10.

Aim:

The aim of this program is to implement the policy iteration algorithm to solve a simple grid-world problem.

Algorithm:

Initialize the policy randomly.

Evaluate the policy by iteratively applying the Bellman equation until the value function converges to a stable solution.

Improve the policy by selecting the action that maximizes the value function for each state.

Repeat steps 2 and 3 until the policy converges to a stable solution.

program

```
actions = (0, 1)

states = (0, 1, 2, 3, 4)

rewards = [-1, -1, 10, -1, -1]

gamma = 0.9

delta = 10


probs = [

    [[0.9, 0.1], [0.1, 0.9], [0, 0], [0, 0], [0, 0]],

    [[0.9, 0.1], [0, 0], [0.1, 0.9], [0, 0], [0, 0]],

    [[0, 0], [0, 0], [0, 0], [0, 0], [0, 0]],

    [[0, 0], [0, 0], [0.9, 0.1], [0, 0], [0.1, 0.9]],

    [[0, 0], [0, 0], [0, 0], [0.9, 0.1], [0.1, 0.9]],

]


max_policy_iter = 10000

max_value_iter = 10000

pi = [0 for s in states]

V = [0 for s in states]
```

```
for i in range(max_policy_iter):
    optimal_policy_found = True
    for j in range(max_value_iter):
        max_diff = 0
        V_new = [0, 0, 0, 0, 0]
        for s in states:
            val = rewards[s]
            for s_next in states:
                val += probs[s][s_next][pi[s]] * (gamma * V[s_next])
            max_diff = max(max_diff, abs(val - V[s]))
            V[s] = val
            if max_diff < delta:
                break
        for s in states:
            val_max = V[s]
            for a in actions:
                val = rewards[s]
                for s_next in states:
                    val += probs[s][s_next][a] * (gamma * V[s_next])
                # check if new policy is different from current policy
                if val > val_max and pi[s] != a:
                    pi[s] = a
                    val_max = val
                    optimal_policy_found = False
        if optimal_policy_found:
            break
print(pi)
```

Output:

```
[1, 1, 0, 1, 1]
```