Laboratory 6

Variant 5

Group 5

By Jan Szachno and Aleksandra Głogowska

---

1. Introduction

The task is to create an implementation of the Q-Learning algorithm to solve a toy Reinforcement Learning problem using the Gymnasium library. For our variant, we have to use the FrozenLake environment and use the arguments “FrozenLake-v1” nad map_name=”8x8” for gym.make(). The visualization of the environment is shown in Figure 1.1.

Figure 1.1 - Visualization of the FrozenLake environment.

2. Algorithm Description

2.1. General Algorithm Description

In the task, we are supposed to implement the Q-Learining algorithm to solve the reinforcement learning problem using Gymnasium library. Q-Learning is a model-free reinforcement learning algorithm used to find the values of actions for particular states in the environment. It iteratively learns an action-value function, known as the Q-function, which estimates the expected “reward” of taking an action in a given state.Through exploration and exploitation, Q-learning updates Q-values based on observed rewards and transitions, while searching for the long-term rewards and aiming to maximize them. This algorithm keeps the balance between exploration, which in this case is trying new actions, as well as the exploitation - selecting known high-reward actions.

Reinforcement learning is a type of machine learning where an agent learns to make decisions by interacting with an environment to achieve a specific goal. Unlike supervised learning, where the model is trained on labeled data, and unsupervised learning, where the model finds patterns in unlabeled data, reinforcement learning relies on feedback from the environment through rewards or penalties.

In our task variant, we are using the FrozenLake environment, which is an NxN plane that simulates a frozen lake. It involves crossing the lake and finding a present on one of the fields without falling into any of the holes. The character can move up, down, left, and right if there is no wall there. If the character tries to go through a wall it bounces back to the same place. If it falls into any of the holes, it returns back to the starting point. In our variant, there is a reward only if the agent reaches the final field with the present. The other fields does not have rewards or penalties. The agent only “starts to learn” when it reaches the goal for the first time. After it reaches the goal field, it memorizes the field before the reward and the

action that had to be taken to reach the present. In this way, the character memorizes different paths and goes through the learning process.

To achieve the goal of the task, we are using a Q-table, which represents the state-action space of the environment. The state space are the rows of the q-table that correspond to the possible states (fields) that the character can be in, and each row represents a unique state in the environment. Action space are the columns of the table that represent the possible actions that the agent can take in each state, and each column represents a unique action that the character can perform. The q-table is initialized with zeros for all state-action pairs, which indicates that the agent has no prior knowledge of the possible outcomes of their actions. During the training, the values are updated iteratively based on the observed rewards and transitions experienced by the agent. This update follows the Q-learning rule, which includes the reward received, the maximum Q-value of the next state, and the learning rate. The algorithm assumes that the agent learns to update Q-values in the table and understands which actions are more rewarding in different states during the interaction with the environment. After a while, the Q-values in the table converge toward their optimal values, representing the expected cumulative rewards for each action in each state. After the training, the agent can consult the Q-table to make decisions. It selects actions by choosing the one with the highest Q-value for the current state, following the learned policy. If the Q-values are changing only insignificantly, where the difference is close to zero, it means that the algorithm has been through all possible solutions and that it learned the optimal path.

2.2. Implementation Description

Our implementation starts from the environment setup. We are setting the parameters. The number of episodes defines how many times the agent will undergo the training. Each episode represents a complete run of the environment, from the initial field to the goal field.

The learning rate determines the extent to which the agent updates its Q-values based on new information, so in other words, how much weight it gives to new information compared to what it already knows. A higher learning rate means the agent is more responsive to recent experiences.

The discount factor controls the importance of future rewards in the agent's decision-making process. It determines the extent to which the agent values immediate rewards over future rewards. When navigating the Frozen Lake, the agent encounters immediate rewards (reaching the goal) and future rewards (e.g., reaching the goal quickly or efficiently). The discount factor influences how the agent balances these immediate and future rewards. The discount factor accounts for the temporal consequences of actions. It guides the agent to consider not only the immediate reward of a chosen action but also the potential future rewards it might lead to. For example, the agent may choose a longer path if it leads to a higher chance of reaching the goal in the future.

The exploration rate controls the agent's exploration strategy during training. It represents the probability of the agent choosing a random action (exploration) rather than exploiting its current knowledge (exploitation). A high exploration rate indicates that the agent is more inclined to explore, favoring random actions or less-optimal choices in pursuit of new information. This can be beneficial for discovering better strategies but may result in short-term suboptimal performance.

The exploration decay rate controls the rate at which the exploration rate decreases over time. It allows the agent to gradually shift from exploration to exploitation as training progresses.

The maximal episode length sets the maximum number of steps the agent can take in a single episode before it is forcibly terminated. It prevents episodes from running indefinitely and ensures that the agent doesn't spend too much time in a single episode.

The slippery parameter set to True means that the agent can randomly choose another action other than the one that it intended to do. This behavior introduces stochasticity into the environment, making it more challenging for the agent to learn an optimal policy. It also adds an element of randomness to the agent's decision-making process.

After setting the parameters, we create the environment with the gym.make() function, taking the arguments including “FrozenLake-v1,” and map_name=”8x8” (map has 64 fields). Then, we initialize the Q-table and enter the training loop. The loop iterates over a specified number of episodes. Within each episode, the agent interacts with the environment until the episode is terminated or truncated. The terminated flag means that the agent reached the goal field, the present, and the truncated flag means that the number of steps exceeded the limit. The agent selects an action based on an exploration-exploitation strategy. It either explores by selecting a random action or exploits by selecting the action with the highest Q-value. The chosen action is executed in the environment, which returns the next state, reward, termination flags, and additional information. The Q-table is updated based on the observed transition and reward using the Q-learning update function. The exploration rate has decayed over time. Once the episode terminates or reaches the maximum episode length, the loop moves to the next episode. Once all episodes are completed, the training environment is closed.

3. Experiments

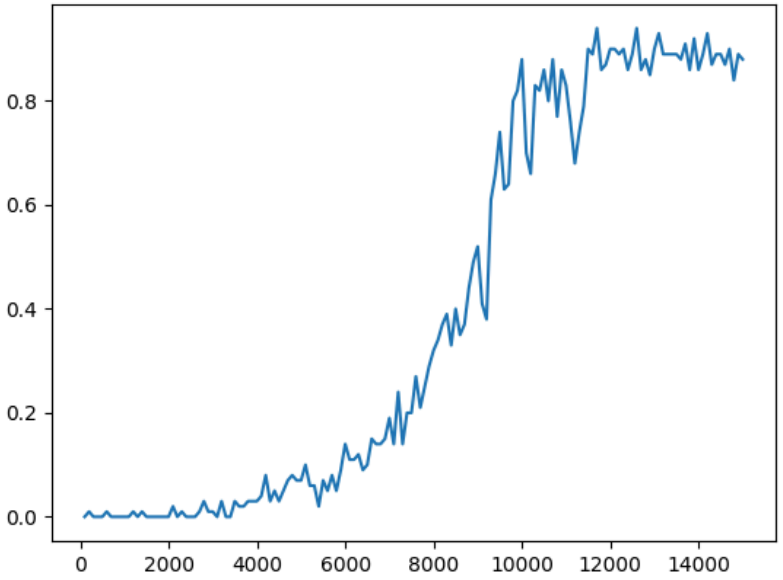
We decided to experiment with the following parameters:

- number of training episodes
- learning rate
- discount factor
- exploration decay rate
- maximum episode length

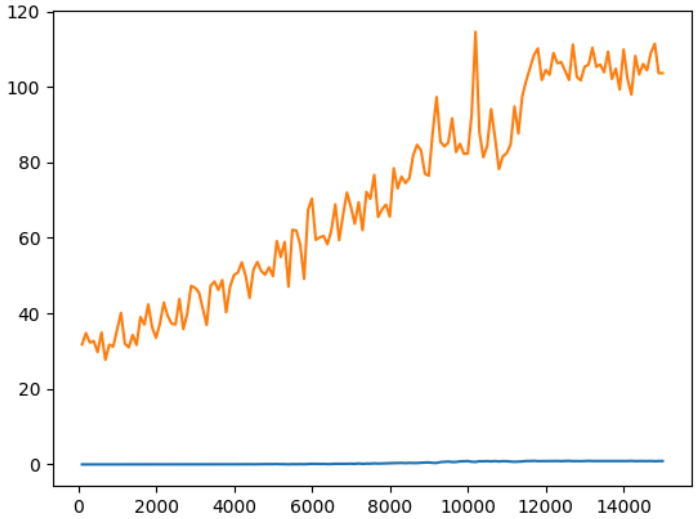
To have a reliable comparison between experiments, we decided to first run a “control experiment”. We will use results from this experiment across all future experiments. Here are the “default parameters” we choose. If not stated otherwise, they will look the following in all experiments:

```
# parameters
NUM_EPISODES = 15000
LEARNING_RATE = 0.1
DISCOUNT_FACTOR = 0.99
EXPLORATION_RATE = 1
EXPLORATION_DECAY_RATE = 0.0001
MAX_EPISODE_LENGTH = 200
```

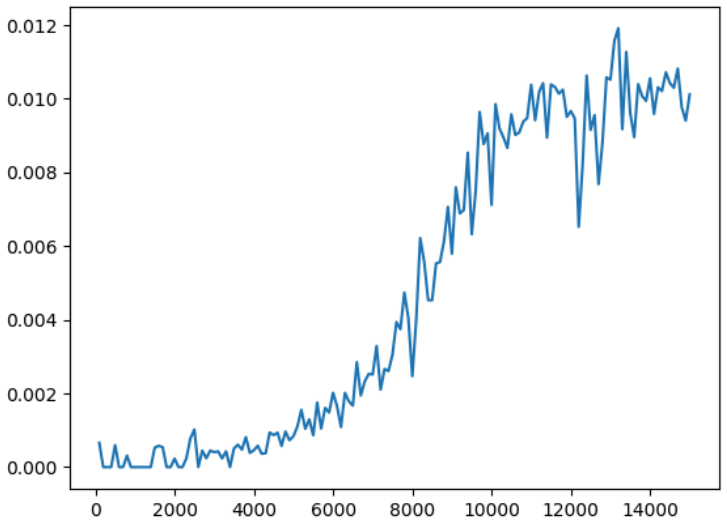
Here are the results of the experiment:



The graph shows us the average success rate from 100 episodes. As we can see the default parameters chosen give us pretty good results, achieving over 0.9 success rate.



This graph shows us the average amount of steps taken in an episode. It does not take into account whether the episode ended in a success or not.

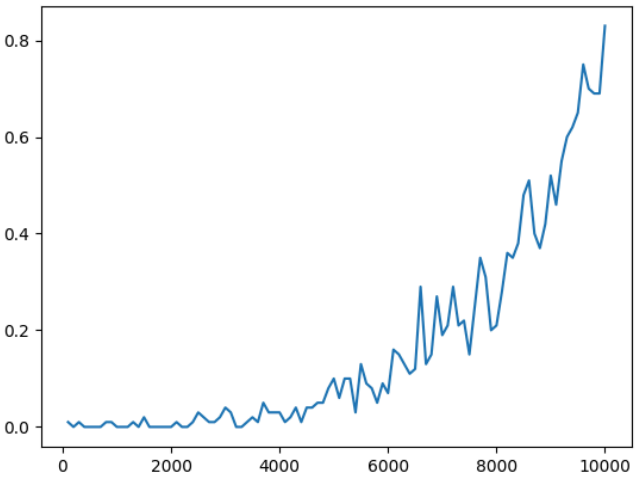


Lastly, we introduce our own custom metric called “score”. Score is equal to the average success rate from 100 episodes divided by the average number of steps per 100 episodes. This means that to maximize the “score”, the algorithm needs to reach the goal reliably and do it in as little steps as possible. This will allow us to compare them.

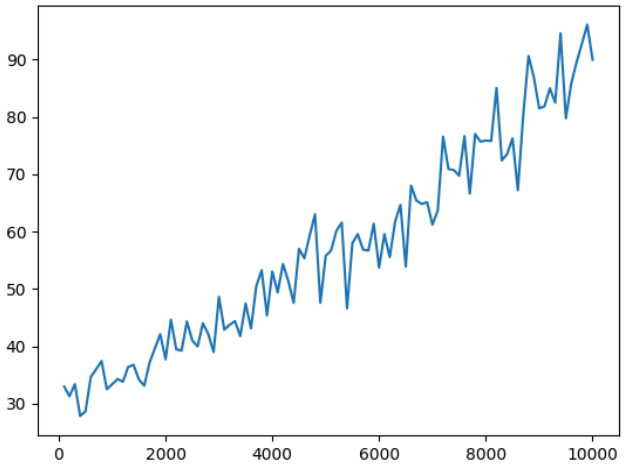
Number of training episodes

NUM_EPISODES = 10000

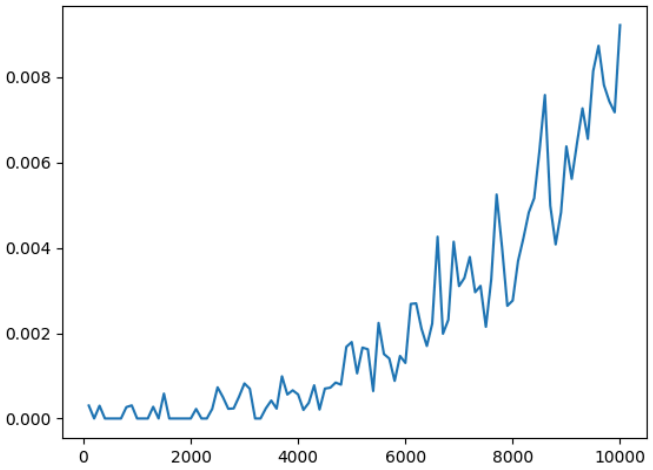
We will start from setting the number of training episodes to a value lower than the “default” one.



As we can see from the chart depicting the success rate, it is not as good as in the control experiment. However, the results are comparable and the benefit of achieving comparable results faster might be a good compromise in this case.



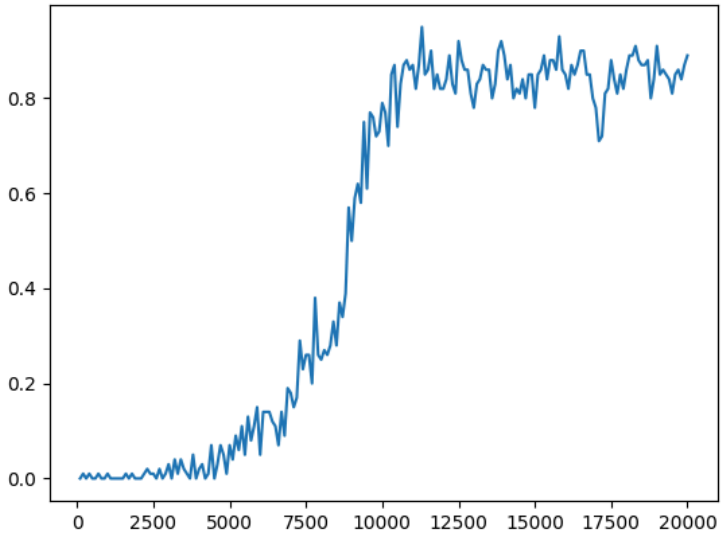
Average number of steps taken per episode is a bit lower than in the control experiment, but the difference is negligible and can be considered the result of randomness of the learning process.



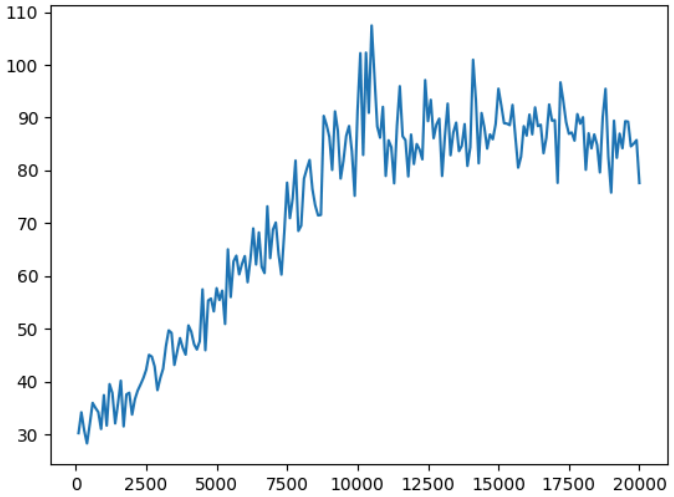
Lastly, the score achieved in this experiment is slightly lower than in the control experiment however the difference is relatively small.

NUM_EPISODES = 20000

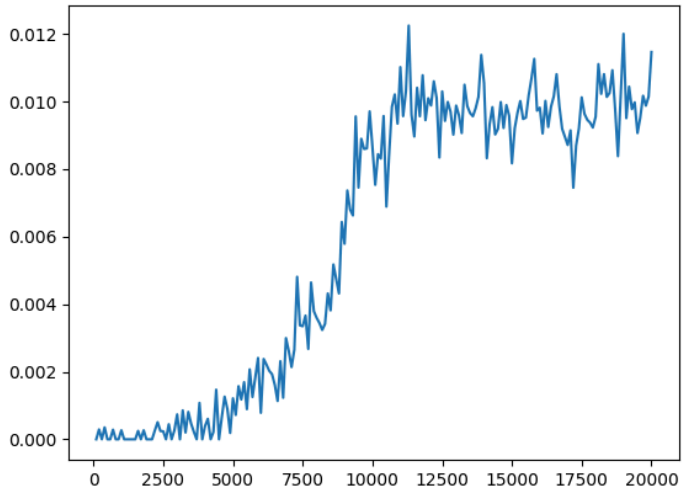
In this experiment we will try an opposite approach and set the number of training episodes to a higher value than the “default” one.



We can observe on the graph showing average success rate that after about 12000 episodes the values do not get better and just oscillate. This means that any value bigger than that is not necessary and is just a waste of computing resources without any added benefit.



The number of steps did not change much compared to the first experiment. It showed a slight decline in the end but the difference is negligible.

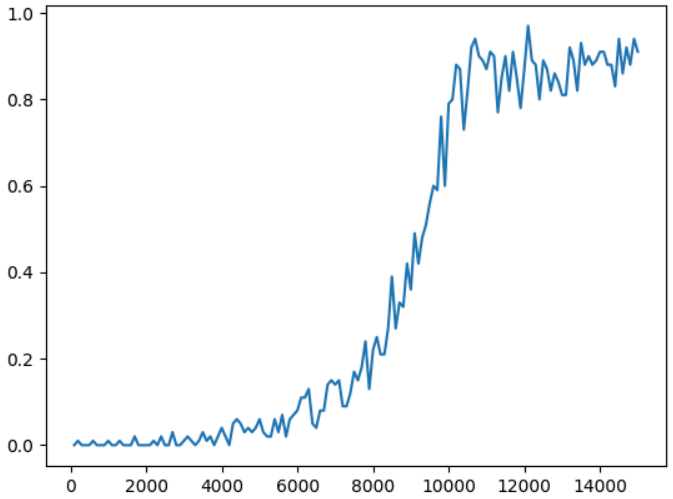


Compared with the first experiment the score did not improve which means that the additional training did not introduce any benefits

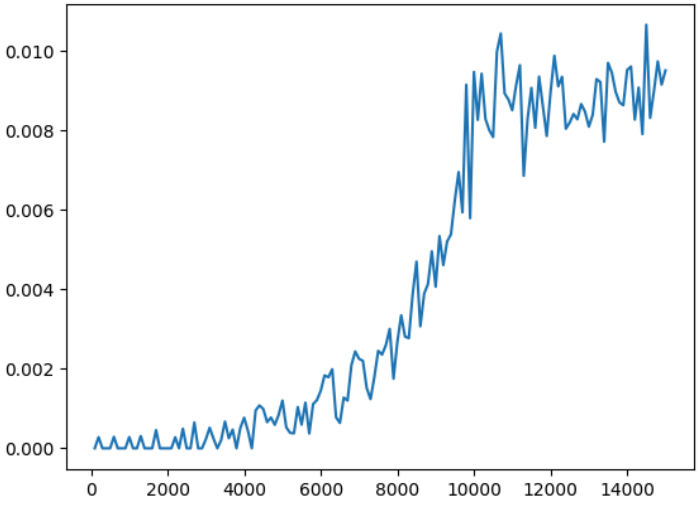
Conclusions from the number of episodes experiment
The value of the number of episodes should be chosen carefully to maintain balance between learning time and the quality of achieved results. Small number of episodes might yield bad results, so a large amount of episodes might be an unnecessary waste of time.

Learning rate

Learning rate = 0.3

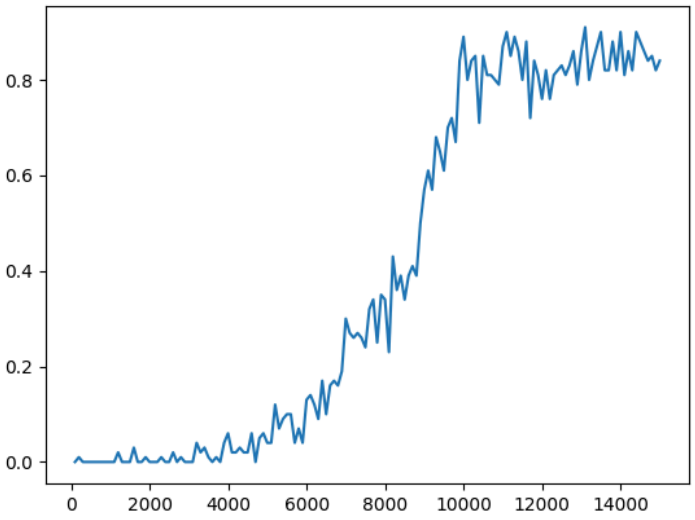


Increasing the learning rate alone did not change how fast we achieved good results. However the final results were slightly better than in the control experiment.

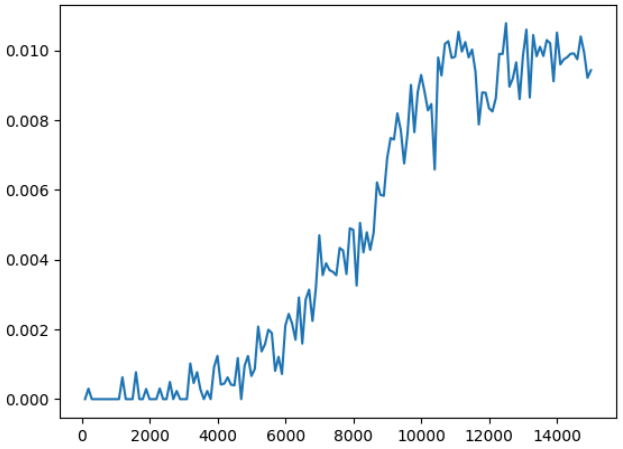


The final score was slightly worse than in the control however it was comparable.

Learning rate = 0.05



In this case also changing only the learning rate did not affect the results too much.

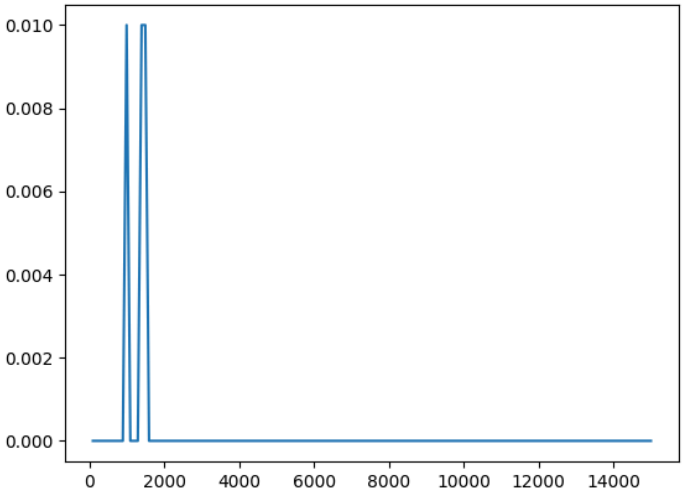


The score was comparable to the control group.

Conclusions from the learning rate
We think that changing the learning rate alone is not a good idea and it should be done in parallel with changing other parameters.

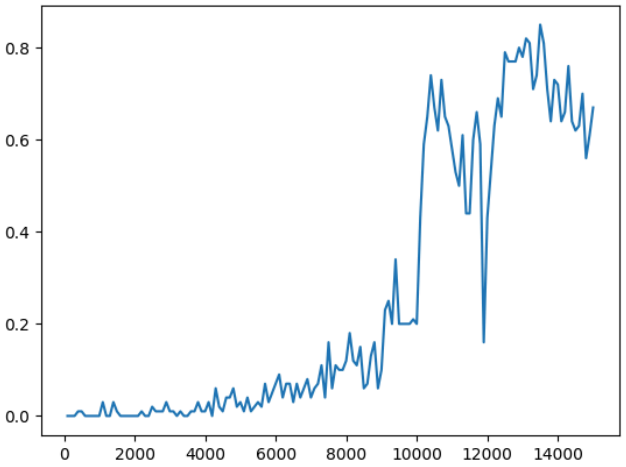
Discount factor

Discount factor = 0
When we set the discount factor to 0, the algorithm does not value future rewards and only considers the current ones. However in this experiment future rewards are very important since the reward is only granted when the agent reaches the final tile.

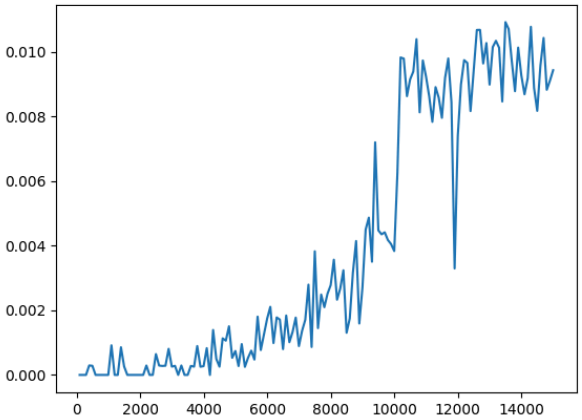


We can see on the graph that the algorithm performed very poorly and it shows us the importance of taking future reward into account.

Discount factor = 0.8



Lowering the discount factor in this case had a negative impact on the speed of learning. Good results were achieved later than in the control experiment. Final results were worse and less stable.

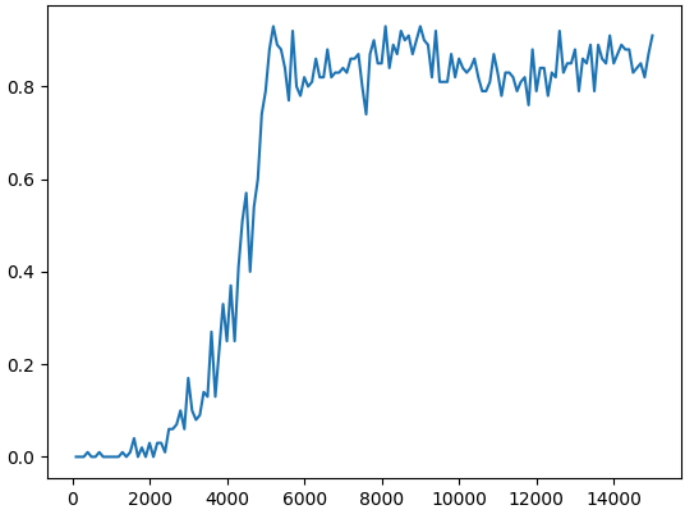


The score was also worse and fluctuated more than in the control experiment.

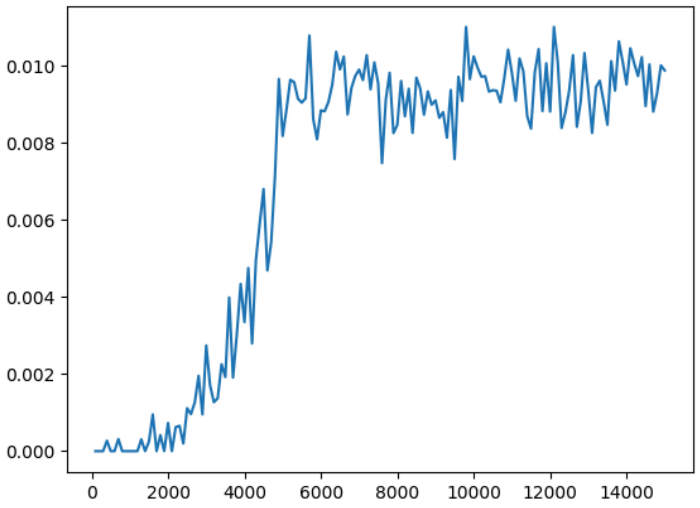
Conclusions from discount factor experiment
Discount factor proved to be an important parameter in this experiment because of the reward distribution on the board.

Exploration decay rate

Exploration decay = 0.0002
Making the exploration decay a bigger number results in less exploration and more exploitation.



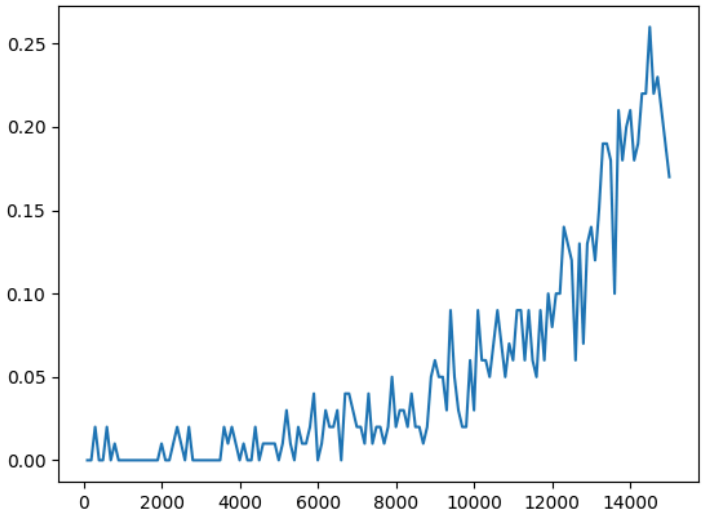
More exploitation in this case proved to be a good strategy since it allowed us to achieve good results in less episodes.



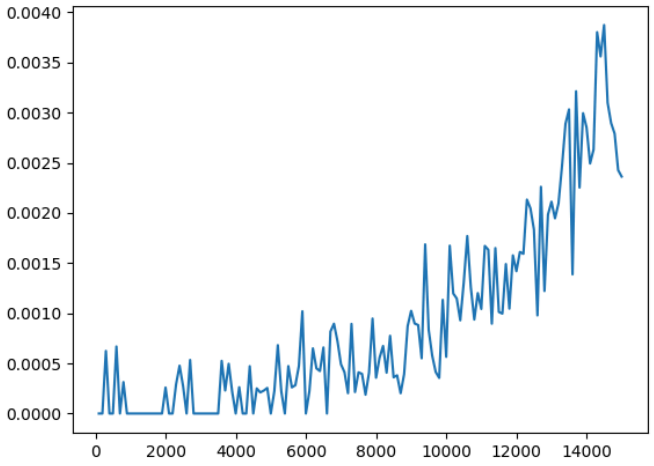
The score is similar to the final experiment, but it was achieved a lot faster.

Exploration decay = 0.0005

More exploration in our case did not yield better results. The speed at which the algorithm learned the best path was too slow and we did not reach satisfactory results in time. More episodes would be probably needed to achieve satisfactory results.



The final score was also bad.

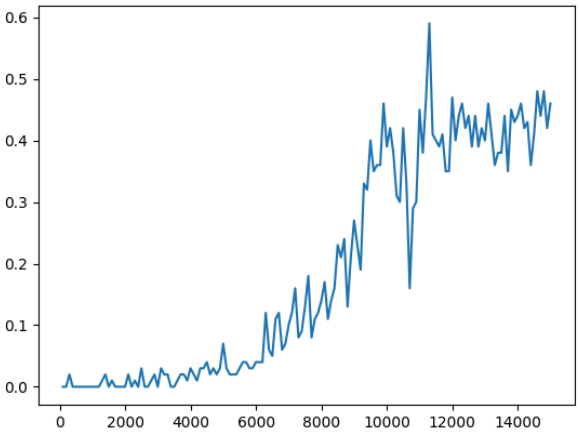


Conclusions from the exploration decay experiment
Exploration decay factor defines balance between exploration and exploitation. The value should be chosen carefully to ensure quick convergence to satisfactory results. If not chosen carefully it will lead to either too much exploration or too much exploitation.

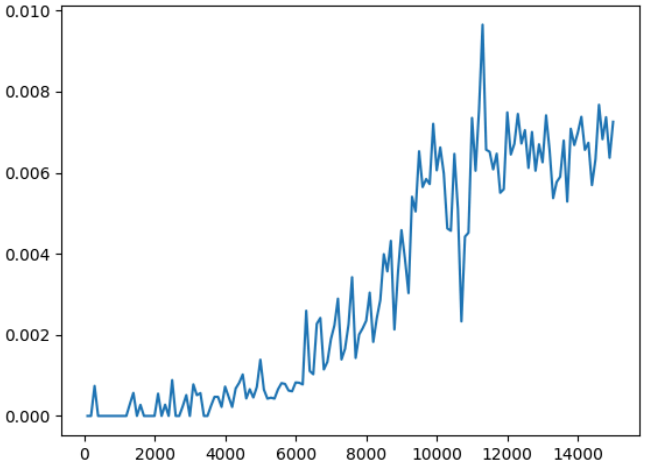
Maximum episode length

Max episode length = 75

Lowering the limit of moves in each episode makes learning faster. We also had a hypothesis that it might somehow “force” the algorithm to achieve the goal faster but this proved to not be the case.



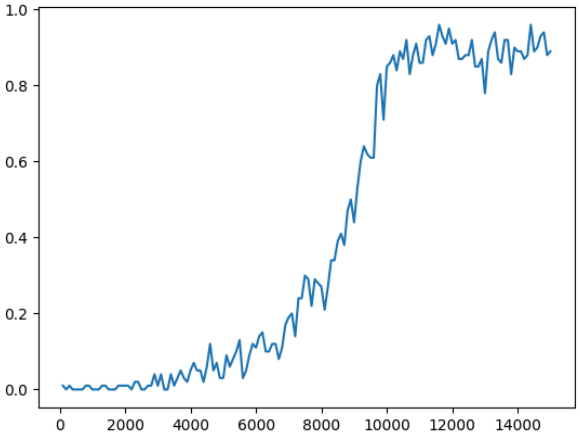
The final results were worse than in the control. The algorithm did not have enough time in each episode to reach the destination



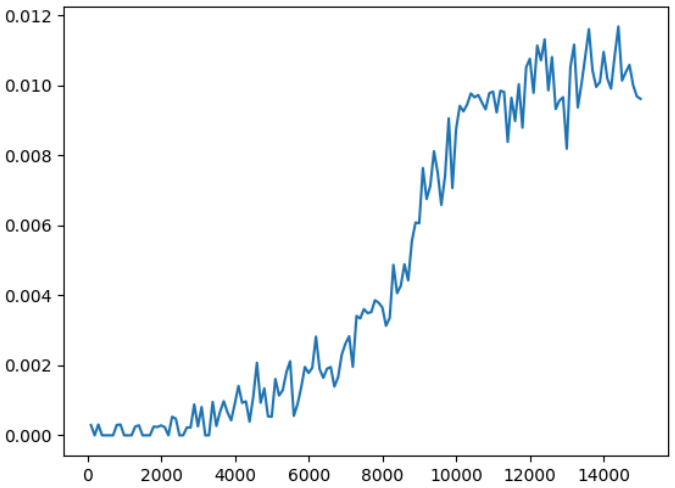
The score was also worse.

Max episode length = 350

Giving the algorithm more time in each episode to reach the end improved the success rate a lot. However, the main gains are caused by the fact that more attempts which previously would be over the limit are not allowed.



Our previous assumptions can be visible on the “score” chart. It did not improve compared to the control group. With the increase of the success rate, the number of steps also increased negating the potential gains.



4. Conclusions

This report presented an exploration of the implementation of the Q-Learning algorithm within the FrozenLake-v1 environment of the Gymnasium library. The laboratory focused on varying several parameters to evaluate their impact on the learning efficiency and performance of the Q-Learning algorithm.

In summary, the successful implementation of Q-Learning in the FrozenLake environment depends on a well-calibrated set of parameters. This laboratory underscores the need for thoughtful tuning of these parameters based on the specific characteristics and requirements of the environment.