```
!pip install cmake 'gym[atari]' scipy
!pip install gym[toy_text]
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: cmake in /usr/local/lib/python3.7/dist-packages (3.22.6)
Requirement already satisfied: gym[atari] in /usr/local/lib/python3.7/dist-packages (0.25.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (1.7.3)
Requirement already satisfied: numpy<1.23.0,>=1.16.5 in /usr/local/lib/python3.7/dist-packages (from scipy) (1.21.6)
Requirement already satisfied: cloudpickle>=1.2.0 in /usr/local/lib/python3.7/dist-packages (from gym[atari]) (1.5.0)
Requirement already satisfied: gym-notices>=0.0.4 in /usr/local/lib/python3.7/dist-packages (from gym[atari]) (0.0.8)
Requirement already satisfied: importlib-metadata>=4.8.0 in /usr/local/lib/python3.7/dist-packages (from gym[atari]) (4.13.0)
Requirement already satisfied: ale-py~=0.7.5 in /usr/local/lib/python3.7/dist-packages (from gym[atari]) (0.7.5)
Requirement already satisfied: importlib-resources in /usr/local/lib/python3.7/dist-packages (from ale-py~=0.7.5->gym[atari]) (5.10.0)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (from importlib-metadata>=4.8.0->gym[atari]) (3.10.0)
Requirement already satisfied: typing-extensions>=3.6.4 in /usr/local/lib/python3.7/dist-packages (from importlib-metadata>=4.8.0->gym[atari]) (4.1.1)
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: gym[toy_text] in /usr/local/lib/python3.7/dist-packages (0.25.2)
Requirement already satisfied: importlib-metadata>=4.8.0 in /usr/local/lib/python3.7/dist-packages (from gym[toy_text]) (4.13.0)
Requirement already satisfied: cloudpickle>=1.2.0 in /usr/local/lib/python3.7/dist-packages (from gym[toy_text]) (1.5.0)
Requirement already satisfied: numpy>=1.18.0 in /usr/local/lib/python3.7/dist-packages (from gym[toy_text]) (1.21.6)
Requirement already satisfied: gym-notices>=0.0.4 in /usr/local/lib/python3.7/dist-packages (from gym[toy_text]) (0.0.8)
Requirement already satisfied: pygame==2.1.0 in /usr/local/lib/python3.7/dist-packages (from gym[toy_text]) (2.1.0)
Requirement already satisfied: typing-extensions>=3.6.4 in /usr/local/lib/python3.7/dist-packages (from importlib-metadata>=4.8.0->gym[toy_text]) (4.1.1)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (from importlib-metadata>=4.8.0->gym[toy_text]) (3.10.0)
```

```
import gym
import os
os.environ["SDL_VIDEODRIVER"] = "dummy"

env = gym.make("Taxi-v3")

env.reset()
env.render()

print("Action Space {}".format(env.action_space))
print("State Space {}".format(env.observation_space))
```

```
/usr/local/lib/python3.7/dist-packages/gym/core.py:50: DeprecationWarning: WARN: You are calling render method, but you didn't specified the argument render_mode at envi
If you want to render in human mode, initialize the environment in this way: gym.make('EnvName', render_mode='human') and don't call the render method.
See here for more information: https://www.gymlibrary.ml/content/api/
  "You are calling render method, "
Action Space Discrete(6)
State Space Discrete(500)
```

```
state = env.encode(3, 1, 2, 0) # (taxi row, taxi column, passenger index, destination index)
print("State:", state)

env.s = state
env.render()
```

```
State: 328
```

```
env.P[328]
```

```
{0: [(1.0, 428, -1, False)],
 1: [(1.0, 228, -1, False)],
 2: [(1.0, 348, -1, False)],
 3: [(1.0, 328, -1, False)],
 4: [(1.0, 328, -10, False)],
 5: [(1.0, 328, -10, False)]}
```

```
env.s = 328  # set environment to illustration's state

epochs = 0
penalties, reward = 0, 0

frames = [] # for animation

done = False

while not done:
    action = env.action_space.sample()
    state, reward, done, info = env.step(action)

    if reward == -10:
        penalties += 1

    # Put each rendered frame into dict for animation
    frames.append({
        'frame': env.render(mode='ansi'),
        'state': state,
        'action': action,
        'reward': reward
        }
    )

    epochs += 1


print("Timesteps taken: {}".format(epochs))
print("Penalties incurred: {}".format(penalties))
```

```
Timesteps taken: 200
```

```
     Penalties incurred: 66

from IPython.display import clear_output
from time import sleep

def print_frames(frames):
    for i, frame in enumerate(frames):
        clear_output(wait=True)
        print(frame['frame'])
        print(f"Timestep: {i + 1}")
        print(f"State: {frame['state']}")
        print(f"Action: {frame['action']}")
        print(f"Reward: {frame['reward']}")
        sleep(.1)

print_frames(frames)

     +---------+
     |R: | : :G|
     | : | : : |
     | : : : : |
     | | : | : |
     |Y| : |B: |
     +---------+
       (North)

     Timestep: 200
     State: 31
     Action: 1
     Reward: -1
```

```
%%time
"""Training the agent"""

import random
from IPython.display import clear_output

# Hyperparameters
alpha = 0.1
gamma = 0.6
epsilon = 0.1
# alpha = 0.7 #learning rate
# discount_factor = 0.618
# epsilon = 1
# max_epsilon = 1
# min_epsilon = 0.01
# decay = 0.01

# train_episodes = 2000
# test_episodes = 100
# max_steps = 100

# For plotting metrics
all_epochs = []
all_penalties = []

for i in range(1, 1001):
    state = env.reset()

    epochs, penalties, reward, = 0, 0, 0
    done = False

    while not done:
        if random.uniform(0, 1) < epsilon:
            action = env.action_space.sample() # Explore action space
        else:
            action = np.argmax(q_table[state]) # Exploit learned values

        next_state, reward, done, info = env.step(action)

        old_value = q_table[state, action]
        next_max = np.max(q_table[next_state])

        new_value = (1 - alpha) * old_value + alpha * (reward + gamma * next_max)
        q_table[state, action] = new_value

        if reward == -10:
            penalties += 1

        state = next_state
        epochs += 1

    if i % 100 == 0:
        clear_output(wait=True)
        print(f"Episode: {i}")

print("Training finished.\n")
```

```
     Episode: 1000
     Training finished.
```

```
CPU times: user 12.2 s, sys: 1.04 s, total: 13.2 s
Wall time: 12 s
```

```
q_table[328]
```

```
array([-2.32097937, -2.32341146, -2.32634681, -2.32173302, -5.73036296,
       -7.96096875])
```

```
"""Evaluate agent's performance after Q-learning"""

total_epochs, total_penalties = 0, 0
episodes = 100

for _ in range(episodes):
    state = env.reset()
    epochs, penalties, reward = 0, 0, 0

    done = False

    while not done:
        action = np.argmax(q_table[state])
        state, reward, done, info = env.step(action)

        if reward == -10:
            penalties += 1

        epochs += 1

    total_penalties += penalties
    total_epochs += epochs

print(f"Results after {episodes} episodes:")
print(f"Average timesteps per episode: {total_epochs / episodes}")
print(f"Average penalties per episode: {total_penalties / episodes}")
```

```
Results after 100 episodes:
Average timesteps per episode: 165.7
Average penalties per episode: 0.0
```