Markov Decision Process II

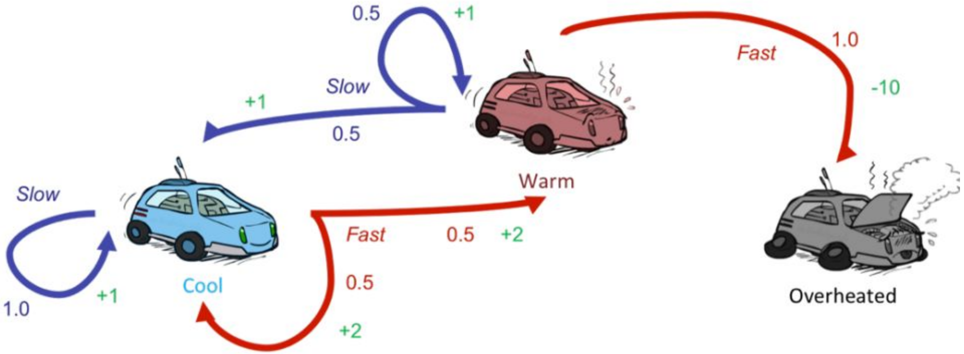
CSE 4617: Artificial Intelligence

Isham Tashdeed
Lecturer, CSE

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V_k(s')\right]$$

Value Iteration

V_2	3.5	2.5	0
V_1	2	1	0
V_0	0	0	0

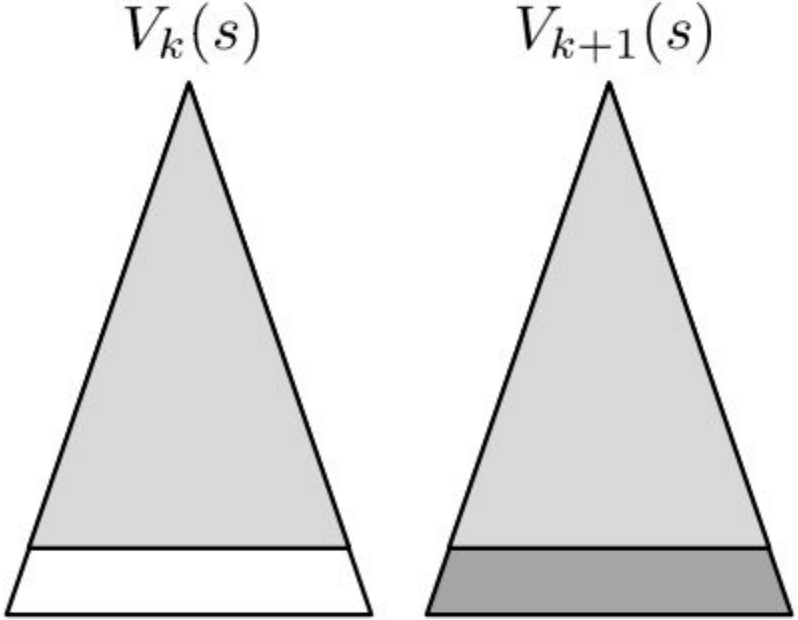


Assume no discount!

Will It Converge?

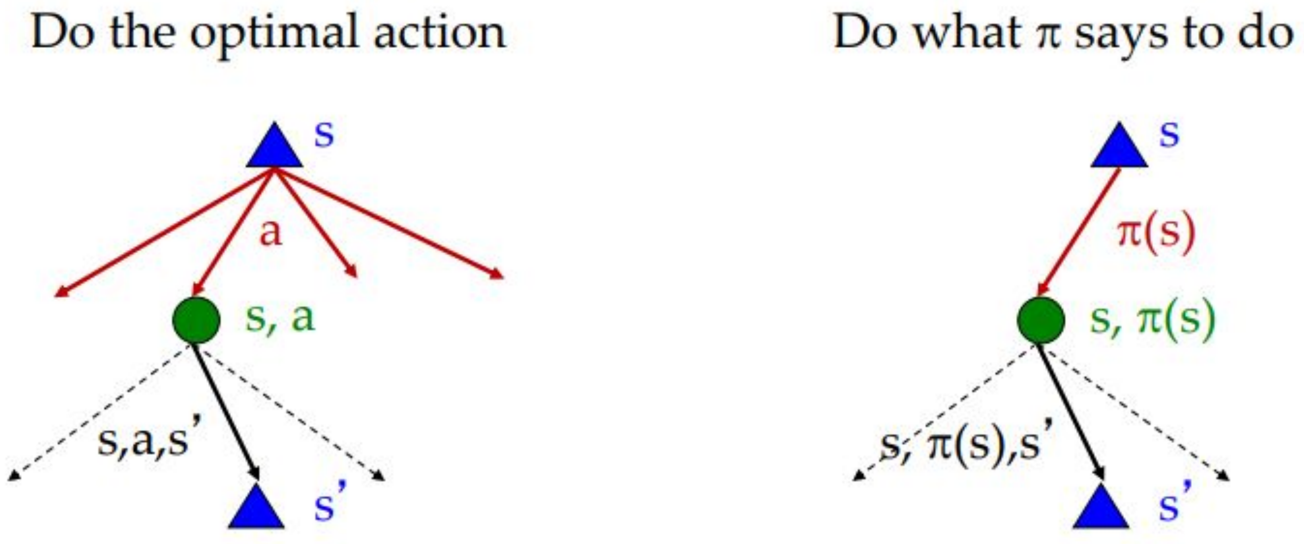
How do we know that the V_k vectors are going to converge?

- Case 1 → Time limited MDPs (Always terminates after m steps)
- Case 2 → Value of $0 < \gamma < 1$
  - The last layer is at best R_{max} or at worst R_{min}
  - The difference between the two trees is at most $\gamma^k \max|R|$



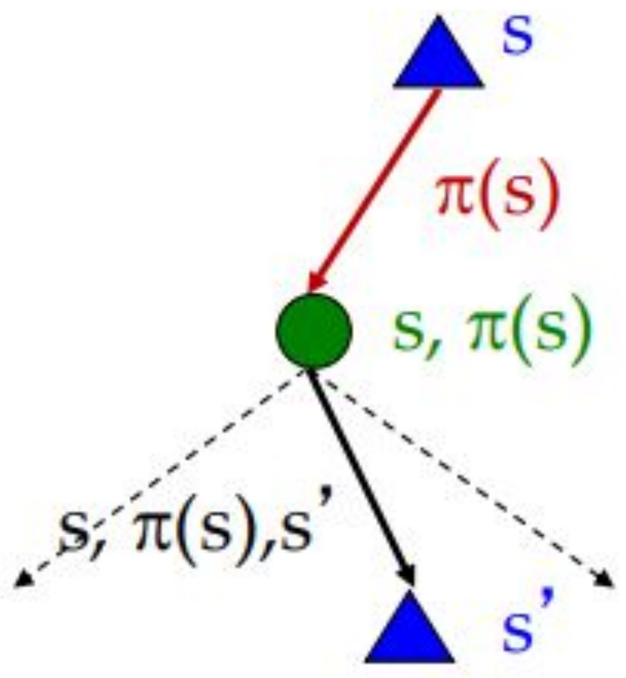
Policy Evaluation

Fixed Policies



- If we have some fixed policy $\pi(s)$ → Only one action per state, No need to max over all actions
- For regular value iteration → $O(S^2A)$ per iteration

Utilities of a Fixed Policy



- Compute the utility of a state *s* under a fixed policy π → usually a non-optimal policy
- $V^\pi(s)$ → Total expected utility when starting from state *s* and following π
- $V^\pi(s) = \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^\pi(s')]$
- How to calculate $V^\pi(s)$ or $V^\pi(s')$?
- $V_0^\pi(s) = 0$

  $V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V_k^\pi(s')]$
- Efficiency → $O(S^2)$ per iteration
- Or try solving the linear equation!

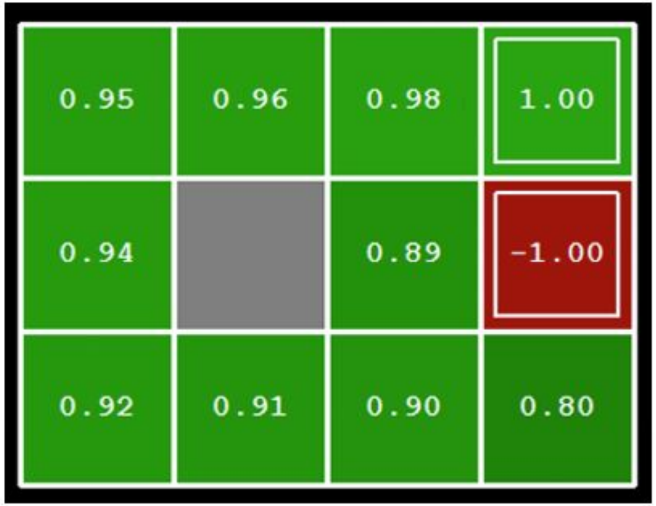
Policy Evaluation

Always Go Right

Always Go Forward

Policy Extraction

Policy Extraction

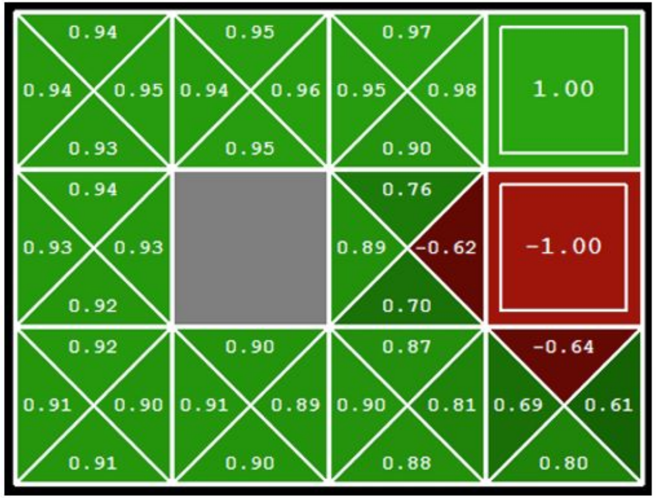


- Given the optimal values $V^*(s)$, how to extract the policy?
- We need to run a one step expectimax to find the policy

$$\pi^*(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V^*(s')\right]$$

- This is policy extraction

Policy Extraction



- Given the optimal q-values $Q^*(s,a)$, how to extract the policy?
- We need to run a one step expectimax to find the policy

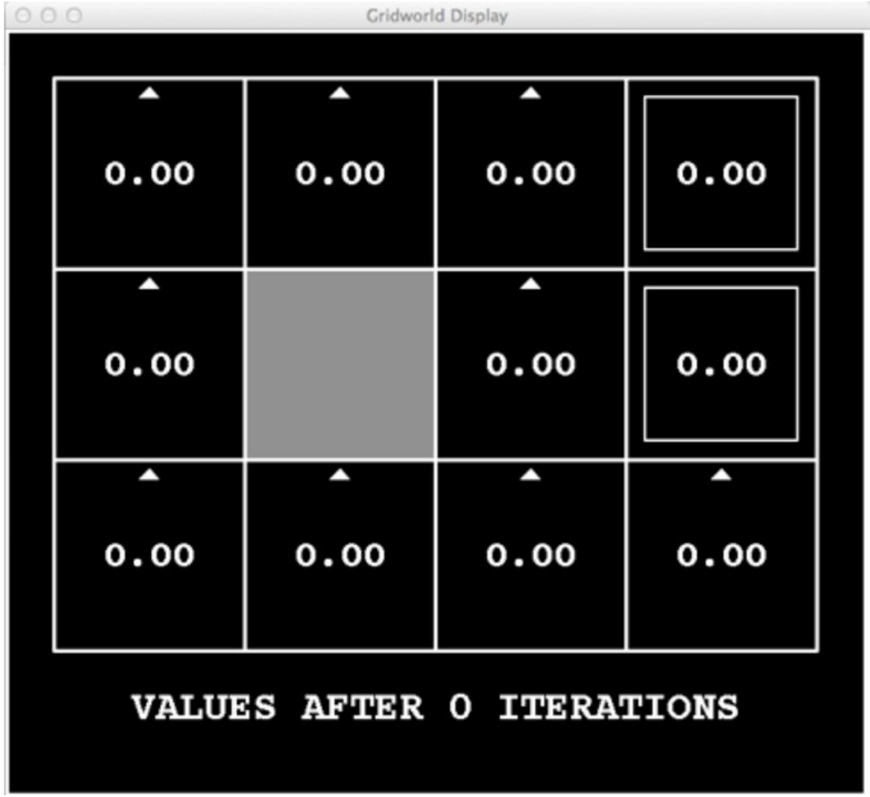
$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q^*(s,a)$$

- Actions are much easier to select from q-values, than values
- This is the very basic of reinforcement learning

Policy Iteration

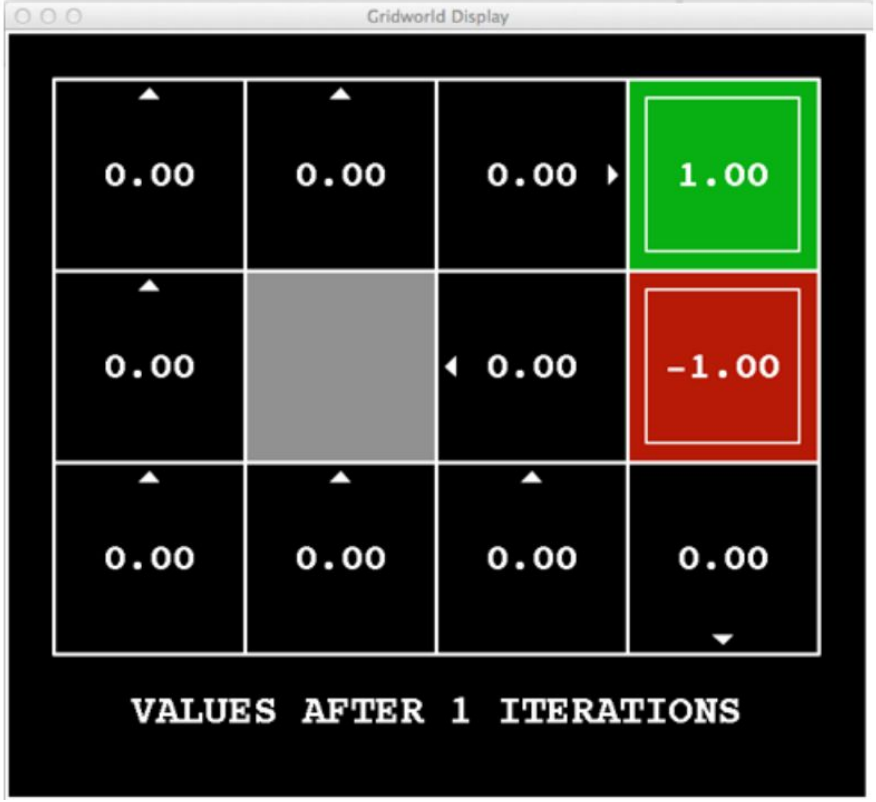
Gridworld Value Iteration

k = 0



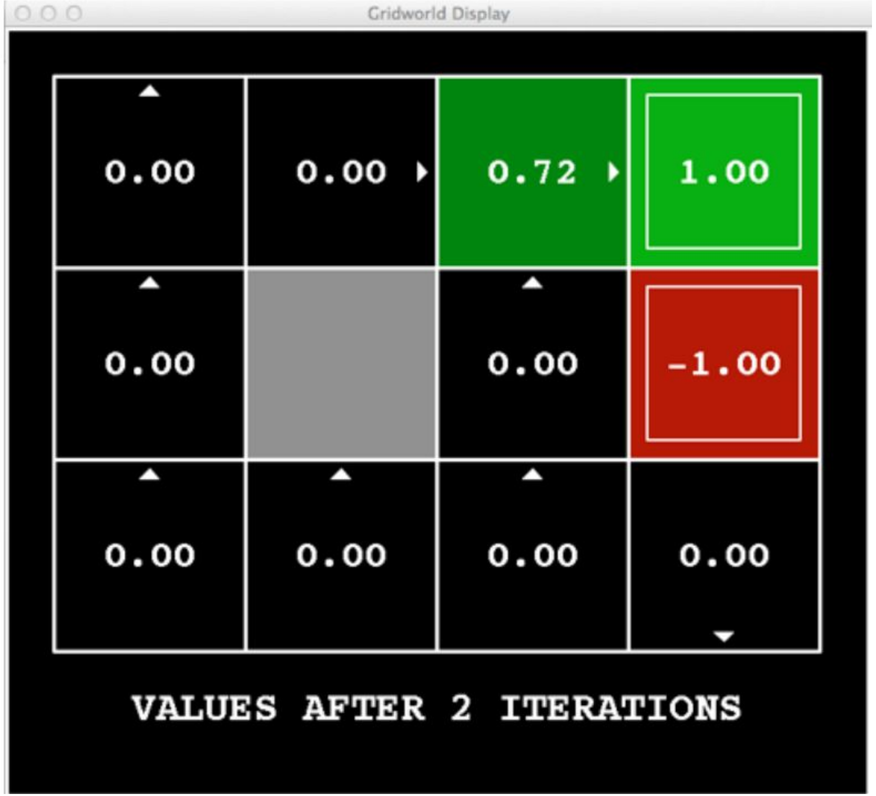
Gridworld Value Iteration

$k = 1$



Gridworld Value Iteration

k = 2

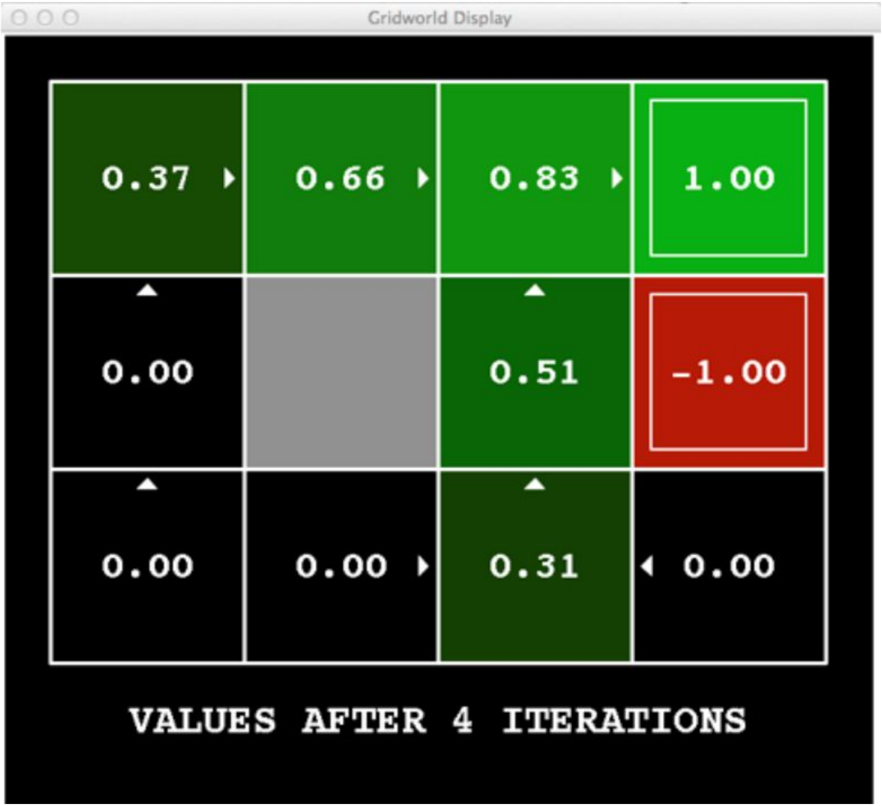


Gridworld Value Iteration

k = 3

Gridworld Value Iteration

k = 4

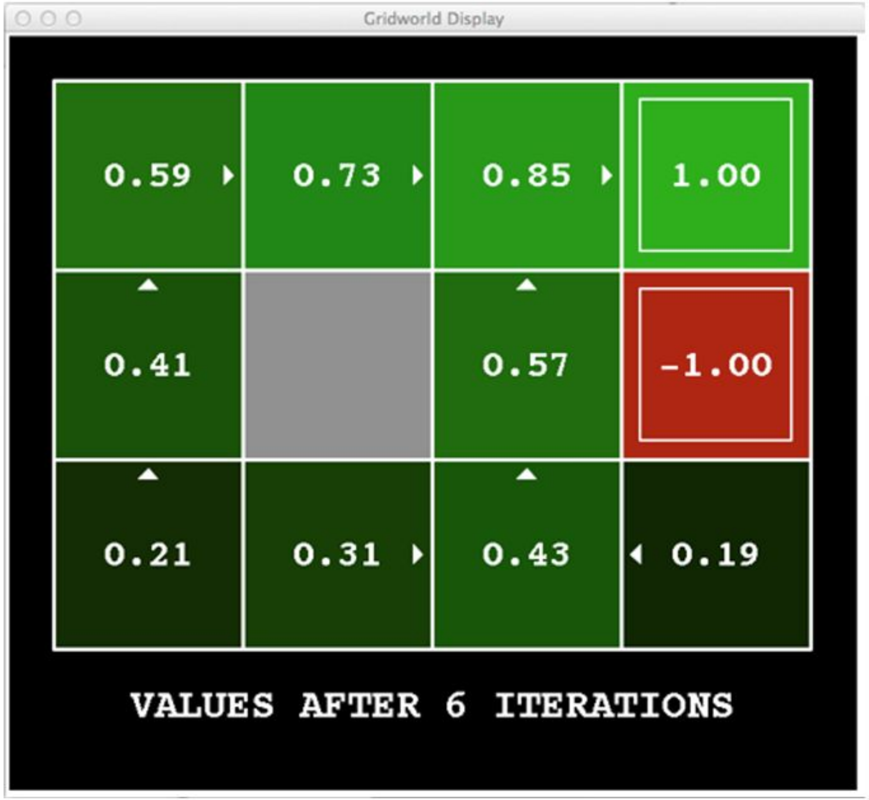


Gridworld Value Iteration

k = 5

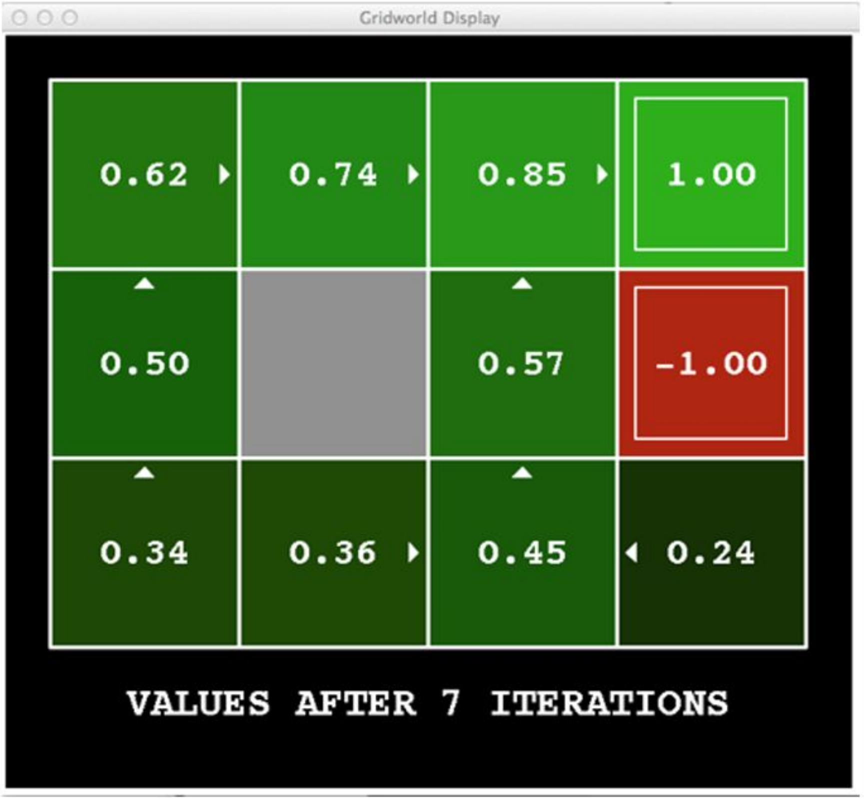
Gridworld Value Iteration

k = 6



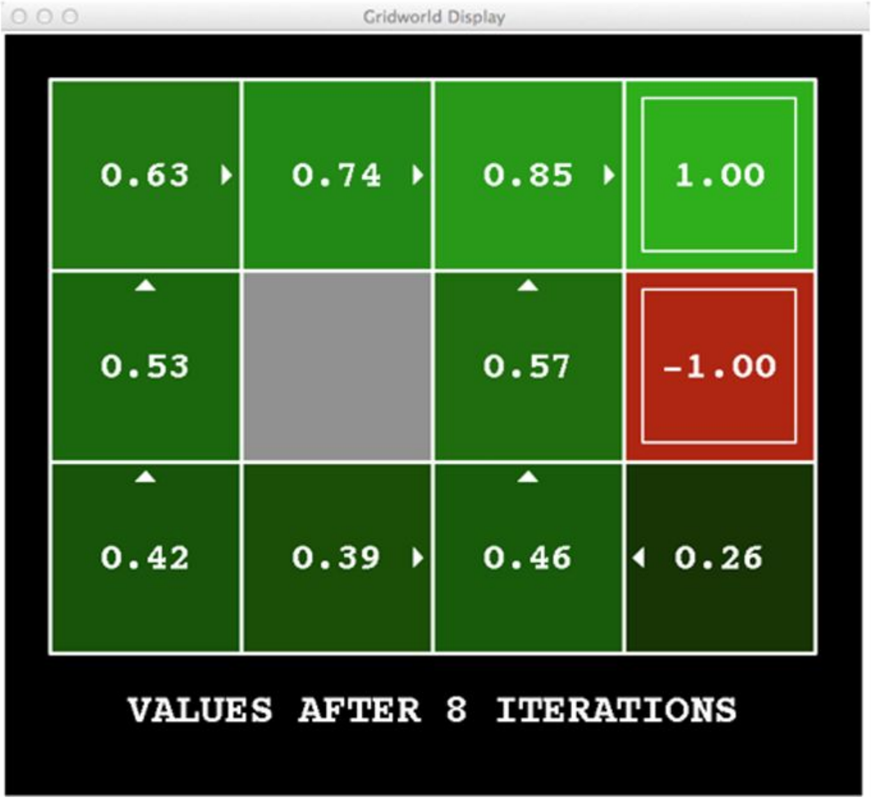
Gridworld Value Iteration

k = 7



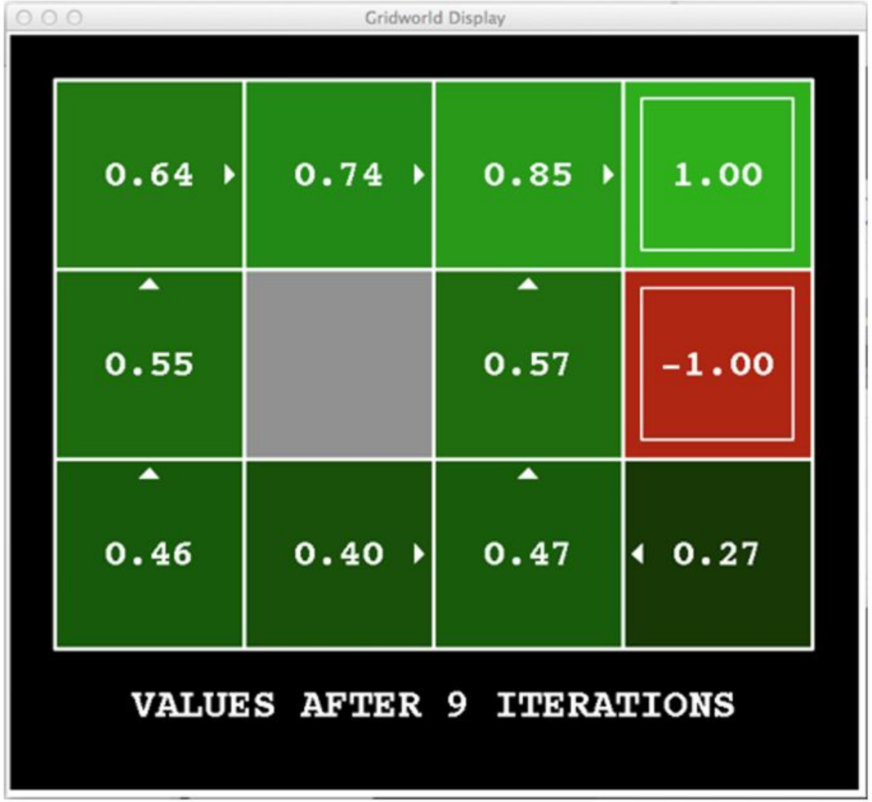
Gridworld Value Iteration

k = 8



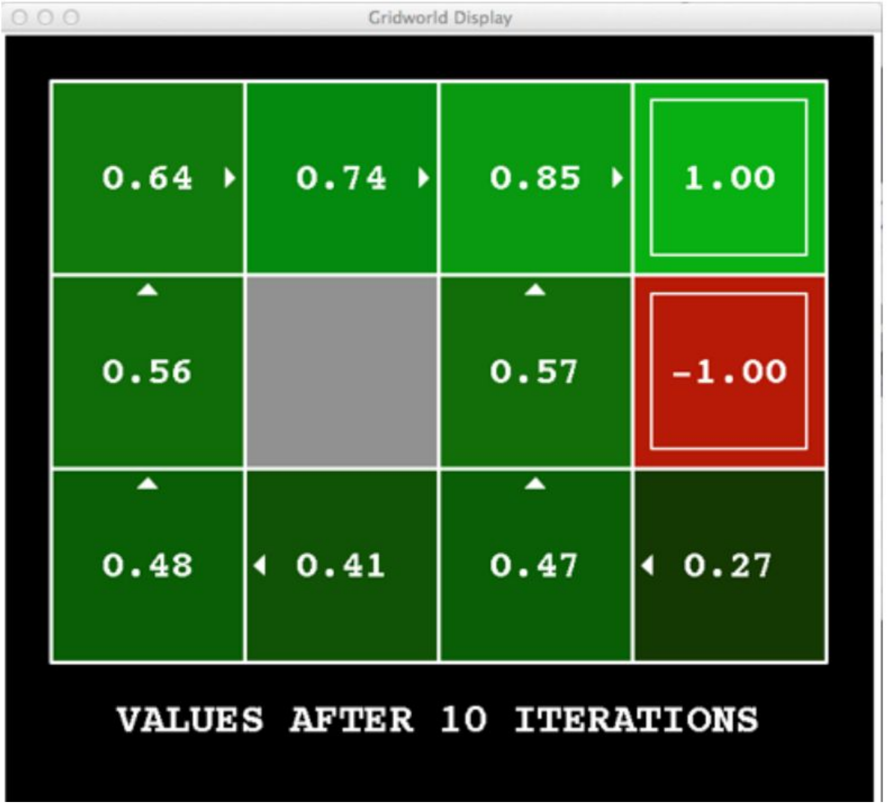
Gridworld Value Iteration

k = 9

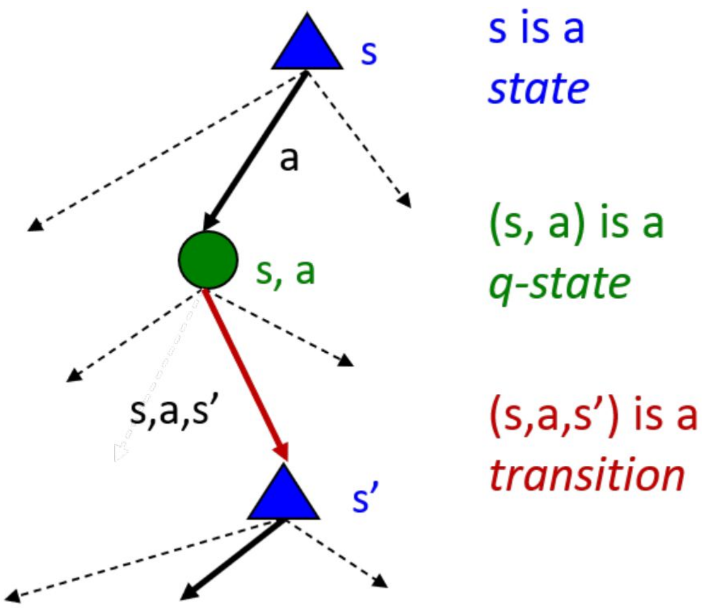


Gridworld Value Iteration

$k = 10$



Problems with Value Iteration



- Repeats the Bellman updates at every state each iteration
- It is slow → $O(S^2 A)$
- The direction at each state rarely changes
- Policy converges long before the values converge

Policy Iteration

- Step 1: Policy evaluation → Calculate utilities for some fixed policy until convergence (Not optimal policy)
- Step 2: Policy improvement → Update policy using one-step look-ahead with resulting converged (but not optimal) utilities as future values
- Repeat until policy converges

Policy Iteration

- Evaluation: for fixed current policy π, find the values with policy evaluation:
  - Iterate until convergence

$$V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} T(s, \pi_i(s), s') \left[R(s, \pi_i(s), s') + \gamma V_k^{\pi_i}(s')\right]$$

Policy Iteration

- Evaluation: For fixed current policy π, find the values with policy evaluation:
  - Iterate until convergence

$$V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} T(s, \pi_i(s), s') \left[R(s, \pi_i(s), s') + \gamma V_k^{\pi_i}(s') \right]$$

- Improvement: For fixed values, get a better policy using policy extraction:

$$\pi_{i+1}(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V^{\pi_i}(s') \right]$$

- Repeat until convergence

Policy Iteration

Both value iteration and policy iteration compute the same thing

In value iteration:

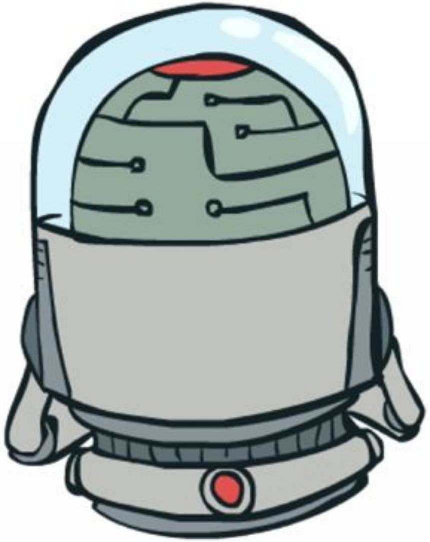
- Both utility values and policy is updated at each iteration
- Slow

In policy iteration:

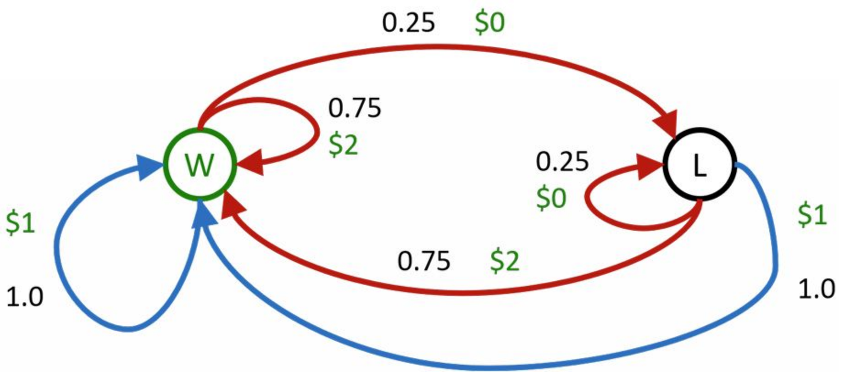
- First we update the utility values for a fixed policy → faster as we only consider one action
- After that, new policy is chosen → Slow, similar to value iteration

Reinforcement Learning

Reinforcement Learning



Reinforcement Learning

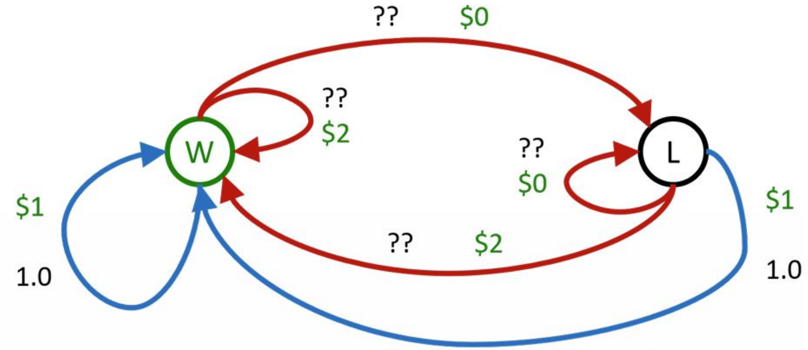


- States → Win, Lose
- Actions → Blue, Red
- $\gamma = 1$
- 100 time steps

Solving MDPs is offline planning

- You know the details of the MDP
- You find all the quantities through computation
- You don't actually play the game!

Reinforcement Learning



- States → Win, Lose
- Actions → Blue, Red
- $\gamma = 1$
- 100 time steps

Reinforcement Learning is online planning

- You don't know the details of the MDP
- You actually play the game!

$0 $0 $0 $2 $0

$2 $0 $0 $0 $0

Reinforcement Learning

In reinforcement learning

- Exploration: Try unknown actions to get information.
- Exploitation: Eventually, you have to use what you know
- Regret: Even if you learn intelligently, you make mistakes
- Sampling: You have to try things repeatedly
- Difficulty: Learning can be much harder than solving a known MDP

Thank you