Utrecht University

Master Thesis

Identifying the Opponent's Strategy by Behavioural Analysis in Repeated 2x2 Games

Author:
Joris Teunisse

Primary supervisor:
Mehdi Dastani

Daily supervisor:
Bas Testerink

August 14, 2020

Abstract

In the context of repeated games, the main goal of an agent is generally to achieve the highest reward possible. To this end, many types of agents estimate the future behaviour of their opponents in order to exploit their actions. We hypothesize that replacing this prediction with the opponents' exact action probabilities will lead to more efficient exploitation. With this in mind, we developed an agent that aims to efficiently identify a single opponent's strategy from a set of candidate strategies. We found that the efficiency of this process depends on the specific candidate set, the rewards that underlie the repeated game and the strategy of the agent itself. In future research, these findings can be used as the baseline for an agent that subsequently exploits this knowledge by applying a counter-strategy.

Acknowledgements

First, I would like to greatly thank Bas for our numerous discussions: they have helped me to grow both as a researcher and as a person. The informal but productive nature of these discussions made them very enjoyable, and they contributed significantly to the quality of this project. More generally, I am glad that we got along as well as we did: this social connection was something I had hoped for at the start of the project, and proved to be extra important during the sudden months of isolation.

Furthermore, my thanks go out to Mehdi for his constructive advice at every stage of the project. As a result, I was confronted with points of improvement that I would not have thought of myself: this helped to elevate the thesis to a higher level.

Finally, I want to thank my family and friends for their mental support: both during my search for a thesis position and during the project itself. I could not have done this without you.

Contents

1 Introduction

Within the world, countless interactions take place every day. For many of these interactions, the parties involved have a specific goal in mind: this can be as concrete as winning a game of chess, or as abstract as preserving peace.

On a fundamental level, this type of interaction consists of a few key elements: the parties (or *agents*) that interact with each other, the goal they have in mind and the actions available to them. In order to choose which actions should be taken, each agent applies a certain strategy: an effective strategy selects actions that lead to the desired goal.

An important factor for establishing an effective strategy is information about the other agents' strategies. By predicting their future behaviour based upon this information, an agent can subsequently take actions that will be beneficial in that context.

This thesis specifically focuses on software agents: studying interactions between multiple such agents is part of the Multi-Agent Systems (MASs) research area. Software agents that take others' behaviour into account have been researched extensively: however, determining the exact strategy used by other agents has received relatively little attention.

Obtaining this knowledge means that the agent does not have to predict future behaviour anymore: it will know exactly what the possible outcomes of its actions will be, which enables it to pursue its goal with maximum efficiency.

In this thesis, we study how an agent's strategy can be identified efficiently by analysing their behaviour. This will be carried out in the context of repeated 2x2 games, which provide a framework that can be used to model interactions. As this framework only considers two agents, we refer to the agent whose strategy has to be identified as the 'opponent'.

For this context, we developed an agent which solely aims to efficiently identify the strategy of their opponent. As there are an infinite number of possible strategies the opponent could be applying, the agent is supplied with a set of candidate strategies: one of which is the opponent's true strategy.

By testing the performance of this agent in a variety of situations, we found that the specific set of candidate strategies and the properties of the repeated 2x2 game greatly influence the efficiency of the identification process. Furthermore, we observed that this efficiency could frequently be improved by applying a strategy designed with this goal in mind.

The structure of this thesis is as follows. Section 2 discusses the prerequisite background for this project. Based upon this knowledge, Section 3 reviews relevant literature with regard to the current scope. This results in the research questions, which will be discussed in Section 4. Experiments are then conducted in Section 5, which provide observations relevant to the research questions. Finally, Section 6 discusses these observations and possible avenues for future work.

2 Background

Within the MASs research area, one of the main topics related to this project is Multi-Agent Reinforcement Learning (MARL). Developments which lead to the establishment of this field will be the primary focus of this section: MARL itself directly underlies multiple agents that will be discussed in Section 3, while the discussion on supporting developments provides the prerequisite background knowledge for this project.

We commence by discussing one of the most relevant scientific breakthroughs for the current scope: the development of the Q-learning algorithm (Watkins, 1989) for the single-agent setting. To understand the value of this algorithm and the concepts supporting it, we first discuss the formal model underlying its implementation: the Markov Decision Process (MDP).

2.1 Markov Decision Processes

An MDP (Bellman, 1957) is a mathematical model which specifies a certain system. The system is described in terms of four fundamental concepts: states, actions, rewards and a transition function.

To model a system as an MDP, all possible states the system could be in are specified. Each state is assigned a set of actions which are available in that state. Every state-action pair is associated with a reward, which is typically a scalar value. Finally, the transition function specifies a probability distribution over all states for each state-action pair. After an action has been taken in the current state, the system transitions to the next state according to the probabilities specified by this function for the current state-action pair.

A question of practical value is how to traverse an MDP while optimising the acquired rewards. Optimisation is conventionally discussed in the context of scalar rewards, which is also the scope of Q-learning. In this context, the question becomes how to maximise the rewards acquired over time. To this end, Bellman (1957) proposed the Bellman equation shown in Equation 1.

$$V(s) = \max_a \left(\sum_{s'} P(s,a,s') \left(R(s,a,s') + \gamma V(s') \right) \right) \tag{1}$$

The Bellman equation calculates the value $V(s)$ of a state s by determining which action a leads to the maximum quality. The quality of a state-action pair (s,a) is the sum of the reward $R(s,a,s')$ for reaching the next state s' and its current value $V(s')$ for every possible s': these are weighted by the probability $P(s,a,s')$ that the system will indeed transition to s'. Note that $V(s')$ is multiplied by a discount factor $\gamma \in (0,1)$: this controls the relative importance of the next state's value.

To use this equation to find the optimal way to traverse an MDP, Bellman (1957) proposed the 'value iteration' algorithm. This algorithm iteratively applies the Bellman equation to all states, while keeping track of their values.

Given an adequate number of iterations, these values converge: they can then be used to determine the optimal policy, which denotes the optimal action to take in each state. This policy is extracted by applying a variant of the Bellman equation to every state, which targets the action that produces the maximum quality rather than the quality itself.

To illustrate the value iteration algorithm, consider the MDP shown in Figure 1. In this figure, arrows from a state s to an action a specify that the action can be taken in that state. The arrows in the reverse direction specify the resulting transitions: the associated numbers indicate the probability $P(s, a, s')$ of the transition, followed by the reward $R(s, a, s')$ provided for it.

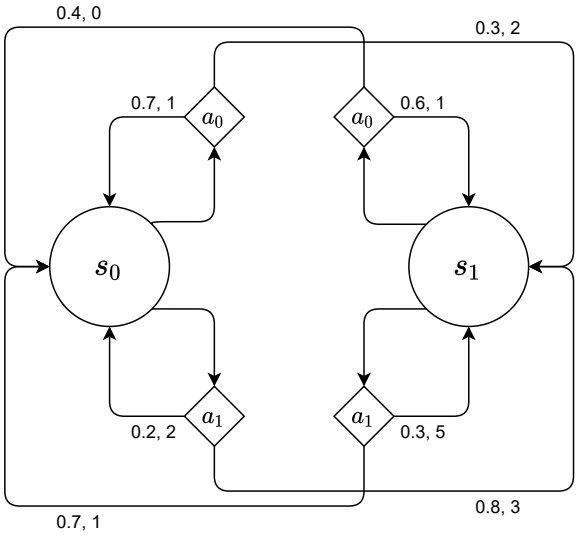

Figure 1: An MDP with two states.

At the start of the algorithm, $V(s)$ is initialised to 0 for both s_0 and s_1. The discount factor γ and the start state are established as well: in this example, we will set them to 0.9 and s_0 respectively.

As s_0 is the start state, the algorithm first calculates $V(s_0)$: this requires the action a which leads to the maximum quality. In s_0, there are two available actions: a_0 and a_1. To calculate the quality of (s_0, a_0), the algorithm considers all possible next states s': in this case, s_0 and s_1.

Under the assumption that $s' = s_0$, the MDP defines $P(s_0, a_0, s_0) = 0.7$ and $R(s_0, a_0, s_0) = 1$: furthermore, recall that $V(s_0) = 0$. This results in the following partial result: $0.7 * (1 + 0.9 * 0) = 0.7$. Similarly, the partial result when assuming $s' = s_1$ is $0.3 * (2 + 0.9 * 0) = 0.6$. Therefore, the quality of (s_0, a_0) is $0.7 + 0.6 = 1.3$.

Subsequently, the algorithm calculates the quality of (s_0, a_1). In the same manner as before, this leads to $0.2 * (2 + 0.9 * 0) = 0.4$ when assuming $s' = s_0$ and $0.8 * (3 + 0.9 * 0) = 2.4$ under the assumption that $s' = s_1$. This amounts

to a quality of $0.4 + 2.4 = 2.8$ for (s_0, a_1). As $2.8 > 1.3$, a_1 currently leads to the maximum quality in s_0. Therefore, $V(s_0)$ is updated to 2.8.

To illustrate the importance of tracking $V(s)$, consider the next iteration in which the algorithm calculates $V(s_1)$. Going through the same process as above, it obtains $P(s_1, a, s')$ and $R(s_1, a, s')$ from the MDP for all a and s'. Keep in mind that $V(s_0) = 2.8$: thus, (s_1, a_0) has a quality of $0.4 * (0 + 0.9 * 2.8) + 0.6 * (1 + 0.9 * 0) = 1.608$. Similarly, the quality of (s_1, a_1) is $0.7 * (1 + 0.9 * 2.8) + 0.3 * (5 + 0.9 * 0) = 3.964$. This results in $V(s_1)$ being updated to 3.964, after which the algorithm advances to the next iteration.

Bellman (1957) proves that $V(s)$ converges to the quality of the optimal state-action pair for every s, when allowed an appropriate number of iterations. This happens in the case of loops as well, as the discount factor limits the quality to a finite number. By consistently taking the action which leads to the highest quality in every state, a policy which optimises the acquired rewards when traversing an MDP is obtained: therefore, our original question has been answered.

2.2 Q-learning

The insights in Section 2.1 can be applied to understand the Q-learning algorithm mentioned at the beginning of Section 2. To understand its benefits, consider a policy derived by value iteration. While it does lead to the maximum reward over time, the underlying structure of the system has to be known in advance: specifically, the rewards for every state-action pair and the transition function.

Q-learning removes this requirement while still guaranteeing convergence to the correct values, which makes it more generally applicable. To find the optimal policy despite the reduced information, the algorithm stores data in a Q-table. Specifically, this table keeps quality approximations for each state-action pair: these approximations are also called Q-values.

The goal of Q-learning is to let each Q-value converge to the quality of its associated state-action pair: when this has been achieved, the optimal reward can be acquired by taking the action with the highest Q-value in each state. There are multiple requirements for achieving this goal, which will now be discussed.

One of the requirements is the application of an appropriate update rule to the Q-values: the formula which accomplishes this is shown in Equation 2.

$$Q(s,a) = (1-\alpha)\, Q(s,a) + \alpha \left(R(s,a) + \gamma \max_{a'} Q(s',a') \right) \tag{2}$$

When updating a Q-value $Q(s, a)$, the learning rate α decides the step size of the update. The update itself consists of a combination of the perceived reward $R(s, a)$ and the expected future reward. To calculate the latter, the discount factor γ is multiplied by the maximum Q-value $Q(s', a')$ of any action a' available in the perceived next state s'.

The Q-learning algorithm applies this update rule in the following manner. Starting from some initial state in an MDP, the algorithm takes the action with the highest Q-value: ties are broken by random selection. This results in the MDP granting some reward and moving to the next state according to the transition function.

However, the dynamics which underlie this process are unknown to the algorithm. Instead, it uses the perceived reward and next state to update the Q-value according to Equation 2. This cycle repeats until some termination requirement is met: afterwards, the process starts over from the initial state while retaining the updated Q-values.

By repeatedly undergoing this procedure, the Q-values are adjusted stepwise in the direction of the state-action pair qualities. Recall that this quality consists of several components: the reward of reaching possible next states combined with their discounted values, which are weighted by the transitional probabilities and summed together.

These components allow to draw a parallel between Equations 1 and 2. Both combine the reward of the next state with the maximum quality of that state, discounting the latter. The main difference is the transition function, which is explicitly used by the value iteration algorithm. However, Q-learning implicitly uses this function as well: as the probability of ending up in a certain next state is guided by the transitional probabilities of the system, the Q-values will average out to the qualities used in Equation 1 over time.

However, this parallel alone does not guarantee convergence to the correct qualities for each state-action pair. When the actions with the highest Q-values are greedily exploited in this manner, actions which initially seem suboptimal might not be considered at all. Therefore, the Q-learning algorithm employs an exploration method: a common approach is ϵ-greedy, which takes a random action with probability $\epsilon \in (0, 1)$ or exploits the action with the highest Q-value otherwise. Given enough iterations, this addition to the algorithm ensures that the Q-values for all state-action pairs converge to the correct qualities.

To clarify the Q-learning process, we will apply it to the MDP in Figure 1 as an example. As with value iteration, we set the initial state to s_0 and the discount factor γ to 0.9. Additionally, we set both the learning rate α and the exploration parameter ϵ to 0.1. The Q-values $Q(s, a)$ are initialised to 0: the evolution of these values per iteration is illustrated in Table 1.

0	a_0	a_1
s_0	0.00	0.00
s_1	0.00	0.00

1	a_0	a_1
s_0	0.20	0.00
s_1	0.00	0.00

2	a_0	a_1
s_0	0.20	0.00
s_1	0.00	0.28

3	a_0	a_1
s_0	0.31	0.00
s_1	0.00	0.28

4	a_0	a_1
s_0	0.31	0.00
s_1	0.33	0.28

Table 1: The evolution of the Q-values in the example below.

At the start of the algorithm, it will try to take the action with the highest Q-value. However, as all Q-values are initialised to 0, both a_0 and a_1 are valid options: this means that the action will be selected randomly. Assuming action a_0 is taken, the system transitions to either s_0 or s_1.

Assuming that the system transitioned to s_1, the perceived reward $R(s_0, a_0)$ is 2. Furthermore, recall that the corresponding Q-value $Q(s_0, a_0)$ is 0. With this in mind, the Q-value of (s_0, a_0) can be updated according to Equation 2: $Q(s_0, a_0) = (1 - 0.1) * 0 + 0.1 * (2 + 0.9 * 0) = 0.2$.

Afterwards, the algorithm advances to the next iteration. As the Q-values for both actions in s_1 are 0, a random action is selected once more: assume this is action a_1, after which the system transitions to s_0. As $Q(s_0, a_0) = 0.2$, the expected future reward now plays a role in this equation: $Q(s_1, a_1) = (1-0.1) * 0 + 0.1 * (1 + 0.9 * 0.2) = 0.28$.

The next iteration starts in s_0, in which the Q-learning algorithm decides to take a_0 since $Q(a_0) > Q(a_1)$: assume the system responds by transitioning to s_0. This results in the following update: $Q(s_0, a_0) = (1 - 0.1) * 0.2 + 0.1 * (1 + 0.9 * 0.28) = 0.3052 \approx 0.31$.

We will discuss one more iteration. Thus far, the exploration parameter ϵ has not interfered in the process. Without this parameter, the algorithm would continue to take a_0 in s_0 and a_1 in s_1: the other actions would be ignored, as their Q-values cannot surpass those of their counterparts.

However, ϵ presents an opportunity to explore the other options: for example, a_1 could be taken in s_0 with probability $0.1 * 0.5 = 0.05$. Assuming that this is presently the case and that the system transitions to s_1, the Q-value of (s_0, a_1) is updated: $Q(s_0, a_1) = (1-0.1)*0+0.1*(3+0.9*0.28) = 0.3252 \approx 0.33$. Note that this value surpasses $Q(s_0, a_0)$: therefore, this action will be preferred by the algorithm the next time s_0 is visited.

As the exploration parameter ensures that all state-action pairs will eventually be considered with nonzero probability, their Q-values are able to converge when given an appropriate learning rate and number of iterations: this is for instance proven in Tsitsiklis (1994). This means that the optimal reward for a given MDP can be acquired by systematically taking the action with the highest Q-value after convergence: therefore, Q-learning fulfills the goal mentioned at the beginning of this section.

2.3 Stochastic Games

While single-agent RL techniques like Q-learning are a part of the MARL research area, the focus on multi-agent rather than single-agent systems has several implications.

In an MDP model of a single-agent system, the rewards and transition function only depend on the action taken in a certain state. This static clause enables the agent to find the optimal policy: a state-action pair always leads to the same reward and distribution over possible next states, both of which can be learned.

When considering a MAS, convergence to the optimal policy is not guaranteed: the same state-action pair might lead to a different result due to the potential presence of other adaptive agents. Furthermore, the MDP model is incapable of separating other agents' behaviour from the dynamics of the environment. While the former issue is still a focus of ongoing debate (Zhang, Yang, & Başar, 2019), the latter can be remedied by using a different type of model.

An example of a model that considers multiple agents is the Stochastic Game (SG), which is a generalisation of the MDP. While the transition function of an MDP conditions on a state-action pair, in SGs (Shapley, 1953) it conditions on the joint action taken in a certain state. Afterwards, each agent is granted an individual reward: the concepts of states and individual actions remain the same.

By using the SG model, an agent can take other agents' behaviour into account. We will analyse this behaviour in a special case of SGs, in which there is only one state: to understand which dynamics might arise in this situation, we provide some relevant background on the Game Theory (GT) research area.

2.4 Game Theory

Laying the groundwork over a decade in advance (von Neumann, 1928), GT is often cited as being pioneered by von Neumann and Morgenstern (1944). By modeling interactions between multiple parties as a matrix game, it enables the analysis of these interactions using a common framework.

Unless otherwise specified, the remainder of this proposal will discuss matrix games with the following properties. A game is played by two parties, commonly called the row player R and column player C. Each player can take one of two actions, which are executed simultaneously: this makes it a normal-form game, in contrast to extensive-form games where actions are executed sequentially. Finally, the game is fully observable: the joint actions and rewards are both visible to the players.

We will refer to matrix games with these properties as '2x2' matrix games, in line with e.g. Robinson and Goforth (2005): an example of such a game is shown in Table 2.

	α	β
a	$3, 3$	$0, 5$
b	$5, 0$	$1, 1$

Table 2: The Prisoner's Dilemma, a 2x2 matrix game.

In the Prisoner's Dilemma, R can take either action a or b whereas C can opt to take α or β. The pair of integer values associated with the resulting joint action represents the rewards for R and C respectively. The interactions between the players can be analysed in multiple ways: one of the most fundamental concepts for this purpose is the Nash Equilibrium (NE).

A NE (Nash, 1951) indicates a joint action from which no player can deviate independently to achieve a better reward. Taking Table 2 as an example, the joint action (b, β) is a pure NE: playing either a or α would lead to the deviating player being worse off. Note that a NE does not necessarily have to grant the highest rewards: (a, α) is more beneficial for both players.

Though it does not always lead to the most optimal solution, playing joint actions which constitute a NE can serve as the goal of players participating in a matrix game. As the other player has no incentive to deviate, it guarantees a certain level of stability regarding the obtained reward. The importance of this stability is highlighted when considering repeated games: matrix games which are played multiple times in succession with the same players.

This project will focus on repeated games which are played by agents. An agent takes actions according to a certain strategy: an algorithm which denotes the probability distribution over actions for every possible situation.

Repeated games are similar to SGs: they consider the execution of successive joint actions, granting a reward after each iteration. However, they have no concept of a state: furthermore, the rewards condition only on the joint action taken. By constructing a SG which takes these properties into account, they can be used to model the dynamics of a repeated game.

Figure 2 illustrates an example of a repeated game modeled as a SG: in this game, agents repeatedly play the Prisoner's Dilemma. This SG takes the properties of the repeated game into account in the following manner. All joint actions only have a single outcome, which leads back to the original state: therefore, the model is essentially stateless. Furthermore, the rewards only condition on the joint action taken: they are represented by the integer pairs on the arrows.

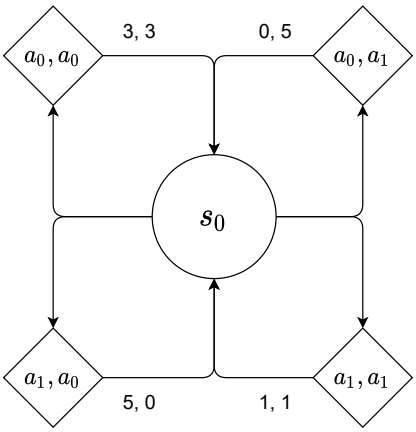

Figure 2: The repeated Prisoners' Dilemma modeled as a SG.

As mentioned in Section 2.3, considering joint actions allows an agent to take other agents' behaviour into account. For the sake of clarity, we will henceforth

describe all other agents as 'opponents': regardless of whether their strategy actually opposes the agent in question.

To illustrate how an an agent could analyse its opponent's behaviour over the course of a repeated game, consider the fictitious play algorithm (Brown, 1951). This algorithm keeps track of the actions its opponent has taken thus far: by normalizing the number of times the opponent has taken each action, a probability distribution is obtained. This probability distribution is subsequently used as an approximation for the opponent's policy: by multiplying these probabilities with the agent's individual rewards and summing them together per action, an agent can determine the expected reward of a certain action.

An important factor for this behavioural analysis are the properties of the repeated game's underlying matrix game: we will refer to this as the stage game, of which several examples are shown in Table 3.

	α	β
a	$4, 4$	$1, 3$
b	$3, 1$	$2, 2$

	α	β
a	$1, -1$	$-1, 1$
b	$-1, 1$	$1, -1$

	α	β
a	$0, 0$	$-1, 1$
b	$1, -1$	$-9, -9$

Table 3: The Stag Hunt, Matching Pennies and Chicken stage games.

Analysing the behaviour exhibited by agents in a variety of stage games will form the main focus of this project: this will be elaborated upon in Section 4.

Concluding the background section, we have shown multiple different ways in which MASs can be modeled. An MDP can represent multiple states but only one agent, while repeated games can model multiple agents in a stateless environment. These models are generalised by SGs, which can express a combination of multiple states and multiple agents: by respectively considering only one agent or state, SGs can also represent an MDP or a repeated game as a special case.

We have also discussed the basic Q-learning algorithm, which uses these models to learn an appropriate policy for the system at hand. Finally, we mentioned related concepts integral to the application of these methods. In the next section, we will use this knowledge to discuss advances in the MARL research area.

3 Literature review

In this section, we use the topics covered in Section 2 to support a critical review of literature relevant to this project. While the previous section has covered concepts derived from seminal work in related fields, the main goal of this section is to discuss articles from the MARL research area specifically. These will be presented in a way that highlights the gap in the literature which our research questions will address.

3.1 Fundamental MARL research

Working towards this gap chronologically, we start by discussing fundamental MARL research. One of the first papers to specifically research RL in the context of a MAS was Tan (1993). While arguably preceded by Whitehead (1991), Tan (1993) was the first to research the trade-offs between an equal number of independent and cooperating agents: it also focused on the effects of specific learning methods rather than a complexity analysis of finding the optimal policy, which is more in line with the approaches discussed hereafter.

In this seminal work, Tan (1993) presents an empirical study on agents that use Q-learning in a grid world: a MAS with a discrete two-dimensional environment, in which the states can be represented by the positions of the agents. Agents are classified as either 'hunter' or 'prey': the goal of the hunters is move to the prey's location in order to capture it, for which they receive a common reward. Both agent types used an MDP model of the system and independent Q-learning: therefore, they considered their opponents as part of the environment.

The variant of Q-learning used in Tan (1993) contains several notable adjustments in comparison to the description in Section 2.2. The hunters have access to a state which describes a limited area around them, rather than the entire system: this significantly reduces the state space.

Furthermore, the algorithm uses the Boltzmann distribution for exploration purposes rather than ϵ-greedy: this results in exploration relative to the current Q-values, which does not change the convergence properties of Q-learning when its parameters are set correctly (Singh, Jaakkola, Littman, & Szepesvári, 2000). However, note that convergence to the correct Q-values was already not guaranteed: as mentioned in Section 2.3, the presence of learning opponents renders this property of Q-learning invalid.

The variant of Q-learning described above is applied to several situations: in each situation, the hunters use this algorithm in conjunction with a specific information sharing method in order to capture the prey more effectively. This cooperative behaviour is shown to improve capture rates in experiments, the efficiency of which varied per method.

Tan (1993) highlights several concerns for MARL research in an early phase of the field's development. While this paper considers explicit communication methods, the author contemplates whether similar behaviour can be learned by the agents themselves. The large computational costs associated with deploying RL agents in a MAS are mentioned as a possible obstacle for future research as well. Lastly, it mentions that additional information can sometimes interfere with the learning process rather than augment it: this would turn out to be one of the critical factors for the perseverance of using independent Q-learning algorithms in MASs (Buşoniu, Babuška, & De Schutter, 2010).

Instead of researching agents that receive common rewards, Littman (1994) discusses a multi-state MAS in which two Q-learning agents play a zero-sum game: one in which they receive inverse individual rewards, like for example the Matching Pennies stage game in Table 3.

Considering that the opponent's behaviour might play a vital role when applying MARL in such a system, Littman (1994) models it using a SG rather than an MDP: thereby being the first to do so in MARL research. The paper proposes a new algorithm which uses this model, called minimax-Q: derived from the basic Q-learning algorithm in Section 2.2, minimax-Q makes several adjustments to perform well in zero-sum games.

One of these adjustments is to condition the agent's policy on the actions taken by its opponent, using the extra information provided by the SG model. It behaves as to maximise its reward in the worst case, under the assumption that their opponent will try to minimise it. Littman (1994) shows that this algorithm leads to more stable results in comparison with regular Q-learning, deploying the agents in a football-like grid world to provide supporting data.

Furthermore, Littman and Szepesvári (1996) provides a proof that minimax-Q converges to the correct Q-values in two-player zero-sum games: as strong theoretical guarantees for MARL are scarce, the development of minimax-Q was fundamental for analytical research in this field. The general idea behind minimax-Q has also contributed to recent breakthroughs like Silver et al. (2016), as for example discussed in Zhang et al. (2019).

Another example of seminal MARL research is Claus and Boutilier (1998). Instead of using the MDP or SG models, they research the dynamics of MARL in repeated games. Similarly to Tan (1993), this concerns an empirical study based on applying a modified Q-learning algorithm in situations where agents receive common rewards: however, several aspects of this paper differ significantly.

Rather than the communicative agents studied by Tan (1993), Claus and Boutilier (1998) considers the performance of joint action learners (JALs) in relation to independent ones. By keeping Q-values for every joint action, JALs have more information at their disposal: however, as mentioned earlier in this section, this does not directly translate to improved results. To make an informed choice about which action to take, JALs keep a model detailing their assumptions about their opponents' policies: this is achieved with a variant of the fictitious play algorithm described in Section 2.4.

Furthermore, Claus and Boutilier (1998) incorporates more game-theoretical elements than its predecessors. It specifically focuses on convergence to an optimal NE over time by applying optimistic policies: these serve as an incentive for their opponents to take the action leading to the optimal reward, even though it might not be beneficial in the short term. This notion of optimistic proactivity will be explored further in Section 3.2.

Note that this practice requires the additional information possessed by JALs: independent Q-learning agents cannot display this behaviour, as they have no concept of the rewards of individual joint actions.

Taking a step back, we have shown multiple examples of seminal research studying MARL in games with common rewards and zero-sum games. The next section will discuss agents designed for general-sum games: those in which agents can prefer different joint actions without necessarily having directly opposed interests.

3.2 MARL in general-sum games

An early example of MARL in general-sum games is Littman (2001). It introduces an algorithm based upon minimax-Q, called Friend-or-Foe Q-learning (FFQ). An generalisation of minimax-Q was not a new concept at the time: Hu and Wellman (1998) investigated this as well, which resulted in the Nash-Q algorithm. However, we will mainly discuss FFQ as it improves on the properties of Nash-Q while using a more simplistic approach.

To explain the relation between Nash-Q and FFQ, consider the following types of Nash equilibria discussed in Littman (2001). A coordination equilibrium is a NE in which all agents achieve their maximum individual reward, whereas an adversarial equilibrium is a NE in which all agents cannot change their policy without making one of their opponents better off.

If one of these equilibria exists in a game, Nash-Q provably converges to it under strict conditions: the Q-values have to be updated according to the global optimum or the saddle point of every stage game encountered (Hu & Wellman, 2003). Furthermore, Nash-Q requires an approximate model of the opponent's Q-table in order to update its own Q-values.

In contrast, FFQ only has one condition: the agent has to be told beforehand whether its opponent is a 'friend' or a 'foe'. Facing a friend, the algorithm will try to achieve a coordination equilibrium: against foes, it will try to achieve an adversarial one.

To converge to these equilibria, FFQ employs a combination of two distinct algorithms. Against friends, it plays the action which leads to the highest individual reward when its opponents coordinate with it. This is an optimistic algorithm similar to the one used in Claus and Boutilier (1998), but Littman (2001) uses an ϵ-greedy exploration strategy rather than Boltzmann exploration. Against foes, FFQ simply applies the minimax-Q algorithm described in Section 3.1.

While the FFQ algorithm mainly works well in games with common rewards and zero-sum games, it has relatively strong theoretical guarantees supported by earlier work on minimax-Q and Nash-Q. However, there are multiple limitations to this approach: the algorithm needs to be told the type of opponent it is facing, and is not guaranteed to converge to Nash equilibria other than coordination or adversarial ones. The author notes that learning the type of opponent faced is an interesting avenue for further research: this suggestion partially underlies the direction of this project.

Littman and Stone (2001) takes a different approach to general-sum games. It investigates the value of agents that use a game-theoretical fixed strategy when facing a Q-learning agent, comparing their performance to agents that use Q-learning themselves. The reasoning behind this idea is that Q-learning agents often take the role of a 'follower': they adopt a policy which performs well against the actions of their opponent. In contrast, the fixed strategies use an approach more akin to a 'leader': they choose an action which will lead to a good result, given that their opponent follows their cues.

To this end, two separate fixed strategies are presented. One of them is

'Bully', which teaches the opponent to follow their lead in order to reach a beneficial compromise. It does so by consistently taking the action which leads to the highest individual reward, under the assumption that the opponent's strategy is to play the best response to each action.

The other strategy is 'Godfather', which works with a system of reward and punishment similar to the tit-for-tat strategy (Axelrod, 1984). First, Godfather chooses an action which leads to a mutually beneficial reward. While the opponent plays the action that leads to this reward, Godfather maintains the situation by doing so as well: however, Godfather forces their opponent into the worst case scenario when they play a different action. The efficiency of this approach depends on whether the opposing agent has some concept of history: if this is the case, they can learn how to respond adequately.

Two variants of Q-learning were used as opponents for the fixed strategies described above: one without a concept of history, and one which conditioned on the previous action of its opponent. While the Bully strategy performed well against both types, the Godfather strategy frequently required the latter Q-learning agent as its opponent in order to be effective.

As these agents employ static strategies, they cannot adapt to agents which specifically try to exploit their behaviour. Littman and Stone (2001) mentions that an agent which displays leader- and follower-like behaviour with regard to its opponent could be a topic for future research: this idea will support our research questions in Section 4.

3.3 Goals of MARL research

Thus far, we have mainly discussed the relations between the games, models and approaches studied in MARL research: a concise overview of the main papers which constituted this discussion is displayed in Table 4.

Paper	Target games	Model
Tan (1993)	Common reward	MDP
Littman (1994)	Zero-sum	SG
Claus and Boutilier (1998)	Common reward	Repeated game
Littman (2001)	General-sum	SG
Littman and Stone (2001)	General-sum	Repeated game

Table 4: An overview of the main papers discussed thus far.

However, we have not yet specifically targeted the relations between the goals of these papers: in order to clearly specify the aims of this project, these will now be discussed.

Indicating the goals of MARL research has been a topic surrounded by debate. For example, Shoham, Powers, and Grenager (2007) specify five separate agendas: computational, descriptive, normative, prescriptive (cooperative) and prescriptive (non-cooperative). They argue that MARL research should be divided between these agendas to provide clarity regarding the goals that are

pursued. To clarify these agendas, we quote Tuyls and Parsons (2007):

> (1) Computational: learning algorithms are a way to compute the properties of a game.
> (2) Descriptive: learning algorithms describe how natural agents learn in the context of other learners.
> (3) Normative: learning algorithms give a means to determine which sets of learning rules are in equilibrium with one another.
> (4) Prescriptive, cooperative: learning algorithms describe how agents should learn in order to achieve distributed control of dynamic systems.
> (5) Prescriptive, non-cooperative: learning algorithms describe how agents should act to obtain high rewards.

Within this context, we are mostly interested in the prescriptive agendas: those concerned with how agents should learn. This interest reflects in the papers discussed thus far: both Tan (1993) and Claus and Boutilier (1998) can be seen as instances of the prescriptive (cooperative) agenda, while Littman (1994) can be viewed as prescriptive (non-cooperative) MARL research.

In line with commentary provided by Tuyls and Parsons (2007), our interests lie at the intersection of the proposed agendas: most importantly, the intersection between the two prescriptive varieties. To address these interests, we will analyse the objectives of the reviewed literature in order to place the goals of this project in context.

One of the main objectives in MARL research is to learn a policy which converges to an equilibrium given certain restrictions. These restrictions can be broadly divided into two categories: those related to opponents, and those pertaining to the environment.

For example, Claus and Boutilier (1998) discusses convergence to the NE which grants the highest individual reward in self-play. This is studied in the context of two repeated games with multiple Nash equilibria: agents have to take high-risk actions in order to reach the NE with the highest value. The paper empirically shows that the optimistic agents are capable of taking these risks, leading them to converge to a more favourable NE in comparison to regular Q-learning.

Convergence to a NE is also one of the objectives of Littman (2001). Given that either an adversarial or a coordination equilibrium exists, this paper proves that FFQ will always converge to the corresponding equilibrium policy regardless of its opponent: however, it has to know which of the two equilibria to expect in advance.

Another objective is to minimise the degree to which an agent's strategy can be exploited. This is one of the goals in Littman (1994), which proves that minimax-Q will reach the best possible reward in the worst case. In the zero-sum games for which it was designed, this means that the agent will at least break even regardless of the other agents involved.

Conversely, an agent can also aim to exploit a certain type of opponent: this is the case in Littman and Stone (2001). Against opponents that always play

a best response to their actions, the leading strategies obtain a higher average reward compared to Q-learning.

Upon reflection, the goals of these papers are often specifically tailored to certain situations. To improve generalisation, an agent could try to identify the strategy of its opponent by analysing their behaviour: based upon this information, it can then adopt an advantageous counter-strategy.

In addition to improved rewards, this approach could also provide a notion of explainability: it allows the agent to provide a reason why it is using a specific counter-strategy. There has been recent interest in algorithms with this property by both industry and academics (Gunning, 2017; Adadi & Berrada, 2018), as it helps humans to understand the otherwise 'black box' behaviour of a learning agent. This notion will be further investigated in the next section, and will also be addressed by the research questions.

3.4 Combined approaches

As mentioned in Section 3.2 and 3.3, an agent which uses a combination of multiple approaches in response to its opponent's behaviour seems to be a worthwhile direction. This is the idea behind the AWESOME algorithm (Conitzer & Sandholm, 2007) for repeated games, which will be explained below.

The AWESOME (Adapt When Everyone is Stationary, Otherwise Move to Equilibrium) algorithm combines fictitious play with a strategy that converges to a NE in self-play. Which approach to take is determined by the behaviour of the opponent.

If the agent observes that its opponent is playing a stationary strategy, it applies fictitious play to compute the best response to this behaviour. This procedure involves consistently playing a certain action, which is changed when the history indicates that taking another action would significantly improve the rewards obtained.

When the agent observes that its opponent is playing a non-stationary strategy, it acts according to the strategy which leads to a NE in self-play: the specific policy used for this purpose is computed in advance. When the opponent changes from stationary to non-stationary behaviour, the history used by fictitious play is discarded.

The method used by the AWESOME agent to determine whether the opponent is playing a stationary strategy uses the difference between several sequences of play, also called epochs: when the opponent's action distribution in this epoch is too dissimilar to that of the previous epoch, this hypothesis is dropped. Similarly, the method used to determine whether the opponent is playing a non-stationary strategy is the difference between the opponent's action distribution in this epoch and the action distribution which would have resulted from the precomputed NE policy.

By using this method to switch between the two approaches, the AWESOME algorithm is able to adapt to various types of opponents: one of which is the fixed-or-best-response opponent discussed in Littman (2001). Conitzer and Sandholm (2007) also proves that this algorithm reaches optimal play against

stationary opponents, and always converges to a NE in self-play. This makes it the first algorithm of its kind to do so without accessing the policies used by the other agents: the authors argue that these two conditions should be a requirement for multi-agent learning in general.

Note that the AWESOME algorithm does not use Q-learning, which has been heavily featured in this proposal thus far. As it uses fictitious play in combination with a fixed NE policy, it falls within the scope of GT research as discussed in Section 2.4.

Concluding Section 3, we have discussed agents from relevant MARL literature which have been designed under different assumptions. The strategies of these agents will be studied by the research questions, which are proposed in the next section.

4 Research questions

In Sections 2 and 3, we have seen examples of agents that apply a strategy in order to achieve their goal: this results in the agent exhibiting a certain behaviour. We will research how this behaviour can be used to identify the underlying strategy, such that the resulting knowledge can be applied to provoke favourable actions.

The experiments will study the behaviour of agents in repeated 2x2 games: this is the most basic model of interaction which possesses the following three fundamental properties. It considers two agents, which is the minimum number of agents required for interaction. Furthermore, the agents can choose between two actions: this is the minimum number of actions which allows for multiple potential strategies. Finally, the stage game is played repeatedly by the same agents: this allows for analysis of their behaviour over time.

To enable identification of the exact strategy which underlies this behaviour, the number of potential strategies has to be demarcated. This limitation is required because the probability distribution over actions which results from a strategy is continuous: thus, the number of potential strategies for a repeated 2x2 game is infinite.

As we want to identify a specific strategy, we will therefore reduce the number of potential strategies to a set of candidates. These candidate strategies will be selected from the literature covered in Section 3, as well as the background on GT in Section 2.4.

With regard to the points above, we will investigate the following research questions:

- How can an agent's strategy be identified efficiently from a set of candidate strategies, based upon the agent's behaviour in repeated 2x2 games?
  - What are properties of strategies and stage games that contribute to distinguishability?
  - What is the influence of the stage game's properties on the efficiency of identification?

– What is the influence of the candidate set on the efficiency of identification?

– Which methods for behavioural analysis are appropriate for which combinations of candidate sets and stage games?

To answer these questions, we have developed an agent which solely aims to efficiently identify the strategy of its opponent: the agent is indifferent to the individual reward it obtains. The agent's strategy consists of taking actions which allow for effective analysis of its opponent's behaviour. We will research the influence of two main factors: the properties of the current stage game and the candidate set which is currently under consideration.

We will illustrate the importance of these factors with two examples: one which considers the same candidate set for different stage games, and one which considers different candidate sets for the same stage game. For the sake of clarity, the strategies which will be discussed in the examples are explained below.

- Pure 0: Take the first action.
- Random: Take either action with equal probabilities.
- Single Nash: Take the action which represents your part of the stage game's NE. To avoid ambiguity of this description, the examples only consider stage games with one NE.
- ϵ-greedy Q: Apply the Q-learning algorithm as described in Section 2.2: take either action according to the ϵ-greedy exploration method.

Different stage games

To show the importance of the current stage game's properties, consider a candidate set consisting of the Random and Single Nash strategies: the opponent is playing either strategy with equal probabilities.

These strategies can be distinguished in some stage games. For example, consider the Prisoner's Dilemma shown in Table 2. The NE for this stage game is (b, β), as established in Section 2.4. In this example, we will assume our agent is the R player with the C player as its opponent: note that all following statements hold vice-versa as well, as the stage game is symmetrical.

To support that these two strategies can be distinguished, we consider the expected behaviour of both. The Random strategy will take either action with equal probabilities: in contrast, the Single Nash strategy will always take action β. With this in mind, the agent can identify the strategy of its opponent.

If the opponent takes action α, the Random strategy is being applied: the Single Nash strategy cannot exhibit this behaviour. When the opponent continually plays action β, the probability that it is applying the Single Nash strategy increases each iteration. However, there is still a possibility that the opponent is applying the Random strategy regardless of how many successive iterations

it took action β: though unlikely, this behaviour could still have happened by chance.

To prevent the agent from not making a judgment at all in this type of situation, we implemented a confidence threshold: the agent confirms that the opponent applies a certain candidate strategy when this threshold has been surpassed. This presents an inherent margin of error, as the agent is not entirely certain whether its judgment is correct: however, we do not presently believe this can be circumvented.

Regardless of the specific value of the confidence threshold, the agent will eventually identify either strategy in this example when given enough iterations. In contrast, consider the same candidate set for the Matching Pennies stage game shown in Table 3. The NE for this game is mixed, which is achieved by both agents taking either action with equal probabilities.

As a result of this NE, the expected behaviour of the Single Nash strategy is identical to that of the Random strategy: thus, the two candidate strategies are indistinguishable by their behaviour. As they could be distinguished in the Prisoner's Dilemma, this example shows that analysis of an opponent's behaviour is influenced by the properties of the current stage game.

Different candidate sets

To illustrate that different candidate sets influence how well a certain candidate strategy can be identified for a given stage game, consider the Random and Pure 0 candidate strategies for the Prisoner's Dilemma. These can be distinguished in the same manner as the Random and Single Nash strategies in the previous example: the opponent's strategy can be identified when action β is taken, or when the probability of the Pure 0 strategy exceeds the confidence threshold.

In contrast, consider the Random and ϵ-greedy Q strategies as the candidate set. Furthermore, suppose the exploration parameter ϵ is initialised to 1 and decays over time. In this case, the behaviour of both strategies is initially indistinguishable: both candidate strategies take either action with equal probabilities. Thus, the confidence threshold will likely not be surpassed until after this initial period of frequent exploration.

With this in mind, consider the case where an opponent is applying the Random strategy in the Prisoner's Dilemma. The examples above show that the speed with which the agent is able to identify this strategy depends on the current candidate set. To take this factor into account, we will analyse multiple candidate sets per stage game.

Methods of analysis

In addition to the two examples above, we will discuss the impact of the agent's analytical method: which method is appropriate for which combination of candidate sets and stage games will be a main point of focus in this project.

For each combination, we will evaluate the distinguishability of each pair of candidate strategies within the candidate set. This represents the speed

with which an agent can confirm or rule out either of the strategies within the confidence interval, given its current method of analysis.

Which method the agent adopts influences the distinguishability of a candidate pair: to illustrate this, consider the Prisoner's Dilemma once more. Assume the candidate pair consists of the Random strategy, alongside the tit-for-tat strategy: the latter takes action β when its opponent has taken b in the previous iteration, and α otherwise.

If the agent consistently takes either a or b, these strategies can be easily distinguished: the tit-for-tat strategy will consistently take a single action as well, which is likely to contrast the behaviour displayed by the Random strategy. However, the action distributions of the two candidate strategies will be similar if the agent takes either action with equal probabilities: therefore, distinguishability is influenced by the method adopted by the agent.

Counter-strategies

Finally, we will illustrate the possibilities with regard to the goal briefly mentioned at the beginning of this section: an agent which applies knowledge of its opponent's strategy to provoke them to take favourable actions. This agent would apply the acquired knowledge to devise a beneficial counter-strategy.

As our approach leads to the identification of an exact strategy, we hypothesize that our agent could apply more specific counter-strategies compared to agents which only approximate their opponent's strategy. Furthermore, the agent might be able to use the acquired knowledge as motivation for their counter-strategy of choice.

For example, consider a game in which the candidate set consists of the minimax-Q strategy discussed in Section 3.1 and the game-theoretical minimax strategy. Both strategies are designed with the same idea in mind: maximising the individual reward under the assumption that the opponent will try to minimise it. To this end, minimax can access the reward of each joint action directly: in contrast, minimax-Q approximates its quality.

Suppose this is the candidate set for the Stag Hunt game shown in Table 3. In this stage game, the idea described above results in a strategy which consistently takes action β: this leads to a minimum reward of 2, compared to the minimum reward of 1 for action α. Regardless of the action taken by the agent, this leads to a suboptimal individual reward for both agents.

However, this situation can be circumvented when the opponent's strategy is identified as minimax-Q. If the agent uses this knowledge to apply a counter-strategy which consistently takes action α, the opponent's Q-values for (b, α) and (b, β) will stay at 0.

Therefore, the opponent effectively has to choose between a reward of 4 or 3: this means that minimax-Q will converge to consistently taking action α rather than β, which results in the optimal joint action. Though the candidate strategies were designed with the same idea in mind, this example shows that identifying a specific strategy can be beneficial.

5 Experiments

To provide answers to the research questions discussed in Section 4, we set up several experiments. These experiments were carried out by means of a hand-built framework, which will be presented in Section 5.1. The implementation details of our agent will then be discussed in Section 5.2. Afterwards, Section 5.3 will outline the specific setup of each experiment. Finally, Section 5.4 presents the corresponding results.

5.1 Framework

The experimentation framework consists of a list of stage games and a list of strategies, which are subsequently used as a baseline for the general experiment procedure. This section will discuss these three components in order.

5.1.1 Stage games

The stage games used in the experiments were selected with two requirements in mind. First, each stage game should consider two agents and two actions: in line with the goals of the project. Second, this 2x2 stage game should be significantly different from all others used in the experiments: this notion is defined below.

We consider two stage games to be significantly different if their Nash equilibria and Pareto optima are not identical: this should also hold for equivalent games obtained by mirroring the originals. A joint action is a Pareto optimum if taking any other joint action would reduce the reward granted to at least one of the agents.

The stage games which were selected for the experiments are shown in Tables 5 and 6. Table 5 first illustrates the stage games, after which Table 6 provides individual details.

While most of the notation used in these tables has been used previously in this thesis, we will clarify one extra concept. As mentioned briefly in Section 4, recall that a NE can either be pure or mixed. In Table 6, mixed Nash equilibria are displayed using fractional notation. This notation represents the probability that the agent will take either action a or α: otherwise, action b or β is taken.

Note that each stage game is significantly different from the other stage games as per the definition at the start of this section: there is no pair of stage games with both identical Nash equilibria and identical Pareto optima.

We hypothesize that these fundamental differences between stage games correlate with the efficiency of the identification process. More specifically, we think that the Nash equilibria and Pareto optima indicate strategic options that could influence the actions taken by agents playing the stage games: this hypothesis will be evaluated in Section 6.1.

0	α	β
a	$3, 2$	$0, 0$
b	$0, 0$	$2, 3$

1	α	β
a	$0, 0$	$-1, 1$
b	$1, -1$	$-9, -9$

2	α	β
a	$1, 1$	$0, 0$
b	$0, 0$	$0, 0$

3	α	β
a	$2, 2$	$0, 0$
b	$0, 0$	$1, 1$

4	α	β
a	$1, 1$	$0, 3$
b	$3, 0$	$2, 2$

5	α	β
a	$1, -1$	$-1, 1$
b	$-1, 1$	$1, -1$

6	α	β
a	$3, 3$	$0, 5$
b	$5, 0$	$1, 1$

7	α	β
a	$4, 4$	$1, 3$
b	$3, 1$	$2, 2$

Table 5: The stage games used in the experiments.

#	Name	Nash equilibria	Pareto optima
0	Bach or Stravinsky	$(a, \alpha), (b, \beta), (\frac{3}{5}a, \frac{2}{5}\alpha)^1$	$(a, \alpha), (b, \beta)$
1	Chicken	$(a, \beta), (b, \alpha), (\frac{8}{9}a, \frac{8}{9}\alpha)^1$	$(a, \alpha), (a, \beta), (b, \alpha)$
2	Coordinate First	(a, α)	(a, α)
3	Coordination	$(a, \alpha), (b, \beta)$	(a, α)
4	Deadlock	(b, β)	$(a, \beta), (b, \alpha), (b, \beta)$
5	Matching Pennies	$(\frac{1}{2}a, \frac{1}{2}\alpha)^1$	$(a, \alpha), (a, \beta), (b, \alpha), (b, \beta)$
6	Prisoner's Dilemma	(b, β)	$(a, \alpha), (a, \beta), (b, \alpha)$
7	Stag Hunt	$(a, \alpha), (b, \beta), (\frac{1}{2}a, \frac{1}{2}\alpha)^1$	(a, α)

Table 6: Details of each stage game in Table 5.

5.1.2 Strategies

Next, we will discuss the strategies used in the experiments. As mentioned in Section 4, these have been selected based upon both the literature covered in Section 3 as well as the background on GT in Section 2.4: descriptions of each strategy are listed below. For notational convenience, we will refer to actions a and α as the first action: similarly, actions b and β will be referred to as the second action.

- Pure 0: Take the first action.
- Pure 1: Take the second action.
- Random: Take either action with equal probabilities.

[1] These mixed Nash equilibria can differ between variants of their respective stage games, as they depend on the specific rewards granted.

- Nash: Take the action which represents your part of a random NE. For example, suppose an agent applying this strategy plays the Bach or Stravinsky stage game. At the start of each experiment, one of the three Nash equilibria is selected: this NE then dictates the probabilities of taking either action for the entire repeated game.
- TFT: Initially take the action which represents your part of a random Pareto optimum. Afterwards, mimic the action of your opponent: take the first action if your opponent has taken the first action in the previous iteration, and the second action otherwise. This strategy generalises the tit-for-tat algorithm mentioned in Section 3.2, putting the emphasis on replication of the opponent's previous action.
- ϵ-FP: Apply the Fictitious Play algorithm, as described in Section 2.4. If the difference between the expected rewards is smaller than ϵ, take either action with equal probabilities. If not, take the action with the highest expected reward.
- Relative FP: Apply the Fictitious Play algorithm. Normalise the expected rewards between 0 and 1: take either action with the resulting probabilities.
- ϵ-greedy Q: Apply the Q-learning algorithm as described in Section 2.2: take either action according to the ϵ-greedy exploration method.
- Boltz-Q: Apply the Q-learning algorithm: take either action according to the Boltzmann exploration method, rather than ϵ-greedy. This method uses a decaying temperature parameter to guide the exploration-exploitation trade-off. The exact equation used to determine the action probabilities is defined as follows:

$$p(x) = \frac{e^{q_x/\tau}}{\sum_i e^{q_i/\tau}}$$

  In this equation, $p(x)$ represents the probability of taking action x. Furthermore, q_x represents the Q-value of action x while τ represents the temperature parameter. Applying this equation results in action probabilities which depend on both the relative Q-values and the current degree of exploration indicated by the temperature.

These strategies were chosen based upon their ability to support a proof-of-concept of the identification process: therefore, complex strategies were mostly omitted. However, we also wanted to indicate the value of this approach with regard to learning strategies. Broadly speaking, the resulting selection can be classified as follows:

- Static strategies: Pure 0, Pure 1, Random, Nash.
- Dynamic strategies (non-learning): TFT, ϵ-FP, Relative FP.

- Dynamic strategies (learning): ϵ-greedy Q, Boltz-Q.

Selecting these strategies of varied complexity enabled us to evaluate our approach against different types of opponents: an analysis of our agent's performance with regard to each opponent type will be provided in Section 6.1.

5.1.3 Experiment procedure

In each experiment, our agent repeatedly plays a single stage game from Table 5 against its opponent: implementation details of this agent will be discussed in Section 5.2. The opponent applies one of the strategies from the candidate set, which is in turn a subset of the strategies listed in Section 5.1.2.

In our experiments, we will research candidate sets of size 2: these will be referred to as candidate pairs. This restriction provides the opportunity to study every candidate set individually: the number of possible combinations is relatively small. It also reduces the number of variables in each experiment, which improves the clarity of the results.

At the start of each experiment, the opponent's strategy is selected from the candidate set with equal probabilities. Afterwards, our agent tries to identify this strategy given the candidate set. This is achieved by empirically evaluating each candidate strategy's behaviour: our agent cannot access their implementation details. The experiment ends when identification takes place, or when the set maximum number of iterations has been reached.

The experiment is then evaluated, based upon whether our agent identified a strategy. If this is the case, the identified strategy is compared to the opponent's true strategy: if the two strategies are equal, the experiment is marked as a success. If the identification was incorrect or if our agent did not identify a strategy due to reaching the maximum number of iterations, the experiment is marked as a failure.

This experiment is repeated a set number of times, after which our agent is evaluated based upon the number of successful experiments: we refer to this as the "accuracy". Furthermore, the mean number of iterations which was required for successful identification is recorded. These two metrics will form the basis of our results in Section 5.4, which illustrate our agent's performance in different settings.

5.2 Agent implementation

This section discusses the implementation details of our agent, which solely aims to efficiently identify the strategy of its opponent: from this point onward, we will refer to it as the Identification Agent (IA). The IA has two main features: the identification process itself, and a look-ahead strategy which attempts to optimise this process. These components will be discussed individually in Section 5.2.1 and 5.2.2 respectively.

5.2.1 Identification process

The IA identifies the strategy of its opponent as follows. It keeps a probability distribution over the candidate set, indicating the likelihood of each candidate being the opponent's strategy. These probabilities are initially equal to $\frac{1}{n}$, with n defining the number of candidate strategies. In the case of our experiments, they will be equal to $\frac{1}{2}$ as $n = 2$.

After each iteration of the stage game, the IA updates these candidate probabilities based upon their respective action probabilities: the latter is defined as the current probability that the candidate strategy will take each action. For example, the action probabilities of the Random strategy are always equal to 0.5 for both actions.

The IA cannot access the action probabilities directly: they are private variables that belong to the candidate strategy. Instead, the IA approximates these probabilities by sampling each candidate's actions given the joint action history.

After sampling a candidate's actions for a set number of times, the observed frequency of each action is normalised between 0 and 1 to approximate the action probabilities. The candidate probability is then multiplied by the approximated action probability of the action taken by the IA's opponent.

When each candidate probability has been updated in this manner, the resulting probability distribution is normalised between 0 and 1. Each candidate probability is then compared to the confidence threshold mentioned in Section 4. If one of these probabilities exceeds this threshold, the IA has identified the opponent's strategy: if not, the experiment advances to the next iteration. An overview of the identification process is shown in Figure 3.

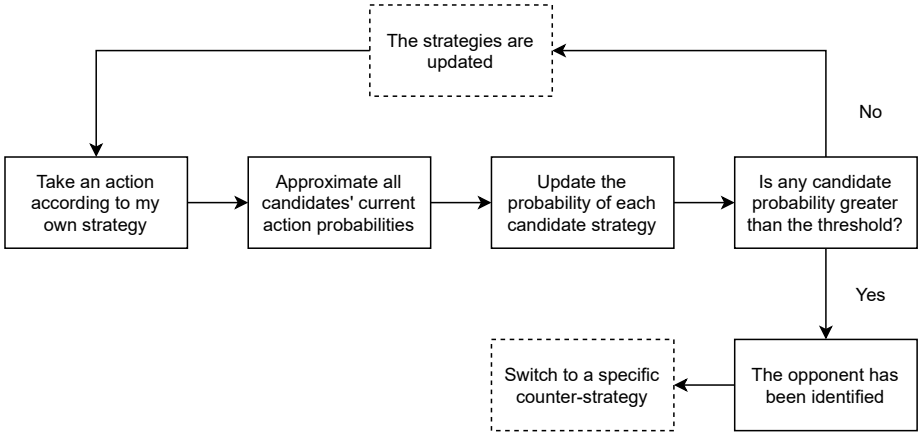

Figure 3: The identification process of the IA.

We will illustrate this process with an example. Suppose the IA is trying to identify the opponent's strategy in the Prisoner's Dilemma, with the candidate

set consisting of two strategies: Pure 0 and Random. The IA is playing as R, while the opponent is playing as C.

Assume the IA itself applies the Pure 1 strategy: a strategy which specifically targets efficient identification will be discussed in Section 5.2.2. The opponent applies one of the strategies from the candidate set: in this example, we will assume the Pure 0 strategy is selected. Furthermore, the confidence threshold will be set to a relatively low value of 0.8.

The initial candidate probabilities are $\frac{1}{2}$ for both Pure 0 and Random, as the opponent could be applying either strategy with equal probabilities from the IA's perspective. As both agents' strategies lead to a static action, the joint action in this iteration is guaranteed to be (b, α).

After this joint action has been taken, the IA samples the actions of both candidate strategies. The Pure 0 strategy will always play action α: regardless of the number of samples, this leads to an approximated action probability of 1 for α and 0 for β. As the action taken by the opponent was α, the candidate probability of Pure 0 is adjusted to $\frac{1}{2} * 1 = 0.5$.

In contrast, the Random strategy takes either action with equal probabilities. Suppose that the IA takes a low number of samples, and concludes that Random currently takes action α with probability 0.4. As a result, the candidate probability of Random is adjusted to $\frac{1}{2} * 0.4 = 0.2$.

The candidate probabilities are then normalised between 0 and 1, resulting in $\frac{0.5}{0.5+0.2} = \frac{5}{7} \approx 0.71$ for Pure 0 and $\frac{0.2}{0.5+0.2} = \frac{2}{7} \approx 0.29$ for Random. As neither value exceeds the confidence threshold of 0.8, the experiment advances to the next iteration.

As both agents' strategies are static, the joint action taken in the next iteration is (b, α) once more. Similarly to the previous iteration, the IA approximates the probability that each candidate strategy would currently take action α. The approximated probability of Pure 0 playing this action is not subject to change: however, suppose sampling the Random strategy results in the IA approximating a probability of 0.6 for action α this time.

This results in respective candidate probabilities of $\frac{5}{7} * 1 = \frac{5}{7}$ and $\frac{2}{7} * 0.6 = \frac{6}{35}$. Omitting the full calculation for readability, these probabilities are then normalised to $\frac{25}{31} \approx 0.81$ and $\frac{6}{31} \approx 0.19$. As $0.81 \geq 0.8$, the confidence threshold has been surpassed: thus, the IA identifies the opponent's strategy as Pure 0. As Pure 0 is also the true strategy used by the opponent, the IA has correctly identified the opponent's strategy in two iterations.

5.2.2 Look-ahead strategy

In the example given in Section 5.2.1, the IA took actions according to the Pure 1 strategy. Its own strategy can influence the efficiency of the identification process: this will be further discussed in Section 6.1.

However, the strategies listed in Section 5.1.2 have not been designed for this purpose: they are either static, or designed to optimise the individual reward. This prompted us to design a strategy that specifically targets efficient identification, which was achieved by using a look-ahead strategy.

As the name implies, a look-ahead strategy looks several iterations into the future to determine the best course of action. This allows for optimisation of the identification process: by evaluating the consequences of both of its actions against each candidate strategy, the IA estimates which action will lead to faster identification on average.

The look-ahead strategy is divided into three phases: the exploration phase, the simulation phase and the backpropagation phase. Each phase will now be discussed in more detail.

Exploration phase

In the exploration phase, the look-ahead strategy evaluates the consequences of all possible joint actions up to a set number of iterations: we will call this the "maximum depth". It is worth noting that the optimal version of this strategy does not require such a limit, which would result in an evaluation of every possible sequence of joint actions.

However, this approach is infeasible due to performance constraints: as the number of evaluations required increases up to fourfold per iteration, a maximum depth is required for non-trivial identifications.

Setting a maximum depth resolves the performance issue outlined above, but results in the look-ahead strategy not identifying the opponent's strategy at all for some joint action sequences: specifically, those for which none of the associated candidate probabilities have passed the confidence threshold at the maximum depth.

In these cases, the look-ahead strategy advances to the simulation phase. Rather than evaluating all possible joint actions, the strategy evaluates simulations of the remaining iterations: a detailed description of this phase will be provided in the next section. An example which visualises the look-ahead strategy up until the simulation phase is shown in Figure 4.

Similarly to the example given in Section 5.2.1, Figure 4 depicts a situation in which the IA is playing as R while its opponent plays as C. The current depth i is represented by D_i: the maximum depth in this example is 2.

The look-ahead strategy starts at the current situation, which we call the "root" or R. It then evaluates all possible joint actions: (a, α), (a, β), (b, α) and (b, β). This evaluation consists of applying a single iteration of the identification process, as depicted in Figure 3.

If a strategy is identified, the look-ahead strategy has finished evaluating the joint action sequence in question: in Figure 4, these situations are highlighted in green. The specific joint actions for which this holds have been selected arbitrarily in this example, as they cannot be determined without further context.

Note that each evaluation is conducted separately: the results of evaluating one joint action do not influence the evaluation of others at the same depth. Until the maximum depth has been reached, this process repeats for all joint actions for which evaluation has not concluded.

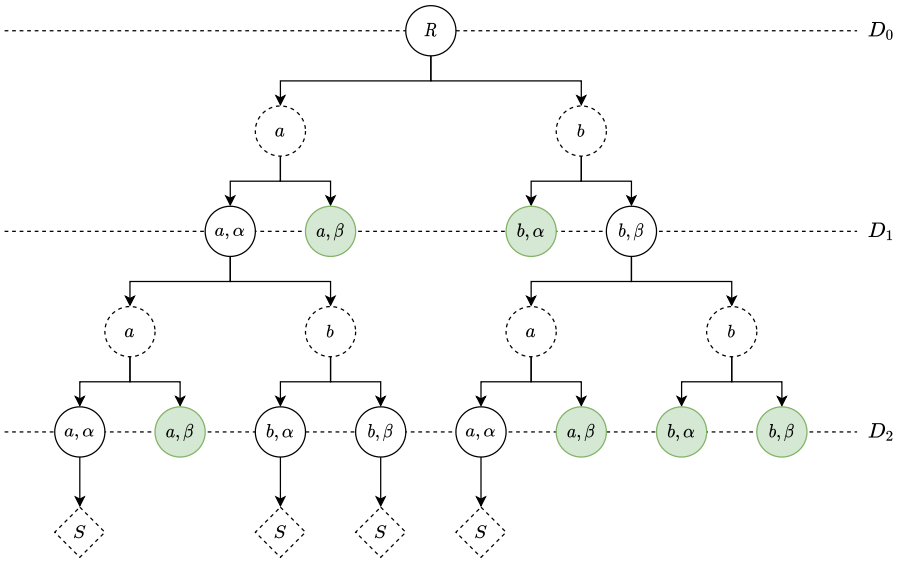

Figure 4: The look-ahead strategy up until the simulation phase.

Simulation phase

As stated previously, the look-ahead strategy advances to the simulation phase when the maximum depth has been reached but the opponent's strategy has not been identified yet. This phase is marked with an S in Figure 4, and consists of a set number of simulations.

Instead of explicitly considering every possible joint action, a simulation consists of the IA and the candidate strategy playing the stage game from the current situation onward: in these simulations, the IA changes its own strategy from look-ahead to Random.

The simulation is stopped when identification takes place due to reaching the confidence threshold, or when the maximum number of iterations has been reached. As with the evaluation of joint actions in the exploration phase, each simulation is conducted separately.

This process is repeated for the number of simulations dictated by the parameter settings. Afterwards, the joint action which lead to the simulation phase receives a value that represents the average number of simulated iterations.

Backpropagation phase

After all simulations have concluded, the look-ahead strategy advances to the backpropagation phase. In this phase, it calculates the most efficient action with regard to the candidate strategy currently under consideration: Figure 5 visualises an example calculation for the situation depicted in Figure 4.

We will use Figure 5 to clarify how the look-ahead strategy selects the most

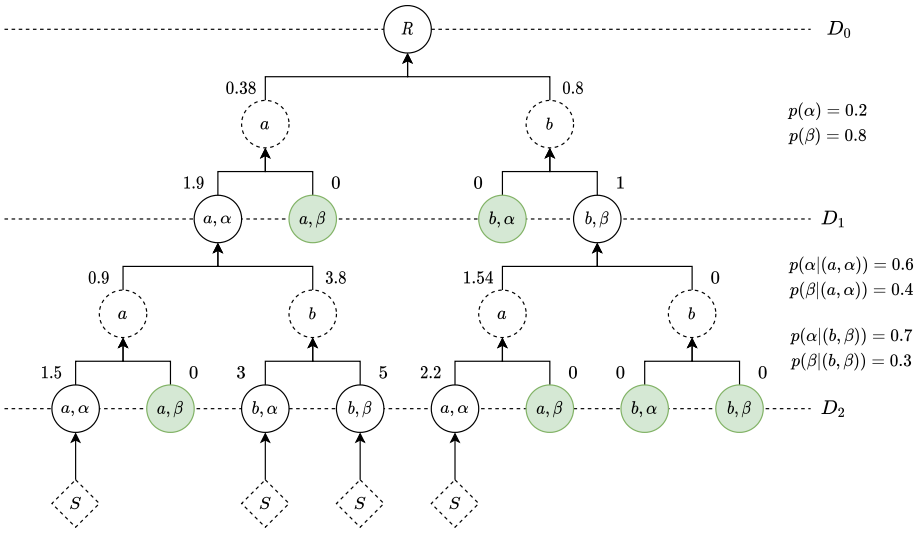

Figure 5: The backpropagation phase of the look-ahead strategy.

efficient action. In the previous two phases, all joint actions at D_2 have received a value: this value represents the mean number of iterations required for identification from that point onward.

If the joint action lead to identification in the exploration phase, this value is equal to 0: no additional iterations are required, as identification has already been achieved. This is the case for the joint actions highlighted in green.

If identification was not achieved, the joint action's value is equal to the average number of iterations in the simulation phase. This holds for the four non-highlighted joint actions at D_2: as this example has no specific context, their values were chosen arbitrarily in this case.

Using the values of each joint action at D_2, the look-ahead strategy can calculate the value of each of its own actions: a and b. This requires the candidate strategy's approximated action probabilities, which depend on the history of joint actions. These probabilities are displayed on the right side of Figure 5: as with the simulation results, their values were chosen arbitrarily in this case.

We will illustrate how the look-ahead strategy calculates the value of one of its own actions with an example, which concerns action a after taking (a, α) at D_1. The result of taking this action can either be (a, α) or (a, β). The simulations estimated that (a, α) requires another 1.5 iterations on average to to identify the opponent: in contrast, (a, β) leads to direct identification.

In addition to the values of (a, α) and (a, β), the value of a also depends on the joint action history. In this case, the history is (a, α): the corresponding approximated action probabilities are $p(\alpha|(a, \alpha)) = 0.6$ and $p(\beta|(a, \alpha)) = 0.4$.

These probabilities represent the chance that the opponent will take either action α or β. By multiplying them with the values of the resulting joint actions,

the value of taking action a is calculated: in this case, this results in a value of $1.5 * 0.6 + 0 * 0.4 = 0.9$.

After calculating the value of all four of its own actions at D_1 in this manner, the look-ahead strategy can determine the value of the joint actions at D_1. Once more, this value is 0 if the exploration phase lead to identification.

If this is not the case, the joint action is assigned the lowest value of the two actions available to the look-ahead strategy: it assumes the most efficient action will always be taken. For example, this is 0.9 in the case of (a, α) at D_1. This value is then incremented by one to account for the iteration required to reach the next depth: resulting in a final value of 1.9.

After calculating the values of all joint actions, the look-ahead strategy calculates the value of its own actions at D_0: applying the process described previously. In this case, this evaluation results in values of $0.2 * 1.9 + 0.8 * 0 = 0.38$ and $0.2 * 0 + 0.8 * 1 = 0.8$ for taking action a and b respectively at D_0.

As the values of taking a and b at D_0 have now been calculated, evaluation of the efficiency of either action against this candidate strategy has concluded: after saving these values, the look-ahead strategy continues by applying the entire process to the next candidate strategy.

When the values of both actions have been calculated with respect to each candidate strategy, the IA takes the action which was assigned a lower value on average. This results in a strategy which specifically targets efficient identification of the opponent's strategy: a comparison between this approach and the strategies listed in Section 5.1.2 will be provided in Section 5.4.

5.3 Setup

This section will describe the setup of the experiments displayed in Section 5.4, which provide the data required to answer the research questions in Section 4. To obtain answers to the main research question, we will first investigate its sub-questions: the experiments related to each individual question are described below.

To test the properties of strategies that contribute to distinguishability, we will research the differences between the three types of strategies mentioned in Section 5.1.2: static, dynamic (non-learning) and dynamic (learning). Furthermore, the properties of stage games will be investigated with regard to their Nash equilibria and Pareto optima as discussed in Section 5.1.1.

To evaluate the influence of the candidate set on the efficiency of identification, we will research the performance of individual candidate pairs averaged over all stage games. This performance will be measured in terms of the metrics discussed in Section 5.1.3. Similarly, the influence of the stage game's properties will be investigated by comparing the average performance of all possible candidate pairs in each individual stage game.

Finally, we will discuss which methods for behavioural analysis are appropriate for which combinations of candidate sets and stage games. To this end, we will investigate the possible benefits of using the look-ahead strategy for different such combinations.

5.4 Results

This section discusses the results of the experiments. Before discussing the results themselves, we will first introduce the parameter settings which were used: while implementation details have been provided throughout this thesis, some extra information is required to enable full re-enactment of the experiments.

- Regarding the experiment: The maximum number of iterations within which the opponent's strategy has to be identified is 10: if this limit is exceeded, identification has failed. Furthermore, the metrics which will be provided in this section average over 100 individual experiments.
- Regarding the IA: The number of samples taken from the candidate strategies in order to approximate their action probabilities is 100. Either the Random or the look-ahead strategy is applied: this is indicated per experiment. The confidence threshold required for identification is 0.99.
- Regarding the look-ahead strategy: The maximum depth is set to 1, as this requires significantly less computational resources in comparison to the maximum depth of 2 discussed in Section 5.2.2. The value of a joint action that reaches the simulation phase averages over 100 individual simulations. The maximum number of simulated iterations equals the maximum number of iterations remaining at the lowest depth: with the current settings, this value ranges from 0 to 9.
- Regarding the strategies: The value of ϵ used by ϵ-FP is 0.01. The ϵ-greedy Q strategy uses an ϵ value of 0.2. Boltz-Q uses an initial temperature of 1, which decays with 0.001% in each iteration. Finally, both ϵ-greedy Q and Boltz-Q use a learning rate of 0.1 and a discount factor of 0.9.

We used the settings above to run pair-wise experiments, in which all possible candidate sets of size 2 were evaluated for every stage game. Sections 5.4.1–5.4.4 will now present several sets of experiments, which are accompanied by observations that will be relevant to our conclusions in Section 6.1.

5.4.1 Individual stage games

Results from two individual stage games are displayed in Figures 6 and 7: in both stage games, the IA used the look-ahead strategy to distinguish each candidate pair. Similar figures for the other stage games can be found in Appendix A.

Figures 6 and 7 display the following information. The titles of their sub-figures denote the stage game that was played and the statistic that was measured. The latter can be either "accuracy" or "mean": definitions of these terms are given in Section 5.1.3.

The sub-figures themselves then provide this statistic for each candidate pair. The accuracy is within $[0, 1]$: this represents the percentage of times the opponent's strategy was identified correctly. The required mean iterations is

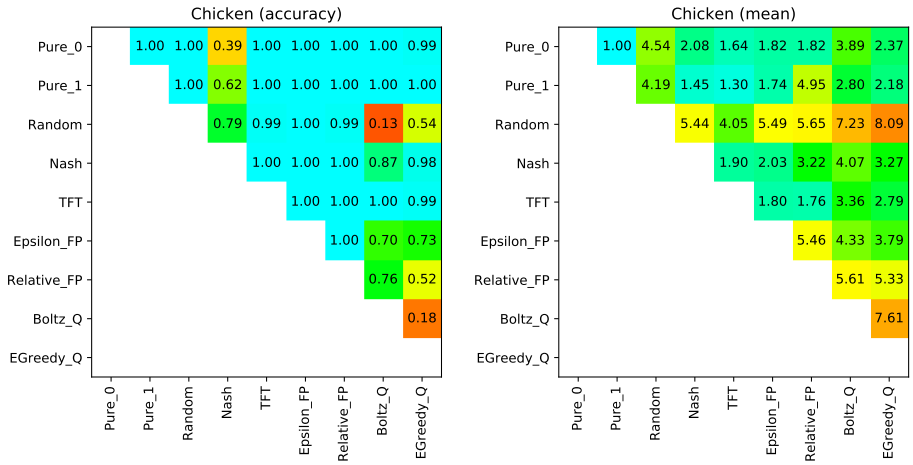

Figure 6: The accuracy of identification (l) in Chicken, along with the mean iterations required (r).

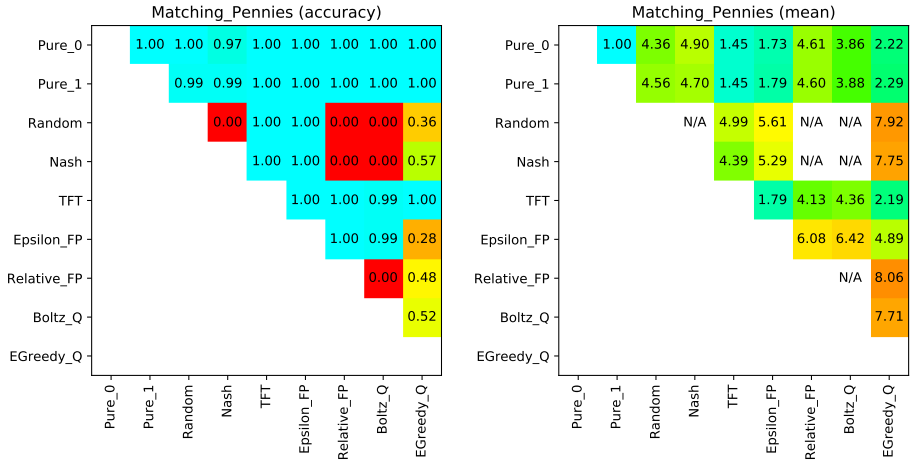

Figure 7: The accuracy of identification (l) in Matching Pennies, along with the mean iterations required (r).

within $[1, 10]$: the IA always requires at least one iteration to distinguish a candidate pair, while 10 is the current maximum number of iterations.

For the sake of clarity, duplicate statistics are removed by only displaying the upper triangular matrix. The main diagonal is excluded as well: candidate pairs consisting of identical strategies cannot be distinguished.

Furthermore, each statistic has been given a background colour: blue and green entries illustrate higher distinguishability than orange and red ones, while

results in between are coloured yellow.

Note that Figure 7 includes "N/A" as a result in some cases: this represents that no valid data was available. The cause of this result is the failure of all experiments for that particular candidate set: more specifically, the IA was never able to identify the opponent's strategy within the maximum number of iterations. This is reflected in the accuracy as well, which is 0 in those cases.

With regard to these figures, we will provide a few relevant observations. In Figure 7, notice that the IA was unable to distinguish any combination of the Random, Nash, Relative FP and Boltz-Q strategies in the Matching Pennies stage game.

This contrasts all other stage games used in the experiments, where candidate pairs consisting of these strategies can at least be distinguished to some extent. For example, Figure 6 reports an accuracy of over 70% for all but one of these pairs in the Chicken stage game.

Another interesting observation specifically concerns the statistics of the Random and Boltz-Q pair in Figure 6. While this pair is nearly indistinguishable in all other stage games, Chicken seems to have the properties required to enable identification in some cases.

5.4.2 Random versus look-ahead

We will now compare the performance of the IA's default strategy (Random) to that of the look-ahead strategy. Figures 8 and 9 respectively show the accuracy and the required mean iterations for each candidate pair, averaged over all stage games.

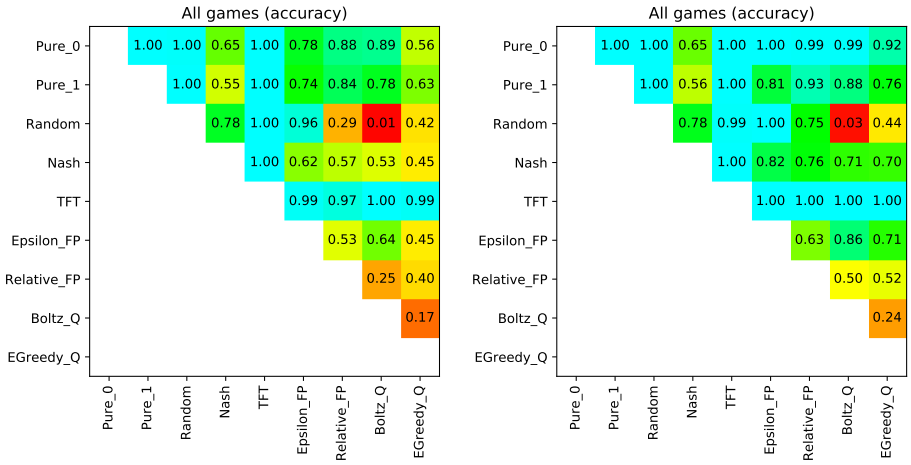

Figure 8: The accuracy of each candidate pair while using either the Random (l) or the look-ahead (r) strategy, averaged over all stage games.

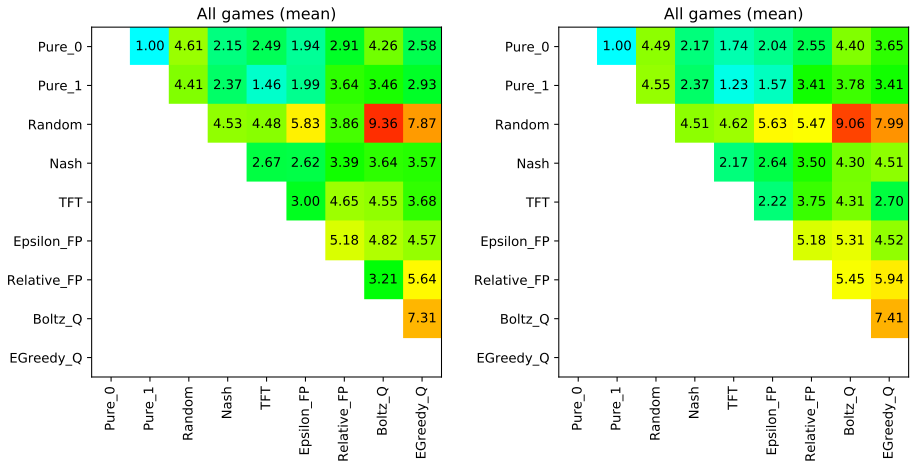

Figure 9: The mean iterations required to distinguish each candidate pair while using either the Random (l) or the look-ahead (r) strategy, averaged over all stage games.

Figure 8 shows that each strategy is identified with various degrees of efficiency, depending on the other strategy in the candidate pair. There seems to be one exception to this statement: the TFT strategy, which achieves near-perfect accuracy in all cases.

However, the specific candidate set is still of relevance in this case. To support this claim, we refer to Figure 9. This figure shows that the mean iterations required to distinguish candidate pairs that include the TFT strategy does vary: for instance, pairing TFT with Pure 1 leads to more efficient identification on average compared to pairing TFT with Random.

Additionally, note that some candidate pairs lead to extreme results. For instance, the pair consisting of the Random and Boltz-Q strategies is nearly indistinguishable.

The means displayed in Figure 9 are not only related to the accuracy, but also to the number of times all experiments failed for a given candidate pair: Figure 10 displays how many times this was the case for each pair. Additionally, Figure 11 illustrates the performance changes in Figures 8 and 9 respectively.

Figure 11 provides several points of discussion. First, note that candidate pairs consisting of two static strategies experience little to no change in distinguishability: as these strategies are unable to react to the IA's actions, their behaviour remains the same regardless of the IA's own strategy.

Furthermore, we will highlight that using the look-ahead strategy influences the Random and Relative FP candidate pair the most: the accuracy of identifying this pair increases with 46%. Finally, note that an increase in accuracy frequently comes at the expense of an increased number of required mean iterations. These observations will be further discussed in Section 6.1.4.

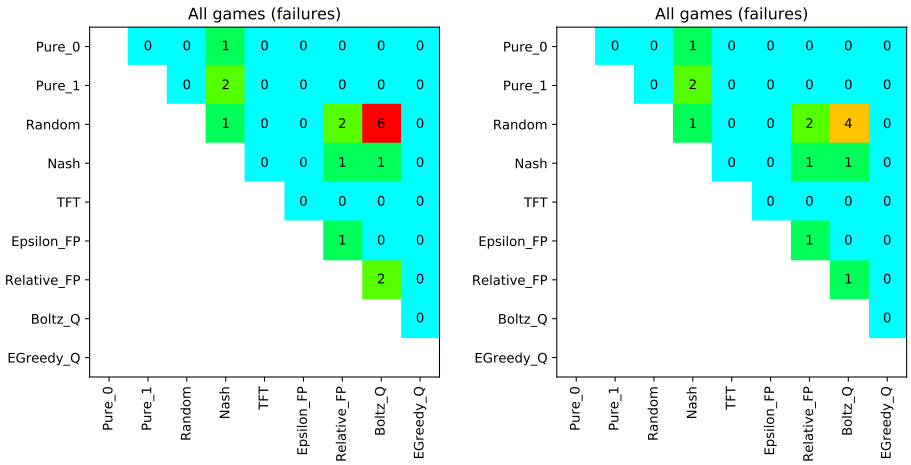

Figure 10: The number of times all experiments failed while using either the Random (l) or the look-ahead (r) strategy.

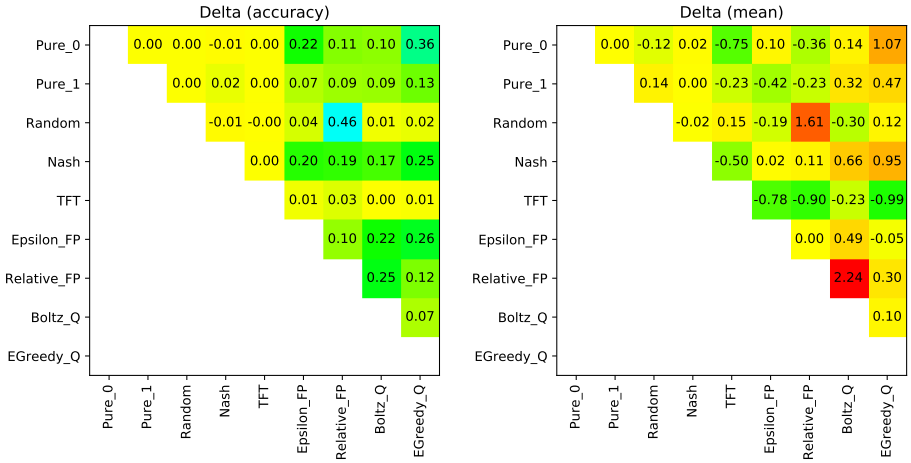

Figure 11: The changes in accuracy (l) and mean iterations required (r) as a result of using the look-ahead strategy.

5.4.3 Statistics per stage game

Providing another point of view, Tables 7 and 8 display the average accuracy and required mean iterations for each stage game: averaged over all candidate pairs. The total number of failures is displayed as well, which has been defined in Section 5.4.1. Afterwards, Table 9 displays the difference between these two tables.

Stage game	Avg. accuracy	Avg. mean	Failures
Chicken	0.79	3.84	0
Coordination	0.74	3.71	1
Bach or Stravinsky	0.71	3.80	1
Coordinate First	0.71	3.84	3
Matching Pennies	0.71	4.49	6
Stag Hunt	0.68	3.92	3
Prisoner's Dilemma	0.64	3.56	1
Deadlock	0.63	3.37	2

Table 7: Statistics per stage game (Random).

Stage game	Avg. accuracy	Avg. mean	Failures
Chicken	0.87	3.61	0
Coordination	0.87	3.76	1
Bach or Stravinsky	0.85	4.04	1
Coordinate First	0.85	4.08	3
Prisoner's Dilemma	0.77	3.90	1
Stag Hunt	0.75	4.20	1
Matching Pennies	0.75	4.30	6
Deadlock	0.70	3.48	1

Table 8: Statistics per stage game (look-ahead).

Stage game	Avg. accuracy	Avg. mean	Failures
Bach or Stravinsky	0.14	0.24	0
Coordinate First	0.14	0.24	0
Coordination	0.13	0.05	0
Prisoner's Dilemma	0.13	0.34	0
Chicken	0.08	-0.23	0
Deadlock	0.07	0.11	-1
Stag Hunt	0.07	0.28	-2
Matching Pennies	0.04	-0.19	0

Table 9: Differences between Tables 7 and 8.

Note that both tables rank the stage games in roughly the same order. This ranking leads us to believe that our hypotheses require some amendments: if they provided the right picture, Coordinate First would be located near the bottom of the table while Matching Pennies would rise to the top. Furthermore, notice that Matching Pennies leads to a noteworthy number of failures: these observations will provide a baseline for the discussion in Section 6.1.1.

5.4.4 Statistics per strategy

Finally, Tables 10–12 provide the accuracy and mean iterations required for each individual candidate strategy: averaging over all their respective candidate pairs in every stage game.

Strategy	Avg. accuracy	Avg. mean	Failures
TFT	0.99	3.37	0
Pure 0	0.84	2.74	1
Pure 1	0.82	2.66	2
ϵ-FP	0.71	3.74	1
Random	0.68	5.62	9
Nash	0.64	3.12	6
Relative FP	0.59	4.06	6
ϵ-greedy Q	0.53	5.08	0
Boltz-Q	0.51	4.77	9

Table 10: Statistics per candidate strategy (Random).

Strategy	Avg. accuracy	Avg. mean	Failures
TFT	1.00	2.84	0
Pure 0	0.94	2.75	1
Pure 1	0.87	2.67	2
ϵ-FP	0.85	3.64	1
Relative FP	0.76	4.41	5
Nash	0.75	3.27	6
Random	0.75	5.79	7
Boltz-Q	0.66	5.02	6
ϵ-greedy Q	0.65	5.50	0

Table 11: Statistics per candidate strategy (look-ahead).

Strategy	Avg. accuracy	Avg. mean	Failures
Relative FP	0.17	0.35	-1
Boltz-Q	0.15	0.25	-3
ϵ-FP	0.14	-0.10	0
ϵ-greedy Q	0.12	0.43	0
Pure 0	0.10	0.01	0
Nash	0.10	0.16	0
Random	0.07	0.17	-2
Pure 1	0.05	0.01	0
TFT	0.01	-0.53	0

Table 12: Differences between Tables 10 and 11.

With regard to these tables, we will provide relevant observations in the context of our hypothesis in Section 5.1.2. This hypothesis assumed that static, dynamic (non-learning) and dynamic (learning) strategies would be increasingly difficult to distinguish.

First, note that the accuracy of both dynamic (learning) strategies is significantly lower than that of their peers: regardless of the IA's own strategy. This indicates that these strategies are generally harder to distinguish within the maximum iterations, which conforms to our hypothesis.

Another result which is in line with our hypothesis is the distinguishability of both Pure strategies. We assumed that these strategies would be easy to distinguish given their low complexity: this seems to indeed be the case, as both are found in the upper echelons of both Table 10 and 11.

However, there are multiple noteworthy deviations from our initial theory. First, note that the TFT strategy was distinguished with near-perfect accuracy: the loss of 1% in Table 10 can likely be attributed to the confidence threshold. This result does not match the hypothesis, as the TFT strategy was assumed to be harder to distinguish than any static strategy.

Another inconsistency is the accuracy of distinguishing both the Random and Nash strategies. Given that these strategies apply a static probability distribution over the actions, we assumed that they would be easier to distinguish than any dynamic strategy: however, this is not the case. Section 6.1 will elaborate on these observations, alongside those presented previously.

6 Discussion

This section discusses the findings of this thesis. First, Section 6.1 reviews the results displayed in Section 5.4 in the context of the research questions provided in Section 4. Afterwards, Section 6.2 discusses several examples of possible future work.

6.1 Conclusions

Initially, Sections 6.1.1–6.1.4 will discuss the sub-questions in order. Section 6.1.5 then summarizes the acquired knowledge to provide insights with regard to the main research question.

6.1.1 Properties of strategies and stage games

The first sub-question stated the following: "What are properties of strategies and stage games that contribute to distinguishability?". We will discuss these properties separately, starting with those of strategies.

Properties of strategies

Our hypothesis with regard to the properties of strategies that contribute to distinguishability is mentioned in Section 5.1.2. We divided the strategies used

in the experiments into three categories: static, dynamic (non-learning) and dynamic (learning). We assumed that strategies within these categories would be increasingly difficult to distinguish, as we based the categories upon the strategies' supposed complexity.

This hypothesis can be evaluated using the observations in Section 5.4.4. For instance we noted that the TFT strategy was distinguished with near-perfect accuracy: in contrast to all other strategies, which depended heavily on the specific candidate set.

To explain this result, we will approach the strategies' distinguishability from a different perspective. Rather than the complexity of the strategy itself, suppose that distinguishability can be measured by the degree to which a strategy results in unique behaviour in comparison to its peers.

When looking at the results from this angle, the distinguishability of the TFT strategy can be derived from its uniqueness. Most of the strategies used in the experiments either apply a static strategy or adapt to their opponents with the goal of obtaining the highest individual reward. In contrast, TFT matches the opponent's previous action without explicit regard for the consequences.

Furthermore, Section 5.4.4 noted that the Random and Nash strategies were not as easy to distinguish as we hypothesized. If the uniqueness of these strategies is considered, this could be explained as follows.

With regard to the Random strategy, many of the strategies used in the experiments incorporate some stochastic elements as well: this could lead them to occasionally exhibit roughly the same behaviour. On the other hand, the Nash strategy displays the same behaviour as one of the Pure strategies when using any pure NE as its baseline.

These similarities to other strategies used in the experiments could be the reason for the relatively low accuracy values: if the behaviour of two strategies is roughly the same, distinguishing them becomes more complex.

Concluding, we found that the apparent complexity of a strategy is seemingly not the only factor which contributes to its distinguishability: additionally, one likely has to consider the uniqueness of the strategy's behaviour in comparison to its peers. We hypothesize that in general, a strategy's distinguishability can be described as a combination of these two factors: further research is required to make more specific statements.

Properties of stage games

The properties of stage games that we assumed would contribute to distinguishability are discussed in Section 5.1.1. We introduced our notion of significant difference, which concerns the combination of Nash equilibria and Pareto optima in a given stage game.

We hypothesized that these two concepts would provide strategic options to the agents playing the stage game. For instance, the Coordinate First stage game has one NE: the associated joint action is identical to that of the sole Pareto optimum.

By using these two concepts as an indication of the joint actions that agents are likely to pursue, we assumed that this single strategic option would result in them exhibiting similar behaviour. This could in turn diminish the presence of the strategies' individual features, leading to low distinguishability.

On the contrary, a stage game such as Matching Pennies seemingly provides many strategic options. There is no clear joint action which should be taken, as all joint actions are Pareto optima: furthermore, none of these options are matched by the single NE. We hypothesized that this type of stage game would lead to divergent behaviour, resulting in high distinguishability.

These hypotheses can be evaluated with the help of Section 5.4.3. Within this section, we observed that the Coordinate First and Matching Pennies stage games respectively ranked higher and lower than expected: furthermore, there were relatively many candidate pairs for which all experiments failed in Matching Pennies.

These results could be explained as follows. First, a straightforward stage game such as Coordinate First might actually lead to high distinguishability in some cases: though this stage game does indeed cause most dynamic strategies to roughly exhibit the same behaviour, this leads to a situation in which deviations from this action pattern are relatively easy to pick up.

In the case of Coordinate First, the specific situations in which an agent takes the second action are therefore more indicative of its strategy in comparison to stage games that encourage mixed action patterns.

Second, the low distinguishability in the Matching Pennies stage game might be related to its NE. This equilibrium encourages playing each action with equal probabilities, which might lead dynamic strategies to devolve into a strategy similar to Random: leading to a lower distinguishability on average.

Additionally, this NE could be the reason for the high number of failures in Matching Pennies. If two strategies would take each action with near-equal probabilities as a result of adaptation, they are unlikely to be distinguished within the maximum number of iterations. This problem also occurs when such strategies have to be distinguished from the Random strategy.

Note that this argument cannot be made for stage games in which dynamic strategies roughly exhibit the same behaviour as one of the Pure strategies: there is a high probability that a dynamic strategy will still take the other action at some point, due to the frequent application of stochastic elements mentioned earlier in this section.

6.1.2 Influence of the stage game's properties

The second sub-question concerns the influence of the stage game's properties on the efficiency of identification. While Section 6.1.1 has already answered this question to a large extent, this section will discuss two examples to further highlight this influence. These examples are supported by the supplementary data in Appendix A.

In Section 5.4.1, we noted that the Random, Nash, Relative FP and Boltz-Q strategies could not be distinguished in Matching Pennies. As mentioned

in Section 6.1.1, this is likely due to the stage game's properties encouraging strategies to take each action with equal probabilities.

Furthermore, we observed that the Chicken stage game reported a significantly improved accuracy for all but one combinations of these strategies: further substantiating that the properties of a specific stage game can have a large influence on the efficiency of identifying certain candidate pairs.

Finally, Section 5.4.1 mentioned that the Chicken stage game could sometimes distinguish the Random and Boltz-Q strategies: in contrast to all other stage games. This could be the result of the large negative reward for taking (b, β): as this reward presents Boltz-Q with a strong incentive to refrain from reaching this joint action again, it will likely respond by taking the first action more frequently. As a result, its behaviour diverges from that of Random: enabling more efficient identification.

6.1.3 Influence of the candidate set

The third sub-question stated: "What is the influence of the candidate set on the efficiency of identification?". In Section 4, we gave an example which illustrated that the specific candidate set can influence the efficiency of identifying a specific strategy. In this section, we will argue that this holds for all strategies in the current setting.

Section 5.4.2 noted that the candidate set influences the accuracy of identification for all strategies except TFT. However, the candidate pair consisting of TFT and Pure 1 could be distinguished in less iterations on average in comparison to TFT and Random.

This is likely the case because the former pair can be distinguished immediately when the opponent takes the first action: Pure 1 cannot exhibit this behaviour, which means the opponent's strategy is TFT by elimination.

In contrast, Random could always have taken any action pattern by chance: therefore, it cannot be ruled out based upon a single action. As additional iterations are required as a result, identification is less efficient on average.

We also observed in Section 5.4.2 that the Random and Boltz-Q pair was nearly indistinguishable. This could be a result of the current parameter settings: the Boltz-Q strategy might not have experienced enough iterations to enable a noticeable shift from exploration to exploitation, which results in behaviour that is largely the same as that of the Random strategy.

Concluding this section, we have seen that the efficiency of identification is always influenced by the candidate set in some way: there is a high degree of variation depending on the specific strategies involved. We hypothesize that this influence is related to the probability of the candidate strategies displaying similar behaviour, which seems to form the baseline for the observations presented.

6.1.4 Methods for behavioural analysis

The last sub-question states: "Which methods for behavioural analysis are appropriate for which combinations of candidate sets and stage games?". To this end, we mainly investigated the effects of the IA using the look-ahead strategy rather than the Random strategy: this represents the variable part of the behavioural analysis, which otherwise consists of the identification process displayed in Figure 3.

A few observations relevant to this question were provided in Section 5.4.2. While the look-ahead strategy did not lead to improvements when considering candidate pairs consisting of static strategies, strategy pairs that allowed adaptation to the IA's strategy could be distinguished more effectively in general.

This was most pronounced in the candidate pair consisting of Random and Relative FP, for which the accuracy of identification experienced an increase of 46%. This likely results from the look-ahead strategy allowing the IA to play fixed action patterns: the Relative FP strategy can then follow this lead, thereby making it distinguishable from the Random strategy.

Similar interactions might account for some of the smaller improvements seen in other candidate pairs as well: the large change in this specific pair might be attributed to its relatively low accuracy in the first place.

Furthermore, we noted that an improvement in accuracy was frequently accompanied by an increased number of mean iterations required. This might be the result of the look-ahead strategy enabling some identifications that were previously impossible within the maximum number of iterations. We hypothesize that these identifications still require a relatively large number of iterations, which in turn raises the mean iterations required on average.

6.1.5 Efficient identification

In Sections 6.1.1–6.1.4, we have discussed the sub-questions provided in Section 4. This section will summarize this knowledge to provide our insights with regard to the main research question: "How can an agent's strategy be identified efficiently from a set of candidate strategies, based upon the agent's behaviour in repeated 2x2 games?".

Investigation of the sub-questions lead to the following conclusions. First, the properties of strategies which contribute to distinguishability likely consist of its complexity and its relative uniqueness. Second, the strategic options which are provided by stage games contribute to distinguishability: however, the degree to which these options provoke unique behaviour is more important than the range of possibilities.

Third, the candidate set always influences the efficiency of identification in some way: it is necessary to specifically consider the impact of each set, though some generalisations can be made. Furthermore, individual stage games can drastically affect the efficiency of identifying certain candidate sets as well.

Finally, using a look-ahead strategy is an appropriate method to improve the analysis of the opponent's behaviour in most situations. Those in which

one only deals with static strategies are an exception to this rule, as the extra computational power required does not lead to increased performance: if all strategies in the candidate set are unable to react to their opponent's actions, analysis of their behaviour does not depend on the opponent's strategy.

6.2 Future Work

This section will discuss possible avenues for future work, which apply the information presented in this thesis.

First, recall that our experiments researched candidate pairs: candidate sets of size 2. We expect that studying larger candidate sets will also provide interesting results, as they add an extra dimension to the identification process.

For example, a candidate pair can be distinguished immediately when one of the strategies does not support the action taken: this results in the identification of the other strategy by elimination. However, this is not the case when studying larger candidate sets: when one candidate strategy does not support an action, multiple other options still remain.

Second, most experiments in this thesis involved a look-ahead strategy with a maximum depth of 1. Further research could study the costs and benefits of different maximum depths: however, it could also analyse the results of applying a different strategy for efficient identification.

For instance, one could apply a strategy based upon the Monte Carlo Tree Search (Coulom, 2006) algorithm instead. This algorithm assigns more resources to promising move sequences: in our case, joint actions that frequently lead to identification.

Furthermore, we hypothesized in Section 6.1.1 that a strategy's distinguishability is determined by two factors: its complexity and its uniqueness in comparison to its peers. Further research could try to support this hypothesis by providing a more nuanced view: alternatively, data that dismisses this assumption could be presented.

References

Adadi, A., & Berrada, M. (2018). Peeking inside the black-box: A survey on explainable artificial intelligence (xai). *IEEE Access*, *6*, 52138–52160.

Axelrod, R. (1984). *The evolution of cooperation*. Basic Books.

Bellman, R. (1957). A markovian decision process. *Journal of mathematics and mechanics*, 679–684.

Brown, G. W. (1951). Iterative solution of games by fictitious play. *Activity analysis of production and allocation*, *13*(1), 374–376.

Buşoniu, L., Babuška, R., & De Schutter, B. (2010). Multi-agent reinforcement learning: An overview. In *Innovations in multi-agent systems and applications-1* (pp. 183–221). Springer.

Claus, C., & Boutilier, C. (1998). The dynamics of reinforcement learning in cooperative multiagent systems. *AAAI/IAAI*, *1998*(746-752), 2.

Conitzer, V., & Sandholm, T. (2007). Awesome: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents. *Machine Learning*, *67*(1-2), 23–43.

Coulom, R. (2006). Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games* (pp. 72–83).

Gunning, D. (2017). Explainable artificial intelligence (xai). *Defense Advanced Research Projects Agency (DARPA), nd Web*, *2*.

Hu, J., & Wellman, M. P. (1998). Multiagent reinforcement learning: theoretical framework and an algorithm. In *Icml* (Vol. 98, pp. 242–250).

Hu, J., & Wellman, M. P. (2003). Nash q-learning for general-sum stochastic games. *Journal of machine learning research*, *4*(Nov), 1039–1069.

Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994* (pp. 157–163). Elsevier.

Littman, M. L. (2001). Friend-or-foe q-learning in general-sum games. In *Icml* (Vol. 1, pp. 322–328).

Littman, M. L., & Stone, P. (2001). Leading best-response strategies in repeated games. In *In seventeenth annual international joint conference on artificial intelligence workshop on economic agents, models, and mechanisms.*

Littman, M. L., & Szepesvári, C. (1996). A generalized reinforcement-learning model: Convergence and applications. In *Icml* (Vol. 96, pp. 310–318).

Nash, J. (1951). Non-cooperative games. *Annals of mathematics*, 286–295.

Robinson, D., & Goforth, D. (2005). *The topology of the 2x2 games: a new periodic table* (Vol. 3). Psychology Press.

Shapley, L. S. (1953). Stochastic games. *Proceedings of the national academy of sciences*, *39*(10), 1095–1100.

Shoham, Y., Powers, R., & Grenager, T. (2007). If multi-agent learning is the answer, what is the question? *Artificial intelligence*, *171*(7), 365–377.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., ... others (2016). Mastering the game of go with deep neural networks and tree search. *nature*, *529*(7587), 484.

Singh, S., Jaakkola, T., Littman, M. L., & Szepesvári, C. (2000). Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine learning*, *38*(3), 287–308.

Tan, M. (1993). Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning* (pp. 330–337).

Tsitsiklis, J. N. (1994). Asynchronous stochastic approximation and q-learning. *Machine learning*, *16*(3), 185–202.

Tuyls, K., & Parsons, S. (2007). What evolutionary game theory tells us about multiagent learning. *Artificial Intelligence*, *171*(7), 406–416.

von Neumann, J. (1928). Zur theorie der gesellschaftsspiele. *Mathematische annalen*, *100*(1), 295–320.

von Neumann, J., & Morgenstern, O. (1944). *Theory of games and economic behavior*. Princeton university press.

Watkins, C. J. (1989). *Learning from delayed rewards.* King's College, Cambridge.

Whitehead, S. D. (1991). A complexity analysis of cooperative mechanisms in reinforcement learning. In *Aaai* (pp. 607–613).

Zhang, K., Yang, Z., & Başar, T. (2019). Multi-agent reinforcement learning: A selective overview of theories and algorithms. *arXiv preprint arXiv:1911.10635*.

A Supplementary results

This section depicts the accuracy and mean iterations required for each individual stage game: the IA applies the look-ahead strategy as its own strategy in each case. The parameter settings are the same as those used in Section 5.4.

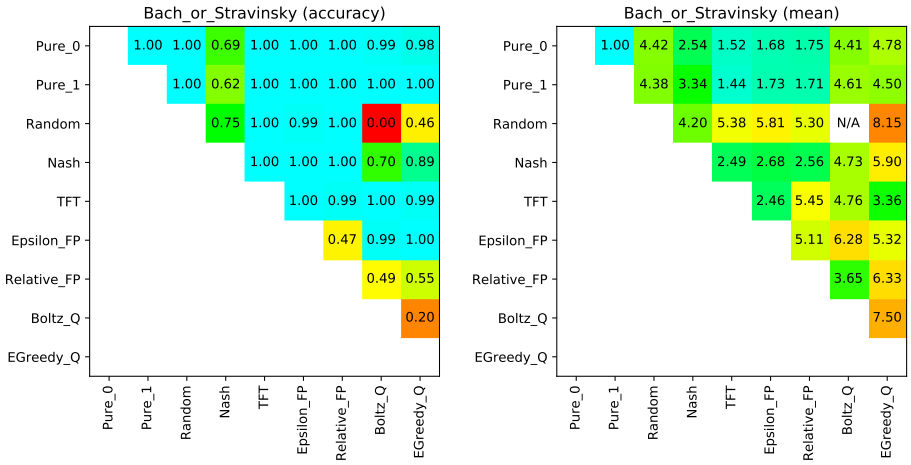

Figure 12: The accuracy of identification (l) in Bach or Stravinsky, along with the mean iterations required (r).

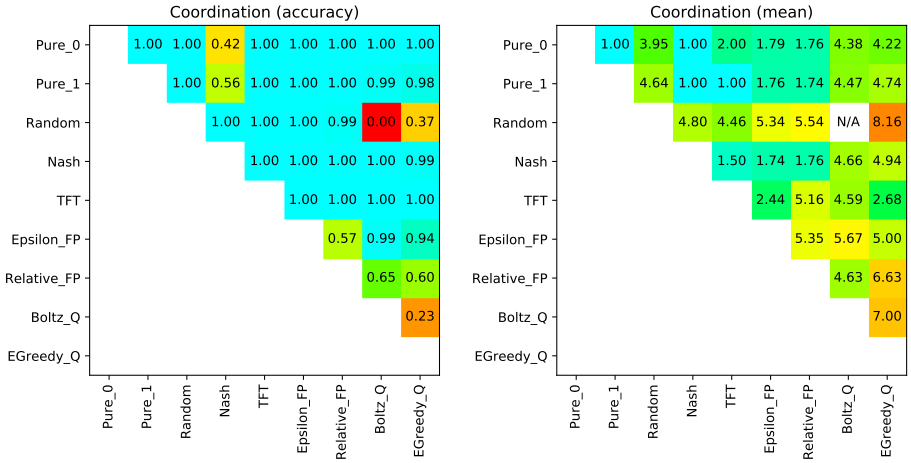

Figure 13: The accuracy of identification (l) in Coordination, along with the mean iterations required (r).

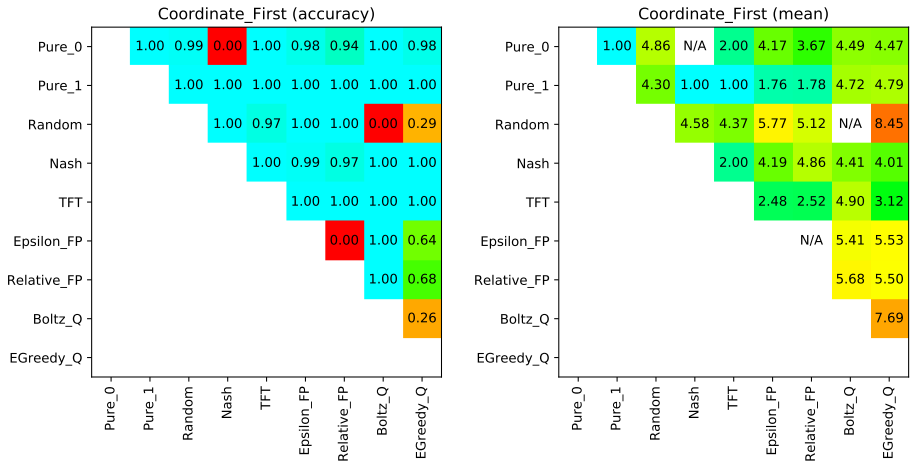

Figure 14: The accuracy of identification (l) in Coordinate First, along with the mean iterations required (r).

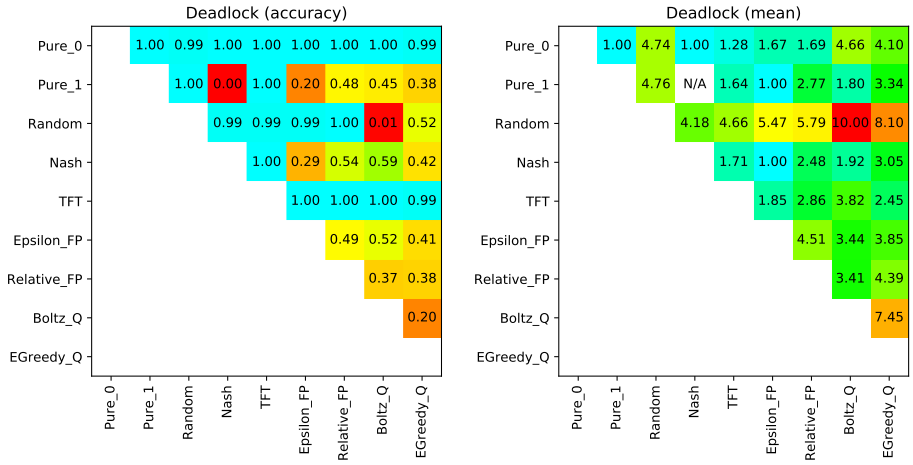

Figure 15: The accuracy of identification (l) in Deadlock, along with the mean iterations required (r).

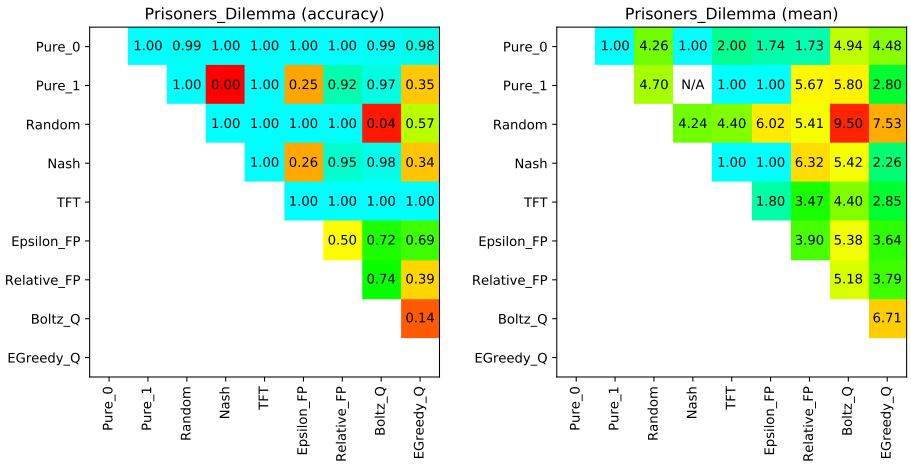

Figure 16: The accuracy of identification (l) in Prisoner's Dilemma, along with the mean iterations required (r).

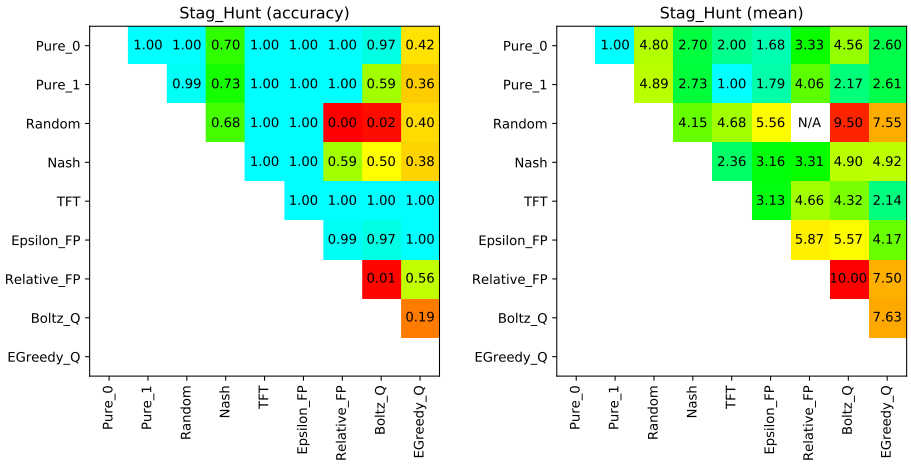

Figure 17: The accuracy of identification (l) in Stag Hunt, along with the mean iterations required (r).