DEPARTMENT OF
APPLIED IT

MIXED MEMORY Q-LEARNER

An adaptive reinforcement learning algorithm for the Iterated Prisoner's Dilemma

Anna Dollbo

Thesis:	15 hp
Program:	Cognitive Science
Level:	First Cycle
Year:	2021
Supervisor:	Robert Lowe
Examiner:	Alberto Montebelli
Report nr:	2021:081

Abstract

The success of future societies is likely to depend on cooperative interactions between humans and artificial agents. As such, it is important to investigate how machines can learn to cooperate. By looking at how machines handle complex social situations, so-called social dilemmas, knowledge about the components necessary for cooperation in artificial agents can be acquired. In this study, a reinforcement learning algorithm was used to study the Iterated Prisoner's Dilemma (IPD), a common social dilemma game. A reinforcement learning algorithm can make decisions in the IPD by considering a given number of its opponent's last actions, thus representing the agent's memory. This study investigated the role of different memory lengths on the performance of the agent in the IPD. The results showed that different memory lengths are preferable depending on the opponent. A new algorithm was created called Mixed Memory Q-Learner (MMQL), which could switch memory length during play to adapt to its opponent. It could also recognise its opponent between games, thus continuing its learning over several interactions. MMQL performed better against certain opponents in the IPD but did not learn to cooperate with cooperative players. Further capabilities might therefore be added to the algorithm to invite cooperation, or the environment can be manipulated. The results suggest that flexibility in how a situation is represented and the ability to recognise opponents are important capabilities for artificial agents in social dilemmas.

Keywords

Machine learning, reinforcement learning, game theory, iterated prisoner's dilemma, state representation, Q-learning.

Sammanfattning

Framgången för framtida samhällen kommer säkerligen bero på lyckade sammarbeten mellan människor och artificiella agenter. Därför är det viktigt att undersöka hur maskiner kan lära sig att samarbeta. Genom att studera hur maskiner hanterar komplexa sociala situationer, så kallade sociala dilemman, kan vi få kunskap om vilka komponenter hos artificiella agenter som krävs för att samarbete ska uppstå. I denna studie användes en förstärkningsinlärnings-algoritm i den repeterade versionen av spelet Fångarnas Dilemma. En förstärkningsinlärnings-algoritm tar i spelet beslut baserat på ett givet nummer av sin motståndares senaste drag. Detta fungerar således som minnet för agenten. En studie gjordes som undersökte betydelsen av olika längd på minne för utfall i spelet. Resultaten visade att olika minneslängd är att föredra beroende på motståndarens strategi. Med denna information skapades en ny algorithm kallad Mixed Memory Q-Learner (MMQL). Denna kunde byta minneslängd under spelets gång för att anpassa sig till sin motståndare. Den kunde också minnas och känna igen sin motståndare från tidigare matcher, och fortsätta lära sig där den sist slutade. Resultaten visade att MMQL presterade bättre mot vissa motståndare, men inte lärde sig att samarbeta med samarbetsvilliga motspelare. För att framkalla samarbete kan därför fler förmågor behöva läggas till hos algoritmen, eller så kan miljön manipuleras. Resultaten tyder på att flexibilitet i hur man representerar en situation, och förmåga att minnas sina motspelare är viktiga kompetenser för artificiella agenter i sociala dilemman.

Nyckelord

Maskininlärning, förstärkningsinlärning, spelteori, fångarnas dilemma, situations-representation, Q-inlärning.

Foreword

Acknowledgements

I would like to express my gratitude to:

- my supervisor Robert Lowe for challenging me to develop my ideas further, for his continued correspondence throughout the project and for sparking my interest in Artificial Intelligence.

- all teachers at the Cognitive Science program for providing a multifaceted, thorough and deeply enriching education.

- my partner Çağın for his loving support and patience with me throughout the project.

Table of contents

1. Introduction **1**

2. Theory **2**

2.1 Game Theory and the Prisoner's Dilemma 2

2.1.1. Game theory 2

2.1.2. The Prisoner's Dilemma 3

2.1.3. The Iterated Prisoner's Dilemma 4

2.1.4. The Axelrod Tournaments 6

2.2. Reinforcement Learning 7

2.2.1. Machine Learning 7

2.2.2. Reinforcement learning 7

2.2.2.2. Elements of reinforcement learning 9

2.2.2.3. Exploration vs. exploitation 9

2.2.3. Q-learning 10

3. Previous studies **12**

3.1. Q-learning in the IPD 12

3.2. Memory in the IPD 13

4. Method **14**

4.1. Design 14

4.2. Materials 14

4.2.1. Axelrod Python Library 14

4.2.2. Strategies 14

4.2.3. Algorithm choice and adaptations 15

4.3. Evaluation Metrics 16

4.4. Test 1 17

4.4.1 Results 17

4.4.2. Interpretation 19

4.4.3. Extension of Test 1 19
5. Mixed Memory Q-Learner 21
5.1. Algorithm description 21
5.1.1. Concept 21
5.1.2. Architecture 22
5.1.3. Limitations 23
5.1.4. Performance prediction 24
5.2. Test 2 24
5.2.1. Design 24
5.2.2. Parameter settings 24
5.2.3. Additional evaluation methods 24
6. Results 25
7. Discussion 32
7.1. Results analysis 32
7.2. MMQL analysis 33
7.3. Limitations 34
7.4. Future research 35
8. Conclusion 35
9. References 36

Acronyms

AI: Artificial Intelligence

APL: Axelrod Python Library

IPD: Iterated Prisoner's Dilemma

MARL: Multi-Agent Reinforcement Learning

MDP: Markov Decision Process

MMQL: Mixed Memory Q-Learner

ML: Machine Learning

MOML: Multi-Objective Meta Learning

MORL: Multi-Objective Reinforcement Learning

PD: Prisoner's Dilemma

RL: Reinforcement Learning

RMSE: Root Mean Square Error

TD: Temporal Difference

TFT: Tit For Tat

1. Introduction

In recent years, research within artificial intelligence (AI) has begun investigating how to teach artificial agents to cooperate. This is deemed an important research area as future societies are likely to require cooperation between artificial systems and artificial systems and humans (Bertino et al., 2020). One way of studying cooperation is by looking at how cooperation emerges from certain tricky situations, so-called social dilemmas. Such dilemmas reveal tensions between collective and individual rationality since cooperation might lead to better outcomes for the group, but freeriding or exploitation can give higher rewards for the individual (Rapoport, 1974). Therefore, studying how AI behaves in such situations is an important measure in ensuring a safe implementation of AI systems into real-world decision-making.

By framing the social dilemma as a game, a game-theoretic framework can be used. One of the most common such games is called the Prisoner's Dilemma (PD). In this game, two players who cannot communicate with each other have to decide whether to cooperate or defect against the other player. If one player chooses to cooperate, but the other does not, the first player receives zero points, and the one who betrayed gets a high payoff. If neither cooperates, they both receive a small payoff, but they both receive a medium high payoff if both cooperate. In this way, choosing to defect against the other player might lead to the highest payoff, but choosing to cooperate is the best choice for both players in the long run. The best strategy in the one-shot PD is always to defect (Axelrod, 1984). However, this is not true in the so-called Iterated Prisoner's Dilemma (IPD), where the game is extended to include multiple interactions with the same agent. The IPD is therefore used to study the emergence of cooperation between different agents (Glynatsi, 2020).

One way of studying AI and cooperation is by testing different machine learning (ML) techniques on games like the IPD to see how these agents learn. Games provide an excellent platform for developing AI as they present complex and interesting problems and realise some of the long term goals of AI, such as creativity, affective interaction and general intelligence (Yannakakis & Togelius, 2018). One ML technique used to study the IPD is called reinforcement learning (RL). In RL, an agent learns to complete a task solely based on the rewards and punishments it receives from the environment (Sutton & Barto, 2018). RL solutions to the IPD are relatively few in the literature (Anastassacos & Musolesi, 2018). However, previous studies on AI in games have uncovered many difficulties and best practices in teaching an agent to learn in an unfamiliar environment. Understanding how these solutions work specifically in non-cooperative games with an asymmetrical payoff matrix is important for AI and cooperation research.

In RL, besides the rewards or punishments received when taking actions in the game, the agent also makes decisions depending on the information available to them in each situation. Therefore, the state-representation (the way a situation is represented at a given time point) of the RL problem can significantly affect how the agent learns to play the game (Yannakakis &

Togelius, 2018). In the IPD, the state representation for an RL algorithm is often some number of the past actions taken by the opponent in the game (Sandholm & Crites, 1996b). This can be likened to the agent's memory, and the agent can thus have a shorter memory (only remember the past action) or a longer memory (for example, the last five actions taken).

Previous research on the IPD often uses tournaments to see which strategies are best in the game (Glynatsi, 2020). The first such tournaments showed that only considering the past action can be sufficient to win the tournament (Axelrod, 1984). However, since then, others have argued that a longer memory is more evolutionarily stable in the game (Li & Kendall, 2014). A study was conducted to answer the research question: how does different memory lengths affect the performance of an RL algorithm in the IPD? The research was done to inspire a new RL approach to the IPD.

2. Theory

2.1. Game Theory and the Prisoner's Dilemma

2.1.1. Game theory

Game theory is a field that analyses situations of conflict, cooperation and competition by mathematical tools and logic (Glynatsi, 2020). Interactions between self-interested agents (so-called *games*) are analysed where the outcome of the interactions depends on what the agents (*players)* think will be the best action for them to take depending on the action they expect the other players will take. The field was first formulated in 1944 by John von Neuman and Oskar Morgenstern and has since been widely used in economics, social sciences and biology (Ross, 2019). A game-theoretic analysis of an interaction asks who the players are, which actions are available to those players, what information different players have when they choose an action and what the payoffs to the different players are as a consequence of the interaction? This frame is then used to reason about the optimal strategy for each player in the game (Jackson, 2011).

Game theory makes two critical assumptions about players in a game: they act to maximise their utility and are perfectly rational agents. Game theoretical analysis supposes that an economic agent has preferences, a concept called *utility* in game theory. The goal of an agent in a game then is to maximise that utility. In its simplest form, that an agent maximises their utility means that an agent acts in a certain way so as to make the something that they tried to bring about by their act more probable (Samuelson, 1938). An agent's utility then can be understood as whichever states of the world that an agent prefers (which can include pure self-interested states or states that also benefit others) and the agent's willingness to act to bring about these states (Leyton-Brown, 2008). Game theory also supposes that players in a game are perfectly *rational* agents (Poundstone, 1993). Rationality here refers to an agent's ability to process information. The agent's expectations of other players' behaviours in the game are rational as they are supposedly derived from a correctly weighted and statistically

sound use of all the information accessible to the agent (Ross, 2019). This conception of rationality has been challenged by Herbert Simon as unrealistic, who argues that rationality is bounded (Simon, 1982). This means that there are constraints to the information, time and processing capacity available to any agent. Still, the notion of perfect rationality is an essential assumption in game theory in order to find how such agents would act if they had perfect knowledge, memory and understanding of the game (Poundstone, 1993).

A standard representation of a game is called a normal form game (Jackson, 2011). Normal form games are usually represented via a table or matrix. Each row corresponds to one possible action for player 1, and each column represents a possible action for player 2. Each of the cells contains the payoff or outcome to both players as a consequence of taking those actions. Player 1's payoff is listed first. In the next section, an example of such a game matrix can be seen which describes the PD.

2.1.2. The Prisoner's Dilemma

PD is one of the most famous games in game theory and an example of a normal form game. It was created in 1950 by RAND scientists Merrill Flood and Melvin Dresher, and was later named "the Prisoner's Dilemma" by their consultant Albert.W. Tucker for the story he told to illustrate it (Poundstone, 1993). The story goes that two people (person A and person B) have been arrested on suspicion of jointly having committed a crime. They are taken to the police station and put in separate rooms with no means of speaking or exchanging messages with each other. The prosecutor comes to each of them and admits that they do not have enough evidence to convict both of them on the principal charge. However, the prosecutor gives both of them a choice: if person A confesses to the crime and testifies against person B in custody, person A will go free. However, if person A does not confess, but person B confesses and testifies against person A, person A will be sentenced to three years in prison. If person A and person B confess to the crime, they will both go to prison for two years. But if none of them confesses, the prosecutor will still charge them for a lighter crime for which there is evidence, and they will both receive one year in prison (Jackson, 2011; Poundstone, 1993).

To cooperate in the PD means to stay silent in the hope that the other player does the same so that both players will get a lighter sentence. To defect refers to confessing against the other player, which can lead to either escaping punishment altogether, or if the other player does the same, going to prison for two years. The game matrix for the PD can be seen in Table 1, where the payoffs represent the time lost to the player while in prison.

Table 1

Prisoner's Dilemma game matrix

	Prisoner B	
	Stays Silent (cooperates)	Betrays (defects)
Prisoner A		
Stays silent (cooperates)	Each serves one year	Prisoner A: gets three years Prisoner B: goes free
Betrays (defects)	Prisoner A: goes free Prisoner B: gets three years	Each serves two years

The PD is a *non-cooperative*, *non-zero-sum* game. In a non-zero-sum game one person's loss is not necessarily another person's gain and vice versa. This can be seen in Table 1's payoff matrix because when both players defect or both players cooperate, the payoff is not zero. The PD is a non-cooperative game because the players cannot make any pact with each other about how they should play. This makes the PD a dilemma because if it was a cooperative game, the players could agree on which strategies to play and wouldn't have to guess the other player's chosen strategy (Rapoport & Chammah, 1965).

The PD is considered the most challenging dilemma for cooperation to arise (Rand & Nowak, 2013). This is because the optimal strategy for the PD is to defect. This is easy to prove: if you are player A and think that player B will cooperate, then the best action to choose is to defect since this means that you will go free. However, if you believe player B will defect, then your best choice is also to defect because if you cooperate, you will get three years in prison, but if you defect, that sentence will be reduced by one year. No matter what you think the other player will do, the best action for player A is always to defect. Defection, therefore, is a *dominant strategy* of the PD, even though cooperation would give a higher cumulative reward for both players (Axelrod, 1984). The PD can be thought of as an abstract formulation of many situations in life where the best outcome for an individual rational agent causes mutual defection when everyone involved would instead benefit from cooperation.

2.1.3. The Iterated Prisoner's Dilemma

The IPD is an extension of the PD where several PD games are repeated (iterated) after one another. The IPD is considered a more realistic implementation of a social dilemma from a human perspective since people often have continuous interactions. When meeting the same people continuously, agents come to know each other and can remember the other agents' previous actions. This way, a person can come to label another person as either "predictable", "selfish", or "cooperative", for example. Thus, the possibility of meeting again enables the

emergence of cooperation since even a selfish player might want to cooperate in the present in order to receive a higher payoff in the future (Axelrod, 1984).

In the IPD, the structure of the traditional PD can be generalised from its original setting to make it more suitable for general-purpose testing. This is done by extrapolating the payoffs from punishments in prison years to rewards while still maintaining the symmetrical relationship. The payoff matrix for the IPD can be seen in Table 2 where R refers to the *Reward for mutual cooperation*, P stands for the *Punishment for mutual defection*, T refers to the *Temptation to defect,* and S stands for *Sucker's payoff.*

Table 2

Prisoner's Dilemma game matrix with numerical payoffs

	Player 2	
	C	D
Player 1		
C	R = 3, R = 3	S = 0, T = 5
D	T = 5, S = 0	P = 1, P = 1

The PD assumes that two players who cooperate will fare better than two players who defect. For this condition to hold, the payoffs are constrained as seen in (1).

$$T > R > P > S \tag{1}$$

The Temptation must be the highest reward, followed by the Reward, then Punishment and lastly, the Sucker's payoff. To make this payoff matrix a social dilemma, another rule must hold, as seen in (2).

$$2R > T + S \tag{2}$$

This rule ensures that two Rewards have a higher payoff than Temptation and Sucker's payoff combined. Applying this condition to the game ensures that even if two players are alternating between cooperation (C) and defection (D) between each round, this behaviour will not give a higher reward than choosing to cooperate on both rounds (Glynatsi, 2020).

In the IPD, there is no dominant strategy, and the choice of strategy can in many cases be a non-trivial problem. This is because optimal play in the IPD depends on the strategy played by the opponent (Axelrod, 1984). Therefore, being able to expect the opponent's behaviour is a useful skill for any intelligent agent in the IPD.

2.1.4. The Axelrod Tournaments

Even though research on the PD had been plentiful since the 1950s, it was political scientist Robert Axelrod who popularised the game in 1980 by creating computer tournaments of the IPD as a way of studying conflict. Axelrod (1980) wanted to find the best way to play the game and asked researchers from different fields to submit strategies that were to compete in a round-robin tournament. 13 strategies were submitted that all competed against the other strategies, themselves and a strategy called Random that defects or cooperates with a 50% probability. The tournament consisted of 200 rounds repeated five times to get a balanced estimate of the scores. The strategy TitForTat (TFT) was the strategy with the highest average score and thus the winner of the tournament. This was a surprising result since TFT was the simplest strategy presented, which cooperated on the first turn and then mimicked the opponent's behaviour for the rest of the game. This meant that TFT could never beat any of its competitors but could end up even many times (Axelrod, 1984). To confirm the results, Axelrod ran a second tournament which consisted of 64 entries. This time also, TFT came out as the winner despite being surrounded by many cleverly designed and complex strategies. Axelrod analysed the strategy and concluded that the success of TFT was due to four properties, which he shaped into four suggestions on how to play the game (Axelrod, 1984):

- Don't be envious: don't strive for a larger payoff than your opponent's.
- Be nice: don't be the first to defect.
- Reciprocate both cooperation and defection: be sensitive both to retaliation and apologies.
- Be clear: be predictable so the other player can learn to trust you.

2.2. Reinforcement Learning

2.2.1. Machine Learning

ML is a subfield of (AI) consisting of techniques for making a machine construct a program from example data that can learn and improve automatically over time (Pumperla & Ferguson, 2019). AI, in turn, is a science and engineering practice in which different techniques are used to make a computer mimic human behaviour and cognition.

In traditional computer programming, clear rules and instructions are written in a program that specifies how the computer should manipulate data and what the output should be. In ML, however, the computer is fed data and an indication of the expected outcome but is not given any instructions on how to create the output. Instead, the computer learns to match and manipulate the data to the expected outcome using an algorithm (Pumperla & Ferguson, 2019). An algorithm is a step by step procedure on how to solve a problem or reach a certain goal, which often includes repetitions of actions until that goal is reached (Merriam-Webster, n.d.). Algorithms are used both by traditional programming and ML but are in ML combined with an element of automaticity. Today, ML techniques and traditional programming are often combined, leveraging the interpretability and predictability of traditional programming with the automaticity and flexibility of ML methods (Pumperla & Ferguson, 2019).

2.2.2. Reinforcement learning

RL is a class of ML- techniques initially inspired by biological learning systems where a computer agent learns from interaction with its environment. Unlike supervised learning, where the machine learns through being given explicit goals and feedback on their performance to these goals, RL is based on a reward signal provided to the agent as a response to its previous action. The reward signal the agent is given is numerical, and the agent tries to maximise this reward signal while interacting with its environment. This means that there is no prior knowledge of what constitutes good behaviour in a specific situation. Thus, the agent has to try different actions to learn which ones get the highest reward for each situation it experiences. This trial and error procedure and the fact that the agent receives the feedback signal (the reward) after its action are the two features which distinguish RL from other ML approaches (Sutton & Barto, 2018). RL is considered both a problem and a class of methods used to solve that problem.

2.2.2.1. Agent and Environment

The two primary entities in an RL problem are the *agent* and the *environment*. These two entities communicate through *actions*, *rewards*, and *observations* (Lapan, 2018). An illustration of the interaction between agent and environment in RL can be seen in Figure 1.

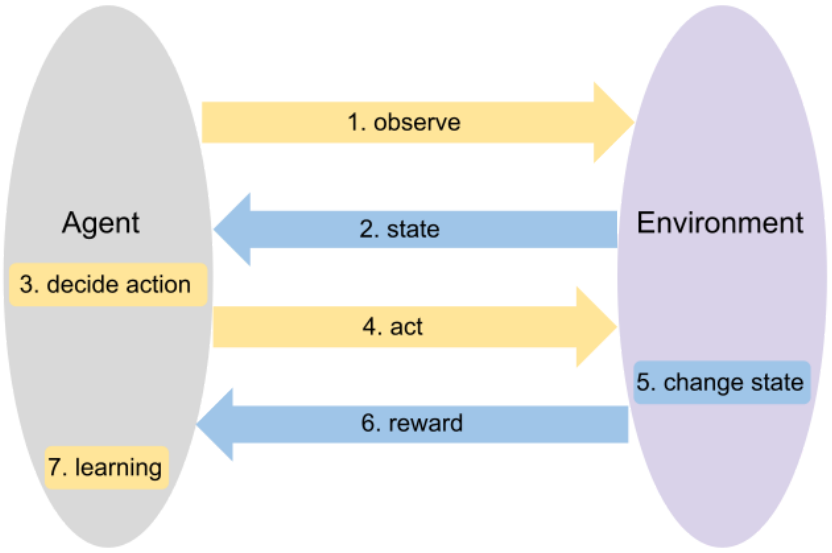
An agent is a goal-directed person or thing that interacts with its environment. The agent can sense the environment (make observations) and can execute actions that influence the environment. After each action the agent performs in the environment, it receives a reward as feedback on its previous action. This reward helps the agent learn about the environment and

how its actions influence it. Thus, the agent acts in the environment despite not knowing how that environment looks like or will respond to its actions. The agent's primary goal is to maximise the reward it receives from the environment over time (Sutton & Barto, 2018).

The environment is everything that exists outside of the agent. The agent only knows and come to know the environment through its interactions with it. Therefore, its knowledge of the environment will always be limited to what it learns through the rewards it receives, the actions it takes, and the observations it makes. Observations of the environment include any information besides the rewards that an agent gains from the environment. It is different from the *state* of the environment, which is the information of how the environment really is at that moment and which changes as the agent interacts with it. Observations are thus the information of the environment's state that is available to the agent (Lapan, 2018).

Figure 1

Illustration of the interaction between agent and the environment



Note. Adapted from "A study of Q-learning considering negative rewards", by Fuchida, T., Aung, K.T., & Sakuragi, A, 2010, *Artificial Life and Robotics, 15,* p. 352. Copyright 2010 by ISAROB.

2.2.2.2. Elements of reinforcement learning

Sutton & Barto (2018) identifies some essential subelements of RL systems beyond the agent and the environment. These are *policy*, *reward signal*, and *value function.*

A policy can be conceived of as the rulebook for the agent on how to behave at a specific time. This rulebook is updated as the agent learns, describing which action the agent should take in a certain state. Thus, it is akin to a set of stimulus-response associations in psychology (Sutton & Barto, 2018). Most policies are probabilistic, and will only tell you that a certain action is more probable to lead to a higher reward than another in a certain state. Hence, a policy is correctly described as a probability distribution over actions for every possible state (Lapan, 2018). The policy is the most critical part of an RL agent as it is the sole component that guides the agent's actions (Sutton & Barto, 2018). The agent's policy is obtained by framing an RL task as a Markov Decision Processes (MDPs), a framework from probability theory used to model sequential decision problems (Seiffertt et al., 2009). A detailed description of MDPs can be found in Appendix B.

The reward signal is the most important information received by an RL agent, as the agent's objective is to achieve the largest accumulated reward over time. The reward is a single scalar value that is received from the environment. The agent obtains the reward on a fixed schedule, which could be at each timestep or perhaps only once in its lifetime. Most commonly, a reward is received after each interaction with the environment (Lapan, 2018). The reward teaches the agent what actions are good and what actions are bad to perform in the environment. An action that leads to a less desirable state of the environment (from the agent's point of view) would receive a negative or low reward. An action that leads to a preferable state of the environment would receive a positive or high reward. The value of the reward signal is informative of how well the policy is working and is thus the primary motivation for a policy update (Sutton & Barto, 2018).

A value function informs the agent what is good for the agent long term (Sutton & Barto, 2018). A value function attaches a value to each state, which describes the total amount of reward the agent can expect to accrue in the future from the outset of that state. Thus, a value function is trying to calculate how much reward will be received from each of the consecutive states that are likely to follow the present state. Establishing the value of a state is a non-trivial but critical problem, as it is the value of the state which determines the best action to take in that state. Unlike rewards, the value of a state needs to be constantly re-evaluated and updated as the agent explores and learns from its environment. Finding a good method for state evaluation is one of the main challenges of RL.

2.2.2.3. Exploration vs. exploitation

Another challenge in RL is managing the balance between exploration and exploitation (Lapan, 2018). The agent's job is to maximise its accumulated reward over time, and so it should prefer to perform the actions that in the past led to high reward (exploitation). However, the agent doesn't know the reward of these actions and must try actions randomly

first (exploration). If the agent only exploits previous efforts, it can get stuck in a *local minimum*; a suboptimal solution. However, if the agent only explores, it is unlikely that the agent will fulfil its task. A policy that only chooses the actions with the highest expected rewards is called a *greedy* policy (Kaelbling et al., 1996). Sometimes the exploitative policy is combined with a parameter epsilon ϵ, which manages the probability of taking a random action other than the one suggested by the greedy policy. This parameter ensures that the agent doesn't get stuck following a policy that might have learned faulty action-state values prematurely in the task by providing new learning experiences. This parameter is often set high to begin with, which leads the agent to explore early in the task, and then slowly decreases so that the last part of the game is played with a greedy policy. The epsilon-greedy policy is one way of managing the exploration vs exploitation problem, but the problem is still an unsolved dilemma in RL (Sutton & Barto, 2018).

2.2.3. Q-learning

Q-learning is arguably one of the most important algorithms in RL history (Sutton & Barto, 2018). It was created by Christopher Watkins (1989) and is a model-free approach to RL, meaning it has no preconceptions about how the environment works. Q-learning works by incrementally improving its evaluations of the merits of taking a certain action at a certain state (Watkins & Dayan, 1992). To learn an optimal policy for a problem, simply having information of the value of a state, *V(s)*, is not informative of which action to take in that state. Q-learning introduces another value: Q-value, *Q(s,a)*, which is the value of taking an action *a* in state *s* (a state-action value). The algorithm calculates the expected return for each action *a* that can be taken in state *s*. The value of the state can be reduced to the highest Q-value, as *V(s)* is the highest expected return that you can get from a state (Lapan, 2018). If Q-values can be calculated for each state then deciding on an optimal policy becomes a trivial problem (Watkins & Dayan, 1992). Q-learning uses a tabular approach, meaning it stores the Q-values in look-up tables that are updated and added to as the agent visits more states. This approach is ineffective when the state-space is very large, as the look-up tables require a large memory space. Therefore, function approximators, usually neural networks, are the preferred method for more complex problems (Yannakakis & Togelius, 2018).

In Q-learning, the agent learns by using *temporal difference* (TD) learning. This means that the agent learns without a model of the environment and without waiting until the episode is over to discover the correct values for each state. Instead, the agent learns and behaves based on its own estimations of the state-values during run-time. At any time *t*, the agent performs an action *a* in state *s*, which takes it to a new state *s'* at time *t+1*. Once it has arrived in the new state *s'*, it immediately receives the reward *r* of that state. Since the action the agent took was linked to a value estimation of the new state, the agent now has access to the actual value of that state. This means that the agent can now update the Q-value of state *s* in which the agent started. This is TD learning: the estimate of what is the best action to take in the next state is used to update the estimate in the current state. This way of incrementally updating the value of a state based on estimates is called *bootstrapping* (Sutton & Barto, 2018).

Q-learning is a so-called *off-policy* method, which means that even though the learned Q-values are used to update the policy, the agent doesn't necessarily use this policy to decide which action to take (Sutton & Barto, 2018). Q-learning uses the estimate of the best action (the highest Q-value) to update the policy but can then select an action independent of the policy (Kaisers, 2012). This exploration of actions that diverge from the policy suggestions is often reduced over time so that by the end, Q-learning follows the optimal policy that it has been updating. Since Q-learning is still updating its Q-values based on the best actions in each state the algorithm is still able to converge to an optimal policy (Watkins & Dayan, 1992). The Q-value function that the agent uses to update its Q-values can be seen in (4).

$$Q(St,\ At) \leftarrow Q(St,\ At) + \alpha[Rt + 1 + \gamma maxQ(St + 1,\ a) - Q(St,\ At)] \qquad (4)$$

This shows how the Q-value of taking action A in state S at time t is equal to: the Q-value for that state plus the reward received at the next state added together with the discounted highest Q-value for the new state minus the current Q-value, with the last three terms being weighted by the learning rate α (Sutton & Barto, 2018). The learning rate decides how much the agent should update the Q-values based on what it has learnt before. With a learning rate close to 0 the Q-values are updated slowly whereas a learning rate close to 1 means the Q-values are updated significantly based on the current sample (Lapan, 2018). The discount rate γ determines how much the agent should consider future rewards. With a high discount rate, the agent considers future rewards to be important, whereas a low discount rate means the agent mainly cares about short term rewards (Sutton & Barto, 2018).

Q-learning is a flexible algorithm as it doesn't need a model of its environment and is bound to converge (find the optimal policy) if it gets enough samples of its environment. This means Q-learning can easily be applied to stochastic games, where the reward received in each state is conditioned on the actions chosen by other players (Kaisers, 2012). A formal description of Q-learning can be seen in Algorithm 1.

Algorithm 1 Q-learning

For each episode:

1. Initialise all states *S*

2. Loop for each time step of episode:

    - Choose action *a* from state *s* using policy derived from *Q*
    - Take action *a*, observe reward *r* and new state *s'*
    - Update Q-value of previous state
    - State *s'* is now the current state *s*

3. Repeat until terminal state

3. Previous studies

3.1. Q-learning in the IPD

Q-learning has historically been used in single-agent use-cases where it has been shown to perform well in zero-sum games (games with a clear winner and loser). However, research using Q-learning and other RL approaches to investigate non-zero-sum games have been relatively few (Anastassacos & Musolesi, 2018). One early study that looked at the performance of Q-learning in the IPD was done by Sandholm & Crites in 1996. In this study, a Q-learning algorithm was pitched against TFT and itself. The Q-learning algorithms varied on three dimensions: the length of memory used as state representation, how they stored Q-values, and their exploration schemes (Sandholm & Crites, 1996a). The results showed that the tabular approach could learn optimal play against TFT much quicker than the neural network approach. When pitched against itself, a tabular approach gained a higher score than the neural network one. The results also showed that exploration was a crucial factor for not being exploited by other exploring strategies. A longer time of exploration in each game against two learning agents lead to fewer cases of consecutive defections and simultaneously fewer iterations of sequential cooperations. The mean payoff per game also decreased monotonically with longer exploration schedules.

One study of the IPD used a decentralised probing method with Q-learning to gain insight into how other learning agents' strategies change as the game goes along (Anastassacos & Musolesi, 2018). This enabled the agent to take actions based on its understanding of the changes in its opponents' behaviour. The results showed that the modified Q-learning algorithm could learn to cooperate against itself, thus avoiding the otherwise strong tendency for two Q-learning agents to fall into loops of defection. As the algorithm was combined with a function approximator, training was first needed to initialise the neural network. The study

showed how modifying the environment during training increased cooperation for Q-learning in the IPD. By creating an environment with different probabilities of meeting opponents and then introducing cooperation-maintaining strategies into this environment (such as TFT), the agents' cooperation rate rose, demonstrating how norms provided by other players can influence the behaviour of the algorithm.

3.2. Memory in the IPD

Sandholm & Crites (1996a) tested the impact of different memory lengths in the IPD for Q-learning algorithms when playing against themselves. Learning against other learning agents is called Multi-Agent Reinforcement Learning (MARL) and is a more complex problem than single-agent RL. Throughout the game or task, the agents' behaviours change as they learn, as does the environment, making it difficult to determine the value of a state since the reward received for one state-action pair might not be the same the second time tried (Hernandez-Leal et al., 2017). This is often referred to as the *non-stationarity* problem of the environment in MARL (Nguyen et al., 2020). The algorithms in the Sandholm & Crites study (1996a) were based on lookup tables and varied by three different lengths: the last one, two, or three of the opponents' actions. The results showed that for memory length three, longer and more varied patterns of play developed. This led to the long memory players fairing slightly better than shorter memory ones, although only marginally. The results also showed that matches between two memory-one players gave rise to more cooperation.

Glynatsi (2020) has also shown that strategies with longer memory have an advantage over memory-one strategies in the IPD. O'Riordan et al. (2003) argue that the disadvantage of a memory-one strategy like TFT is their tendency to fall into a spiral of mutual defection. This tendency can be dissuaded by using a longer memory of the game. This has been proved by Li & Kendall (2014) that showed how well designed longer memory strategies statistically receive a higher payoff than shorter memory strategies in interactions with more arbitrary opponents. This suggests that longer memory strategies are more evolutionarily stable than shorter memory strategies in the IPD. This aligns with the results seen in Lin et al.'s (2020) experiment using RL algorithms in the IPD. They argue that the benefit of having longer state representations is related to the fact that the learning systems in the two-player IPD undergo multiple states. Thus, without a more complex understanding of the context, the reward-mappings can be oversimplified, possibly leading to suboptimal learning of the problem.

4. Method

4.1. Design

To test the performance of an RL algorithm with different memory lengths in the IPD, a round-robin tournament was run using a Q-learning algorithm against 9 default strategies. The tournament was chosen to evaluate the algorithm's performance since it provides a game theoretical framework to study the IPD (Glynatsi, 2020). The aim of the study was to provide information on how state representations affected the overall performance in the tournament and the outcome against individual opponent strategies. The tournament was run with the standard 200 turns per game but was repeated 100 times instead of the original five. This was done to ensure a good estimate of the algorithm's performance by taking into account the inherent stochasticity of the tournaments (Harper et al., 2017). The payoff matrix seen in Table 2 was used for the tournament. Each step of the game represented one PD game, where the actions available were Cooperation (C) or Defection (D). The players had no knowledge of the length of each game in the tournament. At the end of each round, the results were reset, and any memory the agents had stored were wiped. At the end of each tournament, the results were summarised, and the players were ranked based on the total average score. The run of the tournament was named Test 1.

4.2. Materials

4.2.1. Axelrod Python Library

To perform the test, the Axelrod Python Library (APL) was used. The APL is an open-source library created with the principles and goal of easy reproduction of previous research on the IPD. Other goals of the library are to create a useful tool for future research and make it simple for anyone to contribute to the library of IPD strategies. The library hosts over 230 strategies of different categories and dimensions. Round-robin tournaments and matches with different variables can be created and analysed with the library (*Axelrod Python Library*, 2015).

4.2.2. Strategies

The strategies chosen to test the algorithm are nine default strategies for the IPD identified by Jurišić et al. (2012). They differ in their initial actions, levels of cooperation, and in their retaliatory behaviours. The nine strategies were chosen for their simplicity as optimal play against some of the strategies can be easily discerned. They were also chosen based on some of their previous success in the original Axelrod tournament. The strategies were selected with the limitation of the RL algorithm in mind, all being static opponents that creates a stationary environment. These strategies consequently worked as good benchmark strategies to test the RL agent's learning capability and the benefit of different memory lengths. The nine strategies are described in Table 3.

Table 3

The nine default strategies used in the tournament.

Name	Description
Cooperator	Always cooperates
Defector	Always defects
TitForTat (TFT)	Starts by cooperating then mimics the moves of the opponent
Grudger	Cooperates until the opponent defect, then defects the rest of the game
Random	Plays cooperation or defection with a probability of 50%
TitForTwoTats (TF2T)	As TFT but defects after two successive defections
SuspiciousTitForTat	As TFT but starts with defection
TwoTitsForTat (2TFT)	As TFT but retaliates with two defections for every defection
Win-Stay Lose-Shift	Results for an action are divided into 2 groups: positive actions (T and R), and negative actions (P and S). If the previous action's result belongs to the first group, the action is repeated, and if the result belongs in the second group, the action is changed. This strategy is also called Pavlov.

4.2.3. Algorithm choice and adaptations

Q-learning was chosen for an algorithm in the study based on its status as a baseline RL algorithm, its applicability on stochastic games and its ready availability within the Axelrod Python library. Based on the results of previous studies, a tabular approach was chosen (Sandholm & Crites, 1996a). The known limitations of Q-learning in MARL motivated a choice of opponent strategies that would contribute to a stationary environment. These could thus highlight the learning ability of the algorithm for more static and predictable opponents while still giving some indication of its performance in a non-stationary setting as it also plays itself in the tournament (learning agent vs learning agent).

The Q-learning algorithm used in the experiments was adapted from the Risky Q-Learner algorithm in the APL. The Risky Q-Learner was named based on its high learning rate setting, meaning it makes quick assumptions, which can be risky as it can lead to premature convergence and a suboptimal solution to the problem (Yannakakis & Togelius, 2018). The Risky Q-Learner has a state representation containing the last N moves of the opponent as an adjustable parameter together with the total cooperation count over the opponent's history. An example state representation would be 'CCDDD:34', informing the agent of the opponent's last five actions and that the opponent had cooperated 34 times in the game so far. A pre-run of the tournament showed that this state representation was suboptimal since the additive element of the cooperation count made the agent experience every state as new given

the opponent played more cooperations. An evaluation of the code revealed that the Q-function of the Risky Q-Learner was not corresponding to the original description of the algorithm. Changes were made to the algorithm so that the state representation only consisted of the *N* number of states of the opponent's last actions. This state representation was in line with previous research of RL in the IPD (Lin et al., 2020; Sandholm & Crites, 1996b). The Q-function was adjusted to correspond to the original version. The option of annealing epsilon value was also added to the algorithm, ensuring that the agent could start with a certain probability of choosing random actions that then diminished over time. The new version of the algorithm was named Regular Q-Learner.

4.2.4. Algorithm parameter settings

To test the implications of different memory lengths for an RL algorithm in the IPD, the Q-learning algorithm was run with three different memory lengths: 1, 5, and 10 (*Mem1*, *Mem5* and *Mem10*). This corresponds to the state representation of the algorithm as the last *N* actions that the opponent played. Mem1 and Mem5 were used to represent the agent's short and mid-length memory capacity and were motivated by results in previous studies (Lin et al., 2020; Sandholm & Crites, 1996b). Mem10 was chosen experimentally to represent a longer memory as test runs showed a higher cooperation rate and total score than Mem1 or Mem5.

Hyperparameters alpha (learning rate) and gamma (discount rate) were held constant at $\alpha = 0.5$ and $\gamma = 0.95$. The learning rate of alpha 0.5 was chosen experimentally as a mix between learning rate 0.2 as seen in Sandholm & Crites (1996b) work and a learning rate of 1 that slowly decreases as seen in Lin et al.'s (2020) work. A high discount rate was chosen as cooperation can only occur in the IPD if the future is considered important enough for the present action selection (Axelrod, 1984). This has also been proved experimentally (Sandholm & Crites, 1996b). Exploration was managed with an epsilon greedy policy with an epsilon value of 0.1 that slowly decayed so that the last third of the game was played with a deterministic policy.

4.3. Evaluation Metrics

4.3.1. Choice and motivation

The performance of the algorithm in the tournament was evaluated with four metrics: total average score per tournament, rank, normalised payoffs per opponent per turn and cooperation rate. As there exists a variety of metrics to analyse strategy behaviour and results in the IPD, these metrics were chosen as they could easily depict the overall performance of the agent in the tournament as well as how well the agent was fairing against individual strategies. The cooperation rate was added as the IPD is often used to study the evolution of cooperation, and as such, it is considered an important metric to report (Glynatsi, 2020). As the focus was to find general patterns that could be used as inspiration for a new model, an analysis of statistical significance in the results was omitted. However, to better visualise the difference in results, 10 percent error bar plots were created.

4.3.2. Total average score per tournament & rank

The total average score determines the winner of the tournament, which corresponds to the rank of the players. Thus, measuring the total average score and rank is a good indication of how well the algorithm performs compared to other strategies.

4.3.3. Normalised payoffs per opponent and turn

This metric gives a good indication of which strategy the algorithm chooses to play against each opponent. Based on knowledge of the opponent strategy, the normalised payoffs per opponent and turn can tell whether the agent has learnt mainly to cooperate or defect against the opponent. The normalised payoffs per opponent and turn are bounded by the payoff matrix and will therefore be a number between 0 and 5 (Sucker's Payoff and Temptation).

4.3.4. Cooperation rate

Cooperation rate is a morality metric created by Singer-Clark (2014), which describes how many turns the agent cooperated over the whole tournament out of all repetitions of the tournament. The cooperation rate can be informative of the agents' performance in relation to its total average score and normalised payoffs per opponent and turn. Cooperation rate values range from 1.0 (always cooperates) to 0.0 (always defects).

4.4. Test 1

4.4.1. Results

The results of Test 1 can be seen in Table 4. Mem10 obtains the highest score of the three algorithms at 4021, with Mem5 coming second at 4009 and Mem1 doing the worst at 3815. Despite this, it is Mem10 that has the lowest rankings of the three algorithms, which indicates high stochasticity in the tournament. Mem10 performs best against the strategies from the TFT-family but has the lowest score for both Defector and Grudger, which are exploiting strategies (plays majority D). It is instead Mem1 that performs best against the Defector, and Mem5 that performs best against Grudger. Mem1 also performs best of the three against Win-Stay Lose-Shift. The results show that Mem1 has the lowest cooperation rate at 0.48, with Mem5 and Mem10 having higher cooperation rates at 0.67 and 0.71 respectively. The difference in normalised payoffs per opponent and turn for the three different memory lengths is minimal for some opponents and quite considerable for others, as can be seen in Figure 2. Against 2TFT for example, Mem10 scores 0.82 more than Mem1, but for Grudger the difference in payoff for the two is only 0.03.

Table 4

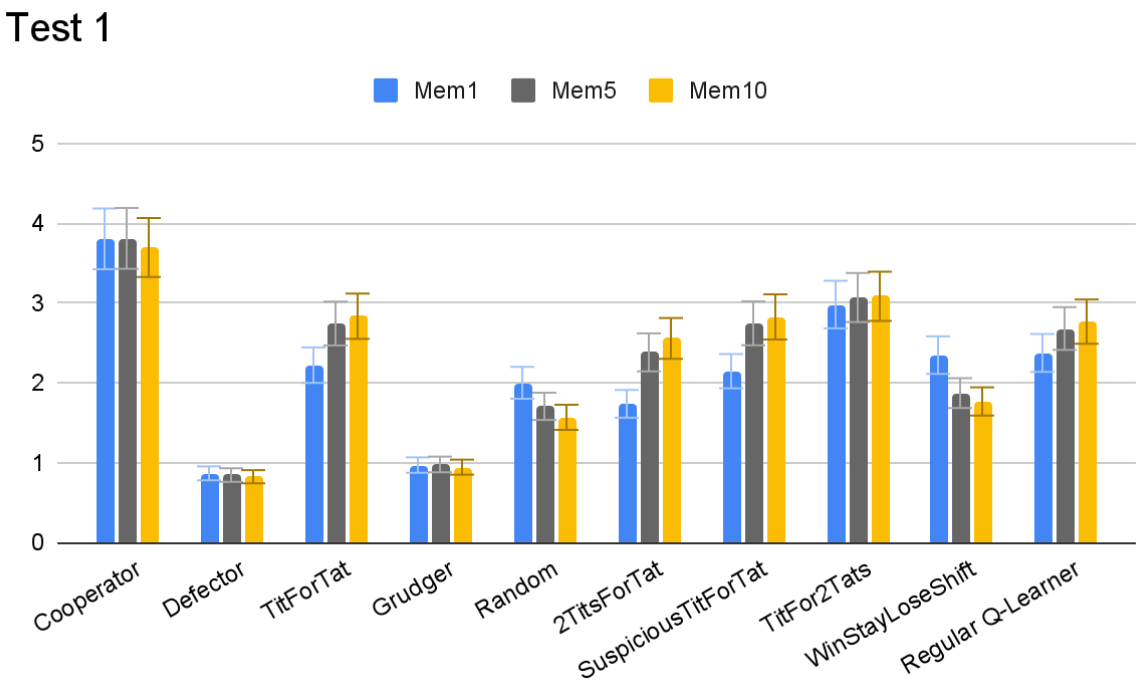
Results for Test 1

	Mem1	Mem5	Mem10
Average score per tournament	3815	4009	4021
Rank	7/10	7/10	8/10
Cooperation rate	0.48	0.67	0.71
Mean payoff per opponent and turn			
Cooperator	3.80	**3.81**	3.70
Defector	**0.87**	0.85	0.83
TitForTat	2.22	2.74	**2.83**
Grudger	0.97	**0.98**	0.95
Random	**2.00**	1.71	1.57
2TitsForTat	1.74	2.38	**2.56**
SuspiciousTitForTat	2.15	2.75	**2.83**
TitFor2Tats	2.98	3.07	**3.08**
Win-Stay Lose-Shift	**2.35**	1.87	1.77
Regular Q-Learner	2.38	2.68	**2.77**

Note. The bold face indicates that the player in column *i* has the highest mean payoff per opponent and turn among the players.

Figure 2

Differences in mean payoff per opponent and turn for the three memory lengths



4.4.2. Interpretation

Since the TFT-strategies are cooperation-maintaining strategies (they will punish you for defection and reward you for cooperation), it is not surprising that Mem10 performs best against them as it has the ability to represent more complex state-reward mappings, leading to a higher cooperation rate. Likewise, the lower cooperation rating of Mem1 seems to indicate that the agent has learnt better (or faster) how not to be taken advantage of by exploiting strategies (like Defector and Win-Stay Lose-Shift) and how to better exploit naively cooperating strategies (like Cooperator). Since Mem10 has more state-representations to evaluate, it takes the algorithm longer to decide on good Q-values for each state. This delay in learning means that Mem10 learns optimal play against exploiting strategies much slower than Mem1 and Mem5. This seems to be a disadvantage of a longer memory for exploiting opponents but an advantage against opponents with whom it is better to cooperate. This raises the question of whether a longer memory means higher cooperation in general or whether this is simply because 200 turns per game were not sufficient for the algorithm to find optimal play against certain opponents.

4.4.3. Extension of Test 1

To determine whether a prolonged learning time would give better results, the tournament was rerun but with 2000 turns per game and 50 repetitions. The results can be seen in Table 5. Again, it is Mem10 that obtains the highest score at 42234 followed by Mem5 at 41321 and

Mem1 at 39795. The trends seen in the first tests are mainly still present. However, the ranking has now changed so that both Mem5 and Mem10 place 6th in the tournament. This means that Mem10 has advanced two ranks compared to Test 1. The cooperation rate of all algorithms have decreased given more time to learn and it is still Mem1 that performs best against exploiting strategies, and Mem10 that performs best against the TFT-family. The results indicate that a longer run time has improved the algorithms' ability to avoid being exploited and to exploit naively cooperating strategies, while mainly maintaining the scores of Test 1 for more complex opponents (TFT-family). The differences in mean payoff per opponent and turn is visualised in Figure 3.

Table 5

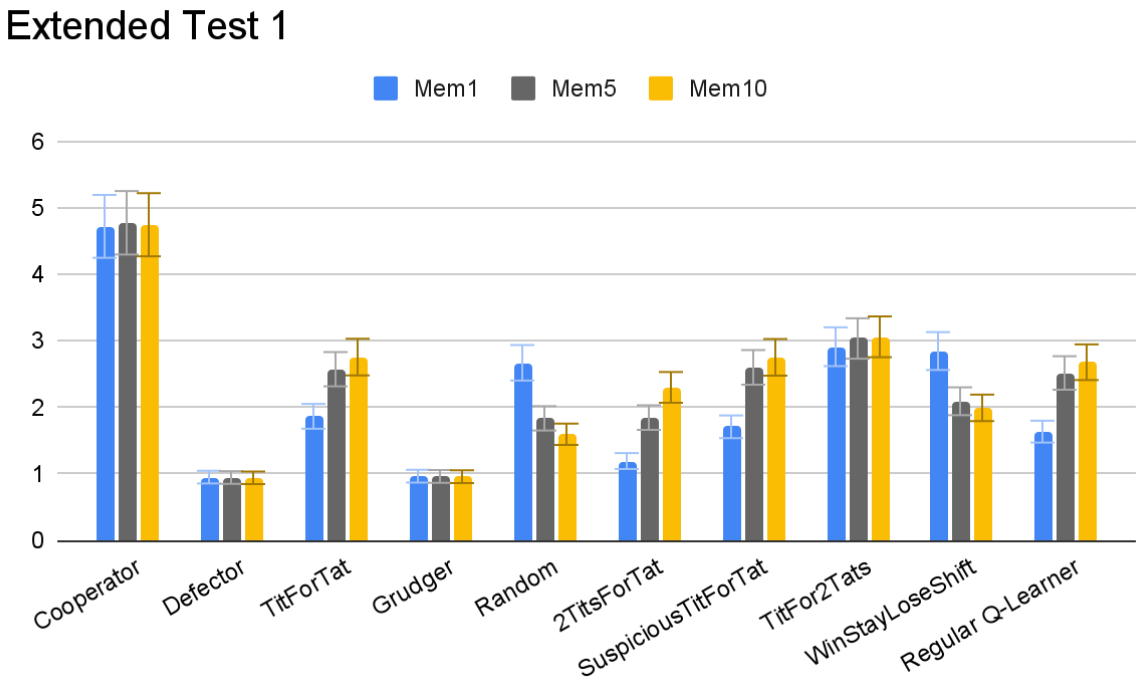
Results for the extended Test 1

	Mem1	Mem5	Mem10
Average score per tournament	39 795	41 321	42234
Rank	7/10	6/10	6/10
Cooperation rate	0.24	0.51	0.59
Mean payoff per opponent and turn			
Cooperator	**4.77**	4.76	4.75
Defector	**0.95**	0.95	0.94
TitForTat	1.87	2.58	**2.76**
Grudger	**0.97**	0.96	0.96
Random	**2.67**	1.84	1.60
2TitsForTat	1.20	1.85	**2.30**
SuspiciousTitForTat	1.71	2.60	**2.75**
TitFor2Tats	2.91	3.04	**3.06**
Win-Stay Lose-Shift	**2.85**	2.09	2.00
Regular Q-Learner	1.64	2.52	**2.68**

Note. The bold face indicates that the player in column *i* has the highest mean payoff per opponent and turn among the players.

Figure 3

Differences in mean payoff per opponent and turn for the three memory lengths



5. Mixed Memory Q-Learner

5.1. Algorithm description

5.1.1. Concept

Given the results of Test 1, a new strategy was proposed: Mixed Memory Q-Learner (MMQL). This strategy leverages the two insights gained from Test 1: that different memory lengths (state-representations) are preferable depending on the opponent's strategy. And secondly, that a longer learning time could lead to better results in the tournament.

The first distinguishing feature of the MMQL is the ability to recognise its opponents between tournament repetitions, thus starting its learning where it ended the last time it played the same opponent. This means that the agent could participate in a standard Axelrod tournament with 200 turns over five repetitions and still learn over 1000 turns. Axelrod (1984) claims that the ability to discriminate between individuals is an essential skill as this enables agents to treat various individuals differently. In combination with memory, this skill can be used to punish defectors from the start in the subsequent interaction while rewarding cooperators with immediate trust and cooperation. The second distinguishing feature of the MMQL is the ability to switch memory length during play. This means the algorithm

performs an evaluation during run-time of which memory length gives the highest payoff and then switches if another memory length is better than the current one used. This enables the strategy to adapt to its opponents and to be self-reflective about its performance.

5.1.2. Architecture

MMQL is a meta-strategy that uses a team of three different Q-learning algorithms simultaneously to find the best policy. The Q-learning algorithms are the Regular Q-Learner with three different memory lengths as analysed in Test 1 (Mem1, Mem5 and Mem10). The algorithm works by letting all three players play the opponent at the same time. During play, there is always one player which has the status as the current policy. This means that the action played at each turn of the game, and the reward signal received, is based on this player's policy. This approach, where each agent has its own individual policy but optimises it from a joint reward signal, is called *concurrent learning* (Gupta et al., 2017). If MMQL has not met an opponent before, the current policy player is chosen randomly. The first 40 turns of each game are used to collect knowledge and initialise Q-values (similar to the concept of experience replay buffer, see for example Lapan (2018)). At turn 40 and up till turn 180, an evaluation will take place at every 20th step (turn). This evaluation is done by looking at the last 20 steps played by each of the three players and measuring the average reward received over those 20 steps. If one player has a higher average reward than the current policy, this player is chosen as the new current policy (similar to the concept of training two policies at once and then switching if one policy outperforms the other by a certain threshold in Silver et al. (2017)). If two policies have the highest average reward, the current policy is randomly selected from the two. This is true unless the current policy is within that list, in which case the current policy is chosen to continue.

To balance the variance in Q-values created by switching policies, the probability of changing the current policy gets smaller as the game goes on. At the end of the game, the Q-values of all the players are stored away and remembered based on the opponent's name. The next time MMQL meets the opponent, the Q-values from the previous match are loaded, and the player that was the current policy at the end of the last game begins in that position again. MMQL is similar to the model used by Hoang et al. (2018), where a multi-agent system is used to generate best-response strategies to particular opponent behaviours, that are then selected from by a high-level planner and adjusted as the opponent behaviour changes. A formulation of MMQL can be seen in Algorithm 2.

Algorithm 2 Mixed Memory Q-learning

For every step in game **do:**

1. If beginning of game:

Check if has met opponent before:

if True:

- load previous Q-values
- set current policy to previous best policy

if False:

- initialise all states S
- randomly select from the 3 policies

2. For all 3 players **do:**

- play opponents

3. Check if at evaluation point:

if True:

- perform policy evaluation
- update current policy

4. Check if end of game:

if True:

- unload Q-values

5. Play action chosen by current policy

6. Update Q-values based on reward signal

7. Repeat until game ends

5.1.3. Limitations

To account for the exploration in the algorithms, the evaluation window was set to 20 steps to ensure that there was enough possibility for variance in all players' results to make a just assessment. Due to more random actions being taken early in the game, one player might by chance get a higher reward than another player. However, as that player is then chosen as the current policy, it will become clear whether its policy has benefits, and if not, it will be sorted out at the next evaluation point. As the exploration rate and the probability of switching policy get smaller throughout the game, the evaluation should become more fair later in each game. This instability in the evaluations at the beginning of each game is deemed to be a necessary tradeoff to ensure that MMQL does not get stuck in a local minimum and continues to explore new ways of acting.

Another potential problem with MMQL is that the players all update their Q-values based on the state they find themselves in, which is a direct result of the actions of the current policy. If a player is not the current policy, it might form a conception of the opponent's behaviour that is not based on its own actions but of the actions of other players. If the players do not get a

chance to play and explore as the current policy, their Q-values may mainly resemble other players', thus creating a blending effect. However, the high exploration rate at the beginning of each game, and the difference in state-representations for the three players, is expected to make sure that their strategies will still be distinguishable from one another.

5.1.4. Performance prediction

The idea of MMQL is to combine the best play of each memory length to get a higher long-term return. MMQL is expected to sense which memory length to use as state-representation for each opponent and to prioritise the use of that memory length as its current policy. By doing so, it should be able to both avoid being exploited by exploitative opponents and cooperate with cooperative opponents. By continuous learning over tournament repetitions, the agent should be able to settle on a good policy for each opponent. This will enable the agent to take part in the standard Axelrod tournament without losing all its memory between rounds, thus performing better overall.

5.2. Test 2

5.2.1. Design

To test the performance of MMQL, a standard Axelrod tournament was run with 200 turns per game over five repetitions. The MMQL played the same nine default strategies used in Test 1. A tournament was also run for the three Q-learning algorithms from Test 1 but with added ability to recognise and continue to learn over the whole tournament (same as MMQL). This was done to discern what effect MMQL's policy switching ability had on the result, and whether it had any benefits to using the same memory length throughout.

5.2.2. Parameter settings

The parameters for the three Q-learners used individually and in the MMQL were the same as in Test 1. Learning and discount rates were held constant at $\alpha = 0.5$ and $\gamma = 0.95$. The players used an epsilon greedy policy starting at 0.1 which slowly decreased over the game. The exploration parameter was reset at the beginning of each game.

5.2.3. Additional evaluation methods

To evaluate the performance of MMQL, two additional methods were used. One method checked for convergence of state-values over the games, thus showing if an extended learning time had enabled the players to settle on a good policy for each opponent. This was done by measuring the difference in state-values at every five steps of each game and calculating the root mean squared error (RMSE) between the current and last evaluation point. The RMSE is a good evaluation metric for model performance when the error distribution is assumed to be Gaussian (Chai & Draxler, 2014). These results were then plotted to see how the values changed over the tournament and whether the state-value updates were small enough to indicate that the algorithms had settled on a policy at the end of the tournament.

The second evaluation method measured how much of each game (in percentage) each player played as the current policy. This was done to indicate whether the algorithm had leveraged the findings of Test 1, thus choosing the memory length that had performed best against each opponent as its current policy for the majority of the tournament. These measurements were then plotted to give a visual indication of the results.

6. Results

The results from Test 2 can be seen in Figure 4 and Table 6. It is still Mem 10 that gets the highest average score per tournament at 3911.2, with MMQL coming second at 3826, Mem5 third at 3660 and Mem1 last at 3521. Despite the predictions, none of the algorithms placed higher in the tournament than rank 7. Compared to the extended Test 1, the cooperation rate has decreased significantly for all algorithms. Mem10 has the highest cooperation rate at 0.4, followed by MMQL at 0.23, Mem5 at 0.18 and Mem1 at 0.08. Regarding mean payoff per opponent and turn, MMQL gets the highest score against Defector at 0.975, with Mem5 coming a close second at 0.970. It also has the highest score for Grudger at 1.01, same as Mem1. Besides this, the highest mean payoff per opponent and turn follows the same pattern as the results in the extended Test 1. This means that it is still Mem1 that performs best against Cooperator, Random, and Win-Stay Lose-Shift and Mem10 that performs best against the TFT-family and itself. However, MMQL comes in as second best for Cooperator, TFT, SuspiciousTFT, TF2T, Win-Stay Shift-Lose and against itself. The differences in mean payoff per opponent and turn are visualised in Figure 5.

Figure 4

The tournament rankings of each strategy with the winner, Tit For Tat, to the left. Y-axis shows mean payoff per opponent and turn.

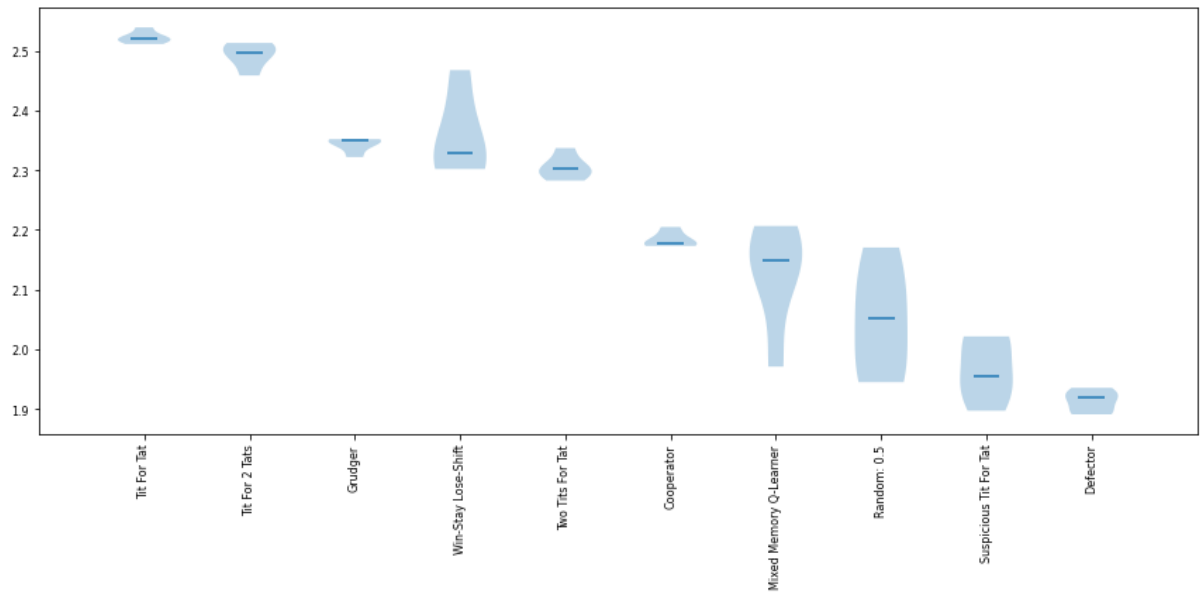


Table 6

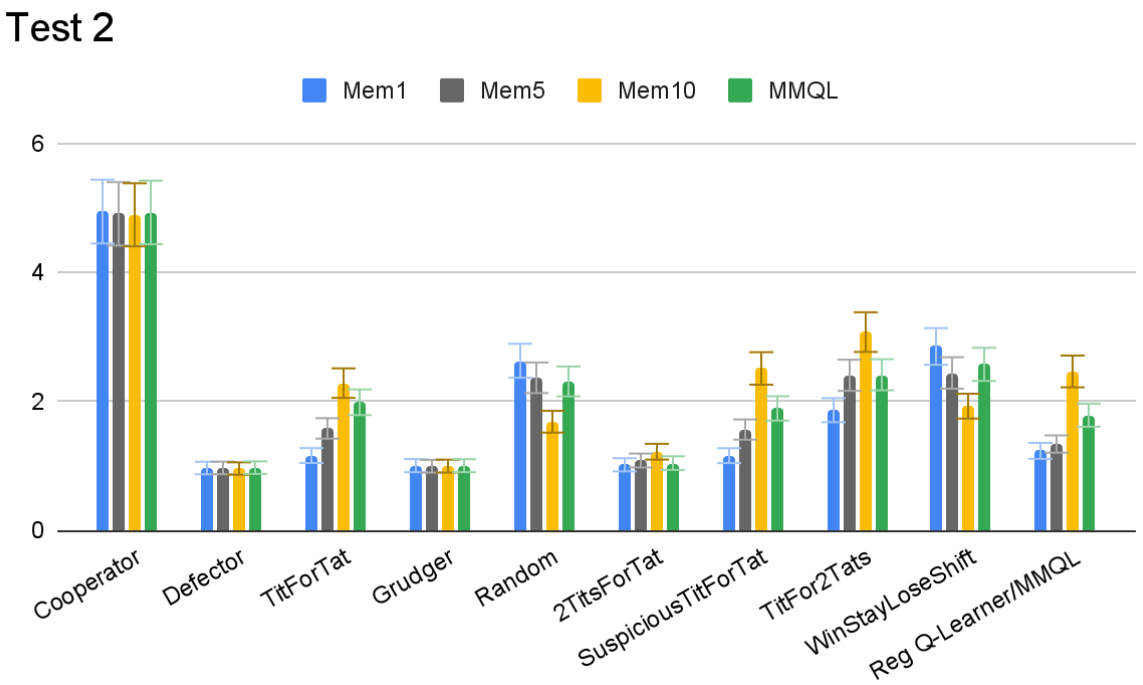
Results for Test 2

	Mem1	Mem5	Mem10	MMQL
Average score per tournament	3521	3660	3911	3826
Rank	7/10	7/10	7/10	7/10
Cooperation rate	0.08	0.18	0.40	0.23
Mean payoff per opponent and turn				
Cooperator	**4.94**	4.91	4.89	4.93
Defector	0.97	0.97	0.96	**0.98**
TitForTat	1.16	1.58	**2.28**	1.99
Grudger	**1.01**	0.97	1.00	**1.01**
Random	**2.63**	2.37	1.69	2.31
2TitsForTat	1.02	1.08	**1.22**	1.05
SuspiciousTitForTat	1.16	1.57	**2.51**	1.89
TitFor2Tats	1.87	2.41	**3.08**	2.41
Win-Stay Lose-Shift	**2.85**	2.42	1.93	2.58
Regular Q-Learner/ MMQL	1.24	1.34	**2.47**	1.79

Note. The bold face indicates that the player in column *i* has the highest mean payoff per opponent and turn among the players.

Figure 5

Differences in mean payoff per opponent per turn



When looking at which player in MMQL is used most as the current policy for each opponent, we can see that MMQL does leverage some of the results from Test 1. According to those findings, Mem1 should give the best results for Cooperator, Defector, Grudger, Random and Win-Stay Shift-Lose. Mem1 is used as the current policy most out of the three players for Defector (70% of the tournament), Grudger (48% of the tournament), Win-Stay Shift-Lose (60% of the tournament) but not for Cooperator (32% of the tournament), where instead Mem5 is used the most (36% of the tournament). Mem5 is also used the most for Random (42% of the tournament) compared to Mem1 (32% of the tournament).

If using the results from Test 1 as a guide, MMQL should play Mem10 for the majority of the game against the TFT-family and against itself. However, Mem10 is used most times only against TF2T (44% of the tournament). Instead, Mem5 is used most against TFT (48% of the tournament), 2TFT (50% of the tournament), and MMQL (42% of the tournament). Mem1 is used most against SuspiciousTFT (50% of the tournament). Calculating the distribution of use of each player as the current policy over the whole tournament for all strategies, we can see that Mem1 is used most (36,6% of the time), closely followed by Mem5 (34,8%) and Mem10 is used the least (28,6%). Table 7 shows how many percent each player played as the current policy against each opponent over the five tournaments.

Table 7

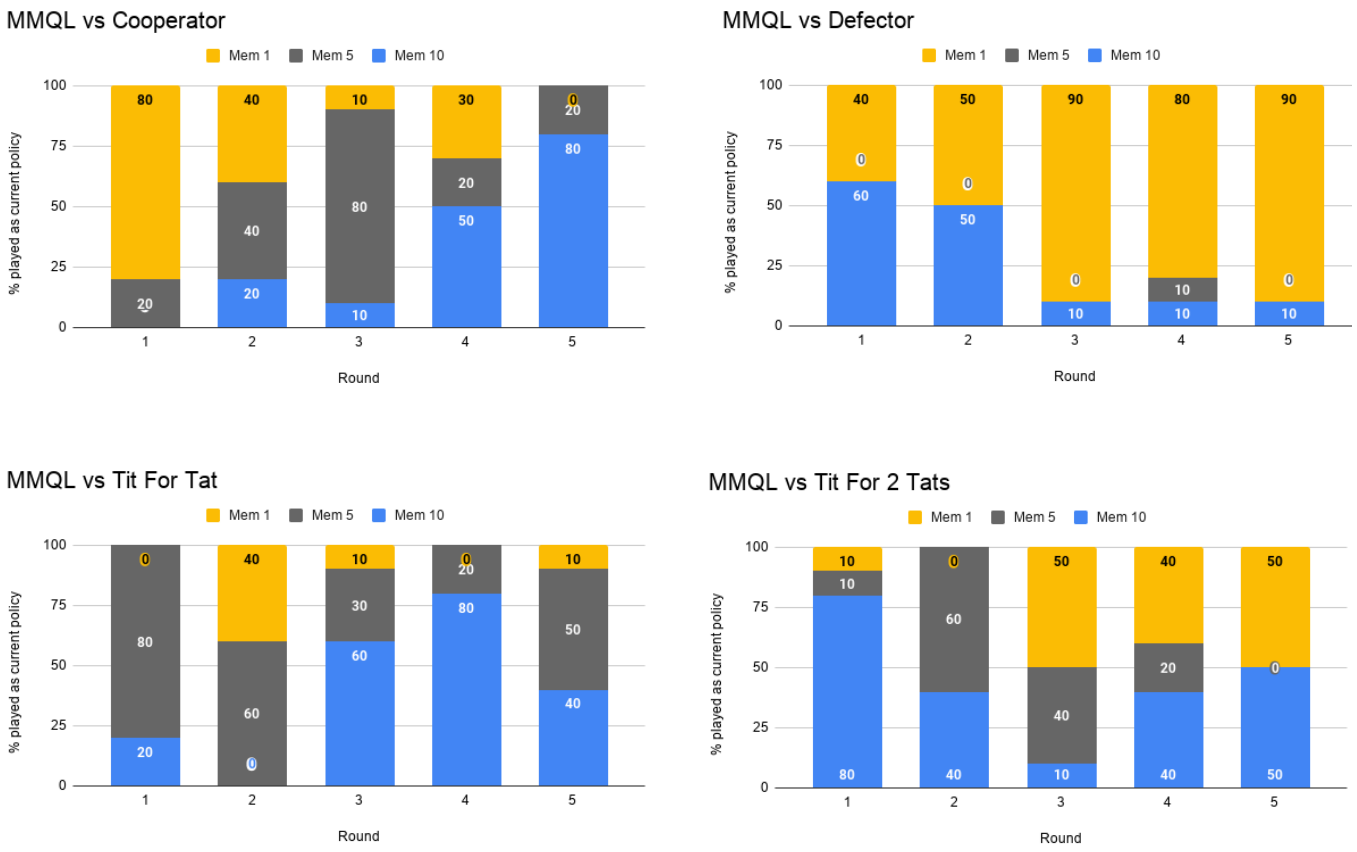
Percentage of the game played with each player of MMQL against each opponent

	Players of MMQL		
	Mem1	Mem5	Mem10
Opponents			
Cooperator	32%	**36%**	32%
Defector	**70%**	28%	2%
TitForTat	12%	**48%**	40%
Grudger	**48%**	16%	36%
Random	32%	**42%**	26%
TwoTitsForTat	10%	**50%**	40%
SuspiciousTitForTat	**50%**	40%	10%
TitFor2Tats	30%	26%	**44%**
Win-Stay Lose-Shift	**60%**	20%	20%
MMQL	22%	**42%**	36%

Note. The bold face indicates that the player in column *i* has been used for most of the time against the opponent.

Figure 6

Percentage played with each player for a) Cooperator, b) Defector, c) Tit For Tat, d) Tit For 2 Tats



The convergence plots show that the state-values for both MMQL and the Regular Q-Learners get disturbed by the off-and-on loading of Q-values between rounds. This can be discerned from the four abrupt spikes of change in RSME seen at the end of each game. This effect is more apparent for MMQL than the Regular Q-Learners and depends on the opponent. In general, Mem1 has the highest variance of RSME throughout the games, followed by Mem5 and, lastly, Mem10. This is true for both MMQL and the Regular Q-Learners. The plots show how all three players in MMQL converge on a policy for Cooperator, Defector and Grudger quickly in the game, while the RSME fluctuation for the other opponents is more volatile. This is especially true when playing against itself. In general, it appears that introducing the ability of policy switching creates more instability in the state-value updates than simply using the same state representation throughout the game. However, this does not appear to be true for SuspiciousTFT and Grudger, where MMQL instead seems to find a more stable policy than the Regular Q-Learners (see Appendix C). Figures 7 & 8 shows the convergence plots for all players for opponents split into two groups: TFT-family and Cooperator & MMQL.

Figure 7

Convergence plots for the three players in MMQL (left column) and the three Regular Q-Learners (right column) against the TFT-family

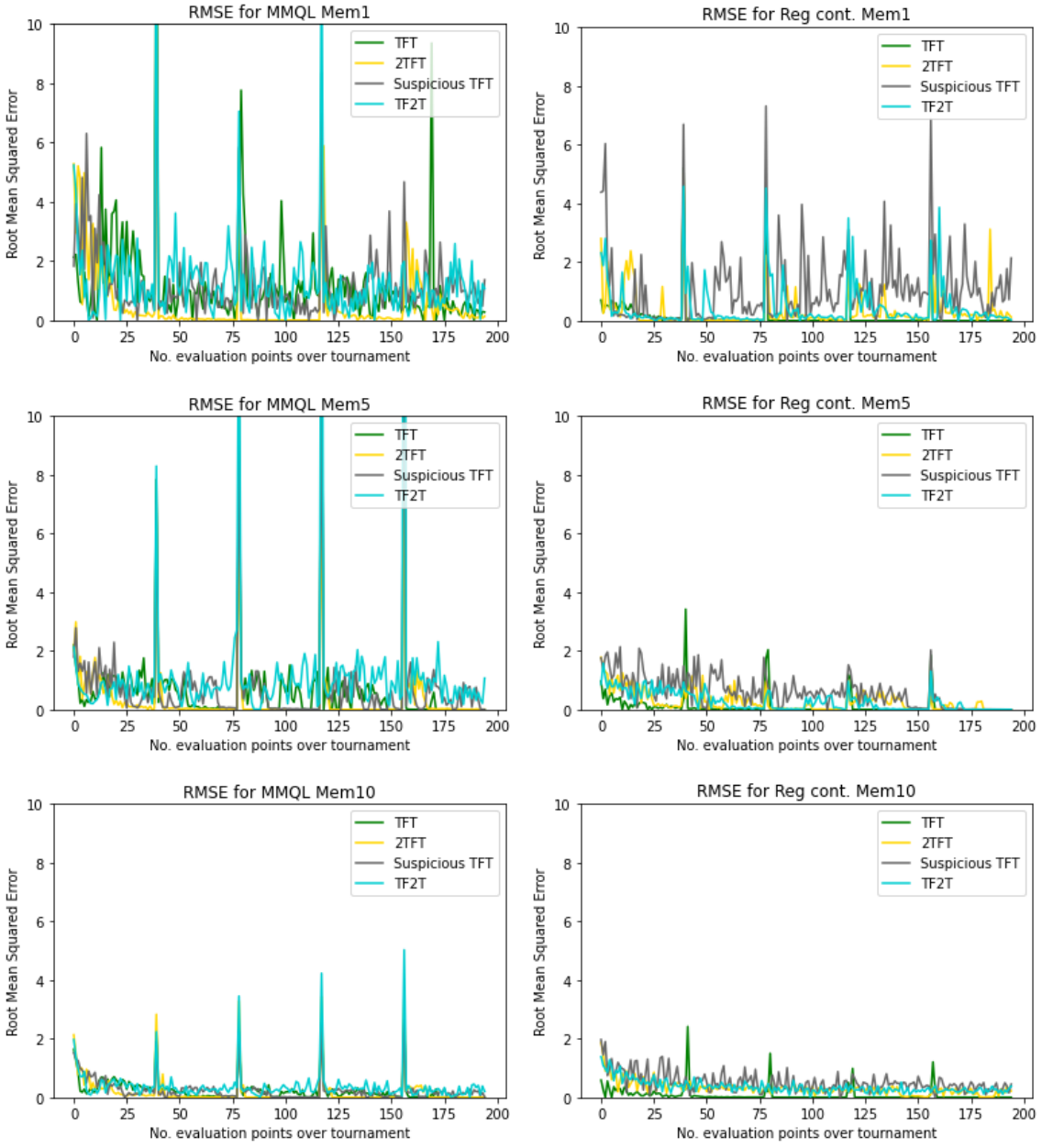
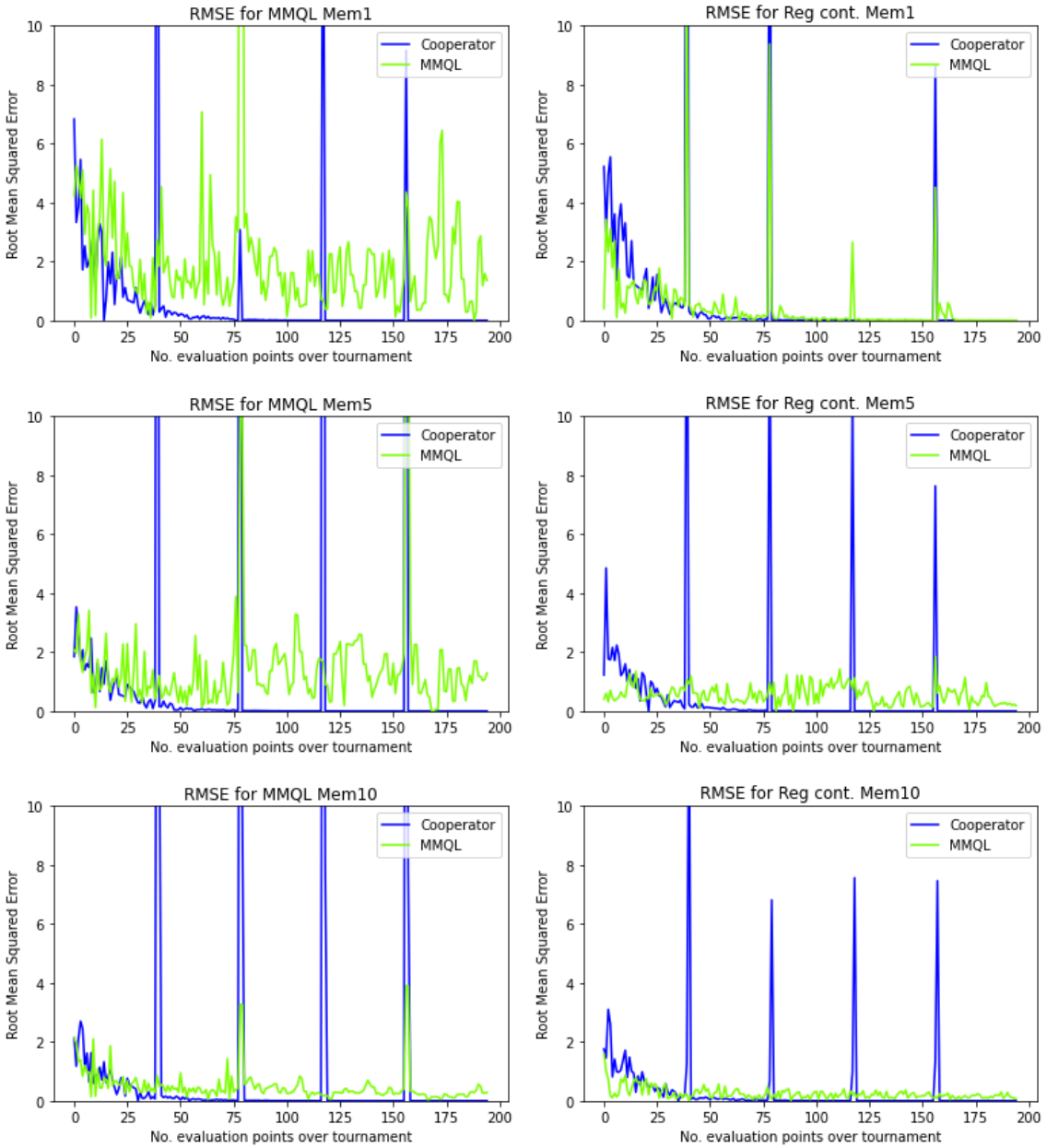


Figure 8

Convergence plots for the three players in MMQL (left column) and the three Regular Q-Learners (right column) against MMQL & Cooperator



Note. The Regular Q-Learners play against themselves, not MMQL. Therefore, the label "MMQL" in the right column is supposed to be "Regular Q-Learner".

7. Discussion

7.1. Results analysis

The result of Test 2 shows that MMQL does leverage some of the effects seen in Test 1. For example, it chooses to use the previously established best memory length for most of the time against certain opponents (for instance, against Defector and Win-Stay Shift-Lose). It also appears that a longer learning time has made the agent better at playing certain opponents (like Cooperator, Grudger & Defector). However, MMQL does not perform better in the tournament than Mem10 with continuous learning ability, and it did not place higher in the tournament as was predicted. Instead, TFT won, same as in the original Axelrod tournament (Axelrod, 1984). The low ranking of MMQL in the tournament seems to be because it did not learn to cooperate with cooperative agents. The ability to recognise its opponent and have a longer learning period only appears to have made the agent better at exploiting naively cooperating agents and avoiding being exploited itself. This is in line with the work done by Leibo et al. (2017) that showed that either cooperation or defection policies might be easier to learn depending on the problem itself. The PD being the most severe social dilemma, it is perhaps not surprising that the agent was quicker to develop a defection policy.

Why didn't MMQL develop more cooperation? One possible answer is that MMQL didn't use Mem10 most of the time against cooperation-maintaining strategies (TFT family), as Mem10 had the highest cooperation rate out of the three players. But it could also be a result of the value function itself. By primarily focusing on maximising reward, it is difficult to learn to cooperate in the long run. The algorithm will always be biased towards the short term high reward of Temptation rather than the long term reward of continuous cooperation. It therefore takes time to learn a cooperating policy. It is possible that adjusting the learning and exploration rate might impact what policy the agent learns and how fast it learns it, potentially leading to more cooperation. However, the bias toward maximum reward will still assert a solid pull to defect for the agent. Another option is to change the payoff matrix, making Sucker's payoff and Punishment into zero or negative values. This could lead the agents to cooperate more, as Punishment would no longer be a positive reward, potentially decreasing defection.

Still, a maximising agent can develop cooperation. As described in chapter 3.1, a Q-learning agent can be taught to cooperate if first trained in an environment with cooperation-maintaining strategies (like TFT). This initialises the state-values to make cooperation a more desirable choice. But since MMQL starts with state-values as zero, it is easily biased towards the higher reward of Temptation, and it seems like only Mem10 can fully represent cooperation as a good choice. This follows Li & Kendall's (2014) insight that longer memory strategies are more evolutionarily stable in the IPD. However, since Mem10 takes longer to settle on an optimal policy (since it has to visit more states several times to find the correct state-values), MMQL is biased toward using Mem5 and Mem1 since they

occasionally get a high reward by exploiting the good nature of its opponent. It's possible that a longer run time still might lead MMQL to select Mem10 more as its current policy when playing cooperation-maintaining opponents (like the TFT-family). Yet, there is also a possibility that a longer run time will lower the cooperation rate of all players further and the solution should therefore focus on making Mem10 perform better sooner in the game. This could perhaps be done by setting a higher learning rate for Mem10 than the other players or simply changing the approach and using a function approximator instead.

To incite cooperation in the MMQL, the agent might need additional capabilities, make use of a neural network or the problem itself could be reframed. The PD could, for example, be constructed as an *n*-person game where agents in a society can punish non-cooperating agents (Axelrod, 1997). This is done by adding further actions and payoffs so that players that see others defect can choose to punish those players (payoff -9), but at a certain cost for themselves (payoff -2). This could significantly affect the value function, as the player would have to adhere to social norms. Reframing the problem could be a useful step to get a self-interested agent to cooperate more. Still, if the objective is to play optimally in the IPD, it might be best to consider how well MMQL follows Axelrod's suggestions on how to succeed in the game. When doing so, it is easy to see that MMQL is the opposite of the suggested best player: it is envious (it strives for a larger payoff than its opponents), it is not nice (it might be the first to defect), it does not reciprocate both cooperation and defection (it is biased towards defection), and it is not clear (it changes its behaviour throughout the game).

RL might be a valuable tool to train other strategies in the IPD (see Harper et al., 2017), but it seems unlikely that a simple RL algorithm like Q-learning will ever place high in the standard tournament. However, the results show that having the ability to represent the problem with different state-representations, in this case, different memory lengths, could be beneficial when meeting heterogenous opponents. This is likely to remain true, even if the problem is reframed or if other capabilities are added to the algorithm.

7.2. MMQL analysis

One problem with MMQL is the instability in the state-value changes, making it more difficult for the players to converge on a good policy. Adjusting two things in the architecture of the algorithm might solve this. The first fix is to have the three players play the opponent independently, meaning their Q-value updates would be based on their own actions and not those of the current policy. This might lead to each player developing a more distinct strategy and is likely to ease convergence. The second fix is to refine the policy evaluation of MMQL by adding a threshold for the difference in the average reward necessary for the current policy to switch. As of now, the policy switches, even if the difference in average reward is 0.1, which might be too small a difference to justify the instability that comes with policy-switching. Since more random actions are taken at the beginning of each game, the threshold could be set at a lower value at first, enabling more policy switching at the beginning of the game, and could then increase later in the game to ensure that only policies

that play considerably better are chosen as the current policy. This parameter could also be set to continue over the different rounds and not be reset for each game, creating more stability. The two solutions are expected to work well together, as the independent policies will be more representative of each player's actual strategy, thus being a better model of the original idea of MMQL.

Even if MMQL does not perform well in the IPD, having the ability to create different strategies and sense which strategy to use, depending on the environment, is bound to be an important function for future AIs in social dilemmas. Thus, the idea behind MMQL can be scaled to fit other problems and situations. The concept of MMQL can be used to create a Multi-Objective Reinforcement Learning (MORL) algorithm. MORL is based on the idea that many tasks have more than one goal or objective and that explicitly accounting for that when implementing the algorithm is better than simply representing the objective as a scalar value (Horie et al., 2019). By having more value functions that all have different goals, specific team players could be instructed to optimise for a cooperative opponent, and others could be instructed to optimise for an exploitative opponent. As shown in this study, different state representations are preferable depending on the opponent's strategy. Having a team of agents with different objectives that can try out and change between different state representations to maximise those objectives might be a successful approach to MARL for social dilemmas.

7.3. Limitations

It is worth mentioning that MMQL's ability to learn over tournament repetitions might in some respects be seen as cheating. This is because none of the other players can recognise their opponent or remember their last interaction. However, this ability was added because of the perception that RL algorithms have an unfair disadvantage in the standard tournament. The nine default strategies used in this study all have a clear strategy to adhere to at every point of the game. Other non-static strategies in the IPD have often had training before participation. In contrast, all RL algorithms in the APL have to learn their strategy from scratch at the beginning of each game. This could be seen as unfair, and the idea was thus to see how an RL algorithm would perform in the tournament if it could keep its memory between games.

7.4. Future research

The ability to change and try out state representations is akin to meta-RL, which deals with how agents learn how to learn (Hospedales et al., 2021). Combining this with multi-objective RL, it becomes Multi-Objective Meta-Learning (MOML) (Ye et al., 2021). Future research could investigate the best way to learn which state representation to use for specific problems and how this evaluation could occur within a meta-strategy with team members with different goals. Representing the state of the world is a complex problem and is described by Sutton & Barto (2018) as "presently more art than science". This shows a knowledge gap, and future

studies should therefore be devoted to filling this gap so that the AI research community can develop best practices for safe and efficient state representations for different problems.

8. Conclusion

This study contributed to new understanding of the role of memory length for an RL algorithm in the IPD by showing how different memory lengths are preferable depending on the opponent. Based on the findings, a new algorithm presenting a novel approach to the IPD was created, called Mixed Memory Q-Learner. MMQL had the ability to switch state representation (memory length in the IPD) during play and to recognise its opponents between interactions. By comparing MMQL to other algorithms with fixed state representations and the ability to learn across several games, the results showed that the ability to flexibly switch between different memory lengths gave the algorithm an advantage against certain opponents. However, despite the predictions, MMQL did not perform better in the tournament overall. A likely explanation for the low ranking is that the algorithm only learned to exploit and defend itself but not cooperate. Future research could therefore extend the concept of MMQL to a Multi-Objective Meta Learning (MOML) problem, using different objectives and state representations depending on the strategy of the opponent, thus inciting more cooperation when advisable. The results hint at the importance of flexibility in the state representations for an RL-algorithm when partaking in a social dilemma.

9. References

Anastassacos, N., & Musolesi, M. (2018). Learning through Probing: a decentralized reinforcement learning architecture for social dilemmas. *ArXiv, abs/1809.10007*.

Axelrod Python library. (2015). Axelrod Documentation Page. https://axelrod.readthedocs.io/en/stable/

Axelrod, R. (1980). Effective choice in the prisoner's dilemma. *The Journal of Conflict Resolution*, 24(1):3–25.

Axelrod, R. (1984). *Evolution Of Cooperation.* Basic Books.

Axelrod, R. (1997). *The Complexity of Cooperation: Agent-Based Models of Competition and Collaboration.* Princeton University Press.

Bertino, E., Doshi-Velez, F., Gini, M.L., Lopresti, D., & Parkes, D. (2020). Artificial Intelligence & Cooperation. *ArXiv, abs/2012.06034*.

Chai, T., & Draxler, R. R. (2014). Root mean square error (RMSE) or mean absolute error (MAE)? – Arguments against avoiding RMSE in the literature. *Geoscientific Model Development*, *7*(3), 1247–1250. https://doi.org/10.5194/gmd-7-1247-2014

Glynatsi, N. (2020) *Understanding responses to environments for the Prisoner's Dilemma: A meta analysis, multidimensional optimisation and machine learning approach* (Doctoral dissertation, Cardiff University, Cardiff, United Kingdom). Retrieved from http://orca.cf.ac.uk/id/eprint/135221

Gupta, J.K., Egorov, M., & Kochenderfer, M.J. (2017). Cooperative Multi-agent Control Using Deep Reinforcement Learning. *AAMAS Workshops*.

Harper, M., Knight, V., Jones, M., Koutsovoulos, G., Glynatsi, N. E., & Campbell, O. (2017). Reinforcement learning produces dominant strategies for the Iterated Prisoner's Dilemma. *PLOS ONE*, *12*(12), e0188046. https://doi.org/10.1371/journal.pone.0188046

Hernandez-Leal, P., Kaisers, M., Baarslag, T., & Cote, E.M. (2017). A Survey of Learning in Multiagent Environments: Dealing with Non-Stationarity. *ArXiv, abs/1707.09183*.

Hoang, T.N., Xiao, Y., Sivakumar, K., Amato, C., & How, J. (2018). Near-Optimal Adversarial Policy Switching for Decentralized Asynchronous Multi-Agent Systems. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 6373-6380.

Horie, N., Matsui, T., Moriyama, K., Mutoh, A., & Inuzuka, N. (2019). Multi-objective safe reinforcement learning. *Artificial Life and Robotics*. Published. https://doi.org/10.1007/s10015-019-00524-2

Hospedales, T.M., Antoniou, A., Micaelli, P., & Storkey, A. (2021). Meta-Learning in Neural Networks: A Survey. *IEEE transactions on pattern analysis and machine intelligence, PP*.

Jackson, M. O. (2011). *A Brief Introduction to the Basics of Game Theory*. SSRN Electronic Journal, 1–21. https://doi.org/10.2139/ssrn.1968579

Jurišić, M., Kermek, D., & Konecki, M. (2012). A review of iterated prisoner's dilemma strategies. *Proceedings of the 35th International Convention MIPRO*, Opatija, Croatia, 2012, pp. 1093-1097.

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

Kaisers, M. (2012). *Learning against learning: evolutionary dynamics of reinforcement learning algorithms in strategic interactions*. (Doctoral dissertation, Maastricht University, Maastricht). Retrieved from: http://michaelkaisers.com/publications/2012_PhD_Kaisers.pdf

Lapan, M. (2018). *Deep Reinforcement Learning Hands-On: Apply modern RL methods to practical problems of chatbots, robotics, discrete optimization, web automation, and more* (2nd edition). Birmingham: Packt Publishing.

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

Leyton-Brown, K. (2008). *Essentials of Game Theory: A Concise, Multidisciplinary Introduction (Synthesis Lectures on Artificial Intelligence and Machine Learning)* (Illustrated ed.). Morgan and Claypool Publishers.

Li, J., & Kendall, G. (2014). The Effect of Memory Size on the Evolutionary Stability of Strategies in Iterated Prisoner's Dilemma. *IEEE Transactions on Evolutionary Computation, 18*, 819-826.

Lin, B., Bouneffouf, D., & Cecchi, G. (2020). Online Learning in Iterated Prisoner's Dilemma to Mimic Human Behavior. *ArXiv, abs/2006.06580*.

Merriam-Webster. (n.d.). Algorithm. In *Merriam-Webster.com dictionary*. Retrieved May 1, 2021, from https://www.merriam-webster.com/dictionary/algorithm

Nguyen, T. T., Nguyen, N. D., & Nahavandi, S. (2020). Deep Reinforcement Learning for Multiagent Systems: A Review of Challenges, Solutions, and Applications. *IEEE Transactions on Cybernetics*, *50*(9), 3826–3839. https://doi.org/10.1109/tcyb.2020.2977374

O'Riordan, C., Griffith, J., & Sorensen, H. (2003). Forgiveness in Strategies in Noisy Multi-agent Environments. *CEEMAS*.

Poundstone, W. (1993). *Prisoner's Dilemma: John von Neumann, Game Theory, and the Puzzle of the Bomb* (First Edition Thus ed.). Anchor.Markov

Pumperla, M., & Ferguson, K. (2019). *Deep Learning and the Game of Go* (1st ed.). Manning Publications.

Rand, D. G., & Nowak, M. A. (2013). Human cooperation. *Trends in Cognitive Sciences*, *17*(8), 413–425. https://doi.org/10.1016/j.tics.2013.06.003

Rapoport, A., & Chammah, A. M. (1965). *Prisoner's Dilemma* (First Edition). University of Michigan Press.

Rapoport, A. (1974). Prisoner's Dilemma — Recollections and Observations. In: Rapoport A. (eds) Game Theory as a Theory of a Conflict Resolution. Theory and Decision Library (An International Series in the Philosophy and Methodology of the Social and Behavioral Sciences), vol 2. Springer, Dordrecht. https://doi.org/10.1007/978-94-010-2161-6_2

Ross, D., (2019). Game Theory. In E. Zalta (Ed.), *The Stanford Encyclopedia of Philosophy* (Winter 2019 ed.). Stanford University (Winter 2019 ed.). https://plato.stanford.edu/entries/game-theory/

Samuelson, P. A. (1938). A Note on the Pure Theory of Consumer's Behaviour. *Economica*, *5*(17), 61. https://doi.org/10.2307/2548836

Sandholm, T. W., & Crites, R. H. (1996a). Multiagent reinforcement learning in the Iterated Prisoner's Dilemma. *Biosystems*, *37*(1–2), 147–166. https://doi.org/10.1016/0303-2647(95)01551-5

Sandholm, T. W., & Crites, R. H. (1996b). On multiagent Q-learning in a semi-competitive domain. *Lecture Notes in Computer Science Adaption and Learning in Multi-Agent Systems,* 191-205. doi:10.1007/3-540-60923-7_28

Seiffertt, J., Mulder, S., Dua, R., & Wunsch, D. C. (2009). Neural networks and Markov models for the iterated prisoners dilemma. *2009 International Joint Conference on Neural Networks*. doi:10.1109/ijcnn.2009.5178800

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., & Hassabis, D. (2017). Mastering the game of Go without human knowledge. *Nature, 550*(7676), 354–359. https://doi.org/10.1038/nature24270

Simon, H. A. (1982). *Models of Bounded Rationality, Volume 1: Economic Analysis and Public Policy* (1st ed.). The MIT Press.

Singer-Clark, T. (2014). Morality Metrics On Iterated Prisoner's Dilemma Players. Retrieved from: https://www.scottaaronson.com/morality.pdf

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning, second edition: An Introduction (Adaptive Computation and Machine Learning series)* (second edition). Bradford Books.

Watkins, C. (1989). *Learning from delayed reward* (Doctoral dissertation, King's College, Cambridge). Retrieved from: https://www.researchgate.net/publication/33784417_Learning_From_Delayed_Rewards

Watkins, C. J., & Dayan, P. (1992). Technical Note. Q-Learning. *Machine Learning, 8*(3-4), 279-292. doi:10.1007/BF00992698

Yannakakis, G. N., & Togelius, J. (2018). *Artificial Intelligence and Games* (1st ed. 2018 ed.). Springer.

Ye, F., Lin, B., Yue, Z., Guo, P., Xiao, Q., & Zhang, Y. (2021). Multi-Objective Meta Learning. *ArXiv, abs/2102.07121*.

Appendix A

Code

The code created to realise MMQL can be found on the author's Github via this link: https://github.com/dollbo/Thesis_project
To run the program, please contact the author for further instructions.

Appendix B

Markov Decision Processes description

A Markov Decision Process (MDP) is a framework from probability theory that is used to model sequential decision problems (Seiffertt et al., 2009). To find a policy for how an agent should behave in a certain environment, the RL problem is modelled as an MDP (Lapan, 2018). An MDP is an extension of a Markov Process (MP) and a Markov Reward Process (MRP).

An MP, also called Markov chain, describes a system that has different states that change over time according to some dynamic rules. The MP cannot be influenced, but only observed as the state changes. The *state space* is the name of the set of all possible states of a system, and the MP or Markov chain is constituted by a number of observations of different states of a system as it happens. To qualify as an MP the system must have the *Markov property.* This means that each state must contain information of all history of interaction that is relevant for the future (Sutton & Barto, 2018). This is because all the future states of the system must depend only on this state, which means that every state has to be distinguishable from other states (Lapan, 2018). An MP thus contains a set of states and a transition matrix that describes the probability of one state shifting to another state.

An MRP extends the MP framework by adding a value to the transitions to different states. This value is the reward. In the MRP, at a certain time point, the rewards from ensuing states are calculated and summed up as the return at that particular time point. However, how much the system considers the rewards gained at distant time steps is adjusted for by the parameter gamma γ. With a low gamma (value close to 0), only the immediate rewards are considered as important information about the state at that time. With a higher gamma (close to 1) distant rewards are also considered in the evaluation. This return quantity for each timestep can be calculated for the whole state space, which will give the state value. This means that every state has a value attached to it which is the expected return to get from that state given we follow the MRP.

To make an MRP into an MDP, actions are added to the process. This affects both the transition matrix and the reward matrix. The transition matrix changes as taking different actions lead to different states and therefore different actions have different probabilities of leading to certain states. The reward matrix is affected by this as the reward the agent receives is not just conditioned on the state it is in anymore, but also on the action that resulted in that state. As the goal of an RL agent is to obtain as high a reward as possible, the solution to an MDP is the policy that informs the agent which is the optimal actions to take in each state to obtain the highest return. An MDP is therefore a useful framework as it captures important aspects of the problem facing any learning agent that is trying to reach a goal by interacting with its environment. The MDP framework captures typical features of learning dynamics such as a sense of cause and effect, the uncertainty of which action will lead to

what state, and explicit goals of the agent (Sutton & Barto, 2018). Below is a formulated description of an MDP:

MDP is a 5-tuple (S, A, P, R, γ), where

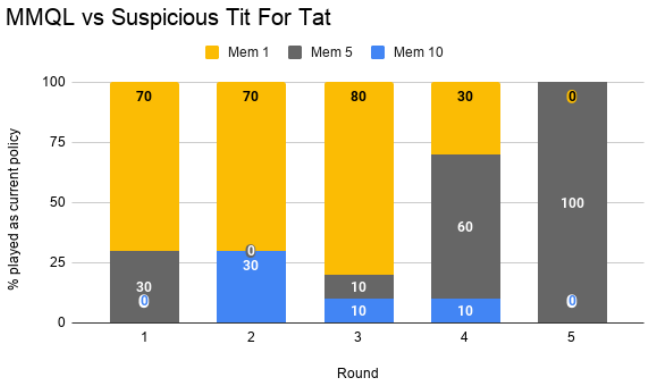
- S is a set of states
- A is a set of actions
- P(*s, a, s'*) is the probability that action *a* in state *s* at time *t* will lead to state *s'* at time *t+1*
- R(*s, a, s'*) is the immediate reward obtained by taking action *a* in state *s* which lead to state *s'*
- γ is the discount factor used to create a discounted reward

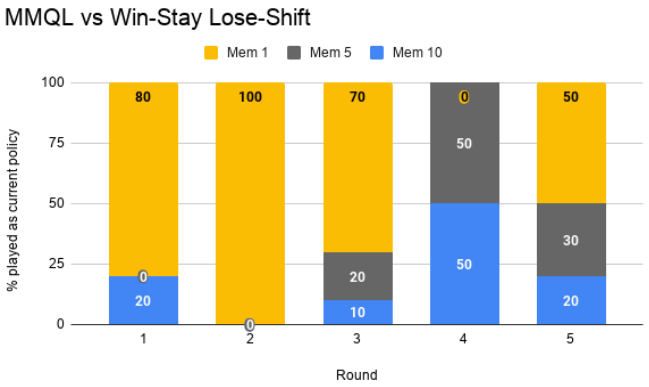
Appendix C

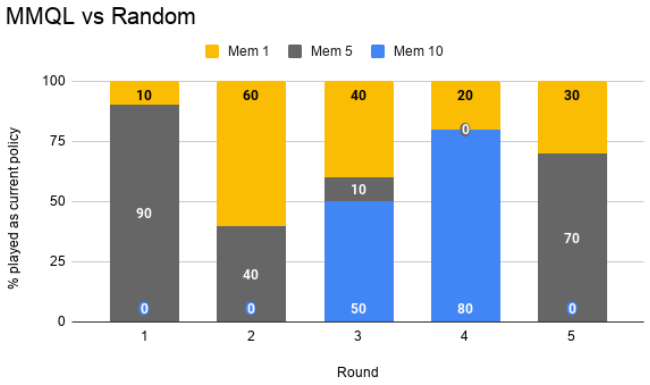
Figures from Test 2

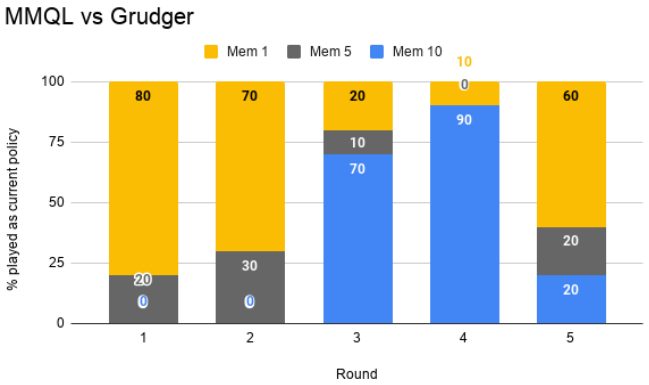
Figure C1

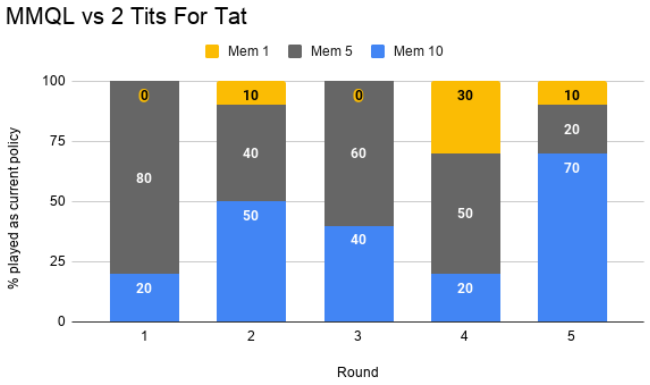
Percentage played with each player for a) Suspicious TitForTat, b) Win-Stay Shift-Lose, c) Random, d) Grudger, e) 2TitsForTat, and f) MMQL











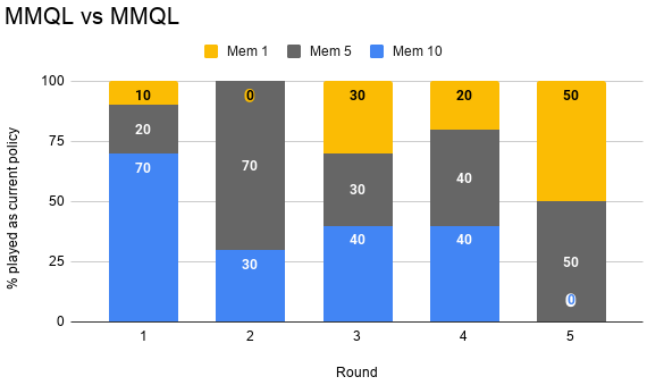


Figure C2

Convergence plots for the three players in MMQL (left column) and the three Regular Q-Learners (right column) against Cooperator and Defector

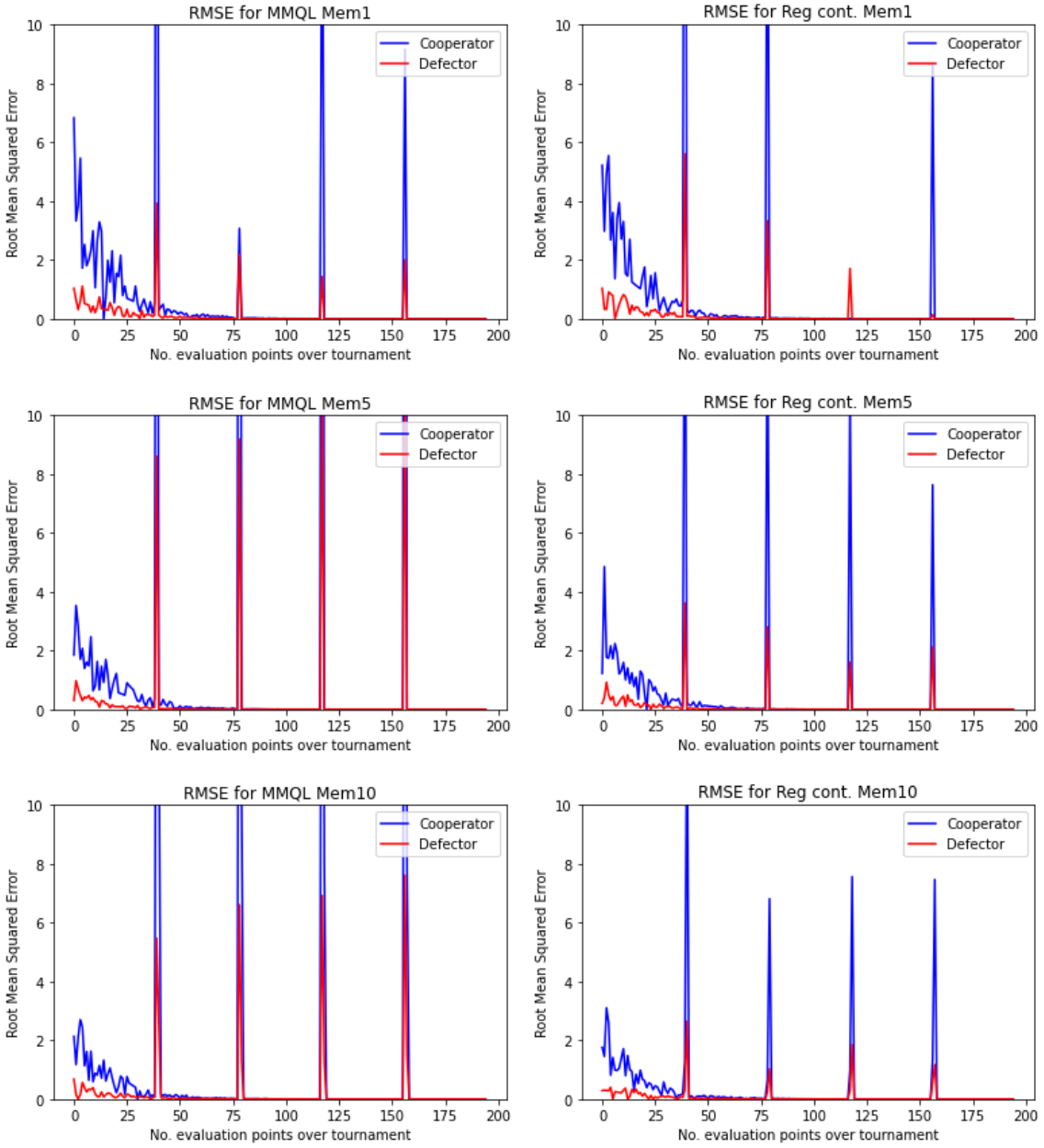


Figure C3

Convergence plots for the three players in MMQL (left column) and the three Regular Q-Learners (right column) against Grudger, Random & Win-Stay Shift-Lose.