Analysis: Q-learning

Name: Lucas Jacobs
Pcn: 490692
Class: S4
Teachers: Iman Mossavat
Date: 10-05-2023
Word count: 983

Table of Contents

Git Link

https://git.fhict.nl/I490692/s4-depth-lucas-jacobs.git

FrozenLake environment

This is the visualised map that I am testing my Q-learning on.

Frozen Lake

This can also be visualised in a array, which will be used in the code:

```
"4x4":[
    "SFFF",
    "FHFH",
    "FFFH",
    "HFFG"
    ]
```

Rewards:

Reaching the goal (G): +1
Reach a hole (H): 0
Reach frozen (F): 0

Frozen Lake: Action- and Observation Space

Action Space

For the action space the agent takes a 1-element vector for the actions. The action space is the direction that the agent can move in which are:

- 0: LEFT
- 1: DOWN
- 2: RIGHT
- 3: UP
-

Observation Space

Observation is a value representing the agent’s current position as current_row * nrows + current_col. So the observation corresponds to the agent’s location on the grid. In our environment, the goal position can be calculated as 3*4 + 3 = 15. The total observations of this 4x4 map are 16.

Analysis code

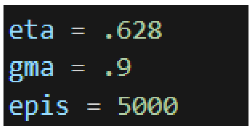
At first it will create the environment of the frozen lake, with the specified parameters, such as how big the map is, and whether the mode slippery is turned on or not. The variable ‘is_slipery’can be set to either true or false. it basically gives more randomness to deal with when set to true. To give an example when it is set to true. When an action is left then the chances are:

- P(move left)=1/3
- P(move up)=1/3
- P(move down)=1/3

Next, we have a variable called ‘Q’, this is a two-dimensional array, where each row represents a state and each column a particular action.
We also have some variables of the Q-learning predefined.
Begin with, 'eta', which essentially defines the learning rate of the Q-learning algorithm, it determines how much the Q-value changes with each iteration.
Next, the variable "gma," which is the discount factor, it will be used to specify the value of a potential reward.
The number of episodes the algorithm will run for will be specified in the variable "epis".
Moreover, it also defines a 'rev_list' variable, a list of the total rewards that have been generated for each episode.
Furthermore, it will run the Q-learning algorithm for the number of episodes that have been defined. When in the loop, it will render the game and chooses an action based on the current state and in the Q-table, it will choose an action in the frozen lake environment. After this, it will update the Q-table based on the new state and reward.

Results

For the results I used the following predefined variables if not mentioned something else:

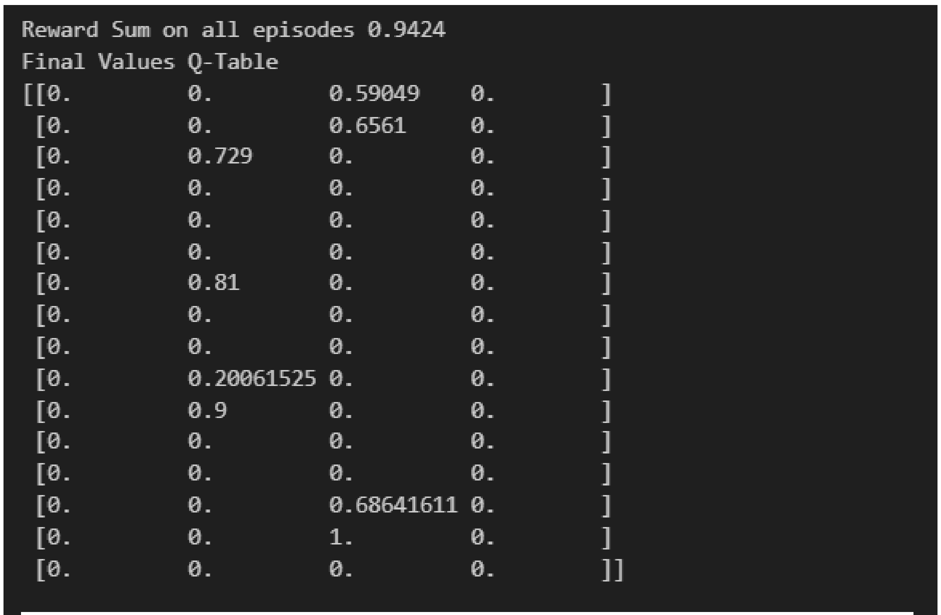


```
eta = .628
gma = .9
epis = 5000
```

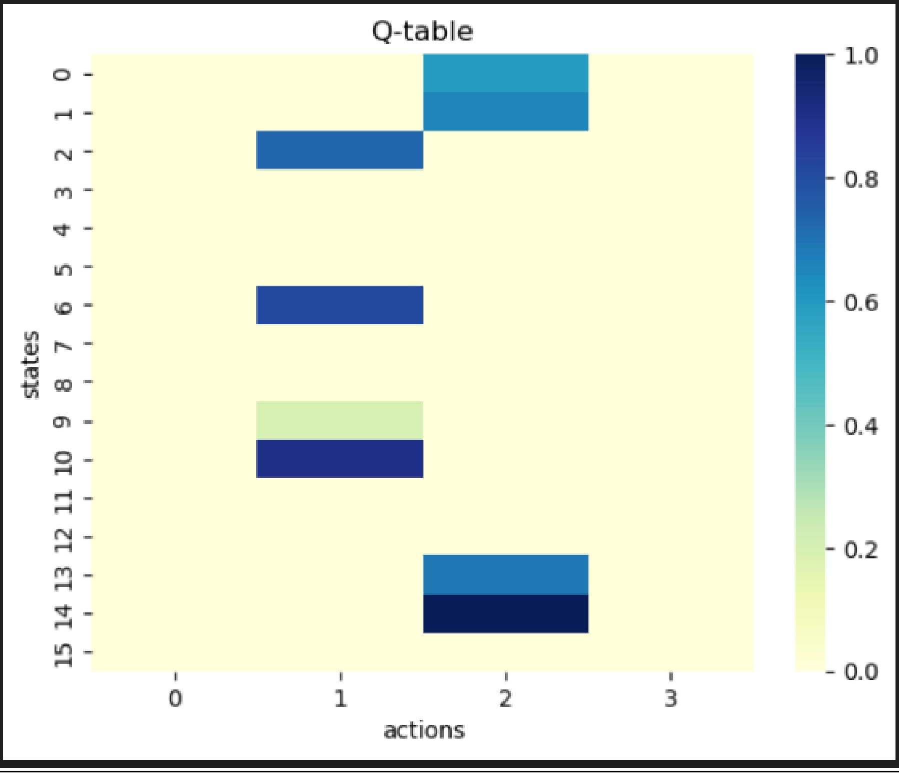
The main thing to experience with is the learning rate when setting ‘is_slippery’ to true or false.
To visualise the results, I used a heatmap, to give a more clear view of what the agent have chosen the most often.

‘is_slippery = False’

These are the results when the variable is turned is set to false.



```
Reward Sum on all episodes 0.9424
Final Values Q-Table
[[0.         0.         0.59049    0.        ]
 [0.         0.         0.6561     0.        ]
 [0.         0.729      0.         0.        ]
 [0.         0.         0.         0.        ]
 [0.         0.         0.         0.        ]
 [0.         0.         0.         0.        ]
 [0.         0.81       0.         0.        ]
 [0.         0.         0.         0.        ]
 [0.         0.         0.         0.        ]
 [0.         0.20061525 0.         0.        ]
 [0.         0.9        0.         0.        ]
 [0.         0.         0.         0.        ]
 [0.         0.         0.         0.        ]
 [0.         0.         0.68641611 0.        ]
 [0.         0.         1.         0.        ]
 [0.         0.         0.         0.        ]]
```



Actions:
0: LEFT, 1: DOWN,
2: RIGHT, 3: UP

After 5000 episodes you can see that for every state it will have a clear action that it will take. But also after trying with only 500 episodes, it was clear what the agent want to choose at certain stages.

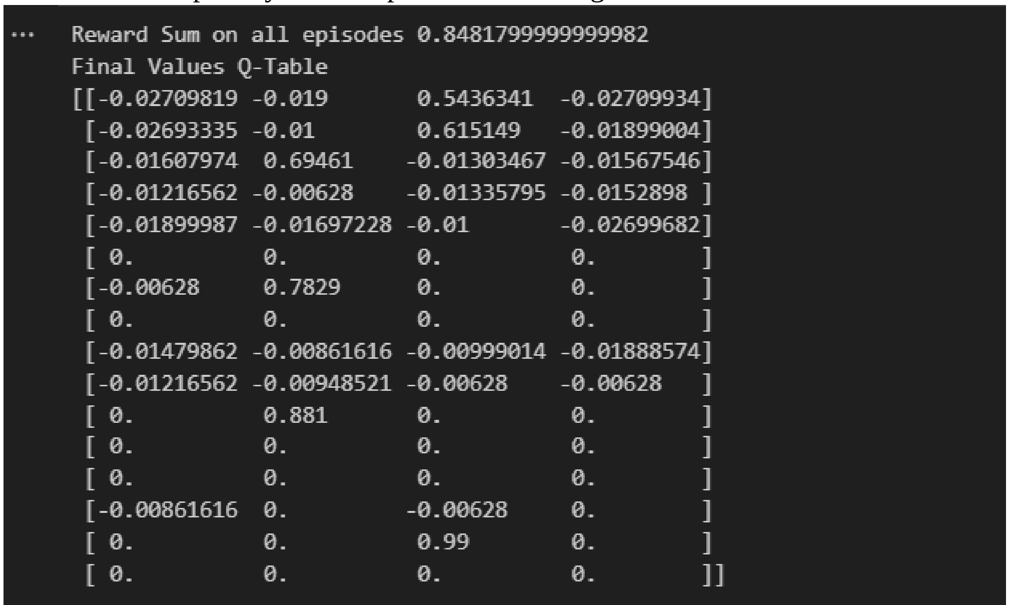
Quantitative performance

In the Q-table, when a value is zero, this means that the agent learned not to make a certain action at a certain state.

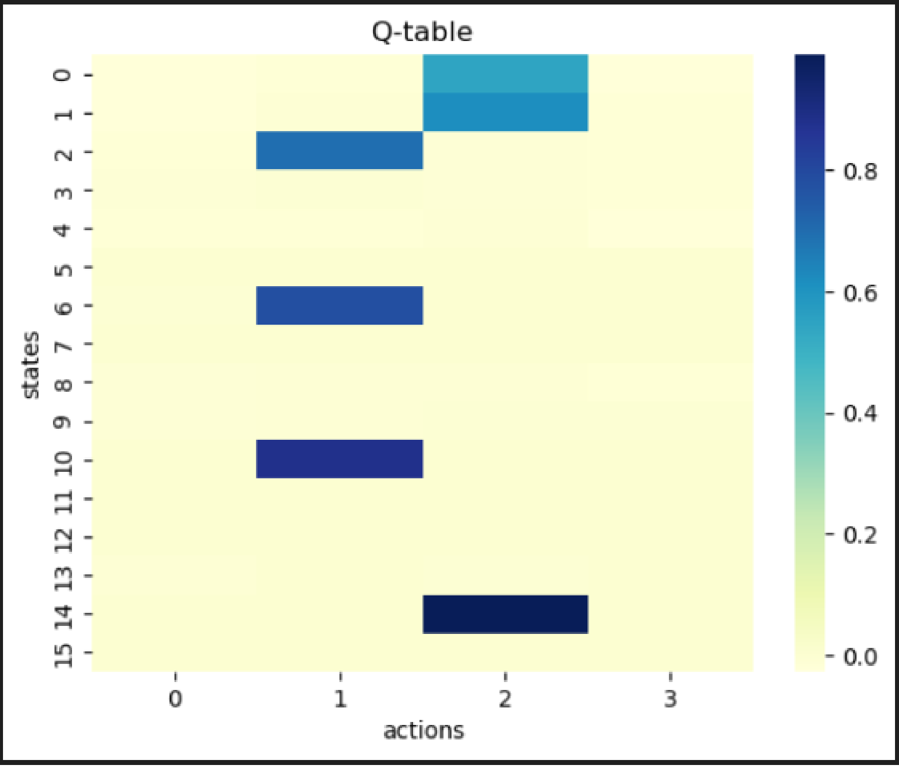
The value of rALL is expected to change over each episode, because the agent improves and learns from its decisions which will improve the policy. In the beginning the values can be low due to the fact that it may not reach the end goal immediately. When it runs enough amount of times rALL is expected to increase. When the optimal policy has been reached. rALL will be the maximum value.

Performing with penalty

When we add a penalty of -0.01 per move. It will give a result of this:



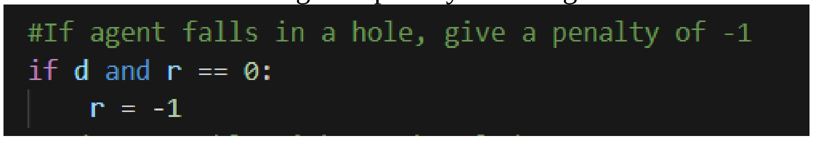
```
...  Reward Sum on all episodes 0.8481799999999982
     Final Values Q-Table
     [[-0.02709819 -0.019       0.5436341  -0.02709934]
      [-0.02693335 -0.01        0.615149   -0.01899004]
      [-0.01607974  0.69461    -0.01303467 -0.01567546]
      [-0.01216562 -0.00628    -0.01335795 -0.0152898 ]
      [-0.01899987 -0.01697228 -0.01       -0.02699682]
      [ 0.          0.          0.          0.        ]
      [-0.00628     0.7829      0.          0.        ]
      [ 0.          0.          0.          0.        ]
      [-0.01479862 -0.00861616 -0.00999014 -0.01888574]
      [-0.01216562 -0.00948521 -0.00628    -0.00628    ]
      [ 0.          0.881       0.          0.        ]
      [ 0.          0.          0.          0.        ]
      [ 0.          0.          0.          0.        ]
      [-0.00861616  0.         -0.00628     0.        ]
      [ 0.          0.          0.99        0.        ]
      [ 0.          0.          0.          0.        ]]
```



As you can see the performance is different and after running it for several times it will give more random answers. So with a penalty of 0.01 per move it will give the agent the task to find a more efficient way to complete the task.

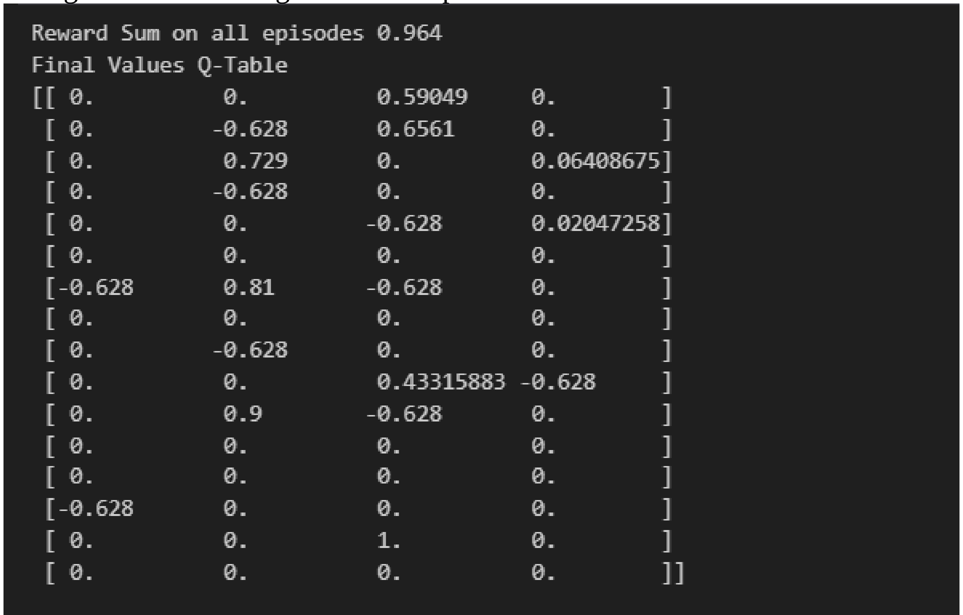
Performing with hole penalty

The code that I added to give a penalty when it goes in a hole:

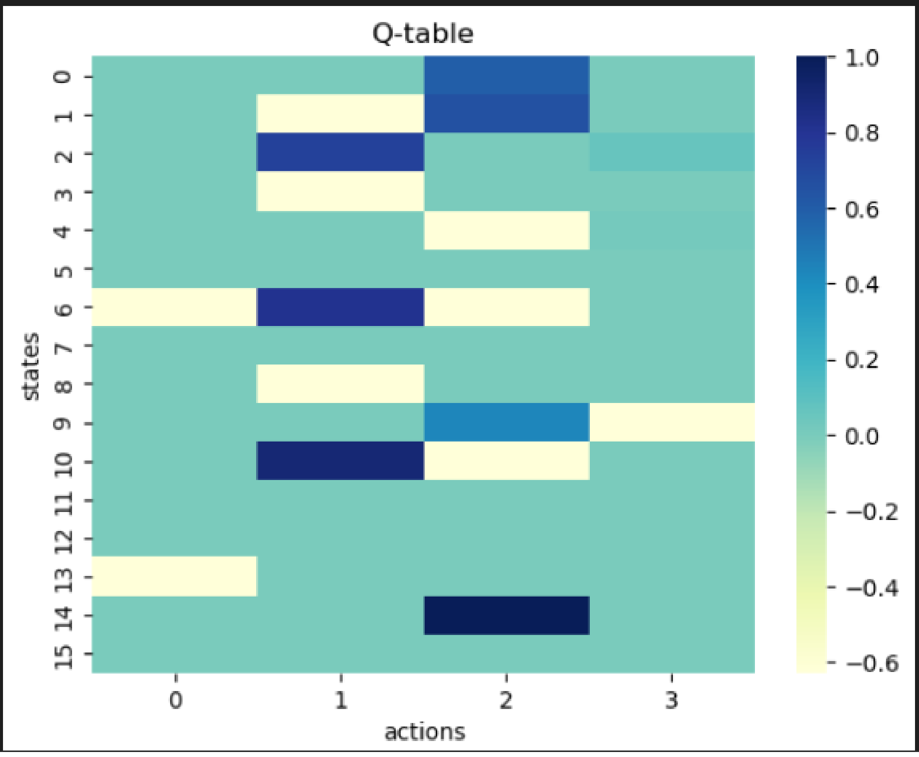


```python
#If agent falls in a hole, give a penalty of -1
if d and r == 0:
    r = -1
```

This gives the following result with epis set to 500:



```
Reward Sum on all episodes 0.964
Final Values Q-Table
[[ 0.          0.          0.59049     0.        ]
 [ 0.         -0.628       0.6561      0.        ]
 [ 0.          0.729       0.          0.06408675]
 [ 0.         -0.628       0.          0.        ]
 [ 0.          0.         -0.628       0.02047258]
 [ 0.          0.          0.          0.        ]
 [-0.628       0.81       -0.628       0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.         -0.628       0.          0.        ]
 [ 0.          0.          0.43315883 -0.628      ]
 [ 0.          0.9        -0.628       0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [-0.628       0.          0.          0.        ]
 [ 0.          0.          1.          0.        ]
 [ 0.          0.          0.          0.        ]]
```

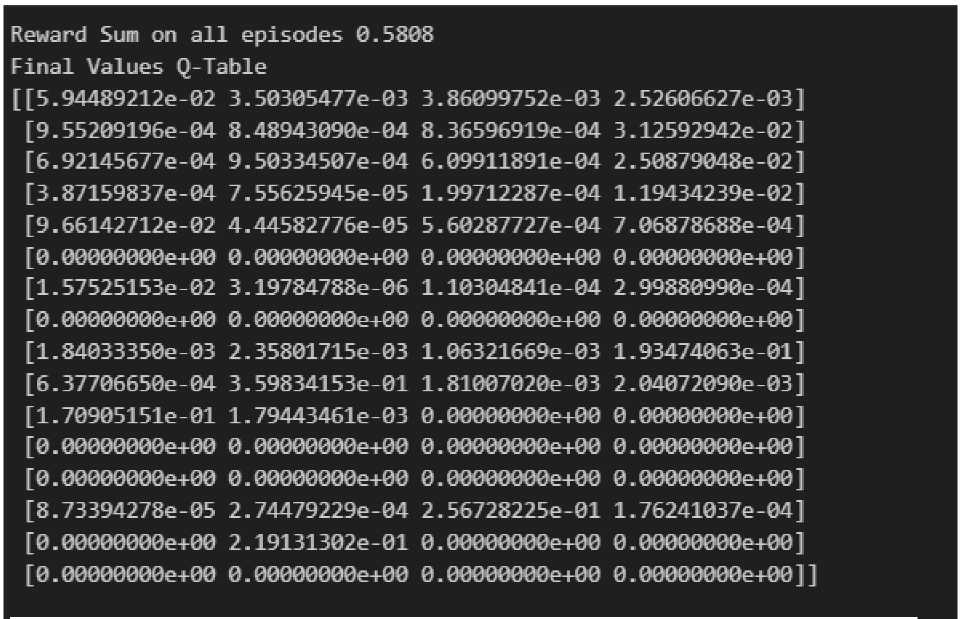


This makes it more challenging for the agent because it needs to learn to avoid the holes in the environment. We can see it is more spread out in the heatmap, with the choices the agent took.

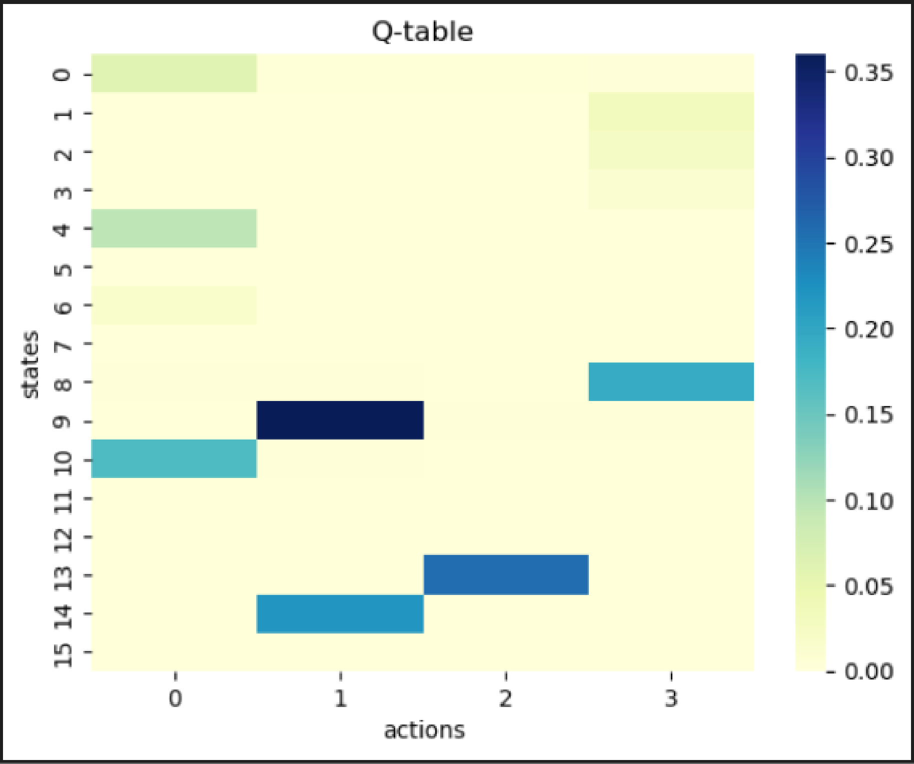
When turning is_slippery = true it will be really hard to learn as a agent because when he wants to make a correct decision it will also have the chance to go in a whole (P(1/3)), so it get punished for making a decision that was good. This will give it a really hard time.

‘is_slippery = True

These are the results when the variable is turned is set to True.



```
Reward Sum on all episodes 0.5808
Final Values Q-Table
[[5.94489212e-02 3.50305477e-03 3.86099752e-03 2.52606627e-03]
 [9.55209196e-04 8.48943090e-04 8.36596919e-04 3.12592942e-02]
 [6.92145677e-04 9.50334507e-04 6.09911891e-04 2.50879048e-02]
 [3.87159837e-04 7.55625945e-05 1.99712287e-04 1.19434239e-02]
 [9.66142712e-02 4.44582776e-05 5.60287727e-04 7.06878688e-04]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [1.57525153e-02 3.19784788e-06 1.10304841e-04 2.99880990e-04]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [1.84033350e-03 2.35801715e-03 1.06321669e-03 1.93474063e-01]
 [6.37706650e-04 3.59834153e-01 1.81007020e-03 2.04072090e-03]
 [1.70905151e-01 1.79443461e-03 0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [8.73394278e-05 2.74479229e-04 2.56728225e-01 1.76241037e-04]
 [0.00000000e+00 2.19131302e-01 0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]]
```



Actions:
0: LEFT, 1: DOWN,
2: RIGHT, 3: UP

In these results you can clearly see that it is **not** done with learning, due to the fact that when an agent wants to make a certain decision, randomness can still decide that it will take another action. Therefore, with the mode ‘is_slippery’ set to true, it will be hard for the Q-learning algorithm to actually find an optimal solution.

References

https://www.gymlibrary.dev/environments/toy_text/frozen_lake/. (n.d.). *Frozen Lake*. Retrieved from gymlibrary.

RANA, A. (2018, 09 21). *Introduction: Reinforcement Learning with OpenAI Gym*. Retrieved from towardsdatascience: https://towardsdatascience.com/reinforcement-learning-with-openai-d445c2c687d2