AMRITA
VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY

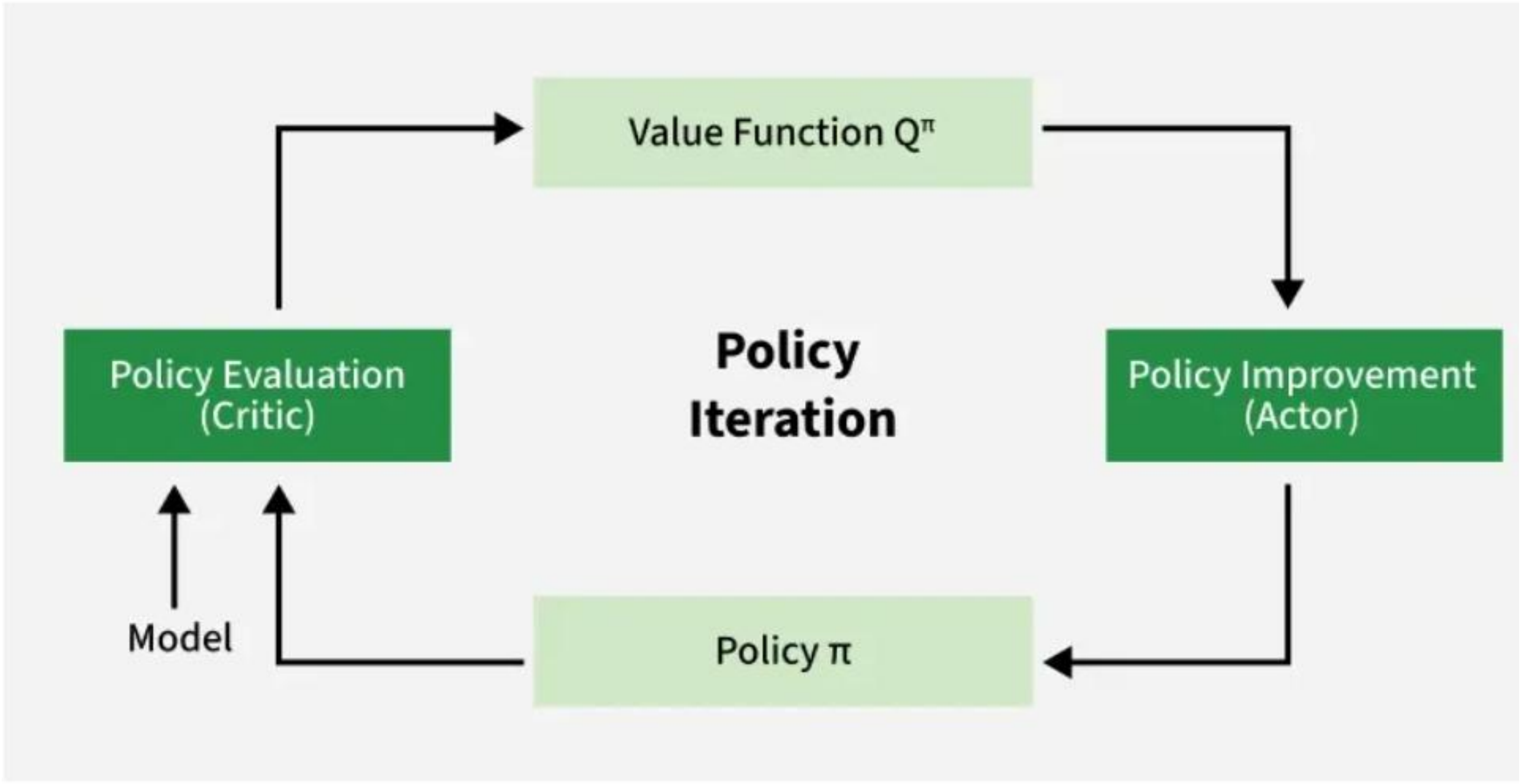
Policy Iteration

Policy Iteration

Solving MDP using Policy Iteration

- A policy is a mapping from states to actions.
- Optimal policy - for every state, there is no other action that gets a higher sum of discounted future rewards.
- For every MDP there exists an optimal policy.
- Solving an MDP is finding an optimal policy.
- Policy iteration is a recursive process of policy evaluation and improvement
- Proposed by Howard (1960)

Policy Iteration



Policy Iteration

Policy Iteration

- **Policy Evaluation**: For a given policy π, the value function $V^\pi(s)$ is computed using the Bellman Expectation Equation:

$$V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s'} P(s'|s, \pi(s)) V^\pi(s')$$

- **Policy Improvement**: Once the value function for the current policy is calculated the policy is updated to improve it by selecting the action that maximizes the expected return from each state:

$$\pi'(s) = \arg\max_a \left[R(s, a) + \gamma \sum_{s'} P(s'|s, a) V^\pi(s')\right]$$

This process repeats until the policy converges meaning it no longer changes between iterations.

Algorithm 12 (Policy iteration)

```
Input : MDP $M = \langle S, s_0, A, P_a(s' \mid s), r(s, a, s') \rangle$
Output : $Policy\ \pi$

Set $V^\pi$ to arbitrary value function; e.g., $V^\pi(s) = 0$ for all $s$
Set $\pi$ to arbitrary policy; e.g. $\pi(s) = a$ for all $s$, where $a \in A$ is an arbitrary act

repeat
    Compute $V^\pi(s)$ for all $s$ using policy evaluation
    for each $s \in S$
        $\pi(s) \leftarrow \text{argmax}_{a \in A(s)} Q^\pi(s, a)$
until $\pi$ does not change
```

Policy Iteration Algorithm

- Objective: Find the optimal policy
- Create a random policy by selecting a random action for each state.
- Until Convergence (no changes in policy)
- (a) Compute the value for each state given in the current policy
- (b) Update state values using Bellman expectation equation
- (c) Select the optimal action for each state for the new values

V(S)← $_{s'}\, p(s'|s,\pi(s))[r(s,\pi(s),s')+\gamma v(s')]$

$$\pi(S) \leftarrow argmax_a \quad \underset{s'}{} \; p(s'|s,a)[r(s,a,s')+\gamma v(s')$$

Policy Iteration Algorithm

```
1. Initialization
        V(s)∈ℝ and π(s)∈A(s) arbitrarily for all s∈S
2. Policy evaluation
        Repeat
        Δ←0
        for each s∈S:
                $v \leftarrow V(s)$
                V(s)← $_{s'} P_{ss'}^{\pi(s)}[R_{ss'}^{\pi(\ )} + \gamma V(s')]$
                Δ←max(Δ, |v-V(s)|)
        Until Δ<θ(a small positive number)
3. Policy improvement
        policy stable ←true
        For each s∈S:
                b←π(s)
                π(s)←argmaxa $_{s'} P_{ss}^{a} [R_{ss'}^{a} + \gamma V(s')]$
                If b≠π(s),then policy stable←false
                If policy stable then stop else go to 2
```

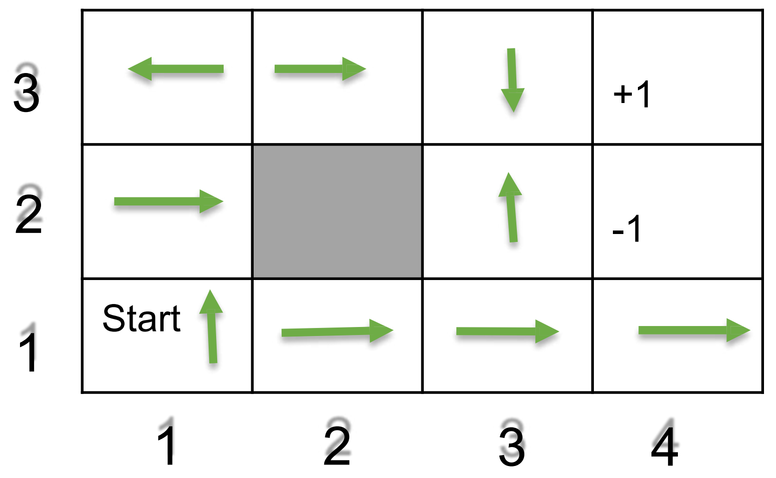
Solving MDP using Policy Iteration

- A policy is a mapping from states to actions.
- Optimal policy - for every state, there is no other action that gets a higher sum of discounted future rewards.
- For every MDP there exists an optimal policy.
- Solving an MDP is finding an optimal policy.
- Policy iteration is a recursive process of policy evaluation and improvement
- Proposed by Howard (1960)

Policy Iteration Steps

1. Initialize random policy
2. Policy Evaluation
3. Policy Improvement

Step 1: Initialize Policy



- Mapping of an action to every possible state
- In each state, agent knows what to do to achieve the reward

Step 2: Evaluate Policy

- Calculate value function for all states s∈S under the given policy

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s]$$

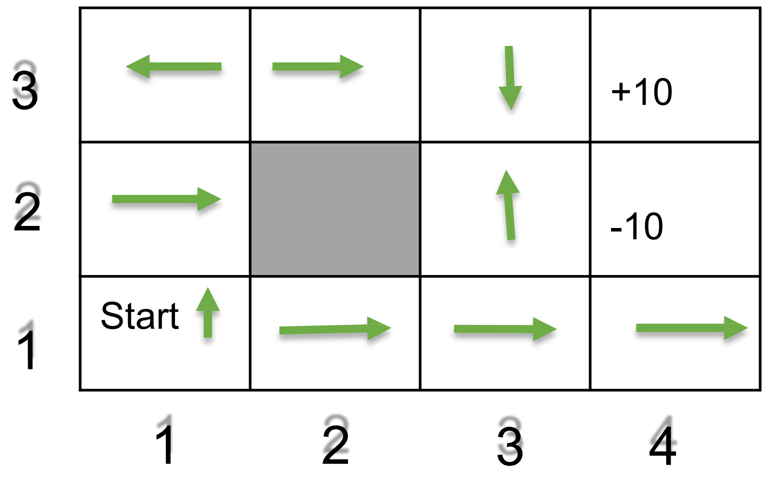
$$= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]$$

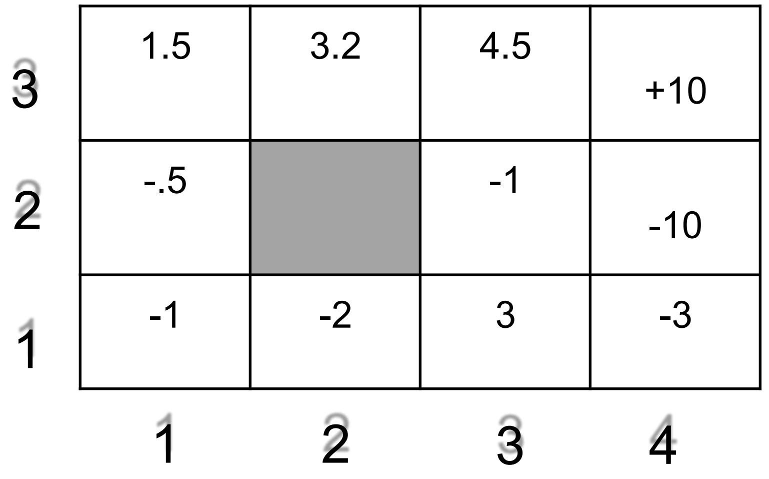
$$= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_\pi(s')\Big]$$

- iterative solution

$$v_{k+1}(s) \doteq \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s]$$

$$= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_k(s')\Big]$$

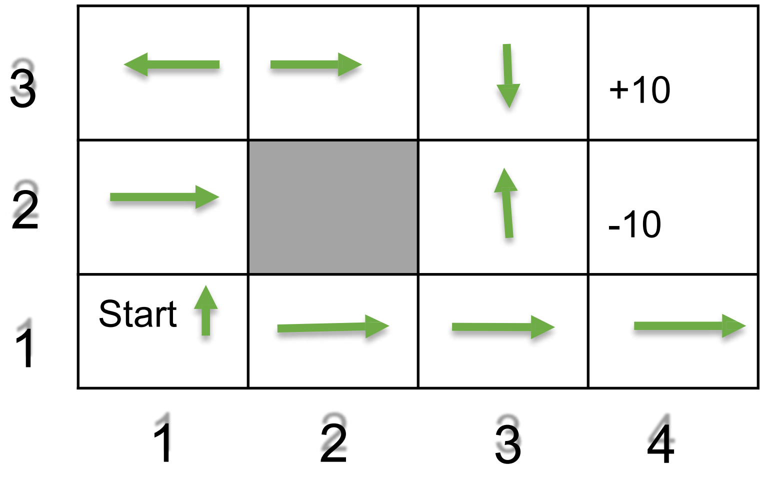




Step 2: Evaluate Policy

- Calculate value function for all states s∈S under the given policy
- Value function using Bellman's equation

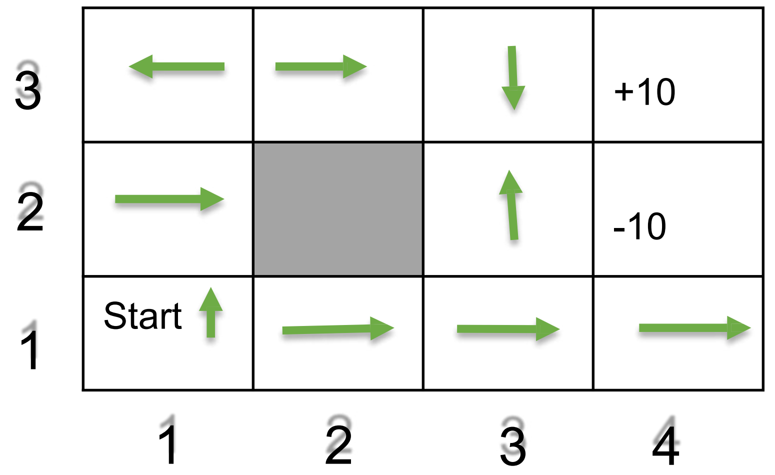
```
1. Input: Policy to be evaluated
2. Initialize V(s)
3. Repeat
        Δ←0
        for each s ∈ S:
        $v \leftarrow V(s)$
        $V(s) \leftarrow {}_{a}\, \pi(a|s)\ {}_{s'} P^{\pi(s)}_{ss'} [R^{\pi(s)}_{ss'} + \gamma V(s')]$
        Δ←max(Δ,| $v$ -V(s) |)
   Until Δ < θ
```

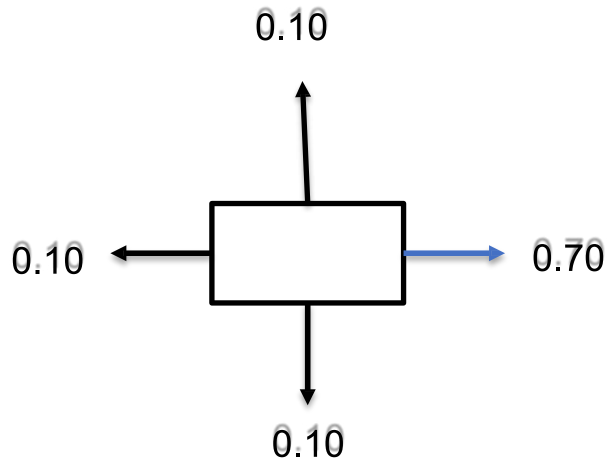


3	1.5	3.2	4.5	+10
2	-.5	▨	-1	-10
1	-1	-2	3	-3
	1	2	3	4

Step 2: Evaluate Policy

- Find the optimal policy for a 3x4 grid
- Nine states s(x, y)
- Four actions: up, down, left, right
- For a given action, the probability that action will be done is 0.70. and the other actions will have the probability at 0.10.
- If an agent is at the goal s(4,3), the agent will stop with probability of 1.
- Reward of non terminal states= 0.1
- Discount factor (γ) equals 0.9.

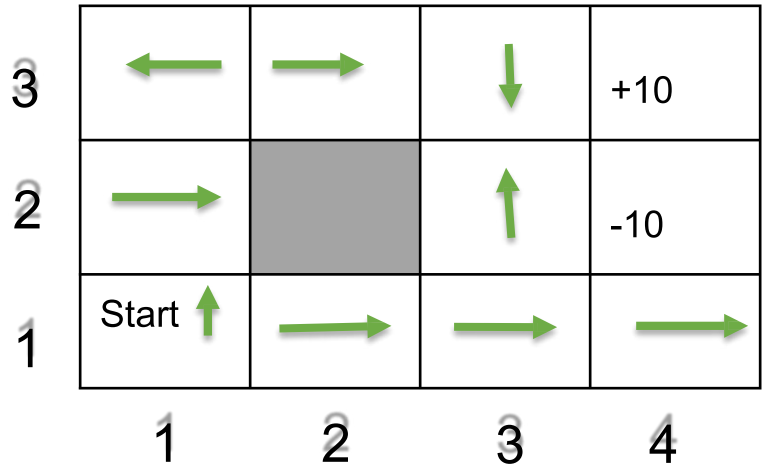


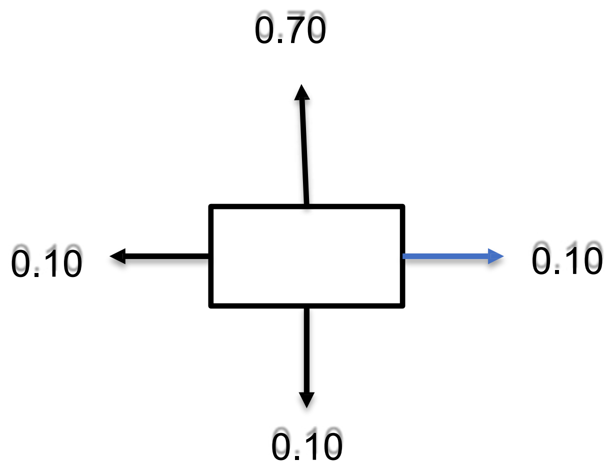


Step 2: Evaluate Policy

- Calculate Value Functions
- V(1, 1)

$$= 0.7(.1+0.9\, v(1, 2))$$
$$+0.1(.1+0.9\, v(1, 1))$$
$$+0.1(.1+0.9\, v(1, 1))$$
$$+0.1(.1+0.9\, v(2, 1))$$



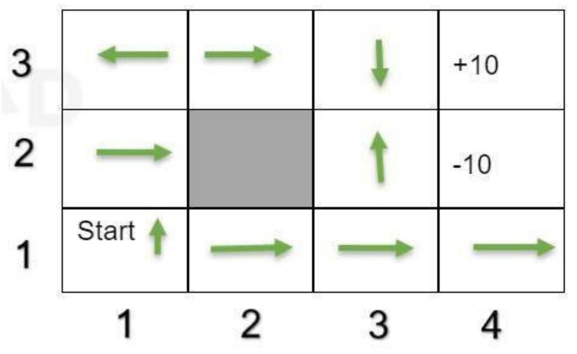


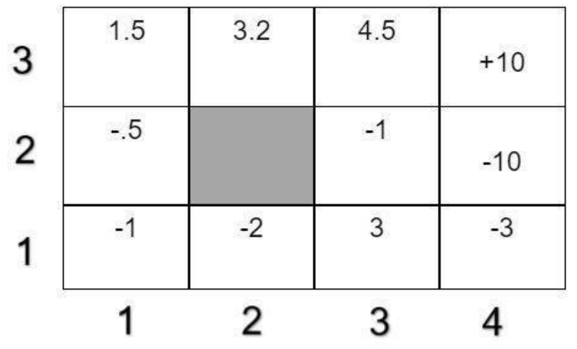
Step 3: Policy Improvement

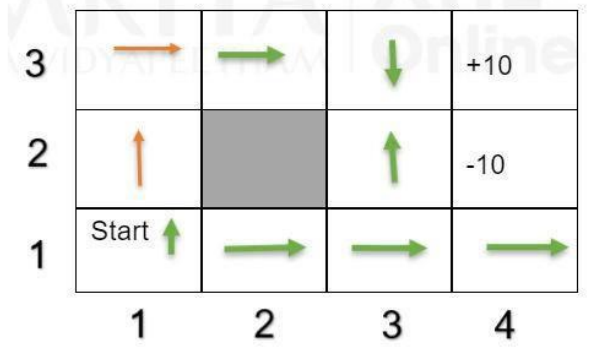
- Find a better action for states s∈S

$$\pi(S) \leftarrow argmax_a \underset{s'}{\quad} p(s'|s,a)[r(s,a,s') + \gamma v(s')]$$

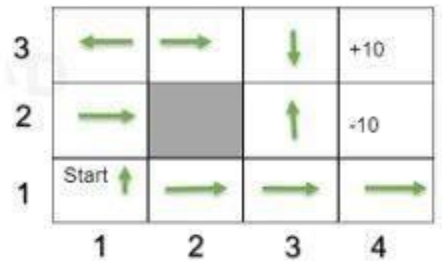
- If the best action is better than the action in current policy, replace the current action by the best action.

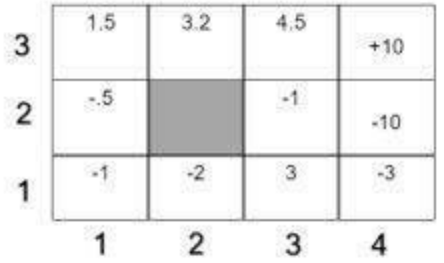


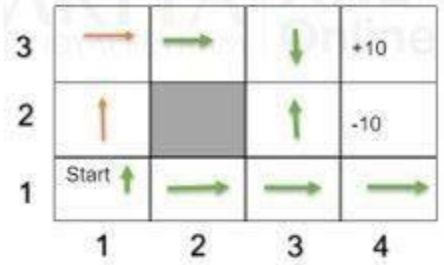




Step 3 : Policy Improvement







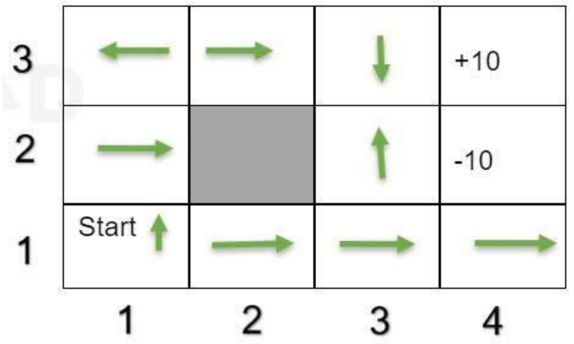
Asimple policy:

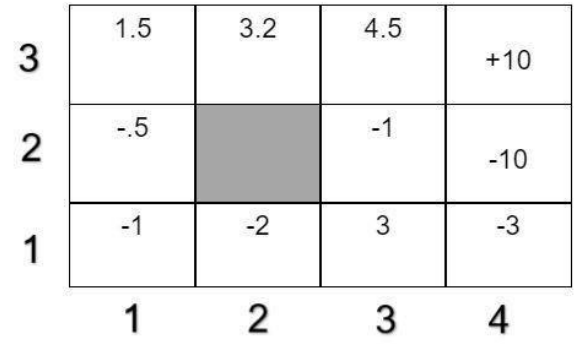
$$\pi_0(1,1) = go \text{ up}, \ (2,1) = go\ right,$$
$$\pi_0(3,1) = go \text{ right}, \pi_0(3,2) = go\ up$$
$$\pi_0(3,3) = go \text{ down}, \pi_0(4,3) = stop$$

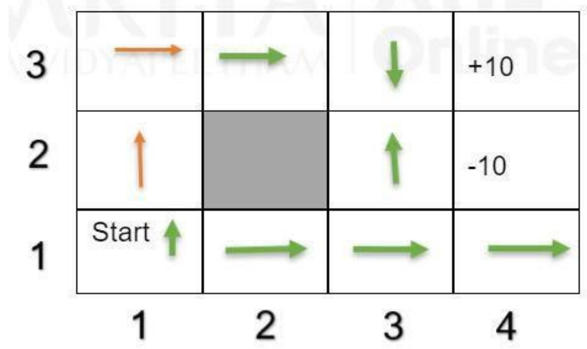
$$\pi_1(1,1)=\text{argmax} \begin{cases} \text{right: } 0.7(.1+0.9 \times -5)+0.1(.1+0.9 \times -1) \ +0.1(.1+0.9 \times -1) \ +0.1(.1+0.9 \times -2) \\ \text{up:} \\ \text{down:} \\ \text{left:} \end{cases}$$

Step 3: Policy Improvement

```
policy stable ← true
For each s∈S:
        b ← π(s)
        π(s) ← argmax a  $_{s'} P^{a}_{ss'} [R^{a}_{ss'} + \gamma V(s')]$
        If b ≠ π(s) then
        policy stable ← false
        If policy stable then
            Stop
        else
            Do Policy Evaluation
```





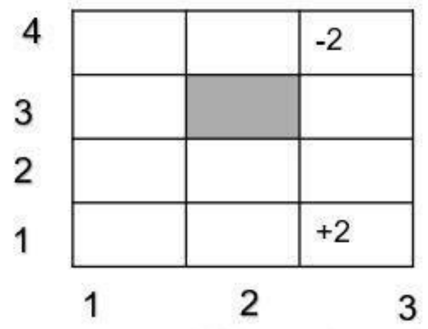


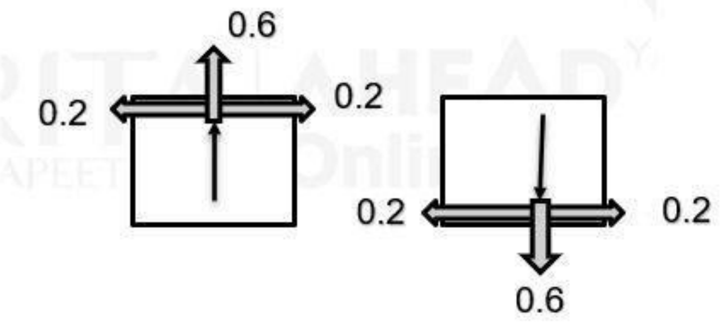
Policy Iteration Algorithm

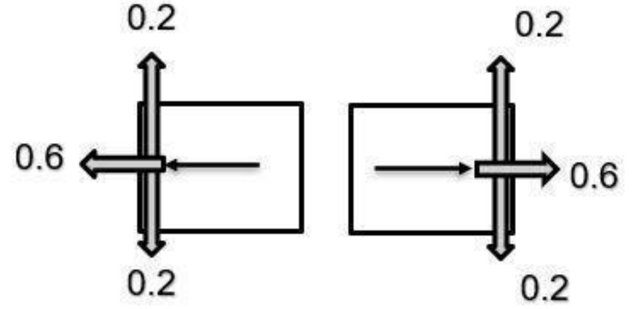
1. Initialize random policy
2. Policy Evaluation
3. Policy Improvement
4. Repeat steps 2 and 3, until convergence

Example MDP

- Environment 3 x 4 grid, one blocked state – 11 states
- Two terminal states: (3, 4), (3, 1)
- Actions: up, down, left, right
- 0.6 to reach extended effect
- 0.2 probability to move at right angle of extended direction
- If the agents bumps into a wall, it says there
- Reward:
- For terminal states +/- 2
- Other states: -0.5

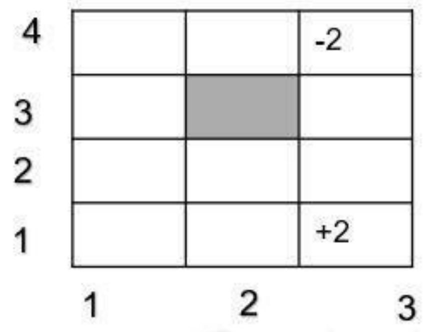


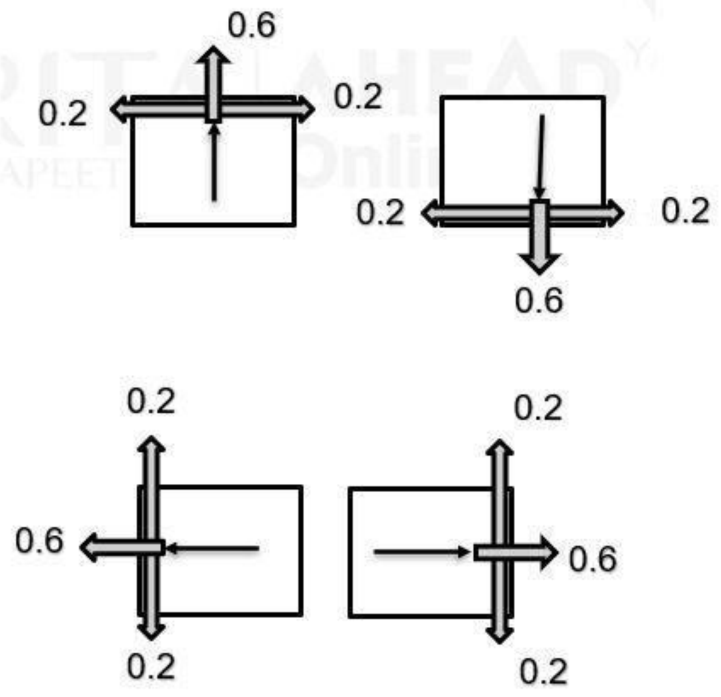




Example MDP

- Environment 3 x 4 grid, one blocked state – 11 states
- Two terminal states: (3, 4), (3, 1)
- Actions: up, down, left, right
- 0.6 to reach extended effect
- 0.2 probability to move at right angle of extended direction
- If the agents bumps into a wall, it says there
- Reward:
- For terminal states +/- 2
- Other states: -0.5



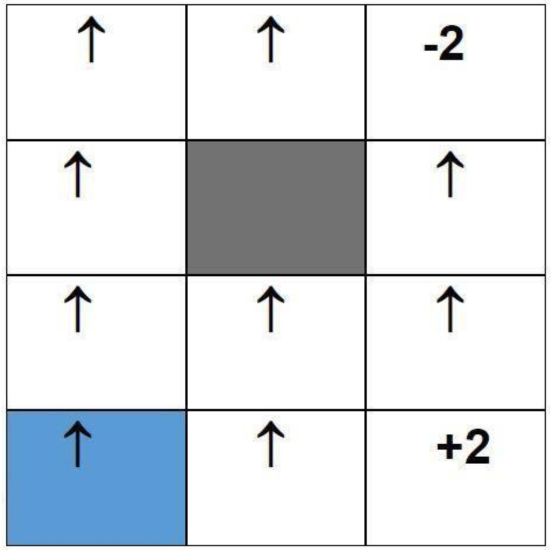


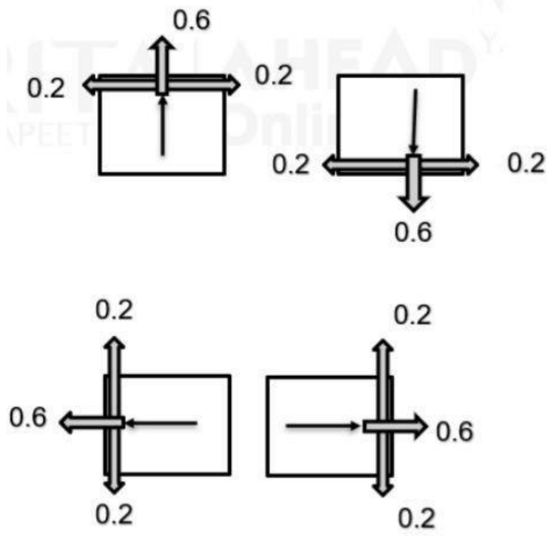
Step 1:Initial Policy

↑	↑	-2
↑	▨	↑
↑	↑	↑
↑	↑	+2

Policy Evaluation

$$V(1,1) = -0.5 + 1\text{x}$$
$$\{$$
$$0.6 \text{ x } v(1,2)+$$
$$0.2 \text{ x } v(2,1)+$$
$$0.2 \text{ x } v(1,1)$$
$$\}$$
$$=-0.5+1\text{x}\{0.6\text{x}0+0.2\text{x}0+0.2\text{x}0\}$$
$$=-0.5$$

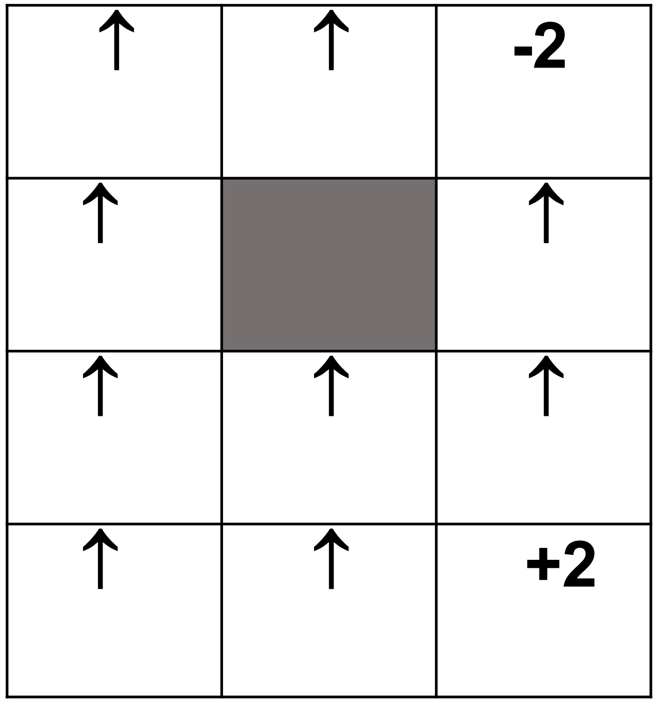




$$V(S) \leftarrow {}_{s'}\, p(s'|s,\pi(s))[r(s,\pi(s),s') + \gamma v(s')]$$

Policy Evaluation

- *V* (2,1) = −0.5 + 1x{o.6xv(2,2)+0.2xv(3,1)+0.2xv(1,1)}
    - =-0.5+1x{0.6x0+0.2x2+0.2x0}
    - =-0.5+1x0.4
    - =-0.1
- *V* (3,2) = −0.5 + 1x{o.6xv(3,2)+0.2xv(2,1)+0.2xv(3,1)}
    - =-0.5+1x{0.6x0+0.2x0+0.2x2}
    - =-0.5+0.4=-0.1



Policy Evaluation

V (3,3) = – 0.5 + 1x{o.6xv(3,4)+0.2xv(3,3)+0.2xv(3,3)}
=-0.5+1x{0.6x-2+0.2x0+0.2x0}
=-0.5+(-1.2)=-0.5-1.2
=-1.7

V (2,4) = – 0.5 + 1x{o.6xv(3,4)+0.2xv(1,4)+0.2xv(3,4)}
=-0.5+1x{0.6x0+0.2x0+0.2x-2}
=-0.5-0.4=-0.9

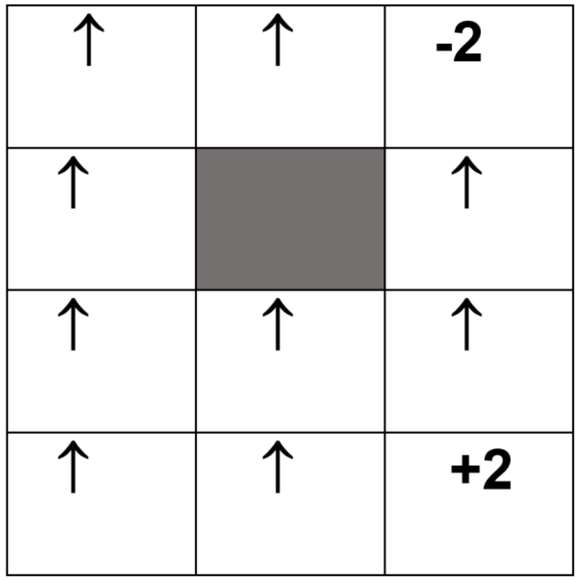
-0.5	-0.9	-2
-0.5	▨	-1.7
-0.5	-0.5	-0.1
-0.5	-0.1	2

Policy Improvement

$\pi(S) \leftarrow argmax_a \sum_{s'} p(s'|s,a)[r(s,a,s') + \gamma v(s')]$

$$\Pi(1) = \text{argmax}\,[-0.5 + 1x\,] \begin{cases} \text{UP} \\ \text{DOWN} \\ \text{LEFT} \\ \text{RIGHT} \end{cases}$$

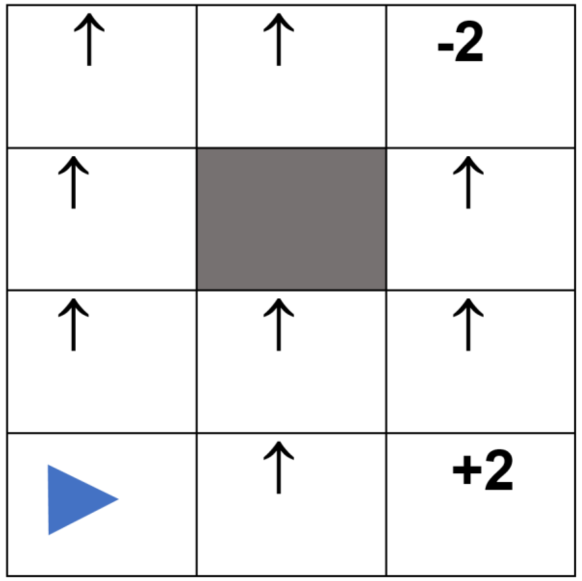
$0.6 \times V(1,2) + 0.2 \times V(1,1) + 0.2 \times V(2,1)$



$$= \text{argmax}\,[-0.5] \begin{cases} 0.6x\text{-}0.5+0.2x\text{-}0.5+0.2x\text{-}0.1 \\ 0.6x\text{-}0.5+0.2x\text{-}0.5+0.2x\text{-}0.1 \\ 0.6x\text{-}0.5+0.2x\text{-}0.5+0.2x\text{-}0.5 \\ 0.6x\text{-}0.5+0.2x\text{-}0.5+0.2x\text{-}0.5 \end{cases}$$

$$= \text{argmax}\,[-0.5] \begin{cases} -0.42 \\ -0.42 \\ -0.50 \\ -0.25 \end{cases}$$

RIGHT

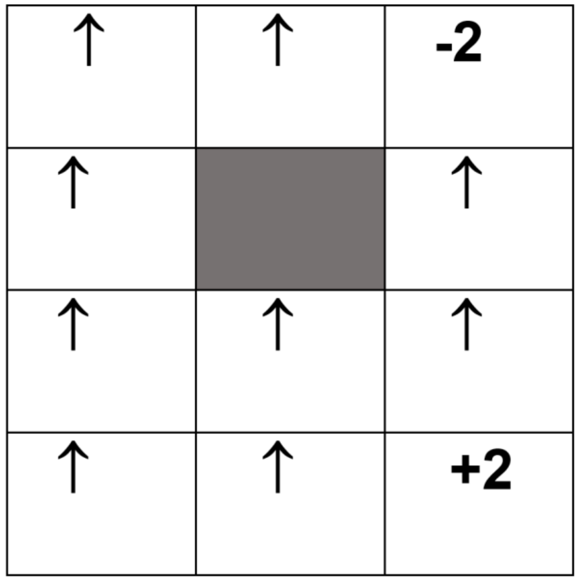


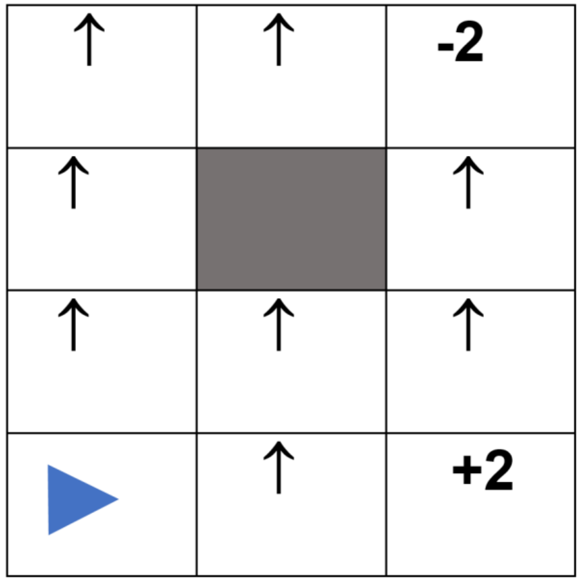
Policy Improvement

$$V(2,1) = \text{argmax}(-0.5 + 1x) \begin{cases} \text{UP } 0.6 \times V(2,2) + 0.2 \times V(1,1) + 0.1 \times V(3,1) \\ \text{DOWN } 0.6 \times V(2,1) + 0.2 \times V(1,1) + 0.2 \times V(2,1) \\ \text{LEFT } 0.6 \times V(1,1) + 0.2 \times V(2,2) + 0.2 \times V(2,1) \\ \text{RIGHT } 0.6 \times V(3,1) + 0.2 \times V(2,2) + 0.2 \times V(2,1) \end{cases}$$

$$= \text{argmax}(-0.5) \begin{cases} 0.6x\text{-}0.5+0.2x\text{-}0.5+0.2x2 \\ 0.6x\text{-}0.1+0.2x\text{-}0.5+0.2x\text{-}0.1 \\ 0.6x\text{-}0.5+0.2x\text{-}0.5+0.2x\text{-}0.1 \\ 0.6x2+0.2x\text{-}0.5+0.2x\text{-}0.1 \end{cases}$$

$$= \text{argmax}(-0.5) \begin{cases} 0 \\ -0.42 \\ -0.42 \\ 0 \end{cases} \quad = -0.50 + 0 = -0.5$$

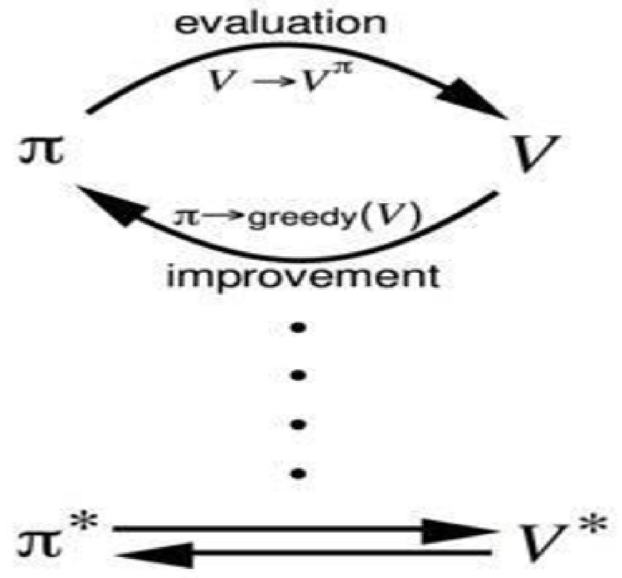


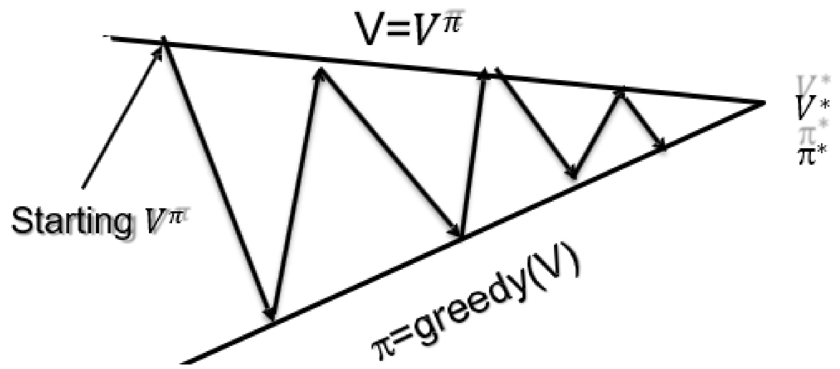


Since 0 is max value for up and right so input policy is up and right

Generalize Policy Iteration

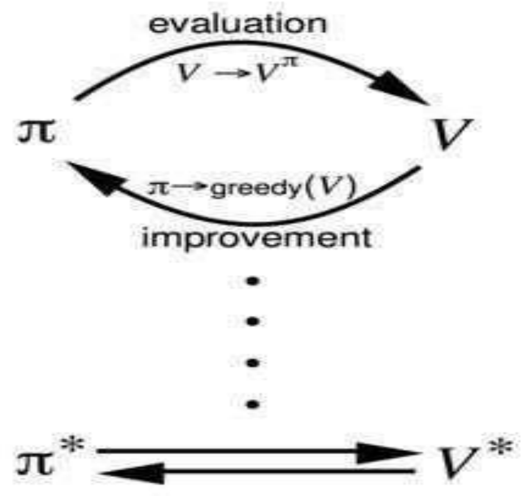
- Policy iteration consists of two simultaneous, interacting processes
  - Policy Evaluation makes the value function consistent with current policy
  - Policy Improvement makes the policy greedy wrt the current value function
- Generalized Policy Iteration (GPI) is the idea of interacting policy evaluation and policy improvement processes

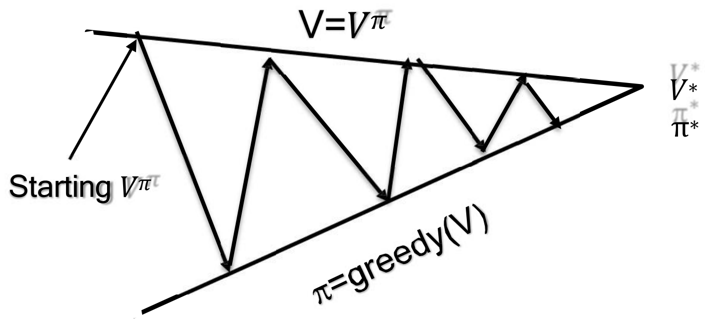




Generalized Policy Iteration

- Policy iteration consists of two simultaneous, interacting processes
  - Policy Evaluation makes the value function consistent with current policy
  - Policy Improvement makes the policy greedy wrt the current value function
- Generalized Policy Iteration (GPI) is the idea of interacting policy evaluation and policy improvement processes





Policy Iteration vs. Value Iteration

```
1. Initialization
        V(s)∈ℝ and π(s)∈A(s) arbitrarily for all s∈S
2. Policy evaluation
        Repeat
            Δ→0
            for each s∈S:
            $v \leftarrow V(s)$
            V(s)← $_{s'} P^{\pi(s)}_{ss'}[R^{\pi(s)}_{ss'} + \gamma V(s')]$
            Δ→max(Δ,|v-V(s)|)
        Until Δ<θ(a small positive number)
3. Policy improvement
        policy stable →true
        For each s∈S:
            b→π(s)
            π(s)→argmax $_a$ $_{s'} P^a_{ss'} [R^a_{ss'} + \gamma V(s')]$
            If b≠π(s),then policy stable→false
            If policy stable then stop else go to 2
```

```
1. Initialization v arbitrarily v(s)=0 for all s∈$S^+$
2. Repeat
        Δ→0
        for each s∈S:
        $v \leftarrow V(s)$
        V(s)← max$_a$ $_{s'} P^{\pi(s)}_{ss'} [R^{\pi(s)}_{ss'} + \gamma V(s')]$
        Δ→max(Δ,|v-V(s)|)
    Until
        Δ<θ(a small positive number)
3. Output a deterministic policy π,such that
    π(s)= argmax $_a$ $_{s'} P^a_{ss'} [R^a_{ss'} + \gamma V(s')]$
```

Policy Iteration vs. Value Iteration

Policy Iteration	Value Iteration
choose an arbitrary policy - iteratively evaluate and improve the policy	compute the optimal state value function by iteratively updating the estimate
Complex algorithm	Simpler Algorithm
Requires few iterations to converge	Requires more iterations to converge
Guaranteed to converge	Guaranteed to converge
Faster	Slower

Dynamic Programming E ficiency

- Worst case time to find an optimal policy is polynomial in the number of states and actions
- Curse of dimensionality : Number of states grows exponentially with number of state variables

Namah Shivaya