VALUE ITERATION ALGORITHM

AIM

The aim of the program is to find the best way for an agent to navigate a grid environment by using the Value Iteration algorithm.

PROBLEM STATEMENT

The goal is to determine the optimal policy for an agent navigating through a grid environment, where each state represents a grid cell, and actions move the agent in one of four directions (up, down, left, or right). The task is to maximize the expected reward, leading the agent to the goal state while avoiding obstacles.

POLICY ITERATION ALGORITHM

Step 1:

Set the value of each state to 0 (initial guess).

Step 2:

Look at all the actions you can take from that state (like moving up, down, left, or right).

Step 3:

Calculate the expected value of each action (i.e., how good that action is based on its possible results).

Step 4:

Pick the action that gives the highest value and update the value of the state with that number.

Step 5:

Keep updating the values for all states until the difference between the old and new values is very small

Step 6:

Once the values have stabilized, go through each state again and pick the action that leads to the highest value. This gives you the optimal action (policy) for each state.

VALUE ITERATION FUNCTION

Name: Meetha Prabhu

Register Number: 212222240065

gym walk installation

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

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)
```

Printing Functions

```
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)
```

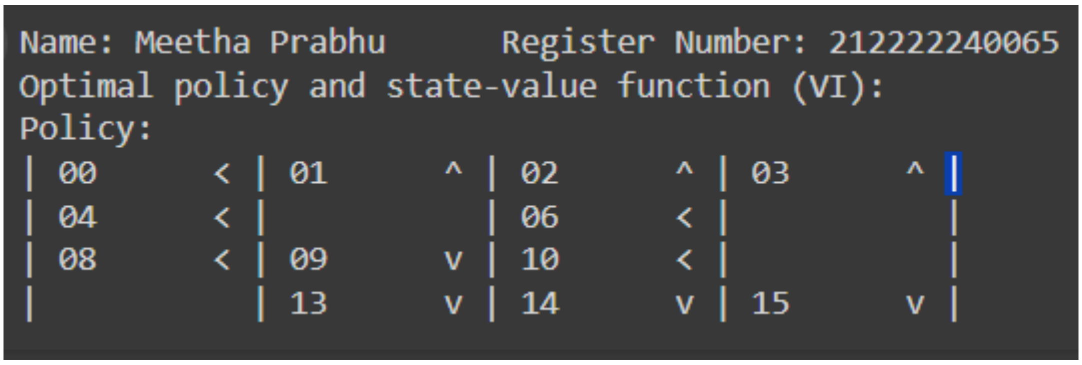
Creating Frozen lake environnent

```
envdesc  = ['SFFF','FHFH','FFFH', 'GFFF']
env = gym.make('FrozenLake-v1',desc=envdesc)
init_state = env.reset()
goal_state = 12
P = env.env.P

P
```

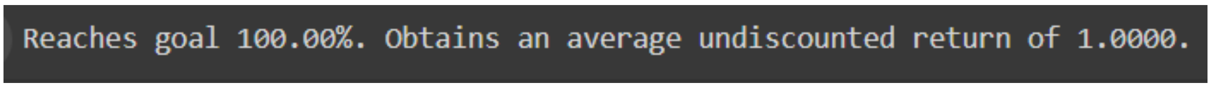
Value Iteration

```
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

# Finding the optimal policy
V_best_v, pi_best_v = value_iteration(P, gamma=0.99)

# Printing the policy
print("Name: Meetha Prabhu      Register Number: 212222240065")
print('Optimal policy and state-value function (VI):')
print_policy(pi_best_v, P)

# printing the success rate and the mean return
print('Reaches goal {:.2f}%. Obtains an average undiscounted return of
{:.4f}.'.format(
    probability_success(env, pi_best_v, goal_state=goal_state)*100,
    mean_return(env, pi_best_v)))


# printing the state value function
print_state_value_function(V_best_v, P, prec=4)
```

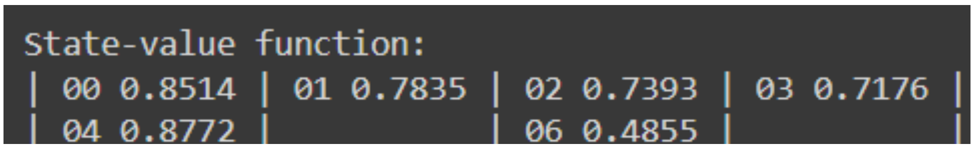
OUTPUT:



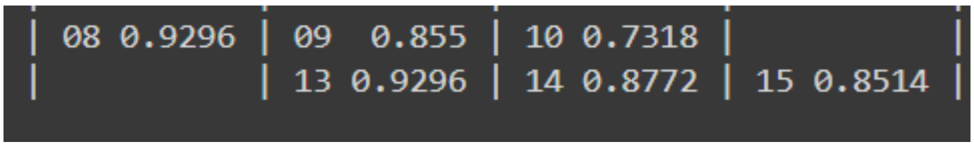
```
Name: Meetha Prabhu      Register Number: 212222240065
Optimal policy and state-value function (VI):
Policy:
| 00      < | 01      ^ | 02      ^ | 03      ^ |
| 04      < |           | 06      < |           |
| 08      < | 09      v | 10      < |           |
|           | 13      v | 14      v | 15      v |
```



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



```
State-value function:
| 00 0.8514 | 01 0.7835 | 02 0.7393 | 03 0.7176 |
| 04 0.8772 |           | 06 0.4855 |           |
```



```
| 08 0.9296 | 09  0.855 | 10 0.7318 |           |
|           | 13 0.9296 | 14 0.8772 | 15 0.8514 |
```

RESULT:

Thus the program to find th optimal policy using value iteration is implemented successfully