```
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-mvgz3o2j/gym-walk_ca5a107c00344ccba9654284348a76c5
  Running command git clone --filter=blob:none --quiet https://github.com/mimoralea/gym-walk /tmp/pip-install-mvgz3o2j/gym-walk_ca5a
  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=3f86fbf8e54658e4cd05c7dceda3a7b9b0087666427e
  Stored in directory: /tmp/pip-ephem-wheel-cache-vbw61wx0/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
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, DOWN, RIGHT, UP = range(4)
```

```
init_state
```

```
0
```

```
state, reward, done, info = env.step(RIGHT)
print("state:{0} - reward:{1} - done:{2} - info:{3}".format(state, reward, done, info))
```

```
state:4 - reward:0.0 - done:False - info:{'prob': 0.3333333333333333}
```

```
pi_frozenlake = lambda s: {
    0: RIGHT,
    1: DOWN,
    2: RIGHT,
    3: LEFT,
    4: DOWN,
    5: LEFT,
    6: RIGHT,
    7:LEFT,
    8: UP,
    9: DOWN,
    10:LEFT,
    11:DOWN,
    12:RIGHT,
    13:RIGHT,
    14:DOWN,
    15:LEFT #Stop
}[s]
print_policy(pi_frozenlake, P, action_symbols=('<', 'v', '>', '^'), n_cols=4)
```

```
Policy:
    | 00      > | 01      v | 02      > | 03      < |
    | 04      v |           | 06      > |           |
    | 08      ^ | 09      v | 10      < |           |
    |           | 13      > | 14      v |           |
```

```
print('Reaches goal {:.2f}%. Obtains an average undiscounted return of {:.4f}.'.format(
    probability_success(env, pi_frozenlake, goal_state=goal_state) * 100,
    mean_return(env, pi_frozenlake)))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-22-5acac3070fe4> in <cell line: 0>()
      1 print('Reaches goal {:.2f}%. Obtains an average undiscounted return of {:.4f}.'.format(
      2     probability_success(env, pi_frozenlake, goal_state=goal_state) * 100,
----> 3     mean_return(env, pi_frozenlake)))

NameError: name 'mean_return' is not defined
```

```
# Create your own policy

pi_2 = lambda s: {
    0: DOWN,
    1: RIGHT,
    2: RIGHT,
```

```
    3: DOWN,
    4: LEFT,
    5: RIGHT,
    6: DOWN,
    7: LEFT,
    8: DOWN,
    9: RIGHT,
    10: LEFT,
    11: DOWN,
    12: RIGHT,
    13: DOWN,
    14: RIGHT,
    15: LEFT  # Stop at goal
}[s]

print("Name: NARESH.R")
print("Register Number: 212223240104")
print_policy(pi_2, P, action_symbols=('<', 'v', '>', '^'), n_cols=4)
```

```
Name: NARESH.R
Register Number: 212223240104
Policy:
| 00      v | 01      > | 02      > | 03      v |
| 04      < |           | 06      v |           |
| 08      v | 09      > | 10      < |           |
|           | 13      v | 14      > |           |
```

```
def policy_evaluation(pi, P, gamma=1.0, theta=1e-10):
    prev_V = np.zeros(len(P), dtype=np.float64)
    # Write your code here to evaluate the given policy
    while True:
      V=np.zeros(len(P))
      for s in range(len(P)):
        for prob,next_state,reward,done in P[s][pi(s)]:
           V[s]+=prob*(reward+gamma *prev_V[next_state]*(not done))
      if np.max(np.abs(prev_V-V))<theta:
        break
      prev_V=V.copy()
    return V
```

```
V1 = policy_evaluation(pi_frozenlake, P,gamma=0.99)
print_state_value_function(V1, P, n_cols=4, prec=5)

V2 = policy_evaluation(pi_2, P, gamma=0.99)

print("\nState-value function for Your Policy:")
print_state_value_function(V2, P, n_cols=4, prec=5)
```

```
State-value function:
| 00 0.11448 | 01 0.08191 | 02 0.13372 | 03 0.06586 |
| 04 0.15053 |            | 06 0.20562 |            |
| 08 0.30562 | 09 0.46997 | 10 0.48938 |            |
|            | 13 0.62915 | 14 0.80739 |            |

State-value function for Your Policy:
State-value function:
| 00 0.05568 | 01 0.03401 | 02 0.06904 | 03 0.03401 |
| 04 0.07905 |            | 06 0.10617 |            |
| 08 0.10481 | 09 0.21279 | 10 0.32173 |            |
|            | 13 0.32309 | 14 0.65598 |            |
```

```
if np.sum(V1 >= V2) == len(V1):
    print("\nThe first policy is the better policy.")
elif np.sum(V2 >= V1) == len(V2):
    print("\nYour policy is the better policy.")
else:
    print("\nBoth policies have their merits.")
V1>=V2

if(np.sum(V1>=V2)==11):
  print("The first policy is the better policy")
elif(np.sum(V2>=V1)==11):
  print("The second policy is the better policy")
else:
  print("Both policies have their merits.")
```

```
The first policy is the better policy.
Both policies have their merits.
```

```
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.max() - values.min())
    values = init_value + (min_value - init_value) * values
    values = np.pad(values, (0, rem_steps), 'edge')

    return values
```

```
def generate_trajectory(pi, env, max_steps=20):
    done, trajectory = False, []

    while not done:
        state = env.reset()

        for t in count():
            action = pi(state)
            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, np.object_)
```

```
import numpy as np
from itertools import count

def generate_trajectory(pi, env, max_steps=20):
    done = False
    trajectory = []

    state = env.reset()

    for t in count():
        action = pi(state)
        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, dtype=object)
```

```
from tqdm import tqdm
def mc_prediction(pi,env,gamma=1.0,init_alpha=0.5,min_alpha=0.01,alpha_decay_ratio=0.3,n_episodes=500,max_steps=100,first_visit=True):
  ns=env.observation_space.n
  discounts = lambda n: gamma ** np.arange(n)
  alpha=decay_schedule(init_alpha,min_alpha,alpha_decay_ratio,n_episodes)
  v=np.zeros(ns)
  v_track=np.zeros((n_episodes,ns))
  for e in tqdm(range(n_episodes),leave=False):
    trajectory=generate_trajectory(pi,env,max_steps)
    visited=np.zeros(ns,dtype=np.bool)
    for t,(state,_,reward,_,_) in enumerate(trajectory):
      if visited[state] and first_visit:
        continue
      visited[state]=True
      n_steps = len(trajectory[t:])
      G = np.sum(discounts(n_steps) * [step[2] for step in trajectory[t:]])

      v[state]=v[state]+alpha[e]*(G-v[state])
    v_track[e]=v
  return v.copy(),v_track
```

```
pi_frozenlake = lambda s: {
    0: RIGHT,
    1: DOWN,
    2: RIGHT,
    3: LEFT,
    4: DOWN,
    5: LEFT,
    6: RIGHT,
    7:LEFT,
    8: UP,
    9: DOWN,
    10:LEFT,
    11:DOWN,
    12:RIGHT,
    13:RIGHT,
    14:DOWN,
    15:LEFT #Stop
}[s]
print_policy(pi_frozenlake, P, action_symbols=('<', 'v', '>', '^'), n_cols=4)
```

```
Policy:
    | 00      > | 01      v | 02      > | 03      < |
    | 04      v |           | 06      > |           |
    | 08      ^ | 09      v | 10      < |           |
    |           | 13      > | 14      v |           |
```

```
import gym

env = gym.make("FrozenLake-v1")
```

```
mc_optimal_V, mc_optimal_pi = mc_prediction(pi_frozenlake,env)
```

```
print("Name: NARESH.R")
print("Register Number: 212223240104")

print_state_value_function(mc_optimal_V, P, n_cols=4, prec=5)
```

```
Name: NARESH.R
    Register Number: 212223240104
    State-value function:
    | 00 0.13678 | 01 0.01566 | 02 0.12722 | 03 0.01672 |
    | 04 0.17189 |           | 06 0.12967 |           |
    | 08 0.70435 | 09 0.70465 | 10 0.63943 |           |
    |           | 13 0.91815 | 14 0.92078 |           |
```