```
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-4861_y3t/gym
  Running command git clone --filter=blob:none --quiet https://github.com/mimora
  Resolved https://github.com/mimoralea/gym-walk to commit b915b94cf2ad16f8833a1
  Preparing metadata (setup.py) ... done
Requirement already satisfied: gym in /usr/local/lib/python3.12/dist-packages (f
Requirement already satisfied: numpy>=1.18.0 in /usr/local/lib/python3.12/dist-p
Requirement already satisfied: cloudpickle>=1.2.0 in /usr/local/lib/python3.12/d
Requirement already satisfied: gym-notices>=0.0.4 in /usr/local/lib/python3.12/d
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
  Stored in directory: /tmp/pip-ephem-wheel-cache-le564hc4/wheels/bf/23/e5/a94be
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, titl
    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 ac
            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 f
    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 ac
            print("".rjust(9), end=" ")
```

```
        else:
            print(str(s).zfill(2), '{}'.format(np.round(v, prec)).rjust(6),
        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)
```

```
envdesc  = ['FSFF','FFFF','FFHF', 'GFFH']
env = gym.make('FrozenLake-v1',desc=envdesc)
init_state = env.reset()
goal_state = 12 #(Goal State)
P = env.env.P
```

```
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, False)],
  1: [(0.3333333333333333, 0, 0.0, False),
   (0.3333333333333333, 5, 0.0, False),
   (0.3333333333333333, 2, 0.0, False)],
```

```
 2: [(0.3333333333333333, 5, 0.0, False),
  (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, False)],
 1: [(0.3333333333333333, 2, 0.0, False),
  (0.3333333333333333, 7, 0.0, False),
  (0.3333333333333333, 3, 0.0, False)],
 2: [(0.3333333333333333, 7, 0.0, False),
  (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, False)],
 2: [(0.3333333333333333, 8, 0.0, False),
  (0.3333333333333333, 5, 0.0, False),
  (0.3333333333333333, 0, 0.0, False)],
 3: [(0.3333333333333333, 5, 0.0, False),
```

```python
def value_iteration(P, gamma=1.0, theta=1e-10):
  V = np.zeros(len(P), dtype=np.float64)
  while True:
    Q = np.zeros((len(P), len(P[0])), dtype=np.float64)
    for s in range(len(P)):
      for a in range(len(P[s])):
        for prob, next_state, reward, done in P[s][a]:
          Q[s][a] += prob * (reward + gamma * V[next_state] * (not done))
    if np.max(np.abs(V - np.max(Q, axis=1))) < theta:
      break

    V = np.max(Q, axis=1)
    pi = lambda s: {s:a for s, a in enumerate(np.argmax(Q, axis=1))}[s]

    return V, pi
```

```
V_best_v, pi_best_v = value_iteration(P, gamma=0.99)
```

```
print("Name: JERUSHLIN JOSE JB        Register Number: 212222240039    ")
print('Optimal policy and state-value function (VI):')
print_policy(pi_best_v, P)
```

```
Name: JERUSHLIN JOSE JB        Register Number: 212222240039
Optimal policy and state-value function (VI):
Policy:
| 00      < | 01      < | 02      < | 03      < |
| 04      < | 05      < | 06      < | 07      < |
| 08      < | 09      < |             | 11      < |
|             | 13      < | 14      < |             |
```

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

```
Reaches goal 100.00%. Obtains an average undiscounted return of 1.0000.
```

```
print_state_value_function(V_best_v, P, prec=4)
```

```
State-value function:
| 00    0.0 | 01    0.0 | 02    0.0 | 03    0.0 |
| 04    0.0 | 05    0.0 | 06    0.0 | 07    0.0 |
| 08 0.3333 | 09    0.0 |             | 11    0.0 |
|             | 13 0.3333 | 14    0.0 |             |
```

```
Start coding or generate with AI.
```