university of groningen / faculty of science and engineering

Moving object recognition and avoidance using reinforcement learning

Bachelor's Project Thesis

Radu Alexandru Cosma, s3601072, r.a.cosma@student.rug.nl,
Supervisor: Marco A. Wiering

Abstract: This thesis explores the best way to handle multiple consecutive opponents in a reinforcement learning context. Q-learning, Double Q-learning and Sarsa, all using function approximation, are compared on a Gridworld game task with multiple levels in which an agent needs to pass an opponent and reach the goal. Monte-Carlo rollouts are used to improve action selection together with opponent models, represented using Multi-Layer Perceptrons. A comparison is made between different opponent modelling setups, with one opponent model for all opponents and two novel techniques compared. The novel techniques involve recognising whether an opponent has been previously encountered or is a new opponent. This is done by comparing the prediction losses on trajectories on which an opponent model was trained with the prediction losses on the new opponent. One method checks if these losses come from the same distribution, while the other whether there was a change-point in the losses when represented as a time series. Results show that the novel methods can predict the opponent considerably better on an illustrative example, with a more reduced improvement on more general deterministic opponents and no improvement on opponents with randomness. Overall mean final rewards are similar regardless of opponent modelling technique, with Sarsa performing best.

1 Introduction

Reinforcement Learning (RL) is a machine learning subset that enables an agent to learn how to behave in an environment by maximising a reward signal obtained from interacting with it (Sutton & Barto, 2018). This reward signal is an abstraction that represents a task that the agent needs to do, embedded in the environment associated with it. An environment can contain opponents that run counter to the agent's goals. This complicates the environment's representation (usually done using Markov Decision Processes (Bellman, 1957)), as well as how to best perform in such an environment, as the opponent's strategies and behaviour cannot be ignored. In such a situation opponent modelling is useful, where a model of the opponent's behaviour patterns is kept and taken into account when considering actions to take towards the agent's goals.

An example of an opponent modelling approach is Knegt, Drugan, & Wiering (2018b), where a Multi-Layer Perceptron (MLP) (Rumelhart, Hinton, & Williams, 1985) is used to predict the next move of an opponent, which is then used in Monte-Carlo rollouts (Tesauro & Galperin, 1996) in order to pick the best action to take in a state by simulating future game states. This MLP representation improves upon other probabilistic opponent modelling methods in that it can generalise well (Knegt et al., 2018b). In one versus one games such as Tron (Knegt et al., 2018b) that will only have one opponent for their entire duration, only one opponent model is sufficient. However, such an approach might have problems in environments where different opponents appear consecutively (and might appear more than once), as in a tournament or a level-based game. This is because different opponents might act differently in the same states, making the single opponent model pick the action most often encountered in those states, which does not model all opponents correctly. Even using a Long Short-Term Memory network (Hochreiter & Schmidhuber, 1997) might not be appropriate if the state-action combinations do not have a useful short-term temporary component.

In such cases, the opponent modelling needs to account for multiple possible opponents in a successful way. This includes remembering previous opponents and recognising when the current opponent behaves like a previous one, in order to use the already created model for that previous opponent instead of learning a new one.

The aim of this thesis is to find the best approach for learning to perform in such a multiple-opponent environment. Two novel approaches are proposed for opponent recognition. They attempt to match an MLP to one opponent and record state-action histories for that opponent. When a new opponent is encountered, a short state-action history is recorded and the existing MLPs are evaluated on both histories. One approach entails evaluating whether those loss samples come from the same overall distribution, while the other checks whether a change-point did not occur between the two loss samples. An affirmative answer to either of those queries can indicate that the new opponent is the same as the one the MLP and history belong to.

A level-based game is designed for this task, and combinations of learning algorithms using function approximation (Sarsa (Sutton & Barto, 2018), Q-Learning (Watkins & Dayan, 1992; Mnih et al., 2015) and Double Q-Learning (Van Hasselt, Guez, & Silver, 2016)) and opponent modelling techniques (one opponent model for all opponents and the two novel methods) are compared. Similarly to Knegt et al. (2018b), the opponent models are used in Monte-Carlo rollouts and state is represented using vision grids (Shantia, Begue, & Wiering, 2011) (representations focused on the proximities of the agent and opponents) with a global goal component added. There will be three instances of the game examined in order to see the robustness of the algorithms used across different settings. One has specifically made opponents for illustrating the benefits of the novel approaches, one has more general deterministic opponents and one has those same general opponents but with a random component added to them.

1.1 Contributions of this thesis

This thesis proposes two new novel techniques for opponent recognition as presented above. They are an alternative for tasks where usual recognition methods are inadequate, such as low-data scenarios or tasks with a variable number of opponents. Furthermore, the loss-based approaches explored in this thesis could be extrapolated to other areas (beyond opponent modelling) where comparing two MLPs in such a way can prove useful.

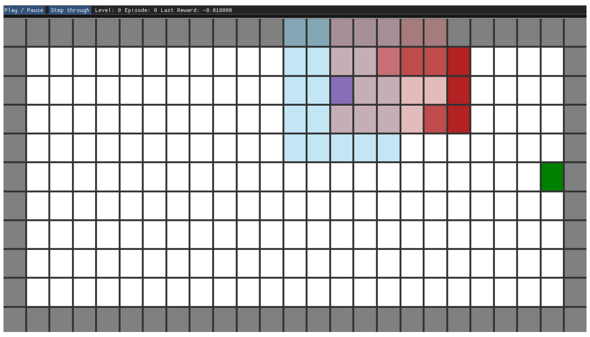

Figure 2.1: Screenshot of the simulation user interface representing a level of a game in progress. The agent is blue, the opponent's head is red, the opponent's tail is dark red and the goal is green. The vision grids for both agent (seeing the opponent's head and the wall) and opponent (seeing some of its own tail and the wall) are also represented with lighter colours.

This thesis also tests the usefulness of vision grids as a local state representation when combined with a global goal representation. It also provides insight into whether vision grids and function approximation can generalise across multiple opponents.

2 Game Description

The game used in this thesis is a multiple level sequential Gridworld (Sutton & Barto, 2018) game, where the purpose of the agent is to pass all levels by reaching their goals. The levels always appear in the same order and each level has the agent's starting position, one goal, walls, and an opponent resembling a snake. These differ from level to level. The agent and opponent can only move up, down, left or right, and the walls and goal are static. At each time step, the agent and opponent both make a move simultaneously. When the goal from one level is reached by the agent, it is transported to the next level and is placed randomly in one square of a 7-square vertical line with the centre at the level's starting position. When the agent is hit by the opponent, it loses the game. When the agent moves into a wall, its position is the same as its previous one. When the final level's goal is reached, the agent wins the game. If 1000 time steps have passed in a game, the game also ends. An episode consists of one game. An example of a level from a game in progress is shown in Figure 2.1.

The opponent has a head, representing its current position, and a tail of a certain size that follows its movements. The opponent follows a preset route sequentially. There are no intersections within the route. When the agent is transported to a new level, the opponent's head is set to a random position along this route and its tail occupies the closest previous positions along the route.

The opponent can be of a deterministic type, always moving on the set route, or have a stochastic component (referred to as "random opponent type"), where after each move, an alternative way to get to the route's next position is picked $RandCoef\%$ of the time, where $RandCoef$ is a hyperparameter describing how stochastic the opponent's movement is. This alternative way represents the shortest possible detour towards the next position without collisions with the walls, goal or the opponent itself. For example, if the next position is left, the opponent will attempt to go down, then left, then up. If this is not possible due to collisions, then it will attempt to go up, then left, then down. If that is not possible either, no detour is made. The same process is valid for all directions, with the random detours first attempting to go down, then up, then left and then right in that order.

3 Reinforcement Learning

3.1 Markov Decision Process

Since the game's next state depends only on the game's last state and the action taken in that state, and since the levels are predetermined and not changing, the game can be modelled as a Markov Decision Process (MDP) (Bellman, 1957). An MDP is defined by a set of states (states of the game), a set of actions that can be taken in those states (left, right, up, down in the level), a transition function that describes the probability of reaching a state from another when taking a specific action (game rules implemented by a computer simulation), and a reward function specifying the rewards that are received for those transitions.

For this game, the rewards are 1 for reaching the goal, -1 for hitting the opponent and -0.01 for any other move. If 1000 time steps are reached, the game simply stops with no reward being sent to the agent. The negative step reward ensures that the quickest path is learned, and the small values ensure that it can be properly approximated with function approximation (and also yielded better results in initial experiments).

Each level is trained as if it is the only one, meaning that reaching the goal (as opposed to only the final level's goal) and hitting the opponent are considered terminal states. This is done because the agent has no information about which level it is in (the level transitions are done by the simulation), so it cannot model the entire game state properly. This setup makes the agent's behaviour more general with respect to the level and initial experiments have shown that performance is better with it.

3.2 Policy and action-value function

In order to perform well, the agent needs to learn a policy π, which matches each state with the probabilities of picking each possible action from that state (Sutton & Barto, 2018). The expected return from a state s when following the policy π and taking action a is defined by the action-value function $q_\pi(s,a)$:

$$q_\pi(s,a) = E_\pi\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, a_0 = a\right] \tag{3.1}$$

where E_π is the expected value, r_t represents the reward at time t and γ is the discount factor that discounts future rewards so that immediate rewards are deemed more important, with $\gamma \in [0,1]$. Since computing these exact values is not feasible due to them involving all future state sequences, they are estimated using reinforcement learning, with $Q(s_t, a_t)$ being the estimate for $q_\pi(s_t, a_t)$.

In order to perform well, the agent needs to balance exploring the environment to find the best state-action pairs and exploiting those pairs in order to perform well. This balance is achieved using an $\epsilon - greedy$ exploration policy for picking an action, where ϵ is a hyperparameter controlling the amount of exploration:

$$A(s_t) = \begin{cases} \arg\max_a Q(s_t, a) & \text{with probability } 1-\epsilon \\ \text{a random action} & \text{with probability } \epsilon \end{cases} \tag{3.2}$$

3.3 Function approximation

Since our problem has a large number of states due to the large number of possible routes that the opponents might take, all the state-action pairs cannot be realistically represented in, for example, a lookup table. In such a situation, function approximation is used, where a function approximator (an MLP in this thesis) is used to estimate the Q-values $Q(s, a)$ for each action a from state s when given state s as input. The MLP is trained using backpropagation (Rumelhart et al., 1985) by updating the weights of the MLP based on a target Q-value $Q_{target}(s_t, a_t)$ with respect to the predicted Q-value of the performed action. The MLP also has a learning rate hyperparameter α, that determines how big the updates taken towards the target are.

The reinforcement learning algorithms used are presented in the following sections. Since the function approximator already has a learning rate α, the methods below will use a learning rate of 1, simplifying the formulas involved.

3.4 Sarsa (SAR)

State-action-reward-state-action, or Sarsa, is an on-policy control algorithm where the updates are modelled based on transitions between state-action pairs. This means that after a reward r_t is received and a new state s_{t+1} obtained, the next action to be taken a_{t+1} is picked and the estimated Q-value $\hat{Q}(s_{t+1}, a_{t+1})$ is used for the update. When the state is not terminal, the target Q-value is:

$$Q_{target}(s_t, a_t) \leftarrow r_t + \gamma \hat{Q}(s_{t+1}, a_{t+1}) \tag{3.3}$$

When the state is terminal, the target is updated solely using the reward obtained:

$$Q_{target}(s_t, a_t) \leftarrow r_t \tag{3.4}$$

The MLP is updated with stochastic gradient descent and Q_{target}.

3.5 Experience replay

Experience replay (Lin, 1992) is a technique which stores the agent's experiences, allowing them to be reused multiple times for updating the function approximator. In this way, the experiences may be utilised better (Knegt, Drugan, & Wiering, 2018a; Mnih et al., 2015) than when only using each experience once as described above with Sarsa. An experience is of the form (s_t, a_t, r_t, s_{t+1}), indicating the action a_t performed in state s_t, upon which reward r_t was received and the state s_{t+1} was reached. Experiences are used in mini-batches for the mini-batch gradient descent update, computing the average gradient over *batchSize* (hyperparameter) experiences and using that to update the network. This mini-batch update can make learning more stable. The mini-batches are sampled randomly from the experience history, which is at first filled with *historySize* (hyperparameter) experiences. Afterwards, when a new experience is observed, it replaces the oldest experience in the experience history. This matches the design of Mnih et al. (2015). A design where the history was emptied and refilled periodically instead was also tried, but it performed slightly worse in initial experiments. Experience replay was only used with Q-learning and Double Q-learning (see Subsections 3.6 and 3.7), not Sarsa (see Subsection 3.4).

3.6 Q-learning (QL)

Q-learning (Watkins & Dayan, 1992) is an off-policy control algorithm for estimating $q_\pi(s, a)$. Unlike Sarsa, the maximum Q-value over all actions in the next state s_{t+1} is used for the update. However, one more modification is made, as per Mnih et al. (2015). A target MLP is used in the update, whose Q-values are denoted with Q'. It is a copy of the current MLP taken every C (hyperparameter) time steps. This is done so that the update is more stable and also produced slightly better results in initial experiments. For terminal states, the target Q-value is taken from Equation 3.4. For non-terminal states, it is:

$$Q_{target}(s_t, a_t) \leftarrow r_t + \gamma \max_a \hat{Q}'(s_{t+1}, a) \tag{3.5}$$

3.7 Double Q-learning (DQL)

A problem with Q-learning is that Q-values can be overestimated due to always picking the action corresponding to the maximum Q-value. A solution is Double Q-learning, which uses another function approximator (or lookup table) in order to pick the action which will be used in the update. Van Hasselt et al. (2016) devised a way to use the current

MLP as the second function approximator, picking the action that is used by the target MLP. Therefore, if Q-learning suffers from a maximisation bias that hinders performance, Double Q-learning will perform as expected. For terminal states, the target Q-value is the same as in Equation 3.4. For non-terminal states, it is:

$$Q_{target}(s_t, a_t) \leftarrow r_t + \gamma \hat{Q}'(s_{t+1}, \arg\max_a \hat{Q}(s_{t+1}, a)) \tag{3.6}$$

4 State representation and opponent modelling

4.1 Vision grids

Both the agent's and the opponents' state are represented using vision grids (Shantia et al., 2011), similarly to Knegt et al. (2018b). A vision grid is a grid representation of the presence of an object in a square area centred around the item of interest (in this case, agent or opponent). Therefore, if that object is in a certain position in that area that position will be encoded with a 1 and with a 0 otherwise. There is a vision grid for each type of relevant object. For the agent, there will be two vision grids: for the walls and for the opponent. For an opponent, there will be three vision grids: for the walls, for the opponent itself and for the goal. There are two vision grid sizes explored, a 3x3 square with area 9 and a 5x5 one with area 25 (hyperparameter *visionGridArea*). An example of vision grid areas of size 25 is shown in Figure 2.1 for both agent (light blue) and opponent (light red).

The goal is not represented as a vision grid for the agent. The reason is that the agent, knowing the current level's layout, also knows where the goal is, and not providing access to that goal information unless it is in the agent's proximity means that the agent has no sense of direction towards it. Whatever bias towards the goal's general direction might be encoded in the Q-values is lost if the goal positions vary across levels or if the agent is forced to take a detour by the opponent.

Therefore, the goal is represented as four values in the state as the difference in positions between the goal and the agent on both axes. The difference on an axis $dist$ is represented by two values: $d_{pos} = \max(dist, 0)$, $d_{neg} = |\min(dist, 0)|$. These values are always divided by 5, making them similar in scale to the vision grid values. This representation led to remarkably improved performance in initial experiments.

The final input sizes for the agent and opponent MLPs depend therefore on the vision grid sizes, which will be passed in a flattened form to the MLPs: for the agent, it is $visionGridArea * 2 + 4$, and for an opponent it is $visionGridArea * 3$.

4.2 Opponent MLP setup

An opponent modelling MLP will receive the opponent's state as described above and output 4 probabilities, each corresponding to the probability that the opponent takes that action given the input state. It will use the Softmax activation function for the output layer, which outputs probabilities $P(s_t, a_i)$ corresponding to the probability of taking action a_i in state s_t according to the formula:

$$P(s_t, a_i) = \frac{e^{a_i}}{\sum_{k=1}^{4} e^{a_k}} \tag{4.1}$$

4.3 Opponent modelling techniques

In this thesis, three different opponent modelling techniques (referred to as OMTs) are compared. The first one is where one MLP is used to model all opponents simultaneously. This technique will be referred to as SMA (Single MLP for All opponents). This can be considered the baseline and most straightforward approach. Additionally, two novel methods are proposed.

The purpose of both approaches is to distinguish between different opponents. The set of opponents that they find will be kept and updated continuously (they are not reset by episode or level). If there are no stored MLPs when entering a level, a new one is created that starts training on the opponent, as well as keep a history of opponent state-action pairs encountered. This history has a maximum size defined by the hyperparameter *maxHistorySize*. When this level is finished, the MLP and its history are only kept if the history size for that MLP is larger than or equal to *minHistorySize*, another hyperparameter. Otherwise, they are deleted.

When reaching a new level, *minHistorySize* state-action pairs for the new opponent are recorded while the agent picks the stored MLP with the smallest cumulative prediction loss on the newly recorded pairs to model the new opponent (until an opponent model is found).

When $minHistorySize$ pairs have been recorded, a statistical test is performed for each MLP and their history. This test is between the MLP losses when predicting the next action from the MLP's state-action history and the MLP losses when predicting the new opponent's next action from its state-action pairs. These statistical tests (detailed below) produce a significant value when the two opponents are likely not the same. The highest p-value between those tests is compared to a hyperparameter $pValueThreshold$ (threshold of significance). If it is higher, the MLP associated with that p-value is deemed fit to model the new opponent as well. If it is lower, none of the stored opponent MLPs correctly model the new opponent and a new MLP is created.

In any case, an MLP is found for the new opponent. The new opponent's state-action pairs are added to its history and the MLP is trained on them in order. The MLP is then trained on and used to predict the new opponent's movement. The MLP's state history is filled as needed under the $maxHistorySize$ constraint. Once this new level is over, the MLP and history are kept if the history size is larger than or equal to $minHistorySize$ and the next level is handled in the same way as above.

The difference between the two methods is in the statistical test performed and the way the state history is kept. In the first method, the statistical test is the Kolmogorov-Smirnov two-sample test (Verzani, 2004), that determines whether two one-dimensional probability distributions that generated two samples of data belong to the same overall distribution. This method will be referred to as LDC (Loss Distribution Comparison) from now on. This is done by computing the distance between the two empirical distribution functions (ECDF) (Verzani, 2004) of the samples. The ECDF is given by:

$$F_n(x) = \frac{\#\{i : x_i \leq x\}}{n} \tag{4.2}$$

Which determines the number of elements smaller than x from the sample with respect to sample size n. The Kolmogorov-Smirnov statistic is then:

$$D_{n,m} = \max_x |F_n(x) - G_m(x)| \tag{4.3}$$

where F_n and G_m are the ECDFs of the samples of sizes n and m. Large values of $D_{n,m}$ are significant. The exact determination of significant values is convoluted and not presented. The simulation code for this test was adapted from (Brun & Rademakers, 1997), which used Eadie, Drijard, & James (1971) as reference.

In this case, the samples are defined as above and the distribution is defined by the evaluation of the MLP on the opponent. Therefore, the two samples of losses are tested to see if they were generated by the same MLP-opponent combination. This allows the two samples to be taken in a discontinuous fashion. Thus, state-action pairs are added to the MLP's history until it is filled. These pairs will likely be from the opponent the MLP was created for if $maxHistorySize$ is small enough. In this way, even if misclassifications occur in the future, the history is unlikely to get filled with states from another opponent.

For the second method it is desired to see whether there is a change-point in the sequence of losses defined by evaluating an MLP on its state-action history and the state-action pairs of the new opponent, point at which the underlying probability distribution changes. This change would correspond to a different opponent being modelled, since the distribution is defined by the evaluation of the MLP on one opponent. This method is referred to as CPD (Change-Point Detection). The test performed is an adapted version of Pettitt's change-point detection test (Pettitt, 1979). The test computes U_k using the ranks $r_i, \ldots, r_n$ of the sample sequence of length n (Pohlert, Pohlert, & Kendall, 2016):

$$U_k = 2\sum_{i=1}^{k} r_i - k(n+1), k = 1, \ldots, n \tag{4.4}$$

The test statistic $\hat{U}$ is usually the maximum absolute value of U_k. However, since the change-point is known to either reside or not between the two samples, the test will be adapted to pick $\hat{U} = |U_i|$, where i is the index corresponding to the first data point in the second sample. This performed better than the original test in initial experiments. The null hypothesis of no change-point can be rejected for confidence level α if $p < \alpha$, where p is:

$$p = 2\exp\frac{-6\hat{U}^2}{(n^3 + n^2)} \tag{4.5}$$

The simulation code for the test was adapted from Pohlert et al. (2016).

Since the states are now part of a time series, the history mechanism changes to a queue, where when $maxHistorySize$ capacity is reached, the oldest state is discarded and the newest state added to the state history. This constantly updates the history even in cases of misclassification and might be useful to model groups of opponents together.

Since both tests are non-parametric, they do not assume any underlying distribution characteristics and are therefore suitable for the loss distributions to which they are applied. An important note is that the MLPs used for the tests are evaluated, not trained, on the state histories and thus kept constant during the tests. Therefore, the loss samples are independent. An early version of the change-point test was done on the loss while the MLP was updated, but since this broke the assumptions of the statistical tests it was removed. This also performed worse than the two methods presented above.

An attempt was made to use mini-batch gradient descent for the novel techniques by averaging samples for more stable losses. It was, however, discarded due to low sample size for the new opponent when needing to decide on an MLP.

4.4 Monte-Carlo rollouts

In order to make use of the opponent models Monte-Carlo rollouts (Tesauro & Galperin, 1996) are used for action selection for the agent. A value $Q_{sim}(s_t, a)$ representing the simulated Q-value of picking action a in state s_t is computed for each action. This simulated value is computed by performing *nrRollouts* rollouts of *nrSteps*, with *nrRollouts* and *nrSteps* as hyperparameters. A rollout is a simulation of the current level starting from the current state s_t and initially picking action a. The opponent's action is predicted using the current opponent's MLP, the next level state is simulated and the reward is received. The simulations respect the MDP structure outlined in Subsection 3.1.

Afterwards, the process is repeated but with $\arg\max_a Q(s_{t+1}, a)$ as the agent's action. Besides the first already-picked action, there will be *nrSteps* repetitions of the process performed as described above. If *nrRollouts* is 1, the opponent action with the highest probability is picked. If *nrRollouts* is bigger than 1, the action is picked according to the probability distribution over the actions defined by the output of the opponent modelling MLP.

The final $\hat{Q}_{sim}(s_t, a)$ is the sum of obtained discounted rewards during that rollout. If the rollout ends before the level is finished, the discounted Q-value of the last action performed is added to the rewards of that rollout. $\hat{Q}_{sim}(s_t, a)$ for all the rollouts for an action a are averaged to obtain four $Q_{sim}(s_t, a)$ (one for each action), which represent the final values of the state s_t that the reinforcement learning algorithms use. All three algorithms use them to pick the best action to take according to the $\epsilon - greedy$ policy, while Sarsa also uses them to update the function approximator ($\hat{Q}(s_{t+1}, a_{t+1}) = Q_{sim}(s_{t+1}, a_{t+1})$ in Equation 3.3).

5 Experimental setup

There will be eight agents with different random seeds for each experiment in order to see general behaviour. The agents will be trained on 10000 episodes. The hyperparameter ϵ was linearly annealed from its initial value to 0 over the first 75% of episodes and kept at 0 for the rest in order to gain a measure of the final learned performance.

The function approximator MLP and opponent MLPs use one hidden layer of size 200 and one bias, one output layer of size 4 and one input layer of size determined by the vision grid size and goal setup as described in Subsection 4.1. Initial experiments showed that one hidden layer of size 200 is enough for both function approximation and opponent modelling. The activation function for both their hidden layers is the Sigmoid function, defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{5.1}$$

The activation function for the output layer of the function approximator MLP is the linear function and for the opponent MLPs is the Softmax function as described in Subsection 4.2.

All MLPs are initialised using a common heuristic (Glorot & Bengio, 2010), where biases are 0 and the weights of the MLP W_{ij} at each layer are initialised from the uniform distribution U over an interval determined by the number of nodes n in the previous layer:

$$W_{ij} \sim U\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right] \tag{5.2}$$

This performed better in initial experiments than initialising all weights and biases from $U[-x, x]$ (with $x \in \{1, 0.4, 0.05, 0.04, 0.01\}$ tested). The opponent MLPs will be trained using stochastic gradient descent. The function approximator MLP will be trained as detailed in Section 3.

5.1 Game designs

Two different games will be explored. The Simple game consists of three levels, shown in Figure 5.1 (see caption). This game is an example that shows the benefits of the novel opponent modelling techniques. As can be seen, from some specific states, each opponent takes a different action (left, right or forward). The Complex game consists of nine levels, shown in Figure 5.1 (see caption). The opponents are more general and any overlapping states that they may have are due to the relatively simple movement patterns.

Both games have opponents that have a tail size of 6 (an example of which is shown in Figure 2.1) and respect the game and randomness rules shown in Section 2. The Simple game will be tested with Sarsa, all three opponent modelling techniques, and the deterministic opponent. The Complex game will be tested with all combinations of reinforcement learning algorithms, opponent modelling techniques and opponent types (deterministic and random). The random opponent type's $RandCoef$, as discussed in Section 2, will be set to 0.2.

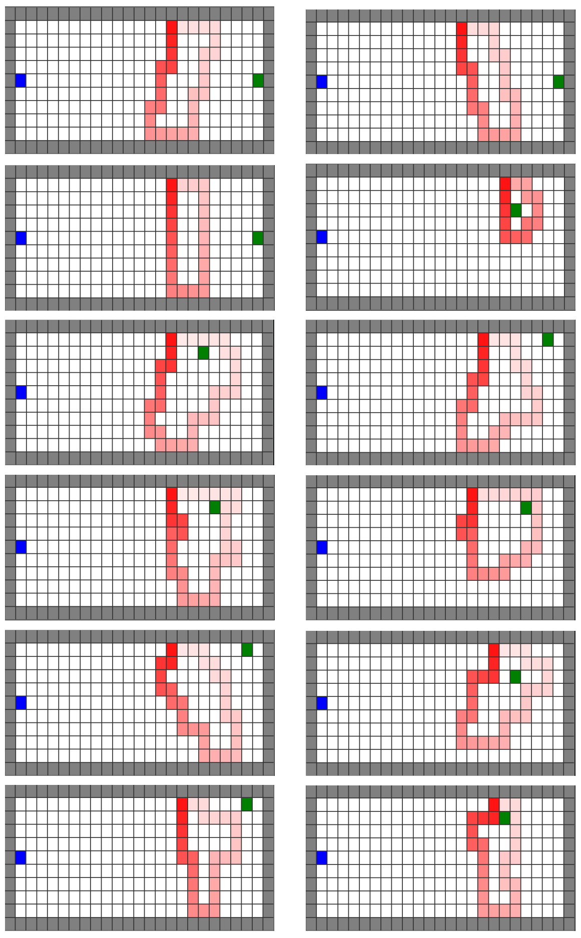
Figure 5.1: Levels used in the two games. The first three levels are for the Simple game and the next nine are for the Complex game, counting right and then down in order. The opponent moves in the direction in which the red becomes less pronounced. The starting position of the agent is shown with blue.

5.2 Hyperparameter optimisation

The hyperparameters were first optimised on the Complex game. The metric to maximise over was defined as the mean over all eight agents' mean rewards over the last 2500 episodes. This was done by performing a grid search on ranges of hyperparameter values for each algorithmic component. The best values would then be used for the next component and so forth. Good initial values were found manually. The order of optimisation was: opponent modelling techniques, learning algorithm, experience replay, function approximator MLP, opponent modelling techniques again, opponent MLP and Monte-Carlo rollouts. Components that did not apply to a particular setting (such as experience replay with Sarsa) were skipped.

For the Simple game hyperparameters, the best values for the corresponding algorithm combinations were taken from the Complex game, but the opponent modelling techniques were optimised again with the same metric as above. Furthermore, 0.2 was also added to the $pValueThreshold$ hyperparameter range when optimising. The ranges of

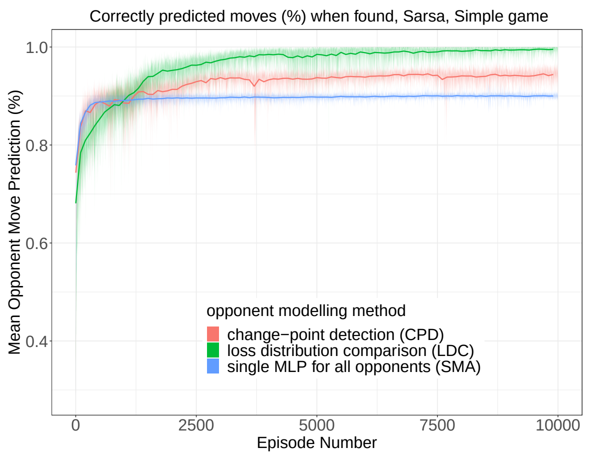

Figure 6.1: Mean opponent move prediction percentages after opponent model was found (FPP), Simple game. All plots represent the mean over 8 agents. The transparent areas for each algorithm are standard errors. The lines are defined by 100 points, the mean over the 100 closest episodes, to help perceive differences.

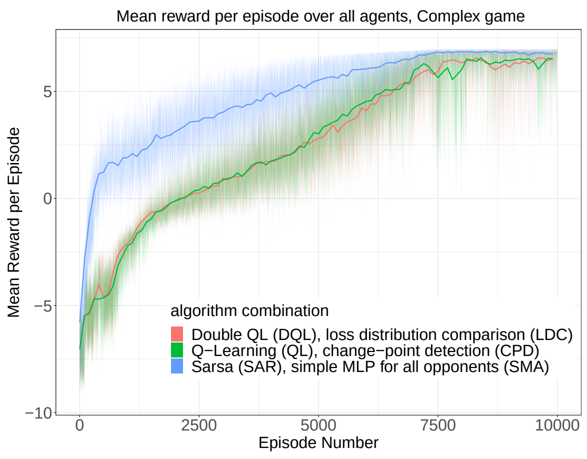

Figure 6.2: Mean rewards of the best opponent modelling technique (OMT) for each learning algorithm, Complex game.

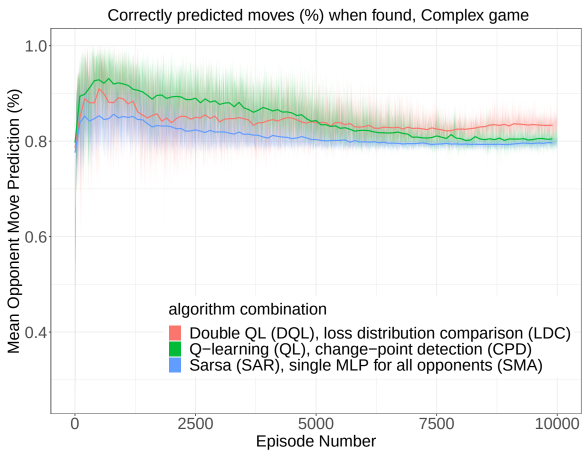

Figure 6.3: Mean FPPs of the best OMT for each learning algorithm, Complex game.

values, initial values and obtained values for both games are given in Appendix A.1.

6 Results

All results specified in this and following sections refer to the mean over all eight agents' mean performance over the last 2500 episodes. The results evaluated were the rewards (referred to as "mean final rewards"), opponent move prediction percentages (referred to as "PP") and opponent move prediction percentages after an opponent model has been found for the two novel methods (referred to as "FPP", this term refers to PP if used for SMA for ease of use). Statistical tests were done to compare the best algorithm combination for each opponent type against variations in each algorithm separately. See Appendix B.1 for the exact procedure and results.

Table 6.1: Means and standard deviations over all 8 agents of opponent move prediction percentages (PP) and PPs after opponent model was found (FPP) for the last 2500 episodes, Simple game.

	PP (%)	FPP (%)
SMA	0.9±0.001	-
LDC	0.958±0.006	0.993±0.007
CPD	0.914±0.009	0.941±0.013

For the Simple game, the novel methods performed significantly better than the SMA baseline in terms of both PP and FPP (see Table 6.1, Table B.1). The difference can also clearly be seen in Figure 6.1, where despite the slow increase in FPP, LDC manages to almost perfectly predict the opponents' movements.

For the Complex game and both opponent types,

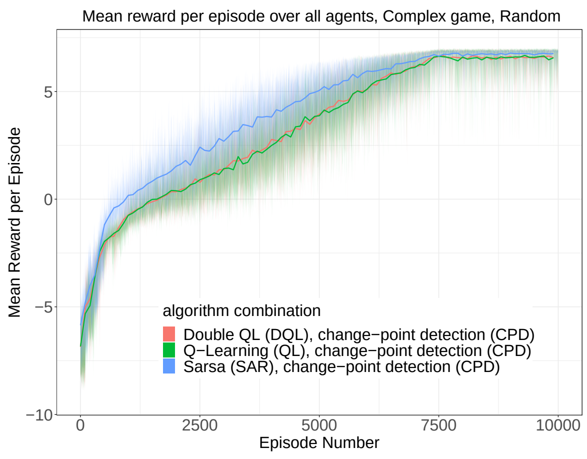

Figure 6.4: Mean rewards of the best OMT for each learning algorithm, Complex game, random opponent type.

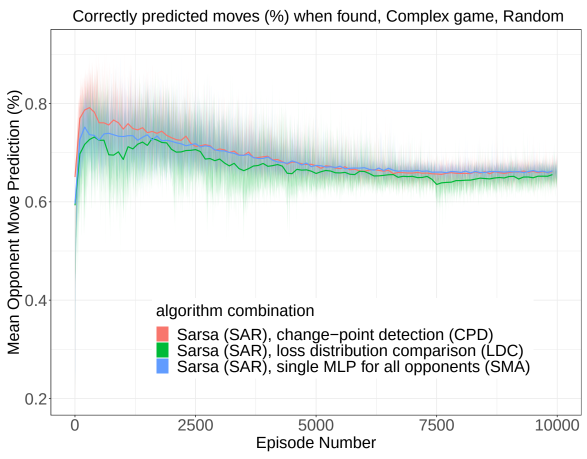

Figure 6.5: Mean FPPs for all OMTs for Sarsa, Complex game, random opponent type.

Sarsa outperforms the other learning algorithms (Figure 6.2 for deterministic, Figure 6.4 for random opponent types). The different OMTs produce similar final rewards for the same learning algorithm (see Table 6.2).

For the deterministic opponent type, in terms of PP, the statistical results are fluctuating between the SMA baseline being better or worse than the novel methods. However, FPP is always significantly higher with the novel methods (see Table B.2), as can also be seen in Figure 6.3 for the best performing OMT for each learning algorithm.

For the random opponent type, CPD was the method that performed best across all learning algorithms (see Table B.3). However, SAR&CPD predicted only one opponent (see Table A.1), leading to the very similar results to SMA. It still had lower PP than SMA, as well as higher PP than the rest. This suggests that the novel methods performed worse in terms of PP.

FPPs are also worse or not significantly worse than SAR&CPD, an example of which is Figure 6.5, where LDC has the lowest FPP despite reporting much more opponents. This indicates that a single opponent model was superior for the random opponent type regardless of whether this was achieved through SAR&CPD or the SMA baseline.

Overall, for the Complex game, differences in PP or FPP do not seem to impact mean final rewards for either opponent type.

7 Conclusion and Discussion

According to the results obtained, Sarsa is the best performing learning algorithm. This could be because Sarsa is helped by learning from the Monte-Carlo rollout Q-values in an on-policy fashion as described in Subsection 4.4. Learning from Monte-Carlo rollouts can also be done for QL (Knegt et al., 2018a), where it led to better performance, and similarly for DQL, but this was not explored in this thesis. QL and DQL performed similarly, meaning that most likely no maximisation bias was encountered.

The benefit of the novel OMTs can most easily be seen in the Simple game. Since in that game, each opponent takes a different action from some specific states (either left, right, or forward), the baseline SMA cannot properly represent them all. Therefore, the PPs and FPPs for the novel methods are much better than the SMA. The heuristic chosen until an opponent is found hinders PP accuracy as seen in Tables 6.1 and 6.2, but still performs satisfactorily.

A smaller benefit in terms of FPP was seen for the deterministic opponent in the Complex game, which has more general opponents. The decrease is likely because of only having a small subset of the opponent's full route to sample from. This is due to *minHistorySize* being small, which is a direct consequence of the negative step size reward that encourages finding the fastest paths towards

Table 6.2: Means and standard deviations over all 8 agents of mean final rewards, PPs and FPPs for the last 2500 episodes, Complex game, both opponent types.

	deterministic opponent type			**random opponent type**		
	reward	**PP (%)**	**FPP (%)**	**reward**	**PP (%)**	**FPP (%)**
SAR&SMA	6.81±0.08	0.794±0.002	-	6.73±0.04	0.661±0.001	-
SAR&LDC	6.76±0.10	0.791±0.008	0.818±0.009	6.69±0.05	0.632±0.006	0.647±0.01
SAR&CPD	6.78±0.05	0.771±0.002	0.801±0.002	6.73±0.03	0.645±0.002	0.659±0.002
QL&SMA	6.14±0.39	0.666±0.003	-	6.5±0.1	0.639±0.003	-
QL&LDC	6.16±0.42	0.809±0.011	0.834±0.013	6.52±0.1	0.624±0.01	0.639±0.012
QL&CPD	6.27±0.28	0.779±0.003	0.805±0.004	6.57±0.06	0.641±0.003	0.66±0.002
DQL&SMA	5.95±1.48	0.668±0.009	-	6.55±0.11	0.664±0.002	-
DQL&LDC	6.35±0.39	0.806±0.008	0.83±0.008	6.53±0.05	0.632±0.008	0.648±0.009
DQL&CPD	5.8±0.35	0.773±0.006	0.801±0.009	6.58±0.08	0.632±0.001	0.65±0.002

the goals. This was by design, as the comparison between OMTs should be done in a common setting. In most adversarial environments, the agent cannot wait indefinitely until it has perfectly determined its opponent since the opponent might move first and gain an overwhelming advantage.

This constraint on sample size can lead to low statistical power, which is supported by the Simple game's hyperparameter optimisation picking very high *pValueThresholds* of 0.1 and 0.2. Another possibility is that the agent has learned how to avoid opponents without over-relying on the opponent modelling technique and when reaching the second level it can also reach the third one and so on reliably. This means that it does not have enough time to train on the second opponent before moving onto the third and thus it cannot distinguish between them because both of their sample losses are large and random-looking. This is supported by the constant, as opposed to abrupt, FPP decreases in Figures 6.3 and 6.5. Initially, the agent can only pass few or no levels and thus the OMTs model only a few opponents. As the agent gets better it encounters more opponents which the OMTs need to model, making their task more complex and resulting in the FPP decrease. These issues, combined with the randomness of the random opponent type are most likely what lead to the lower PPs and FPPs for the novel techniques for that opponent type when compared to the SMA baseline.

Given this, it is likely that vision grids did help the agent towards a more general avoidance policy, not one specific to a single opponent. Furthermore, their combination with the goal distance performed well. This indicates that vision grids can be combined with other global representations of information in order to provide more robust behaviour that is aware of the local dynamics of the environment while chasing a specific goal.

The mean final rewards were similar between OMTs for both opponent types despite significant differences in PP and FPP. This indicates that the problem itself does not seem to need a very accurate opponent modelling technique. This could be because the Monte-Carlo rollouts act as a correction, so even if the PP is relatively low, the 5 rollouts per action usually chosen by the hyperparameter optimisation (see Appendix A.1) can mitigate that and still choose good actions. Opponent modelling is still needed, however. The agent was initially tested both without Monte-Carlo rollouts and with a new opponent MLP for each level (PP around 25-30%) and performed considerably worse.

In addition, the opponent modelling methods seem to be unstable, with large variance in how many opponents are reported (see Table A.1). This is also probably due to low statistical power, as well as randomness in the agent's early exploratory behaviour. CPD is more stable than LDC and reports less opponents, which is consistent with the idea that it groups opponents together due to its history queue setup.

While the game designed in this thesis is somewhat simple, it is still representative of the different opponents problem and how different actions taken in the same state cannot be modelled successfully with only one opponent model. It would therefore be interesting to see how the novel opponent modelling techniques perform on more complex tasks with longer episodes that strongly depend on good opponent modelling. Furthermore, further research into the applications of the novel MLP comparison techniques outside opponent modelling could also be useful. They could serve as a means to quickly switch between a small number of MLPs in real-world, unpredictable tasks, where full recognition setups are impractical due to lack of data or a variable number of items to model.

References

Bellman, R. (1957). A Markovian decision process. *Journal of mathematics and mechanics*, *6*(5), 679–684.

Bonferroni, C. (1936). Teoria statistica delle classi e calcolo delle probabilita. *Pubblicazioni del R Istituto Superiore di Scienze Economiche e Commericiali di Firenze*, *8*, 3–62.

Brun, R., & Rademakers, F. (1997). ROOT—An object oriented data analysis framework. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, *389*(1-2), 81–86.

Eadie, W. T., Drijard, D., & James, F. E. (1971). Statistical methods in experimental physics. *Amsterdam: North-Holland*.

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (pp. 249–256).

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Knegt, S. J., Drugan, M. M., & Wiering, M. A. (2018a). Learning from Monte Carlo rollouts with opponent models for playing Tron. In *International Conference on Agents and Artificial Intelligence* (pp. 105–129).

Knegt, S. J., Drugan, M. M., & Wiering, M. A. (2018b). Opponent Modelling in the Game of Tron using Reinforcement Learning. In *ICAART (2)* (pp. 29–40).

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

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... others (2015). Human-level control through deep reinforcement learning. *nature*, *518*(7540), 529–533.

Nachar, N., et al. (2008). The Mann-Whitney U: A test for assessing whether two independent samples come from the same distribution. *Tutorials in quantitative Methods for Psychology*, *4*(1), 13–20.

Pettitt, A. N. (1979). A non-parametric approach to the change-point problem. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, *28*(2), 126–135.

Pohlert, T., Pohlert, M. T., & Kendall, S. (2016). Package 'trend'. *Title Non-Parametric Trend Tests and Change-Point Detection*.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1985). *Learning internal representations by error propagation* (Tech. Rep.). California Univ San Diego La Jolla Inst for Cognitive Science.

Shantia, A., Begue, E., & Wiering, M. (2011). Connectionist reinforcement learning for intelligent unit micro management in starcraft. In *The 2011 international joint conference on neural networks* (pp. 1794–1801).

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Tesauro, G., & Galperin, G. (1996). On-line policy improvement using Monte-Carlo search. *Advances in Neural Information Processing Systems*, *9*, 1068–1074.

Van Hasselt, H., Guez, A., & Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 30).

Verzani, J. (2004). *Using R for introductory statistics*. Chapman and Hall/CRC.

Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine learning*, *8*(3-4), 279–292.

A Appendix

A.1 Hyperparameters

The hyperparameters optimised for the complex game can be found below. For each hyperparameter, there is a range of values that was used in grid-searches for each component as described in Subsection 5.2. The bold numbers are the initial values. For each value the combination of learning algorithm, opponent modelling technique and type of opponent that uses it is shown in parentheses with an abbreviation that uses the first letter of each of their names. For example, Sarsa with Loss distribution comparison and the Deterministic opponent is listed as SLD.

A.1.1 Opponent modelling

$minHistorySize$ ∈ {**8** (SLD, SLR, SCR, QLR, DLR), 10 (SCD, QLD, QCD, DLD, DCD, QCR, DCR)}
$maxHistorySize$ ∈ {20 (SLD, QLD, QLR, DLR, DCR), **30** (QCD, SLR), 50 (DLD, DCD, QCR), 100 (SCD, SCR)}
$pValueThreshold$ ∈ {0.025 (SLD, QLD, DLD, SLR, SCR), **0.05** (SCD, QCR, DLR, DCR), 0.1 (QCD, DCD, QLR)}

A.1.2 Learning algorithm

ϵ_{SAR} ∈ {0.3 (SSD, SLD, SCD), **0.5** (SSR, SLR, SCR)}
$\epsilon_{QL,DQL}$ ∈ {**0.5** (QSD, QLD, QCD, DLD, QSR, QLR, QCR, DSR, DLR, DCR), 0.75 (DSD, DCD)}
γ ∈ {**0.9** (QLD, QCD, DCD), 0.95 (SSD, SLD, SCD, QSD, DSD, DLD, SSR, SLR, SCR, QSR, QLR, QCR, DSR, DLR, DCR), 0.99}

A.1.3 Experience replay

C ∈ {100 (QLD, DLD, QSR, QCR, DLR), **1000** (DSD, QSD, QCD, DCD, QLR, DCR, DSR)}
miniBatchSize ∈ {8 (DSD, QSR, QLR, QCR, DLR, DCR, DSR), **16** (QSD, QLD, QCD, DLD, DCD)}
historySize ∈ {10000 (DSD, QSR, QLR, DSR), **100000** (QSD, QLD, QCD, DLD, DCD, QCR, DLR, DCR)}

A.1.4 Function approximator MLP

visionGridArea ∈ {9, **25** (SSD, SLD, SCD, QSD, QLD, QCD, DSD, DLD, DCD, SSR, SLR, SCR, QSR, QLR, QCR, DSR, DLR, DCR)}
α ∈ {0.0001, **0.0005** (SLD, SCD, QSD, QLD, QCD, DLD, DCD, SLR), 0.001 (SSD, DSD, SSR, SCR, QSR, QLR, QCR, DSR, DLR, DCR)}

A.1.5 Opponent MLP

visionGridArea ∈ {9 (QSD, DSD), **25** (SSD, SLD, SCD, QLD, QCD, DLD, DCD, SSR, SLR, SCR, QSR, QLR, QCR, DSR, DLR, DCR)}
α ∈ {0.0001 (QSR), 0.0005 (QSD, SSR, DSR), **0.001** (SSD, SLD, SCD, QLD, QCD, DSD, DLD, DCD, SLR, SSR, QLR, QCR, DLR, DCR)}

A.1.6 Monte-Carlo Rollouts

nrSteps ∈ {**1** (SSD, SLD, SCD, QSD, QLD, QCD, DSD, DLD, DCD, SSR, SLR, SCR, QSR, QLR, QCR, DSR, DLR, DCR), 3, 5}
nrRollouts ∈ {1 (DCD), 3, **5** (SSD, SLD, SCD, QSD, QLD, QCD, DSD, DLD, SSR, SLR, SCR, QSR, QLR, QCR, DSR, DLR, DCR)}

A.1.7 Simple Game Opponent Modelling

$pValueThreshold$ = 0.1 for LDC, 0.2 for CPD
$minHistorySize$ = 8 for LDC, 10 for CPD
$maxHistorySize$ = 30 for both LDC and CPD

A.2 Predicted number of opponents

Table A.1: Means and standard deviations of reported number of opponents over all 8 agents for each algorithm combination and opponent type. "Simple" denotes the Simple game where only Sarsa (SAR) was used.

	deterministic	**random**
LDC Simple	4.875 ± 0.83	-
CPD Simple	2.5 ± 0.53	-
SAR&LDC	4 ± 0.53	4.625 ± 0.74
SAR&CPD	2 ± 0	1 ± 0
QL&LDC	4.75 ± 0.7	7.875 ± 1.64
QL&CPD	3.25 ± 0.46	3 ± 0
DQL&LDC	5.875 ± 0.99	5.625 ± 1.06
DQL&CPD	3 ± 0	5.25 ± 0.46

Table B.1: Results of Mann-Whitney U tests performed in order to see differences in PP and FPP for the Simple game and deterministic opponent type. The tests were conducted against a Bonferroni-adjusted alpha level of 0.0125 (0.05/4). NS denotes that the test was not significant. (>) denotes that the algorithm on the left in the row label has a greater mean for the result tested than the one on the right. (<) denotes the opposite.

	PP (%)	**FPP (%)**
SMA, CPD	U=1, p<0.001, (<)	U=0, p<0.001, (<)
CPD, LDC	U=0, p<0.001, (<)	U=0, p<0.001, (<)

Table B.2: Results of Mann-Whitney U tests performed to see differences in mean final reward, PP and FPP for the Complex game and both opponent types. See Table 6.2 for the mean final rewards. The algorithm combination with the best mean final reward, listed as first in the row labels, was compared with the same learning algorithm, but a change in opponent modelling technique. It was also compared with the best performing algorithm combination in terms of mean final reward for the other two learning algorithms. The tests were conducted against a Bonferroni-adjusted alpha level of 0.0041 (0.05/12). The table below is for the deterministic opponent type.

	rewards	**PP (%)**	**FPP (%)**
SAR&SMA, SAR&LDC	U=50, p=0.06, NS	U=38, p=0.57, NS	U=0, p<0.001, (<)
SAR&SMA, SAR&CPD	U=54, p=0.02, NS	U=64, p<0.001, (>)	U=0, p<0.001, (<)
SAR&SMA, QL&CPD	U=63, p<0.001, (>)	U=64, p<0.001, (>)	U=0, p<0.001, (<)
SAR&SMA, DQL&LDC	U=62, p<0.001, (>)	U=4, p=0.0018, (<)	U=0, p<0.001, (<)

Table B.3: The same motivation and type of tests as Table B.2 for the random opponent type.

	rewards	**PP (%)**	**FPP (%)**
SAR&CPD, SAR&SMA	U=31, p=0.95, NS	U=0, p<0.001, (<)	U=18, p=0.16, NS
SAR&CPD, SAR&LDC	U=49, p=0.08, NS	U=64, p<0.001, (>)	U=64, p<0.001, (>)
SAR&CPD, QL&CPD	U=63, p<0.001, (>)	U=58, p=0.004, (>)	U=27, p=0.64, NS
SAR&CPD, DQL&CPD	U=61, p=0.001, (>)	U=64, p<0.001, (>)	U=64, p<0.001, (>)

B Appendices

B.1 Statistical tests

Exploratory plots reveal that the distributions for the mean final rewards, opponent move prediction percentages (PP) and PPs after opponent model was found (FPP) are either normal or have a left skew. This is because the maximum reward achievable has an upper bound, and smaller PP and FPP values are easier to reach than larger ones. Therefore, the Mann-Whitney U test (Nachar et al., 2008) was used for the comparisons, since it does not require the normal distribution. Furthermore, to correct for multiple comparisons, Bonferroni corrections (Bonferroni, 1936) were used for the statistical tests. See Table B.1 for the Simple game tests, Table B.2 for the Complex game with deterministic opponent type tests and Table B.3 for the Complex game with random opponent type tests. The context and procedure for the tests are present in the captions of those tables.

B.2 Source code

The simulation, algorithms, scripts and random seeds used to produce the results reside at https://github.com/raduacosma/BachelorProject. Since the random seeds were fixed and are mentioned in the source code, the results are deterministic and should be reproducible.