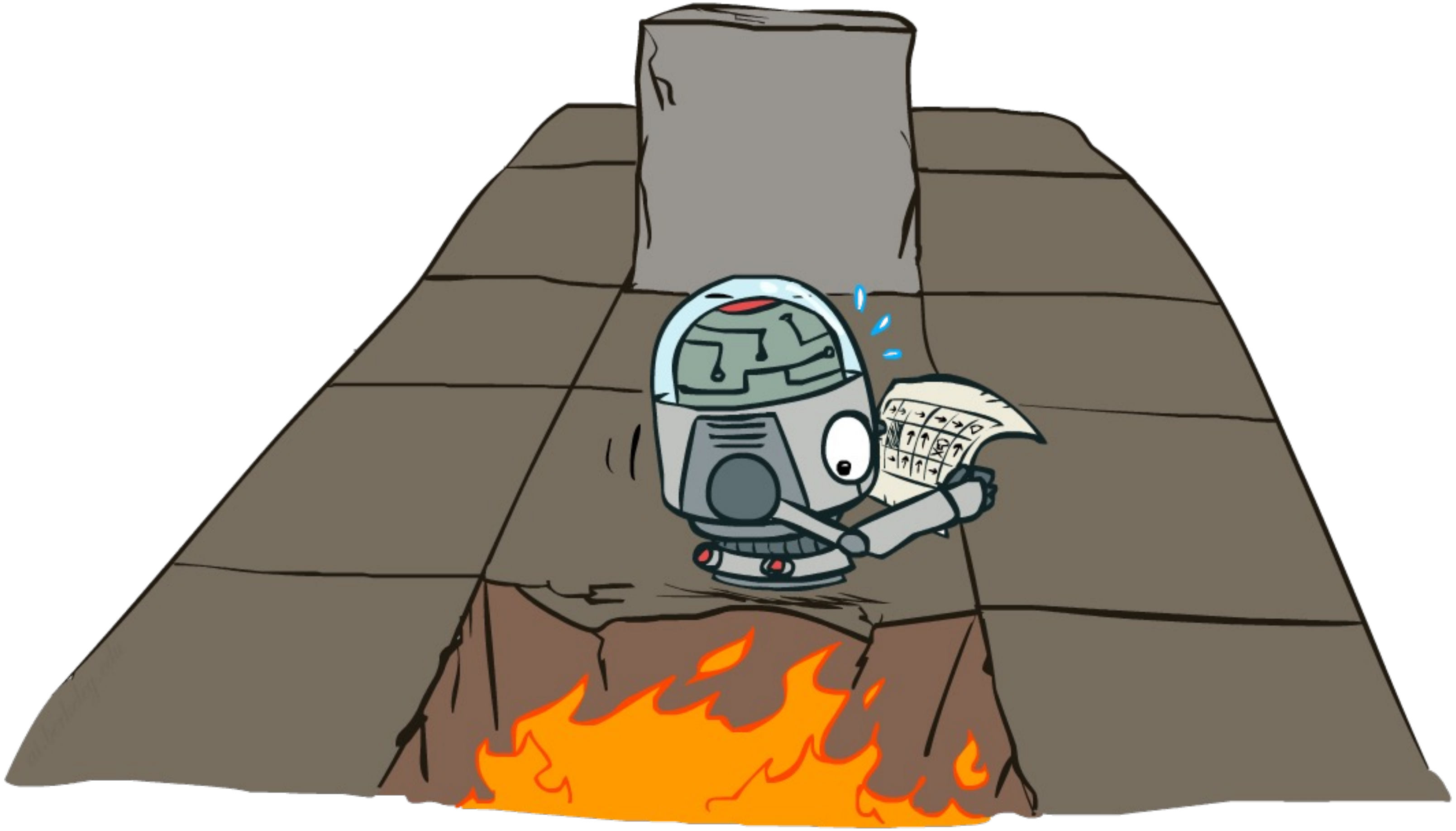
Announcements

- **Project 2** due **this Friday (Sept 22)** at 11:59pm PT
- Igor's Office Hours: **Thursdays 1-2pm PT** in **Soda 734** or **Zoom**
  - https://berkeley.zoom.us/j/2939939817

CS 188: Artificial Intelligence

Markov Decision Processes II



[These slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkeley.edu.]

Today

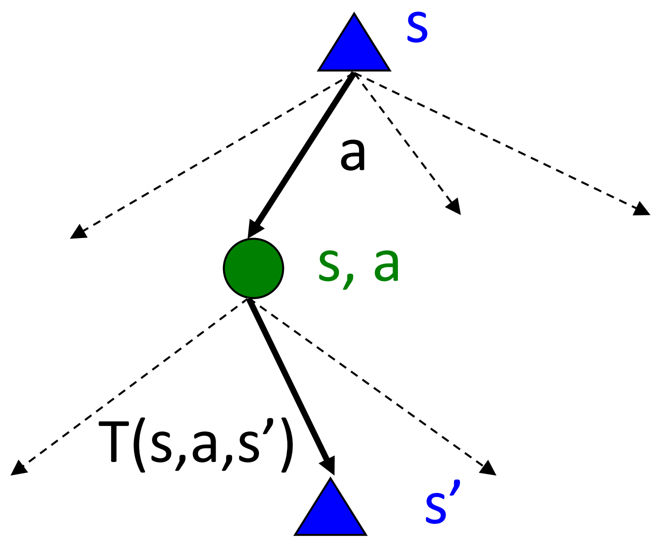
- Review MDPs, Bellman equation, value iteration

- Policy extraction, policy evaluation, policy iteration

    - All based on the Bellman equation

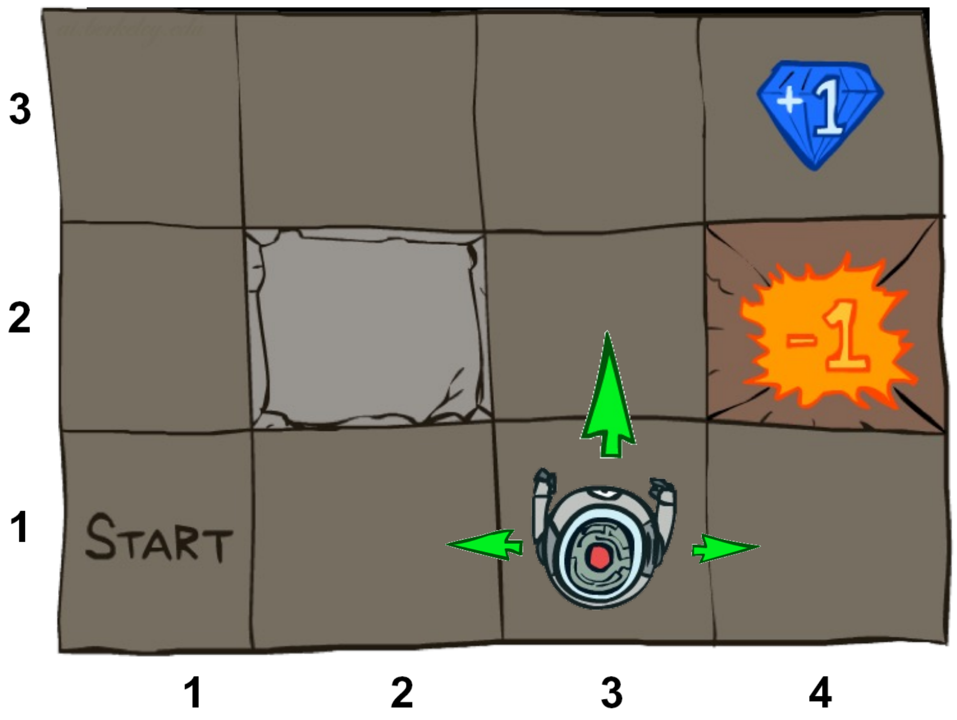
Recap: MDPs

- 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

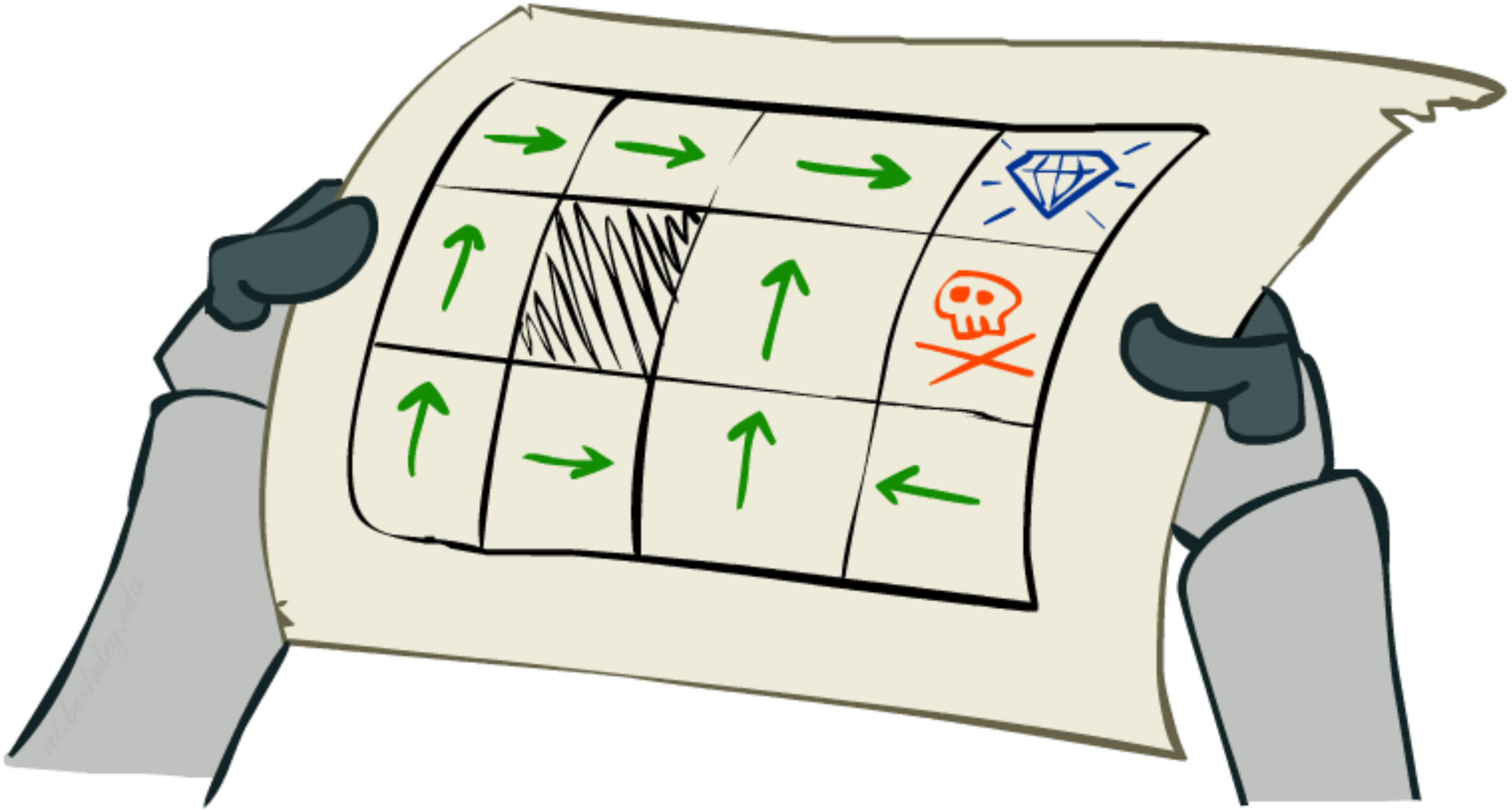


Example: Grid World

- 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



Solving MDPs



Optimal Quantities

- The value (utility) of a state s:

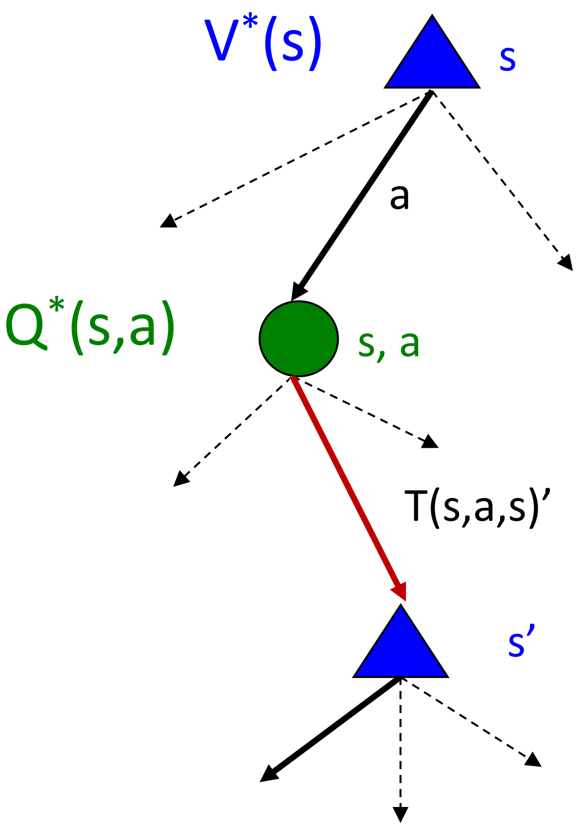
  $V^*(s)$ = expected utility starting in s and acting optimally

- 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



s is a *state*

(s, a) is a *q-state*

(s,a,s') is a *transition*

Optimal Quantities

- The value (utility) of a state s:

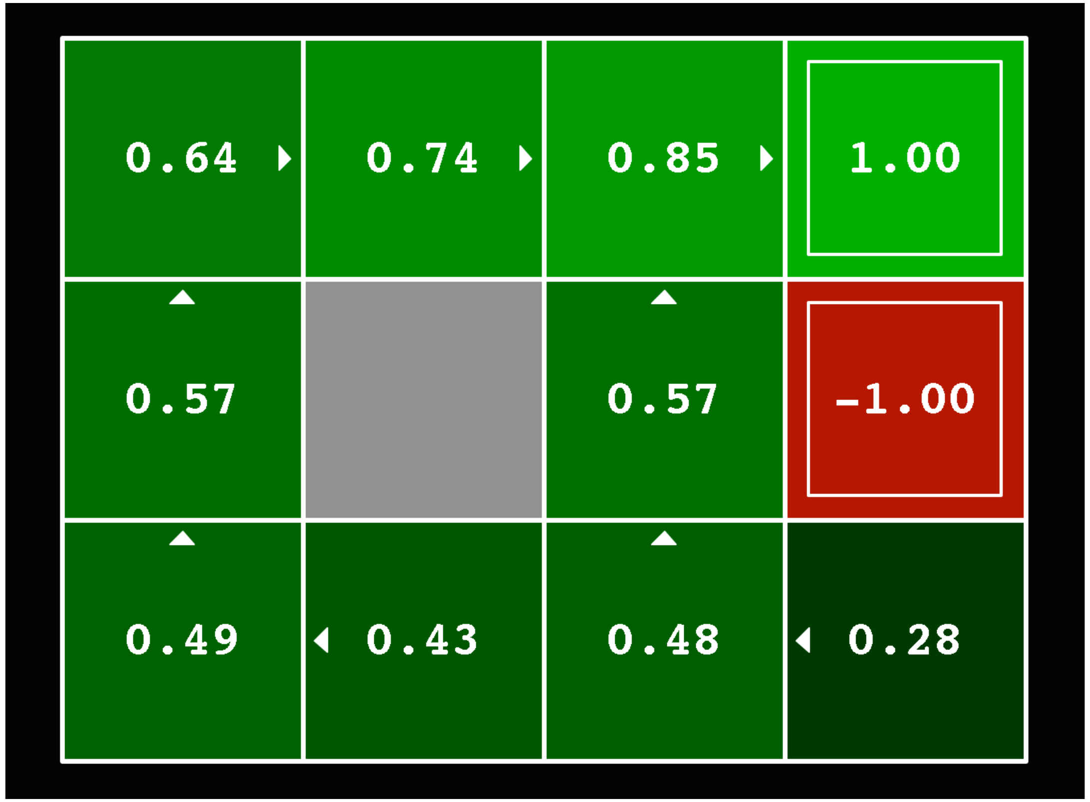
  $V^*(s)$ = expected utility starting in s and acting optimally

- 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



Here V*(s) is a lookup table with 11 entries

Optimal Quantities

- The value (utility) of a state s:

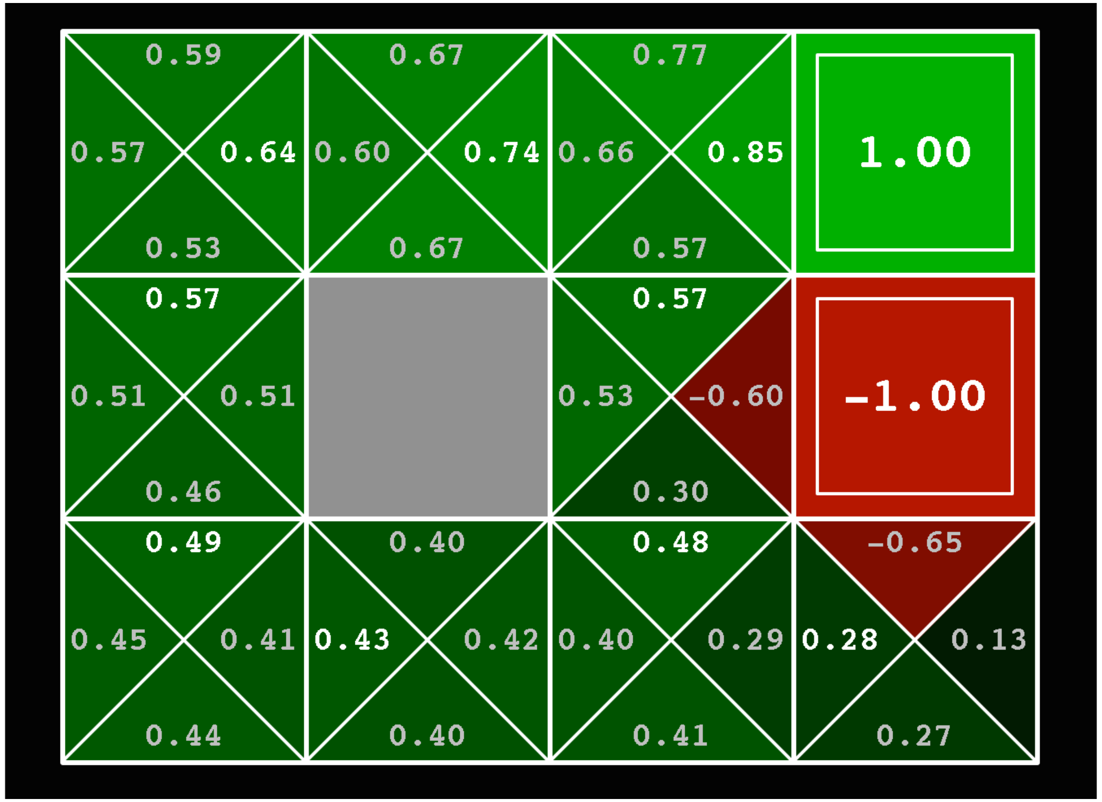
  $V^*(s)$ = expected utility starting in s and acting optimally

- 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



Here Q*(s,a) is a lookup table with 9*4+2 entries

Optimal Quantities

- The value (utility) of a state s:

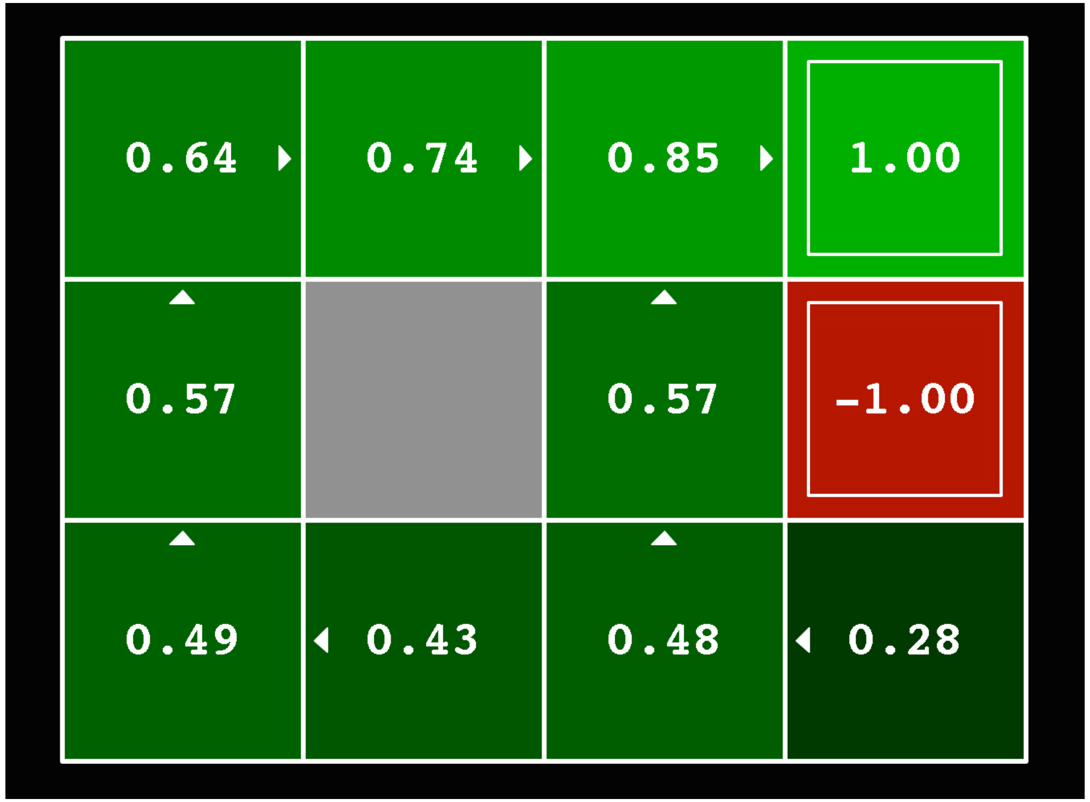
  $V^*(s)$ = expected utility starting in s and acting optimally

- 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:

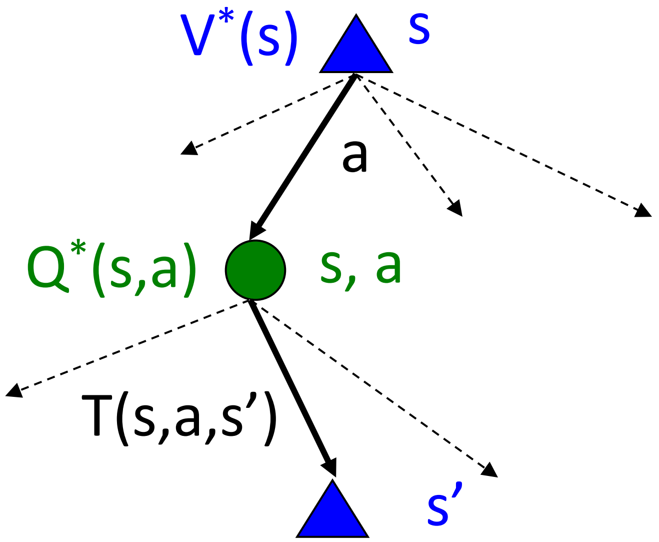
  $\pi^*(s)$ = optimal action from state s



Here $\pi^*(s)$ is a lookup table with 11 entries

The Bellman Equations

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



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

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

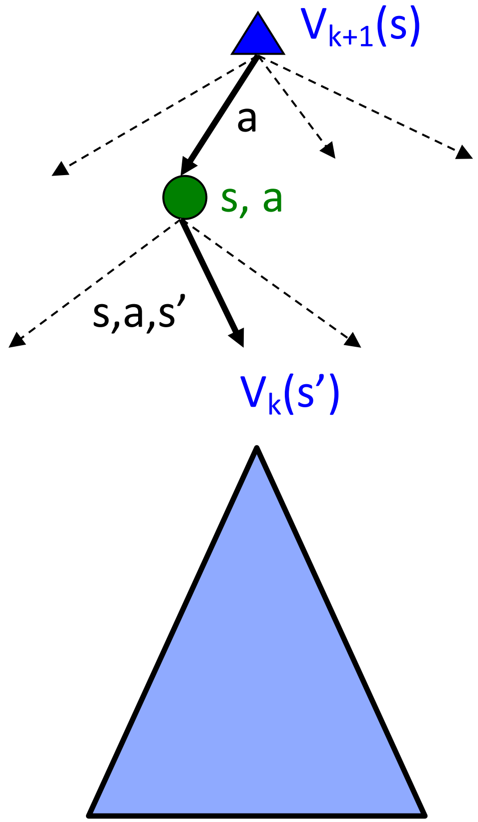
$$V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s') \right]$$

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

Value Iteration

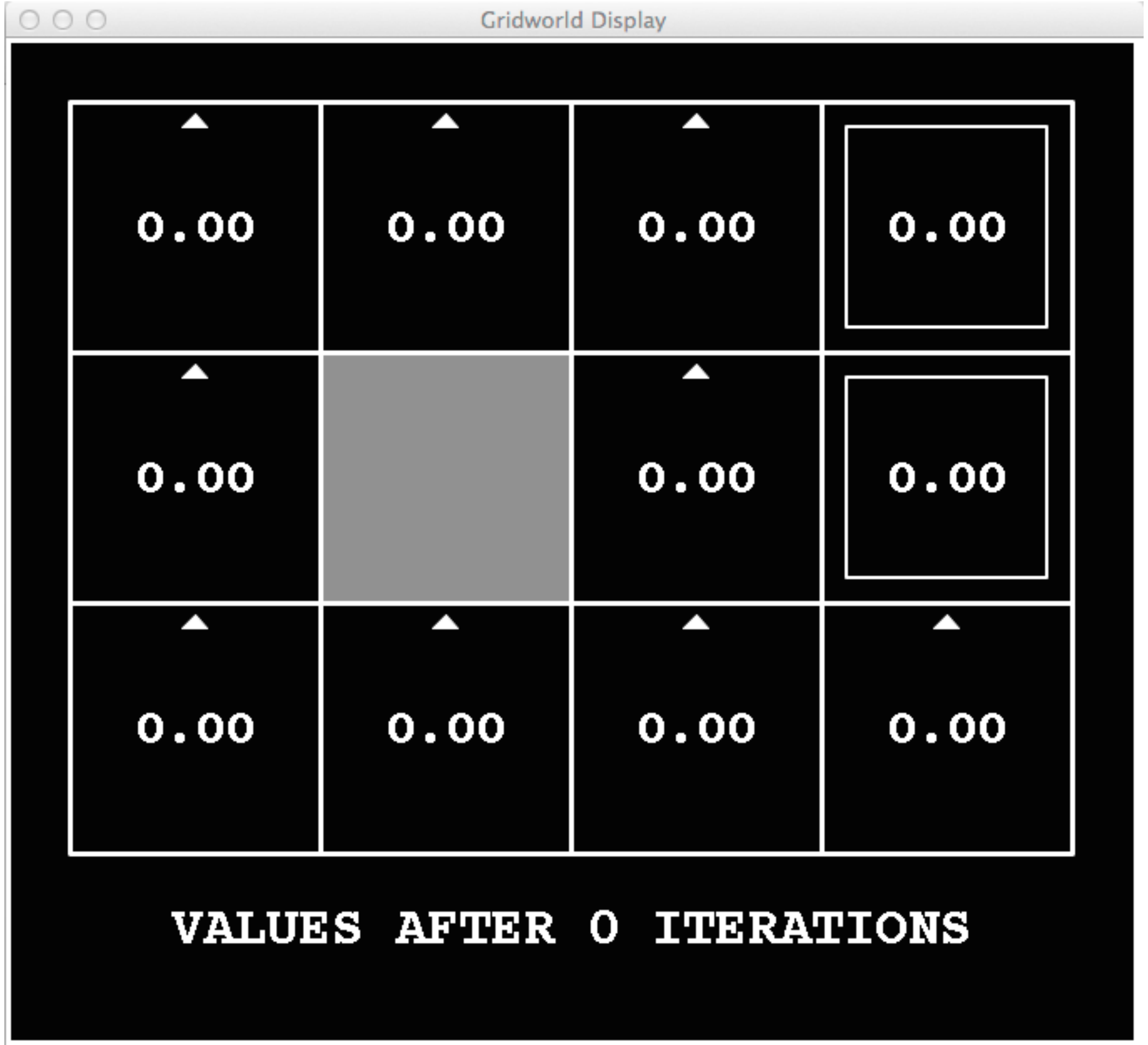
- Start with $V_0(s) = 0$: no time steps left means an expected reward sum of zero
- Given vector of $V_k(s)$ values, do one step of expectimax from each state:

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



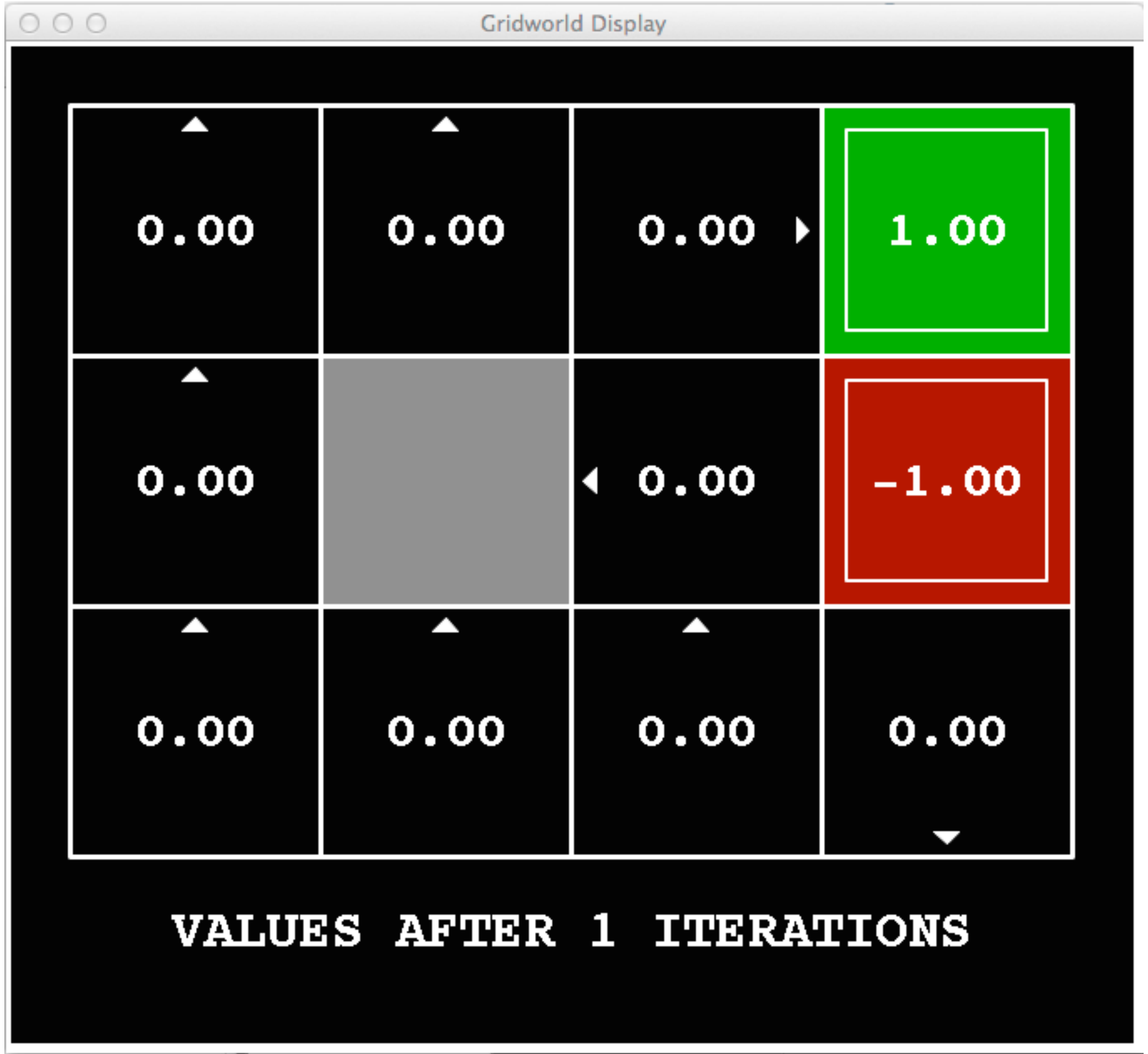
- Repeat until convergence, which yields V*
- Complexity of each iteration: $O(S^2A)$
- Theorem: will converge to unique optimal values
  - Basic idea: approximations get refined towards optimal values
  - Policy may converge long before values do

k=0

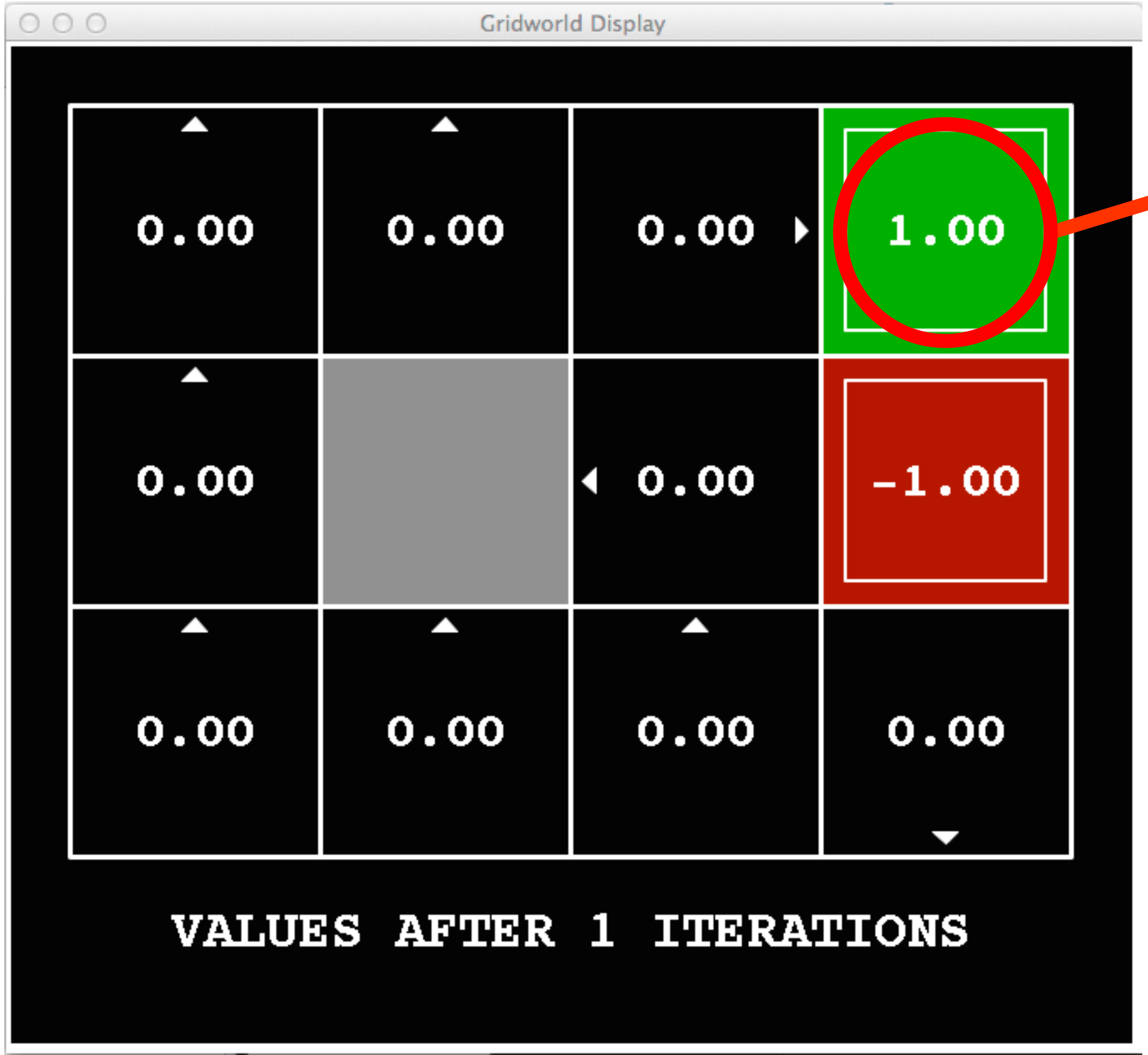


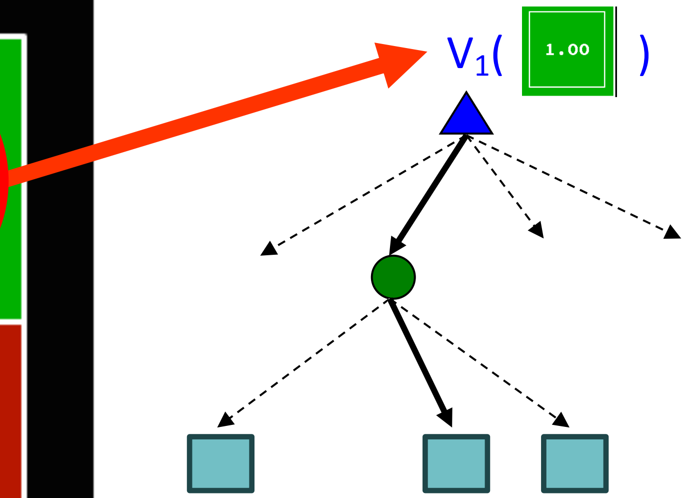
Noise = 0.2
Discount = 0.9
Living reward = 0

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

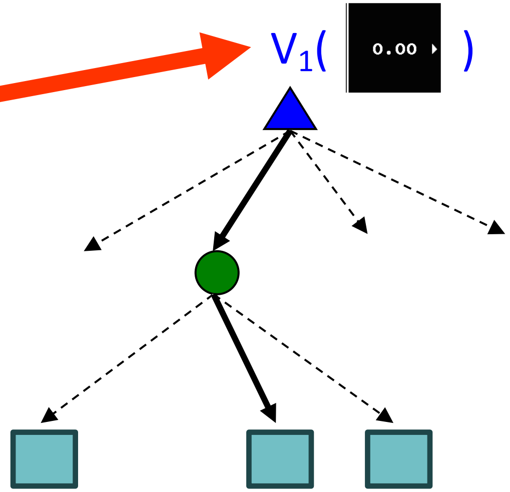
k=1





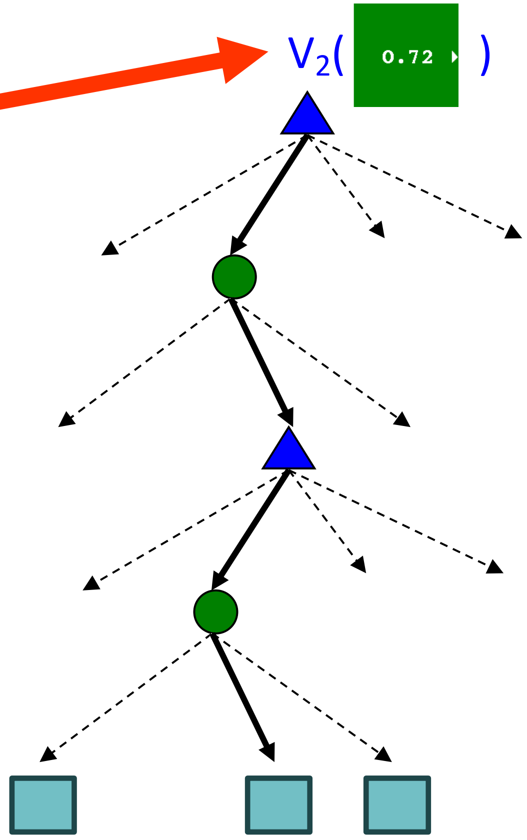
$V_1(s)$ is value of depth-1 expectimax from *s*

k=1



V₁(s) is value of depth-1 expectimax from *s*

k=2

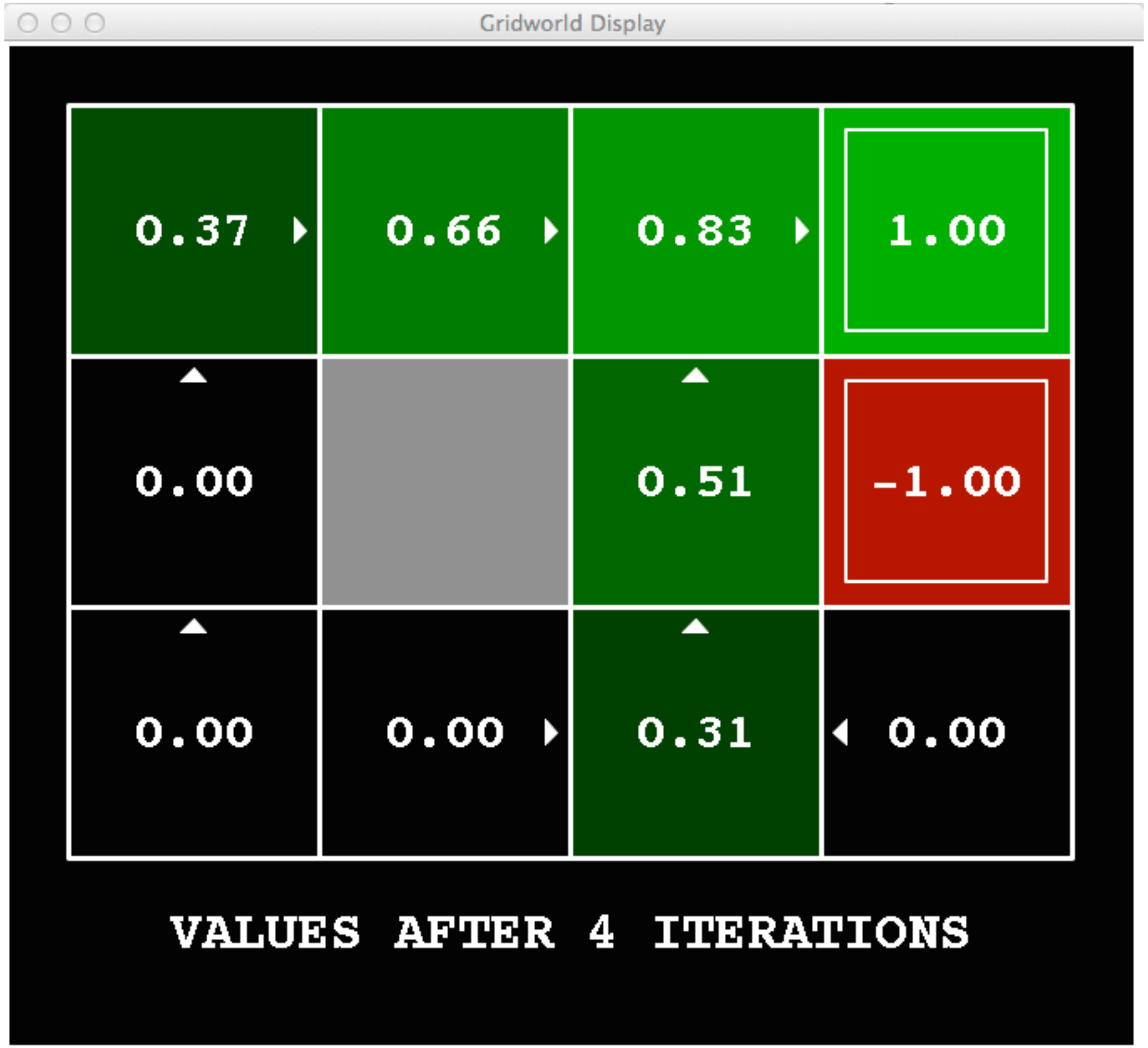


$V_2(s)$ is value of depth-2 expectimax from s

k=3

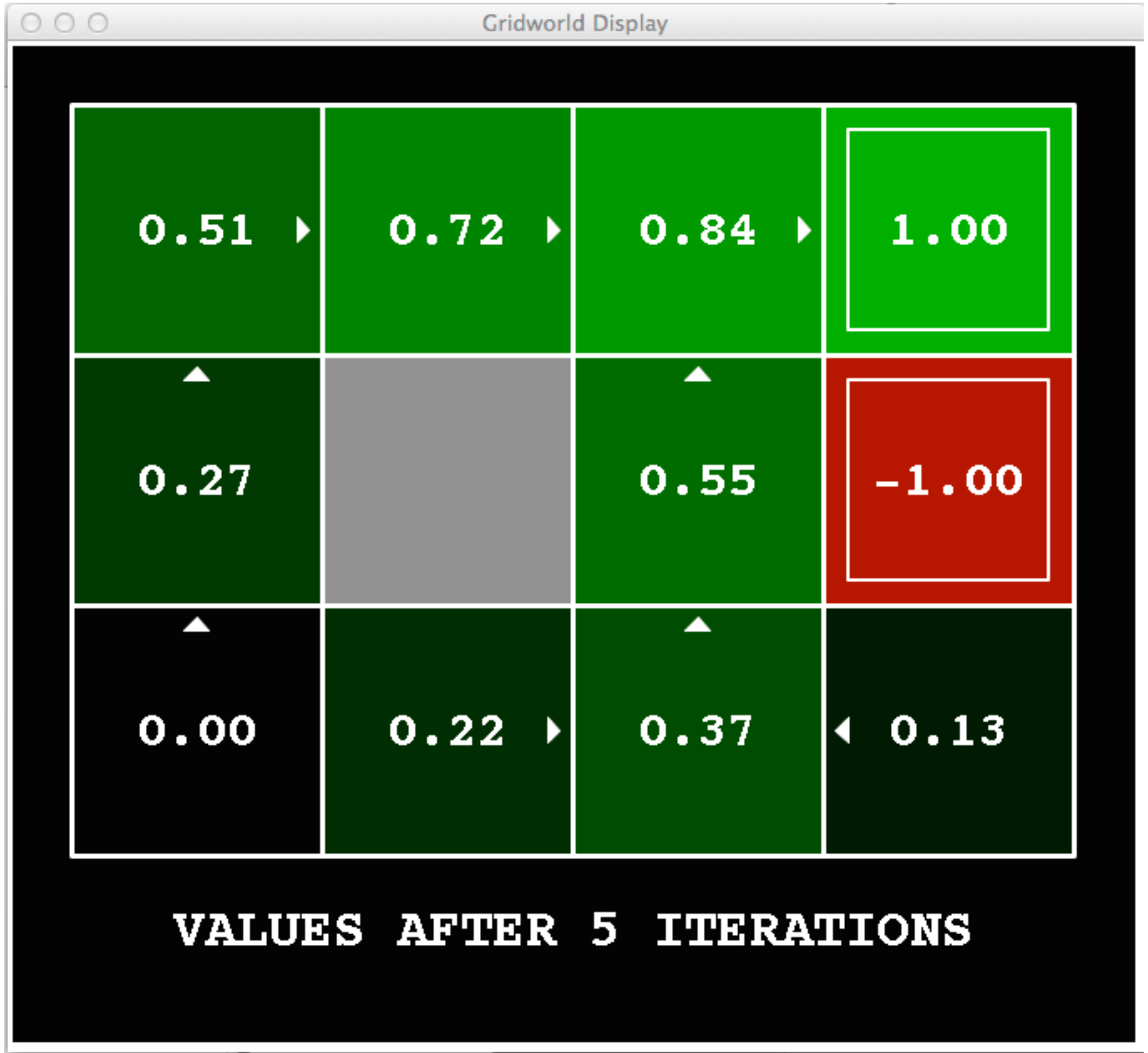
Noise = 0.2
Discount = 0.9
Living reward = 0

k=4



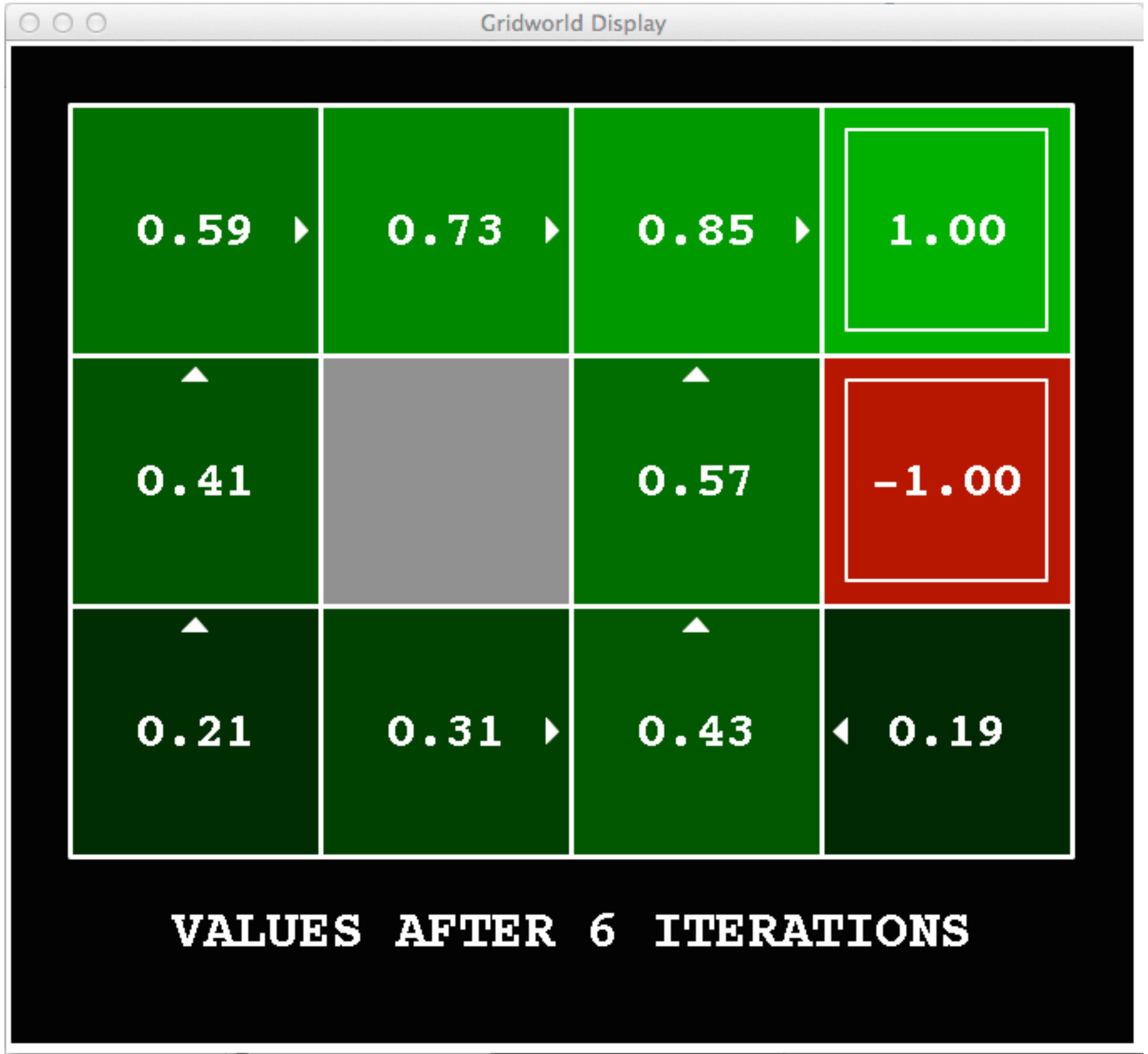
Noise = 0.2
Discount = 0.9
Living reward = 0

k=5



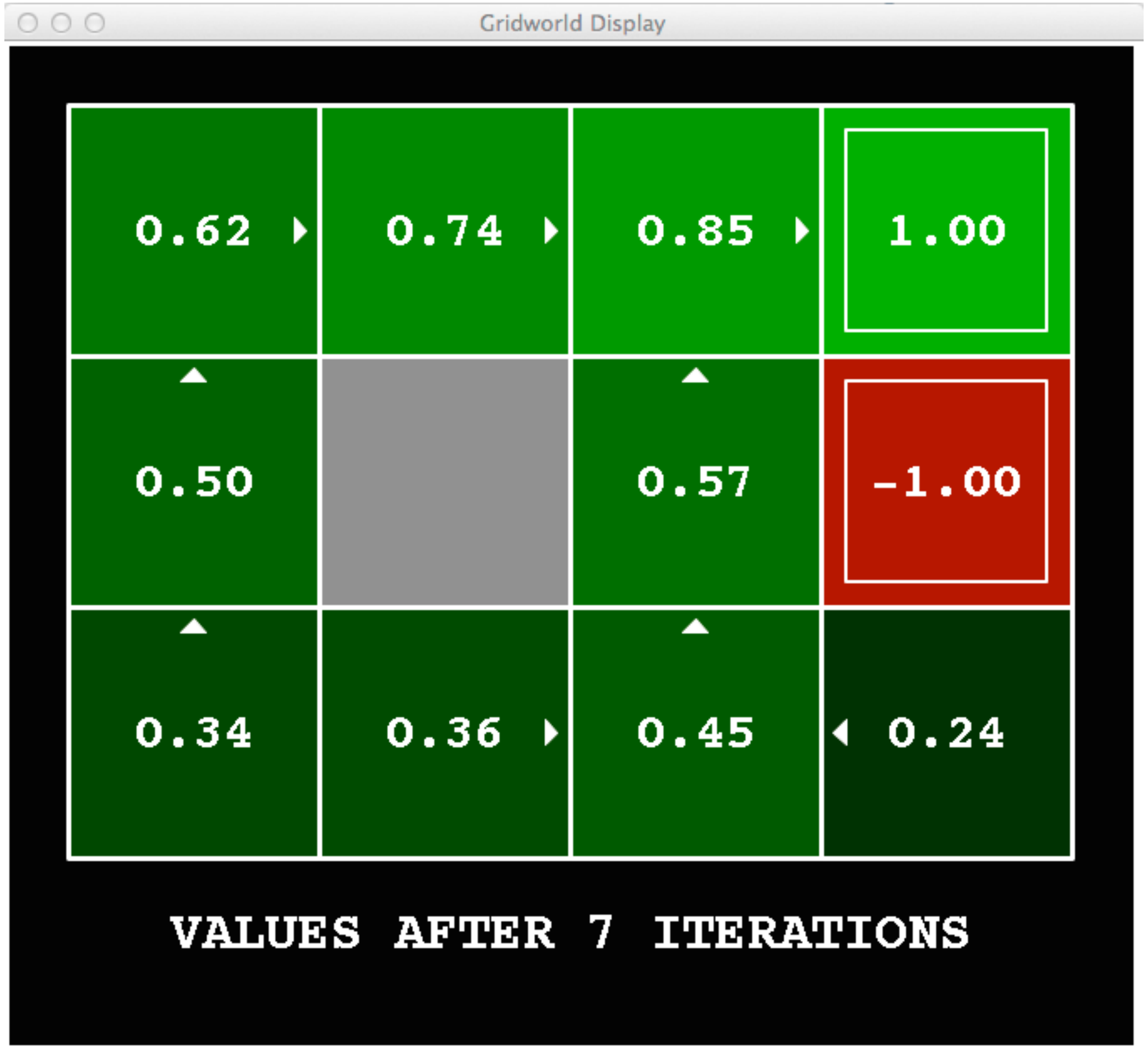
Noise = 0.2
Discount = 0.9
Living reward = 0

k=6



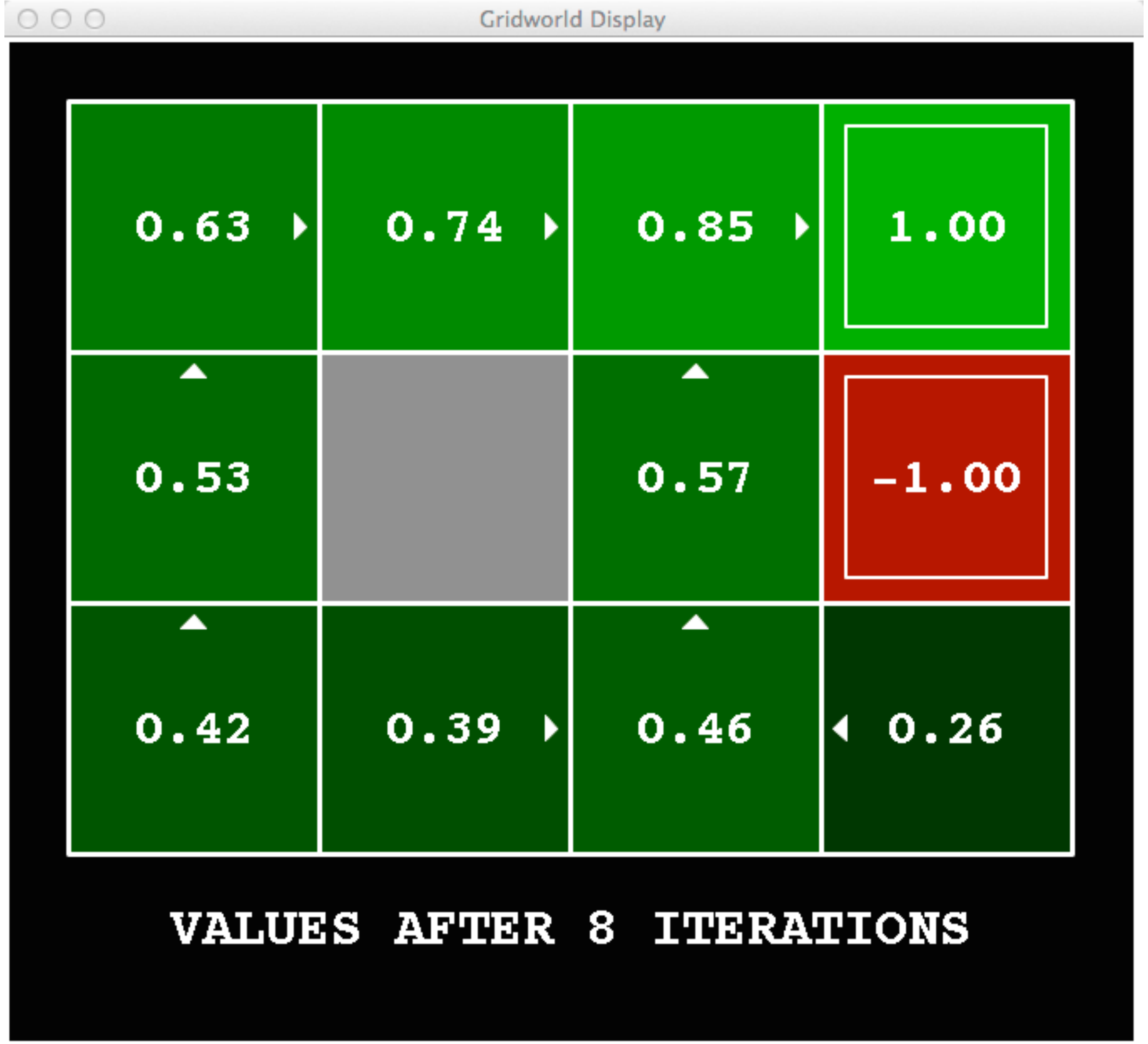
Noise = 0.2
Discount = 0.9
Living reward = 0

k=7



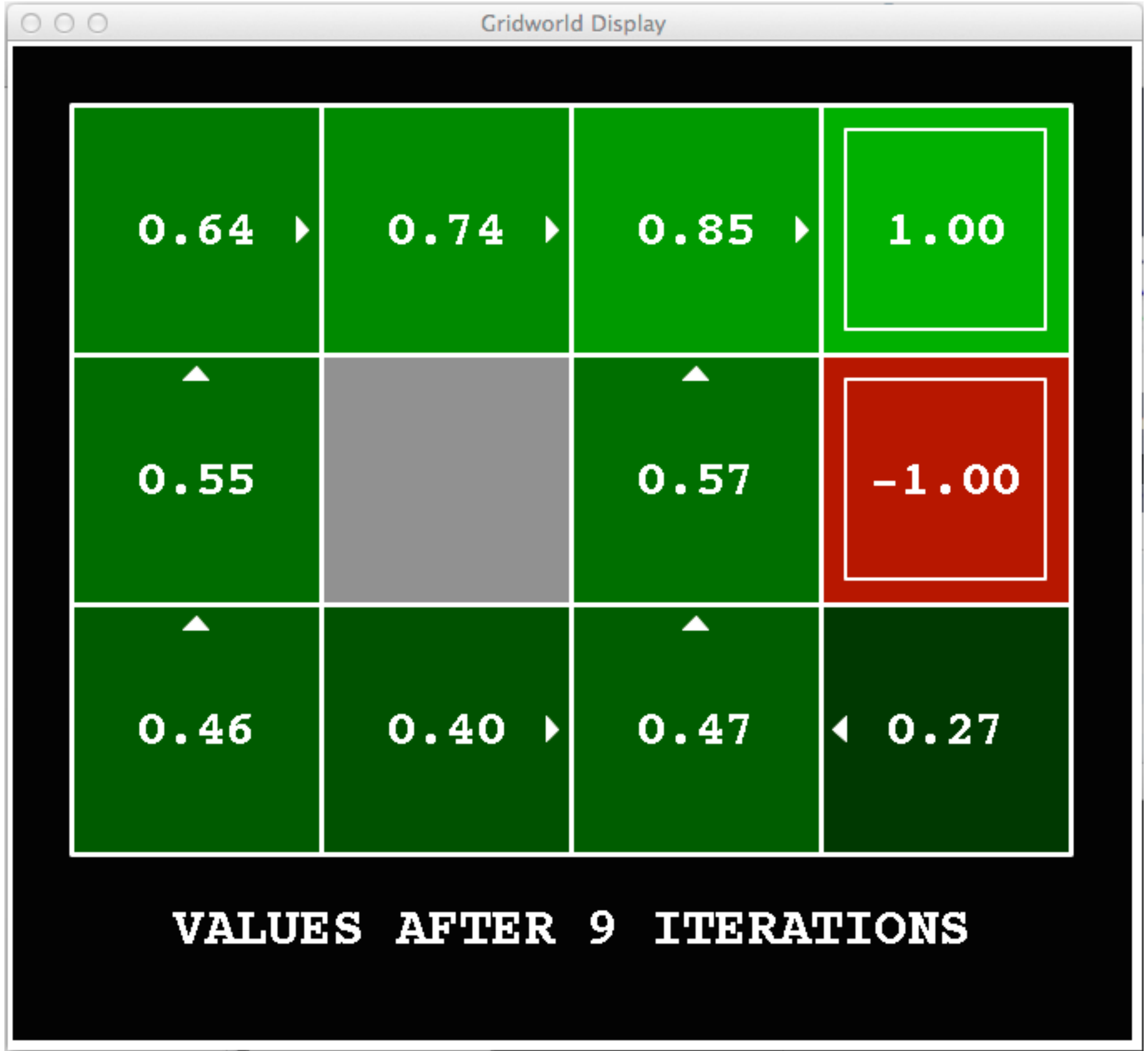
Noise = 0.2
Discount = 0.9
Living reward = 0

k=8



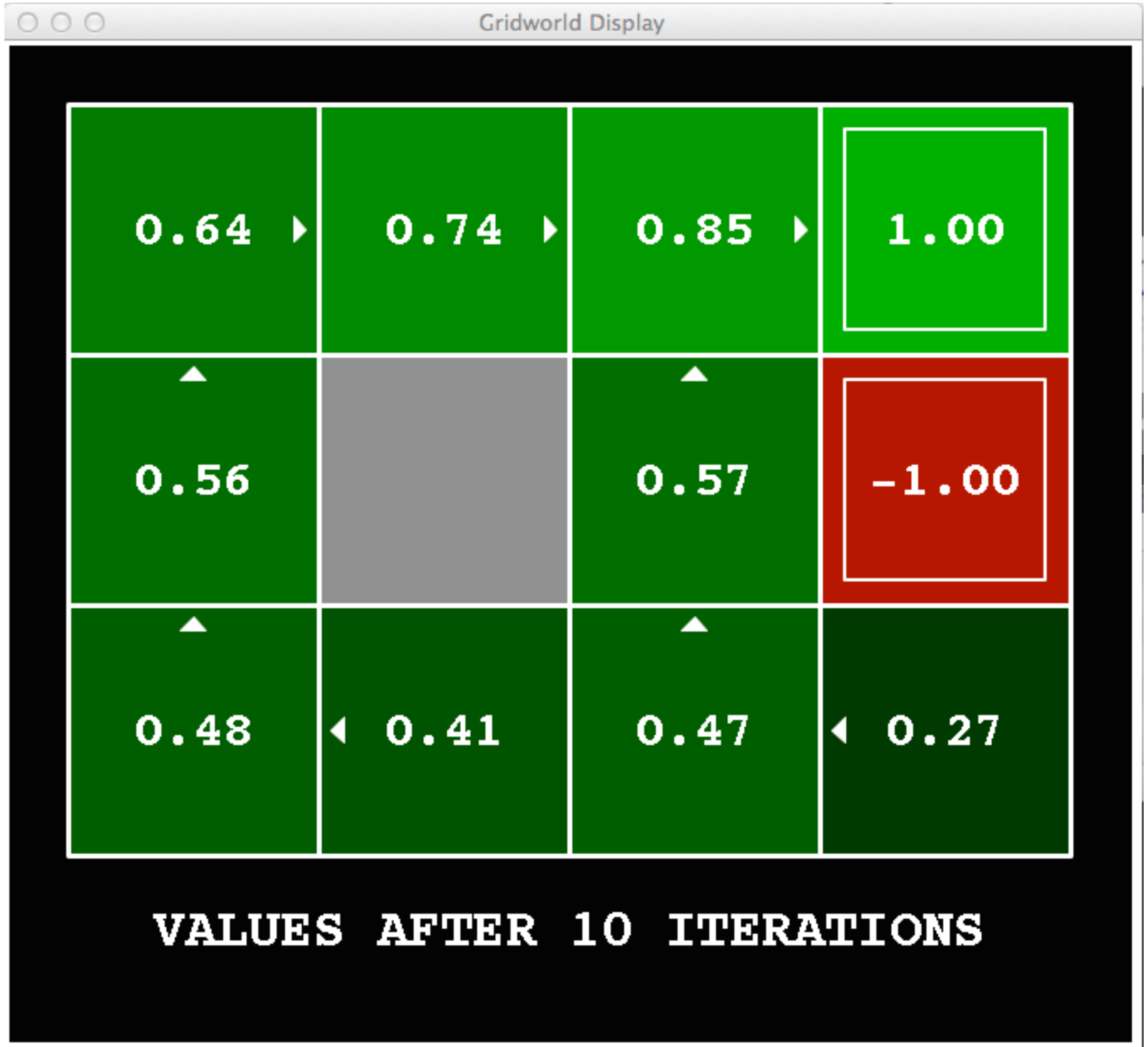
Noise = 0.2
Discount = 0.9
Living reward = 0

k=9



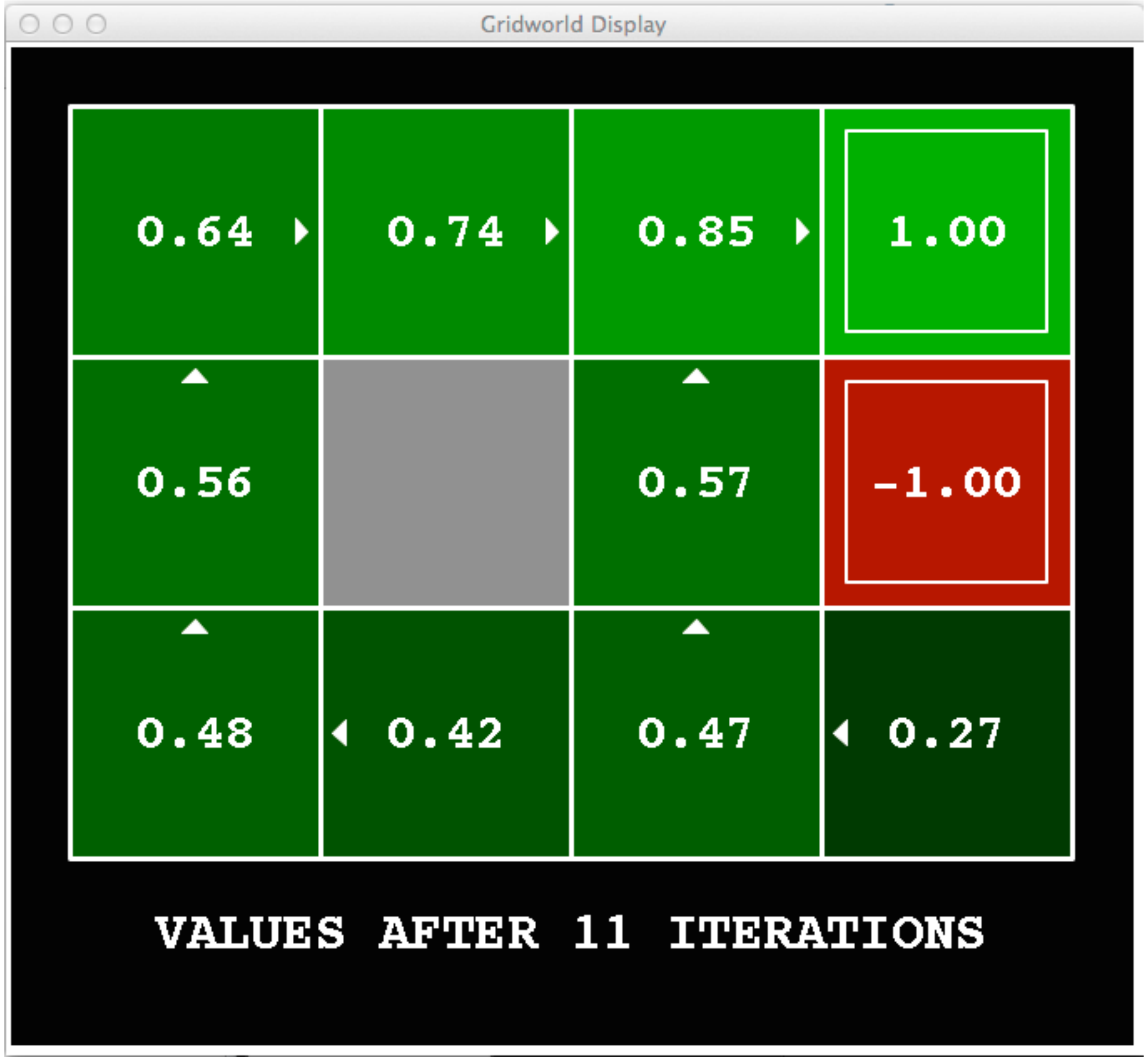
Noise = 0.2
Discount = 0.9
Living reward = 0

k=10



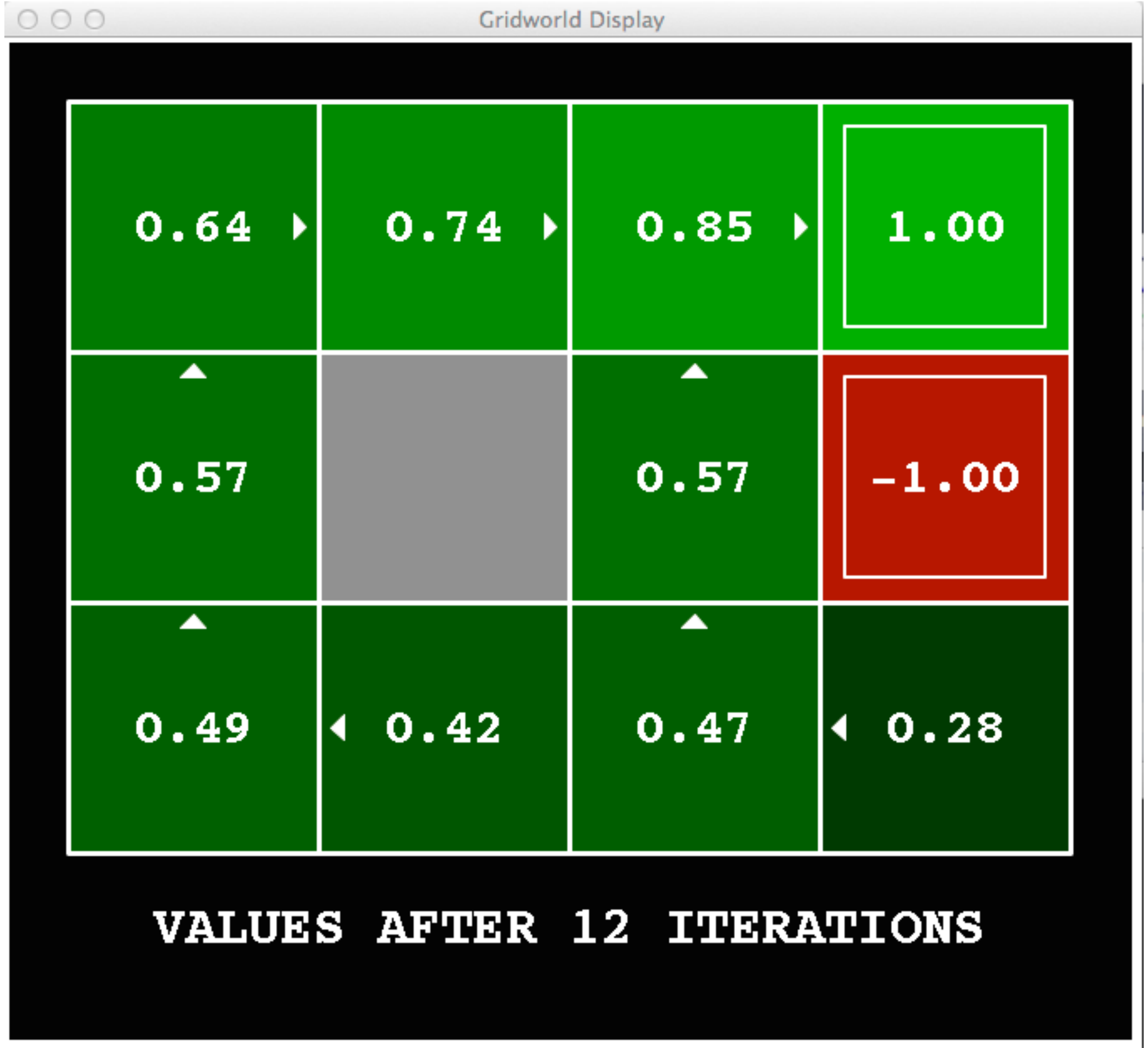
Noise = 0.2
Discount = 0.9
Living reward = 0

k=11



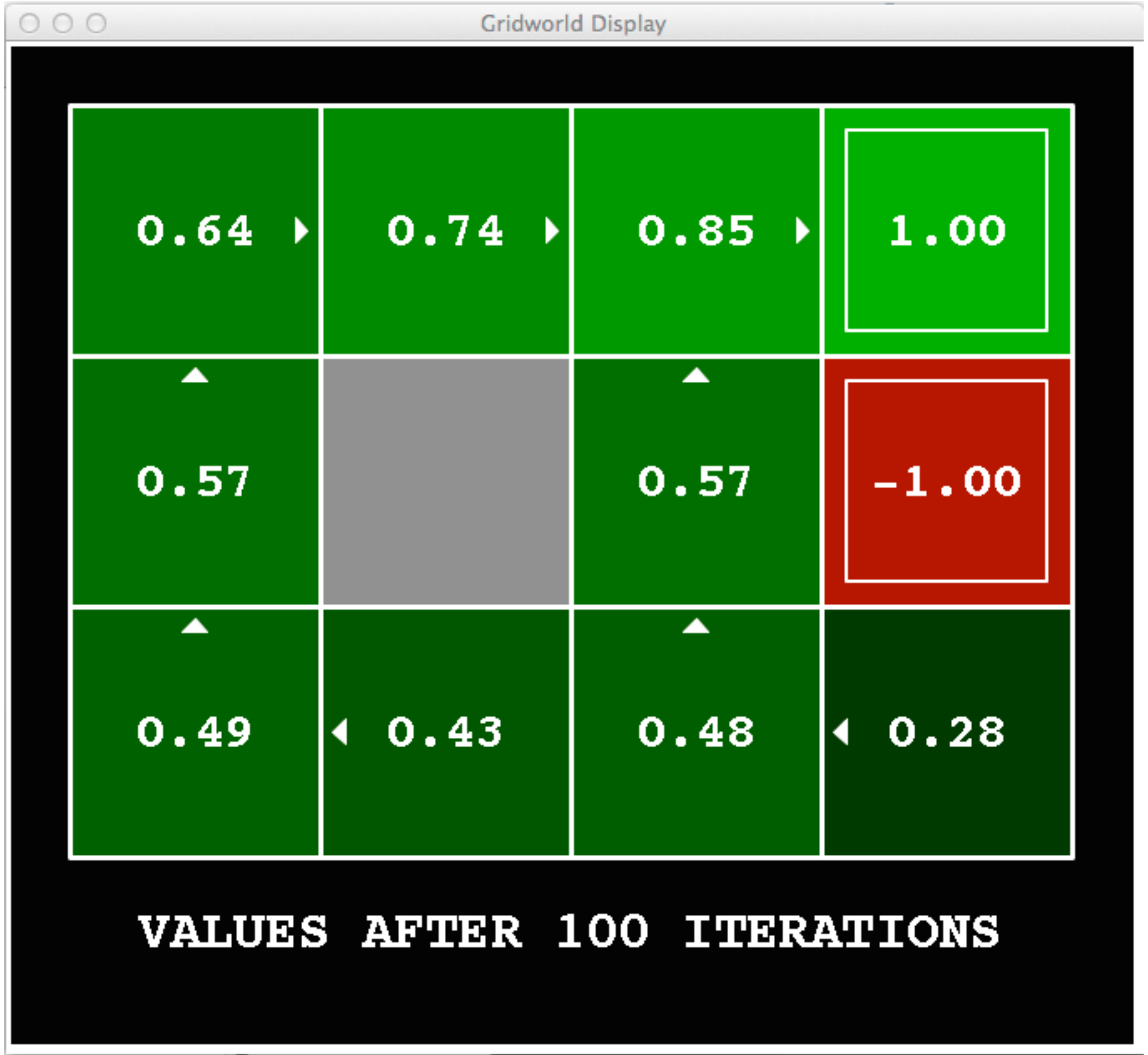
Noise = 0.2
Discount = 0.9
Living reward = 0

k=12



Noise = 0.2
Discount = 0.9
Living reward = 0

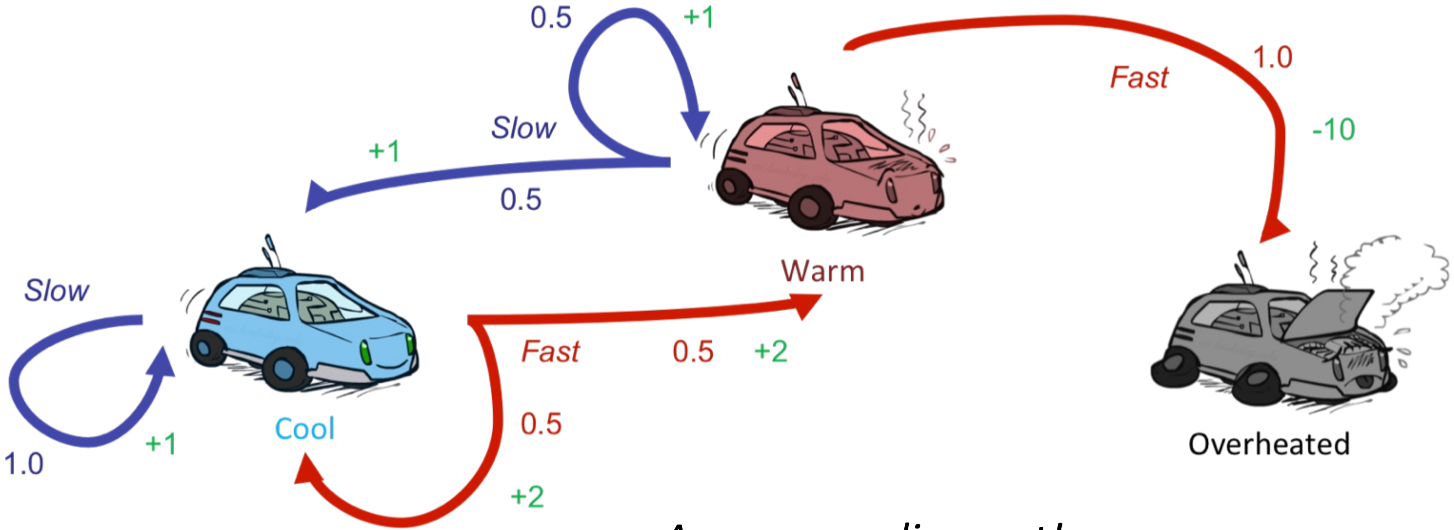
k=100



Noise = 0.2
Discount = 0.9
Living reward = 0

Example: Value Iteration

V_2			
V_1			
V_0	0	0	0

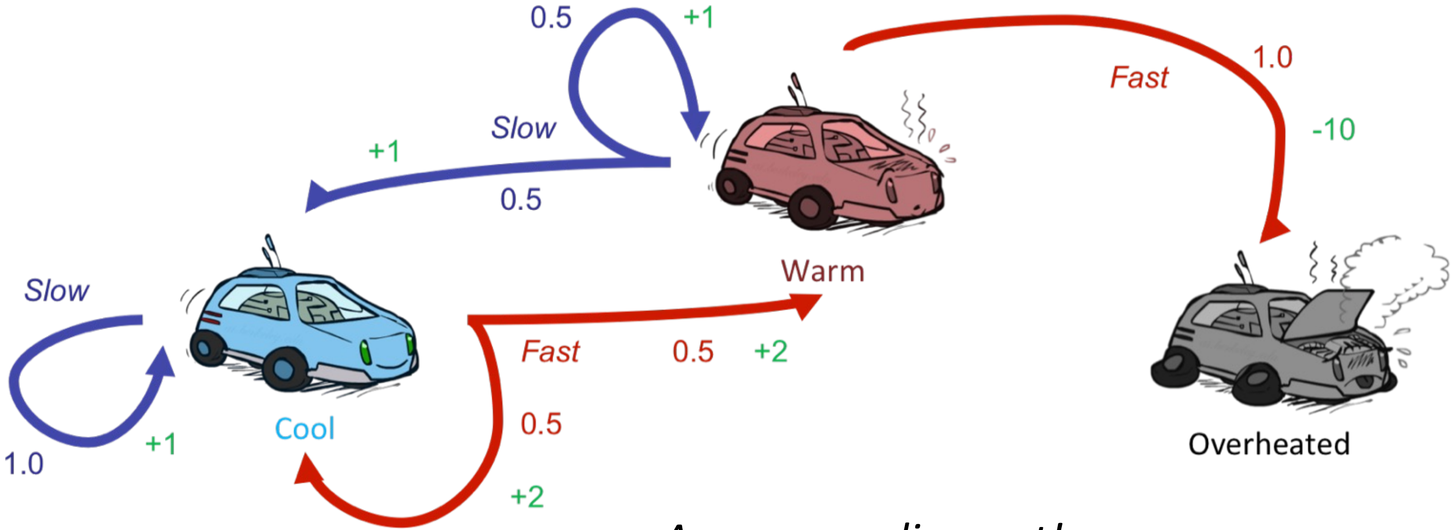


Assume no discount!

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

Example: Value Iteration

V_2			
V_1	?		
V_0	0	0	0



Assume no discount!

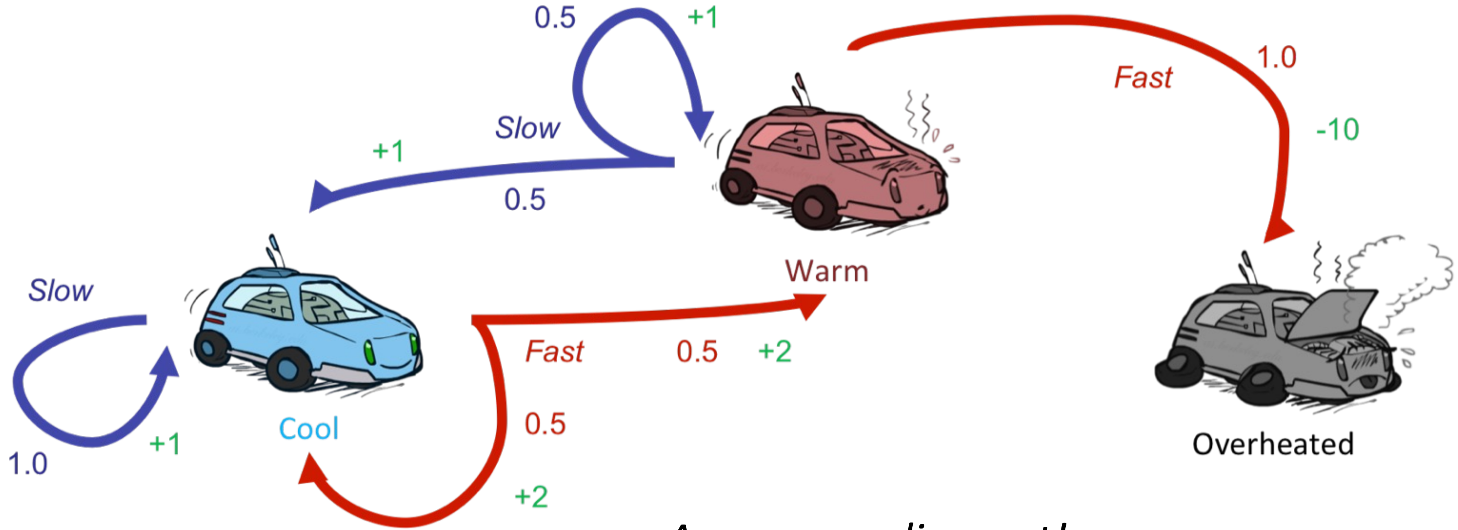
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s')\right]$$

a=slow: 1(1 + 0) = 1

a=fast: 0.5(2 + 0) + 0.5(2 + 0) = 2

Example: Value Iteration

V_2			
V_1	2	?	
V_0	0	0	0



Assume no discount!

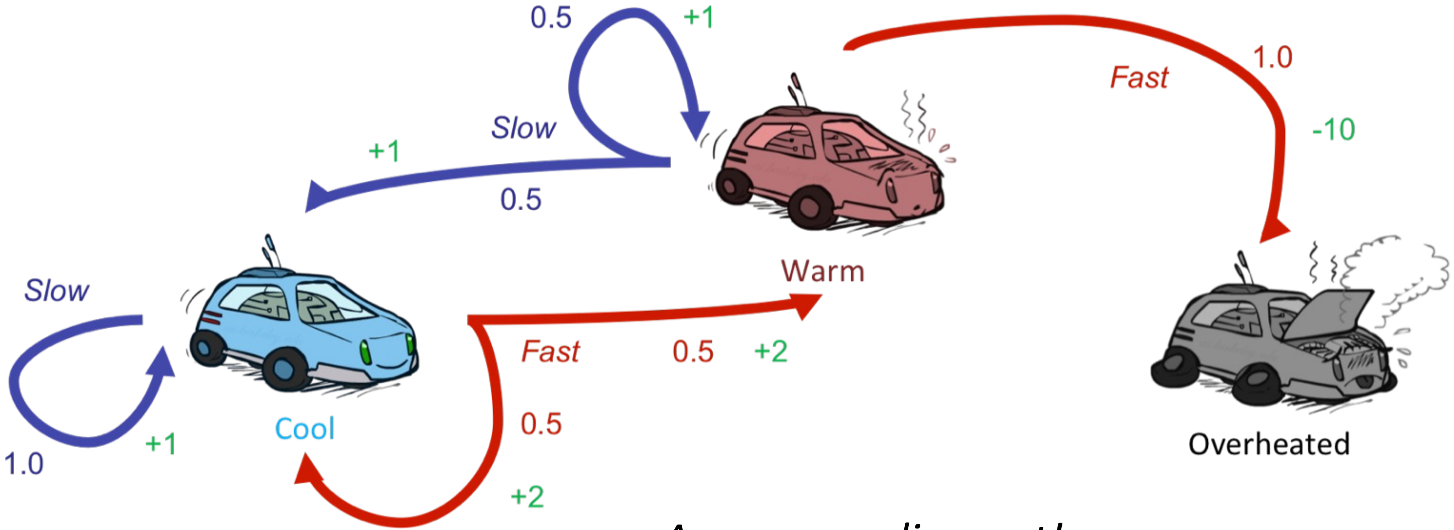
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right]$$

a=slow: 0.5(1 + 0) + 0.5(1 + 0) = 1

a=fast: 1(-10 + 0) = -10

Example: Value Iteration

V_2	?		
V_1	2	1	0
V_0	0	0	0



Assume no discount!

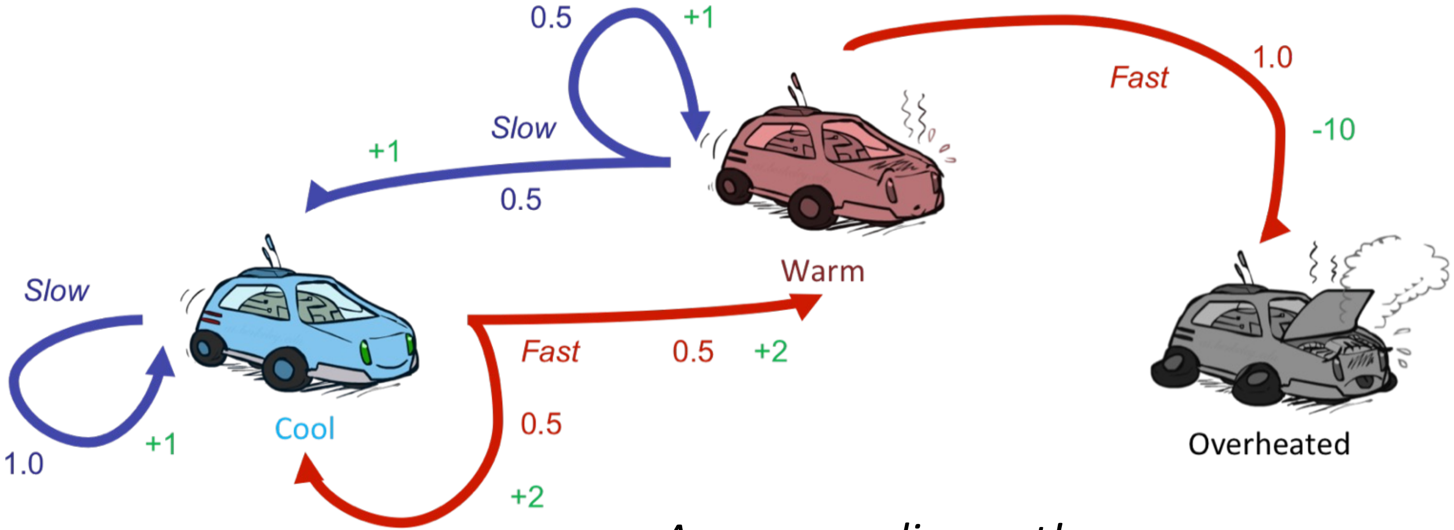
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s')\right]$$

a=slow: $1(1 + 2) = 3$

a=fast: $0.5(2 + 2) + 0.5(2 + 1) = 3.5$

Example: Value Iteration

V_2	3.5	?	0
V_1	2	1	0
V_0	0	0	0



Assume no discount!

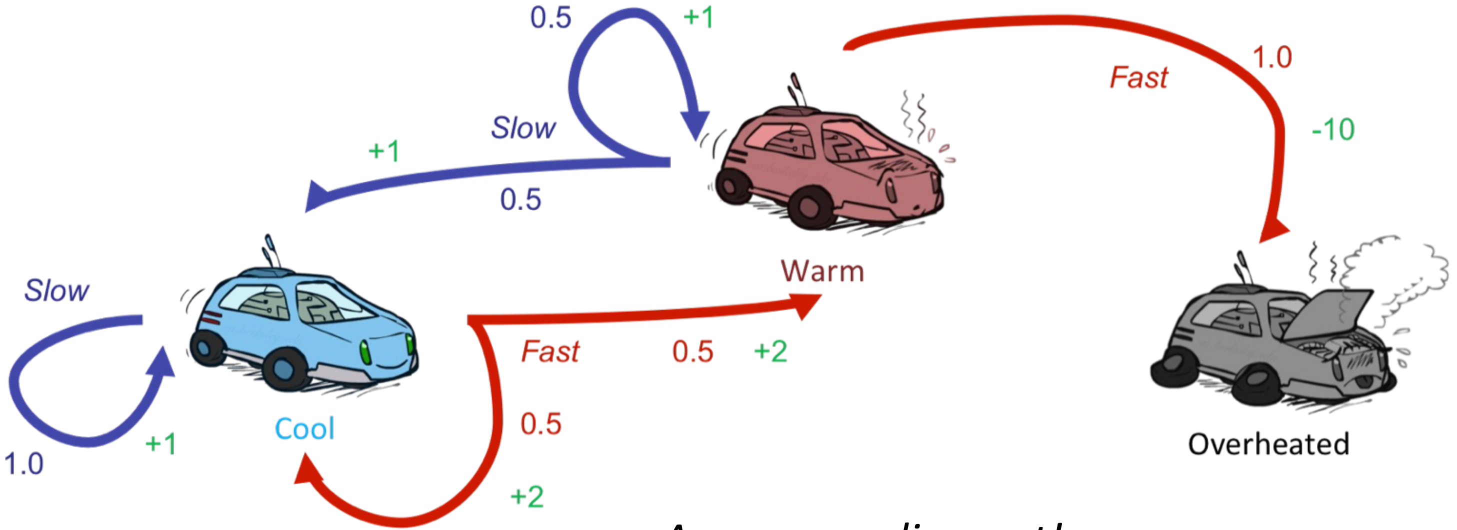
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right]$$

a=slow: 0.5(1 + 2) + 0.5(1 + 1) = 2.5

a=fast: 1(-10 + 0) = -10

Example: Value Iteration

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



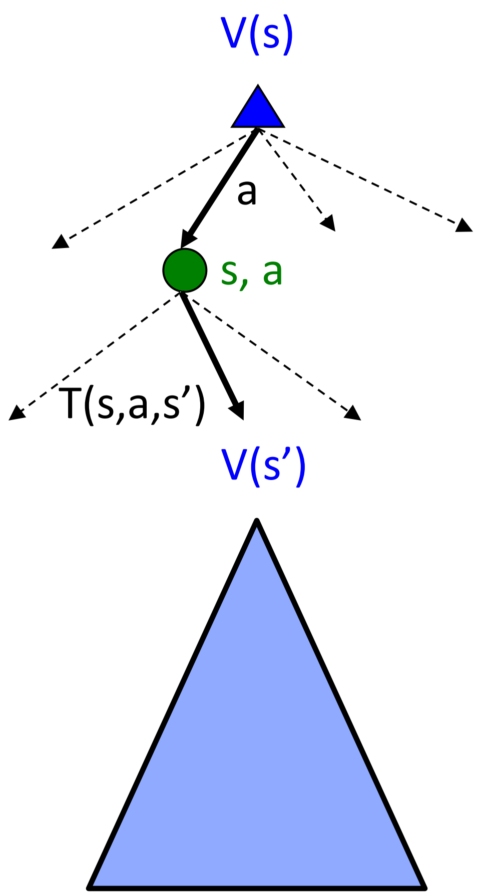
Assume no discount!

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

Value Iteration

- 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:

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

- Value iteration is just a fixed point solution method
  - … though the V_k vectors are also interpretable as time-limited values
  - There may be other methods to solve this Bellman equation

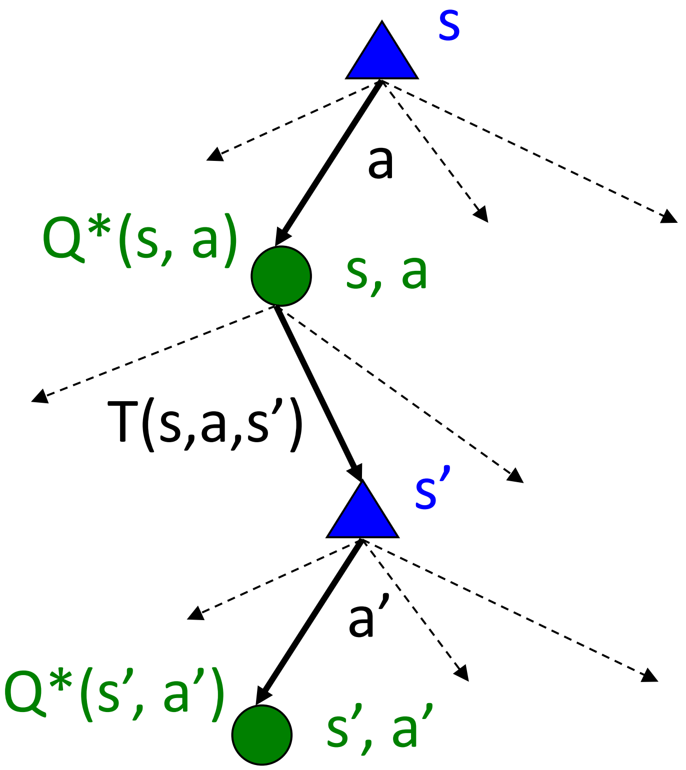
Quiz: Bellman equation for Q values?

- We saw Bellman equation that characterized optimal V*(s)

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

- Can we write down Bellman equation for *Q*(s,a)*?

$$Q^*(s,a) = \qquad ??? \qquad Q^*(s',a')$$



(don't look at the next slide if you're following along with the notes please :)

Quiz: Bellman equation for Q values?

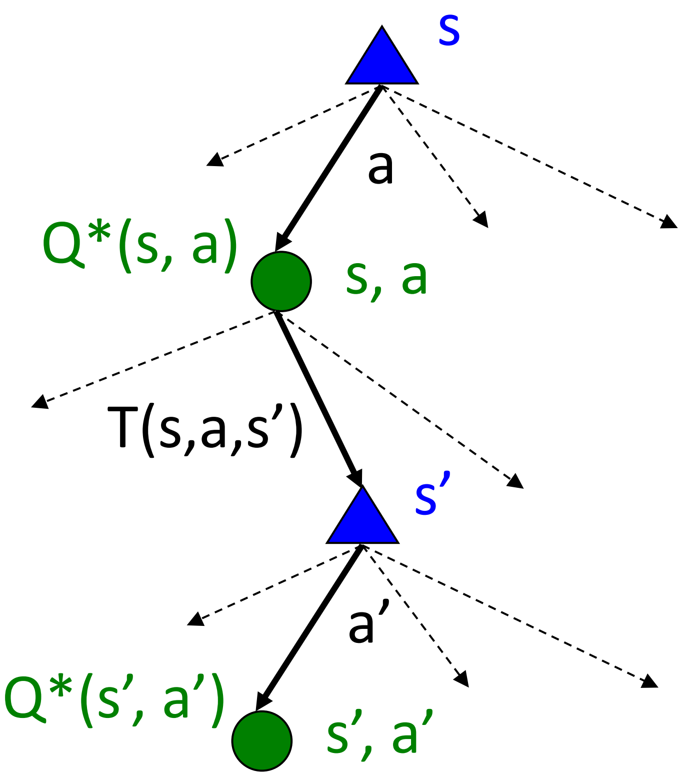
- We saw Bellman equation that characterized optimal V*(s)

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

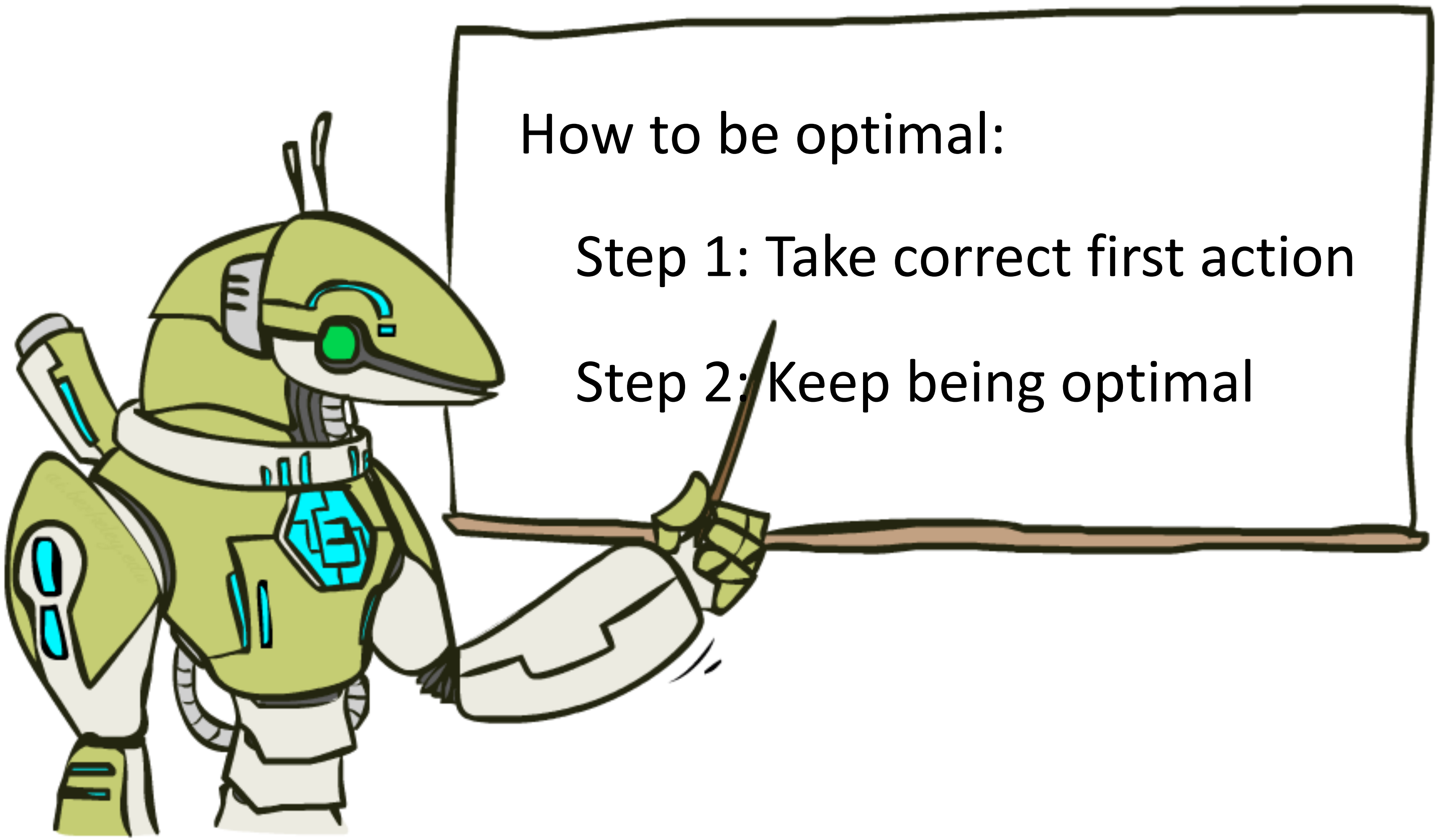
- Can we write down Bellman equation for *Q*(s,a)*?

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

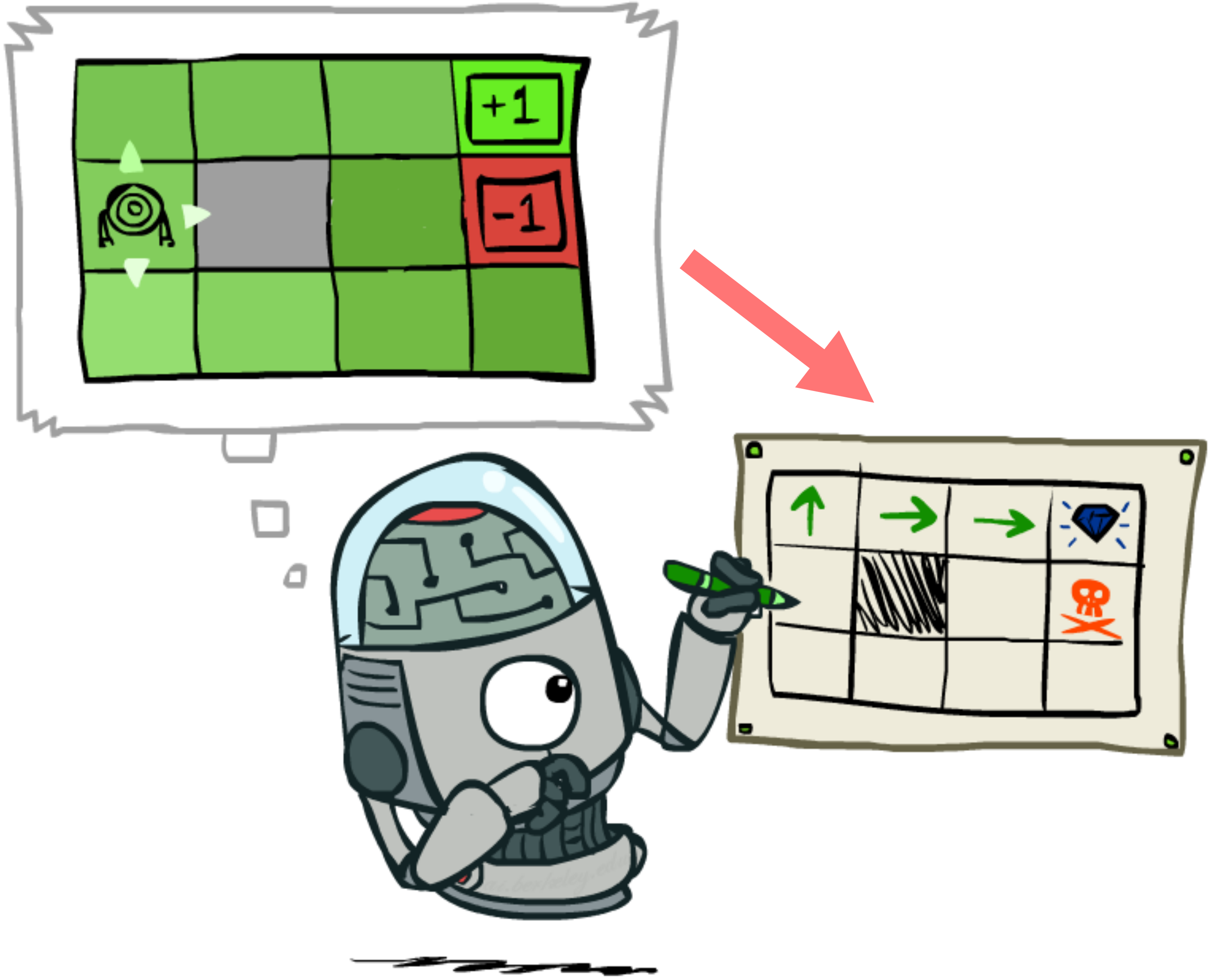
- Leads to *Q-Value iteration* algorithm we'll see next week



The Bellman Equations

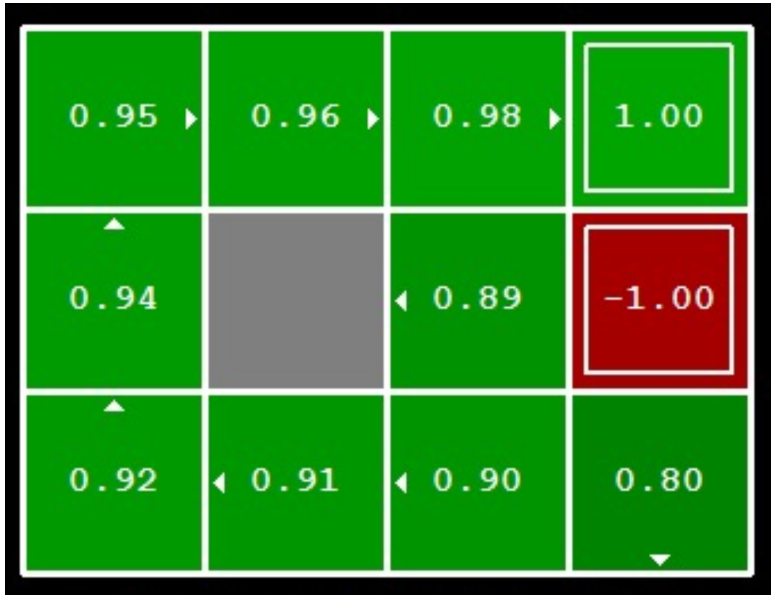


Policy Extraction



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)

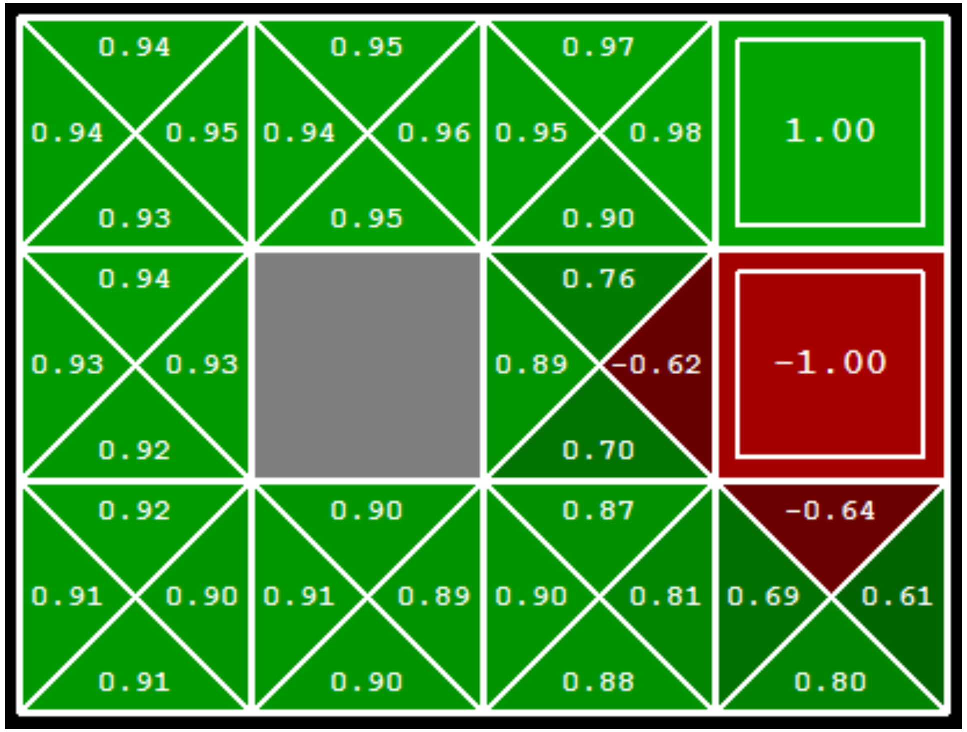
$$\pi^*(s) = \arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

ex:
max [0.5, 1.7, 1.2] = **1.7**
argmax [0.5, 1.7, 1.2] = **1**

- 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!

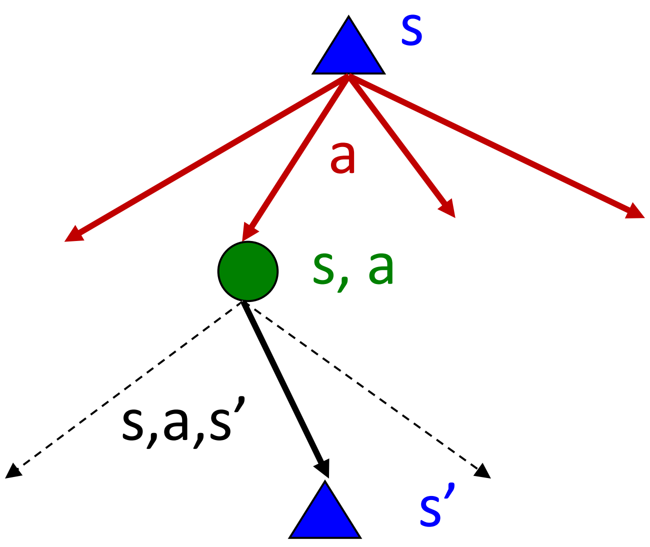
$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

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

Problems with Value Iteration

- Value iteration repeats the Bellman updates:

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



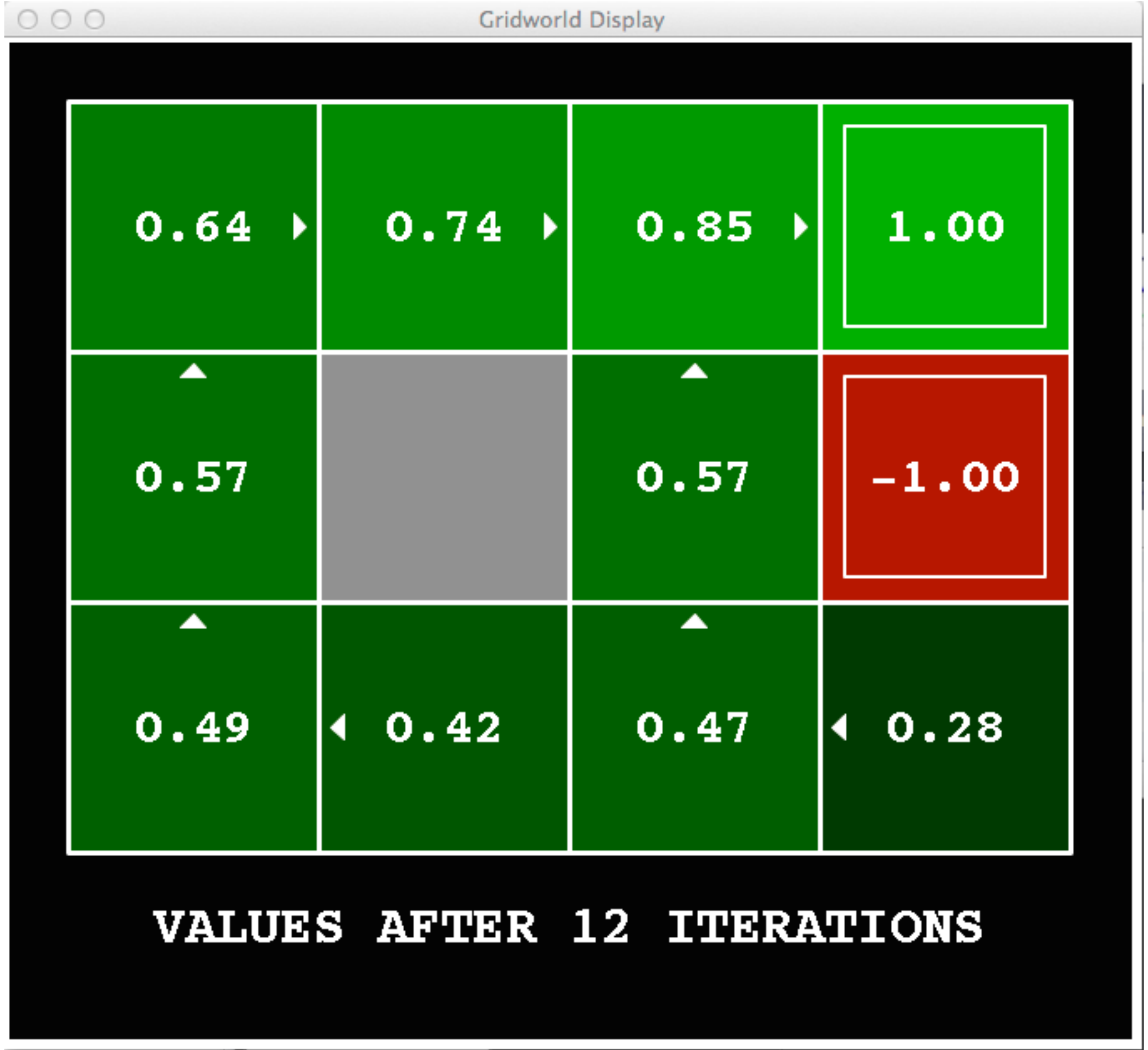
- Problem 1: It's slow – $O(S^2A)$ per iteration

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

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

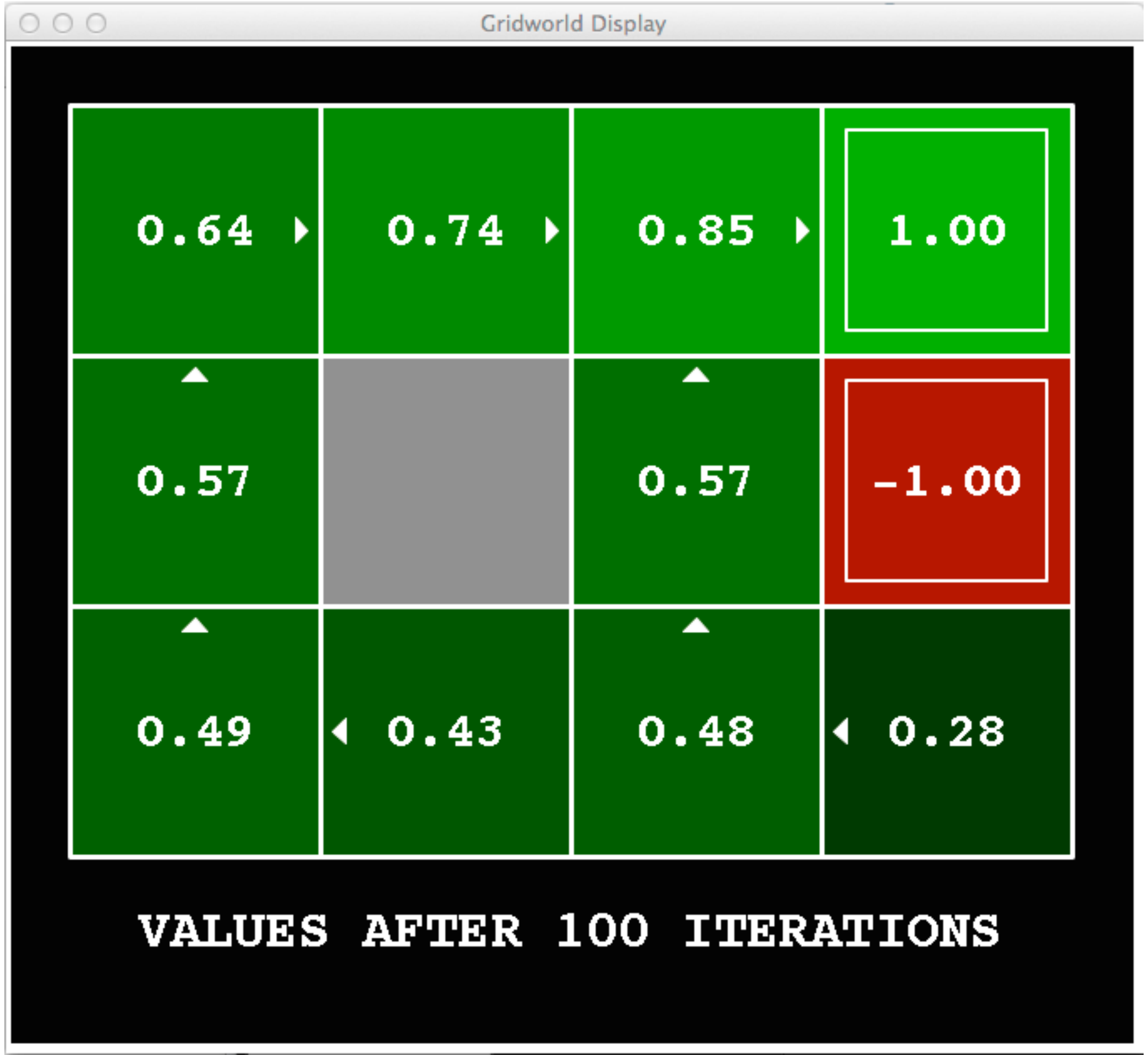
[Demo: value iteration (L9D2)]

k=12



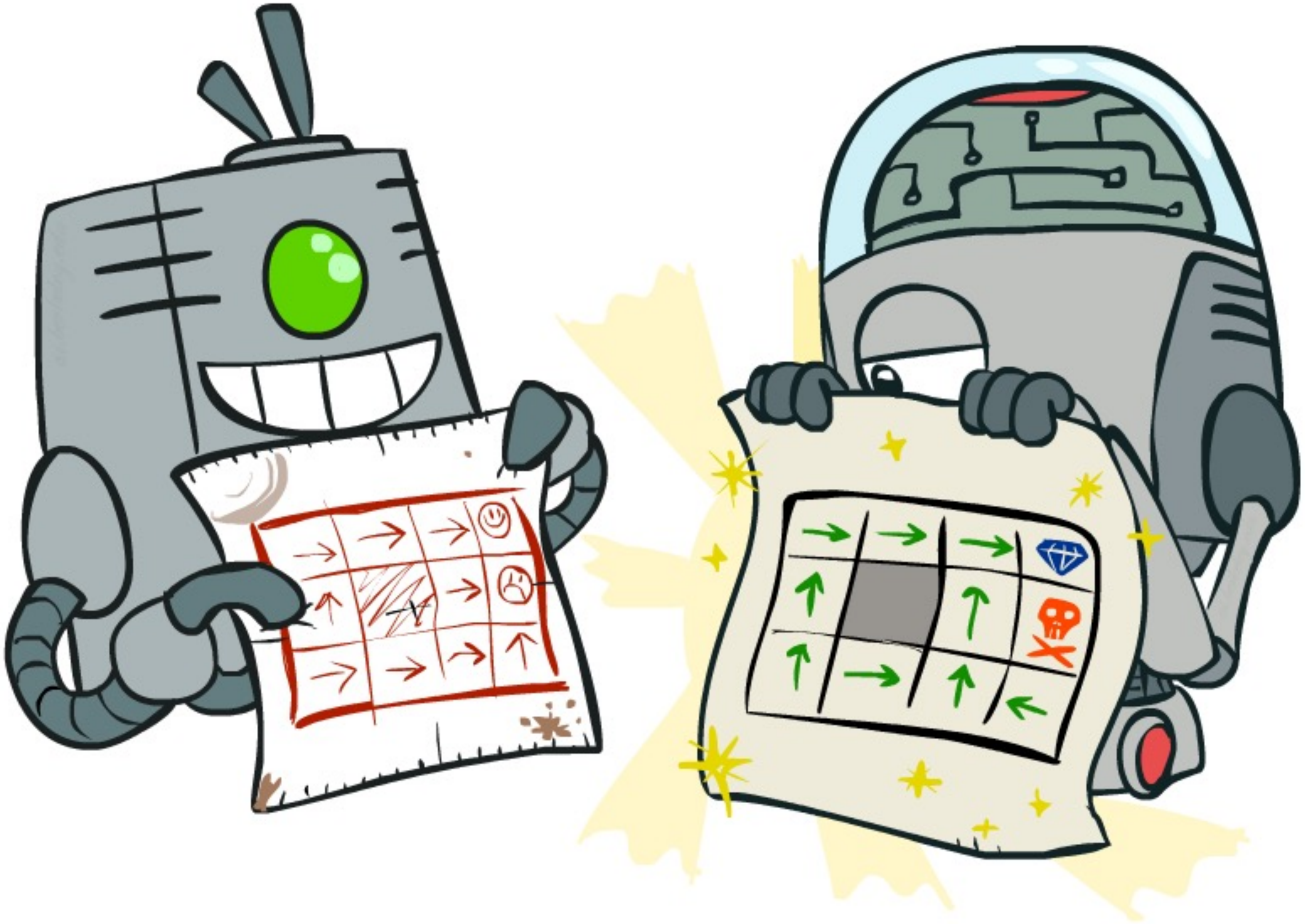
Noise = 0.2
Discount = 0.9
Living reward = 0

k=100

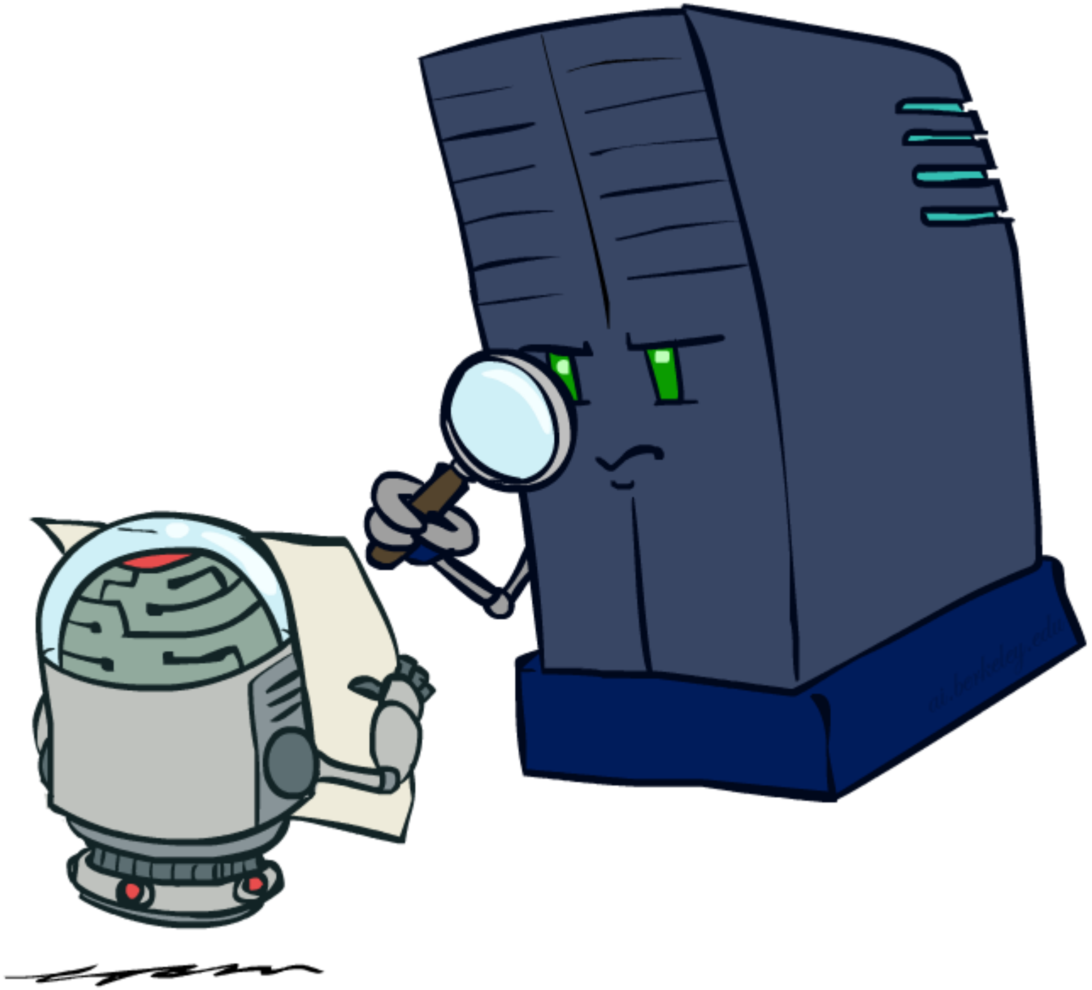


Noise = 0.2
Discount = 0.9
Living reward = 0

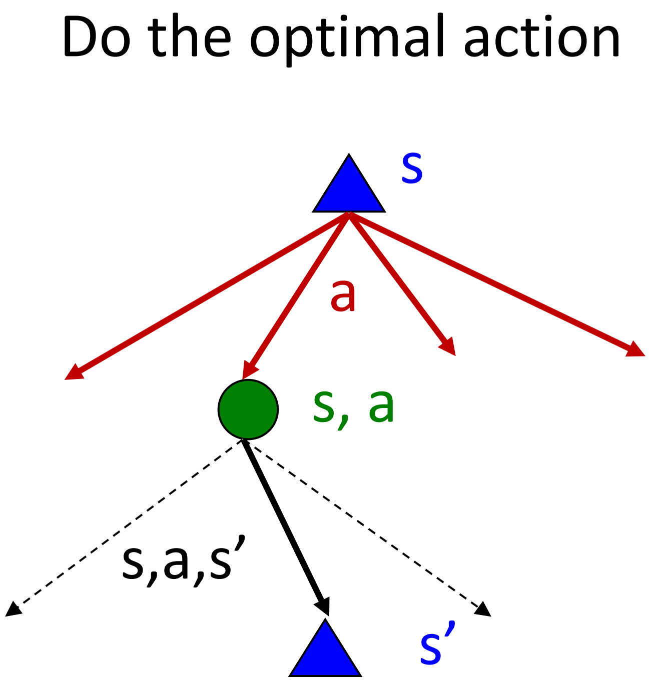
Policy Methods

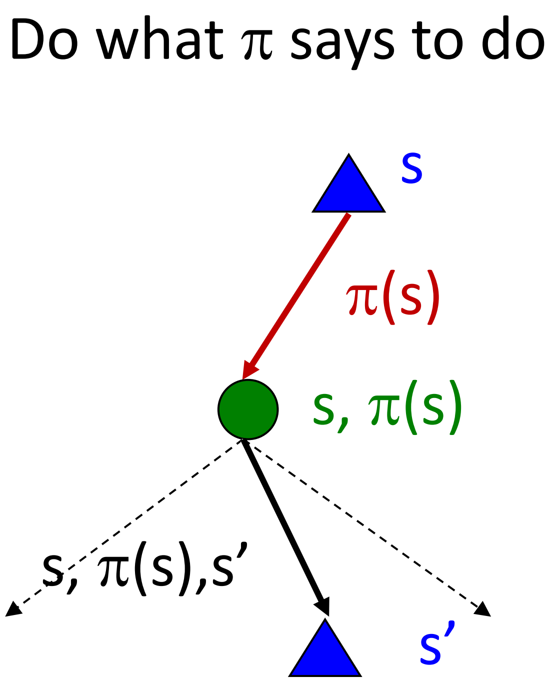


Policy Evaluation



Fixed Policies





- 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

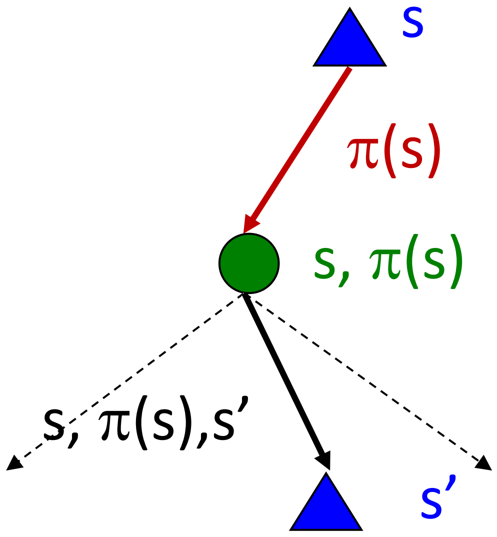
Utilities for a Fixed Policy

- 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

- What is the recursive relation (one-step look-ahead / Bellman equation)?
  - Hint: recall Bellman equation for optimal policy:

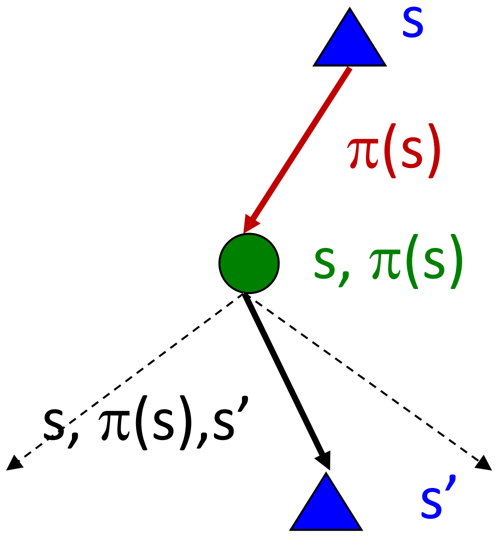
$$V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s')\right]$$



Utilities for a Fixed 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



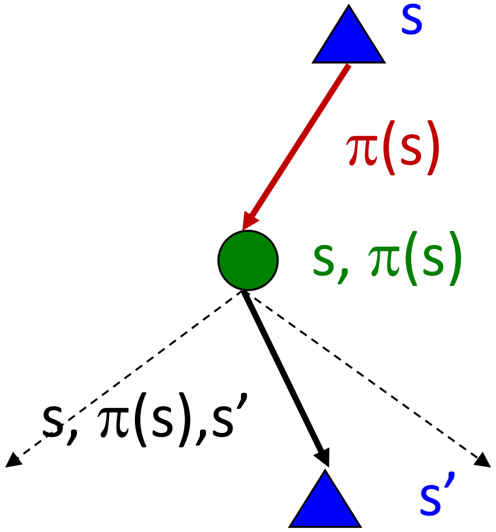
- What is the recursive relation (one-step look-ahead / Bellman equation)?
  - Hint: recall Bellman equation for optimal policy:

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

- Answer:

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

Policy Evaluation

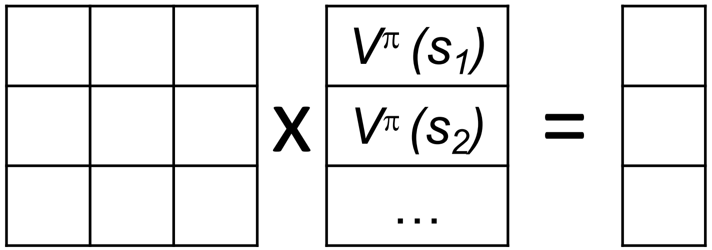


- How do we calculate the V’s for a fixed policy π?
- **Idea 1:** Turn recursive Bellman equations into updates (like value iteration)

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

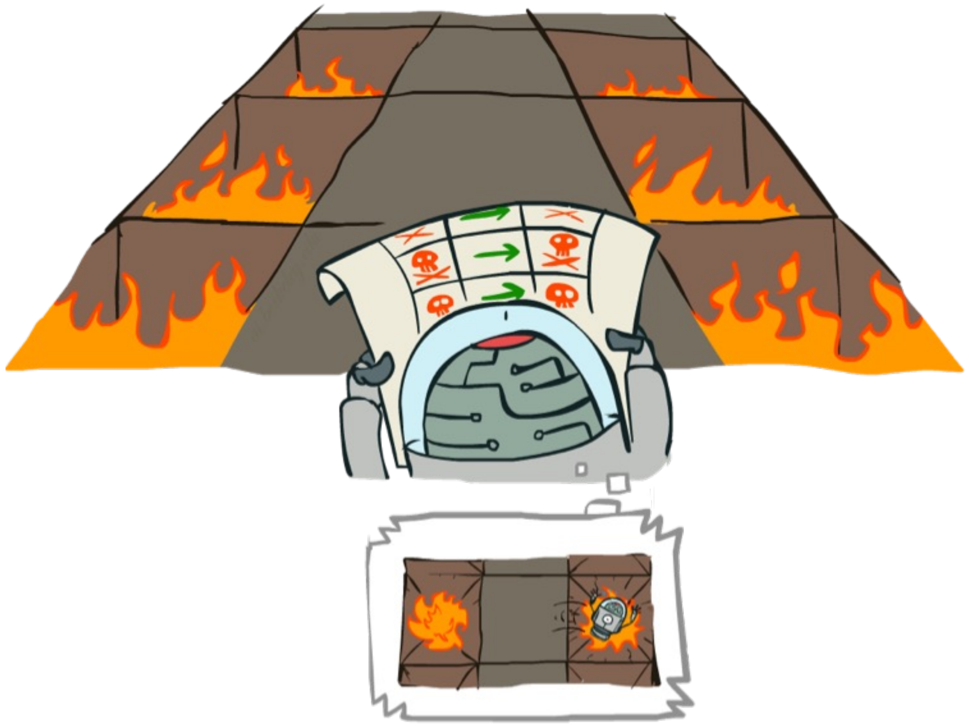
$$V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V_k^\pi(s')]$$

- Efficiency: O(S^2) per iteration
- **Idea 2:** Without the maxes, the Bellman equations are just a linear system
  - Solve with your favorite linear system solver

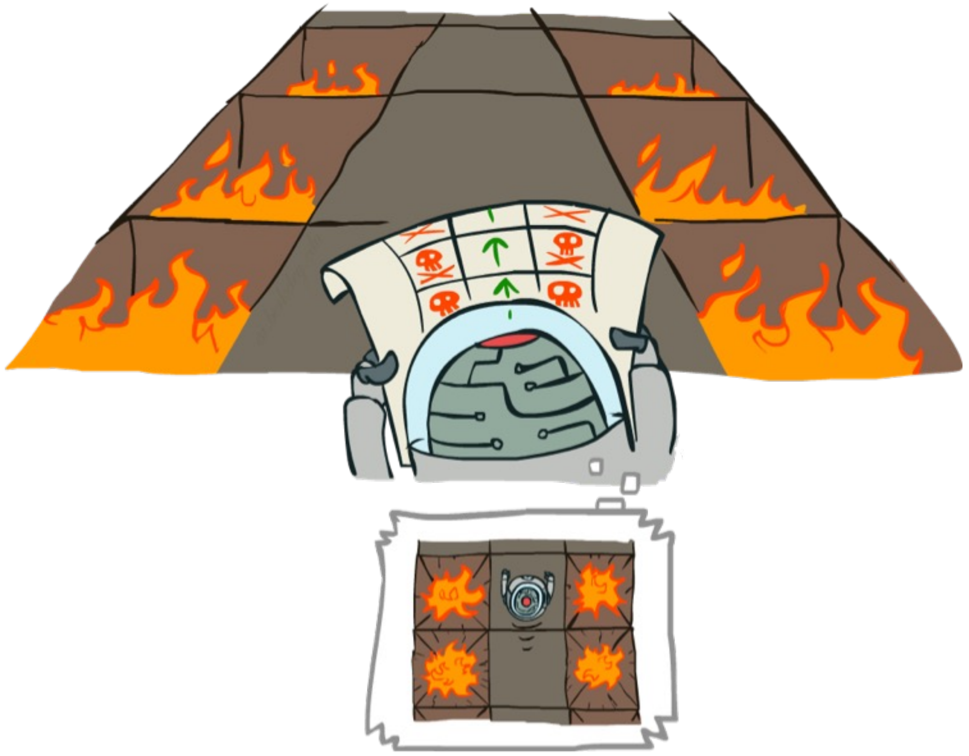


Example: Policy Evaluation

Always Go Right



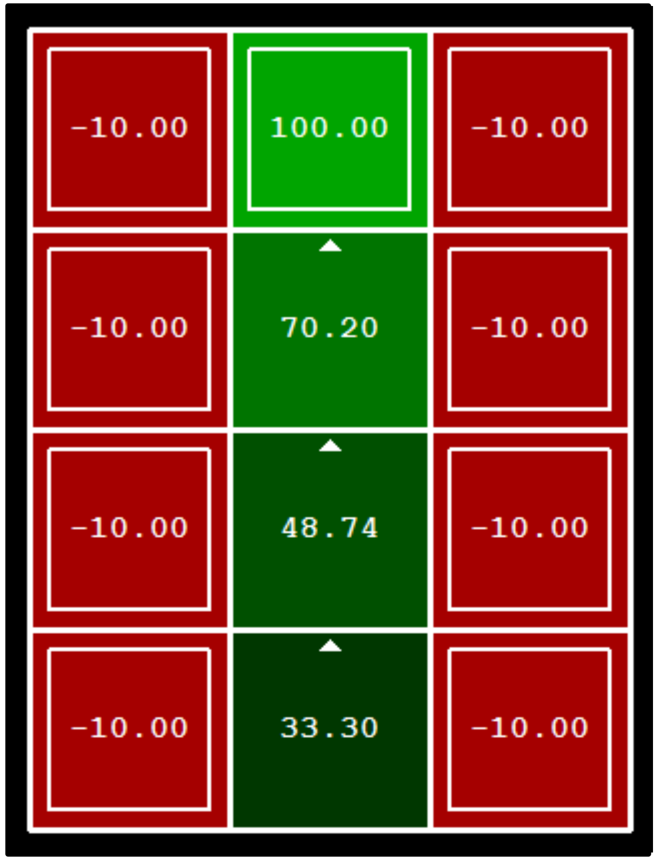
Always Go Forward



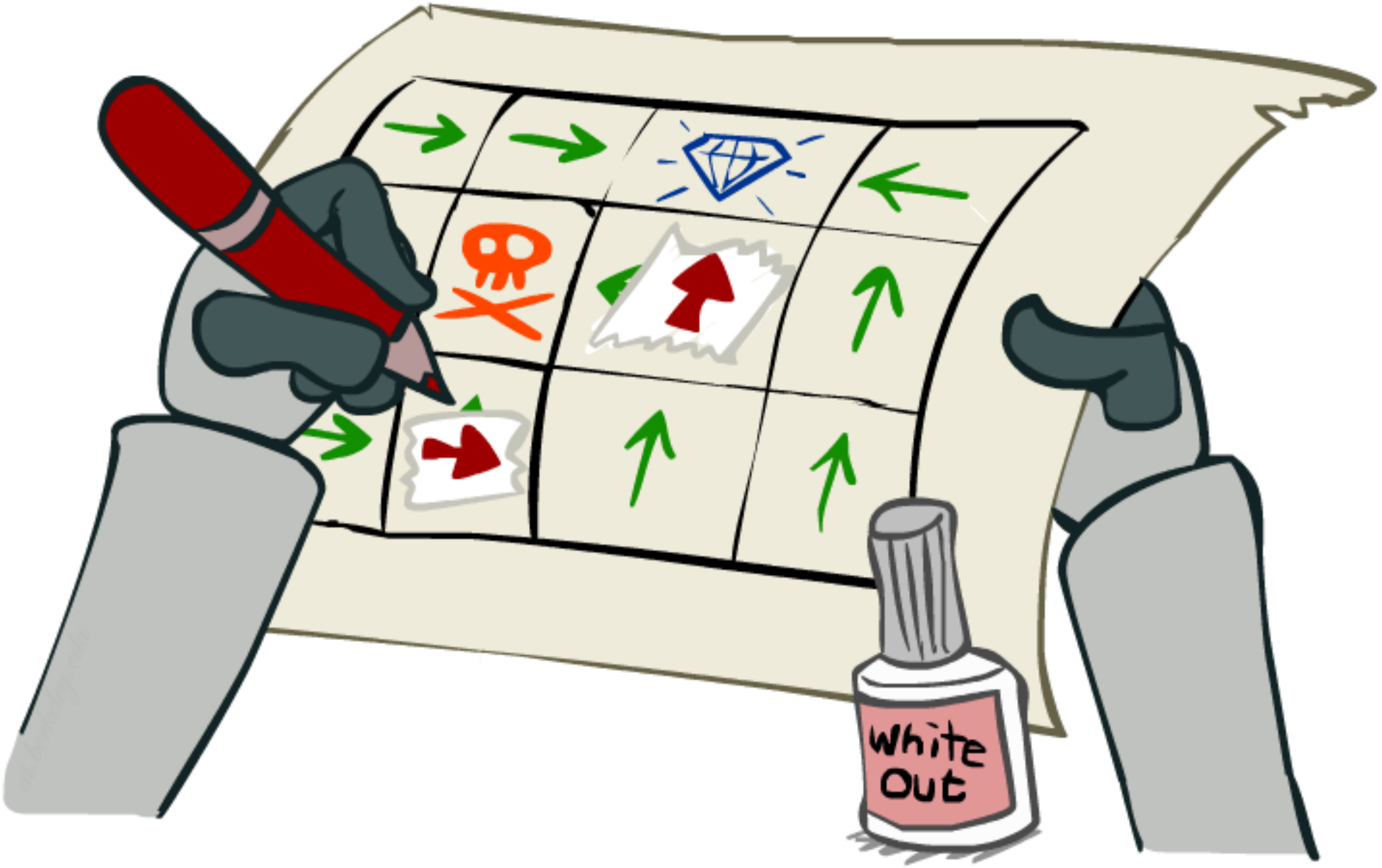
Example: Policy Evaluation

Always Go Right

Always Go Forward



Policy Iteration



Policy Iteration

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

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

  - End up with value function V^{π_i}
- Improvement: For fixed values, get a better policy using policy extraction
  - One-step look-ahead:

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

- Repeat steps until policy converges

Comparison

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

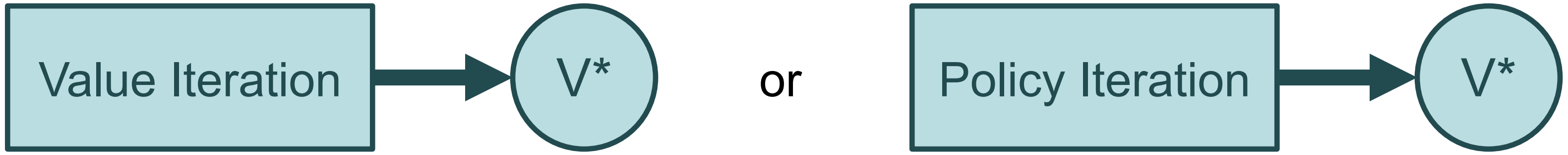
- In value iteration:
  - Every iteration updates both the values and (implicitly) the policy
  - We don't track the policy, but taking the max over actions implicitly recomputes it

- 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)

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

Summary: Bellman Equation Zoo!

- Optimal V and Q value functions:

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

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

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

- Value function for fixed policy π:

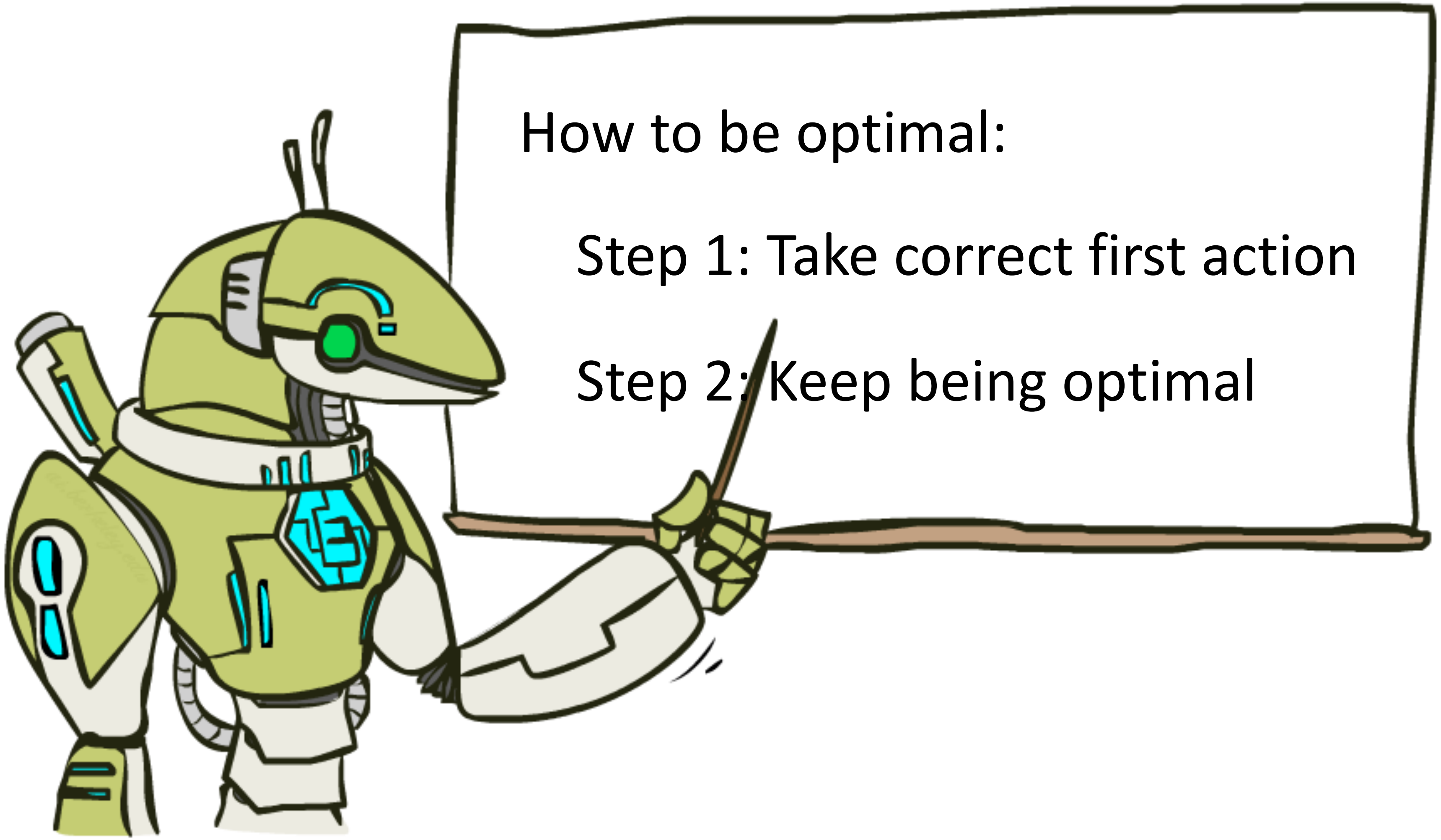
$$V^\pi(s) = \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V^\pi(s')]$$

- Policy π for V and Q value functions:

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

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

The Bellman Equations



"Journey of a thousand optimal steps begins with a first optimal step"

Next Time: Reinforcement Learning!

Extra Time: Convergence*

(won't be on exams or homeworks)

- How do we know the V_k vectors are going to converge?
- Proof sketch (assuming discount $0<\gamma<1$):
  - 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