Assignment 2 - Tabular RL

Advanced Topics in Machine Learning - DAT440

Lisa Samuelsson & Dylan Osolian

Group 19

1. Describe the algorithms

Briefly describe the Q-learning, Double Q-learning, SARSA and Expected SARSA algorithm and how each of them work.

Q-learning

Q-learning is an off-policy TD control algorithm which is used for solving MDPs (Markov Decision Processes). It is model-free, which means that it learns from trial-and-error (as opposed to a model-based method that uses planning), and does not need a model of the environment. In the algorithm, a so-called Q-table is created, and this table corresponds to a state-action pair (s, a). This table can be initialized randomly or with arbitrary values. The agent interacts with the environment by observing the current state, taking an action and then receiving a reward. After that, it moves to the next state. When the agent transitions to this new state, the Q-value of the previous state-action pair is updated using the observed reward and the maximum Q-value of the next state. This process is repeated until convergence of the Q-table or for a predetermined number of times.

Double Q-learning

An issue with Q-learning is maximization bias. We could have a state with several actions that have a true value of zero, but with estimated values that are uncertain, and thus distributed with some higher than zero and some below zero. We then might be prone to select a suboptimal action due to overestimation. To mitigate this issue, avoiding maximization bias, double Q-learning can be used. In this algorithm, we now have two Q-tables instead of a single one. With a probability of 0.5 we update the first Q-table, otherwise we do a symmetric update to the second Q-table. One of the tables is used to select the best action, and the other one is used to estimate its value. We select the action to take from summing the two Q-tables.

SARSA

SARSA is an on-policy algorithm, and similar to the Q-learning algorithm, we also have a Q-table. However, as it is an on-policy algorithm, the action-value function $q_\pi(s, a)$ is continuously estimated for the policy π, the states s and actions a. At the same time, the policy π is changed with regards to the Q-table. With for example an epsilon-greedy policy SARSA always converges to an optimal policy and Q-table under certain conditions: that the state-action space is finite, that all state-action pairs are visited an infinite number of times, and that the exploration rate decreases over time. SARSA stands for State-Action-Reward-State-Action, which describes the steps in the learning process.

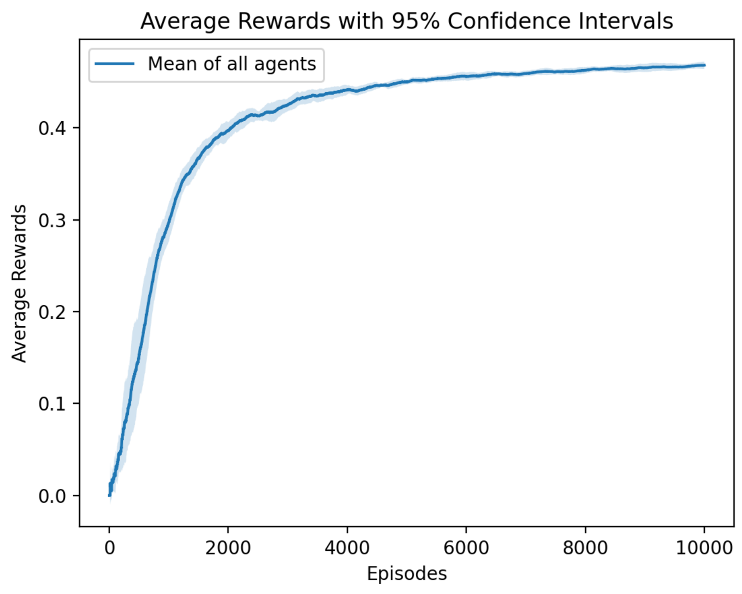
Expected SARSA

Expected SARSA is an algorithm that moves deterministically in the same way as SARSA moves in expectation. The algorithm is similar to the Q-learning algorithm, but it uses the expected value by considering the likelihood of every action instead of selecting the maximum as in Q-learning. Expected SARSA is heavier in computation than SARSA but also performs slightly better.

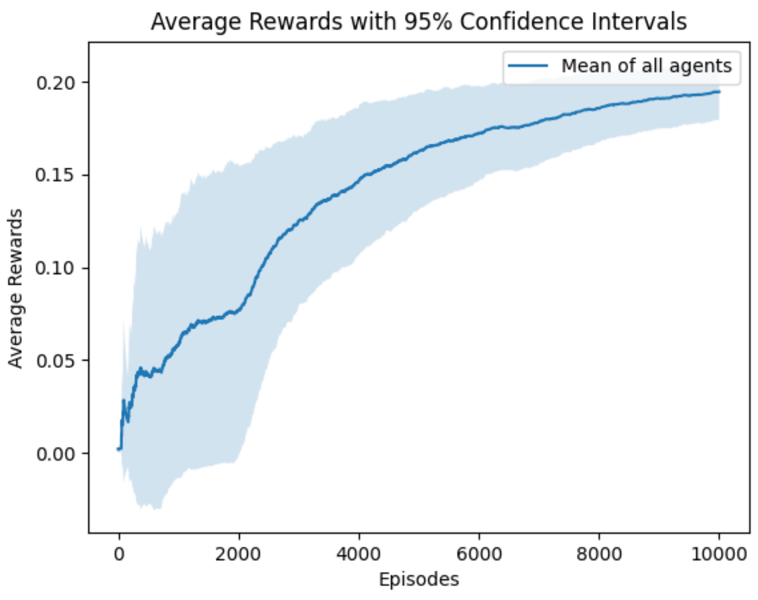
2. Run and plot rewards

Implement Q-learning, Double Q-learning and SARSA agents with 5% epsilon greedy policy (95% greedy action) and gamma=0.95. For every agent, run experiments on both RiverSwim (included locally in this repository) and FrozenLake-v1. Plot the moving average of the rewards while your agent learns, either by episode or step depending on the environment, averaged over 5 runs (i.e., restarting the training 5 times). Include error-bars (or something similar) indicating the 95% confidence intervals calculated from your variance of the 5 runs.

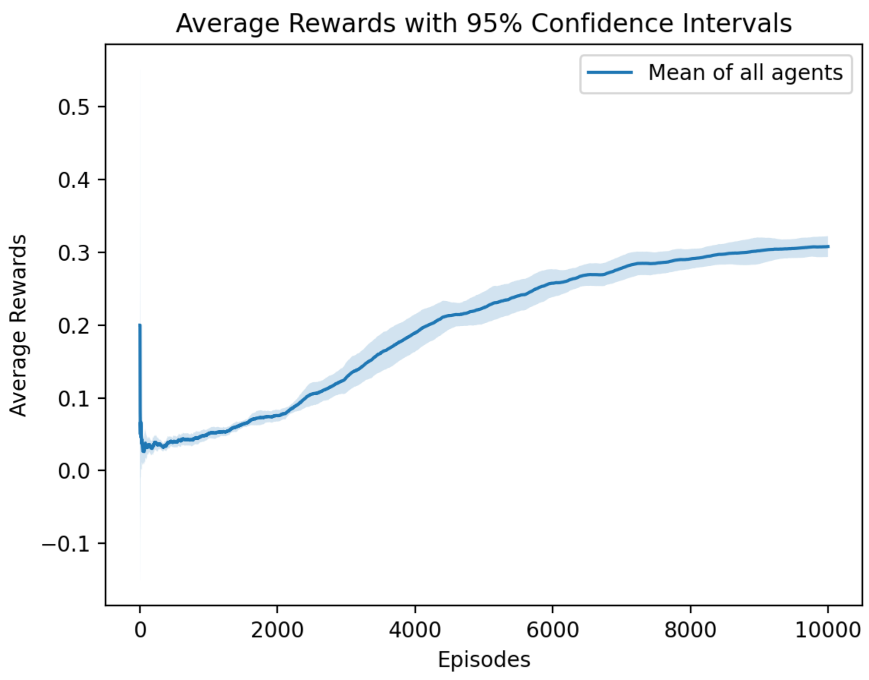
Q-learning (Frozen Lake)



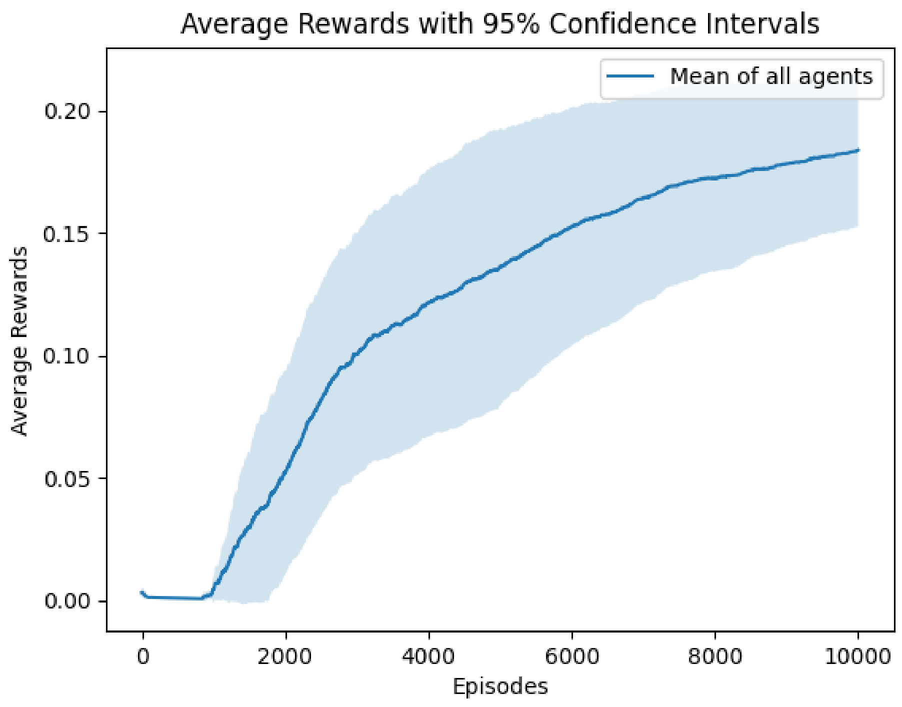
Q-learning (Riverswim)



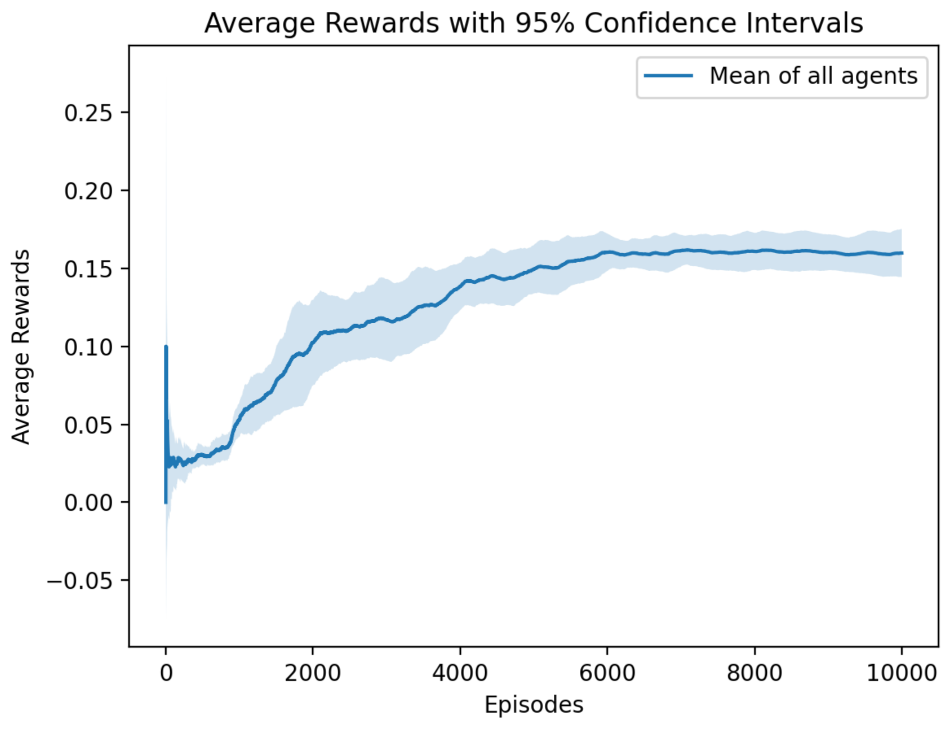
Double Q-learning (Frozen Lake)



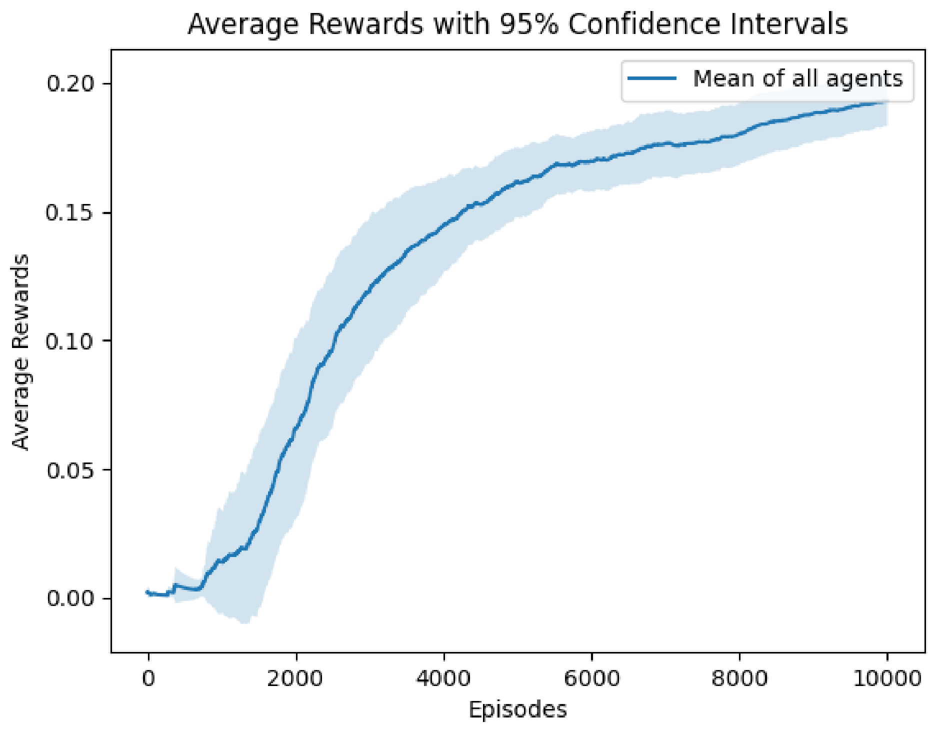
Double Q-learning (Riverswim)



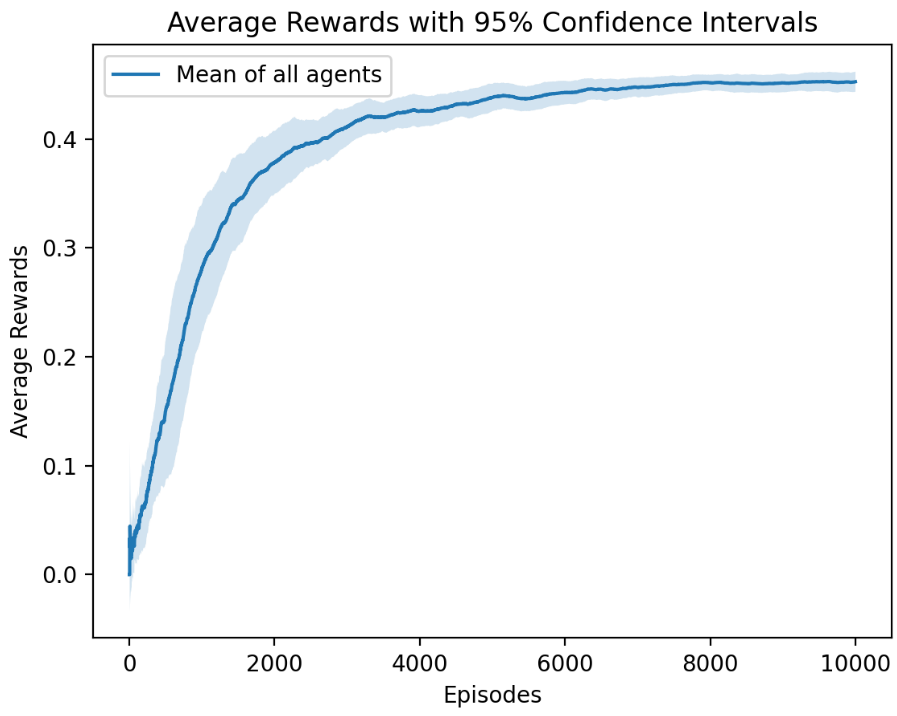
SARSA (Frozen Lake)



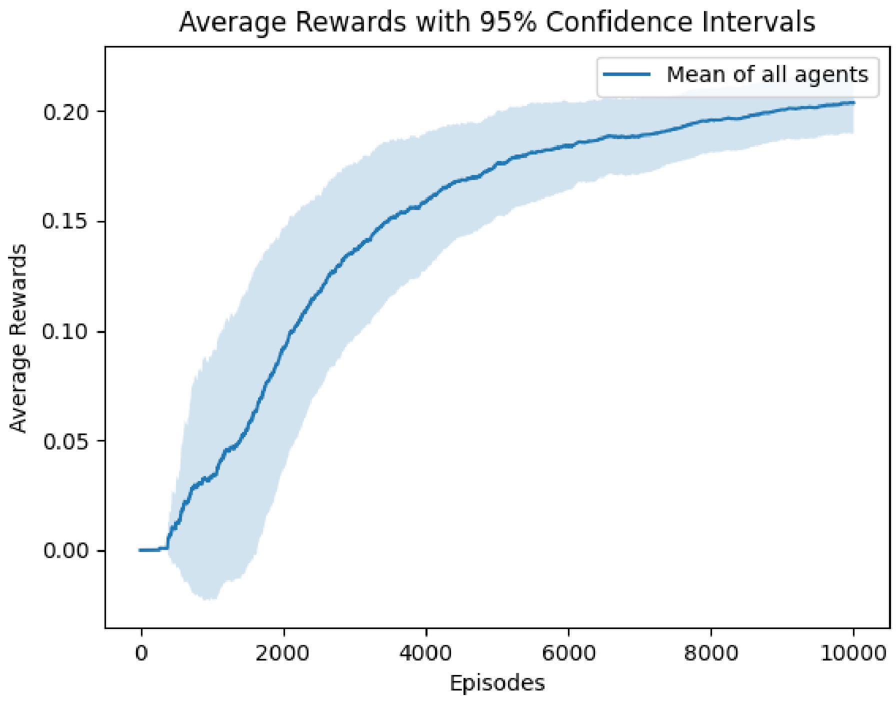
SARSA (Riverswim)



Expected SARSA (Frozen Lake)



Expected SARSA (Riverswim)



3. Visualize Q-values and greedy policy

Visualize the Q-values for each state (make sure it is easy to interpret, i.e., not just a table). Draw/visualize the greedy policy obtained from the Q-values. Does it reach the optimal policy? Briefly describe and motivate the optimal policy.

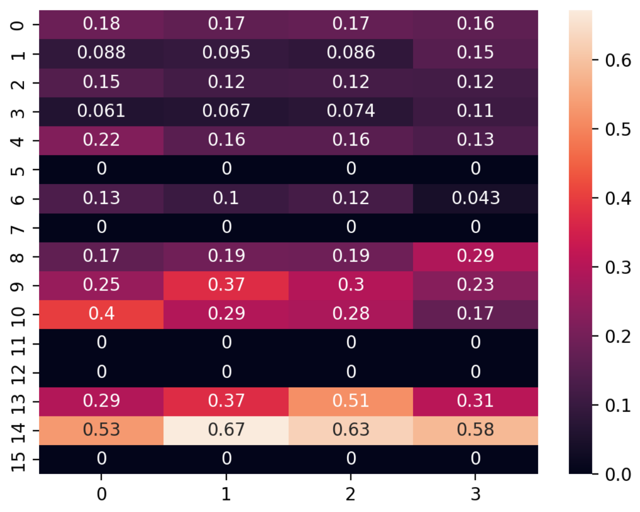
In this section, we are visualizing the Q-values for each state, and we have chosen to do this using heat maps. On the y-axis, each state is represented, and the actions are represented on the x-axis. A higher value (and a brighter color) represents a higher Q-value, and a lower value (and darker color) represents a lower Q-value. With this illustration, we can easily observe the action which has the highest Q-value for each state.

For the Frozen Lake environment, we have visualized the greedy policy obtained from the Q-values with a 4-4 grid, which resembles the frozen lake the agent walks over. In each square of the grid, we have drawn an arrow that shows the direction the agent will move in if following the greedy policy.

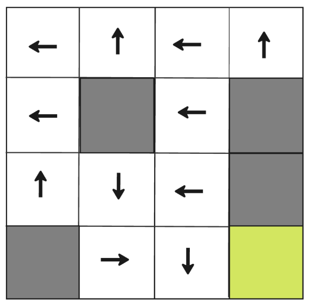
For the River Swim environment we have visualized the policy in a similar way. In this visualization an arrow to the left means action 0 is selected and an arrow to the right represents action 1. Note that action 1 doesn't necessarily mean that we move to the right in the chain.

Q-learning (Frozen Lake)

Visualization of Q-values (states on y-axis, action on x-axis)

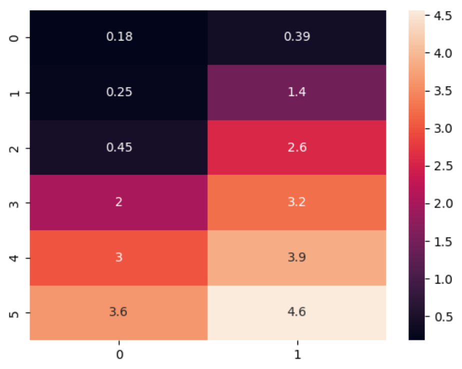


Visualization of greedy policy obtained from the Q-values

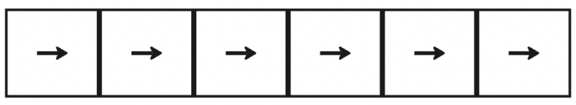


Q-learning (River Swim)

Visualization of Q-values (states on y-axis, action on x-axis)

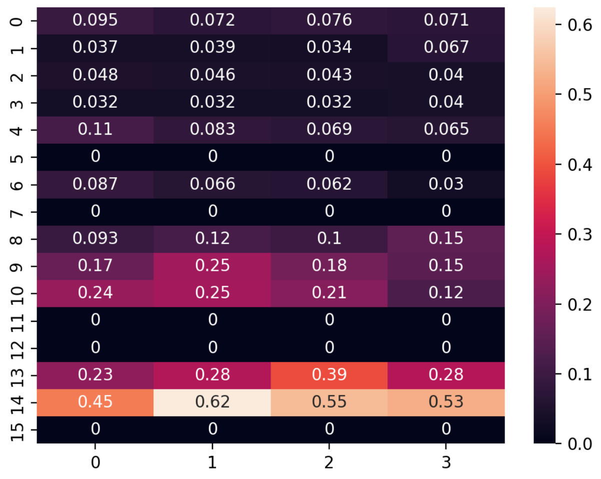


Visualization of greedy policy obtained from the Q-values

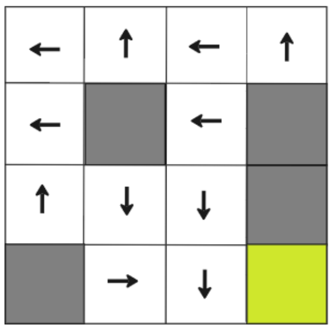


Double Q-learning (Frozen Lake)

Visualization of Q-values (states on y-axis, action on x-axis)

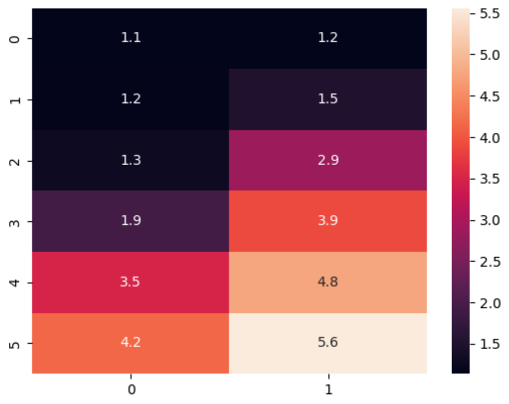


Visualization of greedy policy obtained from the Q-values

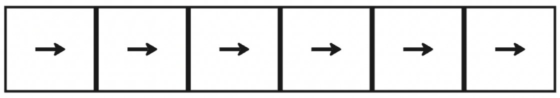


Double Q-learning (River Swim)

Visualization of Q-values (states on y-axis, action on x-axis)

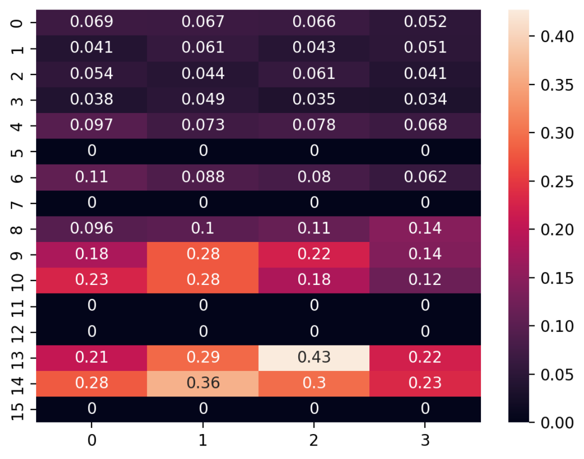


Visualization of greedy policy obtained from the Q-values

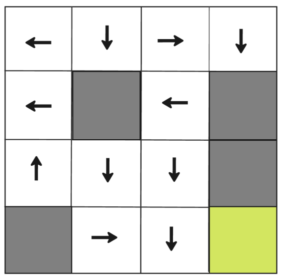


SARSA (Frozen Lake)

Visualization of Q-values (states on y-axis, action on x-axis)

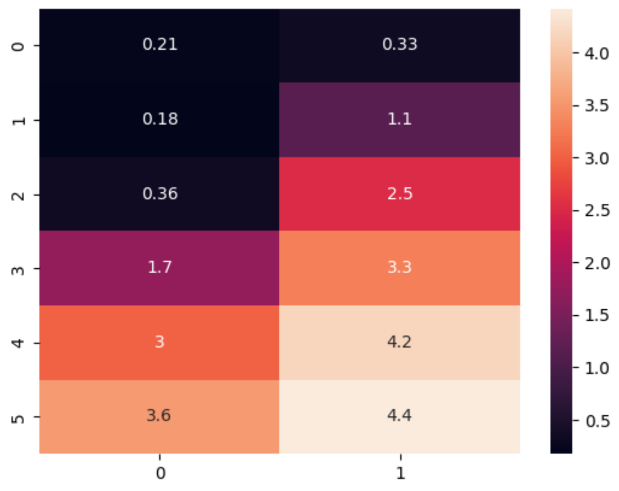


Visualization of greedy policy obtained from the Q-values

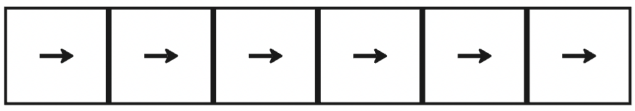


SARSA (River Swim)

Visualization of Q-values (states on y-axis, action on x-axis)

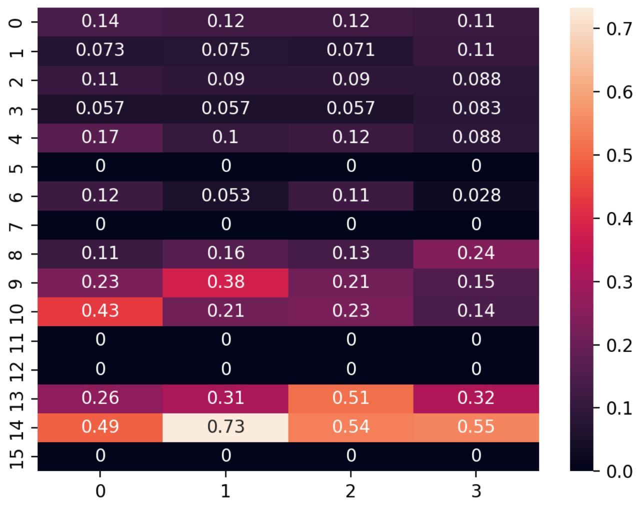


Visualization of greedy policy obtained from the Q-values



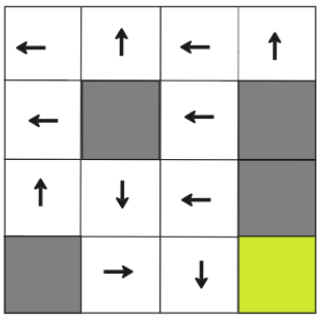
Expected SARSA (Frozen Lake)

Visualization of Q-values (states on y-axis, action on x-axis)



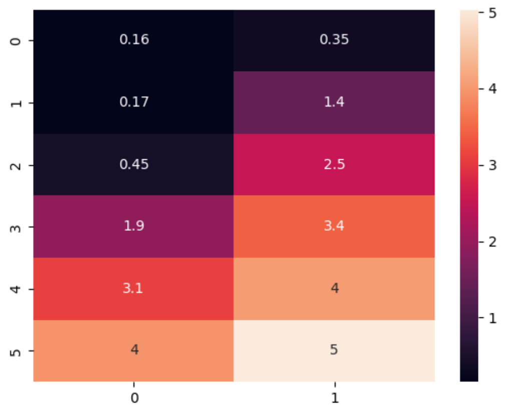
	0	1	2	3
0	0.14	0.12	0.12	0.11
1	0.073	0.075	0.071	0.11
2	0.11	0.09	0.09	0.088
3	0.057	0.057	0.057	0.083
4	0.17	0.1	0.12	0.088
5	0	0	0	0
6	0.12	0.053	0.11	0.028
7	0	0	0	0
8	0.11	0.16	0.13	0.24
9	0.23	0.38	0.21	0.15
10	0.43	0.21	0.23	0.14
11	0	0	0	0
12	0	0	0	0
13	0.26	0.31	0.51	0.32
14	0.49	0.73	0.54	0.55
15	0	0	0	0

Visualization of greedy policy obtained from the Q-values



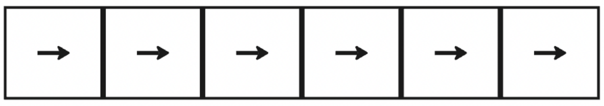
Expected SARSA (River Swim)

Visualization of Q-values (states on y-axis, action on x-axis)



	0	1
0	0.16	0.35
1	0.17	1.4
2	0.45	2.5
3	1.9	3.4
4	3.1	4
5	4	5

Visualization of greedy policy obtained from the Q-values



4. Initialization of Q-values

Investigate how the initialization of the Q-values affects how well the agent learns to solve RiverSwim and FrozenLake-v1. For instance, initializing Q-tables optimistically (larger than you expect the true table to be, not with zeroes). How does it affect the rewards during training and the greedy policy obtained from the Q-values? How does it affect the exploration? Clearly explain and display evidence supporting your arguments.

In *question 2*, we initialized the Q-values to random numbers between 0 and 1. For this investigation, we did four experiments, initializing the Q-values to

- random values between 1 and 5
- random values between 5 and 10
- random values between 10 and 20
- random values between 20 and 50.

We did this for the Q-learning algorithm, and for both the environments River Swim and Frozen Lake. We chose the same number of episodes for all our experiments, 10 000. But it would be interesting to also examine how the policies and convergence would act if we would run the algorithm for a larger number of episodes. However, we felt that this would extend the scope of the assignment a bit too much, as we already perform a rather large number of experiments.

For the Frozen Lake environment, we can see that the algorithm converges no matter what we initialize the Q-values to. However, for both environments, it seems as if the rewards are increasing slower when we initialize the Q-values higher. For the River Swim environment, when the Q-values are initialized to be above 10, the algorithm barely converges with the selected number of episodes (10 000). This is apparent both for the initialization with values between 10-20 and the initialization with values between 20-50. However, it is not unlikely that the algorithm would converge if we let it run for a larger number of episodes. We believe this as it the reward curve seems to be following the same pattern as when convergence occurred, but shifted a bit.

The greedy policy for frozen lake changes slightly when using more optimistic Q-values. It is hard to analyze the changes to the greedy policy since the frozen lake environment is slippery. However if we look at the Q-table after convergence we can see that there are only small changes to the Q-values, which affect the policy. The policy close to the terminal state remains the same for all our tests as the difference in Q-values between the different actions are higher here.

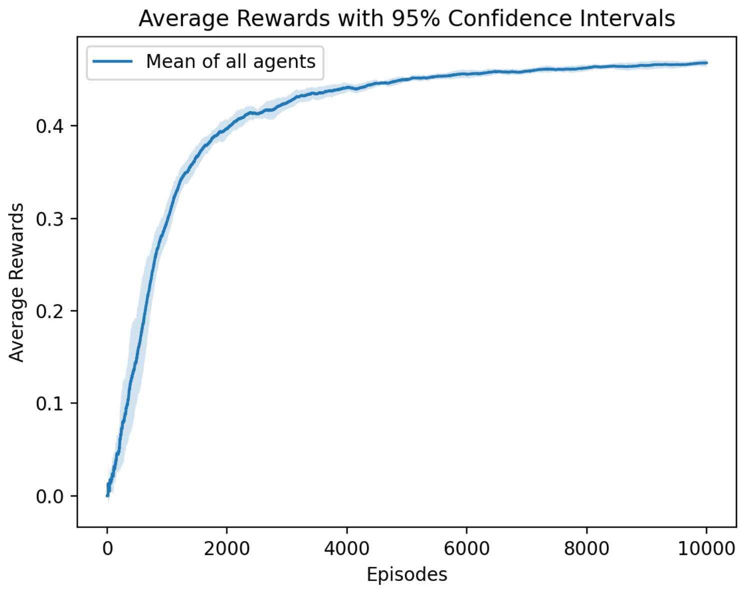
In the river swim environment the policy remains almost the same for all our tests, differing only slightly in the first state. In the end it makes sense that the greedy policy would be the same for all Q-table initializations as the algorithm converges to an optimal result.

Initializing the Q-values optimistically encourages exploration, especially in the beginning of the training. This is since with optimistic Q-values the agent will try one action and the reward will be lower than what the initialized value was. Therefore the agent will be “disappointed” and explore new actions since they have higher Q-values.

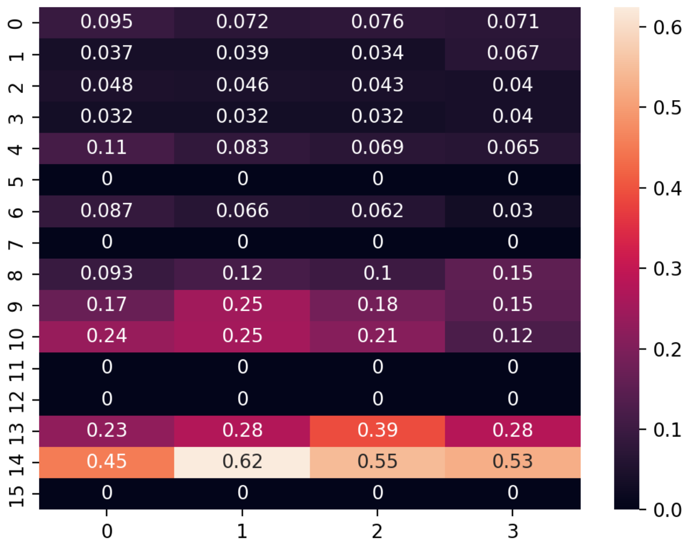
Frozen Lake

Q-values initialized between 0 and 1

Average rewards

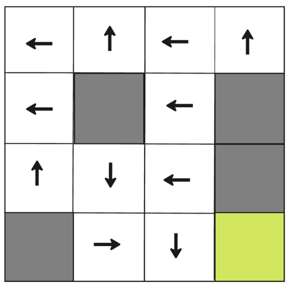


Visualization of Q-values (states on y-axis, action on x-axis)



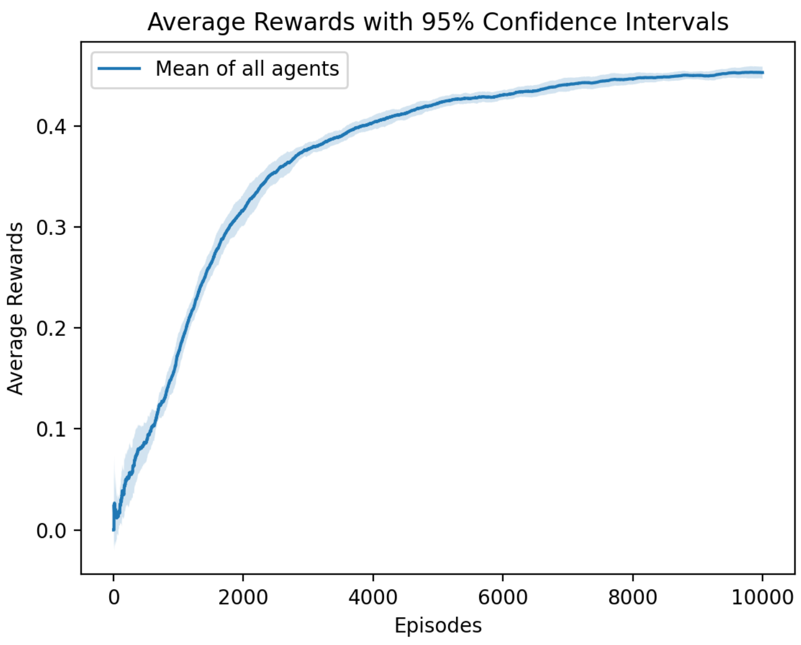
	0	1	2	3
0	0.095	0.072	0.076	0.071
1	0.037	0.039	0.034	0.067
2	0.048	0.046	0.043	0.04
3	0.032	0.032	0.032	0.04
4	0.11	0.083	0.069	0.065
5	0	0	0	0
6	0.087	0.066	0.062	0.03
7	0	0	0	0
8	0.093	0.12	0.1	0.15
9	0.17	0.25	0.18	0.15
10	0.24	0.25	0.21	0.12
11	0	0	0	0
12	0	0	0	0
13	0.23	0.28	0.39	0.28
14	0.45	0.62	0.55	0.53
15	0	0	0	0

Visualization of greedy policy obtained from the Q-values

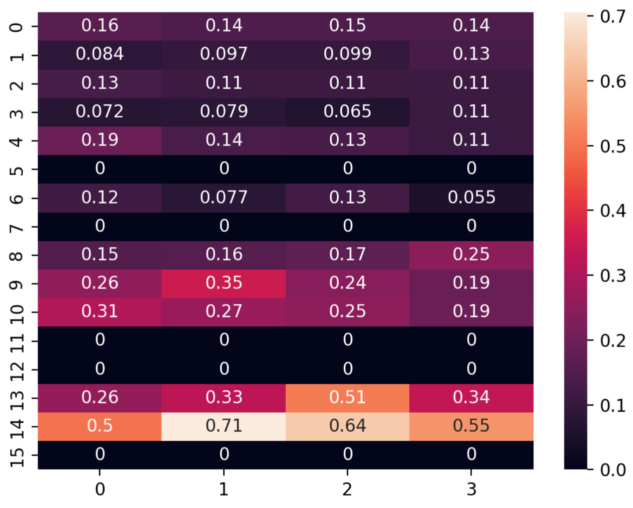


Q-values initialized between 1 and 5

Average rewards

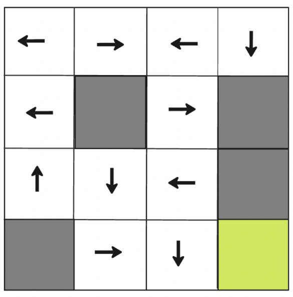


Visualization of Q-values (states on y-axis, action on x-axis)



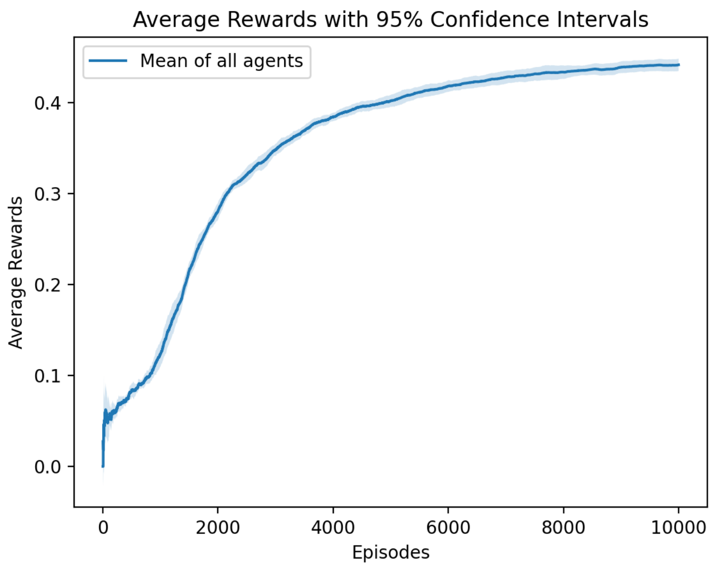
	0	1	2	3
0	0.16	0.14	0.15	0.14
1	0.084	0.097	0.099	0.13
2	0.13	0.11	0.11	0.11
3	0.072	0.079	0.065	0.11
4	0.19	0.14	0.13	0.11
5	0	0	0	0
6	0.12	0.077	0.13	0.055
7	0	0	0	0
8	0.15	0.16	0.17	0.25
9	0.26	0.35	0.24	0.19
10	0.31	0.27	0.25	0.19
11	0	0	0	0
12	0	0	0	0
13	0.26	0.33	0.51	0.34
14	0.5	0.71	0.64	0.55
15	0	0	0	0

Visualization of greedy policy obtained from the Q-values

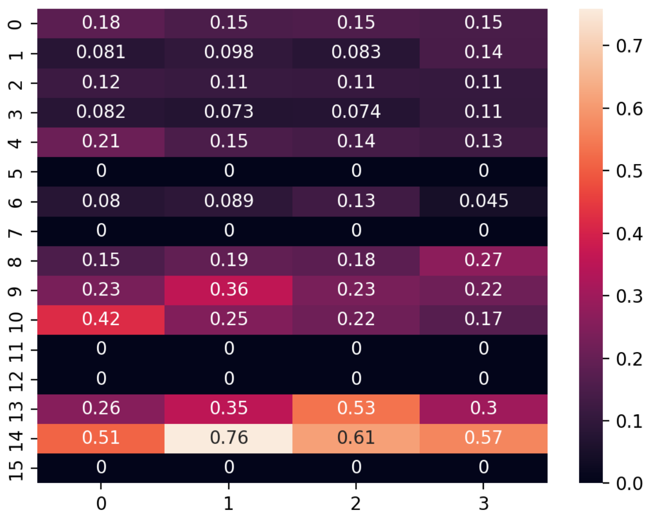


Q-values initialized between 5 and 10

Average rewards

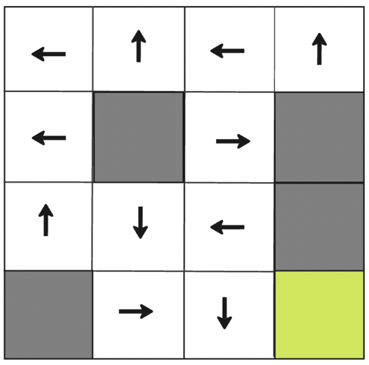


Visualization of Q-values (states on y-axis, action on x-axis)



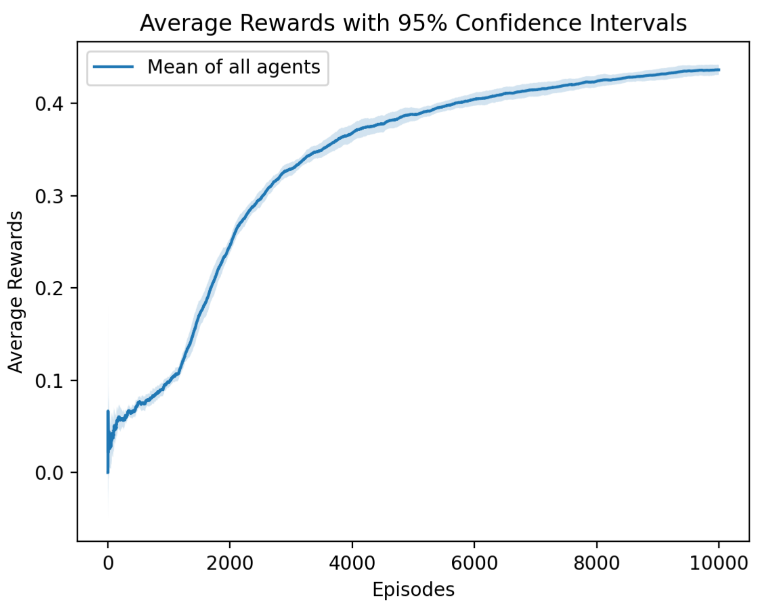
	0	1	2	3
0	0.18	0.15	0.15	0.15
1	0.081	0.098	0.083	0.14
2	0.12	0.11	0.11	0.11
3	0.082	0.073	0.074	0.11
4	0.21	0.15	0.14	0.13
5	0	0	0	0
6	0.08	0.089	0.13	0.045
7	0	0	0	0
8	0.15	0.19	0.18	0.27
9	0.23	0.36	0.23	0.22
10	0.42	0.25	0.22	0.17
11	0	0	0	0
12	0	0	0	0
13	0.26	0.35	0.53	0.3
14	0.51	0.76	0.61	0.57
15	0	0	0	0

Visualization of greedy policy obtained from the Q-values

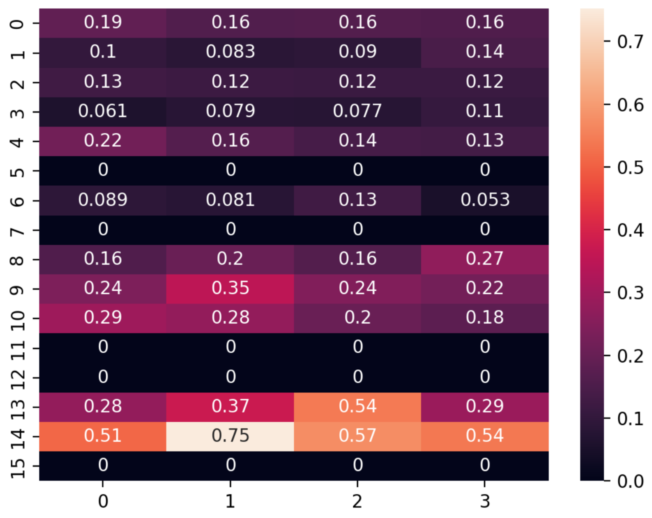


Q-values initialized between 10 and 20

Average rewards

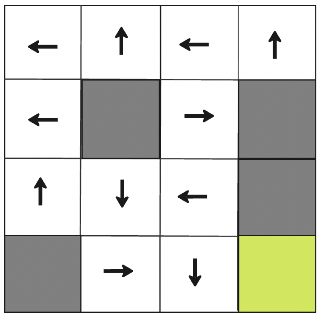


Visualization of Q-values (states on y-axis, action on x-axis)



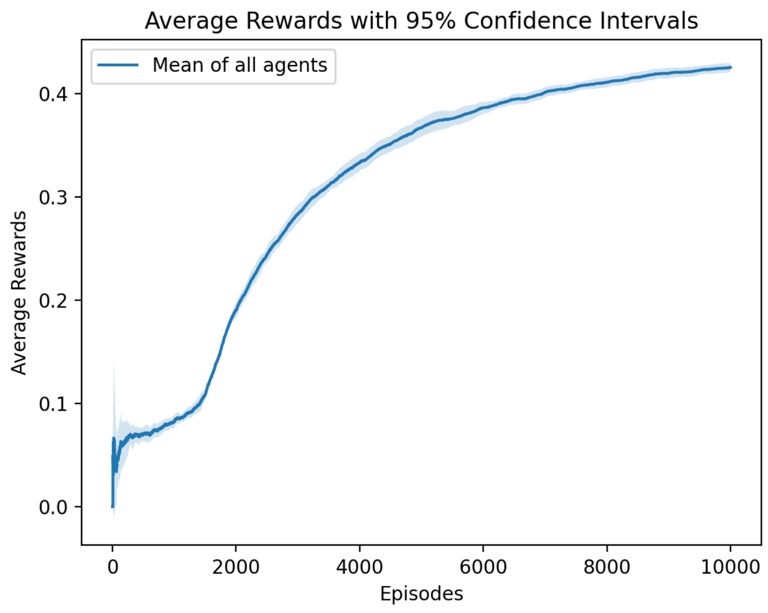
	0	1	2	3
0	0.19	0.16	0.16	0.16
1	0.1	0.083	0.09	0.14
2	0.13	0.12	0.12	0.12
3	0.061	0.079	0.077	0.11
4	0.22	0.16	0.14	0.13
5	0	0	0	0
6	0.089	0.081	0.13	0.053
7	0	0	0	0
8	0.16	0.2	0.16	0.27
9	0.24	0.35	0.24	0.22
10	0.29	0.28	0.2	0.18
11	0	0	0	0
12	0	0	0	0
13	0.28	0.37	0.54	0.29
14	0.51	0.75	0.57	0.54
15	0	0	0	0

Visualization of greedy policy obtained from the Q-values

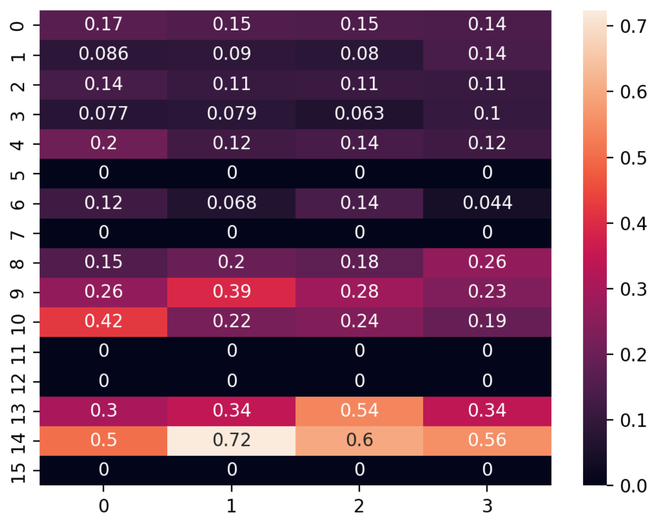


Q-values initialized between 20 and 50

Average rewards

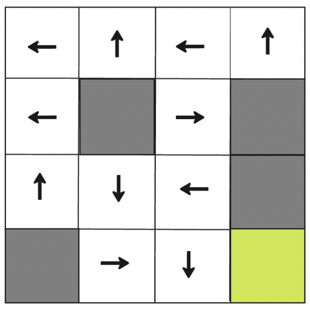


Visualization of Q-values (states on y-axis, action on x-axis)



	0	1	2	3
0	0.17	0.15	0.15	0.14
1	0.086	0.09	0.08	0.14
2	0.14	0.11	0.11	0.11
3	0.077	0.079	0.063	0.1
4	0.2	0.12	0.14	0.12
5	0	0	0	0
6	0.12	0.068	0.14	0.044
7	0	0	0	0
8	0.15	0.2	0.18	0.26
9	0.26	0.39	0.28	0.23
10	0.42	0.22	0.24	0.19
11	0	0	0	0
12	0	0	0	0
13	0.3	0.34	0.54	0.34
14	0.5	0.72	0.6	0.56
15	0	0	0	0

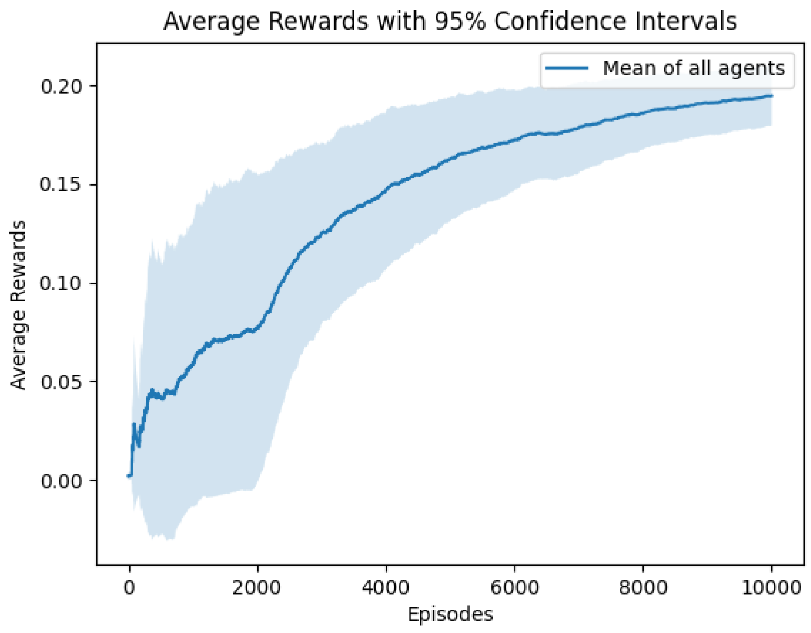
Visualization of greedy policy obtained from the Q-values



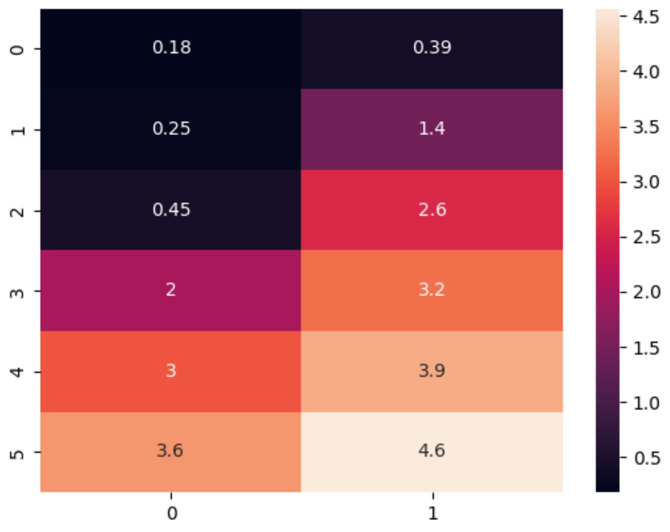
River Swim

Q-values initialized between 0 and 1

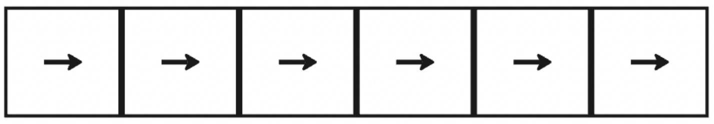
Average rewards



Visualization of Q-values (states on y-axis, action on x-axis)

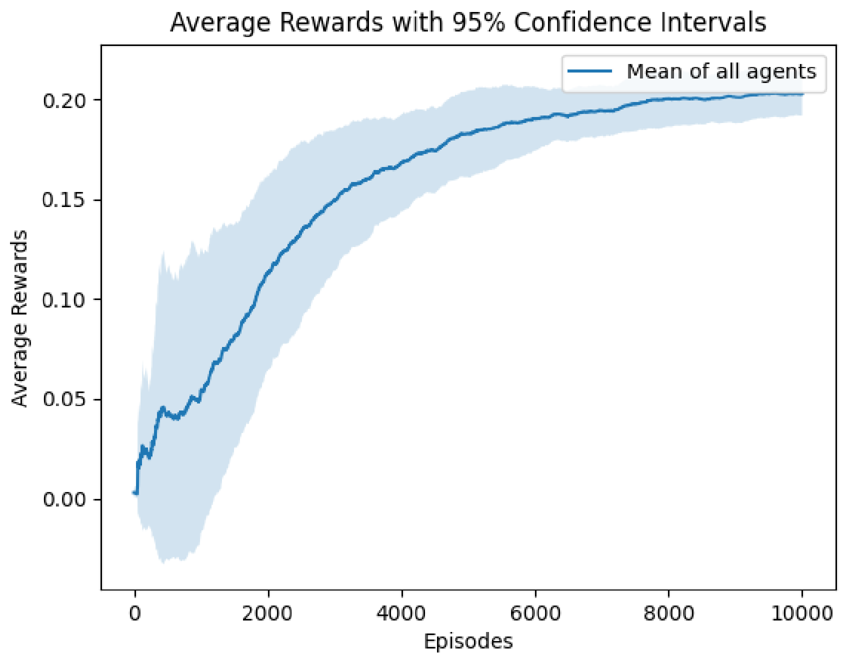


Visualization of greedy policy obtained from the Q-values

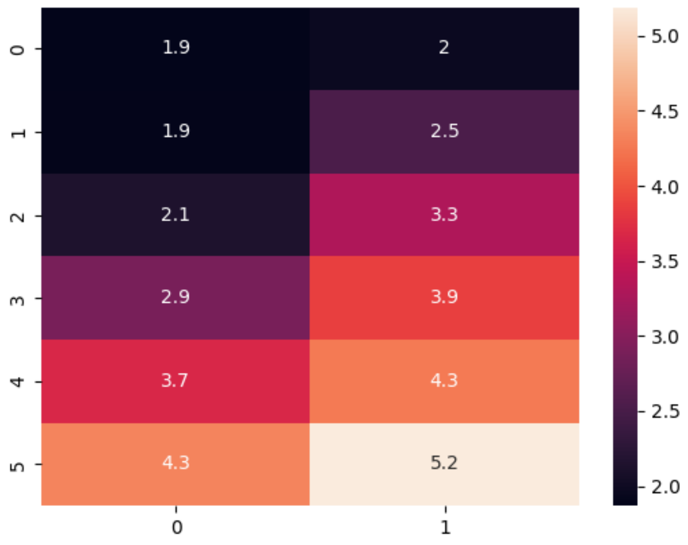


Q-values initialized between 1 and 5

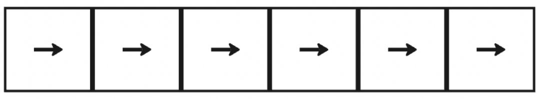
Average rewards



Visualization of Q-values (states on y-axis, action on x-axis)

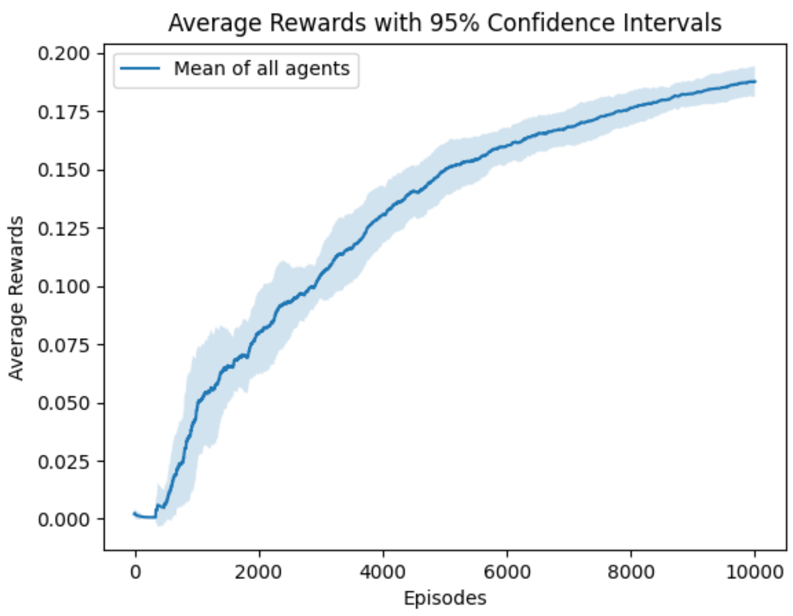


Visualization of greedy policy obtained from the Q-values

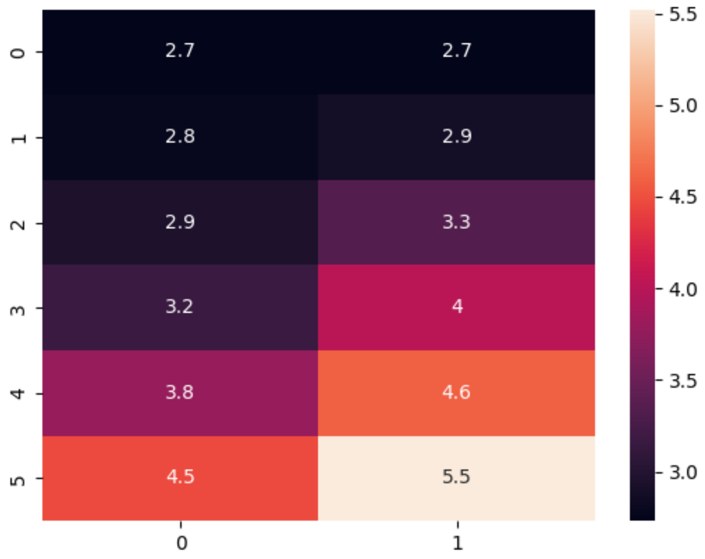


Q-values initialized between 5 and 10

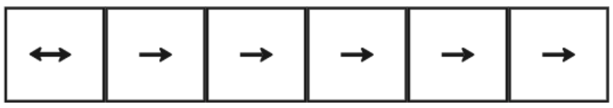
Average rewards



Visualization of Q-values (states on y-axis, action on x-axis)

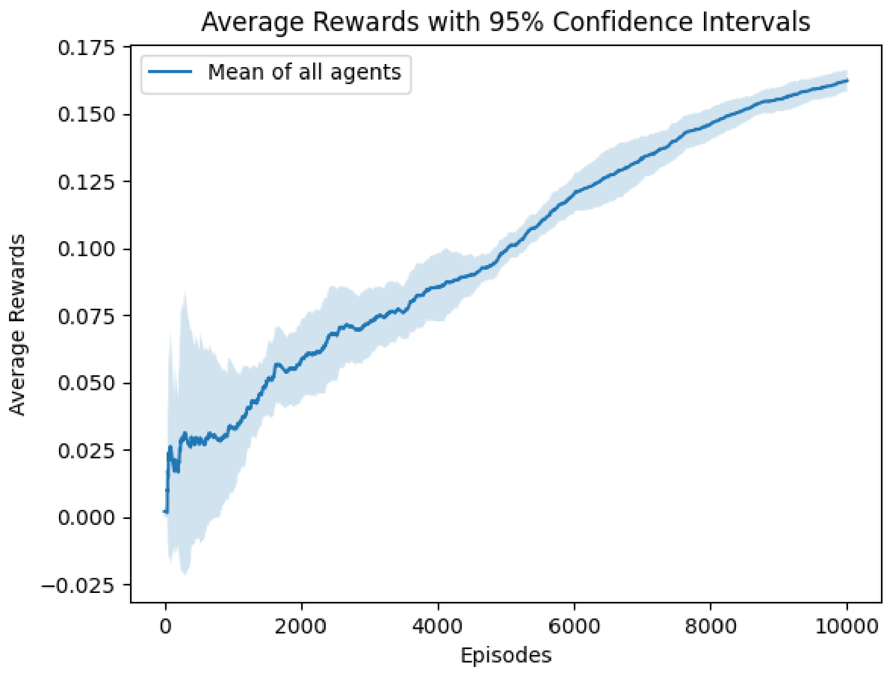


Visualization of greedy policy obtained from the Q-values

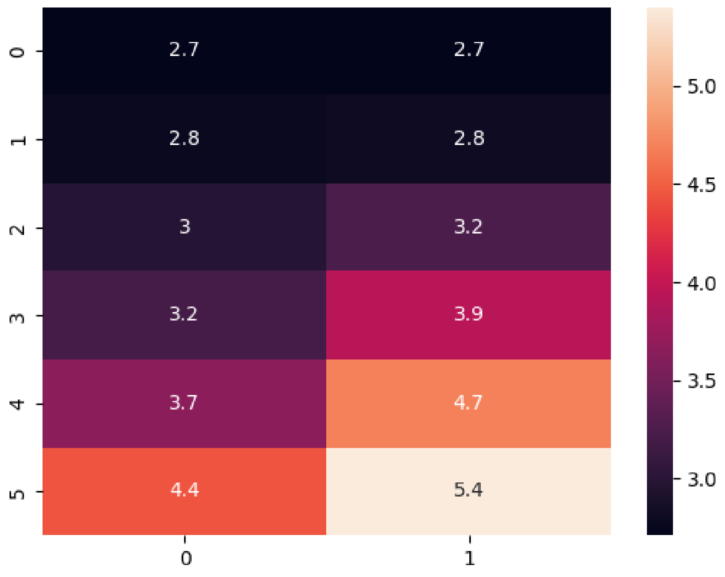


Q-values initialized between 10 and 20

Average rewards

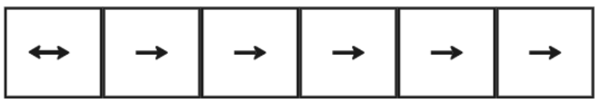


Visualization of Q-values (states on y-axis, action on x-axis)



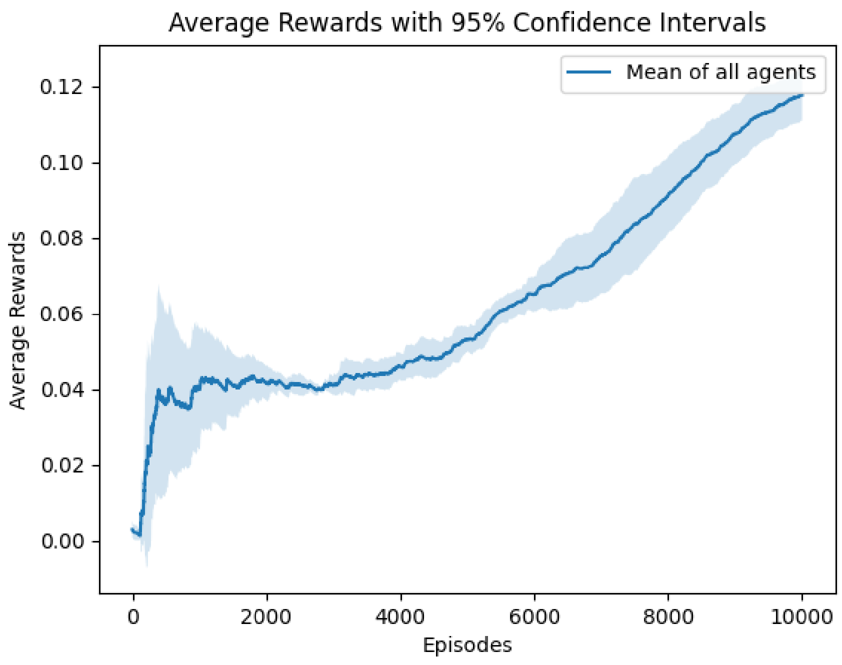
	0	1
0	2.7	2.7
1	2.8	2.8
2	3	3.2
3	3.2	3.9
4	3.7	4.7
5	4.4	5.4

Visualization of greedy policy obtained from the Q-values

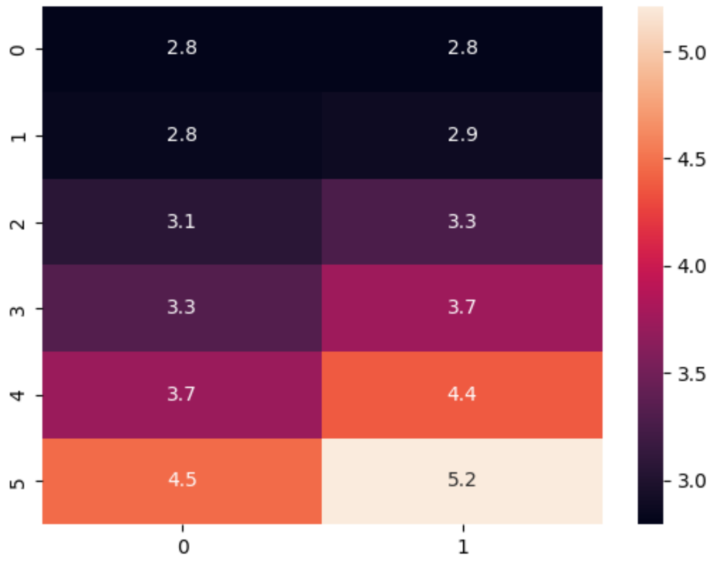


Q-values initialized between 20 and 50

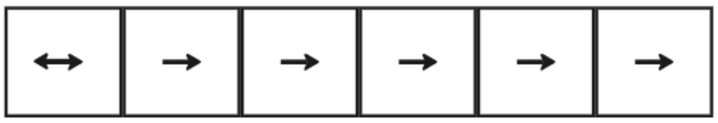
Average rewards



Visualization of Q-values (states on y-axis, action on x-axis)



Visualization of greedy policy obtained from the Q-values



5. Discussion

What are your conclusions? How do the agents perform? Discuss if the behavior is expected based on what you have learned in the course.

Firstly, looking at the policies that the algorithms produce we can see some different things. In the Frozen Lake environment the policies produced by the algorithms are not exactly the same but very similar. Looking at the Q-table we can see that there are only small differences between the actions which give us different policies. Whether the policies produced for Frozen Lake are optimal is hard to say since the environment is slippery meaning that environment is stochastic to some degree and the agent may not move in the intended direction. However the fact that the policies are so similar is a good sign. We did also run the experiment without the environment being slippery and this gave us a policy that looked optimal. For River Swim all the algorithms produced the same policy which looks optimal. For both environments the Q-values for early states are quite similar but the difference increases for states that are closer to the terminal state. This feels quite logical as the algorithm becomes more confident the closer it gets to the goal. This is probably why in Frozen Lake the difference in the policies is usually not close to the terminal state.

In the frozen lake environment, we see that initializing Q-values to be higher leads to a bigger variance in the initial episodes. This is likely due to the fact that initializing the Q-table with bigger values encourages exploration, meaning that the agent will make some decisions that lead to lower rewards. For the river swim environment, we see that setting the Q-values to smaller values initially seems to be making the confidence interval larger. This could be explained by the fact that the converged Q-table shows values around 3 to 5, for all algorithms and all levels of initial Q-values. Initializing the Q-table to values between 0 and 1 is then actually a lot lower than any of the converged values. This means that the agent is less likely to explore, and will therefore “satisfied” with any Q-value that is higher, even if it isn’t optimal. Therefore, if we are lucky, we accidentally perform an action that gives a high reward and then continue to exploit this action. However, if we are unlucky, the action gives a higher Q-value than our initial value, but still being far from the optimal one. We then continue to exploit this lower value, leading to a suboptimal policy. Despite this, we still observe convergence to an optimal policy, and this is likely because we used the averaged Q-values over five agents.

Overall, the initialization of Q-values doesn’t make a big impact on the convergence of the Q-table and thus the policy. We only tried the different initialization levels for the Q-learning algorithm, but we would expect the same result also for the other algorithms. As we seem to balance exploration and exploration rather well, the agent will eventually converge to the true Q-values. However, if we would have run the algorithms for a smaller number of episodes, the initialization may have played a bigger part.

Comparing the performance of the different algorithms we can see that for Frozen Lake Q-learning and Expected SARSA performs the best. The performance of these two algorithms are quite similar which might be expected since they are rather alike. For River Swim all algorithms perform roughly equally well. This might be because the River Swim environment is less complex than Frozen Lake with less states and less actions. Therefore all four algorithms can handle River Swim well and they require less tuning to get all the performance out of the algorithm. SARSA and Double Q Learning perform worse than the other algorithms on the Frozen Lake environment. We don’t know exactly why this is but it could be due to many reasons. Maybe these two algorithms needed more tuning of the parameters to work or maybe they needed more time to converge. It could also be that they just don't work well with the Frozen Lake Environment.

Another interesting aspect that we have not explored exhaustively is the values of the variables for the algorithms. The discount factor (gamma) was given, but we were free to select the step size/learning rate (alpha). We tried some different values, and ended up setting it to 0.1 as it gave a good convergence. With a higher learning rate, the new information has a larger impact. However, if we had more time we could have analyzed this further and carried out more experiments with different learning rates.

Worth noting is that our results display the averages over five agents, which means that we don’t see the performance of each individual agent. As some confidence intervals are larger, it seems that the performances of the different agents aren’t completely stable. It would be interesting to perform the experiments with more agents, but also to try and evaluate the performance of the individual agents. However, to conclude, we believe that the agents generally perform well, as the policies are near optimal for all algorithms in both environments.