Announcements

- Homework 3 is out on Gradescope
  - Due on Exam date (Thursday Oct 7th)
- Exam is next Week (Oct 7th).
  - D01 will have it available on Thursday 9am central time till Saturday midnight.
  - Will still need to do it in 90 minutes
- Exam content
  - Till lecture 11 (MDP)
- Review Session on October 4th week
  - Monday 4th 3-5 pm by TA (Solving questions and review questions)
  - Wednesday 6th 1-3 pm and 6-7 pm by me (Bring your questions session)
  - If you need anything else or cannot make it to any, by appointment
  - All will be hosted via zoom. Links are to be announced later
- Review session questions will be available on blackboard by Thursday

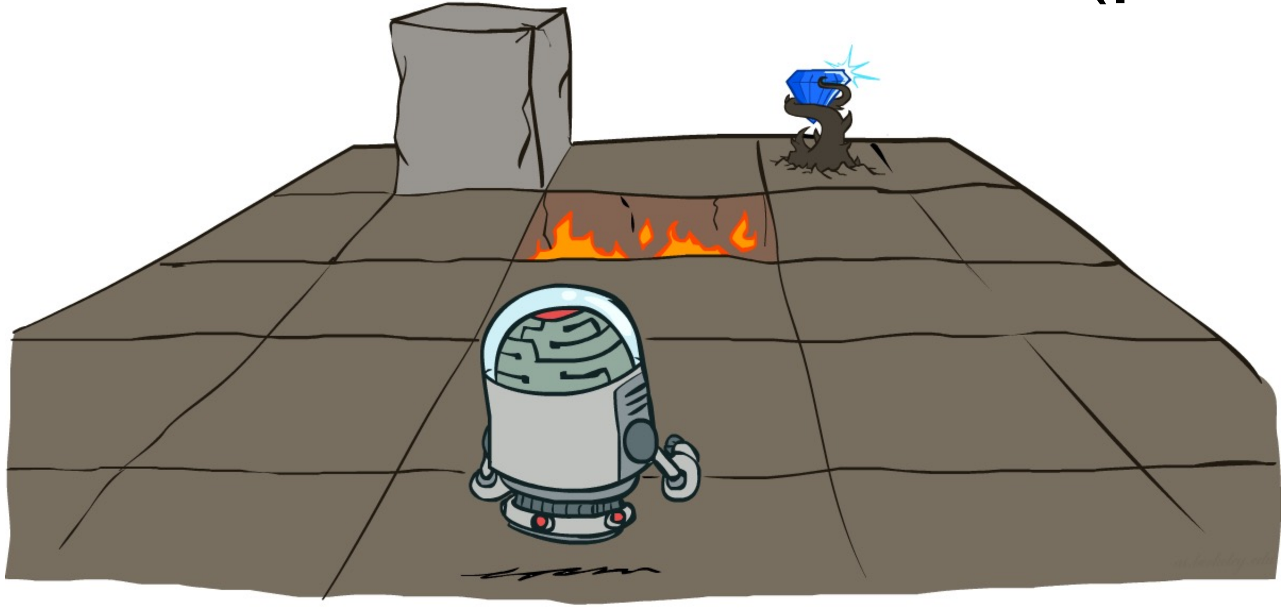
Exam Format

- You are allowed to have a cheat sheet
  - A4 two sides
- You may need a calculator so bring yours

- Exam questions are in the following format
  - 6-8 questions in total
  - One question is True or False (True:1, False:-1, Blank: 0. So, Don’t answer randomly)
  - The other 5-7 questions are solving problems
    - Similar to the exam prep/review question format

CS 3568: Intelligent Systems

Markov Decision Processes (part 3)



Instructor: Tara Salman

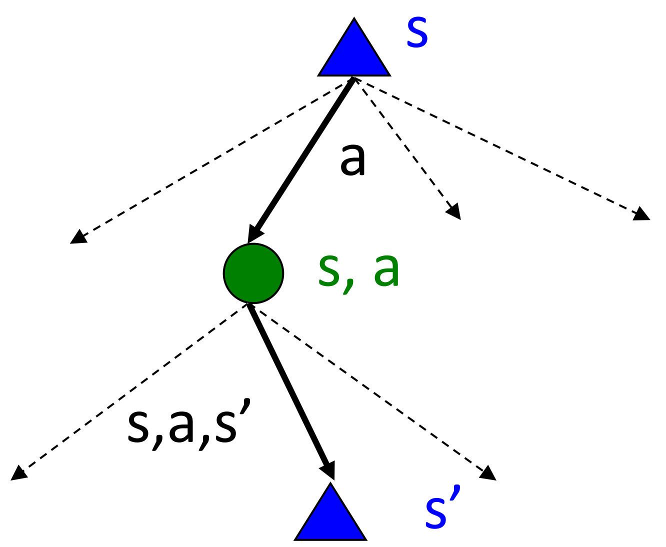
Texas Tech University

Computer Science Department

[These slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley (ai.berkeley.edu).]

Recap: MDPs

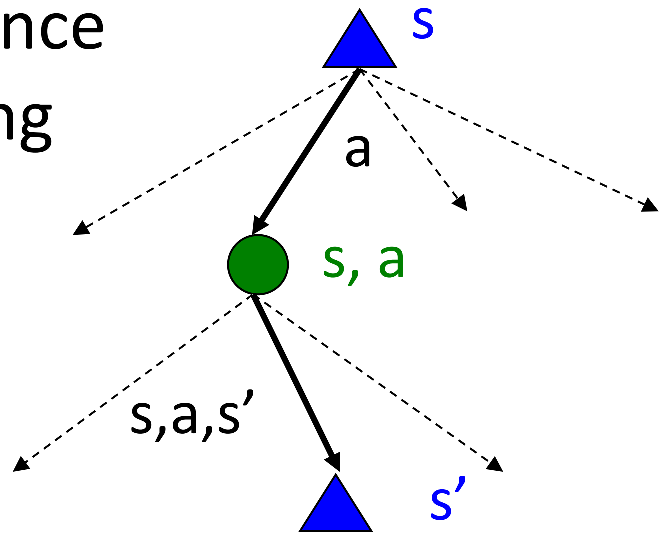
- Markov decision processes:
  - States S
  - Actions A
  - Transitions P(s'|s,a) (or T(s,a,s'))
  - Rewards R(s,a,s') (and discount γ)
  - Start state s_0



- Quantities:
  - Policy = map of states to actions
  - Utility = sum of discounted rewards
  - Values = expected future utility from a state (max node)
  - Q-Values = expected future utility from a q-state (chance node)

The Bellman Equations

- Definition of "optimal utility" via expectimax recurrence gives a simple one-step lookahead relationship among optimal utility values



$$V^*(s) = \max_a Q^*(s,a)$$

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

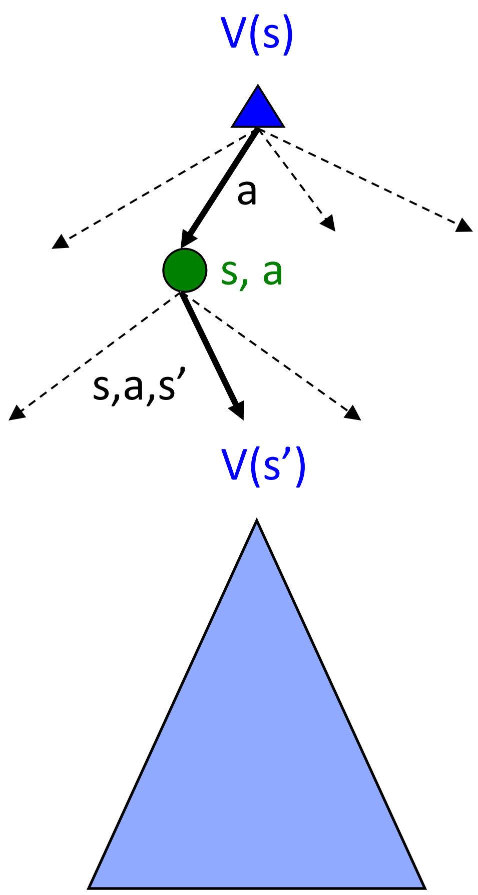
$$V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s')\right]$$

- These are the Bellman equations, and they characterize optimal values in a way we'll use over and over

Value Iteration

- Bellman equations characterize the optimal values:

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



- Value iteration computes them:

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

- Efficiency: $O(AS^2)$ per iteration

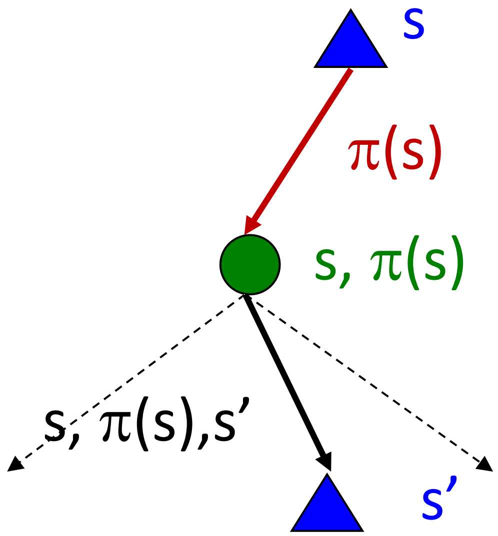
Policy Evaluation

- How do we calculate the V's for a fixed policy π?

- Idea 1: Turn recursive Bellman equations into updates (like value iteration)

$$V_0^\pi(s) = 0$$

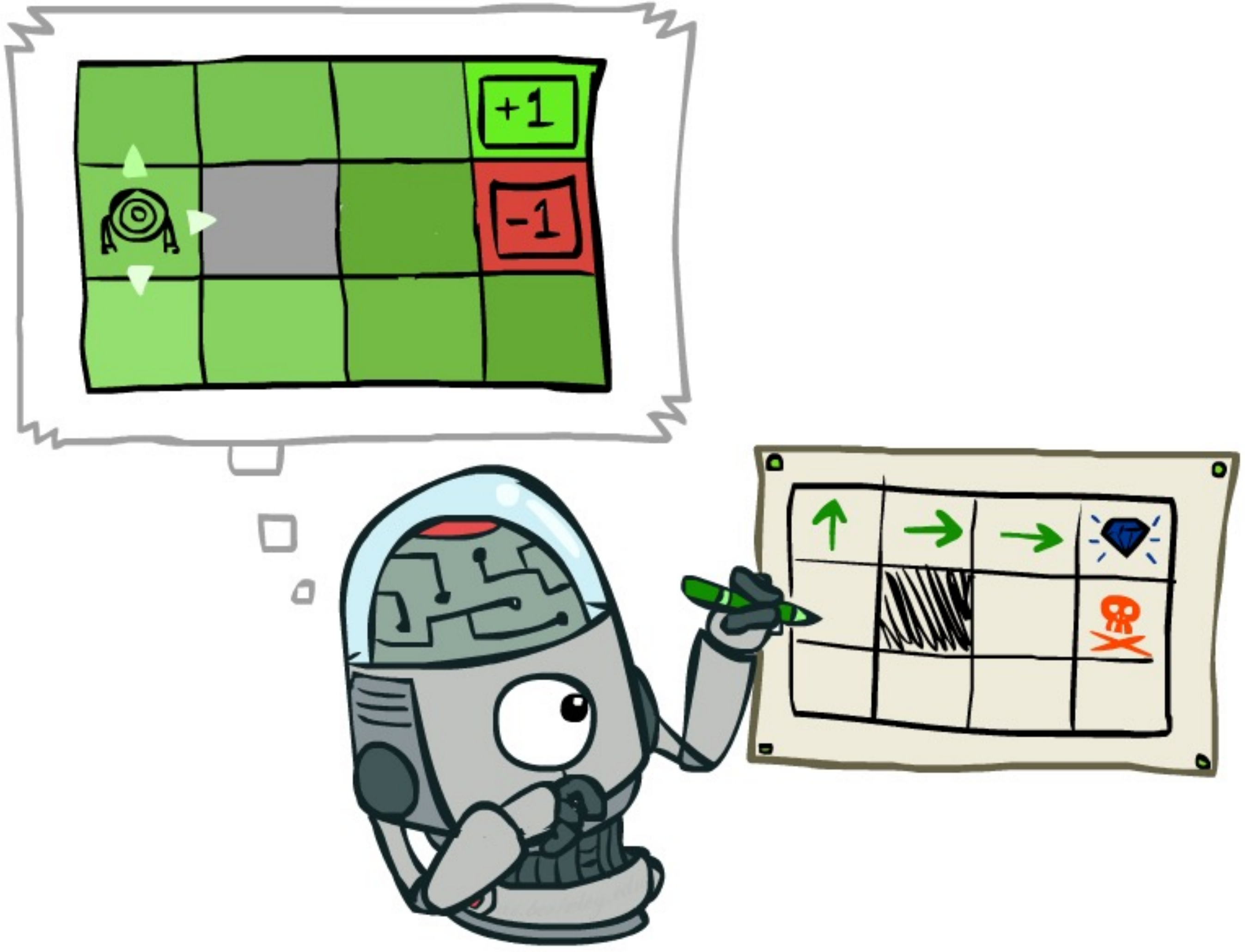
$$V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V_k^\pi(s')]$$



- Efficiency: $O(S^2)$ per iteration

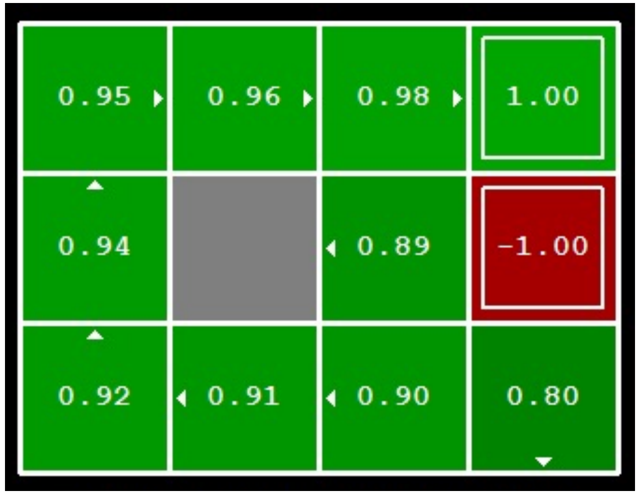
- Idea 2: Without the maxes, the Bellman equations are just a linear system
  - Solve with Matlab (or your favorite linear system solver)

Policy Extraction



Computing Actions from Values

- Let's imagine we have the optimal values V*(s)



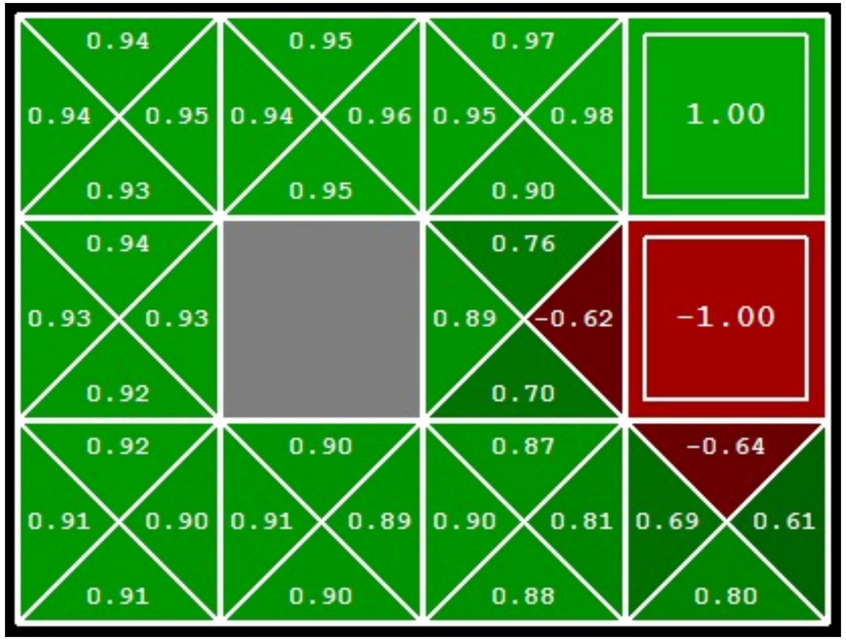
- How should we act?
  - It's not obvious!
- We need to do a mini-expectimax (one step)

$$\pi^*(s) = \arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

- This is called policy extraction, since it gets the policy implied by the values

Computing Actions from Q-Values

- Let's imagine we have the optimal q-values:

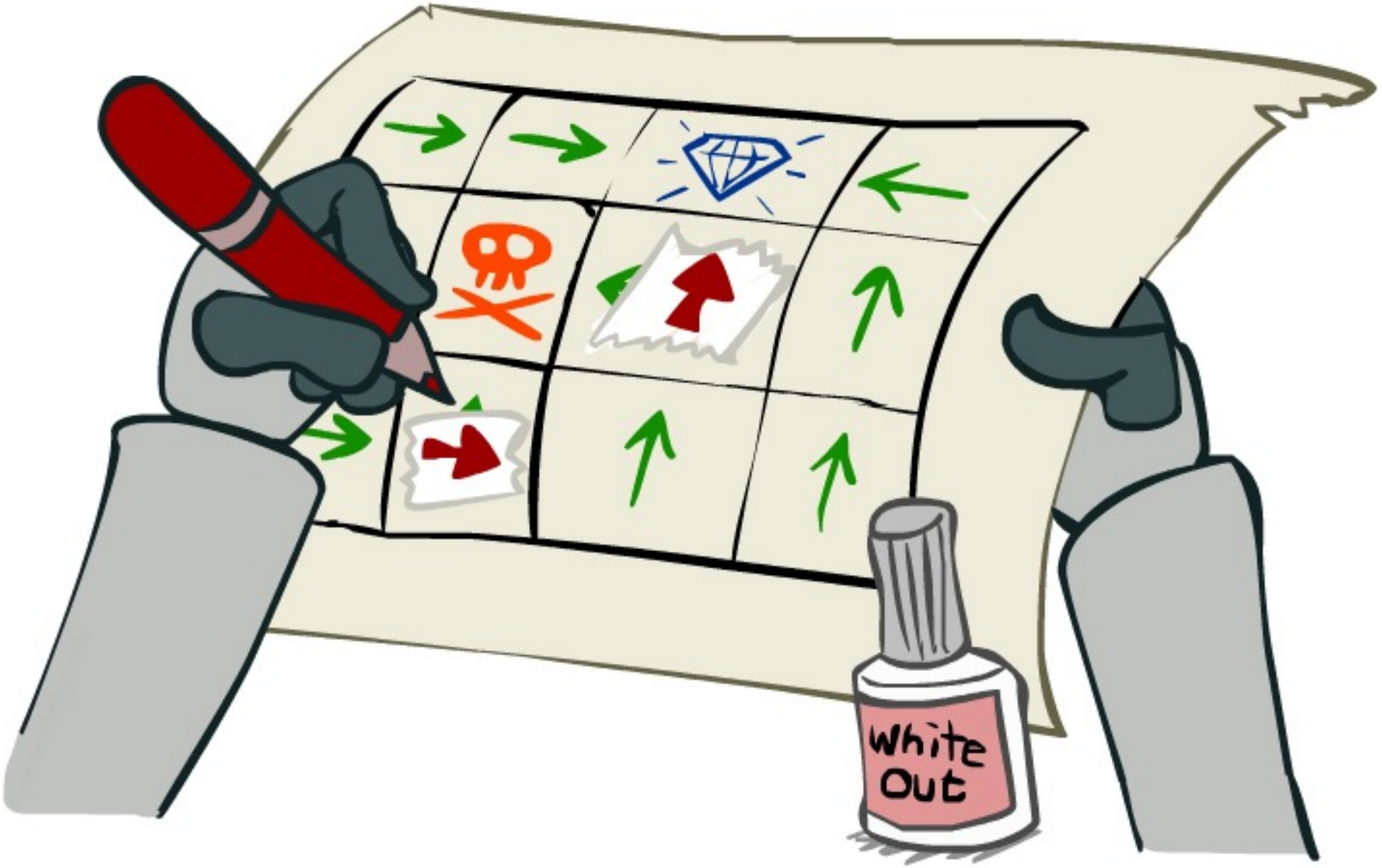


- How should we act?
  - Completely trivial to decide!

$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

- Important lesson: actions are easier to select from q-values than values!

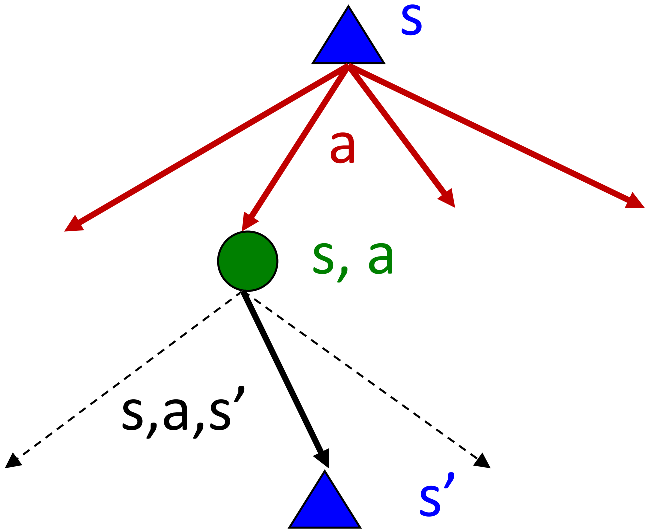
Policy Iteration



Problems with Value Iteration

- Value iteration repeats the Bellman updates:

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

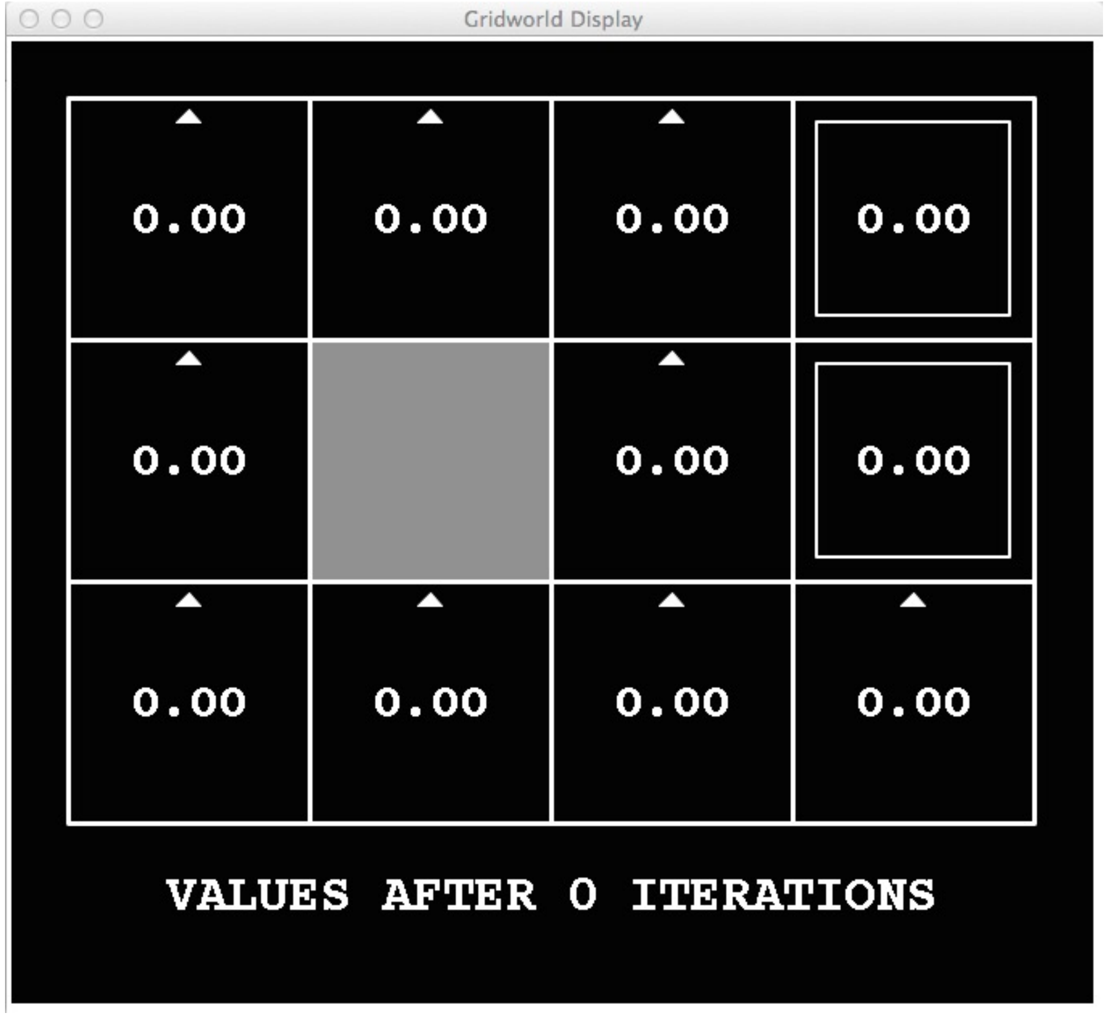


- Problem 1: It's slow – $O(S^2A)$ per iteration

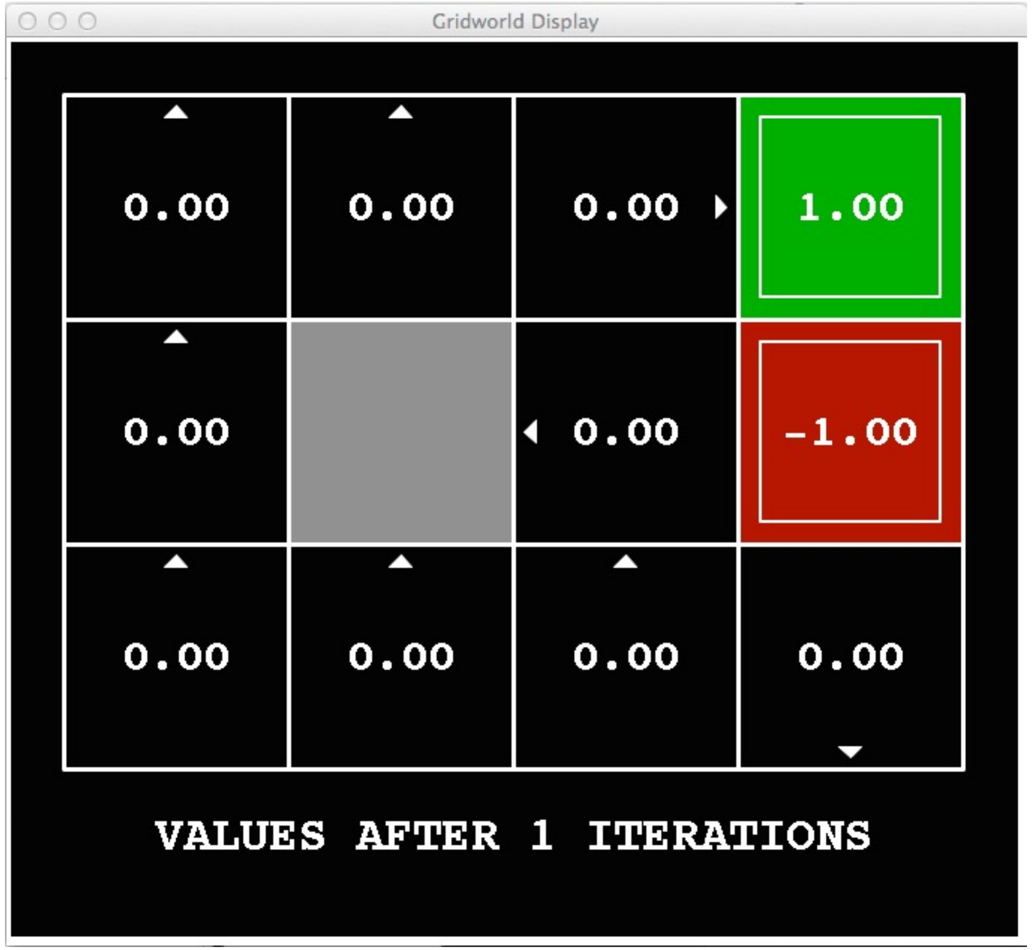
- Problem 2: The "max" at each state rarely changes

- Problem 3: The policy often converges long before the values

k=0


Gridworld Display
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
VALUES AFTER 0 ITERATIONS

k=1


Gridworld Display
0.00
0.00
0.00
1.00
0.00
0.00
-1.00
0.00
0.00
0.00
0.00
VALUES AFTER 1 ITERATIONS

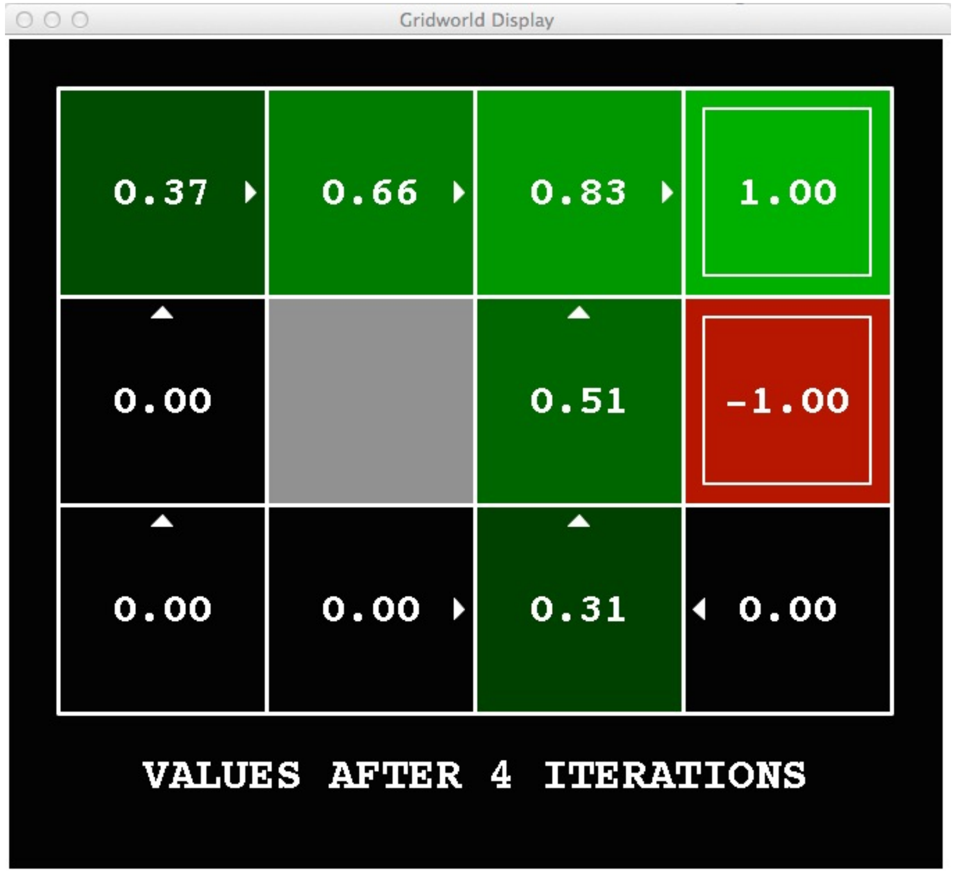
k=2


Gridworld Display
0.00
0.00
0.72
1.00
0.00
0.00
-1.00
0.00
0.00
0.00
0.00
VALUES AFTER 2 ITERATIONS

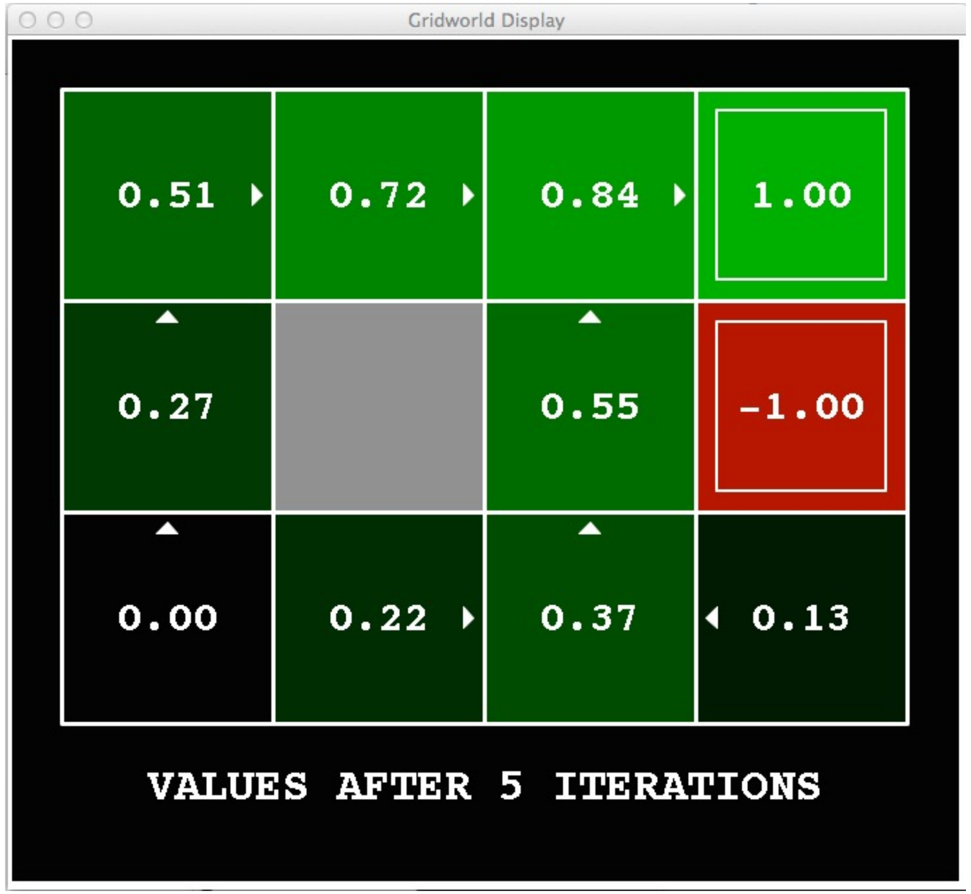
k=3


Gridworld Display
0.00
0.52
0.78
1.00
0.00
0.43
-1.00
0.00
0.00
0.00
0.00
VALUES AFTER 3 ITERATIONS

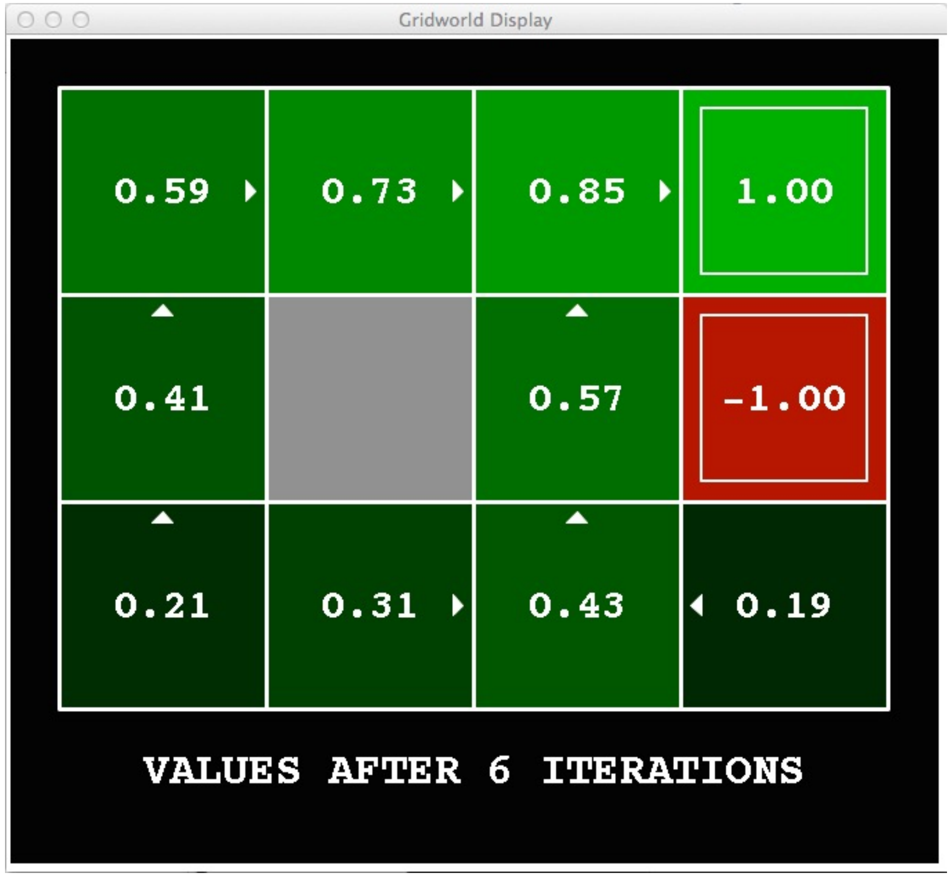
k=4


Gridworld Display
0.37
0.66
0.83
1.00
0.00
0.51
-1.00
0.00
0.00
0.31
0.00
VALUES AFTER 4 ITERATIONS

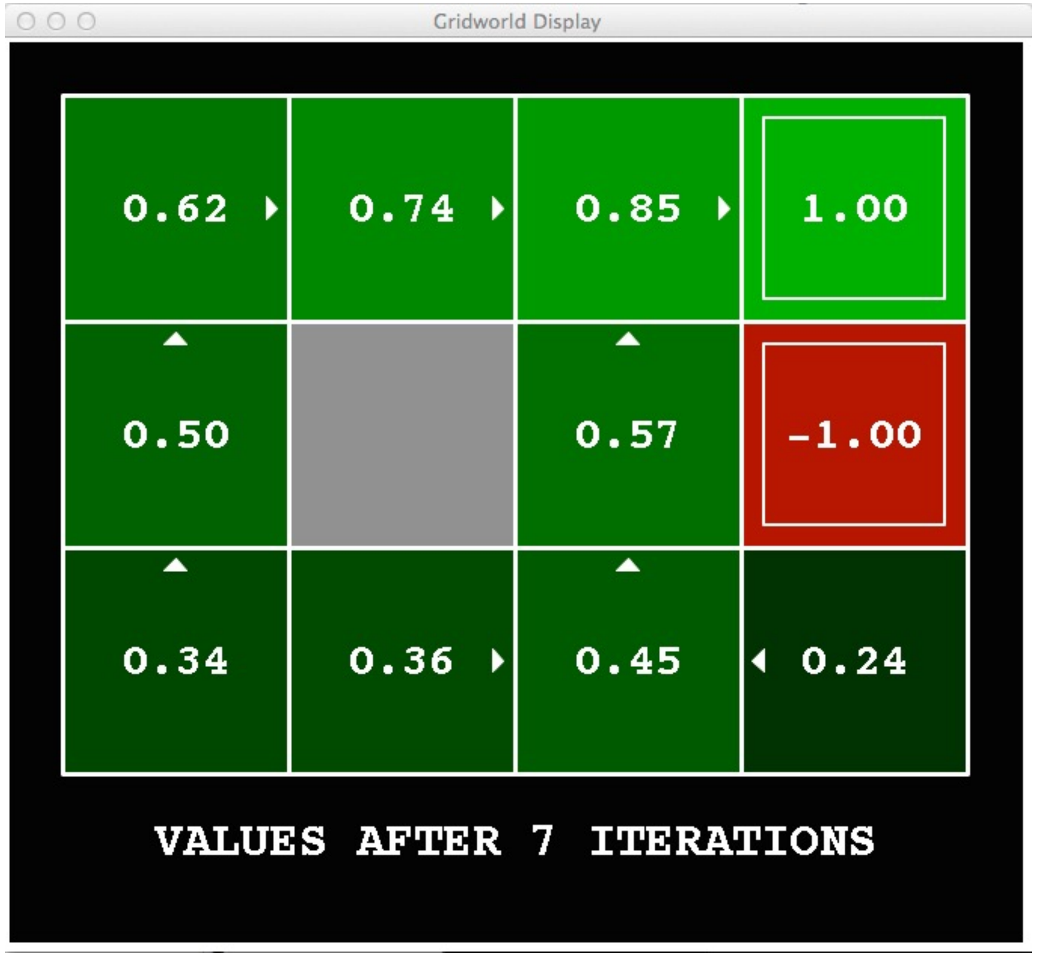
k=5


Gridworld Display
0.51
0.72
0.84
1.00
0.27
0.55
-1.00
0.00
0.22
0.37
0.13
VALUES AFTER 5 ITERATIONS

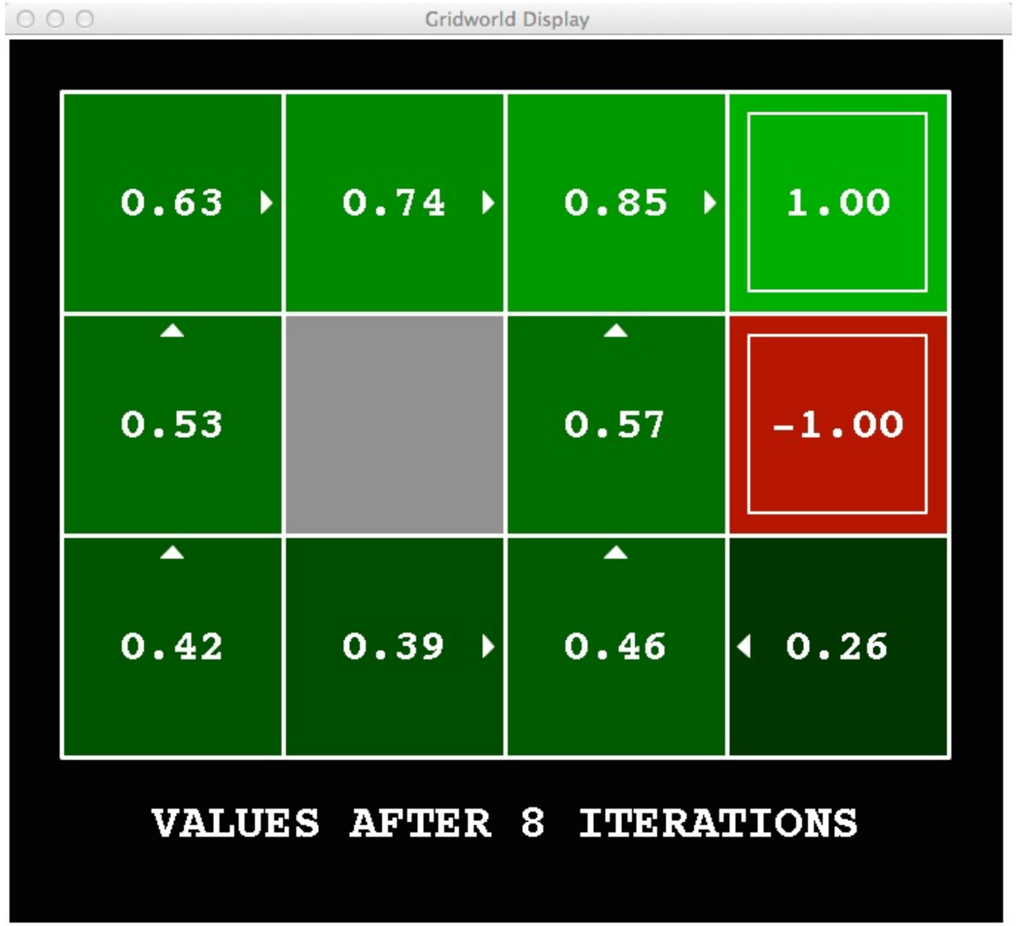
k=6


Gridworld Display
0.59
0.73
0.85
1.00
0.41
0.57
-1.00
0.21
0.31
0.43
0.19
VALUES AFTER 6 ITERATIONS

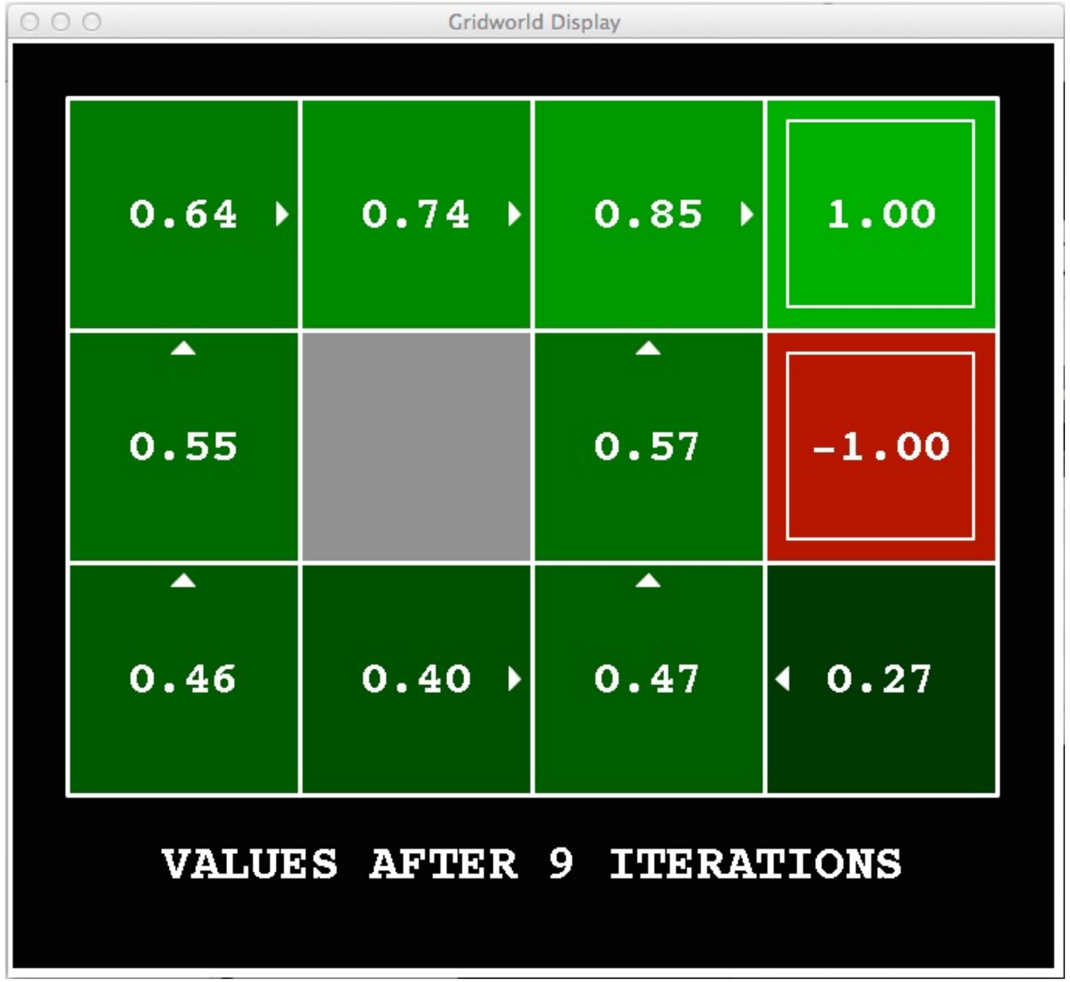
k=7


Gridworld Display
0.62
0.74
0.85
1.00
0.50
0.57
-1.00
0.34
0.36
0.45
0.24
VALUES AFTER 7 ITERATIONS

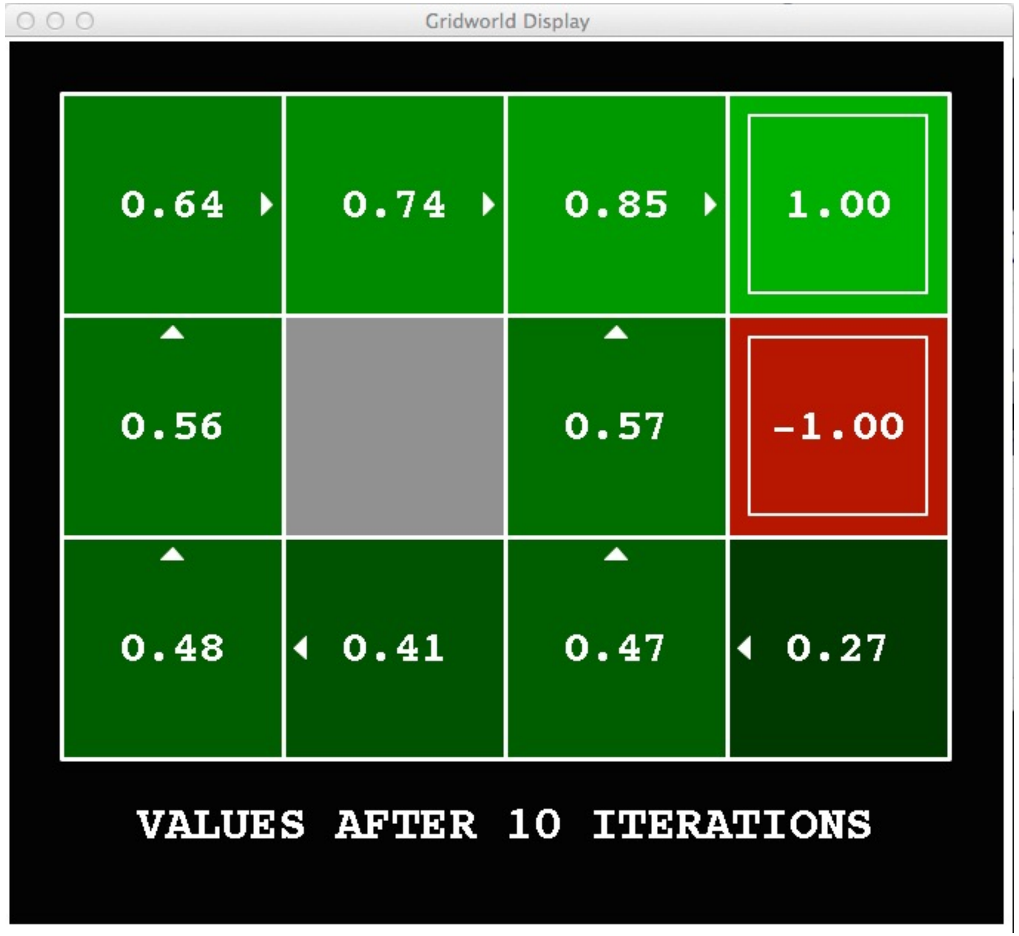
k=8


Gridworld Display
0.63
0.74
0.85
1.00
0.53
0.57
-1.00
0.42
0.39
0.46
0.26
VALUES AFTER 8 ITERATIONS

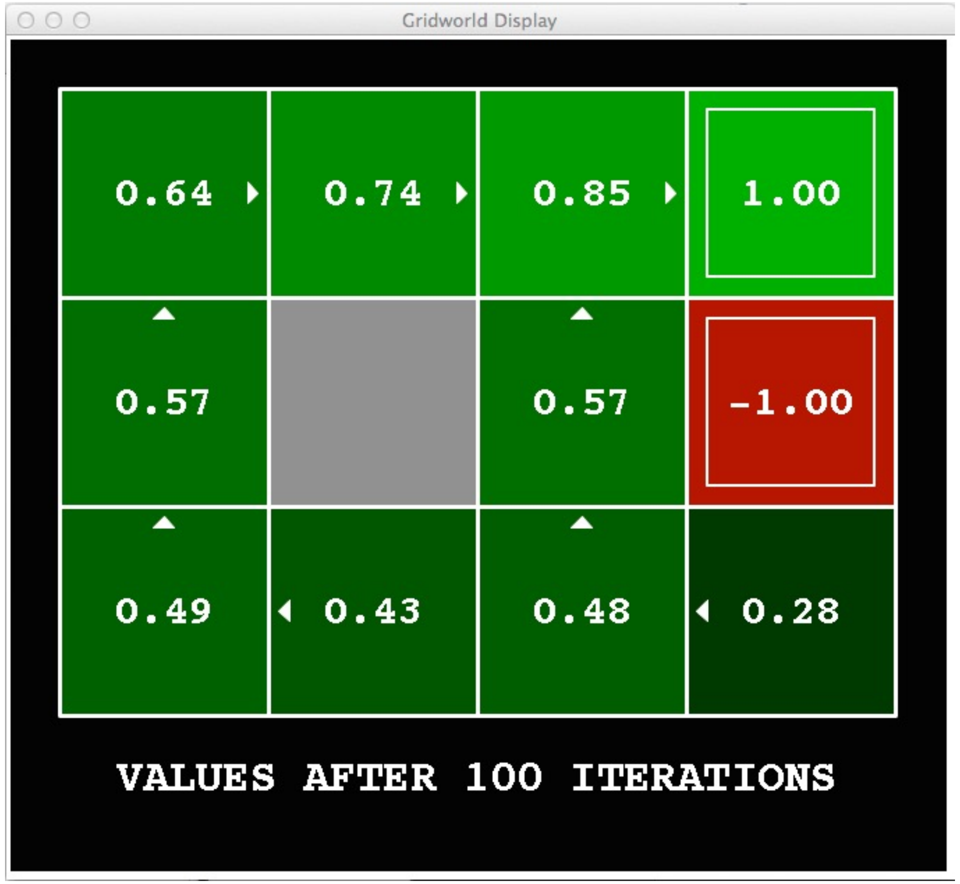
k=9


Gridworld Display
0.64
0.74
0.85
1.00
0.55
0.57
-1.00
0.46
0.40
0.47
0.27
VALUES AFTER 9 ITERATIONS

k=10


Gridworld Display
0.64
0.74
0.85
1.00
0.56
0.57
-1.00
0.48
0.41
0.47
0.27
VALUES AFTER 10 ITERATIONS

k=100


Gridworld Display
0.64
0.74
0.85
1.00
0.57
0.57
-1.00
0.49
0.43
0.48
0.28
VALUES AFTER 100 ITERATIONS

Policy Iteration

- Alternative approach for optimal values:
  - Step 1: Policy evaluation: calculate utilities for some fixed policy (not optimal utilities!) until convergence
  - Step 2: Policy improvement: update policy using one-step look-ahead with resulting converged (but not optimal!) utilities as future values
  - Repeat steps until policy converges

- This is policy iteration
  - It's still optimal!
  - Can converge (much) faster under some conditions

Policy Iteration

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

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

- Improvement: For fixed values, get a better policy using policy extraction
  - One-step look-ahead:

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

Comparison

- Both value iteration and policy iteration compute the same thing (all optimal values)

- In value iteration:
  - Every iteration updates both the values and (implicitly) the policy
  - We don’t track the policy, but taking the max over actions implicitly recomputes it

- In policy iteration:
  - We do several passes that update utilities with fixed policy (each pass is fast because we consider only one action, not all of them)
  - After the policy is evaluated, a new policy is chosen (slow like a value iteration pass)
  - The new policy will be better (or we’re done)

- Both are dynamic programs for solving MDPs

Summary: MDP Algorithms

- So you want to….
  - Compute optimal values: use value iteration or policy iteration
  - Compute values for a particular policy: use policy evaluation
  - Turn your values into a policy: use policy extraction (one-step lookahead)

- These all look the same!
  - They basically are – they are all variations of Bellman updates
  - They all use one-step lookahead expectimax fragments
  - They differ only in whether we plug in a fixed policy or max over actions