora & Doshi, 2020; Ng & Russell, 2000). Furthermore, IRL allows to implement a computational model of the behaviour of humans and animals; in this way, it is possible to realize an agent which learns its preferences from an expert or learns the behaviour of another agent. Finally, the reward function is more easily transferable to another agent, compared to the case in which a policy is learnt directly (Arora & Doshi, 2020; Ng & Russell, 2000).

Even if IRL allows to obtain the reward function of any entity of which behavioural examples are given, conventionally the observed agent is named *expert*, while the learning agent is called *learner* (Arora & Doshi, 2020). The policy guiding the expert is defined as π_E and it might be unknown, while the associated reward is called R_E. Considering these notations, a formal definition of the IRL problem can be presented.

Given a Markov Decision Process that represents the expert's dynamics, without the reward. Considering the demonstrations, which are set of trajectories $\tau, \mathcal{D} = \{\langle(s_0, a_0), (s_1, a_1), \dots, (s_j, a_j)\rangle_1, \dots, \langle(s_0, a_0), (s_1, a_1), \dots, (s_j, a_j)\rangle_{i=2}^{N}\}$, $s_j \in S, a_j \in A, and\ i, j, N \in \mathbb{N}$. Assuming that all the trajectories are perfectly observed, determine R_E that best explains the policy π_E, if it is known, or the expert's behaviour demonstrated in the trajectories (Arora & Doshi, 2020).

Considering this definition, Inverse Reinforcement Learning appears very promising; however, it has multiple challenging aspects. First of all, as mentioned before, there are multiple reward functions that are compatible with the demonstrations provided; they also include degenerate functions such as $R_E = 0$ everywhere (Ng & Russell, 2000). One possible reason is the limited number of examples given to the agent that makes the solution of the IRL problem ambiguous. If the expert's policy is available, this issue can be solved measuring the difference between the inferred policy and the true one, called Inverse Learning Error (Arora & Doshi, 2020). Alternative solutions typically consist in considering the margin between the best policy and the remaining ones or maximizing the entropy (Osa et al., 2018).

Moreover, the demonstrations available to the learner cannot typically cover all possible states associated with a certain task; hence, generalizability is an important property for the reward function. Indeed, a correct reward function reveals the expert's preferences if the current state is present in the examples and if the state is completely new. Obviously, too general reward functions are not accurate enough for specific situations; this is an important trade-off that should be considered when implementing the IRL algorithm (Arora & Doshi, 2020). Another important consideration concerns which examples should be provided to the agent. In fact, if the expert produces only trajectories that always start from the same state and follow the optimal strategy, the agent will probably not be able to manage errors in the trajectory, being in a whole new and different set of states (Hussein et al., 2017).

In addition to these elements, the correctness of IRL depends also on the reward function implementation. Specifically, the reward function is typically expressed as a combination of weighted features.

$$R(s, a) = \omega_1 \phi_1(s, a) + \omega_2 \phi_2(s, a) + \dots + \omega_k \phi_k(s, a) = \omega^T \phi(s, a)$$

where $\phi_k: S \to \mathbb{R}$ is the feature function, while $\omega_k \in \mathbb{R}$ are the weights (Arora & Doshi, 2020). The feature function associates specific features to each state thus it characterizes each trajectory. Each policy, and set of trajectories, can be described computing the feature expectation, which is defined as

$$\mu(\pi) = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t \phi(s_t) | \pi\right] \in \mathbb{R}^k$$

This equation simplifies the policy value formula $\mathbb{E}_{s_0 \sim D}[V^{\pi}(s_0)] = \omega \cdot \mu(\pi)$ (Abbeel & Ng, 2004). Considering the reward composition, solving the IRL problem means finding the more appropriate weights (Arora & Doshi, 2020). Hence, the expert's reward can be recovered only if it can be expressed as a combination of the chosen feature function. Indeed, there are multiple approaches to obtain a suitable feature function; raw data from demonstrations can be usually applied when the task is very simple. An alternative is manually designing the feature function, even if identifying the salient aspects, in a state, that affect the reward computation is not trivial. Finally, there are automatic feature extraction techniques, however this entails additional learning and tuning (Hussein et al., 2017).

Finally, IRL methods are significantly sensible to the problem size. Indeed, this class of algorithms are iterative; thus, if the task difficulty increases, they suffer from complexity escalation. Therefore, even if IRL might be very efficient for small problems, the time complexity required for real tasks can be restricting. Furthermore, the problem size influences the number of demonstrations as well. As a matter of fact, recovering the reward function in a complicated task requires many more examples, compared to a toy problem (Arora & Doshi, 2020).

Inverse Reinforcement Learning algorithms can be classified depending on the central idea used to learn the reward. In particular, the two most common approaches compute the maximum margin or the maximum entropy (Osa et al., 2018). In case of Maximum Margin Optimization, the goal is finding the reward function such as the demonstrated policy corresponds to the optimal one and the difference between the optimal policy and the next-best one is maximized (Arora & Doshi, 2020). Algorithms proposed by (Ng & Russell, 2000) and (Abbeel & Ng, 2004) are based on this concept. However, this method is suitable when there is a unique reward function that is significantly better than the remaining ones. On the other hand, Entropy Optimization identifies the distribution over all trajectories that is characterized by the maximum entropy, among the distributions with same feature expectations of the demonstrated behaviour (Arora & Doshi, 2020).

In addition to this classification, IRL methods can be also subdivided in model-based and model-free methods. The first approach assumes that system dynamics are known, while model-free algorithms do not consider this prior knowledge. Both these techniques are characterized by advantages and disadvantages. Model-free algorithms are suitable for systems with non-linear and unknown dynamics; however, a high number of trajectories must be sampled to estimate their distribution. On the opposite, computing the trajectory distribution is efficient for model-based methods, but learning the correct reward function can be difficult. Most of the IRL methods are model-based (Osa et al., 2018).

2.3.3 Maximum Entropy Inverse Reinforcement Learning

Maximum Entropy Inverse Reinforcement Learning (MaxEntIRL) is an algorithm proposed by (Ziebart et al., 2008) which adopts a probabilistic approach using the Maximum Entropy Principle. A key problem in Inverse Reinforcement Learning is that the demonstrated behaviour is usually imperfect and subject to noise; this method helps to deal with this uncertainty (Ziebart et al., 2008).

In the context of Information Theory, entropy was firstly introduced by (Shannon, 1948). Given a variable x with a discrete set of probabilities $\{p_1, \dots, p_n\}$, the entropy measures the amount of uncertainty associated to this variable which corresponds to the quantity of information contained in x. The formal definition is

$$H = -\sum p_i log(p_i)$$

If the variable x is characterized by a continuous probability distribution function $p(x)$, the entropy is defined as

$$H = -\int p(x) log\, p(x) dx$$

The goal of a IRL problem is recovering the expert's reward function using a set of demonstrations typically provided by the expert himself. Therefore, a condition that could be introduced to solve this problem is that the feature expectation of the demonstrations should match the feature expectation of the learnt policy. However, this equality is ambiguous; hence, the Maximum Entropy Principle states that the best and unbiased criteria to solve this ambiguity is choosing the probability distribution which has maximum entropy (Arora & Doshi, 2020; Jaynes, 1957).

The Maximum Entropy Inverse Reinforcement Learning algorithm is illustrated below using the following notation (Ziebart et al., 2008).

- $\mathcal{D}$ is the set of demonstrations, $\mathcal{D} = \{\langle(s_0, a_0), (s_1, a_1), \dots, (s_j, a_j)\rangle_1,\ \dots\ ,\ \langle(s_0, a_0),\ (s_1, a_1),\ \dots\ ,\ (s_j, a_j)\rangle_{i=2}^{M}\},\ s_j \in S, a_j \in A, and\ i, j, M \in \mathbb{N}$; M is the number of demonstrations.

- c_ω is the cost expressed with respect to the weight parameter ω; the cost is equivalent to the negative reward. The cost associated to the trajectory τ is computed as $c_\omega(\tau) = \omega^T \phi_\tau = \sum_{s\in\tau} \omega^T \phi_s$, where $\phi_k: S \rightarrow \mathbb{R}$ is the feature function.
- $\mathcal{T}: S \times A \rightarrow Prob(S)$ is the transition dynamics. Specifically, it is a probability distribution over the set of next states conditioned on the probability for the agent of taking action a at state s, $p(s_{t+1}|s_t, a_t)$.
- $p(s|\omega, \mathcal{T})$ is the state visitation frequency, which is computed with respect to the weights ω and the transition dynamics $\mathcal{T}$.

In Max Entropy IRL the expert is modelled considering that the probability of having a specific trajectory τ in the demonstrations is proportional to $e^{-c_\omega(\tau)}$ ($p(\tau) \propto e^{-c_\omega(\tau)}$), where $-c_\omega(\tau)$ is the negative cost of that trajectory. This means that trajectories with identical costs are executed with equal probability, while trajectories with higher costs are exponentially less likely.

The entropy is included in the objective of the expert in order to solve the uncertainty mentioned before; in particular, the goal of the demonstrator is expressed as

$$\min_\pi \mathbb{E}_\pi[c_\omega(\tau)] - H(\pi)$$

The IRL problem can be solved finding the parameters ω, since the cost (as the reward) can be expressed as a combination of weights and features. Hence, the best ω maximizes the total probability of the trajectories provided by the expert. Therefore, ω is obtained maximizing the log-likelihood of the demonstrations.

$$\omega = \underset{\omega}{\operatorname{argmax}}\, log \prod_{\tau_d \in \mathcal{D}} p(\tau_d)$$

This can be equivalently written as a minimization problem, changing the sign of the equation.

$$\omega = \underset{\omega}{\operatorname{argmin}} - \frac{1}{M} \sum_{\tau_d \in \mathcal{D}} log\ \frac{1}{Z} e^{-c_\omega(\tau_d)}$$

where the factor $\frac{1}{M}$ does not influence the solution, but it is convenient for the following derivation. $p(\tau_d)$ in the first equation is replaced with $\frac{1}{Z} e^{-c_\omega(\tau_d)}$ because $p(\tau) \propto e^{-c_\omega(\tau)}$; specifically, $p(\tau_d) = \frac{1}{Z} e^{-c_\omega(\tau_d)}$ and $Z = \sum_\tau e^{-c_\omega(\tau_d)}$ is the partition function[2].

[2]The cost function c_ω is always expressed with respect to the parameter ω, however the subscript is omitted in the following for a clearer notation.

The objective function can be further expanded.

$$\omega = \underset{\omega}{\operatorname{argmin}} -\frac{1}{M}\sum_{\tau_d \in \mathcal{D}} log\ \frac{1}{Z} e^{-c(\tau_d)} = \ \underset{\omega}{\operatorname{argmin}} \frac{1}{M}\sum_{\tau_d \in \mathcal{D}} c(\tau_d) + log \sum_{\tau_d \in \mathcal{D}} e^{-c(\tau_d)}$$

The minimization argument is identified with $\mathcal{L}(\omega)$ in the following.

$$\mathcal{L}(\omega) = \frac{1}{M}\sum_{\tau_d \in \mathcal{D}} c(\tau_d) + log \sum_{\tau_d \in \mathcal{D}} e^{-c(\tau_d)}$$

$\mathcal{L}(\omega)$ is a convex function for deterministic MDPs, therefore the minimization problem has a unique solution that can be found using gradient descent. The gradient of $\mathcal{L}(\omega)$ is

$$\nabla\mathcal{L}(\omega) = \frac{1}{M}\sum_{\tau_d \in \mathcal{D}} \frac{dc(\tau_d)}{d\omega} - \frac{1}{\sum_{\tau_d \in \mathcal{D}} e^{-c(\tau_d)}} \sum_{\tau_d \in \mathcal{D}} \left(e^{-c(\tau_d)} \frac{dc(\tau_d)}{d\omega} \right)$$

This expression can be equivalently written as

$$\nabla\mathcal{L}(\omega) = \frac{1}{M}\sum_{\tau_d \in \mathcal{D}} \frac{dc(\tau_d)}{d\omega} - \sum_{\tau_d \in \mathcal{D}} \left(\frac{1}{\sum_{\tau_d \in \mathcal{D}} e^{-c(\tau_d)}} e^{-c(\tau_d)} \frac{dc(\tau_d)}{d\omega} \right)$$

The probability of each trajectory is $p(\tau_d) = \frac{1}{Z} e^{-c_\omega(\tau_d)} = \frac{1}{\sum_\tau e^{-c_\omega(\tau_d)}} e^{-c_\omega(\tau_d)}$ and it depends on the parameters ω and the transition dynamics $\mathcal{T}$ because $p(\tau_d)$ can be expressed as

$$p(\tau_d)\ = \mathrm{p}(\mathrm{s}_1) \prod_t p(a_t|s_t) p(s_{t+1}|s_t, a_t)$$

where $p(a_t|s_t)$ is the policy with respect to ω and $p(s_{t+1}|s_t, a_t)$ represents the transition dynamics $\mathcal{T}$. Therefore, the $\nabla\mathcal{L}(\omega)$ can be expressed as

$$\nabla\mathcal{L}(\omega) = \frac{1}{M}\sum_{\tau_d \in \mathcal{D}} \frac{dc(\tau_d)}{d\omega} - \sum_{\tau_d \in \mathcal{D}} \left(p(\tau_d|\omega, \mathcal{T}) \frac{dc(\tau_d)}{d\omega} \right)$$

Considering the second term, the sum over all possible trajectories in the demonstrations can be also expressed as the sum over all possible states.

$$\nabla\mathcal{L}(\omega) = \frac{1}{M}\sum_{\tau_d \in \mathcal{D}} \frac{dc(\tau_d)}{d\omega} - \sum_{s} \left(p(s|\omega, \mathcal{T}) \frac{dc(s)}{d\omega} \right)$$

where $p(s|\omega, \mathcal{T})$ is the state visitation frequency.

Given the cost expression $c_\omega(\tau) = \omega^T \phi_\tau = \sum_{s \in \tau} \omega^T \phi_s$, the final equation for the gradient $\nabla\mathcal{L}(\omega)$ is presented below.

$$\nabla\mathcal{L}(\omega) = \frac{1}{M} \sum_{\tau_d \in \mathcal{D}} \phi_{\tau_d} - \sum_s p(s|\omega, \mathcal{T}) \phi_s$$

The state visitation frequencies can be computed applying a dynamic programming algorithm, which is briefly illustrated in the following lines.

1. Compute the optimal policy $\pi(a|s)$ given c_ω using value iteration.
2. Compute $\mu_t(s)$ which is the probability of visiting the state s at time t.

$$\mu_1(s) = \ probability\ of\ s\ being\ the\ initial\ state$$

$$for\ t = 1\ to\ T\ (horizon\ of\ the\ MDP):$$

$$\mu_{t+1}(s) = \sum_a \sum_{s_{t-1}} \mu_t(s_{t-1}) \pi(a|s_{t-1}) p(s_t|s_{t-1}, a)$$

3. Compute the final state visitation frequency $p(s|\omega, \mathcal{T}) = \sum_T \mu_t(s)$.

Finally, the complete iterative Maximum Entropy IRL algorithm is summarised below (Ziebart et al., 2008).

1. Initialize the cost parameters ω and collect the expert's demonstrations $\mathcal{D}$.
2. Compute the optimal policy $\pi(a|s)$ with respect to the present cost c_ω using value iteration.
3. Compute the state visitation frequencies $p(s|\omega, \mathcal{T})$.
4. Compute the gradient of the function $\mathcal{L}(\omega)$ with respect to the parameters ω.

$$\nabla\mathcal{L}(\omega) = \frac{1}{M} \sum_{\tau_d \in \mathcal{D}} \phi_{\tau_d} - \sum_s p(s|\omega, \mathcal{T}) \phi_s$$

5. Update the parameters ω with one gradient step $\alpha \cdot \nabla\mathcal{L}(\omega)$.
6. Repeat from step 2 until the gradient $\nabla\mathcal{L}(\omega)$ is sufficiently close to 0 (or it is not possible to obtain further progress).

2.4 Value Alignment

The Value Alignment problem is strictly linked to the theme discussed in the previous section, where the goal is discovering the opponent's strategy. Moreover, this topic is part of the broader field of Machine Ethics whose objective is adding moral concepts and faculties in autonomous machines, which are defined Artificial Moral Agents (AMAs). This is an important subject which is emerging with the significant increase of autonomous systems in the everyday life (e. g. driverless cars and trains)(Allen et al., 2006; Wallach, 2010). Indeed, machine learning systems might exhibit unwanted or dangerous behaviour due to multiple possible problems in their implementation. For example, agents' objective functions could be wrongly specified or hacked; furthermore, the objective functions might be too complicated to be frequently evaluated, thus extrapolation from limited examples can produce adverse actions. Finally, the learning process could be characterized by unwanted behaviour (Amodei et al., 2016)

In the field of Artificial Intelligence, the Value Alignment problem refers to the need of correctly matching human preferences with machines' behaviour. This means that intelligent systems should be implemented in order to be beneficial for humans (Gabriel, 2020; Peterson, 2019; Russell, 2020). However, there are multiple interpretation of the definition of beneficial in this context; (Gabriel, 2020) presents an utilitarian view, where machines should act to satisfy the highest number of people, together with other approaches, such as intelligent systems following abstract notions of fairness and kindness. On the other hand, (Russell, 2020) states three principles summarized as: machine's actions should correspond to human preferences, which are not evident at the beginning of the interaction, thus they must be clarified by human behaviour.

An important question that requires a deep reflection is what values and principles should be included in intelligent agents. Typically, there are two main categories of approaches for defining moral values: top-down and bottom-up. Top-down techniques try to directly introduce the principles proposed by moral philosophers, while bottom-up methods focus on the evolution of mechanisms that favour moral behaviour (Wallach, 2010). However, the value alignment problem results challenging even for the technical aspect of how to encode these principles in the agents (Gabriel, 2020). Indeed, the issue is how to represent the moral values that the machine should always follow.

Moral values are concepts that intrinsically influence human behaviour; for this reason, they can be also interpreted as an intrinsic motivation. In particular, considering a Reinforcement Learning environment, moral values can be implemented through an intrinsic reward.

2.4.1 Define moral values

The problem of modelling moral values can be solved adopting a direct or indirect approach. In the first case, it is necessary to define a specific method to represent the ethical concepts. These form a knowledge base which is included in the agent and it is available at runtime. (Peterson, 2019) presents a solution that is an example of this approach. Specifically, moral principles are geometrically represented using Voronoi tessellations determined by ethical prototypes. These models are called paradigm cases because they are characterized by a known analysis. Hence, ethical considerations are based on the similarity between the new moral situation and formerly analysed paradigm cases. This method entails an accurate definition of ethical principles and prototypes, which are typically domain specific. Moreover, a thorough technique to compute similarities must be identified. As a consequence, this solution to the problem of defining moral values requires a deep analysis of the specific environment which is far from trivial.

On the other hand, instead of manually identifying moral principles and then finding a proper representation, agents could autonomously create their own representation of these values. This is realized presenting examples of moral behaviour to the learning agents. These elements are the key concepts of the indirect approach. (Wu & Lin, 2018) proposes to use reward shaping to integrate ethical behaviour, inferred from human examples, inside the Reinforcement Learning algorithm used to train the agent. Another example is offered by (Ritesh Noothigattu et al., 2018) where the goal is learning a model of preferences that are properly aggregated to solve moral dilemmas and determine the correct behaviour. This method is based on collecting data of human preferences about alternatives in ethical questions; then this data is used to learn a model of the choices for each person who provided the moral data. The single models are then combined in a unique one which summarizes the overall preferences; finally, at runtime, this model is used to select the alternative which is consistent with the collected data.

An indirect approach is adopted by (R. Noothigattu et al., 2019) as well, where a framework with two policies is proposed. More specifically, the first one maximizes the reward received from the environment, while the second policy is computed applying Inverse Reinforcement Learning to the demonstrations of moral behaviour generated by humans or other agents. Thus, the second policy follows the constraint specified by the examples provided. These policies are then combined using a third element, an orchestrator, that chooses which policy should be followed in a certain moment. The orchestrator is implemented using a contextual bandit algorithm. The advantage of this implementation is the interpretability, since it is always possible to determine which policy the agent is following at each point of time. An alternative solution could be directly combining the two rewards; this approach does not clarify the origin of the policy followed, as before. The choice of the best method depends on the specific context and the potential need of keeping the two policies separated.

Finally, a slightly different method is introduced in (Hadfield-Menell et al., 2016). Here, the goal is implementing an agent which learns to interact with humans in order to favour and help them.

Therefore, a formal definition of this problem, called Cooperative Inverse Reinforcement Learning (CIRL), is proposed. In this context, the agent is aware of the presence of the human in the same environment and the agent learns to maximize his reward.

2.4.2 Intrinsic reward

Living beings often act in a certain way without being prompted by the will to solve a practical problem; consequently, what causes this behaviour is called intrinsic motivation. However, learning occurs also in these moments which are fundamental for improving skills that will be applied in practical tasks. Hence, intrinsically motivated behaviour favours the development of independent and autonomous entities. More specifically, in the psychological field, a distinction is made between extrinsic and intrinsic motivation. The first one refers to the intention to act with the prospect of receiving a rewarding result, while intrinsic motivation promotes a certain behaviour because it is inherently satisfying (Barto et al., 2004; Singh et al., 2004). Therefore, this second motivation favours exploration without an external reward; in a Reinforcement Learning setting, intrinsic motivation helps learning the optimal policy. Because of these properties, it is possible to interpret this type of behaviour by associating an intrinsic reward to these actions. This analysis is also reflected in neuroscience as dopamine production has been associated with exploratory behaviours and the discovery of new situations (Barto et al., 2004). Consequently, there are multiple studies to introduce intrinsic reward in Reinforcement Learning algorithms in order to achieve various goals.

Considering the biological mechanism of dopamine production, a possible implementation of intrinsic reward consists in computing the novelty of significant events that the agent experiences. This approach is adopted by (Singh et al., 2004), where the agent uses intrinsic rewards to learn a set of skills. These skills form a knowledge base that is created and is available at runtime; this is used to increase the number of possible actions for the agent. During execution, every time that a salient event occurs, the intrinsic reward is proportional to the error made while predicting the experienced event. A similar approach is presented in (Şimşek & Barto, 2006), where the goal is executing optimal exploration with the purpose of learning a policy which allows exploitation in the following. Here, the agent computes two value functions: the task value function and the behaviour value function. The first one is applied to effectively solve the given task in the future, while the second one is used to choose the actions in the present. Two different policies are associated respectively, task policy and behaviour policy. The intrinsic reward is computed considering the task value function before and after each update; the behaviour value function is improved using this intrinsic reward and the external state experienced.

The novelty concept mentioned in the previous paragraph is adopted and revised in (Pathak et al., 2017) where curiosity is introduced. The goal is solving the problem of sparsity of external rewards which characterizes many concrete tasks. More specifically, the agent presents an Intrinsic Curiosity Module which computes the intrinsic reward, named curiosity, given the previous state, the action chosen and the following state. Inside this module, there is a neural network which predicts the future state, considering the actual state and the selected action. The error between the predicted future state and the actual future state is used to define the intrinsic reward; indeed, this error is bigger when the future state is more unpredictable, and this corresponds to the definition of novelty. Finally, the overall policy is computed to maximize the expected sum of the extrinsic and intrinsic rewards. The same idea of novelty is present in (Savinov et al., 2019), where the agent's observation of the environment are saved. This episodic memory is used to compare the present state with previous experiences and unseen scenarios are associated to positive intrinsic reward; searching for unfamiliar observation is consistent with seeking for novelty. This approach slightly differs from the curiosity computation adopted in (Pathak et al., 2017) because a procrastination-like behaviour has been observed in some situations. Indeed, the agent might focus on maximizing the unpredictability of new states instead of moving towards the goal of the task. Using an episodic memory seems to solve this problem; here, instead of implementing a neural network that predicts future states, it is only required a function that computes similarity between observations.

Apart from using intrinsic reward to express curiosity for new situations, encouraging exploration or skill acquisition, it can be also applied to stimulate other behaviours. (Wang et al., 2019) defines an intrinsic reward that collects social preferences. More specifically, there is a population of agents which is involved in an Intertemporal Social Dilemma and this intrinsic reward is computed using a formula that considers the inequity aversion of a specific agent. On the other hand, (Lipton et al., 2018) introduces the intrinsic fear (IF) which is a penalty that expresses the risk associated to a certain state. Indeed, many environments have states which lead to fatal situations; hence, optimal agents should rarely visit these states or completely avoid them. However, Deep Reinforcement Learning agents might fail in that because the use of function approximation leads to forget infrequent experiences. Hence, the fear model is introduced to avoid tragic states or dangerous situations that probably degenerates in them. This model is trained to predict which states probably lead to fatal situations in less than *k* steps. The output is a probability which is scaled with a specific factor and it is treated as a penalty to the reward received from the environment.

Following these different uses of intrinsic reward and the considerations about discovering the opponent's strategy and modelling moral values, ethical principles can be learnt applying Inverse Reinforcement Learning to the demonstrations provided by an expert or another agent. More specifically, with IRL algorithms it is possible to infer the reward function that the demonstrator is maximizing. Hence, these rewards can be considered as an intrinsic reward. The agent, observing the entity with which it interacts, infers his principles and his behaviour in the form of rewards. These are autonomously deduced by the learning agent using an independent algorithm; therefore, it can be seen as information that intrinsically conditions its behaviour and influences the learning process.

2.5 The Social Dilemmas and games

Many real problems that involve individuals or entire populations are social dilemmas. Therefore, they are analysed by multiple disciples such as psychology, biology, mathematics, and economics (Van Lange et al., 2013). In these situations, the options available to each decision maker can be typically categorized as "cooperation" or "defection" (Segismundo S. Izquierdo, 2008). A first definition of these problems is given by (Dawes, 1980), where dilemmas are described by two main properties. More specifically, in a social dilemma, each individual obtains a higher payoff choosing to defect instead of cooperating, independently from the decision of the remaining entities involved. On the other hand, the overall outcome is better if everyone cooperates instead of defecting. This definition describes the tension between collective and individual interests (Leibo et al., 2017), but it does not considers the time dimension. In particular, (Van Lange et al., 2013) highlights that consequences of actions can affect the entities involved in the social dilemma immediately or later in the future. Hence, the original definition is revised adding this concept; usually, non-cooperative choices have a higher immediate reward, while if everyone defects the effects are negative in the long-term.

Social dilemmas properties can be analysed formalizing these problems as two-person games with two possible actions: cooperate (C) or defect (D). Therefore, there are 4 possible outcomes which are associated to a specific payoff. R (reward) is the product of mutual cooperation, while P (punishment) is obtained with mutual defection; S (sucker) and T (temptation) are the results when one player cooperates and the other defects. A game is typically a social dilemma if it satisfies the following social dilemma inequalities, which express the trade-off between cooperation and defection (Leibo et al., 2017; Macy & Flache, 2002).

1. $R > P$ mutual cooperation (CC) is preferred over mutual defection (DD)
2. $R > S$ mutual cooperation (CC) is better than cooperation from a single agent (CD)
3. $2R > T + S$ mutual cooperation (CC) is preferred than identical probability of unilateral cooperation and defection (CD or DC)

4. $T > R$ or $P > S$ unilateral defection (DC) has a better payoff than mutual cooperation (CC) or mutual defection (DD) results better than unilateral cooperation (CD)

The following table clarifies the four payoffs described (Leibo et al., 2017).

	C	D
C	R, R	S, T
D	T, S	P, P

Considering these conditions, defection is the dominant strategy; however, if both players choose this option, the final outcome is the worst possible for everyone. Hence, the resulting Nash Equilibrium is named deficient (Anastassacos & Musolesi, 2018; Dawes, 1980).

Three classic examples of social dilemmas are the Prisoner's Dilemma, Chicken and Stag Hunt. The typical payoff matrices are shown below.

	C	D
C	3, 3	0, 4
D	4, 0	1, 1

Table 1 - Prisoner's Dilemma.

	C	D
C	3, 3	1, 4
D	4, 1	0, 0

Table 2 - Chicken.

	C	D
C	4, 4	0, 3
D	3, 0	1, 1

Table 3 - Stag Hunt.

However, apart from these simple situations, there are more complicated social dilemmas which are characterized by more articulated solutions and behaviours. First of all, these classic examples have also an iterated form which allows the development of different strategies and possibly the emergence of cooperation. It is worth noting that, if the number of repetitions is known, then the last execution is equivalent to an independent game. Hence, using backward induction, the overall strategy corresponds to the Nash equilibrium of a single game (Sandholm & Crites, 1996). On the other hand, if the number of iterations in unknown, then other strategies emerge. For example, in the Iterated Prisoner’s Dilemma (IPD) an interesting strategy is Tit-For-Tat (TFT) which consists in cooperating in the first interaction and then copying the previous action of the opponent. This policy favours the emergence of cooperation since defecting behaviour is punished by the opponent in the following round (Sandholm & Crites, 1996). In iterated or multi-steps games, cooperation and defection are properties of overall policies, instead of single actions (Leibo et al., 2017).

In addition to iterated games, there are many other interesting social dilemmas. Two examples are the Centipede Game and Gathering. The first one is a finite move two players game (McKelvey & Palfrey, 1992) which was introduced by (Rosenthal, 1981). On the other hand, Gathering is described in (Leibo et al., 2017) and it is a grid world game where agents should collect apples. Therefore, this game is a clear example of the tragedy of the commons, introduced by (Hardin, 1968).

2.5.1 Centipede game

The Centipede game is a two players game where the subjects should decide how to share a growing sum of money (Smead, 2008). In particular, the game is originally presented in extensive form and the two players take turns (Nagel & Tang, 1998; Rosenthal, 1981). At every step, the player who has to choose an action can either “take” or “pass”. Taking means obtaining the largest portion of the money, while passing entails the growth of the total sum. In the following turn, the other player has the same options. The game continues in this way until one of the two subjects takes the money or the maximum number of steps is reached. If this second situation occurs, the sum of money is divided in a predetermined way. In the classic formulation of this game there are one hundred turns, this explains the name (Smead, 2008). An example of rewards and its scheme is presented in the following.

The initial total payoff x is \$2; each player, during his turn, can choose to take t the largest part of the money or pass p. If the player passes, the sum is increased by \$2; on the other hand, if the chosen action is “take”, the player obtains $y = \frac{1}{2}x + \$1$, while the other agent receives the remaining part $x - y$. The game ends after 6 turns, if no one decided to take before; if that happens, the money is divided in a predetermined manner, for example as if the player in the next turn decided to take (Smead, 2008).

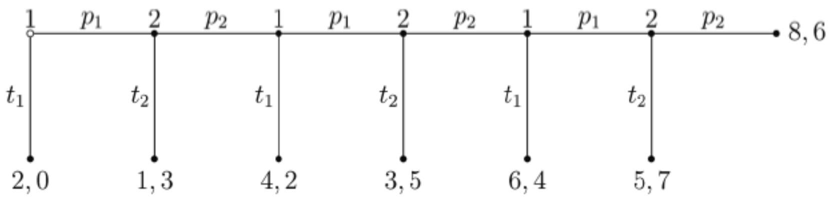


Figure 2 - Six stage Centipede Game in the extensive form (Smead, 2008).

Considering the features of this game, the Nash equilibrium is taking immediately for the first player. Indeed, since the number of steps is known, player 2 should take in the last turn; hence, player 1 would have been better off taking the money in the previous step. This reasoning can be iteratively applied until the first step leading to the result mentioned (McKelvey & Palfrey, 1992; Rand & Nowak, 2012; Smead, 2008). Therefore, the emergence of cooperation against the game theory result has been studied under different points of view.

The Centipede game can be also considered in its normal form, which is equivalent to the extensive one (Nagel & Tang, 1998). More specifically, in this case the players choose an action at the same time; the payoff growth can remain the same.

2.5.2 Gathering

The Gathering game proposed by (Leibo et al., 2017) involves two agents moving within a grid world and picking apples. Every time that an apple is collected, the agent receives a reward; then, the apple respawns after a certain time. Hence, the goal of the agent is to accumulate as many apples as possible. However, the dilemma appears because there are two agents in the same environment with identical conflicting objective. Apart from moving in the four directions, each agent can also use a beam to tag the opponent; if the other agent is tagged twice, it is temporarily removed from the environment. The use of the beam is not associated to any reward; therefore, this action has the only goal of removing the opponent to be able to collect more apples. As a consequence, the agent’s policy can be classified as cooperative or defecting depending on the frequency with which the beam is applied. Changing the respawn rate it is possible to control the number of apples in the environment, while modifying the timesteps in which the tagged agent is removed from the environment influences the potential rise of conflicts.

The game described above is only an example of a broad category of games where there is a shared resource, and two or more agents compete for it. Other examples are Harvest and Cleanup, which are presented in (Jaques et al., 2019).

3 Approach

The main goal of this work is to implement an agent capable of learning the optimal behaviour in a multiagent environment taking into account the information obtained on the opponent's strategy. This objective involves multiple elements that should be analysed and integrated. The overall ambition is obtaining an agent which is able to learn the proper behaviour without any prior knowledge of the environment or the opponent. More specifically, by changing some parameters, the agent should adopt the values of its adversary, cooperate with other entities or exploit them. All the agent's components are analysed in detail below.

First of all, the agent must learn the optimal policy in an unknown environment without any prior knowledge. Indeed, the agent does not have any information before starting to interact with the environment. Therefore, the Reinforcement Learning paradigm is chosen. More specifically, the Q-Learning algorithm results appropriate. As a matter of fact, Temporal Difference procedures do not require a model of the environment and they immediately update the value function; moreover, Q-Learning directly computes the optimal policy. Hence, this algorithm is the most suitable solution for this real-time learning task. Moreover, the agent should be as general as possible, thus a tabular method cannot be applied since normal-size tasks are too complicated for it. As a consequence, a Deep Q-Learning agent is the best solution.

Once the basic structure is defined, the agent must be placed in a multiagent environment. To simplify the interactions, only two entities were considered; however, the implemented agent can be theoretically extended to interact with more opponents. Considering the goal of creating a very adaptable agent which is able to cooperate or exploit the adversary, depending on parameters values, individual learning should be considered. In particular, the task can be classified as a mixed task since the agent sometimes cooperates sometimes competes. Therefore, the agent learns its policy independently in the common environment. The other entity is perceived as part of the environment by the learning agent. This might introduce instability in the learning process; however, it is worth noting that if the opponent's policy is stationary, then the multiagent MDP is equivalent to a single-agent MDP (He et al., 2016). This facilitates learning since the source of non-stationarity introduced by the other agent is removed.

The central problem of this work is how to infer the opponent's strategy; this question is characterized by a broad interpretation since it can be also considered as a value alignment task. After evaluating many different approaches presented in the literature, a possible solution is applying Inverse Reinforcement Learning. Indeed, the agent can understand the opponent's intentions only through its experiences; hence, opponent modelling methods which require previous knowledge must be excluded. Furthermore, considering the versatility of the agent, the most significant information on the opponent's behaviour is its reward function. Therefore, recreating the reward that the opponent is maximizing provides the essential clues on its preferences. Then, this information can be used to adopt the opponent's values, cooperate with the agent, or exploit it. Among the IRL algorithms, Maximum Entropy IRL results particularly suitable since it considers that the demonstrations provided by the opponent might be imperfect and subject to noise.

As mentioned before, recreating the opponent's *preferences* can be also considered as a value alignment problem. More specifically, the agent autonomously infers the values and intentions of the other entity and it is intrinsically motivated to act consistently with them. Therefore, the opponent's reward obtained with the Maximum Entropy IRL algorithm can be considered as an intrinsic reward for the learning agent. In particular, the total reward used by the DQN algorithm, to train the agent, is the composition of the extrinsic and intrinsic reward. Thus, this solution differs from many approaches in the literature, where the opponent's strategy is typically taken into account changing the learning algorithm. Here, the agent's learning rule is not modified, and the opponent's presence is considered in the agent's reward. This overall reward is defined as a linear combination of the two rewards, where the extrinsic reward comes from the environment, while the intrinsic one is the opponent's reward computed with MaxEnt IRL.

$$r_{total} = \alpha \cdot r_{extrinsic} + \beta \cdot r_{intrinsic}$$

The parameters α and β can be interpreted as two scaling factors. Changing their values, it is possible to study the combination that produces the best outcome in the agent's learning. More specifically, the opponent's reward should only bias the reward coming from the environment, to preserve the Q-learning agent behaviour.

Finally, the environment of social dilemmas was chosen because many real problems have these properties. Consequently, these settings appear to be interesting and challenging situations where training the developed agent. Moreover, these problems can be interpreted as games which are analysed by many different disciplines. Hence, results in this area can be useful in various fields of study.

4 Evaluation

The approach presented in the previous section is now implemented concretely. The following paragraphs describe the realization details and how the agent is evaluated.

4.1 Implementation

The learning algorithm chosen for the agent is Deep Q-learning, as illustrated before. In terms of implementation, this method is widely used and there are many libraries that provide an optimized realization. The proposed agent uses the library (*TensorFlow Agents*) which offers many Reinforcement Learning agents, ready to be used and trained. More specifically, the *DqnAgent* is adopted as base agent; it is then elaborated to obtain the desired characteristics. Since this library is not specifically realized for a multiagent setting, an additional object, named collector, is created to properly collect the agents' interactions. This collector receives the actions that the two agents want to execute on the environment, it collects the results from the environment and then it communicates them back to the agents. During these passages, the collector rearranges the data so that it is understandable for agents and environment. As required by the (*TensorFlow Agents*) library, the environment is completely described by a Time Step at every point of time; this is the output obtained when the collector communicates the agents' actions to the environment.

Regarding the intrinsic reward, this is computed applying the Maximum Entropy IRL to a set of demonstrations. In particular, during the training, the DQN agent interacts with its opponent and these experiences are collected in two buffers: the experience replay buffer and the demonstrations buffer. The first one is used by the *DqnAgent* to obtain the optimal policy; every experience is a triple composed by the initial Time Step, the agent's action, and the following Time Step. On the other hand, the demonstrations buffer is required by the Maximum Entropy IRL algorithm; each demonstration is a complete execution of the game and it consists in a set of trajectories. Every trajectory is a tuple composed by the initial state of the environment, the action chosen and the following state (here Time Steps are not required). The opponent's reward, associated to the current state and obtained with the MaxEnt IRL algorithm, is combined with the environment reward by the collector. This overall reward is then associated to the specific agent's action and stored in the proper Time Step, which forms an experience in the experience replay buffer.

Regarding the Maximum Entropy IRL implementation, this algorithm requires the transition dynamics as input. These dynamics are a probability distribution describing the likelihood of experiencing a certain state, given the probability of choosing a specific action in the previous state; this distribution is empirically computed from the agent's experiences. This approach is feasible since the environments considered are relatively simple; however, a model-free IRL algorithm could be used in a future version.

4.2 Experimental settings

The implemented agent is initially tested on a simple environment represented by the Centipede game in its normal form. Indeed, the structure of this game is similar to the classic Prisoner's Dilemma, but each execution has more than one step, allowing more complicated strategies. Recalling that the agent's overall reward is $r_{total} = \alpha \cdot r_{extrinsic} + \beta \cdot r_{intrinsic}$, the goal of these tests is to investigate how the agent's behaviour changes by varying the value of the parameter β that multiplies the intrinsic reward. Moreover, a study on a population of agents is also conducted for this game. Considering different compositions of the population, the objective is analysing the emerging behaviour of the learning agents.

After a deep examination of the Centipede game environment, the agent is applied to a more complicated situation, the Apple Picking game. More specifically, this game is a variation of Gathering, which was described in the previous sections. A similar study on the β parameter is realized for this game as well.

Complete details of the two environments and the agent's parameters are provided in the following paragraphs.

4.2.1 Simulation Environments

Both the simulation environments are slightly modified to be suitable for the goal of the tests. They are presented in detail below, along with a description of the state representation that the agent receives.

4.2.1.1 Centipede game

In the Centipede game environment two players have to choose how to split a growing sum of money, as described previously. In particular, the experiments were executed using the normal form of the game and the maximum number of steps is six. Thus, the two players select the action ("take" or "pass") simultaneously and the game continues until at least one agent selects "take" or the sixth step is reached. Compared to the original formulation, the only change refers to how to divide the money on the last step. More specifically, if both agents select "pass" on the sixth step, the amount of money is equally shared, while if both choose “take” there is a penalty, and each agent receives half of the sum minus the penalty. Indeed, the goal was clearly differentiating a cooperative policy from a defecting one; with this mechanism of dividing the total sum, a cooperative strategy which lasts until the end is rewarded, while a defecting one has the additional risk of being penalized if the opponent behaves in the same way.

The starting amount is 2, every step it grows by 2 and the splitting rule is $y = \frac{1}{2}x + 1$ for the defecting agent and $x - y$ for the cooperative one; the penalty is 1 on each agent. The following table illustrates the agents’ payoffs in every situation; if anyone takes, the game terminates immediately.

		Step 1		**Step 2**		**Step 3**	
		Total: 2		Total: 4		Total: 6	
		Agent 1		Agent 1		Agent 1	
		Pass	Take	Pass	Take	Pass	Take
Agent 2	Pass	0, 0	0, 2	0, 0	1, 3	0, 0	2, 4
	Take	2, 0	0, 0	1, 3	1, 1	4, 2	2, 2
		Step 4		**Step 5**		**Step 6**	
		Total: 8		Total: 10		Total: 12	
		Agent 1		Agent 1		Agent 1	
		Pass	Take	Pass	Take	Pass	Take
Agent 2	Pass	0, 0	3, 5	0, 0	4, 6	6, 6	5, 7
	Take	5, 3	3, 3	6, 4	4, 4	7, 5	5, 5

Table 4 - Centipede game payoffs.

Regarding the state representation, each agent receives an array of four elements representing the current step number, the previous action of agent 1, the previous action of agent 2, and the current sum of money.

4.2.1.2 Apple Picking

The Apple Picking game is a simplification of Gathering from different points of view. First of all, apples do not respawn; when the agents collect all of them, the game terminates. Moreover, the agents can only move inside the grid world to collect apples, thus the use of the beam is not considered in this version. Because of that, another criterion is required to determine if a policy is cooperative or not. For this reason, the grid world is symmetrically structured: the agents always start from opposite positions and the total number of apples is divided so as to have half of the apples closer to one agent and half closer to the other. Therefore, two agents cooperate if they collect apples in their half, equally splitting them. On the other hand, if one agent tries to obtain all the apples available, its behaviour is considered non-cooperative.

The following scheme describes the initial situation of every Apple Picking game. There are 2 agents and 6 apples; the agents can move freely inside the 4×4 grid, but they cannot go outside.

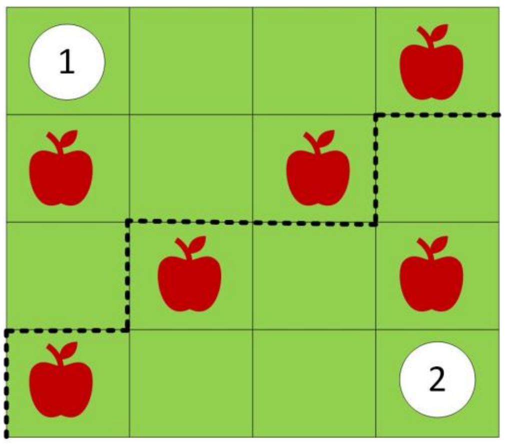


Figure 3 - Apple Picking initial state.

The dotted line represents the ideal division of the grid world, illustrating which apples each agent should harvest. When the apples terminate, the game ends; however, another stopping criteria is introduced to always obtain finite games: after executing 100 actions, the game is over.

Each agent has four different actions available: up, down, right and left. The agent obtains a reward of 10 for every apple collected, while if it tries to move outside the grid, it receives a penalty of -0.25. Moreover, in order to encourage the agents to harvest apples as quickly as possible, every empty cell is associated to a penalty of -0.1. The same punishment is applied if an agent tries to access a cell which is already occupied by the other agent. Finally, to solve the conflict that may arise if both agents choose an action that leads them to land on the same cell, agent 1 is considered as acting first only in this specific case. Therefore, it is the one that occupies the position after the turn.

In terms of state representation, each agent receives a 4×4 array describing the position of agents and remaining apples.

4.2.2 Simulation Parameters

The learning agent's implementation entails the definition of many parameters and properties. First of all, the *DqnAgent* from the (*TensorFlow Agents*) library requires the specification of several values. For some of them, the choice was selecting standard figures since they are reasonable for this situation. These main parameters are the following:

- $replay\_buffer\_max\_length = 100000$, this value specifies the maximum number of experiences saved at the same time in the experience replay buffer;
- $batch\_size = 64$, this is the dimension of the mini-batch of experiences sampled from the replay buffer for one training step;
- $learning\_rate = 10^{-3}$ defines the learning rate of the Adam optimizer, which is the optimizer used to train the *DqnAgent.*

In addition to these elements, the *DqnAgent* requires further parameters which were chosen during some preliminary trainings:

- $epsilon = 0.2$ is the epsilon-greedy parameter. More specifically, the *DqnAgent* adopts an epsilon-greedy collect policy; this means that the agent selects the best next action most of the time (probability $1 - \varepsilon$), while a random action is chosen with probability ε. Therefore, this figure regulates the exploration level of the learning agent.
- $gamma = 0.99$ is the discount factor of the Q-learning algorithm which expresses the present value of future rewards.

Furthermore, during the Centipede game training focused on the behaviour of the single agent against different deterministic opponents, two additional parameters were specified to improve the stability of learning: $target\_update\_period = 20$ and $gradient\_clipping = 10$. These two values indicate the frequency (in terms of training steps) with which the target network of the *DqnAgent* is updated and the norm length to clip gradients. These parameters only favour the learning stability; thus, they are used exclusively in the simulations where the goal is studying in detail the strategy of a single agent. On the contrary, standard figures for them are chosen in the Centipede game population trainings.

Finally, the most important parameter for the *DqnAgent* describes the neural network structure. In particular, the two games examined have different complexities; therefore, the structure of the neural networks is modified accordingly. The Centipede game has very few states and only two actions; hence, a single layer neural network with 64 nodes is enough. On the other hand, the Apple Picking environment is significantly more complicated, thus a two layers neural network with 64 nodes for each layer is chosen.

Apart from the Reinforcement Learning paradigm, the Maximum Entropy IRL algorithm requires the definition of different parameters as well. These are described in the following and chosen during another preliminary study. It is worth noting that the difference in complexity between the two games entails different values for some of these parameters:

- $alfa = 0.01$ is the step size of the Gradient Descent optimization algorithm, this value should be small enough to allow the optimum to be reached, but not too small otherwise the algorithm results very slow. This figure is adopted for both the environments.
- $num\_iterations\_max\_entropy$ indicates the number of iterations of the Maximum Entropy IRL algorithm. Indeed, according to the stopping condition previously illustrated, the algorithm iterates until the gradient is sufficiently close to 0 or it is not possible to obtain further progress. Hence, some specific trainings were completed to understand how many iterations were required to satisfy this condition. For the Centipede game, the chosen value is $num\_iterations\_max\_entropy = 100$, which guarantees the achievement of the stopping condition with a large margin. However, this figure is quite high; therefore, for the Apple Picking game the selected value is $num\_iterations\_max\_entropy = 50$, to reduce the execution time.
- $max\_demonstrations$ specifies the maximum number of demonstrations collected in the demonstrations buffer and it depends on the complexity of the game. As a matter of fact, the number of examples required to infer the opponent's reward increases with the complexity of the task. As a consequence, the chosen figures are: $max\_demonstrations = 50$ for the Centipede game, while $max\_demonstrations = 100$ in the Apple Picking environment.

Furthermore, recomputing the cost vector of the opponent every time that a new demonstration is acquired is not particularly meaningful, because the values will not probably change significantly. In addition to it, this approach is not efficient. Hence, a further parameter is added to specify the frequency of execution of the Maximum Entropy IRL algorithm. In particular, the value selected corresponds to the $max\_demonstrations$ figure: the cost vector is recomputed only when the demonstrations buffer is full of completely new examples.

Finally, applying the Maximum Entropy IRL algorithm involves the definition of the feature function. More specifically, given the current state, this function returns a set of features associated to it; the opponent's cost vector is a weighted combination of these features. As already mentioned, a possible solution is choosing raw features which means using directly the state representation. For the Centipede game case, this is the best choice, since the state is very simple, and it already contains the essential elements that influence the reward. On the other hand, for the Apple Picking environment a custom function might be suitable. In particular, the initial hypothesis was extracting the following information:

$$(num.\ apples,\ \ distance,\ \ up,\ \ down,\ \ right,\ \ left)$$

where $num. apples$ is the number of remaining apples in the grid world, $distance$ is the distance from the closest apple and the last four elements represent the direction of the closest apple expressed considering the orientation components. However, these features do not produce satisfying results, as shown in the following. Therefore, raw features are adopted also for the Apple Picking game.

All the previous parameters are set for all simulations; on the contrary, the properties that vary during the tests concern the type of opponent and the multiplying factor of the opponent's reward. For the Centipede game population study, the composition of the population is changed in each test. Regarding the type of opponents, the Centipede game trainings applied two different agents: an always cooperative entity and a defecting one. More specifically, the first one always plays "pass", while the defecting agent choses "pass" until the last step, when it selects "take". For the Apple Picking environment, only one deterministic agent was used; the following image illustrates its sequence of actions.

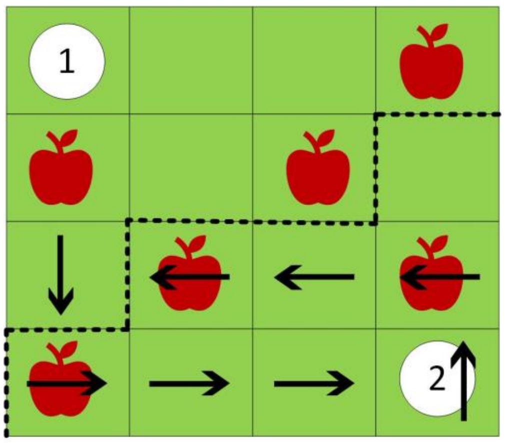


Figure 4 - Strategy of the deterministic agent in the Apple Picking game.

The strategy illustrated in the figure above supposes that the deterministic agent is agent 2, which starts from the lower right cell and then selects the actions described by the arrows. Hence, following the previous definition of cooperation and defection for this game, this agent is cooperative since it only harvests apples in its half of the grid world. This sequence of actions is repeated until one of the two stopping criteria is met.

Regarding the opponent's reward parameter, different possible values are explored. The resulting behaviours and the considerations relating to them are illustrated in the following.

4.3 Results representation

In order to study the learning performance and understand the predominant policy, the most significant measure is the average reward. Therefore, this quantity and its standard deviation are computed for all the simulations. In addition to it, the final policy is also generally collected, especially for the trainings where this was not deductible from the average reward.

Finally, the Centipede game population trainings require an additional method to represent the results. The idea was producing graphs which are similar to the replicator dynamics[3] illustrated in (K. Tuyls & Stone, 2018). More specifically, the most interesting policies that have been identified are "Always pass", "Always take", and "Take last step". The first one characterizes a cooperative agent which always selects "pass", whereas the remaining two are defecting policies. In "Always take" the agent chooses "take" immediately, while in "Take last step" the entity passes until the last step, when it selects "take". This last strategy is particularly worthy of note because it is the optimal policy against a cooperative agent: it guarantees to obtain the highest amount of money. In order to represent the evolution of the policies among the agents, every time that a loop of game executions is completed the final policy of each agent is evaluated. In particular, in the Centipede game population trainings, each agent plays against all the remaining agents following a round robin mechanism. In a single simulation, this loop of games is executed a chosen number of times. At the end of each loop, the agents' policies are classified in "Always pass", "Always take", "Take last step", and "Other" (this last category has been introduced to express all the strategies which cannot be defined as belonging to the first three classes). Therefore, depending on the number of loops in every simulation, it is possible to have an estimate of how many agents choose each policy over time. These data are then displayed on a tetrahedron, where each vertex represents a policy. Every point on the graph expresses

[3] Replicator dynamics describe how the population behaviour evolves over time (Cressman & Tao, 2014). These game dynamics are computed using the replicator equation, introduced by (Schuster & Sigmund, 1983; Taylor & Jonker, 1978), which is an important differential equation in the field of Evolutionary Game Theory.

the policies composition at that moment of the simulation. These points are connected by arrows to clarify the direction of change. A concrete example and a graph are illustrated below to clarify this representation.

In the simulations realized, the number of complete loops of games is 20. Therefore, for each simulation there are 20 policy evaluation and the starting point. The position of each point in the graph expresses the policies composition in terms of the percentage of agents which apply “Always pass”, “Always take”, “Take last step”, and “Other”; the arrows explain the direction of variation over time.

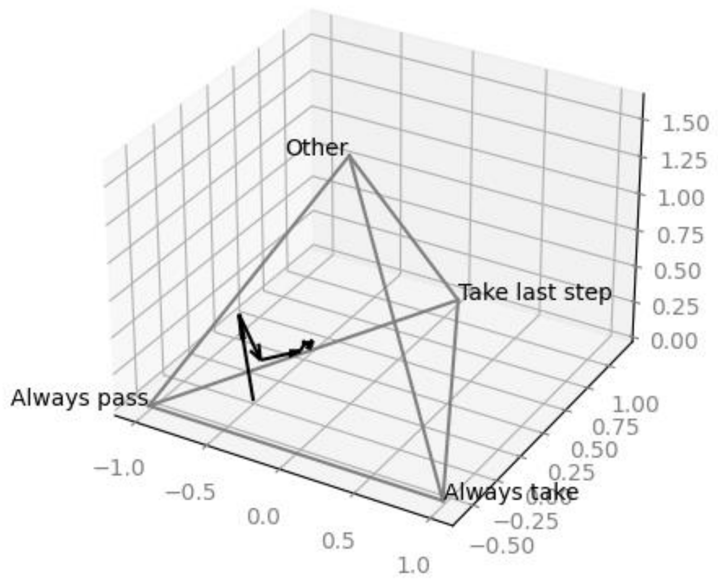


Figure 5 - Centipede game population simulation: example of the policy evolution graph.

4.4 Experimental results

The results of different simulations executed on the Centipede game and the Apple Picking game are collected in this section. In particular, the outcomes are subdivided depending on the game used as environment for the developed agent. All tests were realized preforming 10 simulations and considering the average values, unless otherwise specified.

4.4.1 Centipede game

The first set of experiments consists in training a classic *DqnAgent* against two deterministic agents in order to study the strategies learnt by the pure Reinforcement Learning agent and provide a baseline for the following trainings.

The first group of simulations involves the pure *DqnAgent* and a deterministic agent which plays "Always cooperate", thus it always selects "pass". The game is iterated 5000 times in each training, because the agent reaches its maximum reward in this time span.

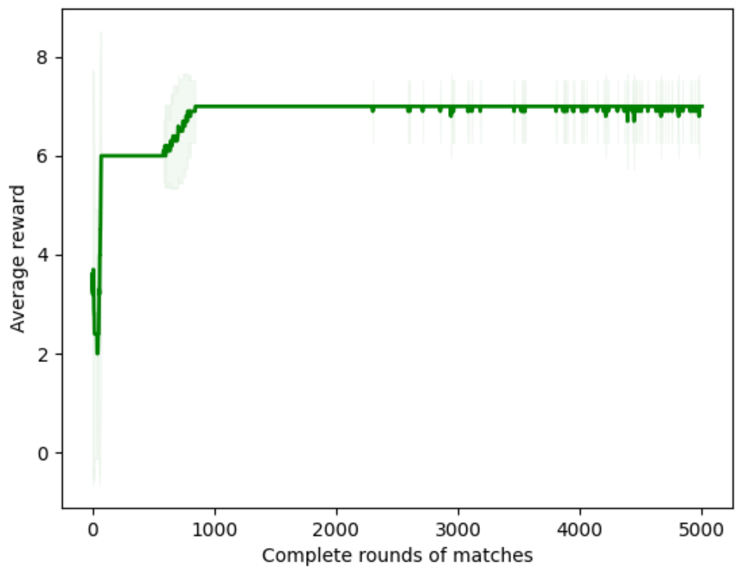


Figure 6 - Centipede game: pure DqnAgent against an always cooperative agent.

According to the graph, the average reward is practically stable at the value of 7. This clearly means that the Reinforcement Learning agent learns to defect its opponent after 1000 rounds and the learnt strategy is "Take last step". Therefore, the RL agent behaves as expected and it reaches its optimal policy quickly.

The same *DqnAgent* was tested against a deterministic agent that plays the strategy "Take last step", which consists in passing until the last step and choose "take" then. However, playing against this agent results harder than the case of the cooperative agent and the learning oscillates significantly. For this reason, the game is iterated 20000 times in each training.

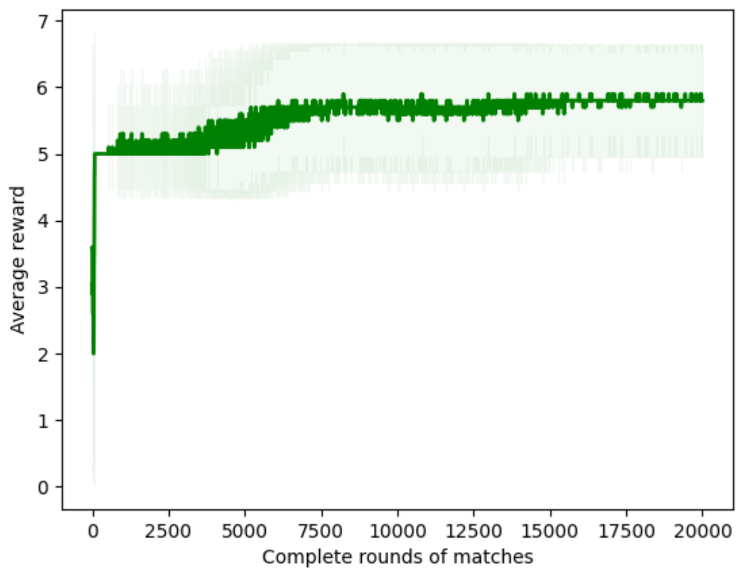


Figure 7 - Centipede game: pure DqnAgent against an agent which plays "Take last step".

As shown in the image, after 13000 rounds the average reward is approximately stable on 6. Considering the final states collected in each simulation, the agent learns to take the sum of money in the second to last step in 80% of the cases; thus, the *DqnAgent* understands how to avoid being exploited by a defecting agent which waits until the last step to choose “take”. However, this strategy results harder to be learnt because significantly more iterations are required.

The previous experiments provide the baseline behaviour of the pure *DqnAgent*, this allows to understand if the introduction of the intrinsic reward in the agent improves its performance towards the desired goal. The following simulations involve the developed agent and the cooperative deterministic entity which plays “Always pass”. As previously clarified, the agent’s overall reward is $r_{total} = \alpha \cdot r_{extrinsic} + \beta \cdot r_{intrinsic}$; hence, the objective is studying different multiplying factors β for the intrinsic reward in order to obtain cooperation or defection, while α is set to 1. More specifically, the intrinsic reward represents the opponent’s reward, and the adversary plays a cooperative strategy; thus, cooperation is favoured if the opponent’s reward is added with a positive multiplying factor to the agent’s extrinsic reward. Indeed, in order to have a cooperative behaviour in the developed agent, the overall reward function should be positively influenced by the rewards of a deterministic cooperative agent. On the other hand, if the agent wants to exploit a cooperative enemy, the opponent’s reward should be subtracted from the agent’s extrinsic reward. The key idea is that the best situation for a cooperative agent is probably not convenient for an exploiting agent. A similar reasoning can be adopted in the case of a non-cooperative opponent. Given these considerations, the following graphs show the behaviour of the developed agent for different values of the multiplying parameter β.

The emergence of cooperation was firstly examined; the graph below illustrates the average reward for different positive values of β. More specifically, only the most significant values of β are represented, among all the figures tested.

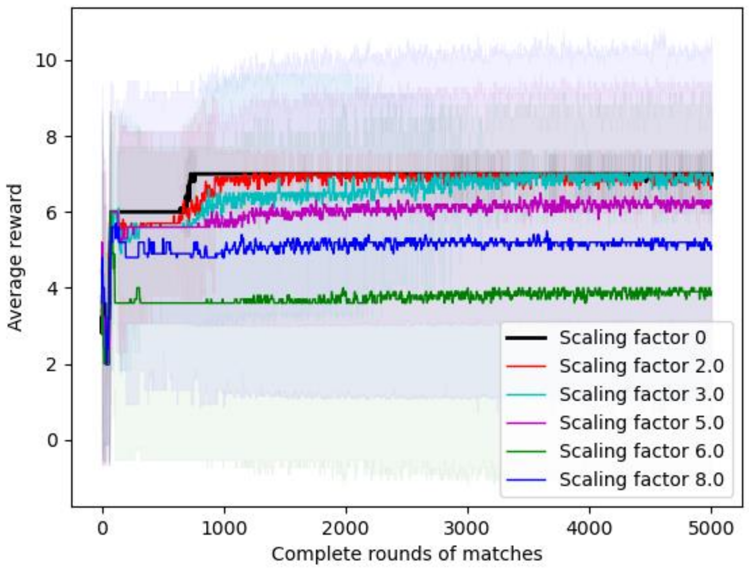


Figure 8 - Centipede game: developed agent against cooperative agent, cooperation case.

According to the image, a multiplying parameter $\beta = 5.0$ produces an average reward which is approximately 6. Therefore, using the cooperative opponent’s reward as intrinsic reward favours the emergence of cooperation. Indeed, the pure RL agent would learn to exploit its opponent without this additional information, as shown by the black line which represents the baseline behaviour.

On the other hand, the effect of this parameter was studied also for the opponent’s exploitation. In particular, the pure RL agent learns to exploit the cooperative adversary. Therefore, the developed agent can be considered better than the *DqnAgent* if it reduces the number of iterations required to reach the stable defecting policy. Different values of β were tested between $\beta = 0$ and $\beta = -5$, the average reward for some of them is illustrated below.

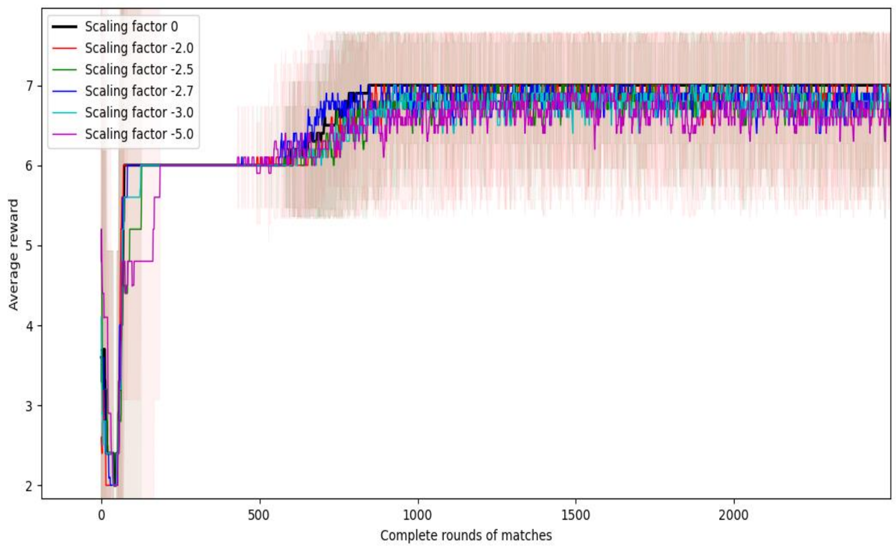


Figure 9 - Centipede game: developed agent against cooperative agent, defection case.

As shown in the graph, the only parameter that entails a steeper increase of the average reward, compared to the baseline, is $\beta = -2.7$. Moreover, the presence of the intrinsic reward produces a fluctuating behaviour; this instability is probably due to the approximation introduced by the Maximum Entropy IRL algorithm, as analysed in the following. An additional graph which compares less parameters and zooms on the interesting region is provided below.

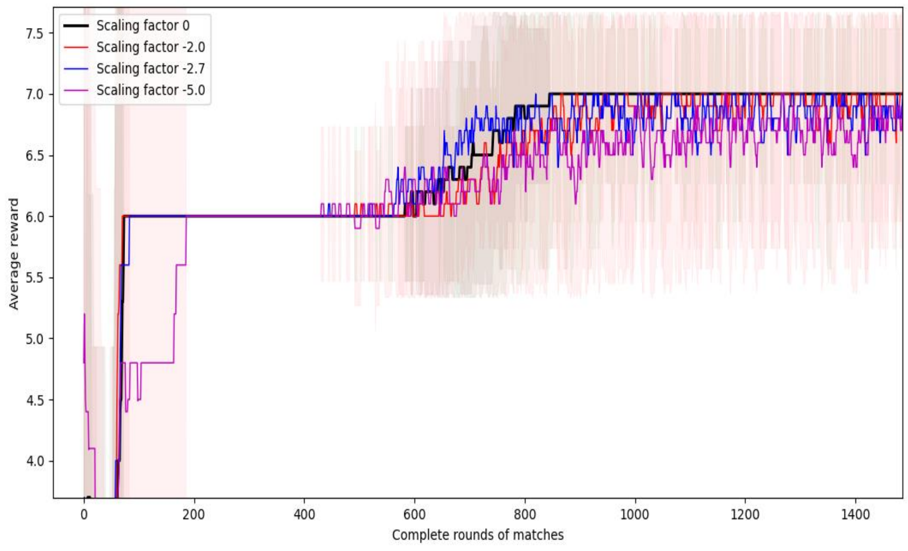


Figure 10 - Centipede game: developed agent against cooperative agent, defection case – zoom.

Even if $\beta = -2.7$ produces an average reward which increases slightly faster than the baseline, then it is significantly instable compared to the pure RL agent case. Therefore, the use of the intrinsic reward does not appear particularly useful in this case. This might be due to the simplicity of the game; indeed, the pure RL agent learns the optimal policy that maximizes the reward, which corresponds to the "Take last step" strategy. However, since the number of states and actions is considerably limited, the pure RL agent reaches the optimal policy very quickly. Hence, obtaining a significant improvement with the opponent's reward is not easy. Moreover, the Centipede game is not a zero-sum game; therefore, the opponent's worst situation might not correspond to the agent's best outcome.

In addition to the cooperative opponent, the other adversary chosen to test the exploiting behaviour of the developed agent is a defecting entity which plays "Take last step". As illustrated before, learning against this agent results harder, thus a higher number of iterations is required.

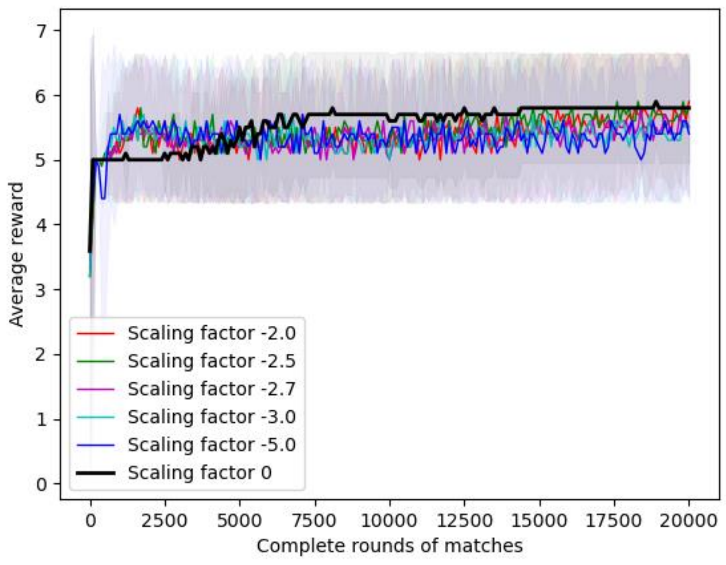


Figure 11 - Centipede game: developed agent against non-cooperative agent, defection case.

According to the graph, the use of the opponent's reward favours a higher average reward at the beginning of the training, probably because the agent can faster converge to a more convenient strategy using this information. However, this decreases under the baseline in the following iterations; this behaviour might be explained considering that the opponent's worst outcome does not necessarily correspond to the agent's best state. For some values of β, the average reward results approximately similar to the baseline towards the end of the simulation.

Another chart with the most significant figures for β is provided to clarify the described tendency.

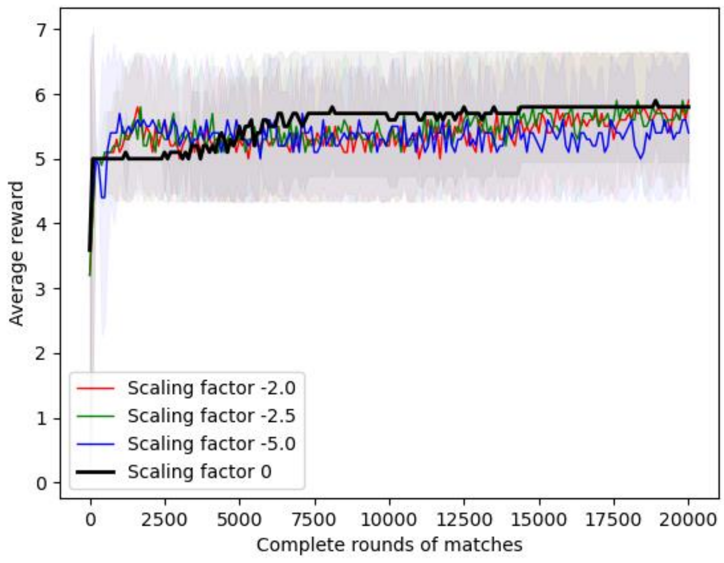


Figure 12 - Centipede game: developed agent against non-cooperative agent, defection case - significant values.

As illustrated in the image, $\beta = -2.5$ produces the best behaviour since the average reward increases faster than the case $\beta = 0$ at the beginning, while it is comparable to the baseline at the end of the simulation. Moreover, considering the final states collected during the training for $\beta = -2.5$, the learnt policy consists in taking the amount of money in the second to last step in 80% of the cases. This corresponds to the behaviour of the pure RL agent.

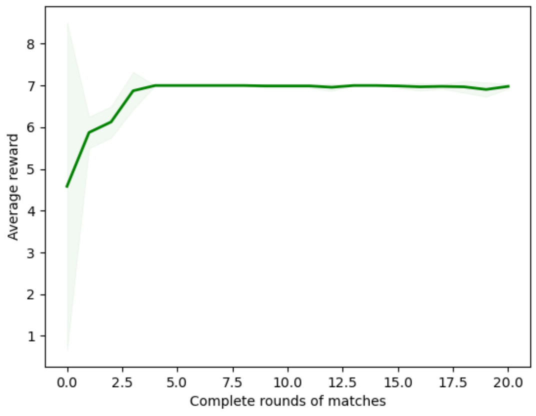
Even if the exploitation experiments did not exhibit a significant advantage in using the opponent modelling agent instead of the pure RL entity, it is worth noting that the best parameters are similar. Indeed, whether the opponent is cooperative or chooses "take" on the last step, the multiplying factor β that entails the best results is analogous: $\beta = -2.7$ and $\beta = -2.5$ respectively.

Regarding the population experiments, the goal was studying the emerging behaviour with several compositions of the population, as mentioned before. During the trainings, each agent plays against all the other ones in a round robin tournament. This loop of games is performed a defined number of times in every simulation. More specifically, four different agents were used: two deterministic entities and two learning ones. The deterministic agents play "Always cooperate" and "Take last step" respectively, while the remaining entities are the developed agent and the pure RL agent (*DqnAgent*). For the developed agent, the value chosen for the multiplying factor β is $\beta = 5.0$ and it is constant for all simulations. Indeed, a positive parameter β produces an agent which aligns with its opponent.

This means that it will tend to learn a cooperative or defecting strategy, depending on the opponent's behaviour. The specific value $\beta = 5.0$ was chosen because it produced the best cooperative behaviour in the previous experiments. Other specific parameters that were selected for these simulations are the number of complete loops executed $num\_loops = 20$, the population dimension $tot\_agents = 20$, and the number of matches played between the same couple $num\_iterations = 50$ for each turn of the round robin tournament. The following set of graphs illustrates the most significant results for different populations; both the average reward of learning agents and the overall policy evolution are displayed.

The first population examined is composed by a single RL agent and 19 deterministic cooperative agents; it provides the baseline for this type of simulations.

Average reward of learning agents



Policy evolution considering the entire population

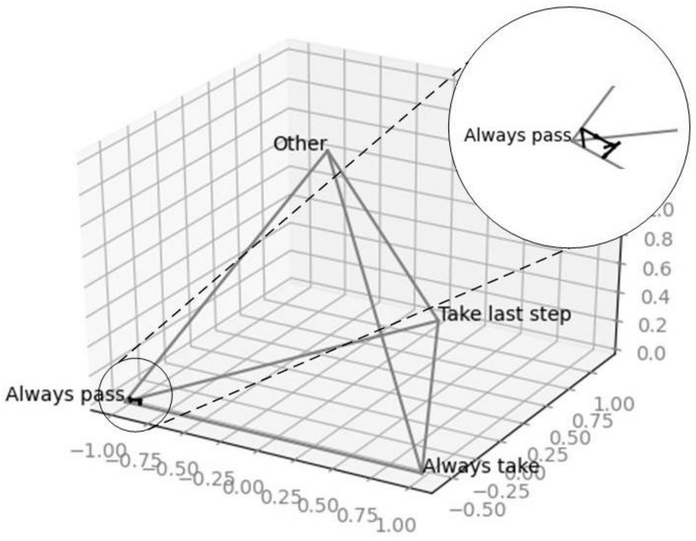
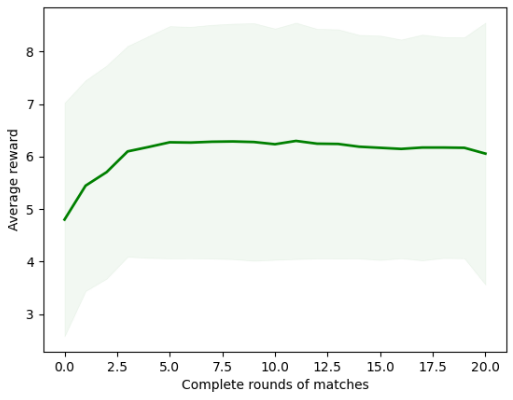


Figure 13 - Centipede game population: 1 RL agent and 19 cooperative agents.

After these basic trainings, the developed agent is introduced in the population which is thus formed by 1 RL agent, 1 opponent modelling agent and 18 cooperative agents.

Average reward of learning agents *Policy evolution considering the entire population*



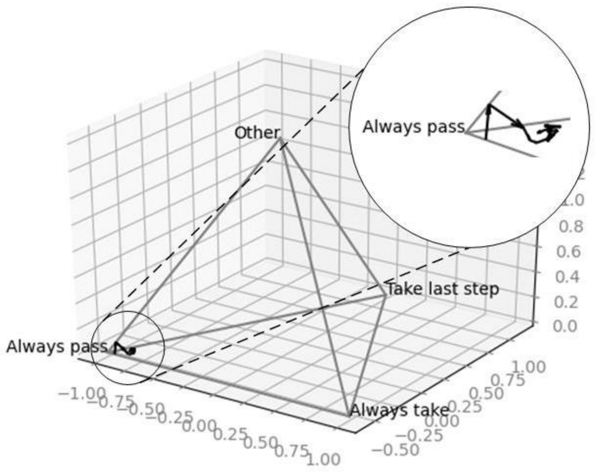
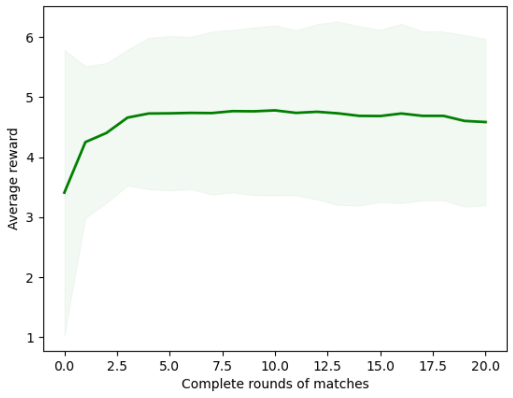


Figure 14 - Centipede game population: 1 RL agent, 1 opponent modelling agent and 18 cooperative agents.

The number of learning agent is further increased; the following graphs refer to a population of 2 RL agents, 2 opponent modelling agents and 16 deterministic cooperative agents.

Average reward of learning agents *Policy evolution considering the entire population*



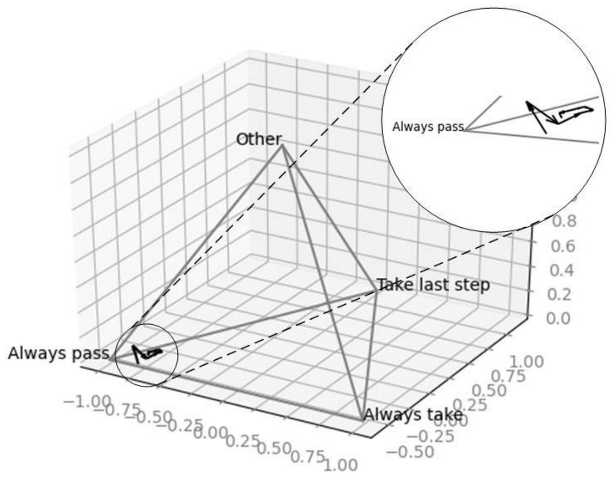
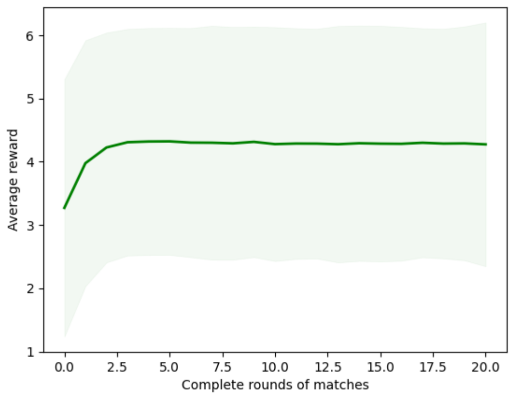


Figure 15 - Centipede game population: 2 RL agents, 2 opponent modelling agents and 16 cooperative agents.

The next images illustrate the outcomes when half of the population is composed by learning agents. More specifically, these are 5 RL agents, 5 opponent modelling agents and 10 deterministic cooperative agents.

Average reward of learning agents



Policy evolution considering the entire population

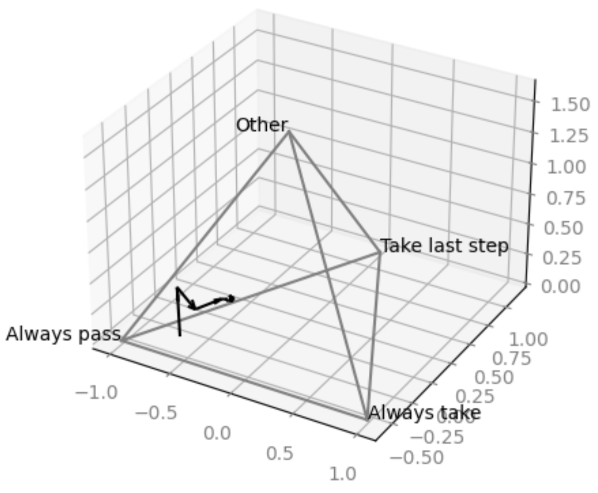


Figure 16 - Centipede game population: 5 RL agents, 5 opponent modelling agents and 10 cooperative agents.

Keeping the number of learning agents fixed, another set of experiments consists in modifying the type of deterministic agents. The following graphs refer to 5 RL agents, 5 opponent modelling agents, 5 deterministic cooperative agents and 5 deterministic defecting agents.

Average reward of learning agents *Policy evolution considering the entire population*

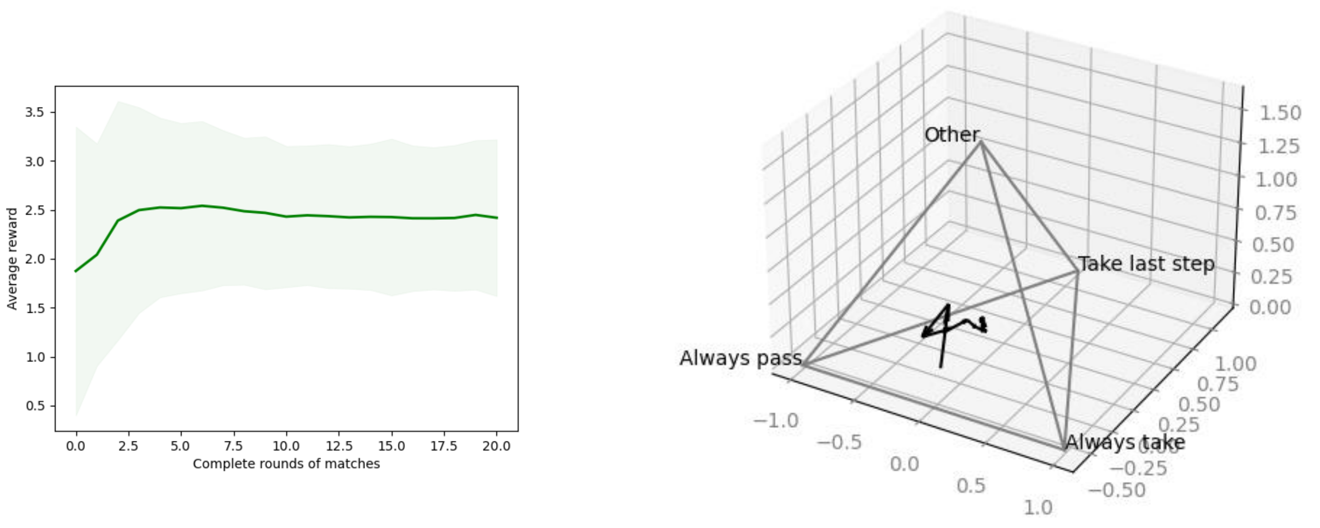


Figure 17 - Centipede game population: 5 RL agents, 5 opponent modelling agents, 5 cooperative agents and 5 defecting agents.

The number of learning agents is increased again, and the images below describe the outcomes with a population of 8 RL agents, 8 opponent modelling agents and 4 deterministic cooperative agents.

Average reward of learning agents *Policy evolution considering the entire population*

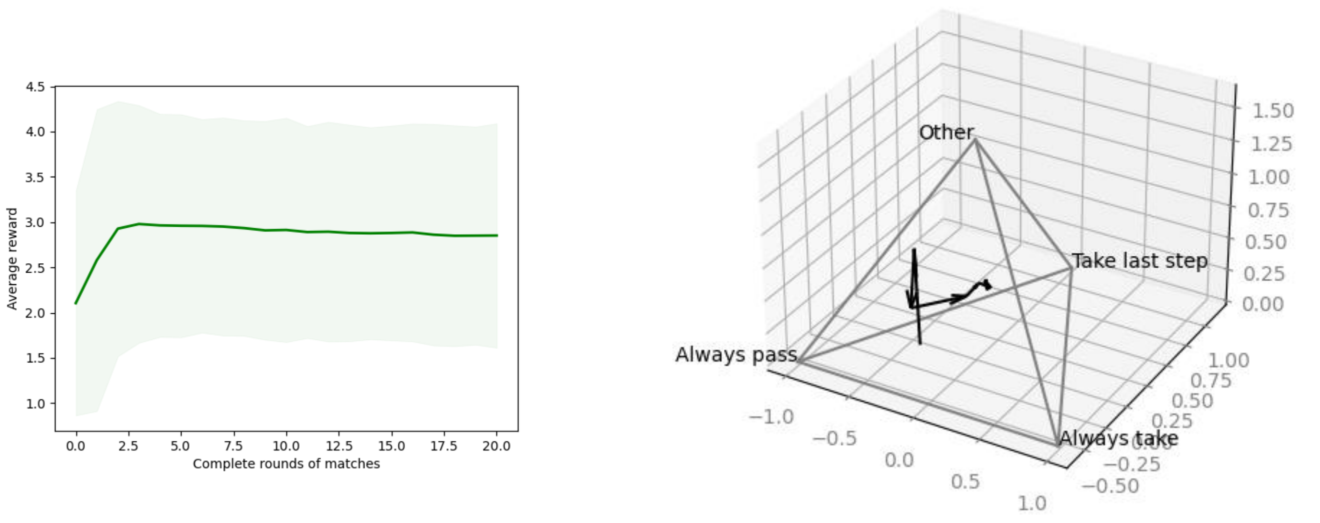
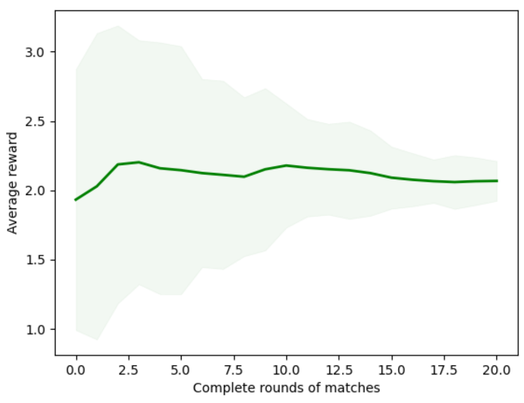


Figure 18 - Centipede game population: 8 RL agents, 8 opponent modelling agents and 4 cooperative agents.

It is also worth considering the results of the population with the same number of learning agents, but different deterministic entities. The graphs presented refer to the case of 8 RL agents, 8 opponent modelling agents, 2 deterministic cooperative agents and 2 deterministic defecting agents.

Average reward of learning agents *Policy evolution considering the entire population*



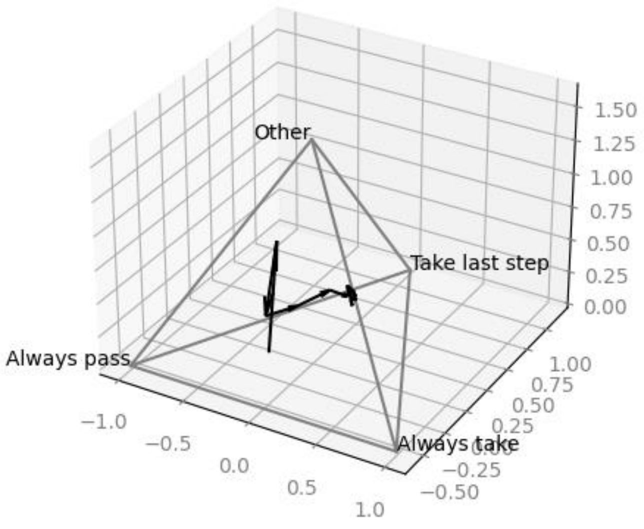
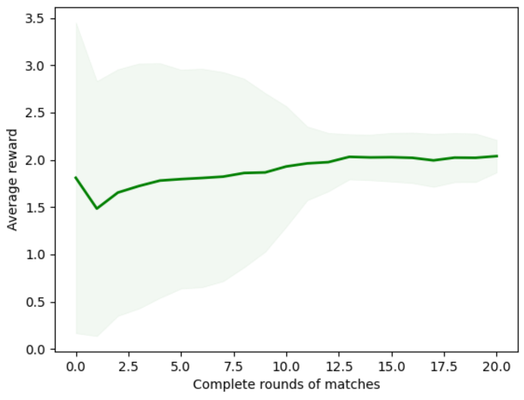


Figure 19 - Centipede game population: 8 RL agents, 8 opponent modelling agents, 2 cooperative agents and 2 defecting agents.

Finally, the last population introduced is constituted by learning agents only. More specifically, there are 10 RL agents and 10 opponent modelling agents.

Average reward of learning agents *Policy evolution considering the entire population*



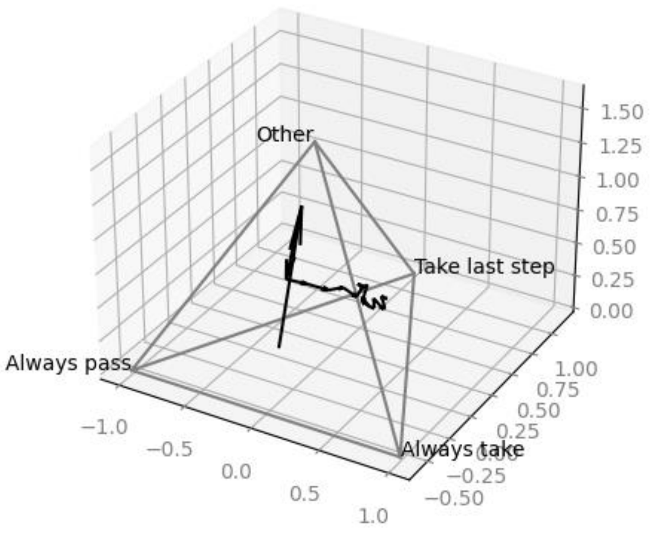


Figure 20 - Centipede game population: 10 RL agents and 10 opponent modelling agents.

Overall, considering the average reward and the policies evolution of various populations, some observations are possible. First of all, increasing the number of learning agents, the average reward is characterized by a greater standard deviation. This underlines a higher instability in the learning process, as expected. Moreover, the average rewards decrease, and the policy evolution graphs are progressively shifted towards the "Always take" policy. Therefore, this appears the dominant strategy. A possible explanation could be linked to the Nash equilibrium for this game, which is taking on the first step: it results that the growth of instability favours the emergence of this equilibrium.

Furthermore, comparing the experiments that have the same number of learning agents but a different composition of deterministic entities, there is a difference in the average reward. More specifically, if half of the deterministic agents play "Always defect", the average reward is lower. Considering the graph that illustrates the policies evolution, it is clear that the presence of defecting agents entails a higher number of learning entities which play a non-cooperative strategy. Indeed, the Reinforcement Learning and the opponent modelling agents are encouraged to play "Always take" with defecting enemies, because in the first case this is the optimal policy against them while the second type of agents is aligning with the opponent's strategy.

4.4.2 Apple Picking

Following the approach chosen for the experiments in the Centipede game, the initial set of simulations involves a classic *DqnAgent* agent and aims to provide a baseline behaviour for all the other trainings. The game is iterated 8000 times in each simulation. The average reward and its standard deviation are illustrated in the following graph.

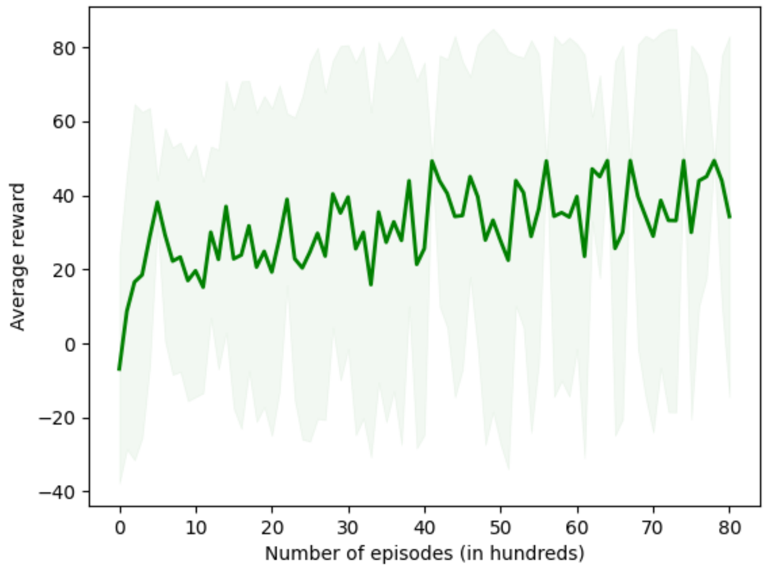


Figure 21 - Apple Picking environment: pure DqnAgent average reward.

As shown in the image, the average reward is nearly constant, but a significant standard deviation is observed over time. This behaviour might be due to the fact that there is not a single optimal strategy to collect all the apples using the fewest possible moves. Indeed, there are multiple policies that lead to the same goal; therefore, the possibility of converging to different paths within the grid world might produce this remarkable variance.

Considering the sequence of actions at the end of each simulation, the agent learns a finite strategy in 70% of the trainings. All these policies are very similar to each other and the most frequent one is described in the following image.

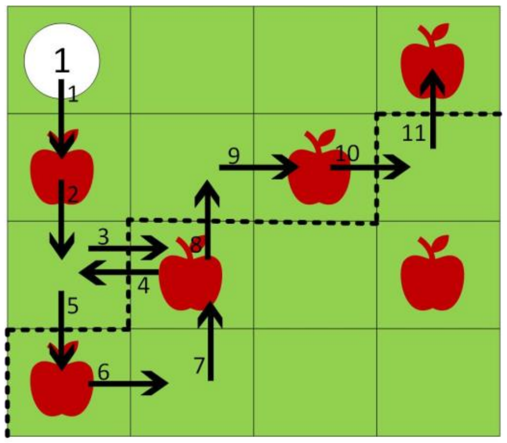


Figure 22 - Apple Picking environment: pure DqnAgent's most frequent strategy.

According to the image, the pure RL agent learns to pick 5 out of 6 apples available. This corresponds to the maximum number of apples that the agent can collect; in fact, the opponent moves in the field as well, starting from the bottom right corner, and its first action is gathering the apple that is closest to it.

In order to progressively study the agent's behaviour, two different types of simulations were executed. The first case takes advantage of the symmetry of the grid world to simplify the use of the opponent's reward. More specifically, in this setting the agent tries to adopt the opponent's strategy aligning with its values. Therefore, before getting the opponent's reward from the cost vector, the state is rotated by 180 degrees so that the agent can consider the state from the opponent's point of view. On the opposite, the other group of simulations eliminates this simplification and considers the original state; thus, this is the most general case.

However, training the developed agent in the Apple Picking environment results significantly more challenging, compared to the Centipede game. These difficulties are mainly linked to the Maximum Entropy IRL algorithm, which is an iterative procedure that considers all the possible states of the game. Therefore, the concrete implementation of the algorithm was slightly modified to increase the efficiency. In addition to it, the demonstrations collected for the IRL procedure contains simplified states. More specifically, the goal of the algorithm is inferring the opponent's reward which depends on the apples position only in this specific case. Thus, the state contains just the location of the remaining apples and the position of the opponent. Removing the opponent modelling agent's position does not affect the computation of the cost vector and reduces the number of states.

Given these changes, the goal of the first set of experiments which involves the opponent modelling agent is determining the best feature function. The simplified approach was adopted, and the agent considers the grid world state from the opponent’s point of view, when computing its reward. Since this is a preliminary study that aims to qualitatively investigate which feature function is most suitable, only one simulation was executed for each multiplying factor β. More specifically, the two feature functions evaluated are a custom feature function and a raw representation of the state. The first one extracts the vector $(num.\ apples,\ distance,\ up,\ down,\ right,\ left)$ from each state, where $num. apples$ is the number of remaining apples in the grid world, $distance$ is the distance from the closest apple and the last four elements represent the direction of the closest apple expressed considering the orientation components. On the other hand, the second approach consists in using directly the raw state; therefore, the feature function is an identity function.

The following graphs compare the opponent modelling agent's behaviour with different feature functions for $\beta = 1.0$, $\beta = 2.5$ and $\beta = 5.0$.

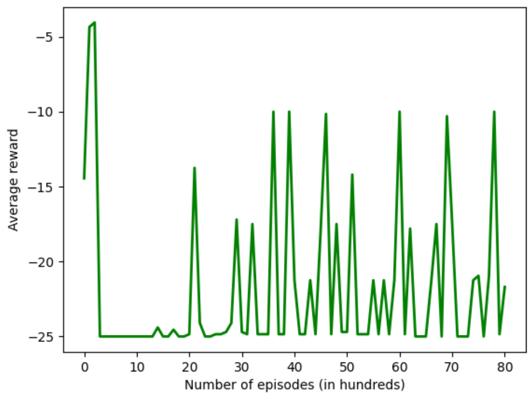
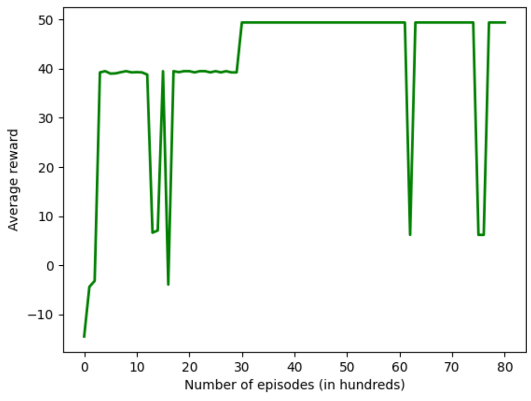
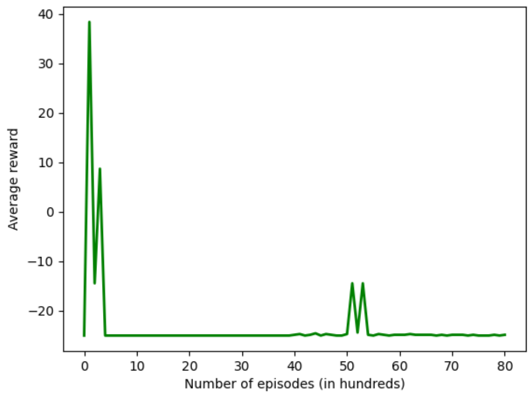
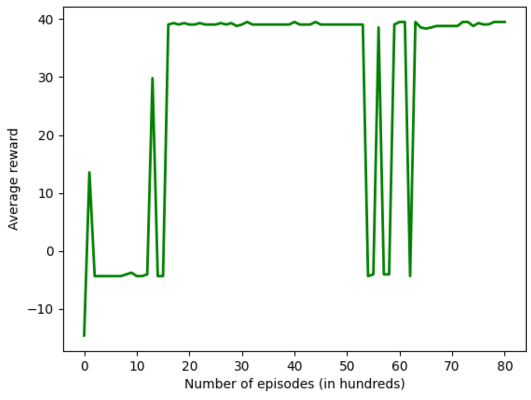
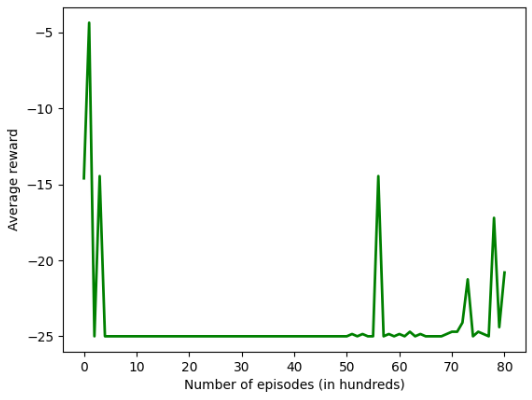
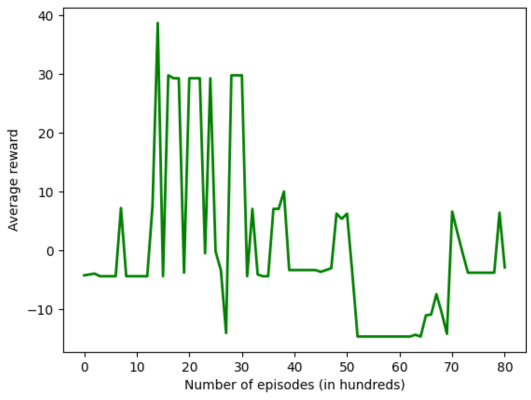
	Custom feature function	*Raw features*
$\beta = 1.0$		
$\beta = 2.5$		
$\beta = 5.0$		

Figure 23 - Apple Picking environment: opponent modelling agent with different feature functions.

According to the graphs, the average reward is characterized by peaks, either up or down, in all simulations. A possible explanation might be linked to the use of an epsilon-greedy policy for the learning agent; therefore, random actions could produce these sudden variations of behaviour.

Analysing the outcomes, raw features produce a significantly better behaviour which is comparable with the baseline average reward presented before. Moreover, it appears that a too high multiplying factor β negatively affects the average reward. All things considered, the raw features approach is the most suitable and the simulations will focus on β values around $\beta = 2.5$.

After defining the feature function for the Maximum Entropy IRL algorithm, the next group of simulations concerns the first type of trainings which considers the symmetry of the grid world to encourage the agent's learning. There are 8000 rounds of the Apple Picking game for each simulation and the learning agent always plays against the deterministic entity previously described. Since the goal is aligning with the opponent's values and the agent perceives the state from its adversary's point of view, only positive multiplying factors are studied. The following graph illustrates the average reward for different positive values of β.

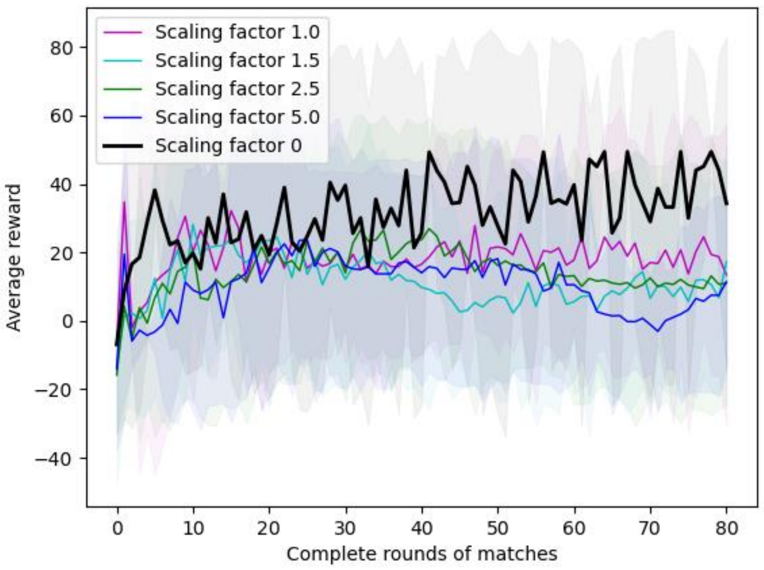


Figure 24 - Apple Picking environment with symmetry: opponent modelling agent's average reward.

According to the image, the average reward is lower compared to the outcome of the pure RL agent. Moreover, considering the series of actions at the end of each training, the agent converges to a finite policy in fewer simulations. This behaviour might be due to the instability introduced by the Maximum Entropy IRL algorithm which approximates the opponent's preferences. Furthermore, the agent is encouraged to collect apples as fast as possible associating a penalty to the empty cells: the opponent's reward could cancel this penalty promoting a wandering strategy which entails a lower extrinsic average reward and an infinite policy. However, the finite strategies obtained are different from the policies found by the *DqnAgent* since they result more cooperative. It is worth noting that the agent learns the exact strategy of its opponent in more than one simulation.

Considering the average reward and the final strategies learnt by the agent, the best outcome is obtained when β = 2.5; indeed, the agent converges to a finite policy in 30% of the simulations and 20% of the time the opponent's exact strategy is adopted. This policy is illustrated in the scheme below.

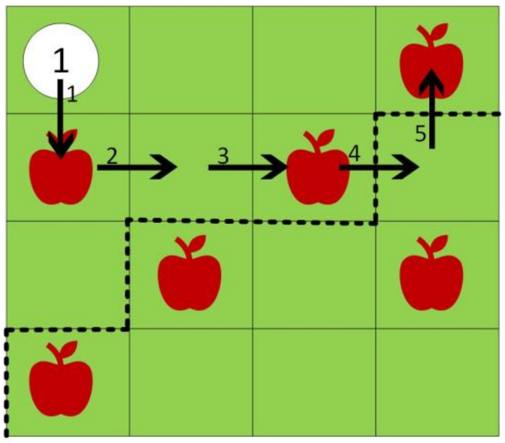


Figure 25 - Apple Picking environment with symmetry: learning agent's strategy when aligning with its opponent.

Overall, the outcomes of this set of experiments highlight that the introduction of the opponent's values produces more instability, and the agent converges to finite policies in fewer simulations. As a matter of fact, the agent should learn both the adversary's preferences and its optimal policy. Thus, learning a stable strategy results more challenging. However, the intrinsic reward positively influences the learnt strategies which appear more cooperative. Moreover, the emergence of a policy which is identical to the opponent's strategy, but which has never been experienced in pure RL trainings, is a clear evidence that the learning agent is properly inferring and using the opponent's values.

The last set of trainings studies the most general case where no additional information is used apart from the opponent's past actions. Therefore, the learning agent computes the opponent's cost vector using previous demonstrations to obtain its adversary's reward for each position that the agent visits. However, preliminary trainings do not appear particularly promising. One of the reasons of this behaviour might be linked to the states collected as opponent's demonstrations. Indeed, the opponent plays a deterministic strategy, and it experiences only few states which are far from the positions that the learning agent could visit at the beginning of the game. Therefore, the opponent's reward inferred with the Maximum Entropy IRL algorithm might not be accurate for states which never appear in demonstrations. Since these situations are crucially the initial states of the learning agent, considering the approximation of the opponent's values could be misleading. Consequently, the implementation of the agent was slightly modified for these experiments only, in order to remember which states the adversary visited. More specifically, the opponent's reward computed with the IRL procedure is considered in the overall reward exclusively if the specific state is present in the adversary's demonstrations; the environment feedback is applied alone otherwise. This updated agent is trained in the Apple Picking environment for different values of the multiplying factor β to study the emergence of cooperation or the ability to exploit its opponent. In particular, the opponent's reward associated to a specific position expresses how much that location is favourable or not from the adversary's point of view. Therefore, positive values of β favours the opponent's exploitation since the learning agent positively evaluates the same position of its opponent. On the other hand, negative parameters should discourage the learning agent to move towards the opponent's most convenient states.

The agent's behaviour for different positive values of β was firstly studied. The following graph compares the average reward of various simulations.

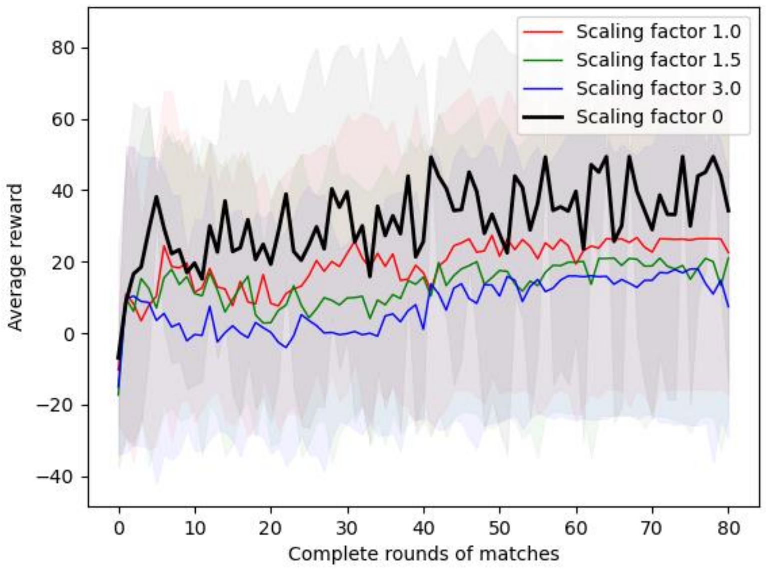


Figure 26 - Apple Picking environment: opponent modelling agent's average reward for positive values of β.

As shown in the image, the average reward is lower compared to the trainings of the pure RL agent. Moreover, the number of simulations in which the agent converges to a finite strategy decreases as the value of β increases. Considering the finite strategies obtained, the agent tries to behave non-cooperatively in most cases; however, the exact opponent's policy emerges in few simulations.

Comparing the outcomes for different values of the multiplying factor, $\beta = 1.0$ entails a slightly higher average reward; however, the agent learns a finite policy in fewer simulations, compared to the case of $\beta = 1.5$.

As an example, if β = 1.5 the agent learns a finite strategy in 60% of simulations; the most frequent policy is illustrated below.

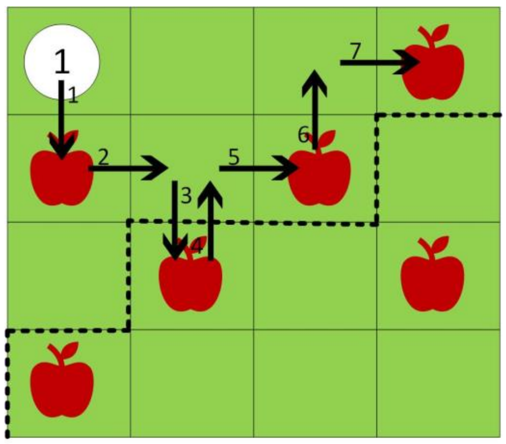


Figure 27 - Apple Picking environment: learning agent's strategy when exploiting its opponent, β =1.5.

According to the graph, the strategy obtained is very similar to the opponent's policy, illustrated in the previous image. However, this is modified to harvest an additional apple. This behaviour clarifies the influence of the opponent's rewards, but also the intention of exploit the other entity collecting one of its apples.

On the other hand, the opponent modelling agent's learning process was studied for negative values of β as well. Given the previous considerations, the agent should behave more cooperatively in this case. The image below compares the average reward for different values of β.

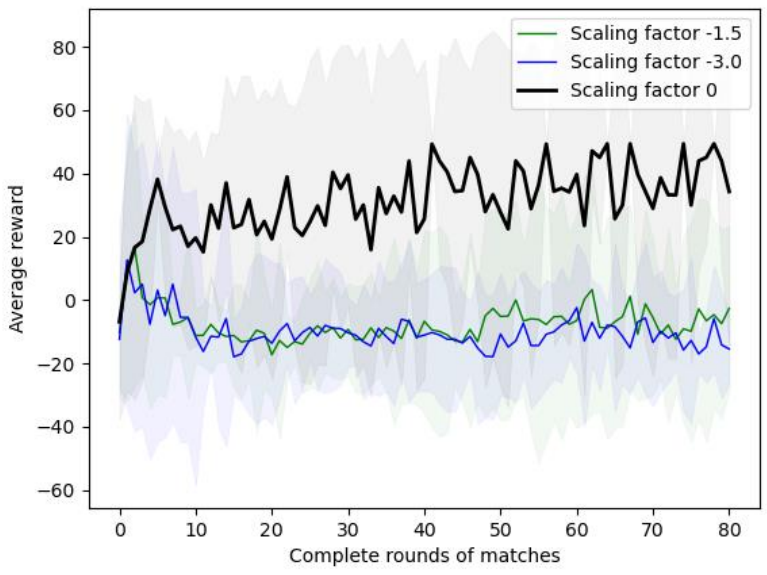


Figure 28 - Apple Picking environment: opponent modelling agent's average reward for negative values of β.

According to the graph, also these simulations highlight that the average reward is lower compared to the *DqnAgent*. Furthermore, learning a finite strategy results harder with negative values of β, compared to the previous trainings. However, considering the few finite policies obtained, they result more cooperative. For example, for $\beta = -1.5$, the agent develops a finite strategy only in 10% of simulations; this policy is illustrated in the following scheme.

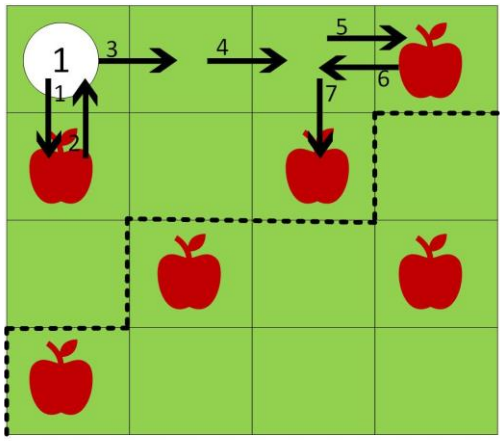


Figure 29 - Apple Picking environment: learning agent's strategy when cooperating with its opponent, β =-1.5.

All things considered, learning in this more general setting is harder and the final behaviour oscillates significantly. More specifically, the average reward is lower compared to the pure RL trainings and finite policies are obtained less frequently, especially for negative values of β. However, it is worth noting that all the final strategies are different from the policies developed by the *DqnAgent*; this proves that considering the opponent's rewards influences the agent's behaviour.

4.5 Discussion

The central idea of the implemented agent, which consists in modelling the opponent's values using Inverse Reinforcement Learning, is adopted by different research studies presented in the literature. However, the novelty concerns the way in which this information is integrated with the agent's learning algorithm. Indeed, the goal is to leave the learning algorithm unchanged and integrate the opponent's information, obtained with IRL, in the agent's reward. This favours the interpretation proposed where the opponent's reward is properly manipulated and considered as an intrinsic reward for the learning agent.

Considering the outcomes obtained during simulations in different environments, it results clear that the opponent's reward can be inferred from demonstrations and it can be used to influence the behaviour of the developed agent. Therefore, it can learn the proper strategy taking into account this information. However, the chosen approach to combine the intrinsic and extrinsic rewards does not always produce the expected behaviour. Therefore, other blending methods should be tested. Moreover, different environments entail different parameters and results; consequently, the possibility of identifying a universal technique to combine extrinsic and intrinsic rewards cannot be guaranteed.

In terms of simulations executed, some specific limitations should be mentioned. First of all, the case in which two opponent modelling agents play against each other was not thoroughly studied. Indeed, only the Centipede game population trainings consider this situation. Moreover, other strategies for the deterministic opponents could have been considered. For example, in the Centipede game case, studying the agent's behaviour against a deterministic entity which plays Tit-For-Tat might be interesting. On the other hand, the deterministic agent in the Apple Picking environment might choose other paths in the grid world. Finally, in the Maximum Entropy IRL algorithm, the opponent's cost vector can be accurately reconstructed only if the selected feature function is suitable for the specific environment. Hence, different functions could have been tested, possibly considering also automatic feature extraction techniques.

A frequent issue in Deep Reinforcement Learning trainings is learning instability; indeed, exploratory actions and function approximation are two main causes. Deep Q-learning is even more exposed to instability since it is characterized by all the three elements of "the deadly triad" described by (Sutton & Barto, 2018). In addition to this, there are two additional elements that produce instability in the opponent modelling agent. First of all, the presence of other agents in the same environment favours an unstable behaviour, especially if they are all learning entities. Moreover, the intrinsic reward representing the opponent's values is another source of instability. As a matter of fact, the Maximum Entropy IRL algorithm provides an approximation of the opponent's cost vector, which is improved with new demonstrations. Therefore, these figures are not exact, and they vary during training. As a consequence, this represents an important source of instability that might also explain the oscillating trend of the average reward for the opponent modelling agent.

Finally, another problem that clearly emerges from simulations is the efficiency issue linked to the Maximum Entropy IRL algorithm. As a matter of fact, this is an iterative procedure that evaluates all the possible states for a specific environment. Therefore, a small growth in the number of states produces a significant increase of the computation time. This limits the use of this algorithm to real problems because reconstructing the cost vector would be infeasible. A potential solution could be improving the algorithm implementation to enhance the efficiency or changing some parameters with the same objective. Another possibility consists in using state approximations or neural networks to represent the rewards.

5 Conclusions

The aim of this work is realizing an autonomous agent which properly interacts with other entities using the information deducted during execution. More specifically, the main goal is implementing a learning agent which is able to infer the opponent's values and use this evidence to find the policy that best satisfies its objective. Therefore, central themes are Reinforcement Learning, Multiagent environments and Value Alignment. The proposed agent adopts the Deep Q-Learning algorithm, and it receives a reward which is a combination of the environment feedbacks and the opponent's values. These latter preferences are obtained through the Maximum Entropy IRL algorithm as the opponent's cost vector; depending on how these rewards are used, a cooperative or exploiting agent may be obtained. The developed entity is tested in two very different environments: the Centipede game and the Apple Picking game.

The central problem in the development of the agent presented concerns the opponent modelling approach. The chosen solution does not alter the Deep Q-Learning algorithm and the information on the opponent's preferences is integrated in the overall reward that the agent receives. This method has some advantages and disadvantages, but it allows the interpretation of the opponent's cost vector as an intrinsic reward for the agent. This cost vector is computed applying an Inverse Reinforcement Learning algorithm, the Maximum Entropy IRL algorithm, on a set of demonstrations that describes the previous interactions between the agent and its adversary. The total reward received is a linear combination of environment feedbacks and opponent's values; changing a multiplying parameter, it is possible to behave cooperatively or not, as previously mentioned.

Comparing the outcomes obtained during various simulations, the opponent modelling agent mainly behaves as expected in the Centipede game, while the Apple Picking environment results significantly more challenging. As previously discussed, there are many possible explanations mostly depending on the IRL algorithm and the blending method of extrinsic and intrinsic rewards. More specifically, exploiting the opponent's strategy appears harder than learning for cooperating with it. This could be a clear indication that the relation between the environment feedbacks and opponent's rewards is more elaborate than a simple linear combination.

Overall, the proposed agent is able to infer the opponent's cost vector, which clarifies its values and preferences. These rewards are properly combined with the agent's own rewards to obtain the desired behaviour. Testing the agent on two different environments, the outcomes result promising. However, the final behaviour can be further improved under different points of view. Moreover, these broad trainings highlight some limitations that should be considered when adopting the agent presented.

5.1 Limitations

The first problem that clearly emerges from different simulations concerns the scalability of the agent. More specifically, the developed agent cannot by directly used in real-world environments because the Maximum Entropy IRL algorithm applies an iterative procedure which considers all the possible states. Therefore, computing the opponent's cost vector when interacting in complicated environments results infeasible. As a matter of fact, moving from the Centipede game to the Apple Picking environment produces a significant increase in the computation time. However, there are different possible solutions to this problem such as state approximations.

Moreover, there is another limitation linked to the IRL algorithm. Indeed, the Maximum Entropy IRL procedure requires known dynamics. These can be computed from demonstrations, but this is not feasible for non-linear dynamics. Therefore, even if the agent were scalable, it would still not be general enough to be applied to any environment.

In addition to this, the agent adopts individual learning in a multiagent environment. This is convenient considering the goal of studying the independent learning process in a shared environment; however, this might introduce learning instability. Indeed, the opponent modelling agent perceives the other entity as part of the environment which results itself unstable. This source of non-stationarity can be removed if the adversary's strategy is deterministic since the multiagent MDP is equivalent to a single-agent MDP (He et al., 2016).

Finally, directly integrating the opponent's reward with the environment feedbacks entails a lower interpretability compared to other solutions where separate policies are learnt. As previously mentioned, (R. Noothigattu et al., 2019) proposes an agent which learns two separate policies, one from the environment rewards and the other applying IRL to a set of demonstrations; the two policies are then properly alternated using an orchestrator. In this solution, it is very clear which policy is followed at a specific point of time; however, the architecture is more complicated and the need of keeping the learnt policies separate is not always required.

5.2 Future work

The opponent modelling agent presented achieves promising results during different simulations; however, the agent's behaviour is not perfect and better outcomes are obtained in the Centipede game environment, compared to the Apple Picking setting. Therefore, further improvements are possible in future studies.

First of all, the current agent assumes that the opponent's reward can be expressed as a linear function. However, this approximation might not correctly represent the true function that the opponent is optimizing, especially for complicated environments. Moreover, in real-world environments a scalability problem emerges as well. Consequently, a more general and scalable approach could involve a Neural Network. More specifically, the Maximum Entropy Deep IRL framework proposed by (Wulfmeier et al., 2016) applies Fully Convolutional Neural Networks to approximate the opponent's values solving the problem of modelling reward functions for large state spaces. This solution could be integrated in the opponent modelling agent to improve its behaviour.

In addition to this, the possibility of applying the implemented agent to a general environment is further limited by the condition of having known or easily computable transition dynamics. (Finn et al., 2016) proposes an Inverse Optimal Control method to infer the opponent's preferences with unknown dynamics. This approach could inspire additional improvements in the opponent modelling agent.

Furthermore, the agent's behaviour depends on the method chosen to blend the opponent's rewards with the environment feedbacks. Indeed, a simple linear combination might not be expressive enough; hence, a further development could be applying meta-learning to identify a more complicated relation between the two rewards. This interesting approach has been applied in the context of opponent modelling (Rabinowitz et al., 2018; Vilalta & Drissi, 2002); however, here the specific goal would be learning how to properly combine the opponent's information with the agent's own extrinsic reward.

Finally, apart from improving the proposed agent, there are many other possible expansions. Indeed, being able to understand the opponent's intentions and properly use this information allow several applications. For example, the agent could learn and save multiple policies, one for each type of entity with which it interacts; this would entail a different behaviour with every specific agent. Another possibility consists in developing an agent which is aware that its opponent understands its intentions and it thus adopts a behaviour which prevents from being exploited. In addition to it, the agent could be pretrained on standard interactions in order to obtain a default behaviour which is applied at the beginning of the trainings. Lastly, the agent could have a short-term and a long-term model; the first one is simpler and requires only few demonstrations to be learnt. On the contrary, the long-term model is more complicated and accurate, thus it can be obtained with longer interactions.

References

Abbeel, P., & Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. *Proceedings of the Twenty-First International Conference on Machine Learning*, 1. https://doi.org/10.1145/1015330.1015430

Adam, S., Busoniu, L., & Babuska, R. (2012). Experience Replay for Real-Time Reinforcement Learning Control. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *42*(2), 201–212. https://doi.org/10.1109/TSMCC.2011.2106494

Albrecht, S. V., & Stone, P. (2018). Autonomous Agents Modelling Other Agents: A Comprehensive Survey and Open Problems. *Artificial Intelligence*, *258*, 66–95. https://doi.org/10.1016/j.artint.2018.01.002

Allen, C., Wallach, W., & Smit, I. (2006). Why Machine Ethics? *IEEE Intelligent Systems*, *21*(4), 12–17. https://doi.org/10.1109/MIS.2006.83

Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., & Mané, D. (2016). Concrete Problems in AI Safety. *ArXiv:1606.06565 [Cs]*. http://arxiv.org/abs/1606.06565

Anastassacos, N., Hailes, S., & Musolesi, M. (2020). Partner Selection for the Emergence of Cooperation in Multi-Agent Systems Using Reinforcement Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, *34*(05), 7047–7054. https://doi.org/10.1609/aaai.v34i05.6190

Anastassacos, N., & Musolesi, M. (2018). Learning through Probing: A decentralized reinforcement learning architecture for social dilemmas. *ArXiv:1809.10007 [Cs]*. http://arxiv.org/abs/1809.10007

Arora, S., & Doshi, P. (2020). A Survey of Inverse Reinforcement Learning: Challenges, Methods and Progress. *ArXiv:1806.06877 [Cs, Stat]*. http://arxiv.org/abs/1806.06877

Barto, A., Singh, S., & Chentanez, N. (2004). Intrinsically motivated learning of hierarchical collections of skills. *Proceedings of the 3rd International Conference on Development and Learning*.

Bratko, I., & Šuc, D. (2002). Using Machine Learning to Understand Operator's Skill. In T. Hendtlass & M. Ali (Eds.), *Developments in Applied Artificial Intelligence* (pp. 812–823). Springer. https://doi.org/10.1007/3-540-48035-8_78

Busoniu, L., Babuska, R., & Schutter, B. D. (2008). A Comprehensive Survey of Multiagent Reinforcement Learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *38*(2), 156–172. https://doi.org/10.1109/TSMCC.2007.913919

Cressman, R., & Tao, Y. (2014). The replicator equation and other game dynamics. *Proceedings of the National Academy of Sciences, 111*(Supplement 3), 10810–10817. https://doi.org/10.1073/pnas.1400823111

Dafoe, A., Hughes, E., Bachrach, Y., Collins, T., McKee, K. R., Leibo, J. Z., Larson, K., & Graepel, T. (2020). Open Problems in Cooperative AI. *ArXiv:2012.08630 [Cs]*. http://arxiv.org/abs/2012.08630

Dawes, R. M. (1980). Social Dilemmas. *Annual Review of Psychology, 31*(1), 169–193. https://doi.org/10.1146/annurev.ps.31.020180.001125

Finn, C., Levine, S., & Abbeel, P. (2016). Guided Cost Learning: Deep Inverse Optimal Control via Policy Optimization. *ArXiv:1603.00448 [Cs]*. http://arxiv.org/abs/1603.00448

Foerster, J. N., Chen, R. Y., Al-Shedivat, M., Whiteson, S., Abbeel, P., & Mordatch, I. (2018). Learning with Opponent-Learning Awareness. *ArXiv:1709.04326 [Cs]*. http://arxiv.org/abs/1709.04326

Gabriel, I. (2020). Artificial Intelligence, Values and Alignment. *Minds and Machines, 30*(3), 411–437. https://doi.org/10.1007/s11023-020-09539-2

Gallego, V., Naveiro, R., Insua, D. R., & Oteiza, D. G.-U. (2019). Opponent Aware Reinforcement Learning. *ArXiv:1908.08773 [Cs, Stat]*. http://arxiv.org/abs/1908.08773

Hadfield-Menell, D., Dragan, A., Abbeel, P., & Russell, S. (2016). Cooperative Inverse Reinforcement Learning. *ArXiv:1606.03137 [Cs]*. http://arxiv.org/abs/1606.03137

Hardin, G. (1968). The Tragedy of the Commons. *Science, 162*(3859), 1243–1248. https://doi.org/10.1126/science.162.3859.1243

He, H., Boyd-Graber, J., Kwok, K., & Daumé III, H. (2016). Opponent Modeling in Deep Reinforcement Learning. *ArXiv:1609.05559 [Cs]*. http://arxiv.org/abs/1609.05559

Hussein, A., Gaber, M. M., Elyan, E., & Jayne, C. (2017). Imitation Learning: A Survey of Learning Methods. *ACM Computing Surveys, 50*(2), 21:1-21:35. https://doi.org/10.1145/3054912

Jaques, N., Lazaridou, A., Hughes, E., Gulcehre, C., Ortega, P. A., Strouse, D. J., Leibo, J. Z., & de Freitas, N. (2019). Social Influence as Intrinsic Motivation for Multi-Agent Deep Reinforcement Learning. *ArXiv:1810.08647 [Cs, Stat]*. http://arxiv.org/abs/1810.08647

Jaynes, E. T. (1957). Information Theory and Statistical Mechanics. *Physical Review, 106*(4), 620–630. https://doi.org/10.1103/PhysRev.106.620

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, *4*, 237–285. https://doi.org/10.1613/jair.301

Leibo, J. Z., Zambaldi, V., Lanctot, M., Marecki, J., & Graepel, T. (2017). Multi-agent Reinforcement Learning in Sequential Social Dilemmas. *ArXiv:1702.03037 [Cs]*. http://arxiv.org/abs/1702.03037

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, *8*(3–4), 293–321.

Lipton, Z. C., Azizzadenesheli, K., Kumar, A., Li, L., Gao, J., & Deng, L. (2018). Combating Reinforcement Learning's Sisyphean Curse with Intrinsic Fear. *ArXiv:1611.01211 [Cs, Stat]*. http://arxiv.org/abs/1611.01211

Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Machine Learning Proceedings 1994* (pp. 157–163). Elsevier. https://doi.org/10.1016/B978-1-55860-335-6.50027-1

Macy, M. W., & Flache, A. (2002). Learning dynamics in social dilemmas. *Proceedings of the National Academy of Sciences*, *99*(suppl 3), 7229–7236. https://doi.org/10.1073/pnas.092080099

Markovitch, S., & Reger, R. (2005). Learning and Exploiting Relative Weaknesses of Opponent Agents. *Autonomous Agents and Multi-Agent Systems*, *10*(2), 103–130. https://doi.org/10.1007/s10458-004-6977-7

McKelvey, R. D., & Palfrey, T. R. (1992). An Experimental Study of the Centipede Game. *Econometrica*, *60*(4), 803–836. https://doi.org/10.2307/2951567

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533. https://doi.org/10.1038/nature14236

Nagel, R., & Tang, F. F. (1998). Experimental Results on the Centipede Game in Normal Form: An Investigation on Learning. *Journal of Mathematical Psychology*, *42*(2), 356–384. https://doi.org/10.1006/jmps.1998.1225

Ng, A. Y., & Russell, S. J. (2000). Algorithms for Inverse Reinforcement Learning. *Proceedings of the Seventeenth International Conference on Machine Learning*, 663–670.

Noothigattu, R., Bouneffouf, D., Mattei, N., Chandra, R., Madan, P., Varshney, K. R., Campbell, M., Singh, M., & Rossi, F. (2019). Teaching AI agents ethical values using reinforcement learning and policy orchestration. *IBM Journal of Research and Development*, *63*(4/5), 2:1-2:9. https://doi.org/10.1147/JRD.2019.2940428

Noothigattu, Ritesh, Gaikwad, S. “Neil” S., Awad, E., Dsouza, S., Rahwan, I., Ravikumar, P., & Procaccia, A. D. (2018). A Voting-Based System for Ethical Decision Making. *ArXiv:1709.06692 [Cs]*. http://arxiv.org/abs/1709.06692

Osa, T., Pajarinen, J., Neumann, G., Bagnell, J. A., Abbeel, P., & Peters, J. (2018). An Algorithmic Perspective on Imitation Learning. *Foundations and Trends in Robotics*, *7*(1–2), 1–179. https://doi.org/10.1561/2300000053

Pathak, D., Agrawal, P., Efros, A. A., & Darrell, T. (2017). Curiosity-driven Exploration by Self-supervised Prediction. *ArXiv:1705.05363 [Cs, Stat]*. http://arxiv.org/abs/1705.05363

Peterson, M. (2019). The value alignment problem: A geometric approach. *Ethics and Information Technology*, *21*(1), 19–28. https://doi.org/10.1007/s10676-018-9486-0

Rabinowitz, N. C., Perbet, F., Song, H. F., Zhang, C., Eslami, S. M. A., & Botvinick, M. (2018). Machine Theory of Mind. *ArXiv:1802.07740 [Cs]*. http://arxiv.org/abs/1802.07740

Rand, D. G., & Nowak, M. A. (2012). Evolutionary dynamics in finite populations can explain the full range of cooperative behaviors observed in the centipede game. *Journal of Theoretical Biology*, *300*, 212–221. https://doi.org/10.1016/j.jtbi.2012.01.011

Rosenthal, R. W. (1981). Games of perfect information, predatory pricing and the chain-store paradox. *Journal of Economic Theory*, 92–100.

Russell, S. (2020). *Human Compatible: AI and the Problem of Control*. Penguin.

Russell, S. (1998). Learning agents for uncertain environments (extended abstract). *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, 101–103. https://doi.org/10.1145/279943.279964

Sandholm, T. W., & Crites, R. H. (1996). Multiagent reinforcement learning in the Iterated Prisoner’s Dilemma. *Biosystems*, *37*(1), 147–166. https://doi.org/10.1016/0303-2647(95)01551-5

Savinov, N., Raichuk, A., Marinier, R., Vincent, D., Pollefeys, M., Lillicrap, T., & Gelly, S. (2019). Episodic Curiosity through Reachability. *ArXiv:1810.02274 [Cs, Stat]*. http://arxiv.org/abs/1810.02274

Schuster, P., & Sigmund, K. (1983). Replicator dynamics. *Journal of Theoretical Biology*, *100*(3), 533–538. https://doi.org/10.1016/0022-5193(83)90445-9

Segismundo S. Izquierdo, L. R. I. and N. M. G. (2008, March 31). *Reinforcement Learning Dynamics in Social Dilemmas* [Text.Article]. JASSS. http://jasss.soc.surrey.ac.uk/11/2/1.html

Shannon, C. E. (1948). A Mathematical Theory of Communication. *Bell System Technical Journal*, *27*(3), 379–423. https://doi.org/10.1002/j.1538-7305.1948.tb01338.x

Shoham, Y., & Leyton-Brown, K. (2008). *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press.

Şimşek, Ö., & Barto, A. G. (2006). An intrinsic reward mechanism for efficient exploration. *Proceedings of the 23rd International Conference on Machine Learning*, 833–840. https://doi.org/10.1145/1143844.1143949

Singh, S., Barto, A. G., & Chentanez, N. (2004). Intrinsically motivated reinforcement learning. *Proceedings of the 17th International Conference on Neural Information Processing Systems*, 1281–1288.

Smead, R. (2008). The Evolution of Cooperation in the Centipede Game with Finite Populations. *Philosophy of Science*, *75*(2), 157–177. https://doi.org/10.1086/590197

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction, second edition*. MIT Press.

Taylor, P. D., & Jonker, L. B. (1978). Evolutionary stable strategies and game dynamics. *Mathematical Biosciences*, *40*(1), 145–156. https://doi.org/10.1016/0025-5564(78)90077-9

TensorFlow Agents. (n.d.). Retrieved January 3, 2021, from https://www.tensorflow.org/agents?hl=it

Tuyls, K., & Stone, P. (2018). Multiagent Learning Paradigms. In F. Belardinelli & E. Argente (Eds.), *Multi-Agent Systems and Agreement Technologies* (pp. 3–21). Springer International Publishing. https://doi.org/10.1007/978-3-030-01713-2_1

Tuyls, Karl, & Weiss, G. (2012). Multiagent Learning: Basics, Challenges, and Prospects. *AI Magazine*, *33*(3), 41–41. https://doi.org/10.1609/aimag.v33i3.2426

Van Lange, P. A. M., Joireman, J., Parks, C. D., & Van Dijk, E. (2013). The psychology of social dilemmas: A review. *Organizational Behavior and Human Decision Processes*, *120*(2), 125–141. https://doi.org/10.1016/j.obhdp.2012.11.003

Vilalta, R., & Drissi, Y. (2002). A Perspective View and Survey of Meta-Learning. *Artificial Intelligence Review*, *18*(2), 77–95. https://doi.org/10.1023/A:1019956318069

Wallach, W. (2010). Robot minds and human ethics: The need for a comprehensive model of moral decision making. *Ethics and Information Technology*, *12*(3), 243–250. https://doi.org/10.1007/s10676-010-9232-8

Wang, J. X., Hughes, E., Fernando, C., Czarnecki, W. M., Duenez-Guzman, E. A., & Leibo, J. Z. (2019). Evolving intrinsic motivations for altruistic behavior. *ArXiv:1811.05931 [Cs]*. http://arxiv.org/abs/1811.05931

Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. Ph.D thesis, King's College, Cambridge, UK.

Wu, Y.-H., & Lin, S.-D. (2018, April 25). A Low-Cost Ethics Shaping Approach for Designing Reinforcement Learning Agents. *Thirty-Second AAAI Conference on Artificial Intelligence*. Thirty-Second AAAI Conference on Artificial Intelligence. https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16195

Wulfmeier, M., Ondruska, P., & Posner, I. (2016). Maximum Entropy Deep Inverse Reinforcement Learning. *ArXiv:1507.04888 [Cs]*. http://arxiv.org/abs/1507.04888

Ziebart, B. D., Maas, A. L., Bagnell, J. A., & Dey, A. K. (2008). Maximum entropy inverse reinforcement learning. *AAAI*, *8*, 1433–1438.