```
!pip install gym[atari,toy_text,accept_rom_license]
```

```
Requirement already satisfied: gym[accept_rom_license,atari,toy_text] in /usr/local/lib/python3.10/dist-packages (0.25.2)
WARNING: gym 0.25.2 does not provide the extra 'accept_rom_license'
Requirement already satisfied: numpy>=1.18.0 in /usr/local/lib/python3.10/dist-packages (from gym[accept_rom_license,atari,toy_te
Requirement already satisfied: cloudpickle>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from gym[accept_rom_license,atari,t
Requirement already satisfied: gym-notices>=0.0.4 in /usr/local/lib/python3.10/dist-packages (from gym[accept_rom_license,atari,t
Collecting ale-py~=0.7.5 (from gym[accept_rom_license,atari,toy_text])
  Downloading ale_py-0.7.5-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.6 MB)
2K     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.6/1.6 MB 14.3 MB/s eta 0:00:00
e==2.1.0 (from gym[accept_rom_license,atari,toy_text])
  Downloading pygame-2.1.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (18.3 MB)
2K     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 18.3/18.3 MB 30.6 MB/s eta 0:00:00
ent already satisfied: importlib-resources in /usr/local/lib/python3.10/dist-packages (from ale-py~=0.7.5->gym[accept_rom_license
Installing collected packages: pygame, ale-py
  Attempting uninstall: pygame
    Found existing installation: pygame 2.5.0
    Uninstalling pygame-2.5.0:
      Successfully uninstalled pygame-2.5.0
Successfully installed ale-py-0.7.5 pygame-2.1.0
```

```
import numpy as np
import random
import gym
env = gym.make('Taxi-v3', new_step_api=True)

# step types
STEPTYPE_FIRST = 0
STEPTYPE_MID = 1
STEPTYPE_LAST = 2

Q = np.random.uniform(size=(500, 6))
```

```
env.reset()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transforr
  and should_run_async(code)
```

```
348
```

```
env.step(0)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transforr
  and should_run_async(code)
```

```
(448,
 -1,
 False,
 False,
 {'prob': 1.0, 'action_mask': array([0, 1, 0, 1, 0, 0], dtype=int8)})
```

```
# wrapper for gym's blackjack environment
def generate_start_step():
    return { 'observation': env.reset(), 'reward': 0., 'step_type': STEPTYPE_FIRST }

def generate_next_step(step, action):
```

```
    obs, reward, done, _, info = env.step(action)
    step_type = STEPTYPE_LAST if done else STEPTYPE_MID
    return { 'observation': obs, 'reward': reward, 'step_type': step_type }
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transfor
  and should_run_async(code)
```

```
epsilon = 0.1

def get_eps_soft_action(step):
    # epsilon-soft greedy policy
    if random.random() < epsilon:
        return np.random.choice(6, 1)[0]
    else:
        observ = step['observation']
        return np.argmax(Q[observ])
```

```
def get_greedy_action(step):
    observ = step['observation']
    return np.argmax(Q[observ])
```

```
def get_random_action(step):
    return random.randint(0, env.action_space.n-1)

behavior_prob_hit = 1. / float(env.action_space.n)
```

```
# return true if (observ, action) exists in epi
def in_episode(epi, observ, action):
    for s, a in zip(*epi):
        if s['observation'] == observ and a == action:
            return True
    return False
```

```
def generate_episode(policy_func=get_random_action):
    episode = list()
    actions = list()
    frames = list()
    step = generate_start_step()
    frames.append(env.render(mode='ansi'))
    episode.append(step)
    while step['step_type'] != STEPTYPE_LAST:
        action = policy_func(step)
        step = generate_next_step(step, action)
        frames.append(env.render(mode='ansi'))
        episode.append(step)
        actions.append(action)
    return episode, actions, frames
```

```
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)
        sleep(.2)
```

```
maxiter = 100000
gamma = 1
epsilon = 0.3
lr_rate = 0.8

Q = np.random.uniform(size=(env.observation_space.n, env.action_space.n))
```

```
for _ in range(maxiter):
    # starting step
    step = generate_start_step()
    action = get_random_action(step)
    done = False
    while not done:
        next_step = generate_next_step(step, action)

        if next_step['step_type'] == STEPTYPE_LAST:
            state = step['observation']
            idx1 = (state, action)
            Q[idx1] = Q[idx1] + lr_rate * (next_step['reward'] - Q[idx1])
            done = True
        else:
            best_action = get_greedy_action(next_step)

            state = step['observation']
            next_state = next_step['observation']
            idx1 = (state, action)
            idx2 = (next_state, best_action)
            Q[idx1] = Q[idx1] + lr_rate * ((next_step['reward'] + gamma * Q[idx2]) - Q[idx1])

            next_action = get_eps_soft_action(step)

            step = next_step
            action = next_action
```

```
epi, actions, frames = generate_episode(policy_func=get_greedy_action)
print_frames(frames)
```

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Dropoff)
```