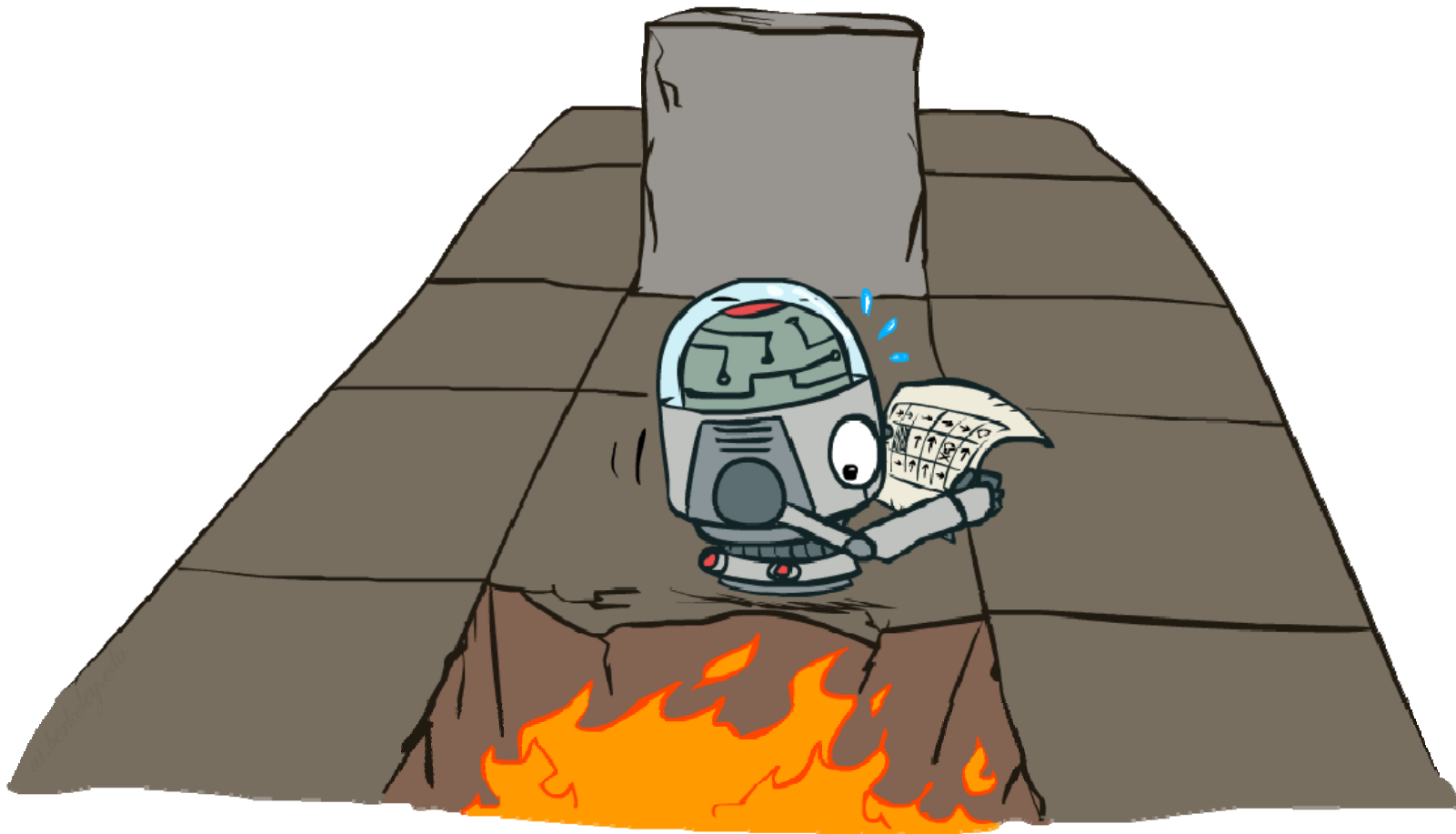


Artificial Intelligence

Markov Decision Processes II



Instructors: Fatemeh Mansoori--- University of Isfahan

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

Example: Grid World

A maze-like problem

- The agent lives in a grid
- Walls block the agent's path

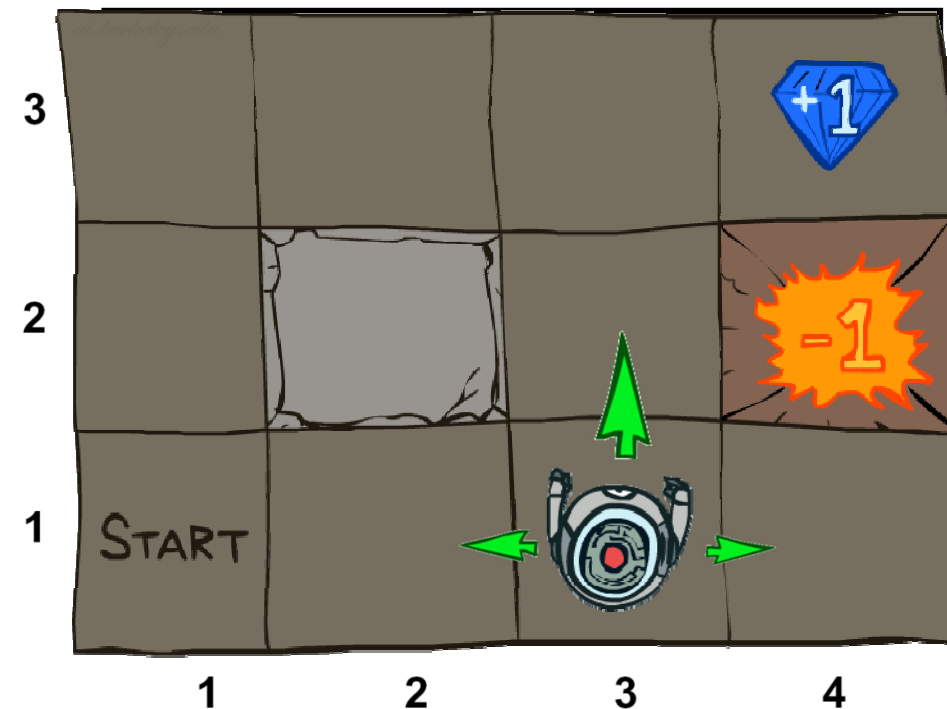
Noisy movement: actions do not always go as planned

- 80% of the time, the action North takes the agent North
- 10% of the time, North takes the agent West; 10% East
- If there is a wall in the direction the agent would have been taken, the agent stays put

The agent receives rewards each time step

- Small "living" reward each step (can be negative)
- Big rewards come at the end (good or bad)

Goal: maximize sum of (discounted) rewards



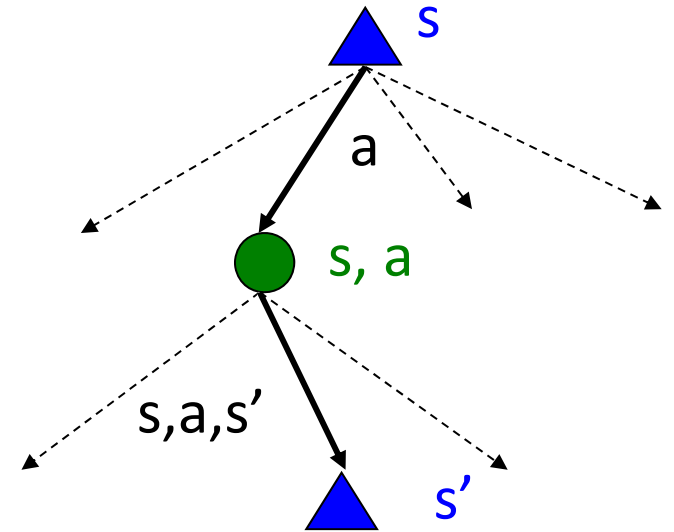
Recap: MDPs

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)



Optimal Quantities

The value (utility) of a state s :

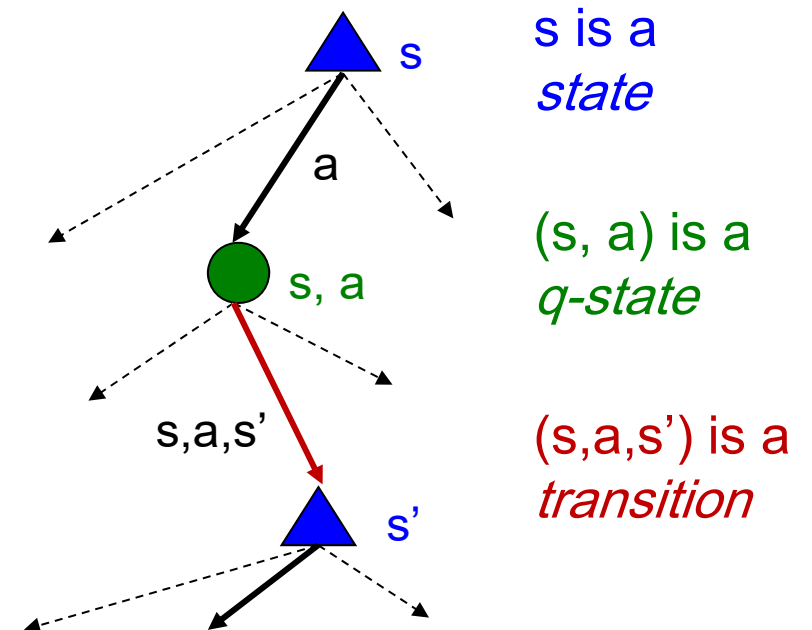
$V^*(s)$ = expected utility starting in s and acting optimally

The value (utility) of a q-state (s,a) :

$Q^*(s,a)$ = expected utility starting out having taken action a from state s and (thereafter) acting optimally

The optimal policy:

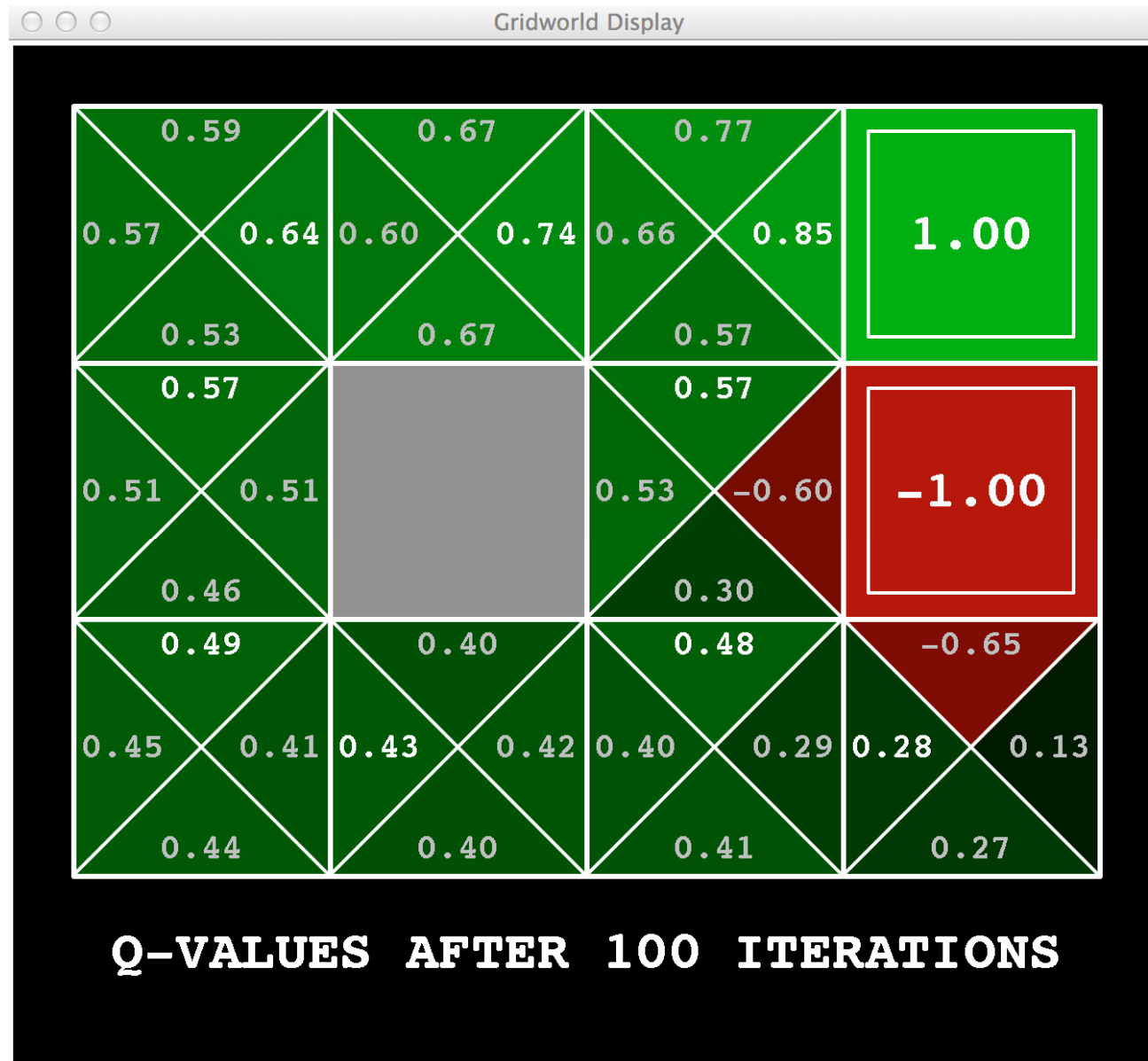
$\pi^*(s)$ = optimal action from state s



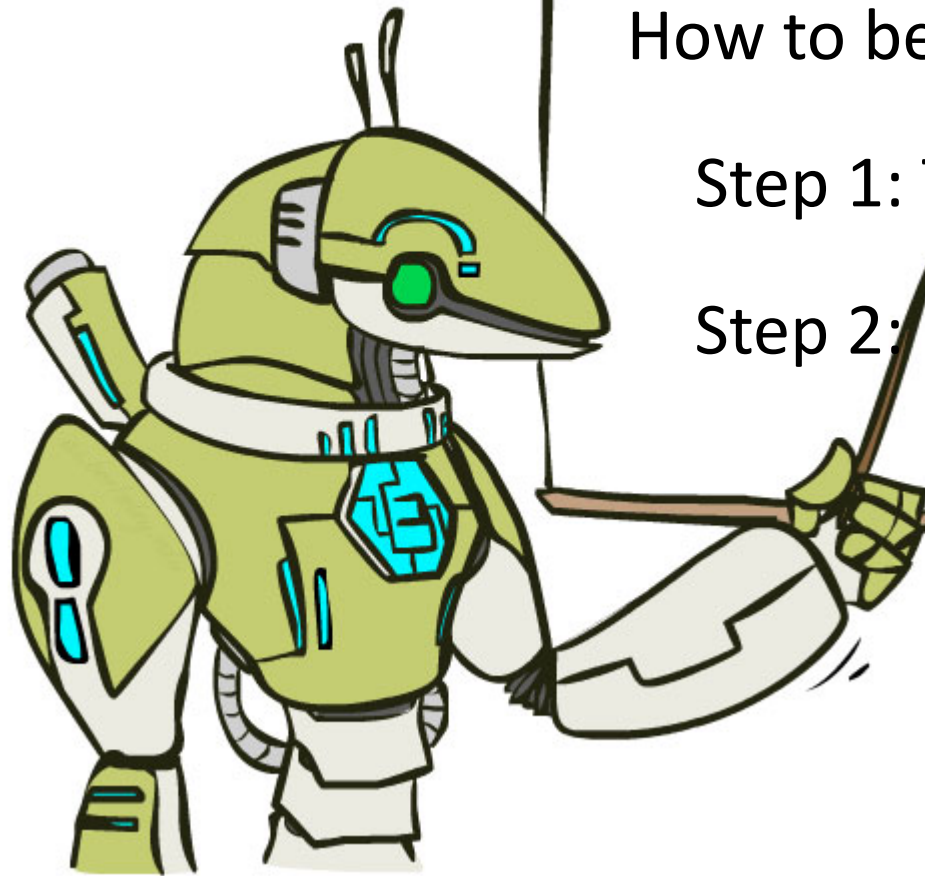
Gridworld Values V^*



Gridworld: Q^*



The Bellman Equations



How to be optimal:

Step 1: Take correct first action

Step 2: Keep being optimal

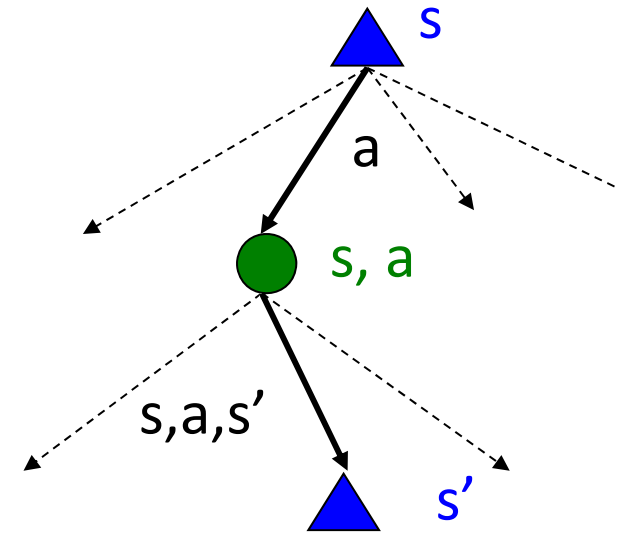
The Bellman Equations

Definition of “optimal utility” via expectimax recurrence gives a simple one-step lookahead relationship amongst optimal utility values

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

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

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



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:

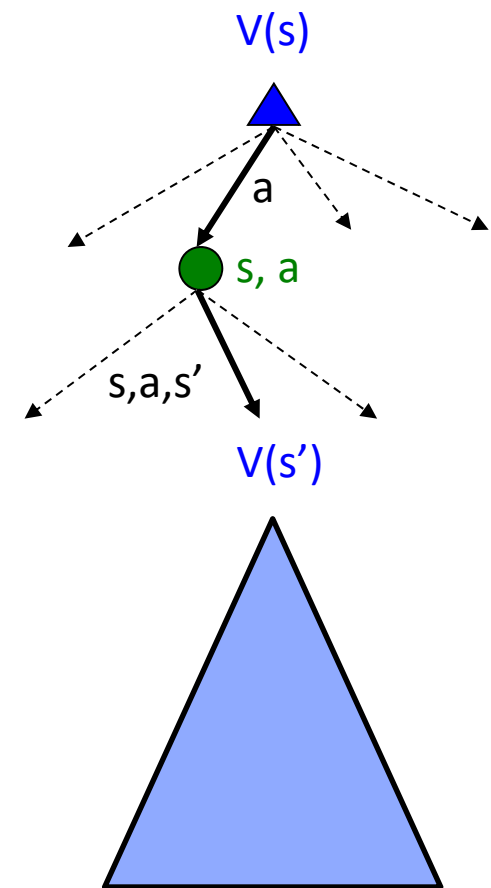
$$V^*(s) = \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')]$$

Value iteration **computes** them:

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

Value iteration is just a fixed point solution method

- ... though the V_k vectors are also interpretable as time-limited values



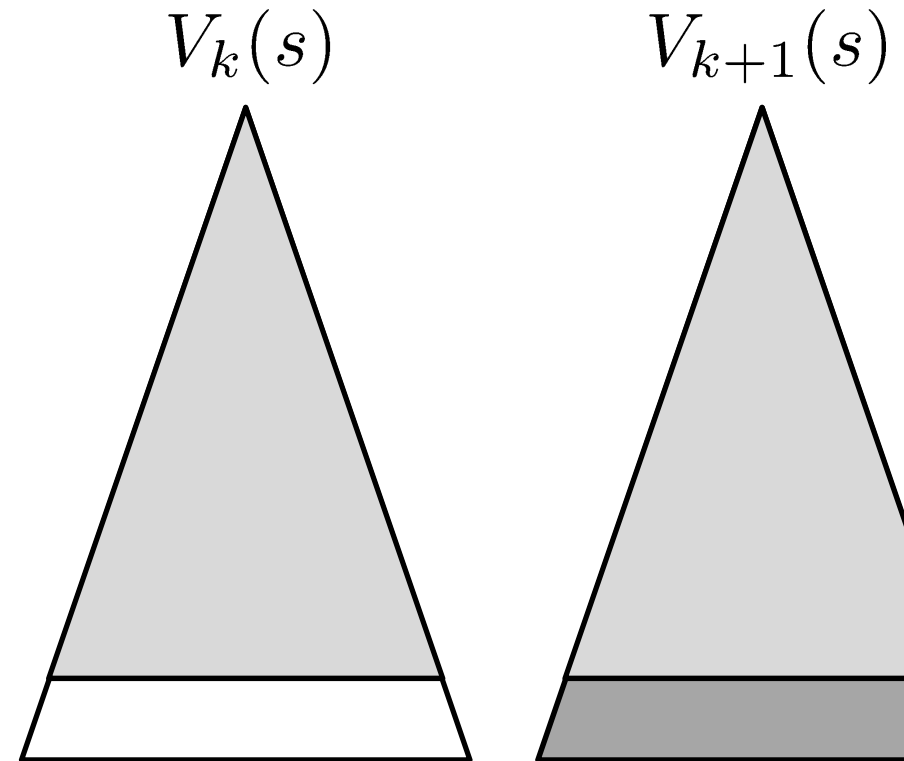
Convergence*

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

Case 1: If the tree has maximum depth M , then V_M holds the actual untruncated values

Case 2: If the discount is less than 1

- Sketch: For any state V_k and V_{k+1} can be viewed as depth $k+1$ expectimax results in nearly identical search trees
- The difference is that on the bottom layer, V_{k+1} has actual rewards while V_k has zeros
- That last layer is at best all R_{MAX}
- It is at worst R_{MIN}
- But everything is discounted by γ^k that far out
- So V_k and V_{k+1} are at most $\gamma^k \max |R|$ different
- So as k increases, the values converge



Question

Suppose that, there is a game that you repeatedly draw a card (with replacement) that is equally likely to be a 2, 3, or 4. You can either Draw or Stop if the total score of the cards you have drawn is less than 6. If your total score is 6 or higher, the game ends, and you receive a utility of 0. When you Stop, your utility is equal to your total score (up to 5), and the game ends. When you Draw, you receive no utility. There is no discount ($\gamma = 1$).

Formulate this problem as an MDP with the following states:

- 0, 2, 3, 4, 5 and a Done state, for when the game ends.

What is the transition function and the reward function for this MDP

Find the optimal values of each state for first 4 iterations

What is the optimal policy

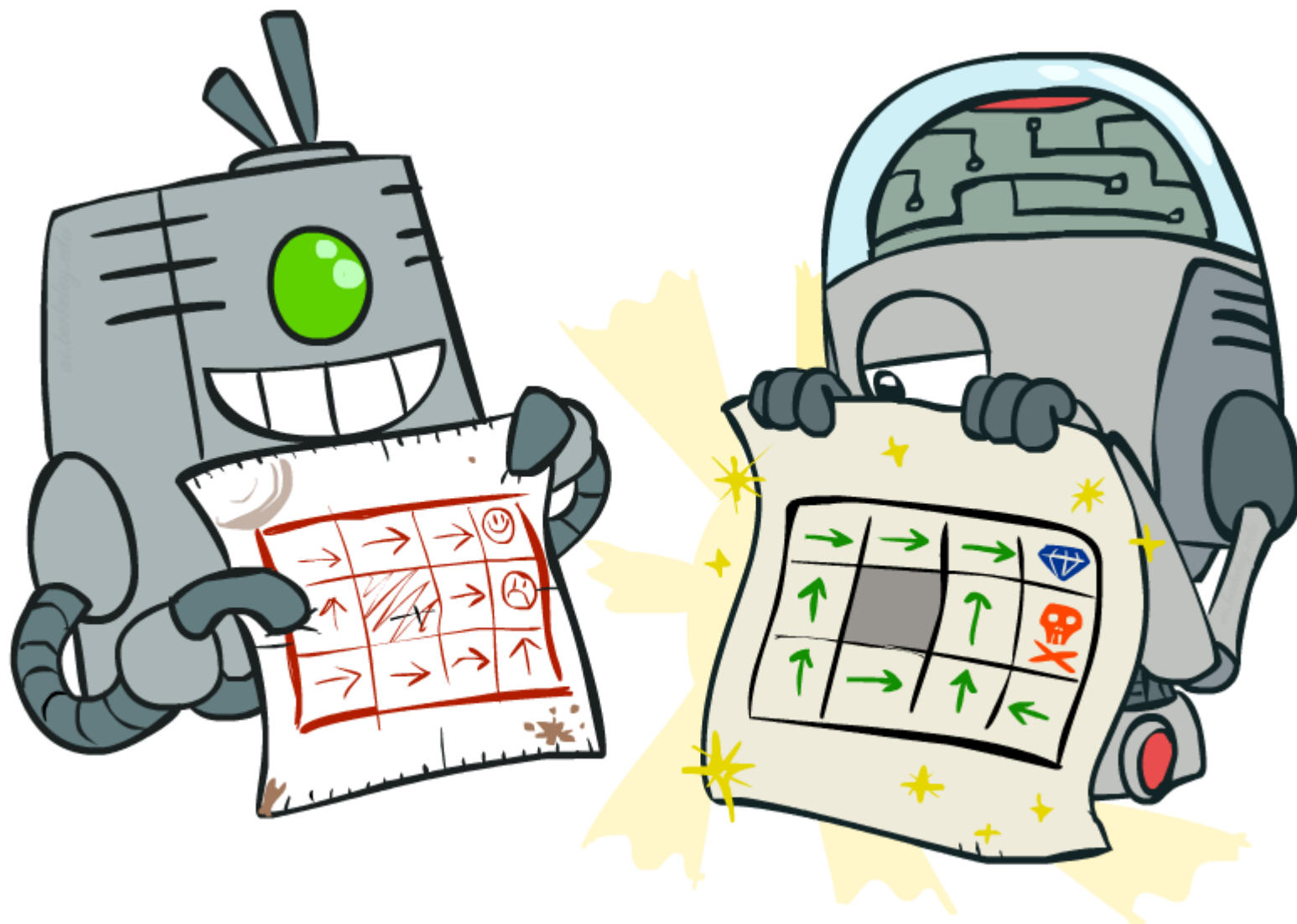
solution

| States | 0 | 2 | 3 | 4 | 5 |
|--------|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 2 | 3 | 4 | 5 |
| 2 | 3 | 3 | 3 | 4 | 5 |
| 3 | | | | | |
| 4 | | | | | |

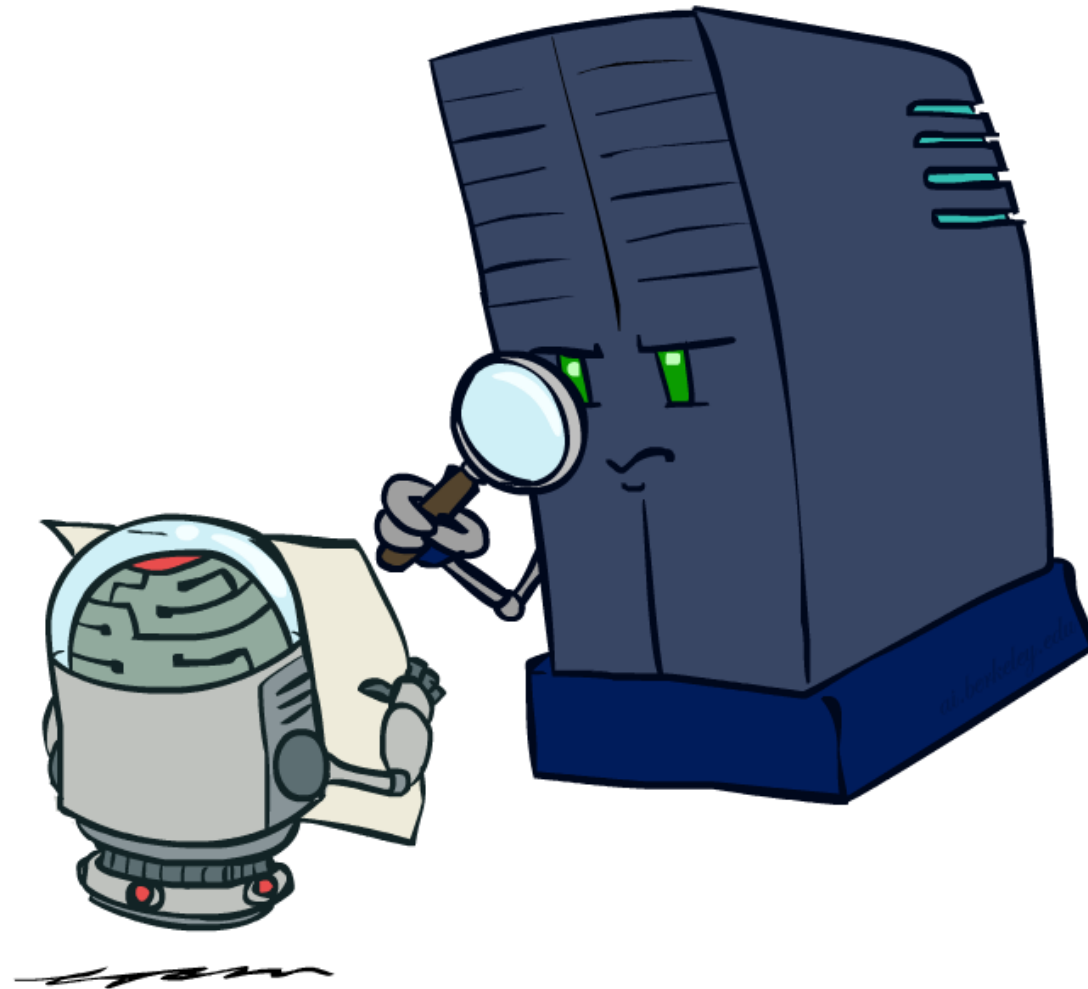
What is the optimal policy for the MDP?

| States | 0 | 2 | 3 | 4 | 5 |
|---------|------|------|------|------|------|
| π^* | Draw | Draw | Stop | Stop | Stop |

Policy Methods

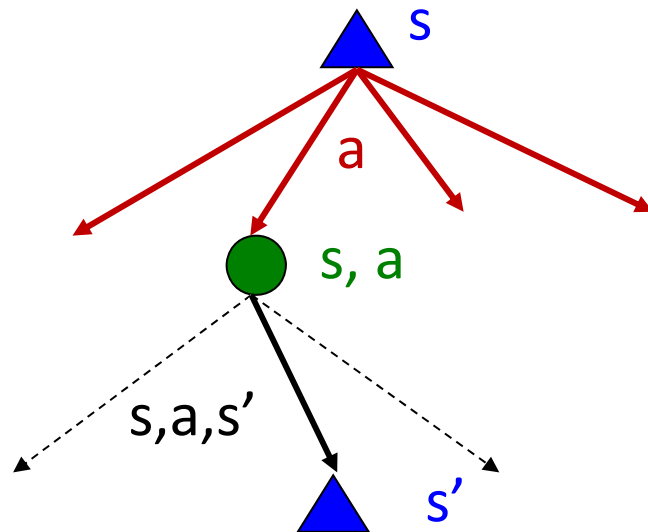


Policy Evaluation

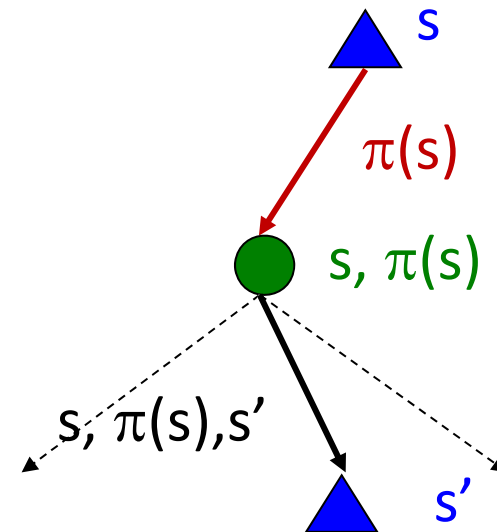


Fixed Policies

Do the optimal action



Do what π says to do



Expectimax trees max over all actions to compute the optimal values

If we fixed some policy $\pi(s)$, then the tree would be simpler – only one action per state

- ... though the tree's value would depend on which policy we fixed

Utilities for a Fixed Policy

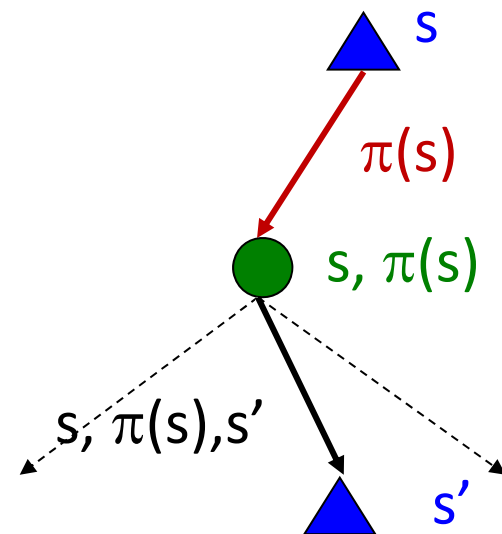
Another basic operation: compute the utility of a state s under a fixed (generally non-optimal) policy

Define the utility of a state s , under a fixed policy π :

$V^\pi(s)$ = expected total discounted rewards starting in s and following π

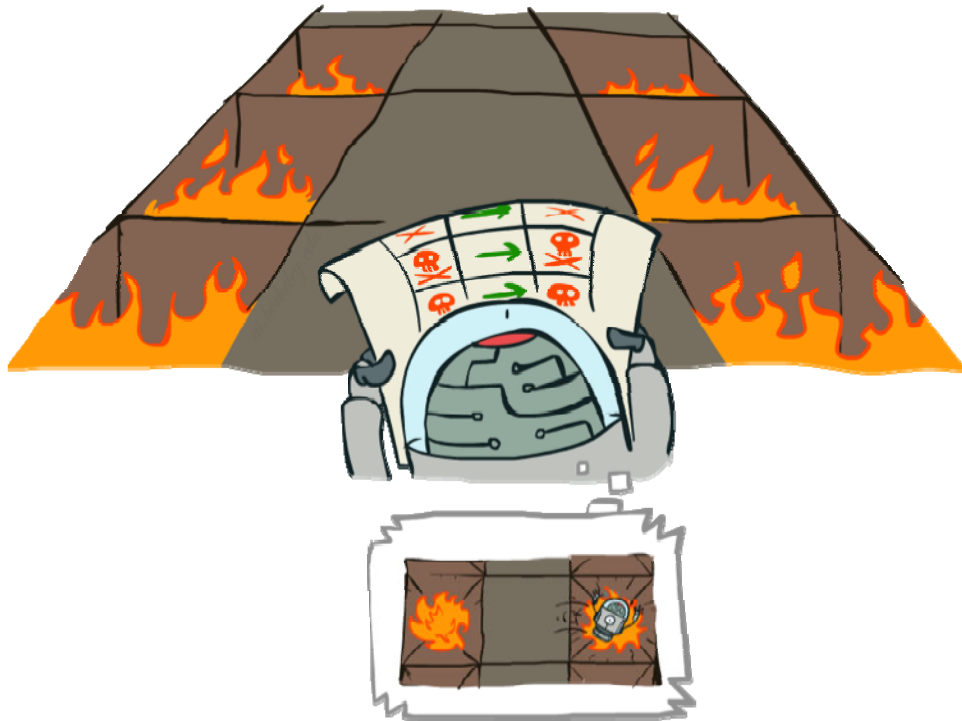
Recursive relation (one-step look-ahead / Bellman equation):

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

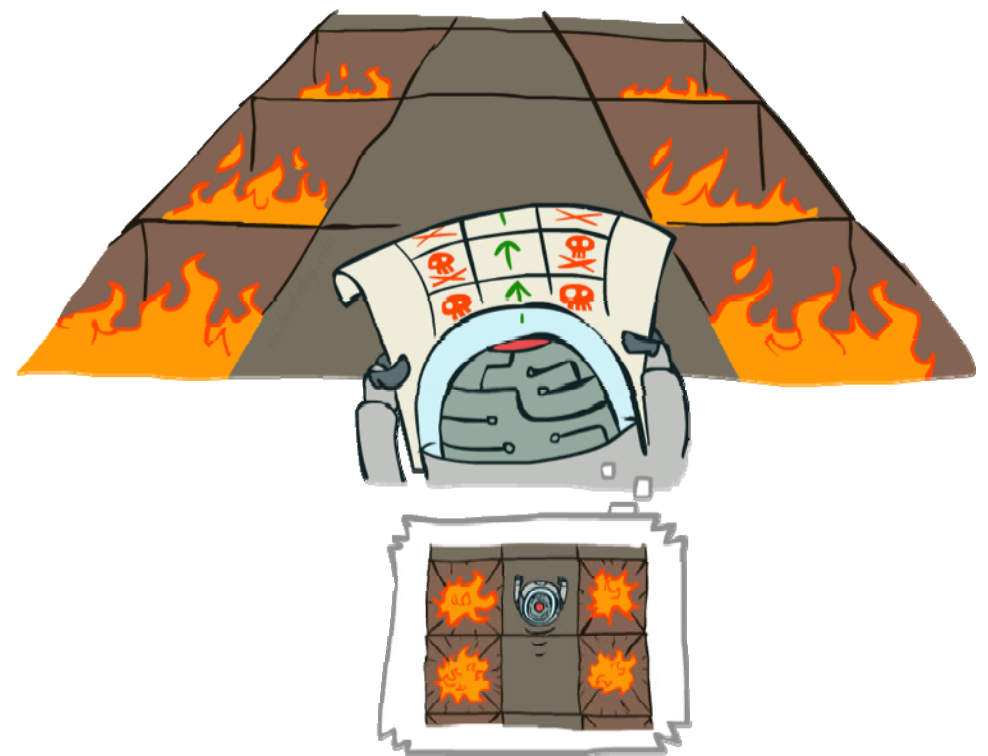


Example: Policy Evaluation

Always Go Right



Always Go Forward

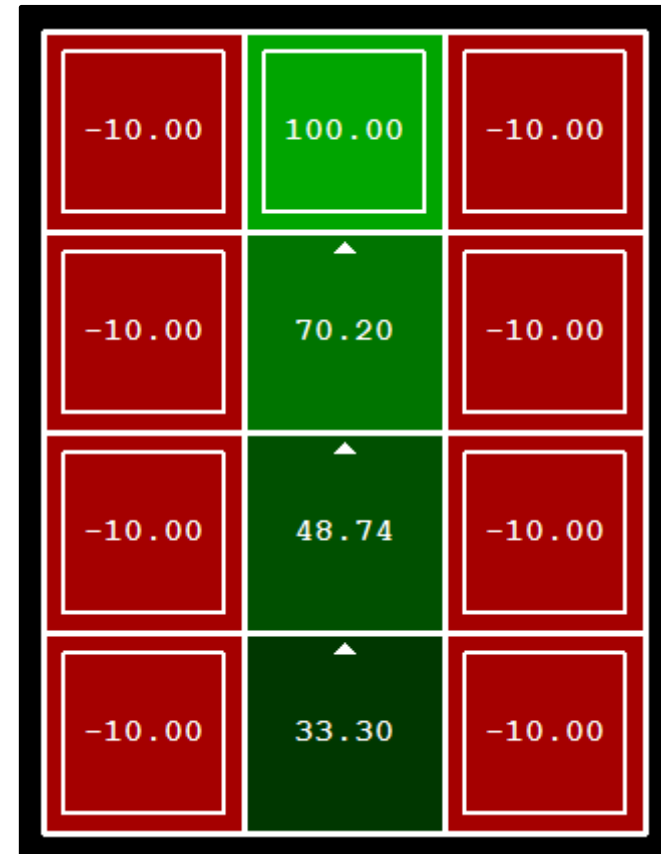


Example: Policy Evaluation

Always Go Right



Always Go Forward



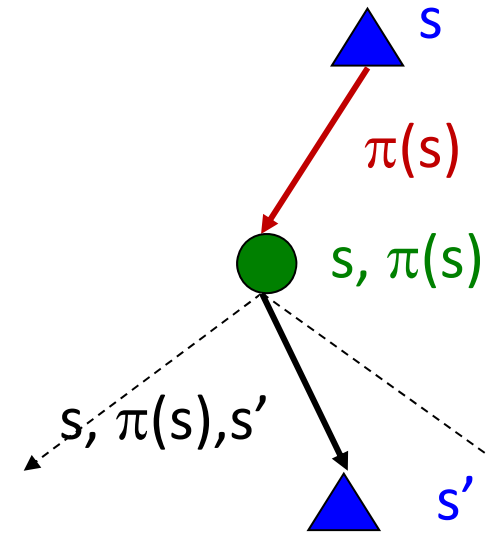
Policy Evaluation

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

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

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

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

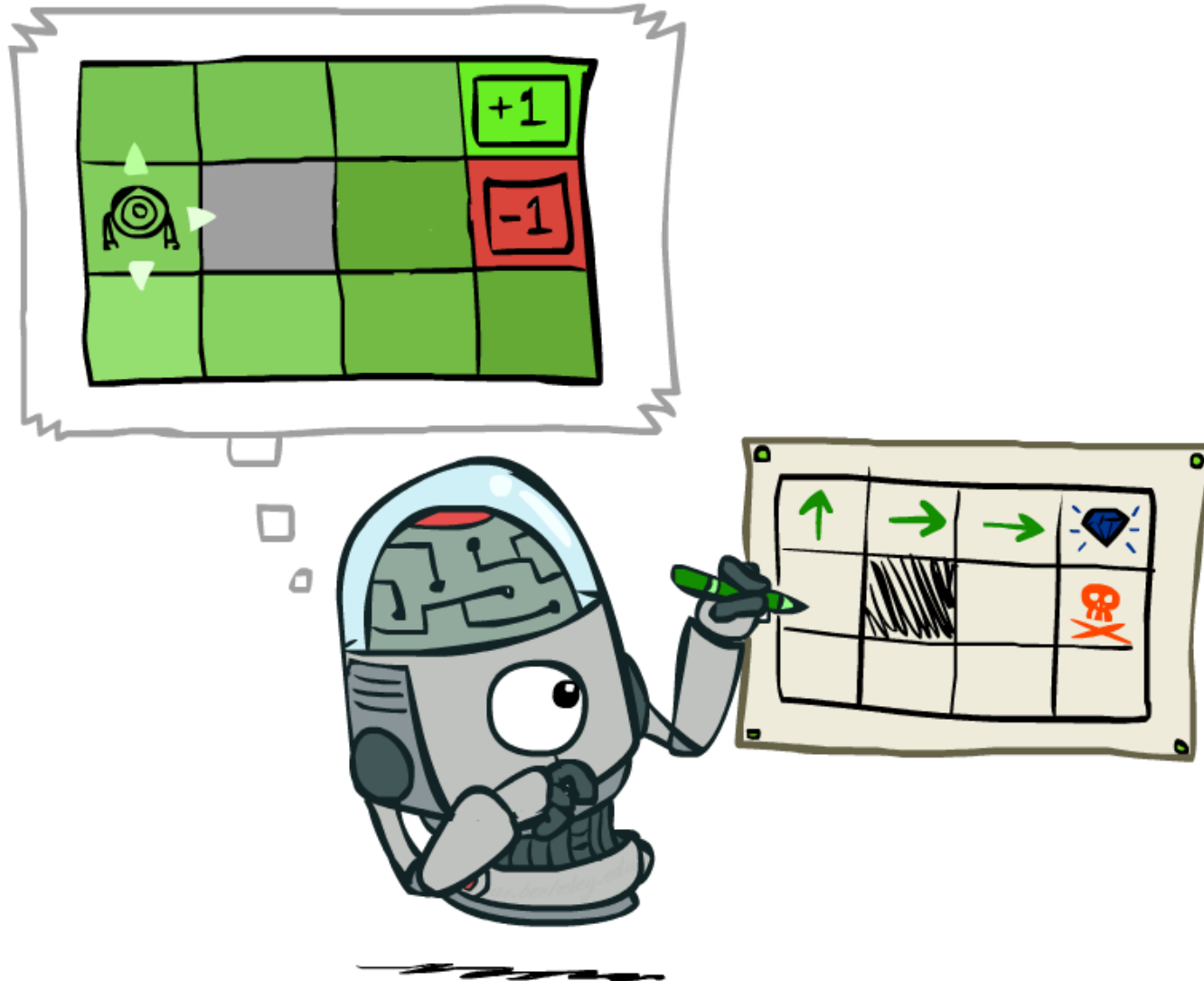


Efficiency: $O(S^2)$ per iteration

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



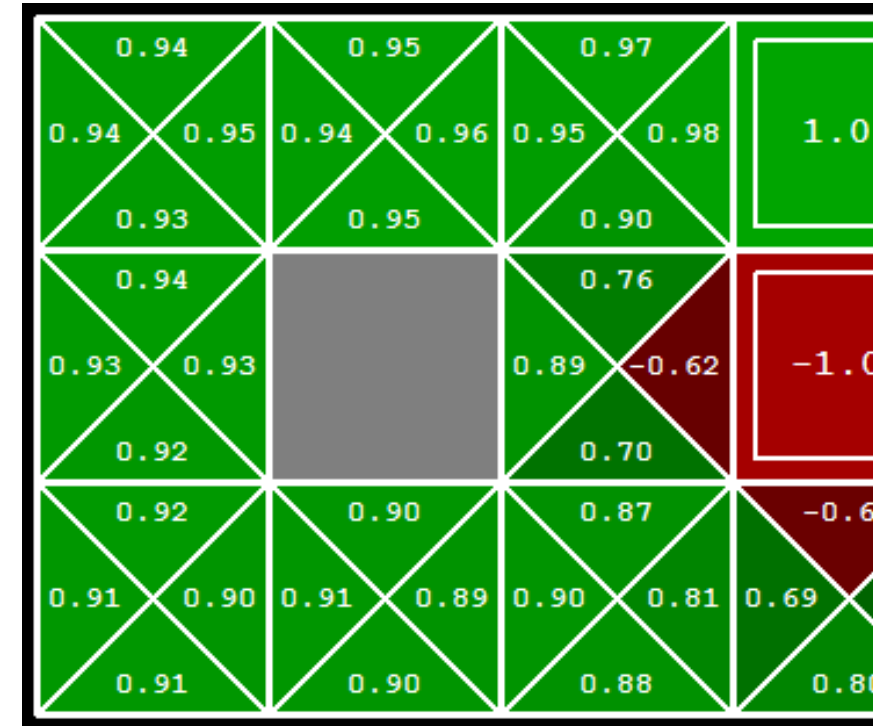
Computing Actions from Q-Values

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

How should we act?

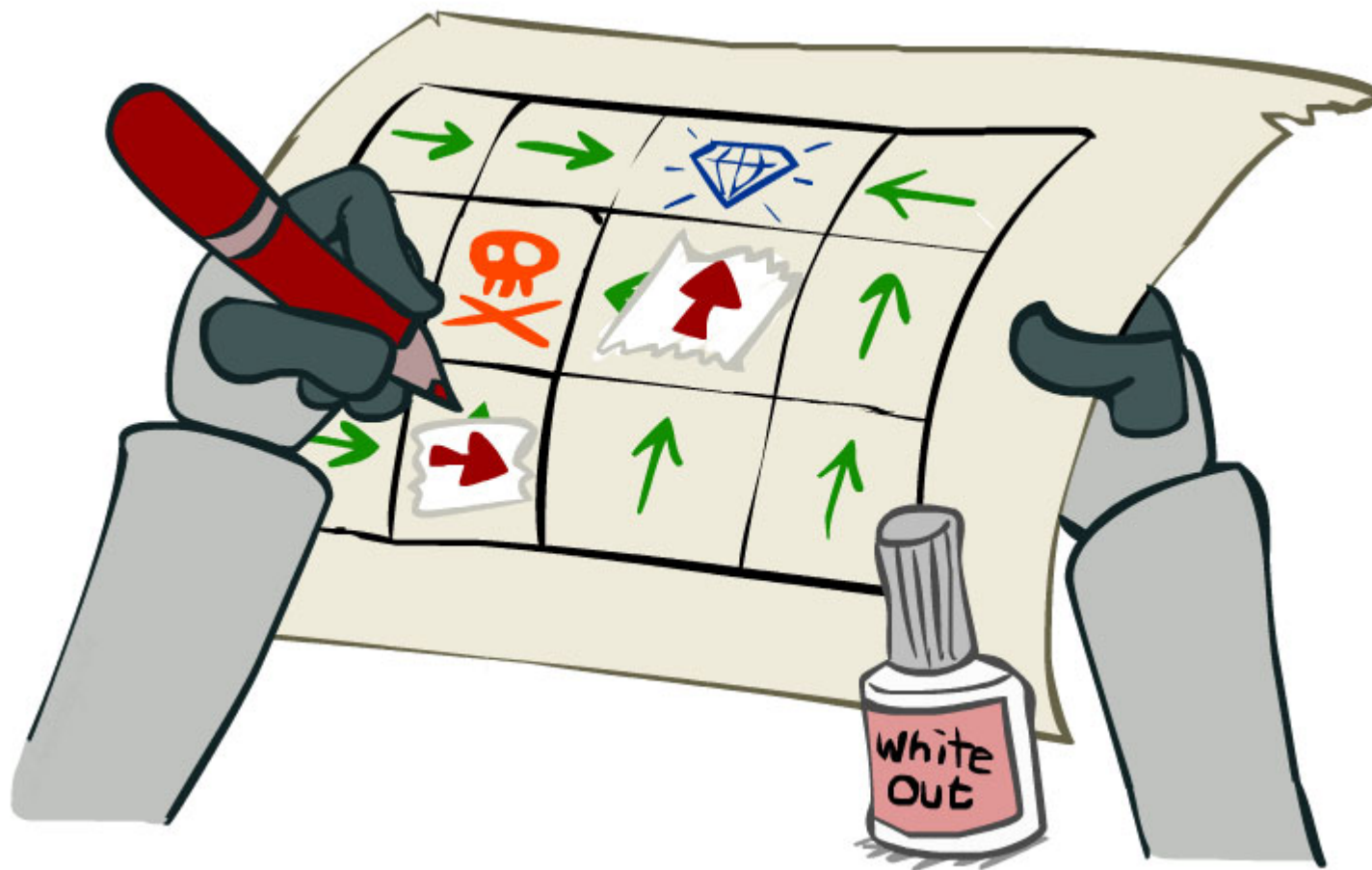
- Completely trivial to decide!

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



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

Policy Iteration



Problems with Value Iteration

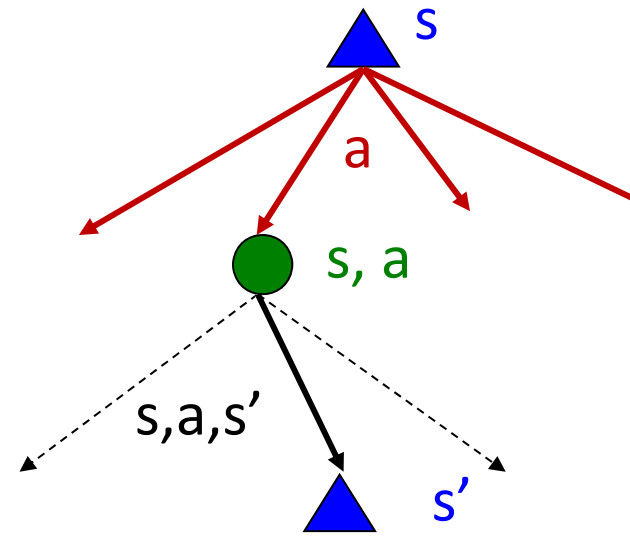
Value iteration repeats the Bellman updates:

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

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

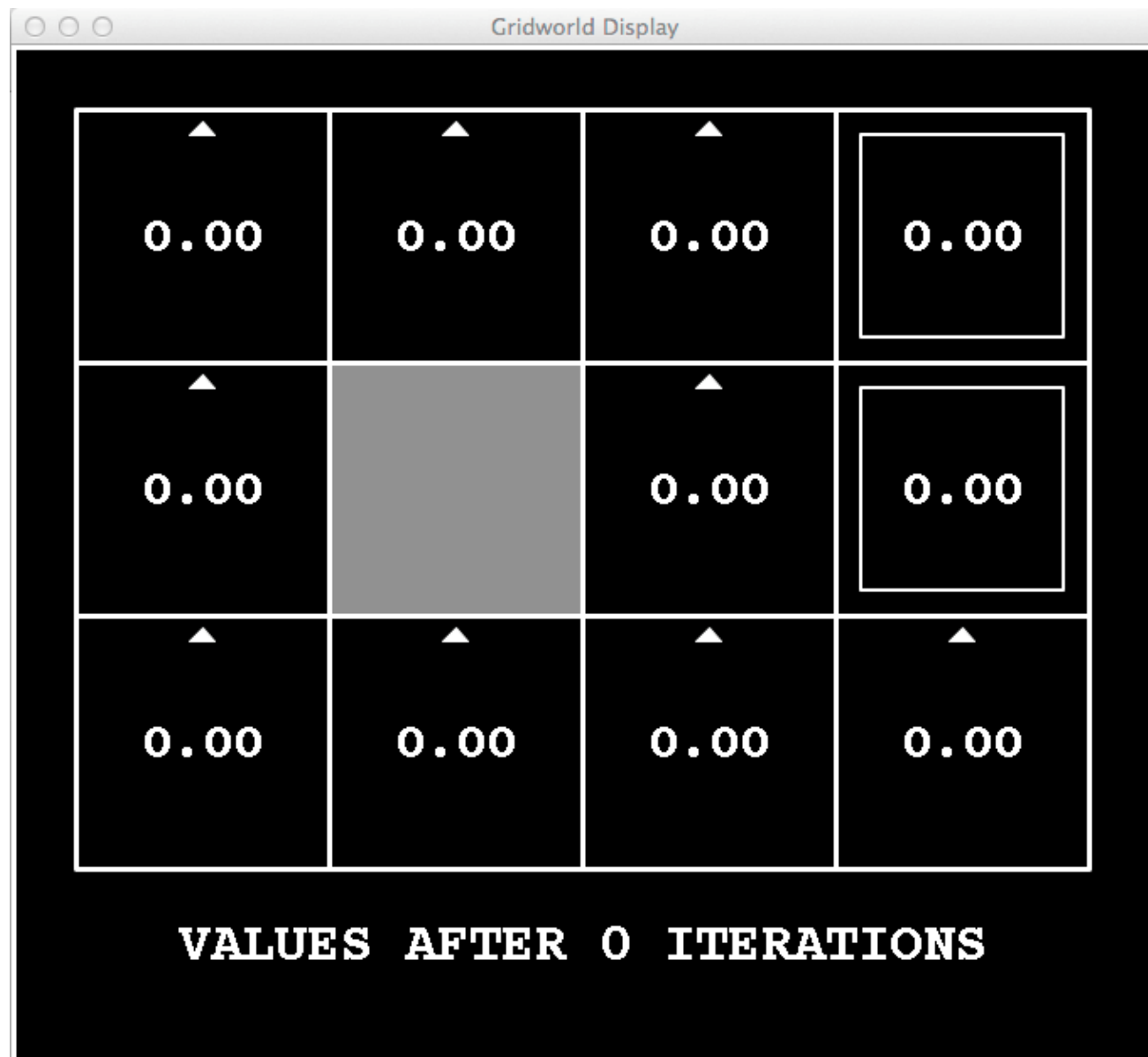
Problem 2: The “max” at each state rarely changes

Problem 3: The policy often converges long before the values



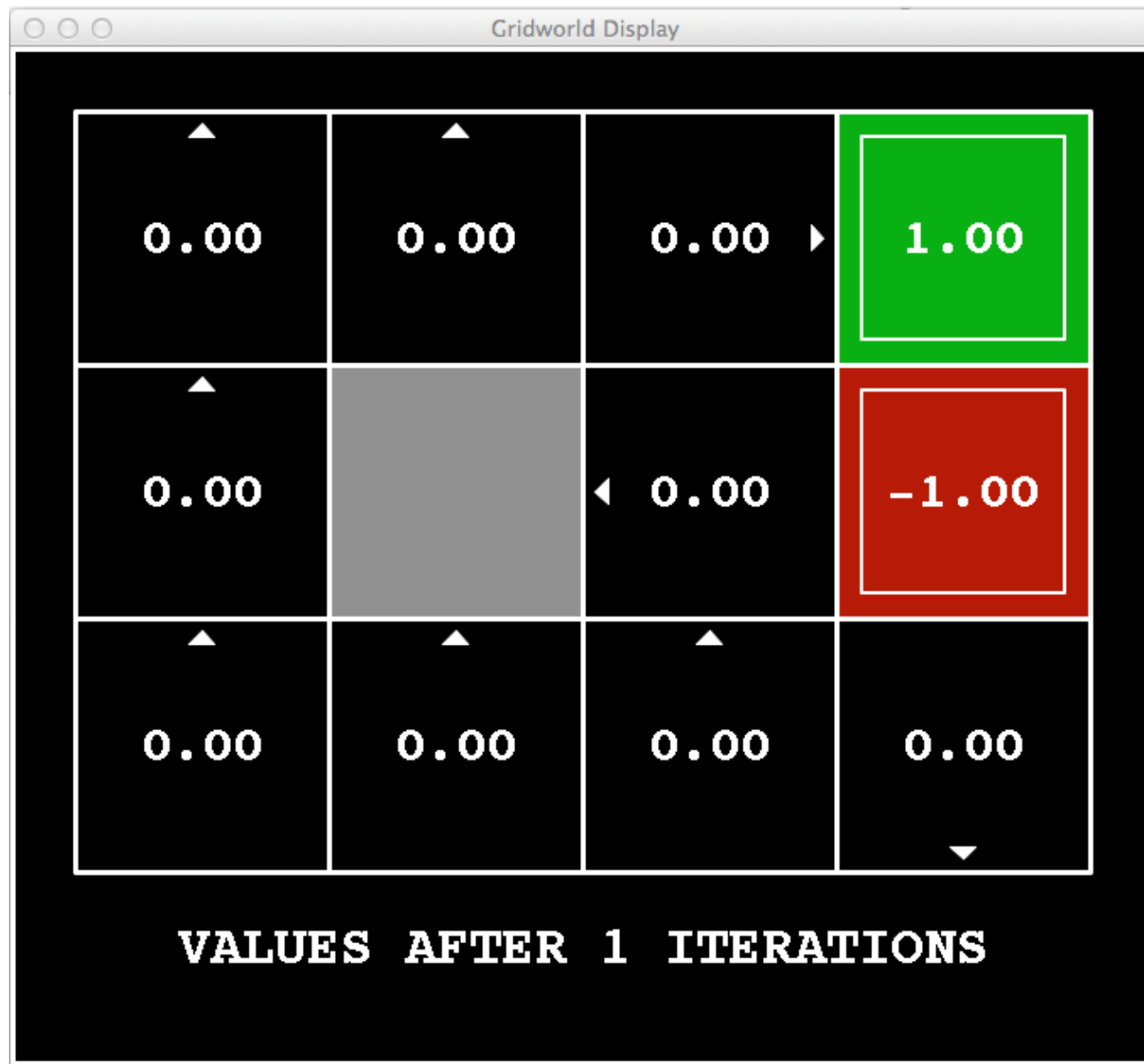
[Demo: value iteration (

$k=0$



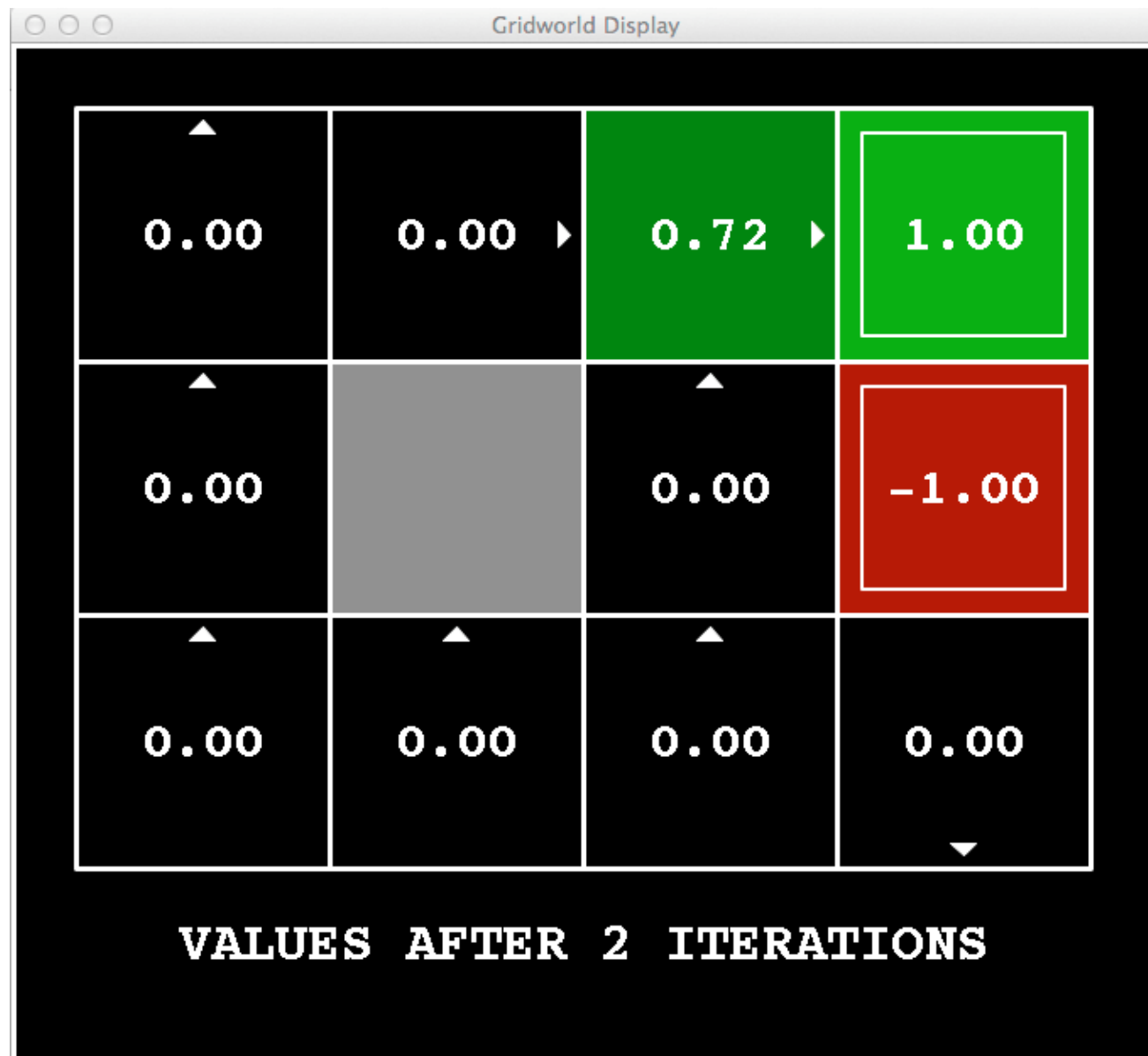
Noise = 0.2
Discount = 0.9
Living reward =

$k=1$



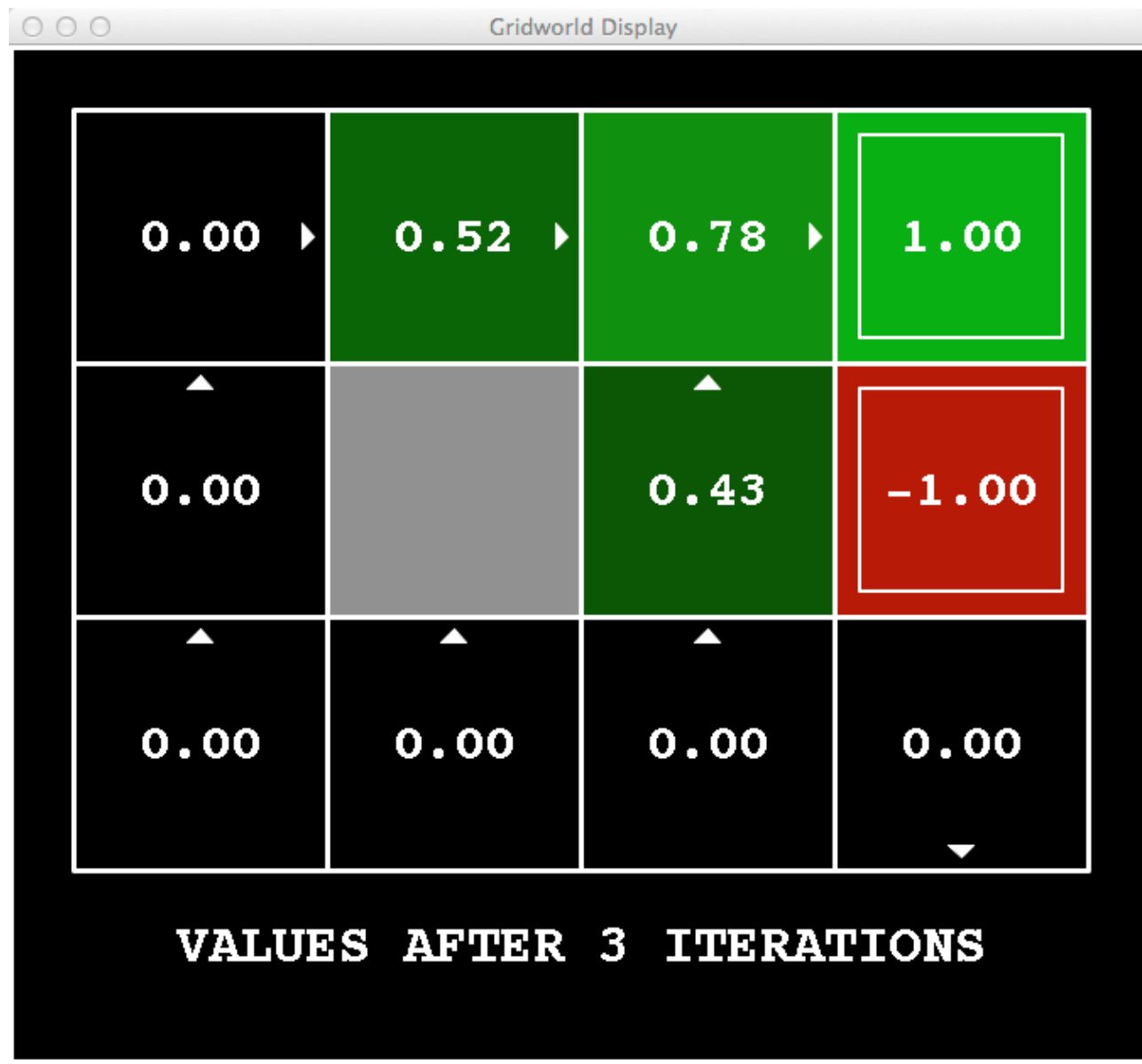
Noise = 0.2
Discount = 0.9
Living reward =

$k=2$



Noise = 0.2
Discount = 0.9
Living reward =

$k=3$



Noise = 0.2
Discount = 0.9
Living reward =

$k=4$



Noise = 0.2
Discount = 0.9
Living reward =

k=5



Noise = 0.2
Discount = 0.9
Living reward =

k=6



Noise = 0.2
Discount = 0.9
Living reward =

$k=7$



Noise = 0.2
Discount = 0.9
Living reward =

k=8



Noise = 0.2
Discount = 0.9
Living reward =

k=9



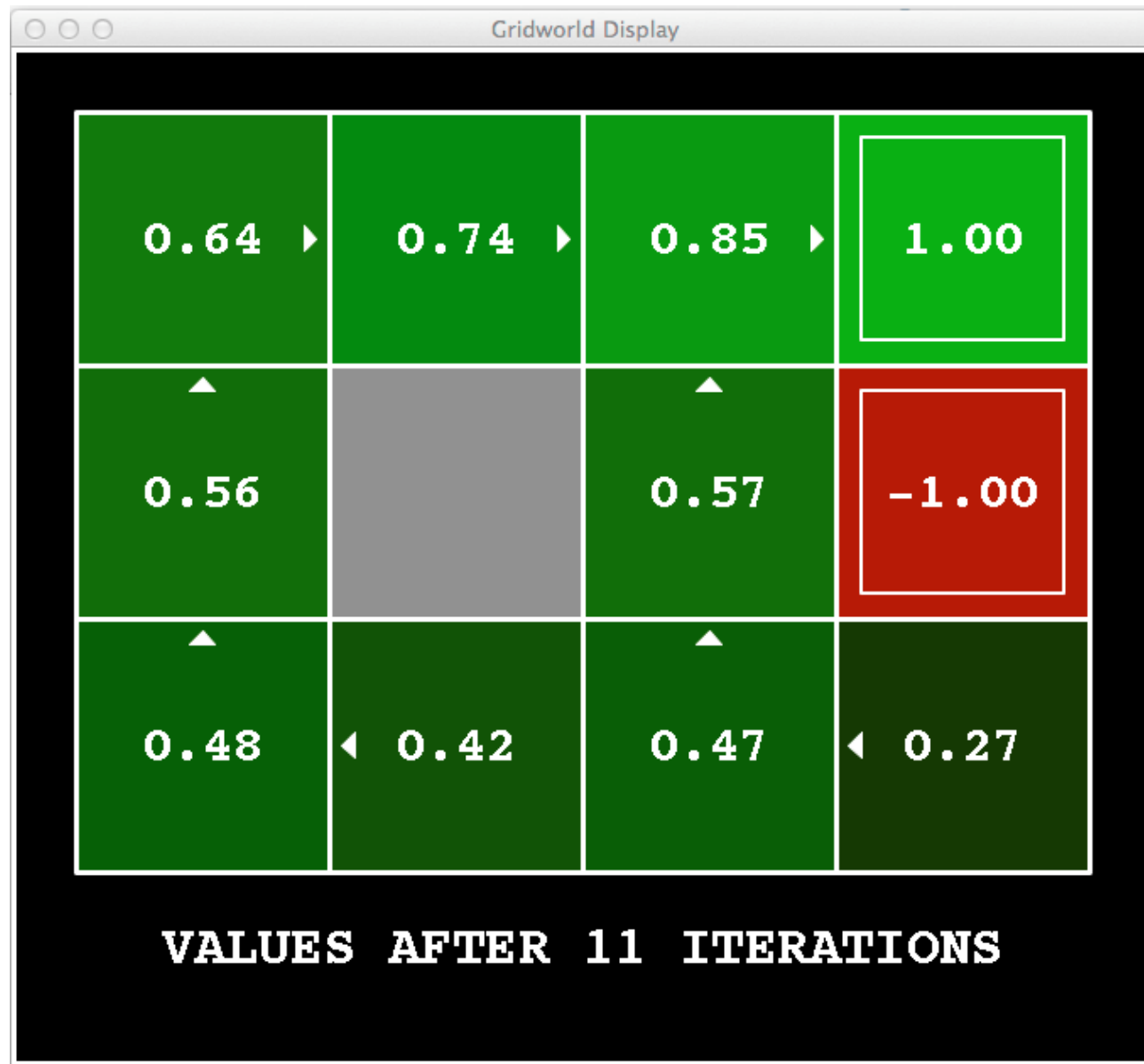
Noise = 0.2
Discount = 0.9
Living reward =

k=10



Noise = 0.2
Discount = 0.9
Living reward =

k=11



Noise = 0.2
Discount = 0.9
Living reward =

k=12



Noise = 0.2
Discount = 0.9
Living reward =

k=100



Noise = 0.2
Discount = 0.9
Living reward =

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:

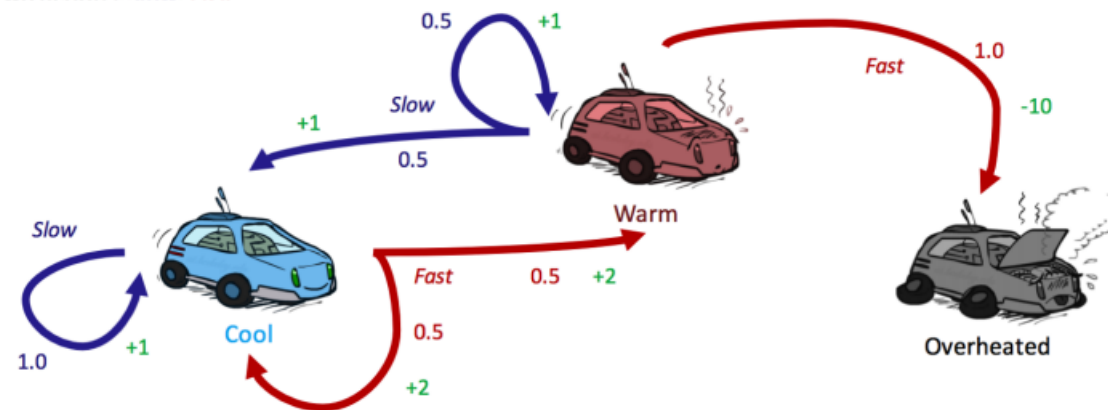
$$V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} T(s, \pi_i(s), s') \left[R(s, \pi_i(s), s') + \gamma V_k^{\pi_i}(s') \right]$$

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

- One-step look-ahead:

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

Question



| | cool | warm | overheated |
|---------|------|------|------------|
| π_0 | slow | slow | — |

$$\gamma = 0.5.$$

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