```
pip install git+https://github.com/mimoralea/gym-walk#egg=gym-walk
```

```
Collecting gym-walk
  Cloning https://github.com/mimoralea/gym-walk to /tmp/pip-install-tt6x0ywp/gym-walk_a1fff9e3c70b43fbbe5f4fd10b1c48ea
  Running command git clone --filter=blob:none --quiet https://github.com/mimoralea/gym-walk /tmp/pip-install-tt6x0ywp/gym-walk_a1f
  Resolved https://github.com/mimoralea/gym-walk to commit b915b94cf2ad16f8833a1ad92ea94e88159279f5
  Preparing metadata (setup.py) ... done
Requirement already satisfied: gym in /usr/local/lib/python3.11/dist-packages (from gym-walk) (0.25.2)
Requirement already satisfied: numpy>=1.18.0 in /usr/local/lib/python3.11/dist-packages (from gym->gym-walk) (2.0.2)
Requirement already satisfied: cloudpickle>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from gym->gym-walk) (3.1.1)
Requirement already satisfied: gym-notices>=0.0.4 in /usr/local/lib/python3.11/dist-packages (from gym->gym-walk) (0.0.8)
Building wheels for collected packages: gym-walk
  Building wheel for gym-walk (setup.py) ... done
  Created wheel for gym-walk: filename=gym_walk-0.0.2-py3-none-any.whl size=5377 sha256=dcbf33888a5c3ff5ff0095be9b9da337d30dee062fba
  Stored in directory: /tmp/pip-ephem-wheel-cache-9t4npgfu/wheels/60/02/77/2dd9f31df8d13bc7c014725f4002e29d0fc3ced5e8ac08e1cf
Successfully built gym-walk
Installing collected packages: gym-walk
Successfully installed gym-walk-0.0.2
```

```
import warnings ; warnings.filterwarnings('ignore')

import gym, gym_walk
import numpy as np

import random
import warnings

warnings.filterwarnings('ignore', category=DeprecationWarning)
np.set_printoptions(suppress=True)
random.seed(123); np.random.seed(123)
```

```
def print_policy(pi, P, action_symbols=('<', 'v', '>', '^'), n_cols=4, title='Policy:'):
    print(title)
    arrs = {k:v for k,v in enumerate(action_symbols)}
    for s in range(len(P)):
        a = pi(s)
        print("| ", end="")
        if np.all([done for action in P[s].values() for _, _, _, done in action]):
            print("".rjust(9), end=" ")
        else:
            print(str(s).zfill(2), arrs[a].rjust(6), end=" ")
        if (s + 1) % n_cols == 0: print("|")
```

```
def print_state_value_function(V, P, n_cols=4, prec=3, title='State-value function:'):
    print(title)
    for s in range(len(P)):
        v = V[s]
        print("| ", end="")
        if np.all([done for action in P[s].values() for _, _, _, done in action]):
            print("".rjust(9), end=" ")
        else:
            print(str(s).zfill(2), '{}'.format(np.round(v, prec)).rjust(6), end=" ")
        if (s + 1) % n_cols == 0: print("|")
```

```
def probability_success(env, pi, goal_state, n_episodes=100, max_steps=200):
    random.seed(123); np.random.seed(123) ; env.seed(123)
    results = []
    for _ in range(n_episodes):
        state, done, steps = env.reset(), False, 0
        while not done and steps < max_steps:
            state, _, done, h = env.step(pi(state))
            steps += 1
        results.append(state == goal_state)
    return np.sum(results)/len(results)
```

```
def mean_return(env, pi, n_episodes=100, max_steps=200):
    random.seed(123); np.random.seed(123) ; env.seed(123)
    results = []
    for _ in range(n_episodes):
        state, done, steps = env.reset(), False, 0
        results.append(0.0)
        while not done and steps < max_steps:
            state, reward, done, _ = env.step(pi(state))
            results[-1] += reward
            steps += 1
    return np.mean(results)
```

```
env = gym.make('FrozenLake-v1')
P = env.env.P
init_state = env.reset()
goal_state = 15
#LEFT, RIGHT = range(2)
```

```
P
```

```
{0: {0: [(0.3333333333333333, 0, 0.0, False),
   (0.3333333333333333, 0, 0.0, False),
   (0.3333333333333333, 4, 0.0, False)],
  1: [(0.3333333333333333, 0, 0.0, False),
   (0.3333333333333333, 4, 0.0, False),
   (0.3333333333333333, 1, 0.0, False)],
  2: [(0.3333333333333333, 4, 0.0, False),
   (0.3333333333333333, 1, 0.0, False),
   (0.3333333333333333, 0, 0.0, False)],
  3: [(0.3333333333333333, 1, 0.0, False),
   (0.3333333333333333, 0, 0.0, False),
   (0.3333333333333333, 0, 0.0, False)]},
 1: {0: [(0.3333333333333333, 1, 0.0, False),
   (0.3333333333333333, 0, 0.0, False),
   (0.3333333333333333, 5, 0.0, True)],
  1: [(0.3333333333333333, 0, 0.0, False),
   (0.3333333333333333, 5, 0.0, True),
   (0.3333333333333333, 2, 0.0, False)],
  2: [(0.3333333333333333, 5, 0.0, True),
   (0.3333333333333333, 2, 0.0, False),
   (0.3333333333333333, 1, 0.0, False)],
  3: [(0.3333333333333333, 2, 0.0, False),
   (0.3333333333333333, 1, 0.0, False),
   (0.3333333333333333, 0, 0.0, False)]},
 2: {0: [(0.3333333333333333, 2, 0.0, False),
   (0.3333333333333333, 1, 0.0, False),
   (0.3333333333333333, 6, 0.0, False)],
  1: [(0.3333333333333333, 1, 0.0, False),
   (0.3333333333333333, 6, 0.0, False),
   (0.3333333333333333, 3, 0.0, False)],
  2: [(0.3333333333333333, 6, 0.0, False),
   (0.3333333333333333, 3, 0.0, False),
   (0.3333333333333333, 2, 0.0, False)],
  3: [(0.3333333333333333, 3, 0.0, False),
   (0.3333333333333333, 2, 0.0, False),
   (0.3333333333333333, 1, 0.0, False)]},
 3: {0: [(0.3333333333333333, 3, 0.0, False),
   (0.3333333333333333, 2, 0.0, False),
   (0.3333333333333333, 7, 0.0, True)],
  1: [(0.3333333333333333, 2, 0.0, False),
   (0.3333333333333333, 7, 0.0, True),
   (0.3333333333333333, 3, 0.0, False)],
  2: [(0.3333333333333333, 7, 0.0, True),
   (0.3333333333333333, 3, 0.0, False),
   (0.3333333333333333, 3, 0.0, False)],
  3: [(0.3333333333333333, 3, 0.0, False),
   (0.3333333333333333, 3, 0.0, False),
   (0.3333333333333333, 2, 0.0, False)]},
 4: {0: [(0.3333333333333333, 0, 0.0, False),
   (0.3333333333333333, 4, 0.0, False),
   (0.3333333333333333, 8, 0.0, False)],
  1: [(0.3333333333333333, 4, 0.0, False),
   (0.3333333333333333, 8, 0.0, False),
   (0.3333333333333333, 5, 0.0, True)],
  2: [(0.3333333333333333, 8, 0.0, False),
   (0.3333333333333333, 5, 0.0, True),
   (0.3333333333333333, 0, 0.0, False)],
  3: [(0.3333333333333333, 5, 0.0, True),
def decay_schedule(
    init_value, min_value, decay_ratio,
    max_steps, log_start = -2, log_base=10):


   decay_steps = int(max_steps* decay_ratio)
   rem_steps=max_steps-decay_steps
   values=np.logspace(log_start,0,decay_steps,base=log_base,endpoint=True)[::-1]
   values=(values-values.min())
   values=(init_value-min_value)*values+min_value
   values=np.pad(values,(0,rem_steps),'edge')

   return values
```

```
from itertools import count
def generate_trajectory(
    select_action, Q, epsilon,
    env, max_steps=200):
  done, trajectory = False, []

  done,trajectory=False,[]
  while not done:
    state=env.reset()
    for t in count():
      action=select_action(state,Q,epsilon)
      next_state,reward,done,_=env.step(action)
      experience=(state,action,reward,next_state,done)
      trajectory.append(experience)
      if done:
        break
      if t>=max_steps-1:
        trajectory=[]
        break
      state=next_state
    return np.array(trajectory,object)
```

```
!pip install --upgrade numpy
```

```
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (2.0.2)
Collecting numpy
  Downloading numpy-2.2.5-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (62 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.0/62.0 kB 1.6 MB/s eta 0:00:00
Downloading numpy-2.2.5-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (16.4 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 16.4/16.4 MB 90.3 MB/s eta 0:00:00
Installing collected packages: numpy
  Attempting uninstall: numpy
    Found existing installation: numpy 2.0.2
    Uninstalling numpy-2.0.2:
      Successfully uninstalled numpy-2.0.2
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the sou
tensorflow 2.18.0 requires numpy<2.1.0,>=1.26.0, but you have numpy 2.2.5 which is incompatible.
numba 0.60.0 requires numpy<2.1,>=1.22, but you have numpy 2.2.5 which is incompatible.
Successfully installed numpy-2.2.5
```

```
from tqdm import tqdm
```

```
!pip install --upgrade numpy
```

```
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (2.2.5)
```

```
def mc_control(env, gamma=1.0,
               init_alpha=0.5, min_alpha=0.01, alpha_decay_ratio=0.5,
               init_epsilon=1.0, min_epsilon=0.1, epsilon_decay_ratio=0.9,
               n_episodes=3000, max_steps=200, first_visit=True):

    ns, na = env.observation_space.n, env.action_space.n
    discounts = np.logspace(0, max_steps, num=max_steps, base=gamma, endpoint=False)
    alphas = decay_schedule(init_alpha, min_alpha, alpha_decay_ratio, n_episodes)
    epsilons = decay_schedule(init_epsilon, min_epsilon, epsilon_decay_ratio, n_episodes)

    Q = np.zeros((ns, na), dtype=np.float64)
    Q_track = np.zeros((n_episodes, ns, na), dtype=np.float64)
    pi_track = []

    select_action = lambda state, Q, epsilon: np.argmax(Q[state]) if np.random.random() > epsilon else np.random.randint(len(Q[state]))
    for e in tqdm(range(n_episodes), leave=False):
        trajectory = generate_trajectory(select_action, Q, epsilons[e], env, max_steps)
        visited = np.zeros((ns, na), dtype=bool)

        for t, (state, action, reward, _, _) in enumerate(trajectory):
            if visited[state][action] and first_visit:
                continue
            visited[state][action] = True
            n_steps = len(trajectory[t:])
            G = np.sum(discounts[:n_steps] * np.array([x[2] for x in trajectory[t:]]))
            Q[state][action] += alphas[e] * (G - Q[state][action])

        Q_track[e] = Q
        pi_track.append(np.argmax(Q, axis=1))

    V = np.max(Q, axis=1)
    pi = lambda s: np.argmax(Q[s])
    return Q, V, pi, Q_track, pi_track
```

```
print('Name:NARESH.R    Register Number:212223240104')
optimal_Q, optimal_V, optimal_pi,_,_= mc_control (env,n_episodes=3000)

print_state_value_function(optimal_Q, P, n_cols=4, prec=2, title='Action-value function:')
print_state_value_function(optimal_V, P, n_cols=4, prec=2, title='State-value function:')
print_policy(optimal_pi, P)
```

```
Name:NARESH.R    Register Number:212223240104
                                                 Action-value function:
| 00 [0.03 0.07 0.05 0.04] | 01 [0.05 0.03 0.05 0.07] | 02 [0.09 0.07 0.06 0.06] | 03 [0.04 0.01 0.   0.  ] |
| 04 [0.07 0.01 0.03 0.03] |           | 06 [0.11 0.04 0.06 0.02] |           |
| 08 [0.03 0.02 0.07 0.02] | 09 [0.12 0.05 0.05 0.03] | 10 [0.07 0.28 0.06 0.05] |           |
|           | 13 [0.1  0.04 0.12 0.22] | 14 [0.08 0.24 0.6  0.16] |           |
State-value function:
| 00   0.07 | 01   0.07 | 02   0.09 | 03   0.04 |
| 04   0.07 |           | 06   0.11 |           |
| 08   0.07 | 09   0.12 | 10   0.28 |           |
|           | 13   0.22 | 14    0.6 |           |
Policy:
| 00      v | 01      ^ | 02      < | 03      < |
| 04      < |           | 06      < |           |
| 08      > | 09      < | 10      v |           |
|           | 13      ^ | 14      > |           |
```

```
# Find the probability of success and the mean return of you your policy
print('Name: naresh.r                      Register Number:212223240104          ')
print('Reaches goal {:.2f}%. Obtains an average undiscounted return of {:.4f}.'.format(
    probability_success(env, optimal_pi, goal_state=goal_state)*100,
    mean_return(env, optimal_pi)))
```

```
Name: naresh.r                      Register Number:212223240104
Reaches goal 12.00%. Obtains an average undiscounted return of 0.1200.
```