Ex12.ReinforcementLearning.FrozenLakeGame

January 16, 2024

1 12th exercise: First Reinforcement Learning Game (*Frozen Lake*) using OpenAI Gym

- Course: AML
- Lecturer: Gernot Heisenberg
- Author of notebook: Finn Heydemann
- Date: 13.01.2024

GENERAL NOTE 1: Please make sure you are reading the entire notebook, since it contains a lot of information on your tasks (e.g. regarding the set of certain paramaters or a specific computational trick), and the written mark downs as well as comments contain a lot of information on how things work together as a whole.
GENERAL NOTE 2: * Please, when commenting source code, just use English language only. * When describing an observation please use English language, too. * This applies to all exercises throughout this course.

1.0.1 DESCRIPTION:

OpenAI Gym In this exercise you will be using Python and OpenAI Gym to develop your reinforcement learning algorithm. The Gym library is a collection of environments that can be used freely with the reinforcement learning algorithms.

Gym has a ton of environments ranging from simple text based games to Atari games like Breakout and Space Invaders. The library is intuitive to use and simple to install. Just run **pip install gym** and you are good to go! The link to Gym's installation instructions, requirements, and documentation is included in the description.

Further reading about OpenAI Gym is available under https://www.gymlibrary.dev/. This notebook is based on this great post and notebook from Rodolfo Mendes.

Frozen Lake This description of the game is copied directly from Gym's website.

Winter is coming. You and your friends were tossing around a frisbee at the park when you made a wild throw that left the frisbee out in the middle of the lake. The water is mostly frozen, but there are a few holes where the ice has melted. If you step into one of those holes, you'll fall into the freezing water and die (Game over). At this time, there's an international frisbee shortage, so it's absolutely imperative that you navigate across the lake and retrieve the disc. However, the ice is slippery, so you won't always move in the direction you intend. The surface is described using a grid like the following:

- SFFF
- FHFH
- FFFH
- HFFG

This grid is your environment! S is your (the agent's) starting point and it's safe. F represents the frozen surface and is also safe. H represents a hole and if your agent steps in a hole in the middle of a frozen lake, the game is over because the agent dies. Finally, G represents the goal, which is the space on the grid where the frisbee is located.

The agent can navigate *left, right, up, down* and the episode ends when the agent reaches the goal or falls in a hole. It receives a reward of **1** if it reaches the goal and **0** otherwise.

Here is the summary:

1.0.2 TASKS:

The tasks that you need to work on within this notebook are always indicated below as bullet points. If a task is more challenging and consists of several steps, this is indicated as well. Make sure you have worked down the task list and commented your doings. This should be done by using markdown. Make sure you don't forget to specify your name and your matriculation number in the notebook.

YOUR TASKS in this exercise are as follows: 1. import the notebook to Google Colab or use your local machine. 2. make sure you specified you name and your matriculation number in the header below my name and date. * set the date too and remove mine. 3. read the entire notebook carefully * add comments whereever you feel it necessary for better understanding * run the notebook for the first time. 4. install gym into your env! 5. You will train an agent to play the *Frozen Lake* game using Q-learning and you will get a playback of how the agent does after being trained. 6. Again the task: Your agent has to navigate the grid by staying on the frozen surface without falling into any holes until it reaches the frisbee. If it reaches the frisbee, it wins with a reward of plus one. If it falls in a hole, it loses and receives no points for the entire episode. 7. Your tasks are highlighted in the notebook (see below) ———————————

1.0.3 Imports

import all important libs including gym

[1]:
```
import numpy as np
import gym
import random
```

```
import time
from   IPython.display import clear_output
```

[2]:
```
print(gym.__version__)
```

```
0.26.2
```

1.0.4 Creating the Environment

For creating your environment, just call *gym.make()* and pass a string of the name of the environment you want to set up. All the environments with their corresponding names you can use here are available on Gym's website (see above). With this *env* object, you are able to query for information about the environment, sample states and actions, retrieve rewards and have your agent navigate the frozen lake. That is all made available to you conveniently with Gym.

[3]:
```
env = gym.make("FrozenLake-v1", is_slippery=False, render_mode="ansi")
env.reset()
# plt.imshow(env.render())
print(env.render())
```

```
SFFF
FHFH
FFFH
HFFG
```

[4]:
```
env.step(2)
print(env.render())
```

```
  (Right)
SFFF
FHFH
FFFH
HFFG
```

```
/home/finn/Documents/python-venvs/aml/lib/python3.10/site-
packages/gym/utils/passive_env_checker.py:233: DeprecationWarning: `np.bool8` is
a deprecated alias for `np.bool_`.  (Deprecated NumPy 1.24)
  if not isinstance(terminated, (bool, np.bool8)):
```

1.0.5 Creating the Q-Table

Now, construct your Q-table, and initialize all the Q-values to zero for each state-action pair. The number of rows in the table is equivalent to the size of the state space in the environment, and the number of columns is equivalent to the size of the action space (see above). You can get this information using *env.observation_space.n* and *env.action_space.n* as shown below in the code. Then, you can use this information to build the Q-table and initialize it with zeros.

[5]:
```
env.reset()
action_space_size = env.action_space.n
state_space_size = env.observation_space.n

q_table = np.zeros((state_space_size, action_space_size))
```

[6]:
```
print(q_table)
```

```
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

1.0.6 Initializing Q-Learning hyperparameters

Now, we're going to create and initialize all the parameters needed to implement the Q-learning algorithm.

First, with *num_episodes*, you define the total number of episodes you want the agent to play during training. Then, with *max_steps_per_episode*, you define a maximum number of steps that your agent is allowed to take within a single episode. So, if by the 100th step, the agent has not reached the frisbee or fallen through a hole, then the episode will terminate with the agent receiving zero points.

Next, you will set your *learning_rate* and your *discount_rate* as well, which was represented with the symbol (lambda (I think he means gamma)) in the course slides (keyword: discounted return G_t).

Now, the last four parameters are all related to the exploration-exploitation dilemma with respect to the epsilon-greedy policy. You are initializing your *exploration_rate* to **1** and setting the *max_exploration_rate* to **1** and a *min_exploration_rate* to **0.01**. The *max* and *min* are just bounds to how large or small your exploration rate can be. Remember, the exploration rate was represented with the symbol (epsilon) when discussed in the course slides.

Lastly, you will set the *exploration_decay_rate* to **0.01** to determine the rate at which the *exploration_rate* will decay.

YOUR TASK in this exercise is as follows (point 7 from the task list above):

All of the above parameters can change! Your task is to create a *testplan* and tune all parameters by yourself and observe how they influence and change the performance of the algorithm. Make notes! They will help you during the exam.

[7]:
```
def reinforcement_learning(env,
                           n_episodes: int,
                           max_steps_per_episode: int,
                           epsilon: float,
                           epsilon_bounds: dict[str, float],
                           epsilon_decay_rate:float,
                           gamma: float,
                           alpha: float):
    rewards = []
    q_table = np.zeros((env.observation_space.n, env.action_space.n))
    for episode in range(n_episodes):
        state, _ = env.reset()
        done = False
        i = 0
        cur_reward = 0
        for step in range(max_steps_per_episode):
            if np.random.uniform(0, 1) < epsilon:
                # print("exploring")
                action = env.action_space.sample()
            else:
                # print("exploting")
                action = np.argmax(q_table[state, :])
            new_state, reward, done, *_ = env.step(action)
            q_table[state, action] = q_table[state, action] * (1 - alpha) +␣
 ↪alpha * (reward + gamma * np.max(q_table[new_state, :]))
            cur_reward += reward
            state = new_state
            i += 1
            if done:
                break
        rewards.append(cur_reward)

        epsilon = epsilon_bounds["lower"] + (epsilon_bounds["upper"] -␣
 ↪epsilon_bounds["lower"]) * np.exp(-epsilon_decay_rate * episode)
    return q_table, np.array(rewards)


q_table, rewards = reinforcement_learning(env, 10000, 100, 1.0, {"upper": 1,␣
 ↪"lower": .01}, .002, .99, 0.2)
q_table
```

[7]:
```
array([[0.94148015, 0.93206535, 0.95099005, 0.94148015],
       [0.94148015, 0.        , 0.96059601, 0.95099005],
```

```
       [0.95099005, 0.970299  , 0.95099005, 0.96059601],
       [0.96059601, 0.        , 0.79622506, 0.7136787 ],
       [0.92382647, 0.82375843, 0.        , 0.94148015],
       [0.        , 0.        , 0.        , 0.        ],
       [0.        , 0.9801    , 0.        , 0.96059601],
       [0.        , 0.        , 0.        , 0.        ],
       [0.21606309, 0.        , 0.55483903, 0.90821052],
       [0.71729901, 0.98009968, 0.75069972, 0.        ],
       [0.97029425, 0.99      , 0.        , 0.97029894],
       [0.        , 0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        ],
       [0.        , 0.63003475, 0.99      , 0.78422554],
       [0.98009998, 0.99      , 1.        , 0.9801    ],
       [0.        , 0.        , 0.        , 0.        ]])
```

[8]:
```
# optimal route

done = False
current_idx = 0
env.reset()
while not done:
    print(env.render())
    step = np.argmax(q_table[current_idx, :])
    current_idx, _, done, *_ = env.step(step)
    print(current_idx)
print(env.render())
```

```
SFFF
FHFH
FFFH
HFFG

1
  (Right)
SFFF
FHFH
FFFH
HFFG

2
  (Right)
SFFF
FHFH
FFFH
HFFG
```

```
6
  (Down)
SFFF
FHFH
FFFH
HFFG

10
  (Down)
SFFF
FHFH
FFFH
HFFG

14
  (Down)
SFFF
FHFH
FFFH
HFFG

15
  (Right)
SFFF
FHFH
FFFH
HFFG
```

[9]:
```
def rewards_every_n_episodes(rewards: np.ndarray, n: int):
    return rewards.reshape(-1, n).mean(axis=1)

rewards_every_n_episodes(rewards, 1000)
```

[9]:
```
array([0.505, 0.933, 0.98 , 0.991, 0.989, 0.989, 0.994, 0.989, 0.99 ,
       0.994])
```

Add a bit of randomness to the environment: Turn on slippery as the default

[10]:
```
env = gym.make("FrozenLake-v1", is_slippery=True, render_mode="ansi")
env.reset()

q_table, rewards = reinforcement_learning(env, 10_000, 1000, 1.0, {"upper": 1,␣
 ↪"lower": .01}, .002, .99, 0.2)
q_table
```

[10]:
```
array([[0.60200805, 0.49721   , 0.48048125, 0.47965473],
       [0.41983964, 0.25748426, 0.28986177, 0.50794081],
       [0.38611611, 0.3734549 , 0.33346012, 0.46328536],
       [0.27586325, 0.23659618, 0.28121685, 0.42768841],
       [0.6211513 , 0.44383977, 0.41258853, 0.44925727],
       [0.        , 0.        , 0.        , 0.        ],
       [0.16889485, 0.07076191, 0.37630933, 0.07650399],
       [0.        , 0.        , 0.        , 0.        ],
       [0.33525618, 0.49648703, 0.5192195 , 0.6813729 ],
       [0.46430056, 0.71032419, 0.45721522, 0.39744875],
       [0.72657169, 0.30664125, 0.27422158, 0.15825209],
       [0.        , 0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        ],
       [0.58235281, 0.36363441, 0.83539847, 0.47411973],
       [0.64424585, 0.90796894, 0.64789038, 0.69947259],
       [0.        , 0.        , 0.        , 0.        ]])
```

[11]:
```
rewards_every_n_episodes(rewards, 1000)
```

[11]:
```
array([0.123, 0.477, 0.709, 0.736, 0.734, 0.708, 0.77 , 0.729, 0.72 ,
       0.737])
```

[12]:
```
## change some hyperparameters

for gamma in [0.2, 0.5, 0.9, 0.99, 0.999999]:
    _, rewards = reinforcement_learning(env=env,
                                        n_episodes=10_000,
                                        max_steps_per_episode=100,
                                        epsilon=1.0,
                                        epsilon_bounds={"upper": 1, "lower": .
 ↪01},
                                        epsilon_decay_rate=.002,
                                        gamma=gamma,
                                        alpha=0.2)
    rewards1000 = rewards_every_n_episodes(rewards, 1000)
    print(rewards1000)
```

```
[0.051 0.05  0.072 0.049 0.074 0.058 0.059 0.106 0.069 0.079]
[0.04  0.071 0.065 0.137 0.131 0.13  0.136 0.051 0.088 0.106]
[0.075 0.275 0.409 0.443 0.452 0.49  0.486 0.395 0.465 0.507]
[0.109 0.471 0.613 0.677 0.672 0.67  0.646 0.671 0.655 0.68 ]
[0.108 0.129 0.074 0.042 0.015 0.04  0.013 0.017 0.019 0.017]
```

[13]:
```
def hyperparamter_finding(env, **hyperparameters):
    n_params = np.array(np.meshgrid(*hyperparameters.values())).T.reshape(-1,␣
 ↪len(hyperparameters))
    for param_set in n_params:
```

```
        print(dict(zip(hyperparameters.keys(), param_set)))
        q_table, rewards = reinforcement_learning(env, *param_set)
        print(rewards_every_n_episodes(rewards, 1000))


hyperparamter_finding(env,
                      n_episodes=(10_000, 50_000),
                      max_steps_per_episode=100,
                      epsilon=1,
                      epsilon_bounds={"upper": 1, "lower": .01},
                      epsilon_decay_rate=(0.0005, 0.001, 0.002, 0.2),
                      gamma=(0.2, 0.5, 0.9, 0.99),
                      alpha=(0.05, 0.1, 0.2, 0.5))
```

```
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.0005,
'gamma': 0.2, 'alpha': 0.05}
[0.02  0.029 0.048 0.049 0.055 0.06  0.039 0.055 0.063 0.071]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.0005,
'gamma': 0.2, 'alpha': 0.05}
[0.016 0.024 0.047 0.055 0.051 0.047 0.054 0.053 0.037 0.043 0.083 0.082
 0.105 0.052 0.056 0.068 0.033 0.055 0.088 0.101 0.063 0.043 0.045 0.035
 0.038 0.035 0.05  0.035 0.038 0.034 0.052 0.043 0.034 0.037 0.03  0.058
 0.077 0.117 0.089 0.101 0.109 0.123 0.033 0.063 0.088 0.102 0.052 0.049
 0.064 0.069]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.001,
'gamma': 0.2, 'alpha': 0.05}
[0.027 0.063 0.06  0.052 0.035 0.057 0.098 0.068 0.085 0.073]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.001,
'gamma': 0.2, 'alpha': 0.05}
[0.021 0.038 0.047 0.045 0.065 0.083 0.066 0.062 0.092 0.078 0.089 0.072
 0.061 0.134 0.091 0.1   0.064 0.037 0.035 0.076 0.052 0.088 0.104 0.079
 0.039 0.035 0.037 0.048 0.086 0.133 0.098 0.135 0.055 0.048 0.089 0.098
 0.035 0.051 0.094 0.1   0.069 0.063 0.062 0.059 0.136 0.045 0.033 0.042
 0.033 0.068]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.002,
'gamma': 0.2, 'alpha': 0.05}
[0.048 0.081 0.041 0.05  0.055 0.044 0.051 0.065 0.078 0.053]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.002,
'gamma': 0.2, 'alpha': 0.05}
[0.026 0.057 0.075 0.091 0.117 0.07  0.066 0.055 0.06  0.064 0.034 0.062
 0.056 0.043 0.052 0.049 0.056 0.052 0.06  0.052 0.042 0.08  0.095 0.037
```

```
 0.086 0.062 0.037 0.056 0.06  0.075 0.084 0.057 0.052 0.061 0.052 0.065
 0.05  0.065 0.077 0.037 0.037 0.074 0.087 0.084 0.058 0.058 0.072 0.067
 0.058 0.037]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.2,
'gamma': 0.2, 'alpha': 0.05}
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.2,
'gamma': 0.2, 'alpha': 0.05}
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0.]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.0005,
'gamma': 0.5, 'alpha': 0.05}
[0.02  0.053 0.053 0.083 0.078 0.108 0.149 0.119 0.171 0.138]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.0005,
'gamma': 0.5, 'alpha': 0.05}
[0.03  0.044 0.057 0.057 0.056 0.059 0.08  0.097 0.099 0.092 0.085 0.084
 0.119 0.085 0.123 0.176 0.189 0.161 0.136 0.182 0.118 0.166 0.116 0.146
 0.081 0.079 0.103 0.166 0.173 0.149 0.156 0.184 0.168 0.149 0.172 0.186
 0.167 0.111 0.101 0.109 0.119 0.137 0.107 0.128 0.173 0.224 0.135 0.1
 0.099 0.049]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.001,
'gamma': 0.5, 'alpha': 0.05}
[0.024 0.063 0.088 0.068 0.099 0.201 0.093 0.141 0.111 0.111]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.001,
'gamma': 0.5, 'alpha': 0.05}
[0.024 0.063 0.058 0.089 0.209 0.12  0.098 0.067 0.107 0.136 0.139 0.097
 0.116 0.149 0.104 0.151 0.191 0.073 0.08  0.087 0.062 0.168 0.112 0.116
 0.083 0.154 0.172 0.078 0.167 0.245 0.185 0.195 0.173 0.126 0.144 0.12
 0.123 0.154 0.097 0.175 0.196 0.21  0.132 0.079 0.095 0.079 0.029 0.063
 0.171 0.208]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.002,
'gamma': 0.5, 'alpha': 0.05}
[0.033 0.067 0.089 0.08  0.101 0.12  0.127 0.083 0.081 0.068]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.002,
'gamma': 0.5, 'alpha': 0.05}
[0.046 0.064 0.068 0.067 0.132 0.087 0.084 0.147 0.142 0.113 0.098 0.085
 0.189 0.121 0.136 0.211 0.172 0.099 0.103 0.092 0.141 0.137 0.272 0.099
 0.119 0.11  0.091 0.082 0.14  0.106 0.128 0.175 0.26  0.201 0.206 0.308
 0.143 0.145 0.109 0.109 0.124 0.135 0.169 0.106 0.194 0.217 0.183 0.176
```

```
 0.215 0.093]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.2,
'gamma': 0.5, 'alpha': 0.05}
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.2,
'gamma': 0.5, 'alpha': 0.05}
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0.]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.0005,
'gamma': 0.9, 'alpha': 0.05}
[0.029 0.053 0.153 0.196 0.257 0.329 0.367 0.497 0.442 0.481]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.0005,
'gamma': 0.9, 'alpha': 0.05}
[0.019 0.055 0.119 0.211 0.331 0.356 0.406 0.517 0.425 0.558 0.583 0.626
 0.474 0.549 0.518 0.54  0.568 0.519 0.587 0.576 0.464 0.526 0.557 0.602
 0.422 0.474 0.517 0.568 0.595 0.43  0.684 0.479 0.569 0.624 0.468 0.564
 0.498 0.559 0.586 0.581 0.559 0.567 0.613 0.554 0.431 0.658 0.523 0.53
 0.54  0.563]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.001,
'gamma': 0.9, 'alpha': 0.05}
[0.038 0.146 0.359 0.415 0.454 0.508 0.544 0.593 0.582 0.542]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.001,
'gamma': 0.9, 'alpha': 0.05}
[0.027 0.11  0.357 0.414 0.504 0.551 0.57  0.517 0.473 0.564 0.535 0.608
 0.54  0.549 0.512 0.589 0.635 0.624 0.542 0.58  0.584 0.502 0.453 0.644
 0.639 0.517 0.616 0.586 0.602 0.566 0.635 0.518 0.619 0.531 0.578 0.582
 0.597 0.585 0.622 0.515 0.609 0.439 0.458 0.53  0.622 0.56  0.556 0.675
 0.57  0.568]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.002,
'gamma': 0.9, 'alpha': 0.05}
[0.008 0.02  0.024 0.042 0.053 0.074 0.072 0.099 0.136 0.115]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.002,
'gamma': 0.9, 'alpha': 0.05}
[0.07  0.384 0.582 0.558 0.567 0.51  0.607 0.559 0.547 0.57  0.652 0.573
 0.457 0.479 0.545 0.539 0.575 0.585 0.607 0.571 0.647 0.499 0.443 0.609
 0.469 0.584 0.507 0.562 0.501 0.477 0.581 0.571 0.617 0.576 0.574 0.576
 0.457 0.553 0.53  0.603 0.525 0.498 0.537 0.488 0.474 0.587 0.566 0.597
 0.554 0.443]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
```

```
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.2,
'gamma': 0.9, 'alpha': 0.05}
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.2,
'gamma': 0.9, 'alpha': 0.05}
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0.]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.0005,
'gamma': 0.99, 'alpha': 0.05}
[0.02  0.072 0.164 0.253 0.378 0.463 0.518 0.597 0.629 0.615]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.0005,
'gamma': 0.99, 'alpha': 0.05}
[0.023 0.054 0.15  0.28  0.375 0.49  0.546 0.611 0.621 0.643 0.674 0.72
 0.675 0.679 0.695 0.708 0.705 0.694 0.677 0.707 0.689 0.673 0.697 0.685
 0.686 0.694 0.701 0.661 0.699 0.689 0.697 0.675 0.688 0.701 0.684 0.702
 0.686 0.69  0.694 0.697 0.708 0.65  0.671 0.705 0.698 0.68  0.686 0.698
 0.697 0.685]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.001,
'gamma': 0.99, 'alpha': 0.05}
[0.037 0.134 0.422 0.566 0.663 0.673 0.667 0.698 0.689 0.703]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.001,
'gamma': 0.99, 'alpha': 0.05}
[0.059 0.188 0.414 0.538 0.619 0.667 0.642 0.683 0.68  0.698 0.717 0.671
 0.675 0.683 0.71  0.687 0.685 0.732 0.66  0.687 0.693 0.717 0.69  0.68
 0.715 0.685 0.665 0.711 0.678 0.695 0.691 0.69  0.679 0.702 0.705 0.703
 0.698 0.686 0.7   0.686 0.69  0.694 0.663 0.689 0.701 0.699 0.695 0.71
 0.672 0.701]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.002,
'gamma': 0.99, 'alpha': 0.05}
[0.038 0.099 0.289 0.434 0.559 0.597 0.544 0.653 0.673 0.641]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.002,
'gamma': 0.99, 'alpha': 0.05}
[0.036 0.102 0.152 0.152 0.141 0.165 0.19  0.264 0.293 0.384 0.43  0.402
 0.45  0.678 0.706 0.692 0.698 0.676 0.688 0.677 0.706 0.715 0.679 0.712
 0.695 0.707 0.718 0.711 0.723 0.706 0.691 0.684 0.693 0.675 0.709 0.694
 0.709 0.697 0.67  0.688 0.709 0.699 0.694 0.702 0.684 0.689 0.688 0.689
 0.685 0.716]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.2,
'gamma': 0.99, 'alpha': 0.05}
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.2,
'gamma': 0.99, 'alpha': 0.05}
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0.]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.0005,
'gamma': 0.2, 'alpha': 0.1}
[0.022 0.03  0.037 0.058 0.043 0.048 0.04  0.085 0.073 0.058]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.0005,
'gamma': 0.2, 'alpha': 0.1}
[0.02  0.026 0.046 0.058 0.055 0.067 0.05  0.066 0.057 0.077 0.051 0.069
 0.065 0.067 0.065 0.05  0.066 0.044 0.043 0.037 0.036 0.064 0.071 0.103
 0.099 0.078 0.053 0.059 0.059 0.05  0.101 0.055 0.057 0.05  0.061 0.028
 0.039 0.07  0.069 0.037 0.044 0.048 0.063 0.073 0.048 0.043 0.043 0.076
 0.056 0.064]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.001,
'gamma': 0.2, 'alpha': 0.1}
[0.026 0.046 0.051 0.063 0.06  0.062 0.054 0.042 0.046 0.054]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.001,
'gamma': 0.2, 'alpha': 0.1}
[0.024 0.048 0.052 0.049 0.057 0.06  0.089 0.064 0.077 0.062 0.065 0.099
 0.05  0.069 0.117 0.082 0.082 0.095 0.064 0.068 0.091 0.063 0.036 0.061
 0.063 0.032 0.064 0.062 0.051 0.044 0.033 0.032 0.057 0.052 0.038 0.043
 0.044 0.038 0.035 0.047 0.07  0.061 0.056 0.093 0.055 0.065 0.057 0.116
 0.048 0.067]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.002,
'gamma': 0.2, 'alpha': 0.1}
[0.037 0.063 0.087 0.041 0.073 0.075 0.043 0.055 0.041 0.05 ]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.002,
'gamma': 0.2, 'alpha': 0.1}
[0.039 0.062 0.04  0.044 0.054 0.041 0.038 0.037 0.031 0.042 0.037 0.035
 0.053 0.061 0.032 0.029 0.04  0.066 0.072 0.113 0.072 0.072 0.057 0.059
 0.073 0.089 0.061 0.076 0.095 0.054 0.092 0.069 0.069 0.063 0.055 0.057
 0.05  0.061 0.054 0.06  0.068 0.084 0.065 0.064 0.035 0.07  0.076 0.05
 0.049 0.059]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.2,
'gamma': 0.2, 'alpha': 0.1}
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
```

```
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.2,
'gamma': 0.2, 'alpha': 0.1}
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0.]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.0005,
'gamma': 0.5, 'alpha': 0.1}
[0.014 0.035 0.037 0.051 0.091 0.091 0.092 0.051 0.075 0.106]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.0005,
'gamma': 0.5, 'alpha': 0.1}
[0.026 0.038 0.048 0.053 0.064 0.095 0.113 0.104 0.107 0.099 0.115 0.116
 0.164 0.209 0.08  0.149 0.145 0.111 0.059 0.106 0.138 0.102 0.157 0.091
 0.116 0.088 0.106 0.133 0.138 0.093 0.106 0.149 0.091 0.065 0.061 0.096
 0.084 0.068 0.097 0.089 0.086 0.112 0.105 0.089 0.106 0.091 0.11  0.173
 0.121 0.197]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.001,
'gamma': 0.5, 'alpha': 0.1}
[0.037 0.053 0.108 0.087 0.116 0.075 0.081 0.105 0.153 0.12 ]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.001,
'gamma': 0.5, 'alpha': 0.1}
[0.032 0.054 0.078 0.06  0.14  0.154 0.135 0.134 0.147 0.098 0.068 0.099
 0.121 0.14  0.097 0.09  0.11  0.128 0.155 0.086 0.166 0.1   0.099 0.065
 0.071 0.086 0.094 0.084 0.109 0.08  0.113 0.116 0.187 0.084 0.162 0.14
 0.16  0.164 0.152 0.139 0.223 0.076 0.068 0.066 0.155 0.164 0.076 0.111
 0.078 0.092]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.002,
'gamma': 0.5, 'alpha': 0.1}
[0.047 0.072 0.095 0.077 0.095 0.092 0.057 0.128 0.159 0.188]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.002,
'gamma': 0.5, 'alpha': 0.1}
[0.025 0.082 0.083 0.125 0.119 0.074 0.135 0.2   0.133 0.079 0.152 0.142
 0.157 0.109 0.113 0.148 0.177 0.084 0.13  0.108 0.12  0.147 0.129 0.18
 0.197 0.173 0.081 0.082 0.074 0.135 0.077 0.097 0.039 0.14  0.077 0.113
 0.067 0.087 0.127 0.107 0.136 0.087 0.084 0.052 0.121 0.196 0.132 0.134
 0.173 0.124]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.2,
'gamma': 0.5, 'alpha': 0.1}
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.2,
'gamma': 0.5, 'alpha': 0.1}
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0.]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.0005,
'gamma': 0.9, 'alpha': 0.1}
[0.032 0.065 0.122 0.201 0.244 0.292 0.328 0.367 0.462 0.487]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.0005,
'gamma': 0.9, 'alpha': 0.1}
[0.017 0.067 0.145 0.184 0.264 0.299 0.365 0.458 0.379 0.444 0.546 0.472
 0.494 0.542 0.578 0.451 0.485 0.44  0.498 0.472 0.481 0.424 0.525 0.444
 0.468 0.517 0.56  0.389 0.569 0.475 0.517 0.511 0.429 0.496 0.504 0.537
 0.451 0.504 0.438 0.482 0.452 0.528 0.504 0.496 0.533 0.469 0.526 0.445
 0.54  0.471]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.001,
'gamma': 0.9, 'alpha': 0.1}
[0.04  0.138 0.238 0.39  0.424 0.438 0.418 0.465 0.601 0.411]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.001,
'gamma': 0.9, 'alpha': 0.1}
[0.037 0.145 0.309 0.415 0.507 0.48  0.408 0.542 0.563 0.526 0.507 0.477
 0.421 0.553 0.492 0.459 0.522 0.621 0.615 0.468 0.559 0.481 0.513 0.432
 0.523 0.539 0.433 0.572 0.515 0.527 0.494 0.474 0.393 0.501 0.517 0.519
 0.437 0.504 0.432 0.473 0.567 0.545 0.471 0.606 0.534 0.506 0.461 0.49
 0.565 0.506]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.002,
'gamma': 0.9, 'alpha': 0.1}
[0.112 0.335 0.352 0.459 0.456 0.475 0.548 0.543 0.468 0.459]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.002,
'gamma': 0.9, 'alpha': 0.1}
[0.093 0.331 0.481 0.489 0.5   0.597 0.45  0.471 0.459 0.576 0.574 0.417
 0.471 0.5   0.504 0.452 0.475 0.464 0.58  0.485 0.537 0.447 0.489 0.517
 0.505 0.512 0.501 0.433 0.549 0.539 0.479 0.483 0.485 0.526 0.454 0.558
 0.499 0.547 0.527 0.492 0.554 0.505 0.523 0.507 0.492 0.497 0.466 0.405
 0.492 0.368]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.2,
'gamma': 0.9, 'alpha': 0.1}
[0.001 0.    0.    0.    0.    0.    0.    0.    0.    0.   ]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.2,
'gamma': 0.9, 'alpha': 0.1}
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

```
 0. 0.]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.0005,
'gamma': 0.99, 'alpha': 0.1}
[0.018 0.081 0.162 0.235 0.366 0.467 0.527 0.596 0.595 0.665]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.0005,
'gamma': 0.99, 'alpha': 0.1}
[0.028 0.074 0.139 0.269 0.362 0.488 0.515 0.595 0.635 0.637 0.673 0.651
 0.685 0.666 0.658 0.663 0.651 0.684 0.707 0.692 0.687 0.695 0.688 0.699
 0.655 0.681 0.671 0.676 0.669 0.669 0.706 0.675 0.702 0.687 0.722 0.694
 0.684 0.678 0.693 0.695 0.684 0.706 0.698 0.66  0.669 0.694 0.701 0.69
 0.709 0.67 ]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.001,
'gamma': 0.99, 'alpha': 0.1}
[0.044 0.189 0.414 0.524 0.638 0.691 0.677 0.677 0.698 0.677]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.001,
'gamma': 0.99, 'alpha': 0.1}
[0.044 0.189 0.413 0.574 0.639 0.663 0.676 0.687 0.687 0.69  0.682 0.693
 0.658 0.686 0.666 0.678 0.696 0.665 0.668 0.692 0.689 0.663 0.691 0.678
 0.705 0.69  0.685 0.706 0.675 0.694 0.699 0.707 0.689 0.696 0.663 0.69
 0.687 0.691 0.692 0.692 0.677 0.676 0.68  0.681 0.718 0.684 0.684 0.702
 0.679 0.69 ]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.002,
'gamma': 0.99, 'alpha': 0.1}
[0.111 0.505 0.618 0.728 0.687 0.685 0.701 0.691 0.691 0.668]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.002,
'gamma': 0.99, 'alpha': 0.1}
[0.065 0.494 0.629 0.647 0.723 0.674 0.686 0.722 0.705 0.685 0.685 0.673
 0.695 0.69  0.648 0.664 0.653 0.688 0.706 0.688 0.693 0.673 0.679 0.723
 0.732 0.677 0.71  0.694 0.688 0.664 0.705 0.682 0.682 0.712 0.701 0.71
 0.669 0.695 0.681 0.667 0.709 0.677 0.663 0.68  0.717 0.704 0.693 0.675
 0.674 0.669]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.2,
'gamma': 0.99, 'alpha': 0.1}
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.2,
'gamma': 0.99, 'alpha': 0.1}
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0.]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
```

```
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.0005,
'gamma': 0.2, 'alpha': 0.2}
[0.022 0.042 0.04  0.044 0.058 0.069 0.082 0.053 0.059 0.036]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.0005,
'gamma': 0.2, 'alpha': 0.2}
[0.02  0.024 0.05  0.043 0.065 0.06  0.064 0.038 0.045 0.051 0.039 0.064
 0.079 0.073 0.048 0.055 0.065 0.087 0.088 0.05  0.033 0.084 0.093 0.058
 0.079 0.068 0.082 0.055 0.061 0.101 0.049 0.051 0.063 0.057 0.058 0.068
 0.044 0.061 0.055 0.039 0.075 0.059 0.053 0.098 0.042 0.072 0.089 0.059
 0.055 0.057]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.001,
'gamma': 0.2, 'alpha': 0.2}
[0.014 0.056 0.049 0.056 0.064 0.1   0.088 0.075 0.099 0.054]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.001,
'gamma': 0.2, 'alpha': 0.2}
[0.028 0.049 0.052 0.053 0.066 0.064 0.044 0.04  0.03  0.048 0.051 0.065
 0.077 0.086 0.079 0.073 0.066 0.064 0.069 0.044 0.058 0.086 0.126 0.077
 0.042 0.081 0.103 0.069 0.052 0.036 0.04  0.093 0.096 0.089 0.047 0.072
 0.042 0.049 0.049 0.101 0.095 0.11  0.094 0.056 0.059 0.056 0.063 0.085
 0.062 0.071]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.002,
'gamma': 0.2, 'alpha': 0.2}
[0.04  0.077 0.089 0.064 0.087 0.036 0.055 0.063 0.071 0.04 ]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.002,
'gamma': 0.2, 'alpha': 0.2}
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0.]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.2,
'gamma': 0.2, 'alpha': 0.2}
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.2,
'gamma': 0.2, 'alpha': 0.2}
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0.]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.0005,
'gamma': 0.5, 'alpha': 0.2}
[0.024 0.033 0.044 0.049 0.053 0.068 0.105 0.108 0.108 0.13 ]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
```

```
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.0005,
'gamma': 0.5, 'alpha': 0.2}
[0.008 0.037 0.058 0.055 0.061 0.052 0.092 0.101 0.106 0.11  0.071 0.154
 0.105 0.09  0.074 0.117 0.188 0.111 0.128 0.134 0.158 0.188 0.073 0.08
 0.084 0.073 0.071 0.08  0.066 0.183 0.107 0.215 0.174 0.085 0.074 0.076
 0.128 0.091 0.095 0.094 0.111 0.148 0.132 0.104 0.141 0.085 0.102 0.108
 0.123 0.106]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.001,
'gamma': 0.5, 'alpha': 0.2}
[0.043 0.059 0.07  0.065 0.098 0.166 0.131 0.052 0.106 0.087]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.001,
'gamma': 0.5, 'alpha': 0.2}
[0.033 0.052 0.063 0.098 0.11  0.087 0.128 0.136 0.08  0.136 0.089 0.051
 0.139 0.068 0.104 0.165 0.115 0.181 0.207 0.078 0.069 0.087 0.139 0.17
 0.106 0.076 0.092 0.128 0.194 0.178 0.122 0.155 0.103 0.122 0.143 0.127
 0.099 0.137 0.103 0.088 0.081 0.078 0.126 0.148 0.113 0.143 0.097 0.125
 0.094 0.112]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.002,
'gamma': 0.5, 'alpha': 0.2}
[0.047 0.07  0.058 0.136 0.094 0.045 0.067 0.157 0.079 0.087]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.002,
'gamma': 0.5, 'alpha': 0.2}
[0.038 0.072 0.161 0.132 0.098 0.079 0.152 0.078 0.134 0.147 0.108 0.123
 0.065 0.1   0.105 0.072 0.079 0.069 0.085 0.061 0.098 0.116 0.13  0.122
 0.071 0.099 0.12  0.074 0.098 0.126 0.15  0.128 0.133 0.157 0.082 0.127
 0.148 0.09  0.095 0.176 0.125 0.107 0.11  0.073 0.098 0.1   0.109 0.136
 0.101 0.098]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.2,
'gamma': 0.5, 'alpha': 0.2}
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.2,
'gamma': 0.5, 'alpha': 0.2}
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0.]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.0005,
'gamma': 0.9, 'alpha': 0.2}
[0.03  0.058 0.089 0.187 0.192 0.202 0.316 0.33  0.402 0.424]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.0005,
'gamma': 0.9, 'alpha': 0.2}
```

```
[0.02  0.043 0.114 0.149 0.205 0.225 0.339 0.355 0.448 0.417 0.455 0.401
 0.417 0.469 0.448 0.483 0.411 0.462 0.536 0.444 0.439 0.47  0.488 0.418
 0.476 0.424 0.472 0.488 0.478 0.475 0.477 0.491 0.493 0.453 0.454 0.456
 0.408 0.454 0.483 0.459 0.451 0.464 0.407 0.476 0.422 0.425 0.449 0.497
 0.459 0.418]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.001,
'gamma': 0.9, 'alpha': 0.2}
[0.041 0.154 0.221 0.326 0.419 0.407 0.4   0.443 0.478 0.395]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.001,
'gamma': 0.9, 'alpha': 0.2}
[0.024 0.142 0.269 0.335 0.403 0.412 0.476 0.452 0.437 0.447 0.462 0.391
 0.441 0.441 0.436 0.458 0.463 0.496 0.399 0.495 0.485 0.452 0.397 0.454
 0.419 0.468 0.484 0.496 0.482 0.434 0.498 0.401 0.453 0.467 0.459 0.357
 0.471 0.462 0.473 0.477 0.453 0.484 0.38  0.404 0.477 0.435 0.394 0.514
 0.498 0.458]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.002,
'gamma': 0.9, 'alpha': 0.2}
[0.085 0.288 0.47  0.374 0.463 0.492 0.493 0.471 0.445 0.5  ]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.002,
'gamma': 0.9, 'alpha': 0.2}
[0.076 0.285 0.363 0.478 0.426 0.464 0.459 0.442 0.433 0.482 0.419 0.467
 0.46  0.406 0.435 0.43  0.498 0.477 0.462 0.46  0.445 0.413 0.434 0.448
 0.513 0.518 0.49  0.493 0.501 0.461 0.414 0.382 0.455 0.443 0.419 0.52
 0.484 0.434 0.397 0.425 0.448 0.47  0.424 0.428 0.473 0.482 0.47  0.477
 0.486 0.483]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.2,
'gamma': 0.9, 'alpha': 0.2}
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.2,
'gamma': 0.9, 'alpha': 0.2}
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0.]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.0005,
'gamma': 0.99, 'alpha': 0.2}
[0.03  0.084 0.157 0.241 0.338 0.416 0.484 0.56  0.581 0.633]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.0005,
'gamma': 0.99, 'alpha': 0.2}
[0.034 0.062 0.155 0.215 0.294 0.44  0.518 0.557 0.637 0.647 0.625 0.635
 0.652 0.626 0.69  0.648 0.663 0.663 0.672 0.671 0.643 0.65  0.662 0.668
```

```
 0.669 0.685 0.666 0.673 0.702 0.69  0.673 0.633 0.656 0.66  0.667 0.66
 0.664 0.676 0.669 0.658 0.675 0.665 0.658 0.649 0.674 0.681 0.646 0.663
 0.647 0.673]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.001,
'gamma': 0.99, 'alpha': 0.2}
[0.038 0.209 0.367 0.536 0.624 0.618 0.662 0.666 0.644 0.673]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.001,
'gamma': 0.99, 'alpha': 0.2}
[0.041 0.18  0.395 0.54  0.608 0.635 0.66  0.663 0.699 0.673 0.682 0.659
 0.66  0.668 0.666 0.671 0.654 0.666 0.657 0.659 0.648 0.691 0.669 0.67
 0.65  0.679 0.713 0.658 0.669 0.666 0.647 0.666 0.658 0.679 0.666 0.688
 0.683 0.66  0.65  0.695 0.637 0.666 0.664 0.628 0.679 0.646 0.66  0.67
 0.655 0.673]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.002,
'gamma': 0.99, 'alpha': 0.2}
[0.136 0.484 0.619 0.669 0.663 0.685 0.675 0.691 0.661 0.645]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.002,
'gamma': 0.99, 'alpha': 0.2}
[0.132 0.477 0.629 0.666 0.664 0.684 0.632 0.668 0.683 0.665 0.67  0.661
 0.688 0.67  0.619 0.652 0.679 0.685 0.632 0.662 0.665 0.665 0.666 0.659
 0.669 0.682 0.655 0.672 0.659 0.687 0.681 0.648 0.663 0.659 0.664 0.677
 0.695 0.644 0.687 0.655 0.66  0.668 0.692 0.666 0.651 0.669 0.697 0.699
 0.667 0.662]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.2,
'gamma': 0.99, 'alpha': 0.2}
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.2,
'gamma': 0.99, 'alpha': 0.2}
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0.]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.0005,
'gamma': 0.2, 'alpha': 0.5}
[0.025 0.023 0.032 0.039 0.058 0.049 0.044 0.056 0.076 0.097]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.0005,
'gamma': 0.2, 'alpha': 0.5}
[0.023 0.023 0.032 0.058 0.046 0.062 0.063 0.073 0.071 0.063 0.093 0.098
 0.07  0.075 0.056 0.088 0.109 0.104 0.08  0.088 0.092 0.122 0.098 0.079
 0.048 0.07  0.06  0.067 0.057 0.076 0.061 0.08  0.077 0.086 0.077 0.056
 0.062 0.057 0.045 0.071 0.059 0.074 0.082 0.104 0.098 0.06  0.079 0.092
```

```
 0.095 0.078]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.001,
'gamma': 0.2, 'alpha': 0.5}
[0.027 0.043 0.052 0.065 0.058 0.071 0.067 0.102 0.048 0.092]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.001,
'gamma': 0.2, 'alpha': 0.5}
[0.018 0.03  0.069 0.055 0.044 0.038 0.068 0.033 0.037 0.105 0.067 0.077
 0.081 0.076 0.095 0.097 0.057 0.081 0.091 0.073 0.086 0.07  0.038 0.058
 0.098 0.076 0.056 0.047 0.083 0.082 0.071 0.07  0.085 0.08  0.124 0.08
 0.1   0.1   0.093 0.08  0.104 0.108 0.064 0.076 0.055 0.071 0.065 0.084
 0.067 0.036]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.002,
'gamma': 0.2, 'alpha': 0.5}
[0.045 0.052 0.055 0.108 0.073 0.069 0.061 0.065 0.061 0.102]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.002,
'gamma': 0.2, 'alpha': 0.5}
[0.016 0.058 0.054 0.081 0.069 0.069 0.071 0.053 0.074 0.051 0.088 0.042
 0.109 0.087 0.072 0.087 0.091 0.065 0.104 0.091 0.053 0.098 0.064 0.07
 0.127 0.078 0.093 0.06  0.062 0.057 0.073 0.072 0.073 0.066 0.055 0.088
 0.077 0.063 0.077 0.076 0.083 0.099 0.056 0.078 0.058 0.085 0.058 0.088
 0.054 0.072]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.2,
'gamma': 0.2, 'alpha': 0.5}
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.2,
'gamma': 0.2, 'alpha': 0.5}
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0.]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.0005,
'gamma': 0.5, 'alpha': 0.5}
[0.014 0.034 0.044 0.059 0.079 0.06  0.064 0.126 0.119 0.116]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.0005,
'gamma': 0.5, 'alpha': 0.5}
[0.024 0.037 0.042 0.042 0.064 0.061 0.089 0.073 0.146 0.103 0.105 0.075
 0.155 0.08  0.125 0.104 0.131 0.068 0.09  0.103 0.108 0.083 0.107 0.221
 0.173 0.122 0.129 0.089 0.126 0.057 0.104 0.162 0.146 0.145 0.176 0.174
 0.129 0.137 0.128 0.15  0.104 0.124 0.105 0.145 0.155 0.129 0.152 0.148
 0.15  0.11 ]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
```

```
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.001,
'gamma': 0.5, 'alpha': 0.5}
[0.027 0.053 0.049 0.105 0.13  0.095 0.124 0.132 0.102 0.12 ]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.001,
'gamma': 0.5, 'alpha': 0.5}
[0.029 0.043 0.079 0.107 0.102 0.126 0.131 0.112 0.16  0.096 0.114 0.084
 0.112 0.126 0.103 0.163 0.076 0.087 0.131 0.081 0.117 0.174 0.146 0.17
 0.166 0.13  0.104 0.117 0.123 0.149 0.123 0.107 0.125 0.118 0.115 0.075
 0.085 0.06  0.088 0.125 0.066 0.119 0.152 0.152 0.137 0.103 0.126 0.191
 0.125 0.156]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.002,
'gamma': 0.5, 'alpha': 0.5}
[0.045 0.087 0.094 0.116 0.118 0.118 0.137 0.178 0.105 0.079]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.002,
'gamma': 0.5, 'alpha': 0.5}
[0.033 0.084 0.086 0.168 0.057 0.14  0.145 0.166 0.134 0.138 0.179 0.119
 0.107 0.168 0.078 0.077 0.126 0.108 0.104 0.136 0.122 0.102 0.139 0.164
 0.121 0.169 0.211 0.181 0.116 0.096 0.169 0.126 0.158 0.095 0.072 0.061
 0.108 0.138 0.1   0.084 0.162 0.143 0.098 0.157 0.09  0.096 0.088 0.138
 0.2   0.087]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.2,
'gamma': 0.5, 'alpha': 0.5}
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.2,
'gamma': 0.5, 'alpha': 0.5}
[0.001 0.    0.    0.    0.    0.    0.    0.041 0.084 0.123 0.098 0.126
 0.13  0.138 0.169 0.113 0.133 0.102 0.185 0.118 0.078 0.103 0.107 0.176
 0.131 0.187 0.106 0.101 0.174 0.171 0.164 0.123 0.119 0.154 0.097 0.08
 0.04  0.125 0.123 0.101 0.161 0.09  0.064 0.139 0.129 0.102 0.139 0.138
 0.149 0.166]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.0005,
'gamma': 0.9, 'alpha': 0.5}
[0.028 0.039 0.074 0.138 0.178 0.229 0.222 0.305 0.341 0.382]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.0005,
'gamma': 0.9, 'alpha': 0.5}
[0.021 0.045 0.063 0.13  0.138 0.195 0.247 0.278 0.339 0.35  0.36  0.402
 0.418 0.449 0.401 0.418 0.421 0.414 0.41  0.409 0.387 0.462 0.416 0.398
 0.419 0.44  0.397 0.49  0.449 0.418 0.45  0.363 0.399 0.398 0.484 0.408
 0.398 0.458 0.476 0.475 0.395 0.398 0.413 0.434 0.451 0.46  0.429 0.405
 0.4   0.413]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
```

```
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.001,
'gamma': 0.9, 'alpha': 0.5}
[0.038 0.099 0.19  0.26  0.374 0.405 0.458 0.457 0.454 0.428]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.001,
'gamma': 0.9, 'alpha': 0.5}
[0.028 0.101 0.183 0.3   0.374 0.394 0.427 0.456 0.394 0.439 0.407 0.41
 0.43  0.413 0.424 0.465 0.412 0.357 0.447 0.452 0.451 0.452 0.444 0.426
 0.417 0.412 0.452 0.374 0.44  0.394 0.437 0.453 0.434 0.417 0.339 0.393
 0.447 0.415 0.453 0.44  0.407 0.402 0.478 0.431 0.376 0.369 0.498 0.411
 0.412 0.36 ]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.002,
'gamma': 0.9, 'alpha': 0.5}
[0.067 0.189 0.356 0.364 0.421 0.456 0.421 0.398 0.365 0.458]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.002,
'gamma': 0.9, 'alpha': 0.5}
[0.05  0.223 0.384 0.466 0.437 0.43  0.406 0.474 0.427 0.387 0.414 0.442
 0.351 0.435 0.444 0.397 0.352 0.424 0.441 0.411 0.432 0.397 0.446 0.462
 0.409 0.443 0.424 0.342 0.465 0.456 0.494 0.426 0.468 0.422 0.475 0.451
 0.349 0.443 0.467 0.471 0.399 0.369 0.496 0.462 0.389 0.486 0.448 0.418
 0.422 0.461]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.2,
'gamma': 0.9, 'alpha': 0.5}
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.2,
'gamma': 0.9, 'alpha': 0.5}
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0.]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.0005,
'gamma': 0.99, 'alpha': 0.5}
[0.028 0.057 0.105 0.171 0.264 0.339 0.394 0.503 0.555 0.543]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.0005,
'gamma': 0.99, 'alpha': 0.5}
[0.02  0.055 0.111 0.144 0.261 0.347 0.433 0.466 0.472 0.556 0.603 0.598
 0.582 0.617 0.601 0.62  0.615 0.54  0.6   0.622 0.588 0.609 0.646 0.643
 0.604 0.596 0.6   0.651 0.628 0.61  0.569 0.564 0.582 0.594 0.613 0.59
 0.628 0.635 0.63  0.611 0.629 0.617 0.59  0.604 0.6   0.644 0.583 0.621
 0.571 0.594]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.001,
'gamma': 0.99, 'alpha': 0.5}
```

```
[0.047 0.153 0.303 0.413 0.538 0.555 0.581 0.597 0.626 0.614]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.001,
'gamma': 0.99, 'alpha': 0.5}
[0.037 0.144 0.329 0.463 0.535 0.609 0.524 0.582 0.617 0.587 0.568 0.623
 0.599 0.577 0.614 0.579 0.593 0.588 0.617 0.603 0.62  0.624 0.58  0.599
 0.597 0.644 0.614 0.603 0.621 0.597 0.564 0.587 0.607 0.594 0.584 0.614
 0.611 0.625 0.589 0.629 0.6   0.6   0.614 0.592 0.589 0.611 0.578 0.603
 0.56  0.605]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.002,
'gamma': 0.99, 'alpha': 0.5}
[0.092 0.365 0.526 0.618 0.588 0.629 0.641 0.601 0.613 0.611]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.002,
'gamma': 0.99, 'alpha': 0.5}
[0.075 0.385 0.496 0.6   0.594 0.616 0.604 0.611 0.634 0.608 0.574 0.608
 0.604 0.579 0.617 0.623 0.631 0.635 0.549 0.611 0.594 0.59  0.615 0.601
 0.634 0.588 0.592 0.596 0.643 0.58  0.574 0.613 0.595 0.603 0.595 0.635
 0.632 0.588 0.604 0.601 0.584 0.586 0.629 0.575 0.561 0.6   0.619 0.612
 0.585 0.611]
{'n_episodes': 10000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.2,
'gamma': 0.99, 'alpha': 0.5}
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
{'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1,
'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.2,
'gamma': 0.99, 'alpha': 0.5}
[0.001 0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
 0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
 0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
 0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
 0.    0.   ]
```

1.1 Results:

When Epsilon Decay Rate is too high the agent will exploit too much and does'nt do enough exploring. He then isn't able to make it to the end, because he turns upwards or left at the beginning. If Epsilon decay rate is too low the episodes are not enough to get a prober result because the agent will not get towrds exploiting enough but is still exploring.

The highest average rewards is received in the last 1000 episodes with the following hyperparamters: {'n_episodes': 50000, 'max_steps_per_episode': 100, 'epsilon': 1, 'epsilon_bounds': {'upper': 1, 'lower': 0.01}, 'epsilon_decay_rate': 0.002, 'gamma': 0.99, 'alpha': 0.05}. Here we have a low learning rate meaning that the q-table isn't updated very quickly. Also we have a pretty low epsilon_decay_rate meaning that the agent will lean towards exploting pretty late and a high gamma which values future rewards high. Plus many episodes.

[ ]:

Create a list to hold all of the rewards you will get from each episode. By means of this you can observe how your game score changes over time.

[14]:
```
num_episodes = 10000
max_steps_per_episode = 100

learning_rate = 0.2
discount_rate = 0.99

exploration_rate = 1
max_exploration_rate = 1
min_exploration_rate = 0.01
exploration_decay_rate = 0.005

rewards_all_episodes = []
```

In the following code section, the entire Q-learning algorithm is implemented as discussed in detail in the AML course. When this code is executed, this is exactly where the training will take place. * The first for-loop contains everything that happens within a single episode. * The second nested loop contains everything that happens for a single time-step.

Read all the red comments, as they contain lots of important information on the implementation.

[15]:
```
# Q-learning algorithm

# loop: for a single episode
for episode in range(num_episodes):
    # initialize 'new episode' parameters
    state, info = env.reset()
    ''' The done variable just keeps track of whether or not your episode is␣
 ↪finished.
    Initialize it to False when first starting the episode and you will see␣
 ↪later where
    it will get updated to notify you when the episode is over.'''
    done = False

    ''' Keep track of the rewards within the current episode as well.
    Hence, set rewards_current_episode = 0 since you start
    with no rewards at the beginning of each episode.'''
    rewards_current_episode = 0

    # nested loop: for a single time-step
    for step in range(max_steps_per_episode):
        # Exploration-exploitation trade-off
        '''For each time-step within an episode set your␣
 ↪exploration_rate_threshold
```

```
        to a random number between 0 and 1. This will be used to determine␣
↪whether
        your agent will explore or exploit the environment in this time-step.'''
        if random.uniform(0, 1) > exploration_rate:
            action = np.argmax(q_table[state,:])
        else:
            action = env.action_space.sample()

        # Take new action
        '''After action is chosen, take that action by calling step() on your␣
↪env object and
        pass your action to it. The function step() returns a tuple containing␣
↪the new state,
        the reward for the action you took, whether or not the action ended the␣
↪episode and
        diagnostic information regarding the environment (helpful for␣
↪debugging).'''
        new_state, reward, done, truncated, info = env.step(action)

        # Update Q-table for Q(s,a)
        '''Compare this implementation with the equation in the course slides.
↪'''
        q_table[state, action] = q_table[state, action] * (1 - learning_rate) +␣
↪learning_rate * (reward + discount_rate * np.max(q_table[new_state, :]))

        '''Set your current state to the new_state that was returned when␣
↪taking the last action
        and then update the rewards from your current episode by adding the␣
↪reward you received
        for your previous action.'''
        # Set new state
        state = new_state
        # Add new reward
        rewards_current_episode += reward
        '''Then, check to see if your last action ended the episode
        (game over by agent stepping in a hole or reaching the goal)!
        If the action did end the episode, then jump out of this loop and start␣
↪the next episode.
        Otherwise, transition to the next time-step.'''
        if done:
            break

    # Exploration rate decay
    '''Once an episode is finished, you need to update your exploration_rate␣
↪using exponential decay,
```

```
    which just means that the exploration rate decays at a rate proportional to␣
 ↪its current value.
    You can decay the exploration_rate using the formula above, which makes use␣
 ↪of all the exploration
    rate parameters that were defined above in the hyperparameter section.'''

    exploration_rate = min_exploration_rate + (max_exploration_rate -␣
 ↪min_exploration_rate) * np.exp(-exploration_decay_rate*episode)

    # Add current episode reward to total rewards list and move on to the next␣
 ↪episode
    rewards_all_episodes.append(rewards_current_episode)

np.array(rewards_all_episodes)
```

[15]:
```
array([0., 0., 0., …, 1., 0., 1.])
```

1.1.1 All episodes training completed

After all episodes are finished you now just calculate the average reward per thousand episodes from your list that contains the rewards for all episodes so that you can print it out and see how the rewards changed over time.

[16]:
```
# Calculate and print the average reward per thousand episodes
rewards_per_thousand_episodes = np.split(np.
 ↪array(rewards_all_episodes),num_episodes/1000)
count = 1000

print("********Average reward per thousand episodes********\n")
for r in rewards_per_thousand_episodes:
    print(count, ": ", str(sum(r/1000)))
    count += 1000
```

```
********Average reward per thousand episodes********

1000 :  0.34700000000000025
2000 :  0.6520000000000005
3000 :  0.7000000000000005
4000 :  0.6370000000000005
5000 :  0.6610000000000005
6000 :  0.6840000000000005
7000 :  0.6430000000000005
8000 :  0.6900000000000005
9000 :  0.6630000000000005
10000 :  0.6660000000000005
```

1.1.2 Interpretation

From this print, you can see that the average reward per thousand episodes did indeed progress over time. When the algorithm first started training, the first thousand episodes only averaged a reward of almost **0.18**, but by the time it got to its last thousand episodes, the reward drastically improved to almost **0.7**.

Let's take a second to understand how you can interpret these results. Your agent played **10000** episodes. At each time step within an episode, the agent received a reward of **1** if it reached the frisbee, otherwise, it received a reward of **0**. If the agent did indeed reach the frisbee, then the episode finished at that time-step.

Hence, that means for each episode, the total reward received by the agent for the entire episode is either **1** or **0**. So, for the first thousand episodes, you can interpret this score as meaning that **18%** of the time the agent received a reward of **1** and won the episode. And by the last thousand episodes from a total of **10000**, the agent was winning almost **70%** of the time.

By analyzing the grid of the game, you can see it is a lot more likely that the agent would fall in a hole or perhaps reach the max time steps than it is to reach the frisbee, so reaching the frisbee **70%** of the time is not too bad, especially since the agent had no explicit instructions to reach the frisbee. It learned that this is the correct thing to do.

- SFFF
- FHFH
- FFFH
- HFFG

At last, print out your updated Q-table to see how that has transitioned from its initial state of all zeros.

[17]:
```
# Print updated Q-table
print("\n\n********Q-table********\n")
print(q_table)
```

```
********Q-table********

[[0.53838499 0.50578313 0.52420846 0.48233792]
 [0.32047055 0.31986608 0.43377636 0.53869119]
 [0.37556151 0.40564378 0.4153974  0.48623833]
 [0.39649431 0.16688594 0.30630989 0.46953826]
 [0.56854119 0.4534341  0.2583359  0.42517826]
 [0.         0.         0.         0.        ]
 [0.30754445 0.09814265 0.12244522 0.07138102]
 [0.         0.         0.         0.        ]
 [0.25375128 0.20912951 0.36267401 0.61580683]
 [0.4056872  0.64718252 0.25597281 0.55027423]
 [0.64662542 0.49437792 0.42056282 0.2208362 ]
 [0.         0.         0.         0.        ]
 [0.         0.         0.         0.        ]
```

```
[0.50564835 0.51242791 0.73628606 0.40880967]
[0.71326707 0.87848523 0.71581137 0.74432412]
[0.         0.         0.         0.        ]]
```

[ ]:

[ ]: