# Assignment 4 for APML2022

**Group 9**
Christian Acosta | Josh Bleijenberg | Mischa van Ek | Sjoerd Vink | Robin Wiersma

## Abstract

This paper examines various reinforcement learning algorithms used for finding a particular goal in an maze environment. The algorithms Q-learning, SARSA and Expected SARSA are compared in finding the goal in the environment as efficient as possible. These algorithms make part of the temporal difference learning paradigm that involves an agent operating in an environment with state & rewards (inputs) and actions (outputs). The Expected SARSA and Q-learning algorithms are the best performing algorithms in the maze environment. Both algorithms return highly similar performance and obtain the maximum reward in fewer episodes than the SARSA algorithm.

## 1 Introduction

In 1997, the then world chess player champion was defeated for the first time by a chess computer under standard time controls. This chess computer uses a learning technique called Reinforcement Learning. Reinforcement learning is a computational approach to understanding and automating goal-directed learning and decision making. Reinforcement learning distinguishes itself by other computational approaches such as supervised learning by its emphasis on learning by an agent from direct interaction with its environment, without requiring exemplary supervision. For this it used the formal framework of Markov Decision Processes to define the interaction between a learning agent and its environment in terms for states, actions and rewards [SB18]. In this paper, 3 different algorithms are compared by the task of finding the end goal in a given environment maze as efficient as possible. The goal for this research is to find the most efficient algorithm, in which the agent finds the goal in the maze in as in as few steps as possible.

The following chapter of this paper describes the Markov Decision Process, its components and the environment mazes. This, together with more information about the algorithms used is discussed in chapter three. The fourth chapter gives an overview of the performance of the different algorithms, followed by an evaluation and a discussion. Finally, the conclusion of the study is discussed in the last chapter.

## 2 Data

The setup that was used to calculate the data is detailed in figure 1. The figure shows a model of the Markov Decision Process (MDP) and its components: states, actions, rewards and transition. The first key concept to understand when exploring the MDP is to answer the question of what the environment is. Figure 2a shows the environment. The environment is a 16 x 16 grid representing the shape of a maze. Inside the maze is an agent whose goal is to find the target. The environment consists of x,y positions with a total of 256 possible states. A state is the observations that the agent receives from the environment about where it is. The state would change as soon as the agent performs an action $a$. The agent can choose an action to move up, down, right or left of its current position $s$. In the MDP, the agent is rewarded for choosing the "right" action, in this case pursuing a state in which the agent achieves the goal. Once the agent reaches the goal, the environment is reset, ending an episode. An episode is any state that comes between an initial state and a state where the agent reaches the final goal.

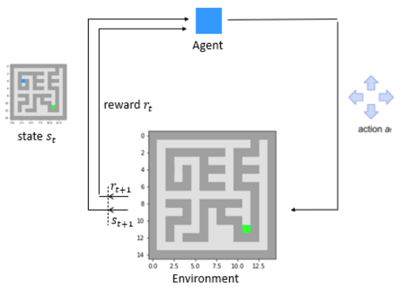
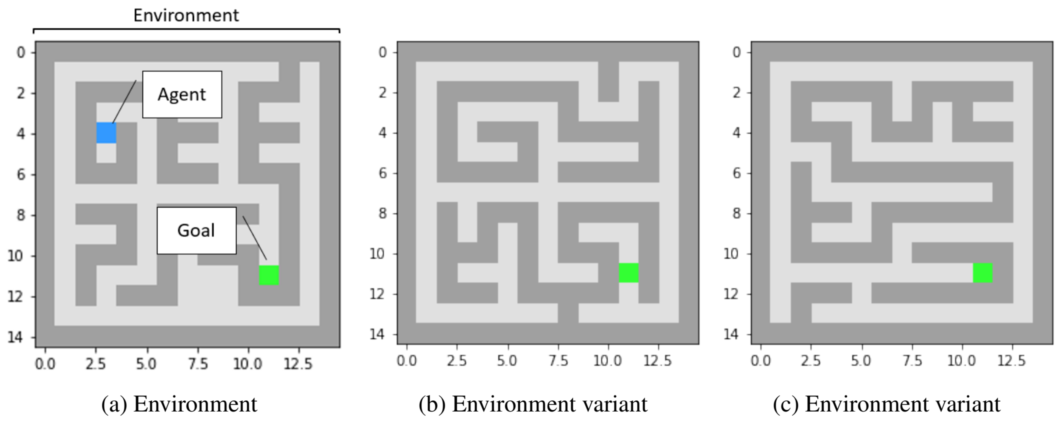


Figure 1: Interaction flow

The purpose of the MDP is to allow the agent to dynamically adapt to new environments. Broadly speaking, the agent must learn a policy that tells the agent what actions to take in each state. "A policy defines the way the learning agent behaves at any given time." [SB18]. Iterative methods to find the optimal solution for the MDP are further discussed in Chapter 3.



(a) Environment (b) Environment variant (c) Environment variant

Figure 2: Environments

## 3 Methods

Temporal difference learning (TD) is a specific sub-paradigm of reinforcement learning. TD can be used in specific problem-sets where the environment dynamics are unknown for the agent. This causes an extra challenge for the agent due to the unknown state-action functions. In order to fully understand the dynamics of this problem, it is important to first understand the context. After the context explanation, the different algorithms are discussed.

### 3.1 Exploration vs exploitation

In order for the agent to develop an optimal policy, it needs to make a trade-off between exploitation and exploration. The goal of an agent is to take actions that maximise the long-term reward. However, the agent doesn't know which actions maximise the reward because environment dynamics are unknown. To discover these actions, it must try actions that it has never selected before. So there is a consideration between exploring new actions or exploiting rewards from familiar actions. This is where action selection strategies come into play.

### 3.2 Action selection strategy

Action selection is the strategy where the agent bases its selection of actions on. The most basic strategy is the greedy strategy, which always goes for the highest reward. In other words, it always exploits the action with the highest estimated reward. However, chances are that this action selection strategy overlooks possible better actions.

Another strategy is $\epsilon$-greedy. $\epsilon$-greedy is a strategy that balances exploration and exploitation based on the epsilon parameter. Instead of always selecting the action with the best estimated value from the Q-table, it gives a small probability (the epsilon) to select possible actions uniformly and randomly. This makes sure the agent doesn't overlook actions that potentially result in higher rewards and maximizes long-term reward without compromising possible better actions.

### 3.3 On-policy and off-policy

The algorithms used in this paper make use of different policy strategies, namely on-policy and off-policy. On-policy agents use the same action selection strategy for the behavior policy and the target policy. Off-policy agents use a different action selection strategy for the behavior policy and target policy. This will get more clear in the explanation of algorithms.

### 3.4 Algorithms

All algorithms make part of the temporal difference learning paradigm that involves an agent operating in an environment with states & rewards (inputs) and actions (outputs). The agent learns about the environment by relying on and updating Q-values in a Q-table. A Q-value indicates the quality of a particular action $a$ in a given state $s : Q_{(s,a)}$. Q-values are current estimates of the sum of future rewards. The difference between the algorithms is the way they calculate the Q-value. Every algorithm has the following parameters:

- Epsilon ($\epsilon$): a value between 0 and 1 that controls the chance that the agent chooses an exploration step
- Learning rate ($\alpha$): a value between 0 and 1 and controls how the Q-value gets updated
- Discount factor ($\gamma$): a value between 0 and 1 that controls the amount that the future rewards are discounted, which makes future rewards less important than the immediate rewards

#### 3.4.1 Q Learning

Q-learning calculates the Q-value of a given action based on the direct reward from the action and the maximum achievable reward from the next state. Q-learning is an off policy method. The reason why Q-learning is off-policy is because it uses different strategies for the behavior policy and strategy policy. It uses an $\epsilon$-greedy strategy for the first step and a greedy strategy (maximum reward) to calculate the maximum achievable reward from the next state $\max_a Q(s_{t+1} \cdot a)$. The equation used to calculate the new Q-value is as follows:

$$Q^{new}(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \cdot \Big(r_t + \gamma \cdot \max_a Q(s_{t+1} \cdot a) - Q(s_t, a_t)\Big) \tag{1}$$

Q-learning can be very fast as it directly learns the best strategy. It tries to maximize the reward with a greedy strategy from the next state, reducing the chance on taking a potential bad action because of the epsilon exploration chance.

#### 3.4.2 SARSA

SARSA is a learning method that stands for 'State, Action, Reward, State, Action'. As the name suggests, it calculates the Q-value of an action based on the current state and action, and the next state and action assuming that it will keep following the policy. SARSA is an on-policy method, because it uses the $\epsilon$-greedy strategy for both the behavior policy and the target policy. The equation used to calculate the new Q-value is as follows:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)) \tag{2}$$

SARSA learns a slightly less optimal policy because it uses an $\epsilon$-greedy strategy from the next state. So, there is a chance that the agent doesn't select the optimal policy from there because it wants to explore. This is in contrary to Q-learning, which uses the greedy strategy from the next state. So, the only difference from Q-Learning is that it replaces the maximization of Q-Learning by expectation over the probability distribution of actions by any exploration policy. It eventually will converge to a solution that is optimal under the assumption that we keep following the same policy that was used to generate the experience. It computes the Q-value according to a certain policy and updates the Q-value in the Q-table. Then the agent follows that policy.

### 3.4.3 Expected SARSA

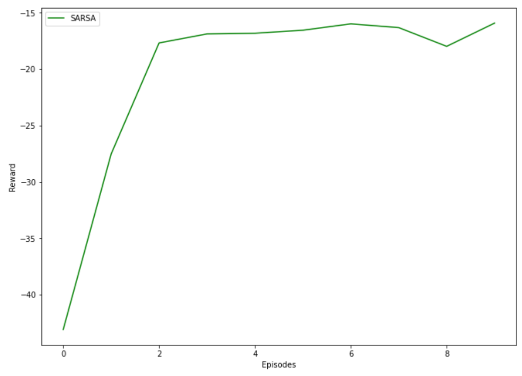
Expected SARSA calculates the Q-value of an action based on the reward of the action and the expected reward from the next state. The expected reward from the next state takes the chance that the agent selects a greedy or non-greedy action into consideration. Consider the learning algorithm that is just like Q-learning except that instead of the maximum over next state-action pairs it used the expected value. With this in mind, expected SARSA can be seen as a on-policy version of Q-learning. The equation used to calculate the new Q-value is as follows:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma E_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t)) \tag{3}$$

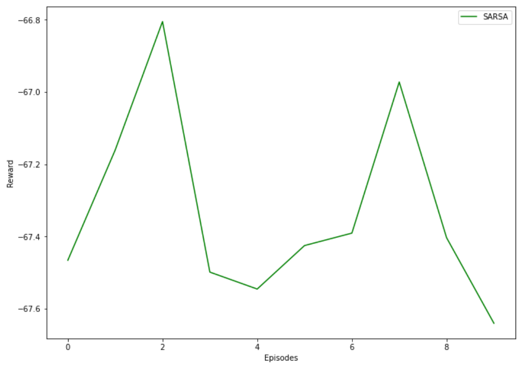
Notice that Expected SARSA takes the weighted sum of all possible next actions with respect to the probability of taking that action. If the expected return is greedy with in mind the expected return, then Expected SARSA is similar to Q-learning. Otherwise, Expected SARSA is an on-policy method and computes the expected return for all actions, instead of randomly selecting an action like SARSA.

## 4 Evaluation

For the SARSA algorithm, the $\epsilon$ parameter is a key parameter. This parameter specifies the probability that a random action will be selected, where a higher epsilon signifies a higher probability. Additionally, the exploration versus exploitation trade-off is influenced by the $\epsilon$ parameter. A higher $\epsilon$ value allows for more exploitation, while a lower $\epsilon$ value allows for more exploration. After experimentation with different values, we found that an $\epsilon$ value of 0.01 is an optimal balance between exploration and exploitation, returning the best results. We observed that higher $\epsilon$ values greatly diminish the performance of the SARSA algorithm. In figure 3, we provide a comparison of two different $\epsilon$ values, with each value on one extreme of the possible value range. Please note that figure 3b has a different scale for the y-axis. It can be observed that a much larger reward is obtained with $\epsilon$ set to 0.1 versus 0.9.



(a) $\epsilon$ = 0.1



(b) $\epsilon$ = 0.9

Figure 3: Influence of $\epsilon$ values on SARSA

After further fine-tuning, we decided to use an $\epsilon$ of 0.01 for our further analyses. This $\epsilon$ value returned the highest reward for the SARSA algorithm, as can be seen in figure 4.

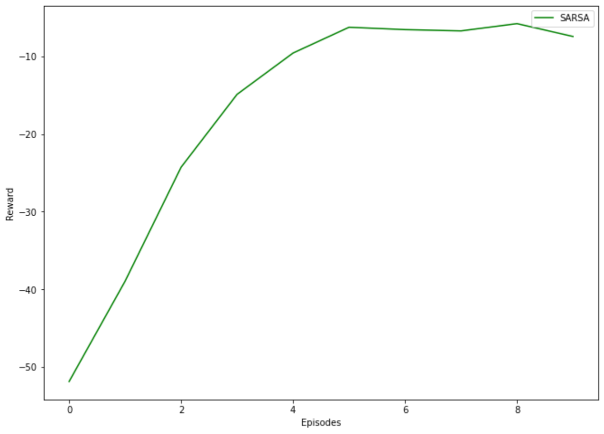
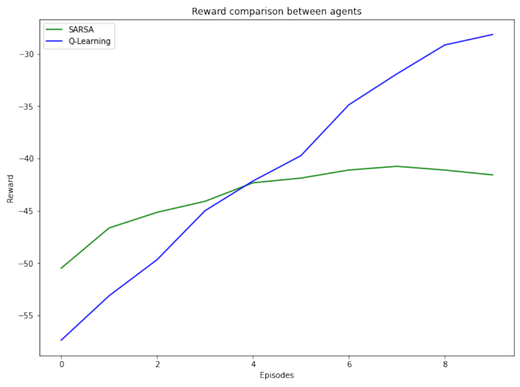
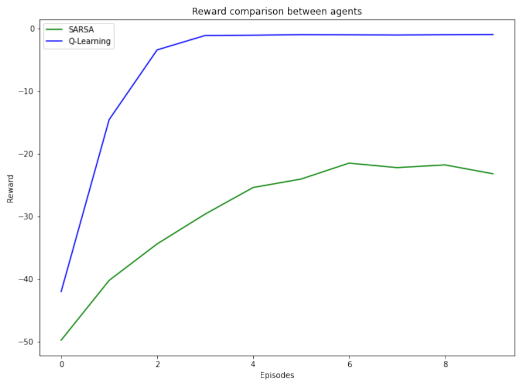


Figure 4: $\epsilon = 0.01$

When comparing the differences between the SARSA and Q-learning algorithms, another parameter enters the picture. The parameter in question is the discount factor ($\gamma$). The discount factor specifies how much an agent 'cares' about rewards in the distant future compared to rewards in the immediate future. A $\gamma$ of 0 means that an agent will only learn from actions that produce an immediate reward, where as a $\gamma$ of 1 means that an agent will learn an action based on the sum total of all of its future rewards. Thus, too low of a $\gamma$ makes an agent short-sighted. Increasing the $\gamma$ ensures an agent takes future rewards into consideration. This ensures a more advantageous long-term reward. Figure 5 shows the influence of higher $\gamma$ values on the algorithms. The Q-learning agent takes much longer to reach the maximum reward, which is also lower than for higher $\gamma$ values. It can also be noted that it starts at a lower point than the SARSA agent, but eventually surpasses the SARSA agent at about half-way through the episode range. The SARSA agent performs noticeably worse here than with higher $\gamma$ values. With $\gamma$ set to 0.5, the Q-learning agent performs much better. It reaches peak-reward much quicker, with the peak itself also being a greater reward. The SARSA agent also improves, but is still outpaced by the Q-learning agent. We eventually settled on a $\gamma$ of 0.9. In figure 6, the Q-learning agent reaches peak reward even quicker than before. The SARSA agent shows significant improvement here, reaching its peak much quicker and coming very close to the Q-learning agent at the end. These visualization have shown a clear difference between the off-policy (Q-learning) and on-policy (SARSA) algorithms. SARSA takes more conservative and 'safer' choices, which results in it taking longer to converge.



(a) $\gamma = 0.1$



(b) $\gamma = 0.5$

Figure 5: Comparison of $\gamma$ values

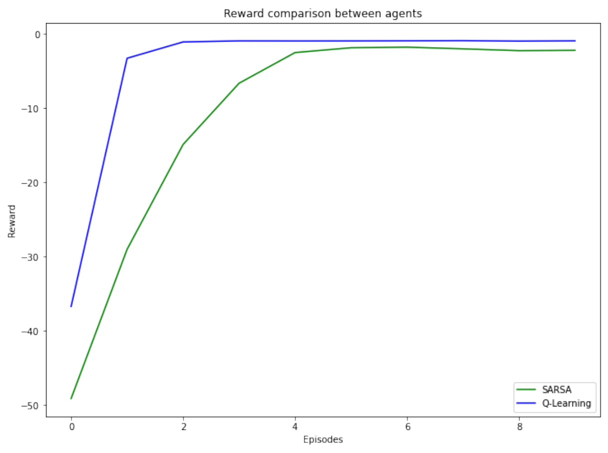


Figure 6: $\gamma = 0.9$

For the quantified evaluation, we have decided to run every algorithm 20 times, with 1000 episodes for each time. The results are saved, averaged, and plotted to a line graph. Using the average from multiple runs allows the results to be more representative and able to be generalized.

For figure 7 and figure 8, each step on the x-axis represents the average of 100 episodes. The original data is condensed to improve the readability of the graphs.

To set a baseline for our algorithms, we use the Random agent. Unfortunately, we were not able to visually represent the performance of the this algorithm and compare it to the others, as its (obviously) random behavior returns very bad results.

Expected SARSA and Q-learning are the best performing algorithms. Both algorithms return highly similar performance that is noticeably better than the SARSA algorithm. This can be seen in figure 7 and figure 8, where the lines for expected SARSA (Red, labelled as ESARSA) and Q-learning (Blue) are almost completely overlapping. For figure 7, it can also be noted that expected SARSA and Q-learning obtain the maximum reward in fewer episodes than SARSA, roughly 200 episodes versus 400 episodes respectively. Additionally, it can be noted that the reward for SARSA drops slightly between episodes 500 and 800 before returning to the maximum value. Around episode 800, yet again, the reward for SARSA drops slightly till the 1000th episode. This behavior is different than that of the other two algorithms, which see their rewards stay constant once the maximum has been reached. Nevertheless, the reward obtained at the end of the 1000th episode is lower, but still close to that of the two better performing algorithms.

Figure 8 shows the amount of steps in each episode. It can be noted that Q-learning and expected SARSA reach the lowest steps per episode more quickly than SARSA, at about episode 200, and stay more or less constant from that point on. The SARSA algorithm shows similar behavior as in figure 3. Moderate fluctuations until the 1000th episode, where it settles at a slightly higher (worse) value than the two other algorithms.

To visualize the brain or 'thought-process' of each algorithm, we have provided figure 9, figure 10, and figure 11. In each figure, two maze-like visualizations are shown. The visualization on the left represent the state value at each position. It can be noted that the reward increases towards the maximum as the agent approaches the goal. This can be observed visually as a gradient approaching the white color. For these specific examples, we can deduce that the goal is towards the bottom-right of the maze, more specifically in the gap between the two 'whitest' blocks. The visualization on the right shows which action the agent chooses at each position.

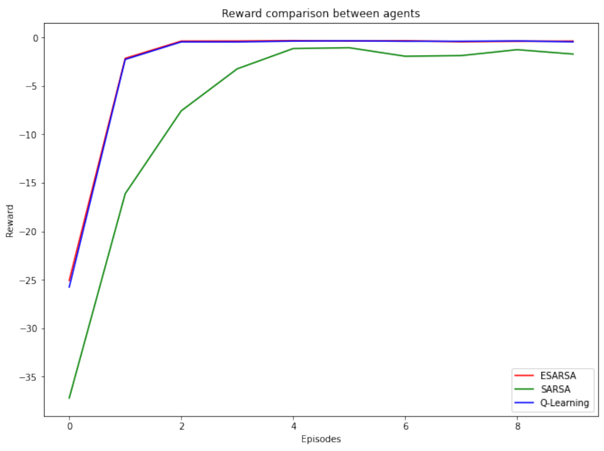


Figure 7: Reward comparison between agents

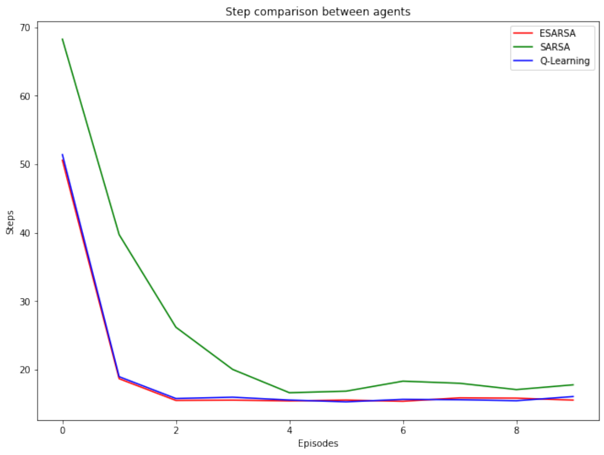


Figure 8: Step comparison between agents

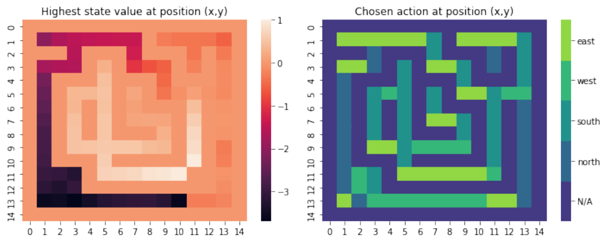


Figure 9: Visualization of the Q-learning brain

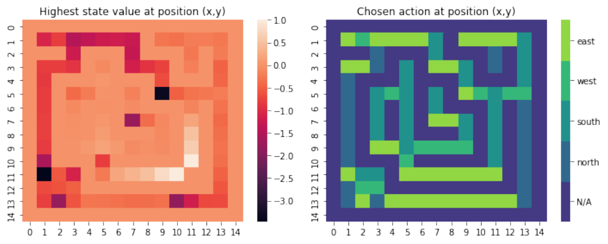


Figure 10: Visualization of the SARSA brain

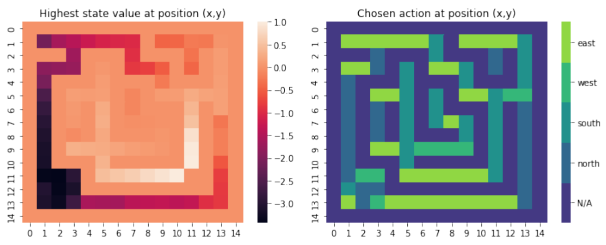


Figure 11: Visualization of the expected SARSA brain

## 5 Discussion

The input parameters are essential to find the best performing agent. There is experimented with several parameter sets. Human judgement was used to determine the optimal parameters for the agents. A disadvantage of this methodology is that it does not compare a large variety of parameter choices.

A quantified evaluation would have been more appropriate for finding the optimal parameters, as it compares more precisely with a wide evaluation range. To do this, a range has to be set for each parameter and all the possible combinations between these parameters have to be tested. The average reward can be used as a metric here. However, lots of computational power is required for such code to run. This is why human judgement is used on a select number of parameter sets.

Despite the lack of a more precise evaluation method for the parameter settings, the eventual evaluation was made very precise and generic. As said in the evaluation, this is done by running all three agents 20 times for 1000 episodes and averaging the result over the 20 agent runs. This makes that a very precise evaluation could be made, making up for the less precise parameter evaluation.

The findings correspond well to the trade-off between on-policy and off-policy. As explained in the algorithms section, SARSA takes longer to converge to an optimal solution in regards to Q-Learning. This is because Q-learning directly learns the optimal policy and SARSA uses the $\epsilon$-greedy strategy for both behavior and target policy. ESARSA decreases that difference and performs more or less similar to Q-learning.

## 6 Conclusions

The goal of this assignment was to use different reinforcement learning algorithms to solve a 'Robot in a maze' problem. Three different algorithms were tested based on both on-policy and off-policy. The expectation was that Q-learning was the best performer because it learns for the optimal policy. This corresponded to the results, where Q-learning and expected SARSA performed best with minimal difference. SARSA took longer to converge to the optimal policy. However, when the optimal policy is reached the performance difference is minimal in regard to the other algorithms.

### 6.1 Future work

Reinforcement learning is one of the most exiting fields in artificial intelligence as it has many applications in the real world. It would for example be very interesting for logistically focused companies to have a reinforcement learning system in place that finds the most optimal path in a warehouse. This can be both for robotic agents and for human agents who work there. Production capacity will increase with the same amount of work capacity. This way, profit margins will be more optimal. In order to accomplish such a system, there needs to be a research about how to model an environment and what algorithm works optimally.

## 7 Contributions

### 7.1 Contributions of group members

| **Assignment 4 - Reinforcement learning** | **Team members** |
|---|---|
| Task 2.2: Implement the Agent | |
| SARSA | Sjoerd Vink |
| Q LEARNING | Mischa van Ek & Sjoerd Vink |
| Task 3: Play with parameters | |
| 3.1 Evaluation | Christian Acosta & Sjoerd Vink |
| Bonus Tasks | Josh Bleijenberg |

| **Report** | **Team members** |
|---|---|
| Abstract | Robin Wiersma & Josh Bleijenberg |
| 1. Introduction | Robin Wiersma & Josh Bleijenberg |
| 2. Data | Mischa van Ek |
| 3. Methods | |
| 3.4.1 Q learning | Mischa van Ek |
| 3.4.2 SARSA | Sjoerd Vink |
| 3.4.3 Expected SARSA | Josh bleijenberg |
| 4. Evaluation | Christian Acosta |
| 5. Discussion | Sjoerd Vink & Mischa van Ek |
| 6. Conclusion | Christian Acosta |
| 6.1 Future work | Sjoerd Vink |

### 7.2 Group reflection

This assignment proved to be the most challenging we have had for our course. We are happy to conclude that the results show that we have achieved the desired goal of creating an agent that can dynamically adapt to its environment. At the beginning of the assignment we did run into problems working in a collaborative environment. The package needed for the assignment turned out not to install properly in the Google Collab environment. Fortunately, the problem was quickly resolved by the teacher's assistant, Olusanmi. We encountered some other hiccups, for example, the agent did not save the rewards when the goal was reached. But this was also handled by the teacher's assistant who visualised the problem by using the agentbrain function. After finding the solution, we were able to complete the task successfully.

## References

[SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.