EX.NO:4 **VALUE ITERATION ALGORITHM**
DATE:28/01/2025

AIM:
To implement the Value Iteration algorithm to solve the Frozen Lake problem in OpenAI Gym, calculate the optimal policy, and determine the optimal state-value function. The goal is to navigate the environment effectively by maximizing cumulative rewards while avoiding hazards.

PROCEDURE:

1. **Environment Initialization**:

   Import the necessary libraries (numpy and gym).

   Create the Frozen Lake environment using gym.make(). Ensure is_slippery=True for a stochastic environment.

2. **Define Value Iteration**:

   Initialize variables:

   a. n_states: Total number of states in the environment.
   b. n_actions: Total number of possible actions.
   c. V: State-value function initialized to zeros.
   d. directions: Symbols ('<', 'v', '>', '^') to represent actions visually.

   Set up parameters:

   e. gamma: Discount factor for rewards.
   f. theta: Threshold for convergence.

3. **Perform Iterative Updates**:

   Use a while loop to repeatedly update the state-value function V(s)V(s)V(s):

   a. For each state sss:
      i. Calculate the action-value function Q(s,a)Q(s, a)Q(s,a) for all possible actions aaa using the Bellman equation: Q(s,a)=∑s′P(s′∣s,a)⋅[R(s,a,s′)+γ⋅V(s′)]Q(s, a) = \sum_{s'} P(s' | s, a) \cdot [R(s, a, s') + \gamma \cdot V(s')]Q(s,a)=s′∑P(s′∣s,a)⋅[R(s,a,s′)+γ⋅V(s′)]
      ii. Update V(s)V(s)V(s) with the maximum Q(s,a)Q(s, a)Q(s,a).
      iii. Record the action with the highest Q(s,a)Q(s, a)Q(s,a) for visualizing the policy directions.
   b. Track the maximum change in V(s)V(s)V(s) to check for convergence.

4. **Extract Optimal Policy**:

   After convergence, extract the optimal policy:

   a. For each state, find the action aaa that maximizes Q(s,a)Q(s, a)Q(s,a).
   b. Store the action in the policy array.
   c. Update a policy matrix with symbols to visually represent the optimal actions.

CODE:

```
import numpy as np

import gym

def value_iteration(env, gamma=0.99, theta=1e-8):

    n_states = env.observation_space.n
```

```
    n_actions = env.action_space.n
    V = np.zeros(n_states)  # Initialize state-value function
    policy = np.zeros(n_states, dtype=int)  # Initialize policy
    while True:
        delta = 0
        for s in range(n_states):
            q_values = np.zeros(n_actions)

            for a in range(n_actions):
                for prob, next_state, reward, done in env.P[s][a]:
                    q_values[a] += prob * (reward + gamma * V[next_state])

            max_value = np.max(q_values)
            delta = max(delta, abs(max_value - V[s]))
            V[s] = max_value

        if delta < theta:
            break
    for s in range(n_states):
        q_values = np.zeros(n_actions)
        for a in range(n_actions):
            for prob, next_state, reward, done in env.P[s][a]:
                q_values[a] += prob * (reward + gamma * V[next_state])
        policy[s] = np.argmax(q_values)

    return policy, V
# Run Value Iteration on Frozen Lake
env = gym.make("FrozenLake-v1", is_slippery=True)
optimal_policy, optimal_values = value_iteration(env)
```

```
print("Optimal Policy:")

print(optimal_policy)

print("Optimal State-Value Function:")

print(optimal_values)
```

OUTPUT:

```
Final Optimal Policy Directions:
[['<' '^' '^' '^']
 ['<' '<' '<' '<']
 ['^' 'v' '<' '<']
 ['<' '>' 'v' '<']]

Optimal Policy:
[0 3 3 3 0 0 0 0 3 1 0 0 0 2 1 0]

Optimal State-Value Function:
[0.54202581 0.49880303 0.47069551 0.4568515  0.55845085 0.
 0.35834799 0.         0.59179866 0.64307976 0.6152075  0.
 0.         0.7417204  0.86283741 0.        ]
```