POLICY ITERATION ALGORITHM

AIM

It aims to implement and evaluate a policy iteration algorithm in a custom environment (gym-walk) to determine the optimal policy that maximizes the agent's likelihood and reward in reaching the goal state.

PROBLEM STATEMENT

The task is to develop and use a policy iteration algorithm to solve a grid-based environment (gym-walk), where the agent learns the optimal sequence of actions to maximize its probability of reaching the goal state and achieving the highest cumulative reward.

POLICY ITERATION ALGORITHM

Step 1: Start with a random policy and an arbitrary value function.

Step 2: Compute the value function for the current policy.

Step 3: Update the policy to be greedy with respect to the current value function.

Step 4: Repeat evaluation and improvement until the policy stabilizes.

Step 5: The final policy is optimal and provides the best actions for each state.

POLICY IMPROVEMENT FUNCTION

Name: Meetha Prabhu

Register Number: 212222240065

Policy Improvement Function

```
def policy_improvement(V,P,gamma=1.0):
  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))
  new_pi=lambda s: {s:a for s, a in enumerate(np.argmax(Q,axis=1))}[s]
  return new_pi

# Finding the improved policy
pi_2 = policy_improvement(V1, P)
print('Name: Meetha Prabhu          Register Number: 212222240065')
print_policy(pi_2, P, action_symbols=('<', '>'), n_cols=7)


print('Reaches goal {:.2f}%. Obtains an average undiscounted return of {:.4f}.'.format(
    probability_success(env, pi_2, goal_state=goal_state)*100,
    mean_return(env, pi_2)))


# Finding the value function for the improved policy
V2 = policy_evaluation(pi_2, P)
print('Name: Meetha Prabhu           Register Number: 212222240065     ')
print_state_value_function(V2, P, n_cols=7, prec=5)

# comparing the initial and the improved policy
if(np.sum(V1>=V2)==7):
  print("The first policy is the better policy")
elif(np.sum(V2>=V1)==7):
  print("The second policy is the better policy")
else:
  print("Both policies have their merits.")
```

POLICY ITERATION FUNCTION

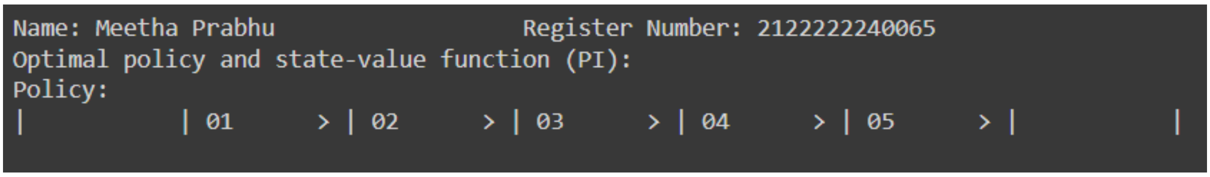
Name: Meetha Prabhu

Register Number: 212222240065

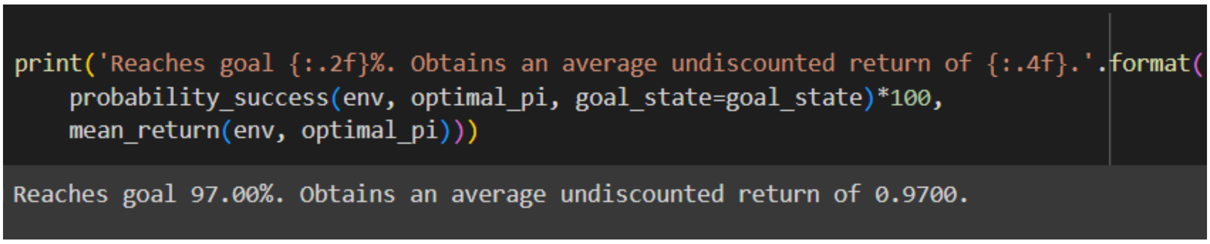
Policy Iteration

```
def policy_iteration(P, gamma=1.0, theta=1e-10):
  random_actions = np.random.choice(tuple(P[0].keys()), len(P))
  ramdon_actions=np.random.choice(tuple (P[0].keys()), len(P))
  pi=lambda s: {s: a for s, a in enumerate(random_actions)} [s]
  while True:
    old_pi={s:pi(s) for s in range (len(P))}
    V=policy_evaluation(pi,P,gamma,theta)
    pi=policy_improvement(V,P,gamma)
    if old_pi=={s:pi(s) for s in range(len(P))}:
      break
    return V,pi

optimal_V, optimal_pi = policy_iteration(P)

print('Name: Meetha Prabhu                  Register Number: 2122222240065
')
print('Optimal policy and state-value function (PI):')
print_policy(optimal_pi, P, action_symbols=('<', '>'), n_cols=7)


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)))

print_state_value_function(optimal_V, P, n_cols=7, prec=5)
```

OUTPUT:

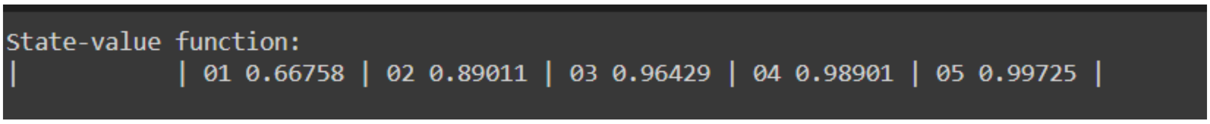


```
Name: Meetha Prabhu                  Register Number: 2122222240065
Optimal policy and state-value function (PI):
Policy:
|           | 01      > | 02      > | 03      > | 04      > | 05      > |           |
```



```
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)))
```

```
Reaches goal 97.00%. Obtains an average undiscounted return of 0.9700.
```



```
State-value function:
|           | 01 0.66758 | 02 0.89011 | 03 0.96429 | 04 0.98901 | 05 0.99725 |
```

RESULT:

Thus, the program to iterate Policy improvement and evaluation is implemented successfully