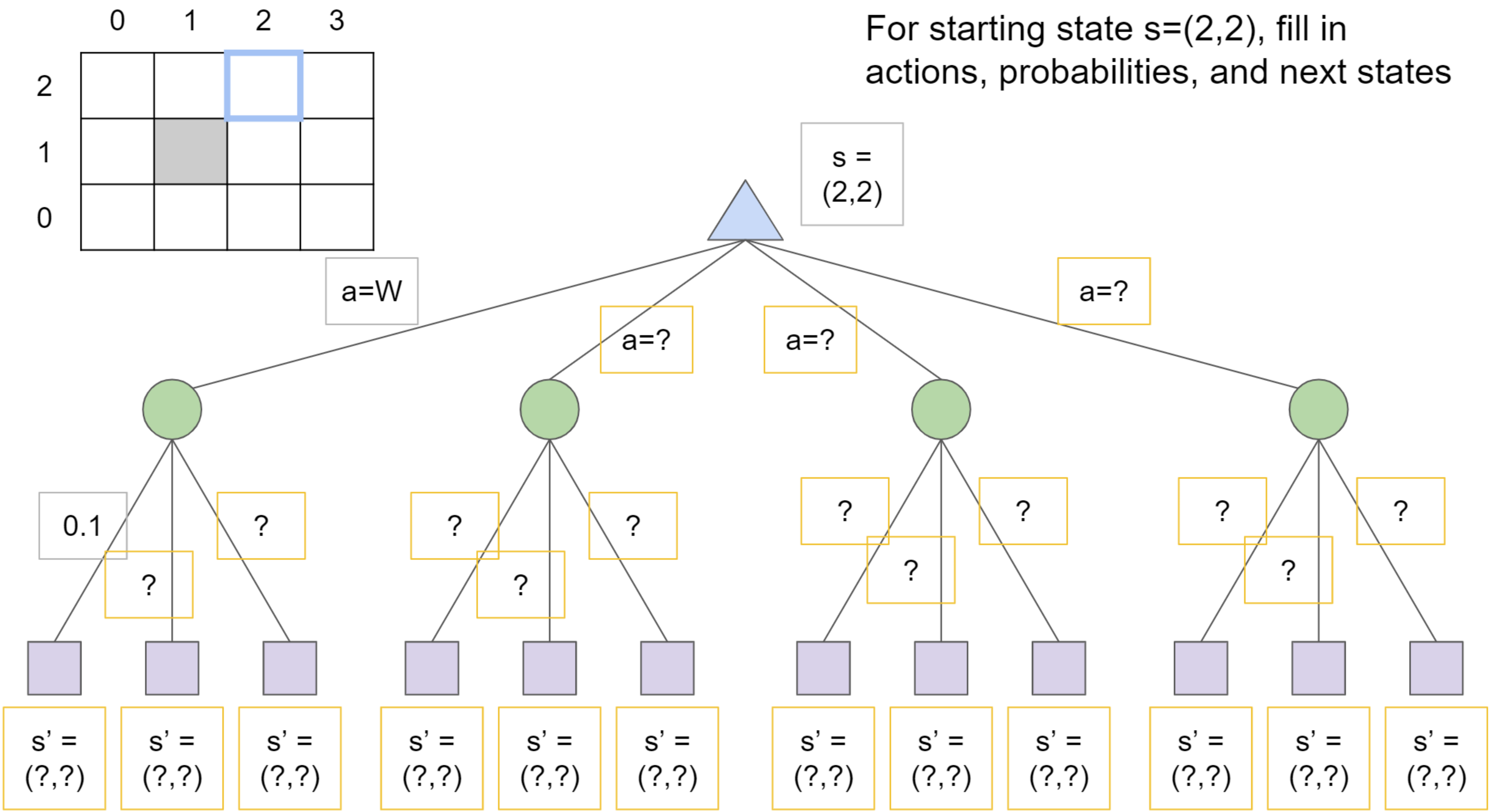
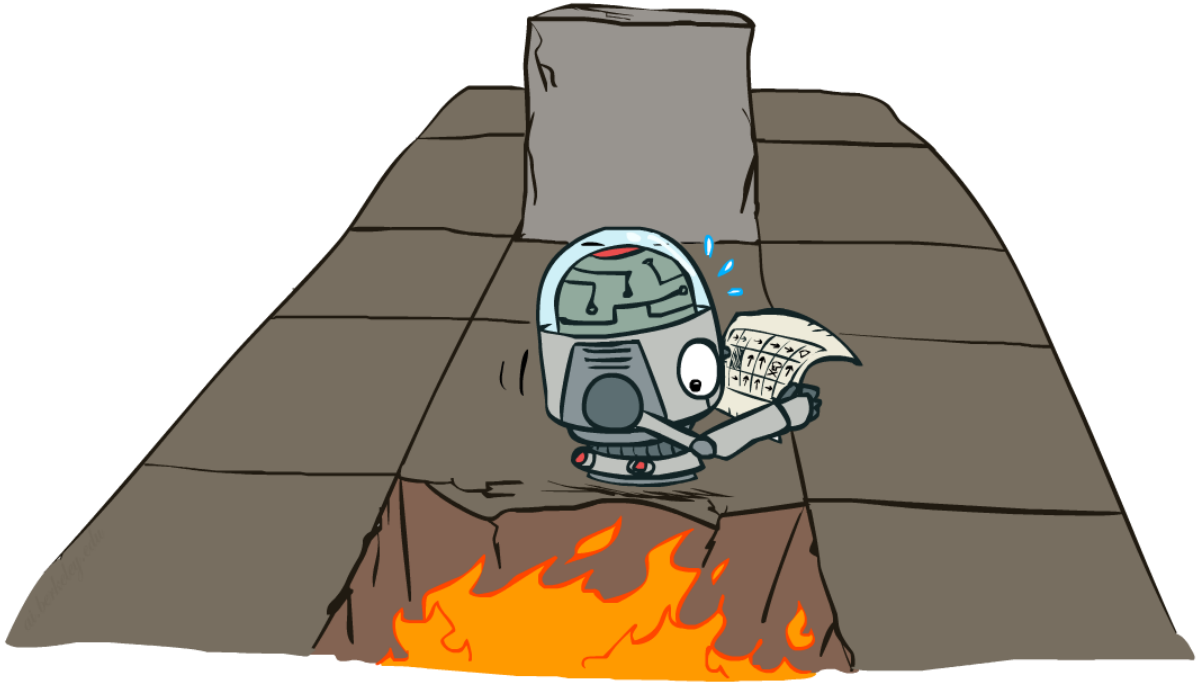
Warm-up as you walk in: Grid World

For starting state s=(2,2), fill in actions, probabilities, and next states



AI: Representation and Problem Solving

Markov Decision Processes II



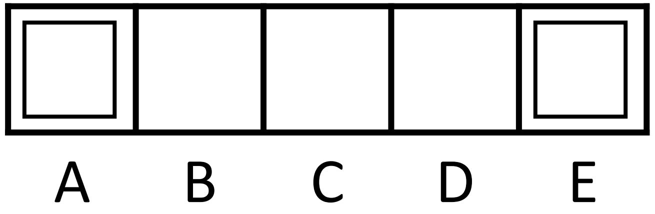
Instructor: Pat Virtue

Slide credits: CMU AI and http://ai.berkeley.edu

Outline

MDP Setup

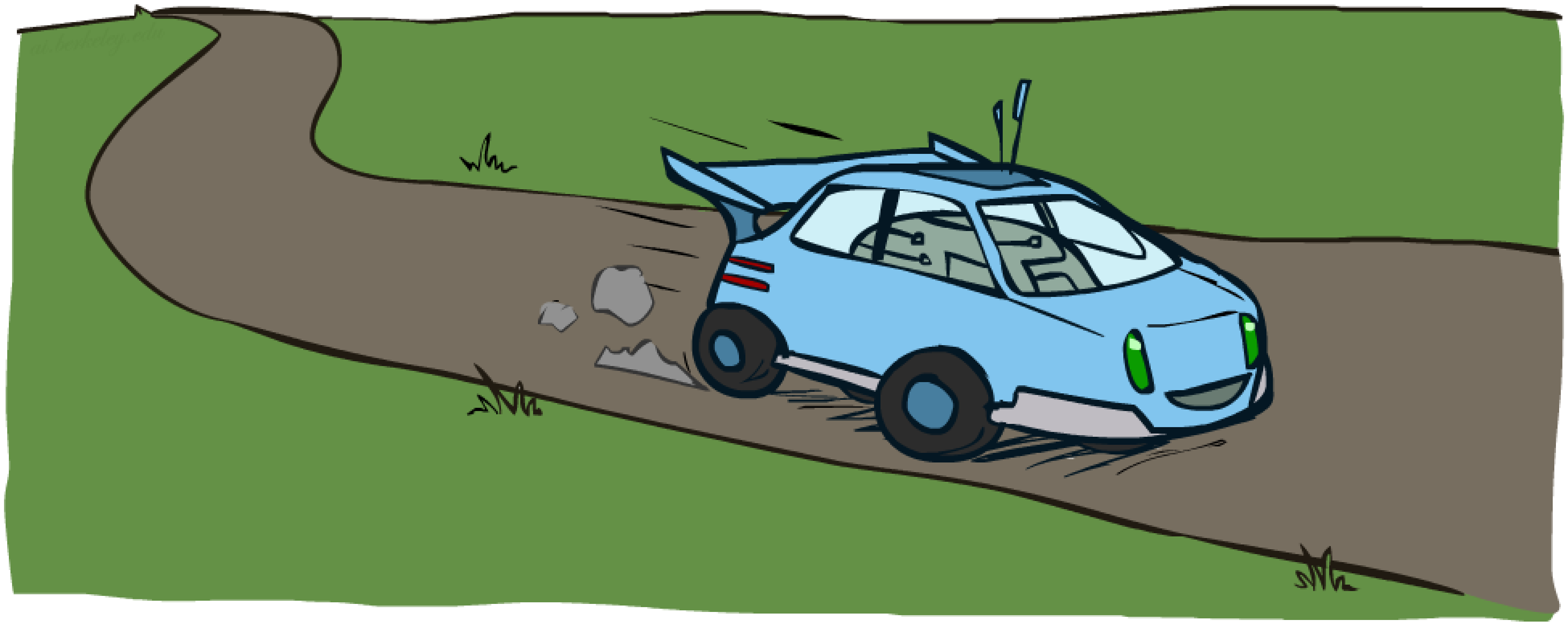
- Expectimax: State, actions, non-deterministic transition functions
- Rewards
  - Walk-through of super-simple value iteration
- Discounting, γ



Solving MDPs

- Method 1) Value iteration
  - Value iteration convergence
- Bellman equations
- Policy Extraction
- Method 2) Policy Iteration

MDP Example: Racing



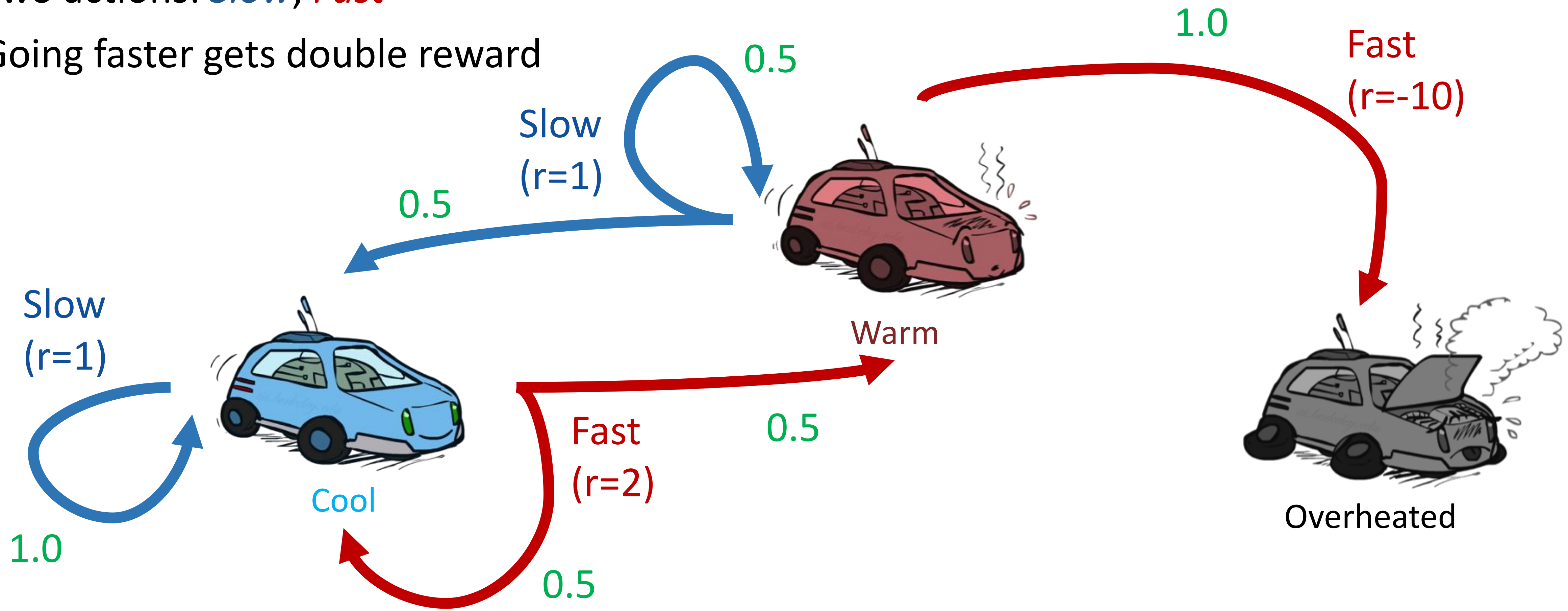
MDP Example: Racing

A robot car wants to travel far, quickly

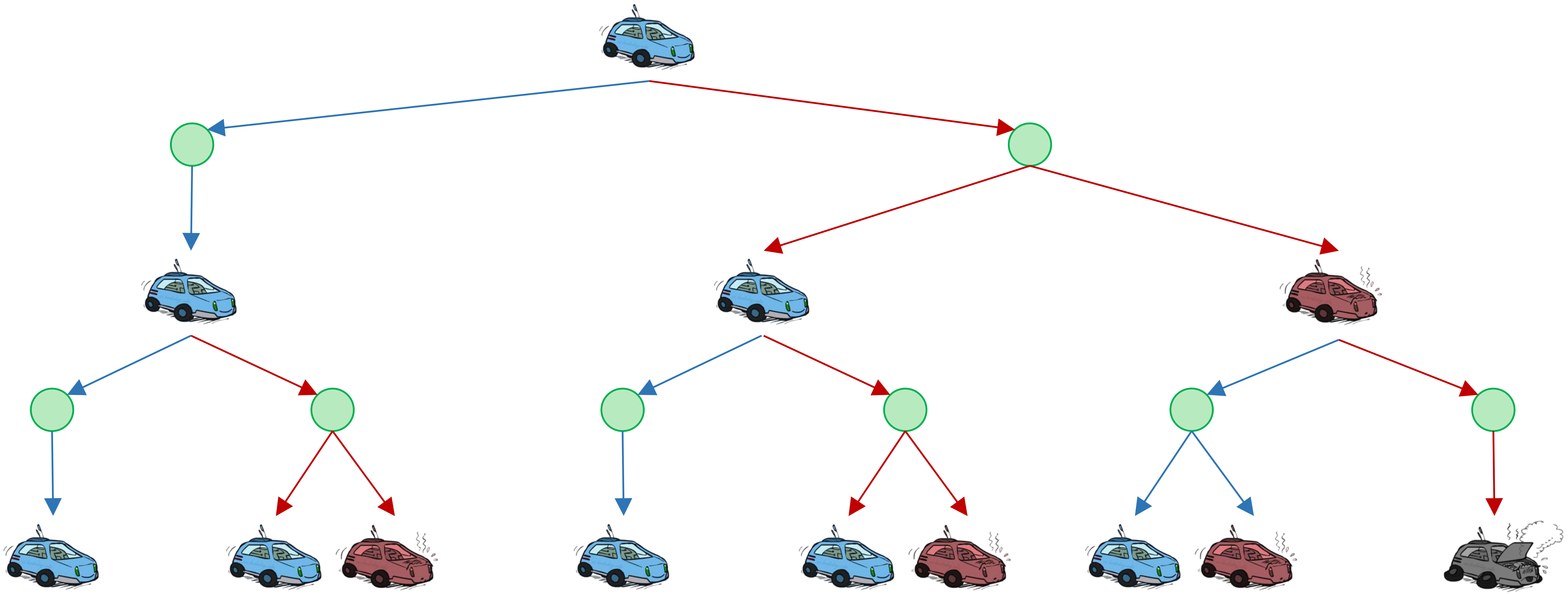
Three states: Cool, Warm, Overheated

Two actions: *Slow*, *Fast*

Going faster gets double reward



Racing Search Tree

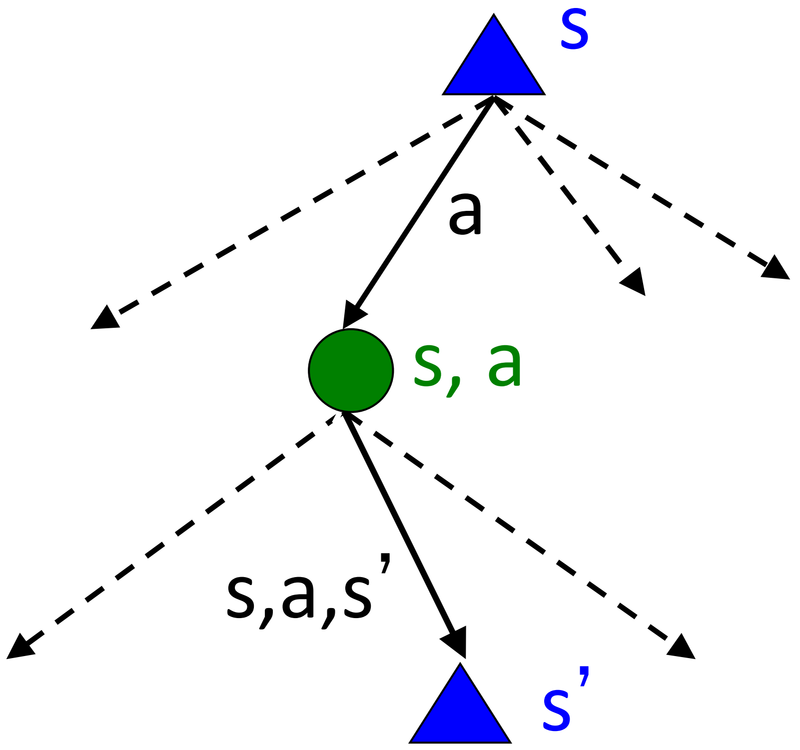


Recursive Expectimax

$$V(s) = \max_a \sum_{s'} P(s'|s,a)\, V(s')$$

Now with rewards:

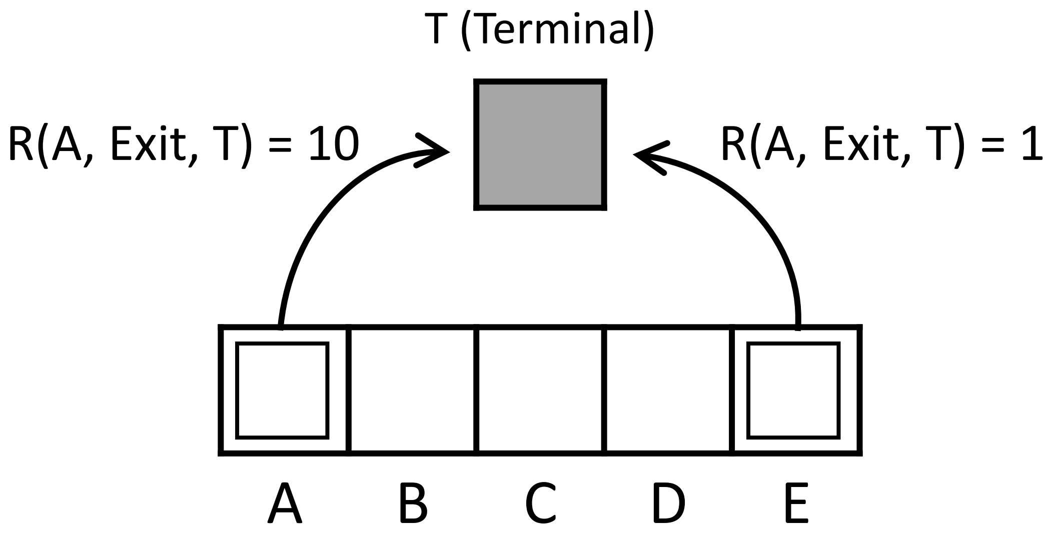
$$V(s) = \max_a \sum_{s'} P(s'|s,a)\, [R(s,a,s') + V(s')]$$



Simple Deterministic Example

- Actions: B, C, D: East, West
- Actions: A, E: Exit
- Transitions: deterministic
- Rewards only for transitioning to terminal state

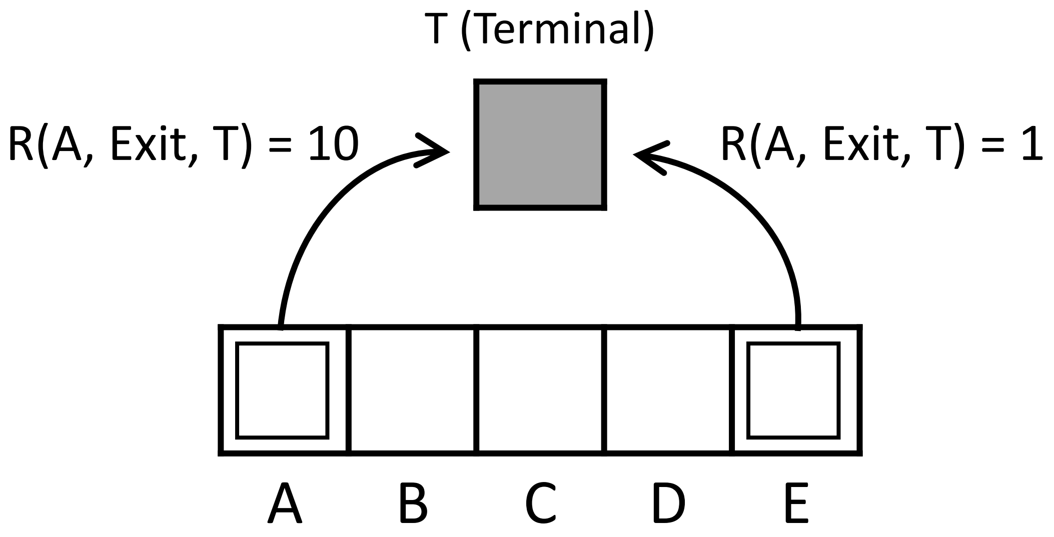
$$V(s) = \max_a [R(s,a,s') + V(s')]$$



Simple Deterministic Example

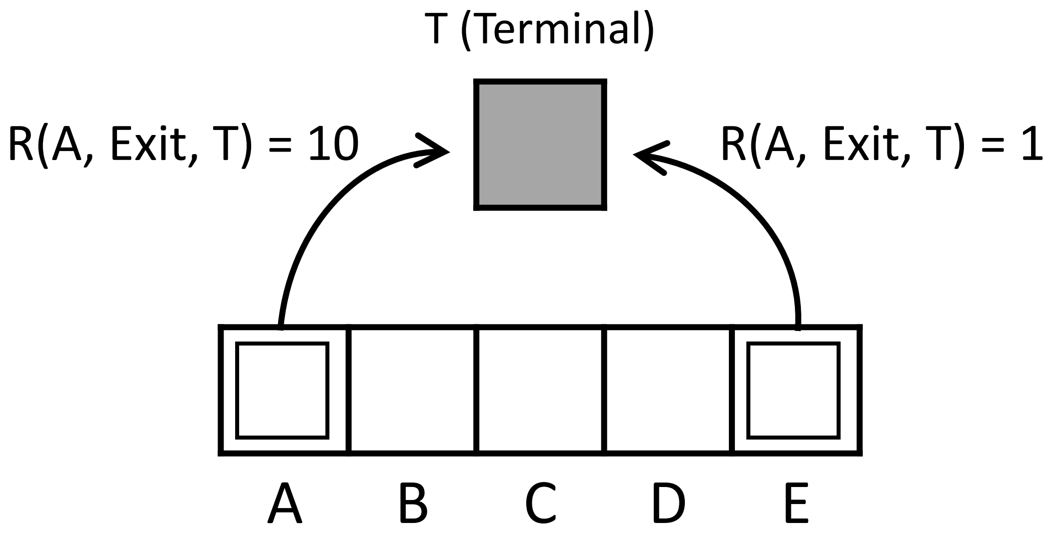
- Actions: B, C, D: East, West
- Actions: A, E: Exit
- Transitions: deterministic
- Rewards only for transitioning to terminal state

$$V_{k+1}(s) = \max_a [R(s, a, s') + V_k(s')]$$



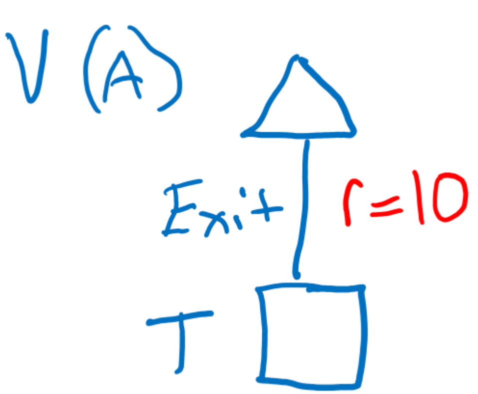
Simple Deterministic Example

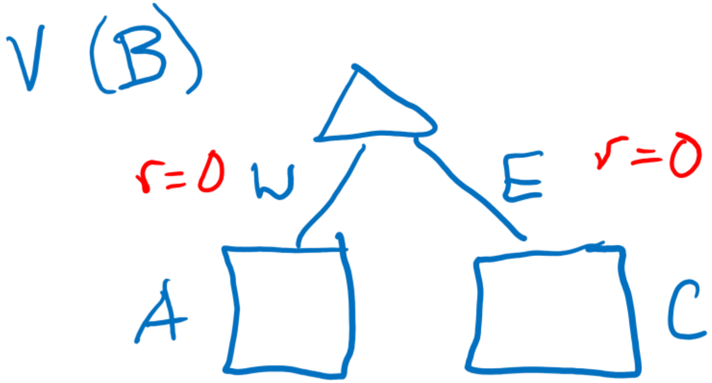
- Actions: B, C, D: East, West
- Actions: A, E: Exit
- Transitions: deterministic
- Rewards only for transitioning to terminal state



$$V_{k+1}(s) = \max_a [R(s,a,s') + V_k(s')]$$

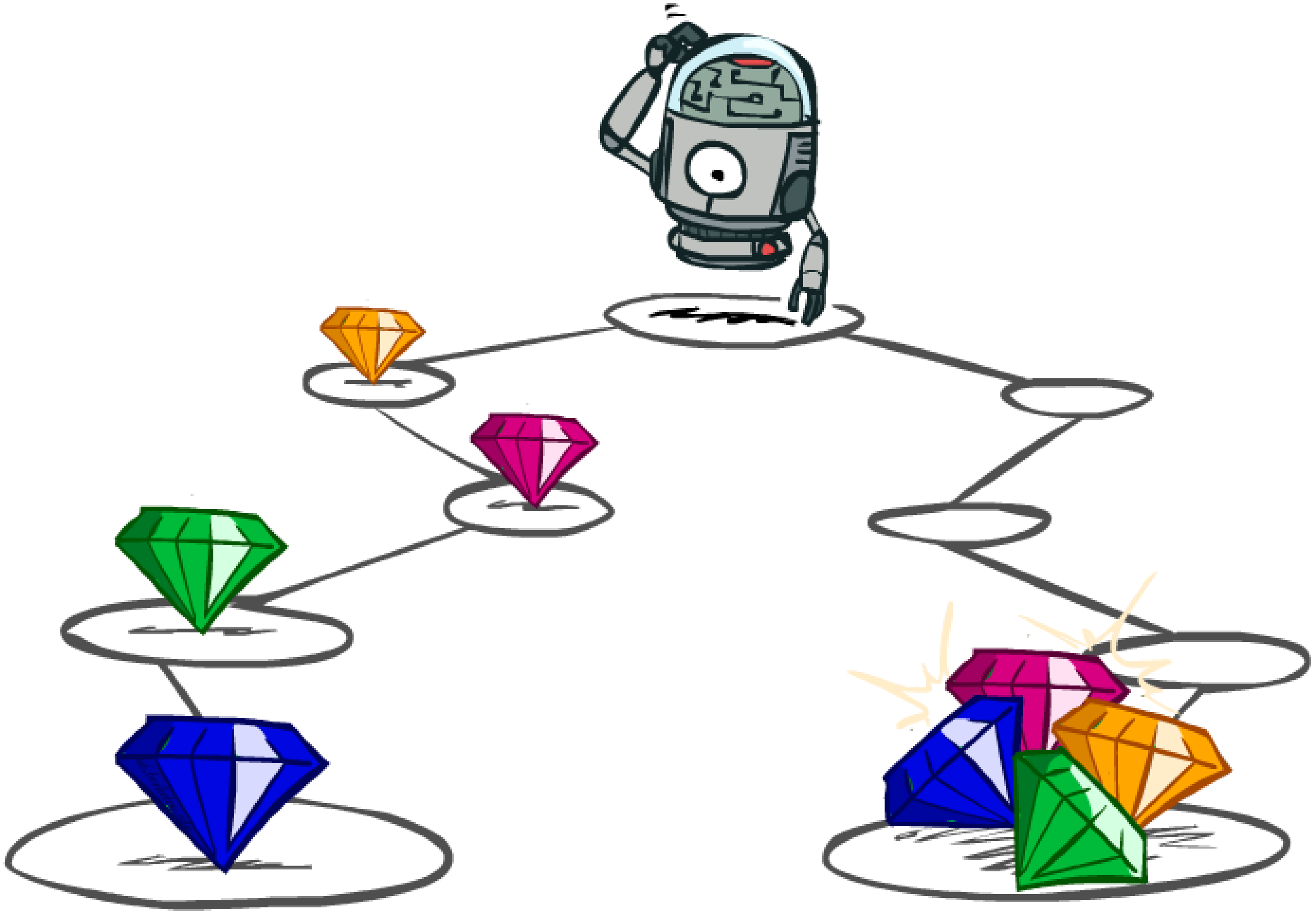
$V_0(s) = 0 \quad \forall s$





	T	A	B	C	D	E
V_0	0	0	0	0	0	0
V_1	0	10	0	0	0	1
V_2	0					
V_3						
V_4						

Utilities of Sequences

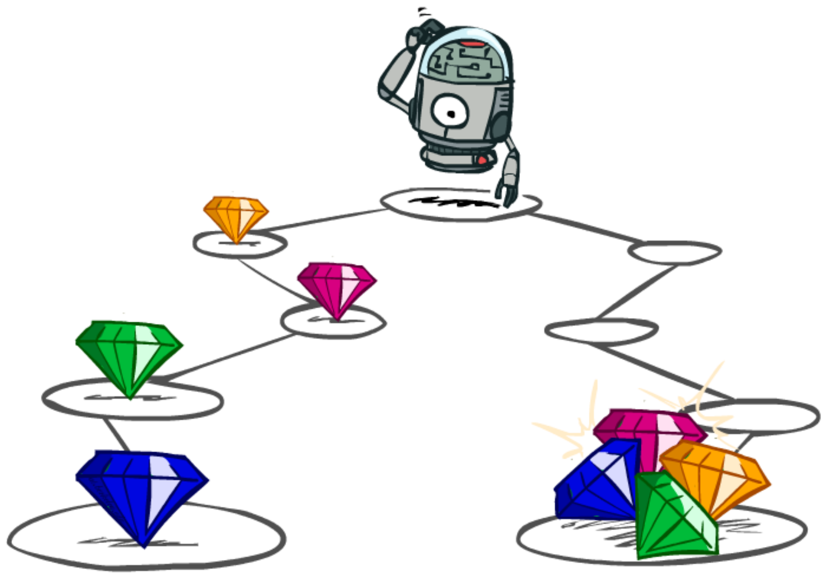


Utilities of Sequences

What preferences should an agent have over reward sequences?

More or less? [1, 2, 2] or [2, 3, 4]

Now or later? [0, 0, 1] or [1, 0, 0]



Discounting

It’s reasonable to maximize the sum of rewards

It’s also reasonable to prefer rewards now to rewards later

One solution: values of rewards decay exponentially

1

Worth Now

γ

Worth Next Step

γ^2

Worth In Two Steps

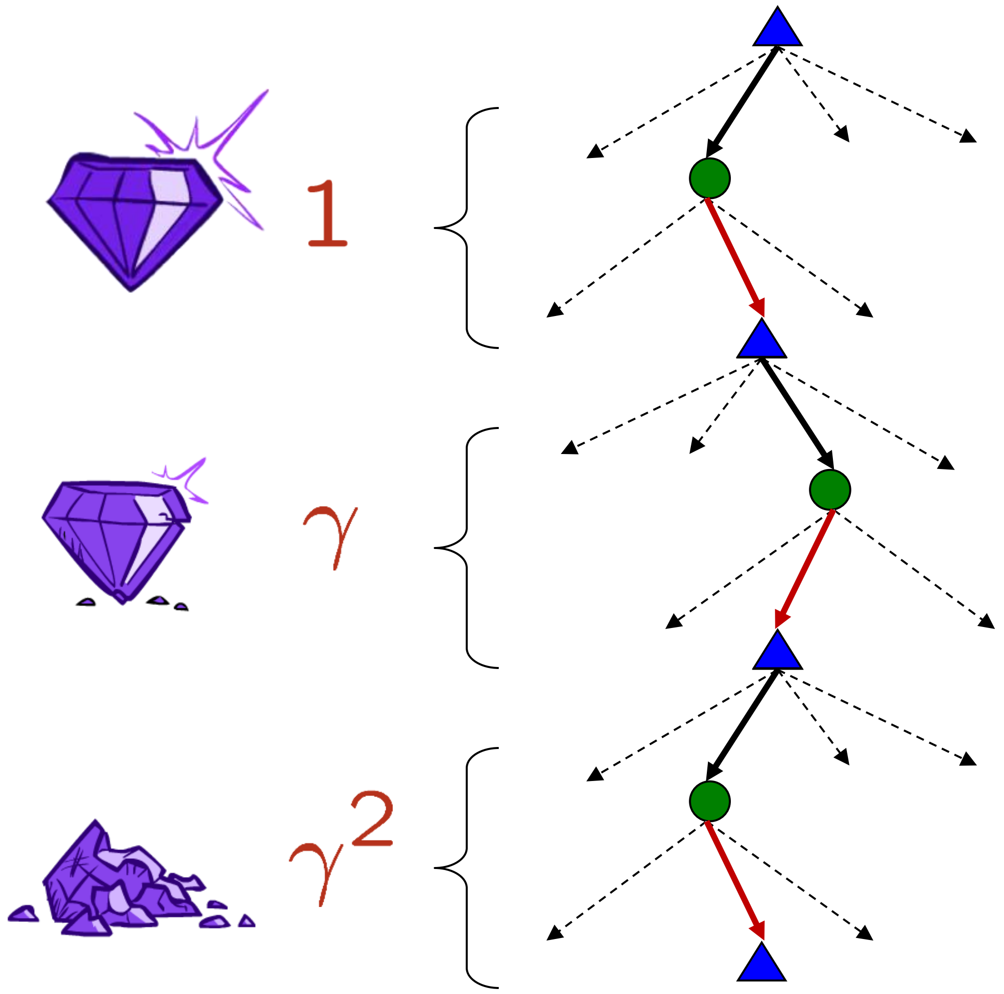
Discounting

How to discount?

- Each time we descend a level, we multiply in the discount once

Why discount?

- Sooner rewards probably do have higher utility than later rewards
- Also helps our algorithms converge
- Important: use $0 < \gamma < 1$



Poll

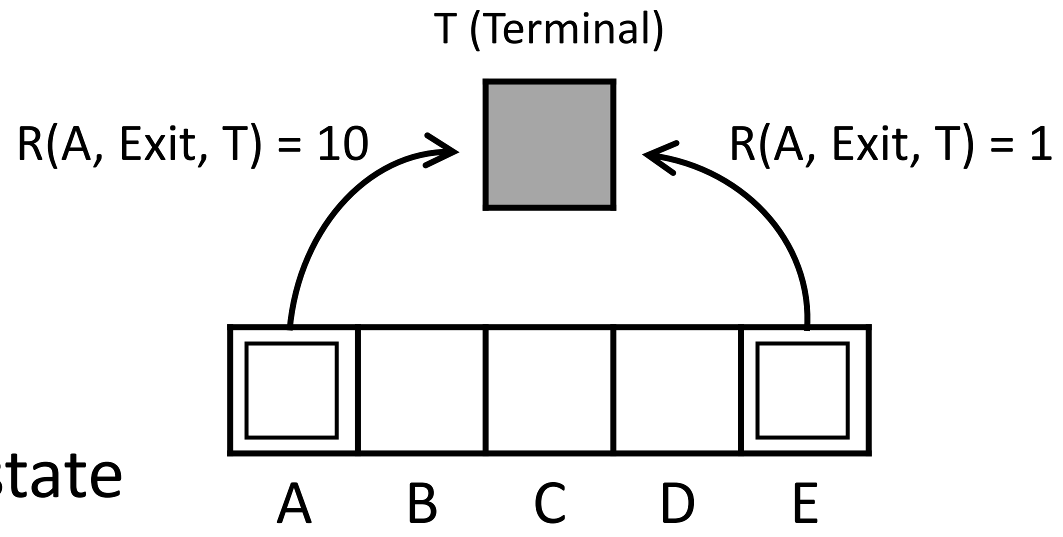
What is the value of this ordered sequence of rewards [2,4,8] with $\gamma = 0.5$?

A. 3
B. 6
C. 7
D. 14

Bonus: What is the value of [8,4,2] with $\gamma = 0.5$?

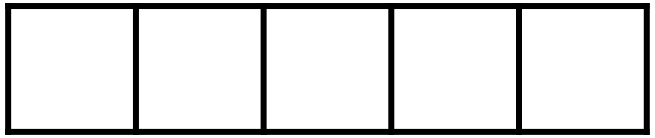
Discounting

- Actions: B, C, D: East, West
- Actions: A, E: Exit
- Transitions: deterministic
- Rewards only for transitioning to terminal state

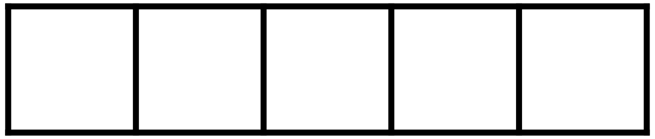


$$V_{k+1}(s) = \max_a [R(s,a,s') + \gamma\, V_k(s')]$$

For γ = 1, what is the optimal policy?



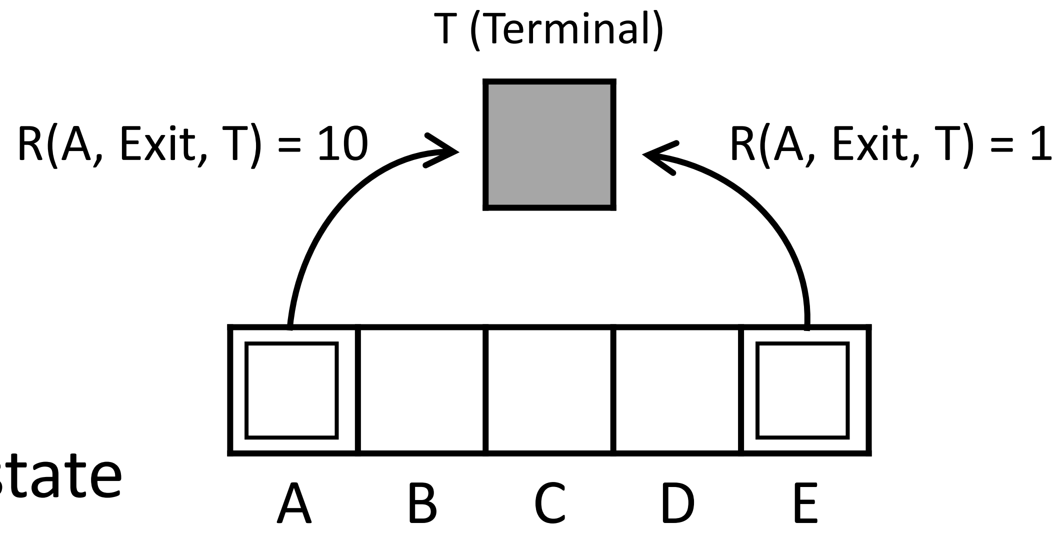
For γ = 0.1, what is the optimal policy?



For which γ are West and East equally good when in state D?

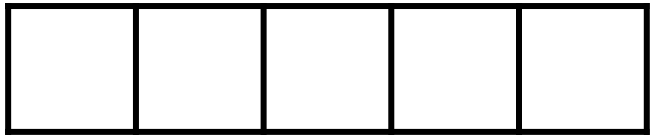
Discounting

- Actions: B, C, D: East, West
- Actions: A, E: Exit
- Transitions: deterministic
- Rewards only for transitioning to terminal state

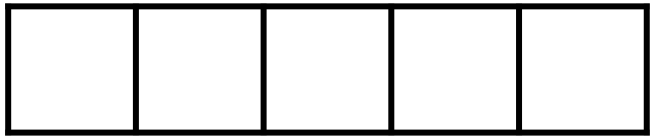


$$V_{k+1}(s) = \max_a [R(s, a, s') + \gamma \, V_k(s')]$$

For γ = 1, what is the optimal policy?

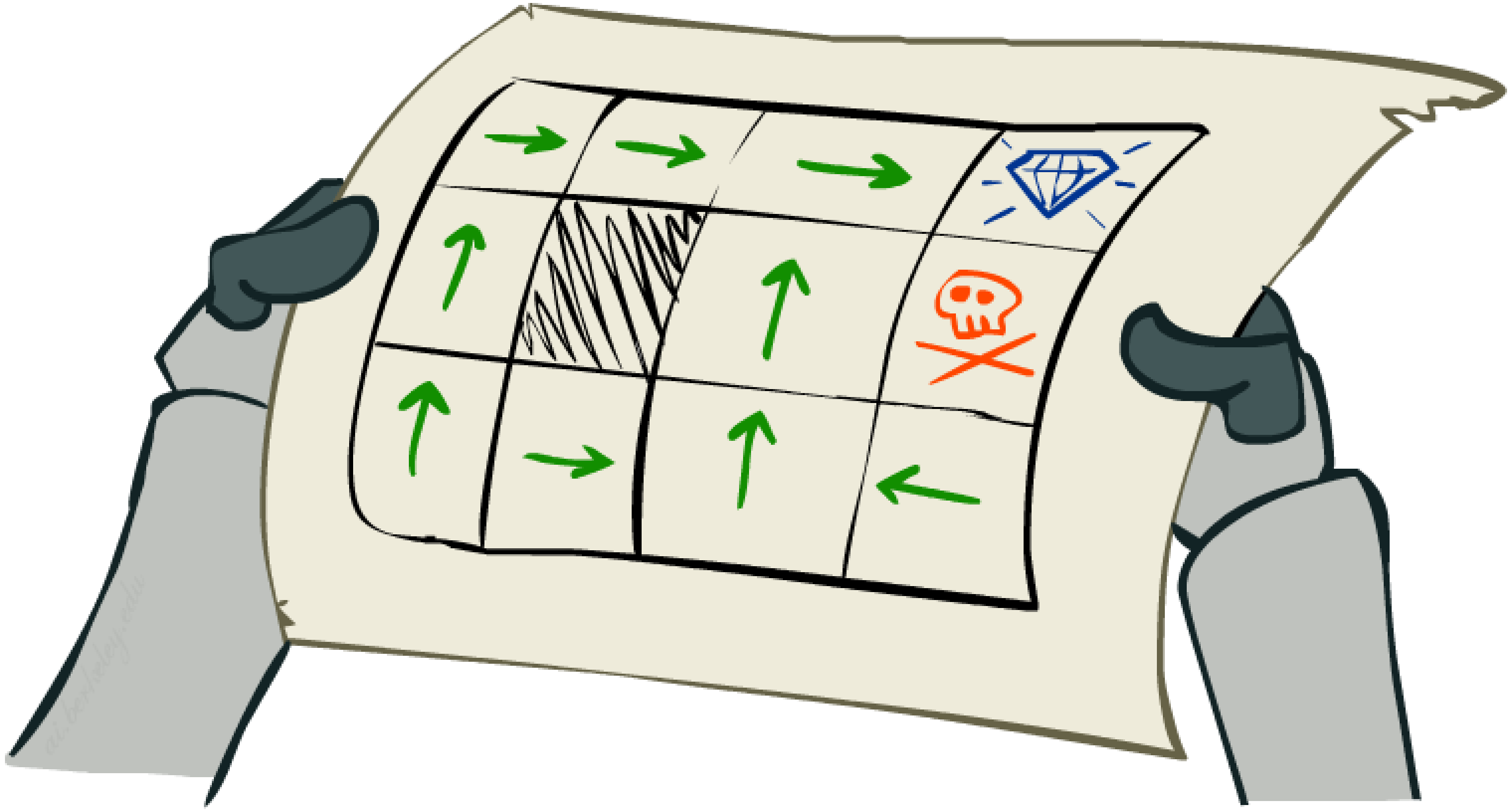


For γ = 0.1, what is the optimal policy?



For which γ are West and East equally good when in state D?

Solving MDPs



Optimal Quantities

- The value (utility) of a state s:

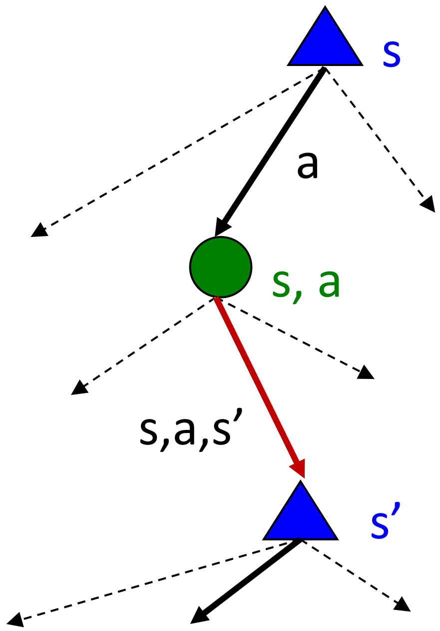
  $V^*(s)$ = expected utility starting in s and acting optimally

- 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



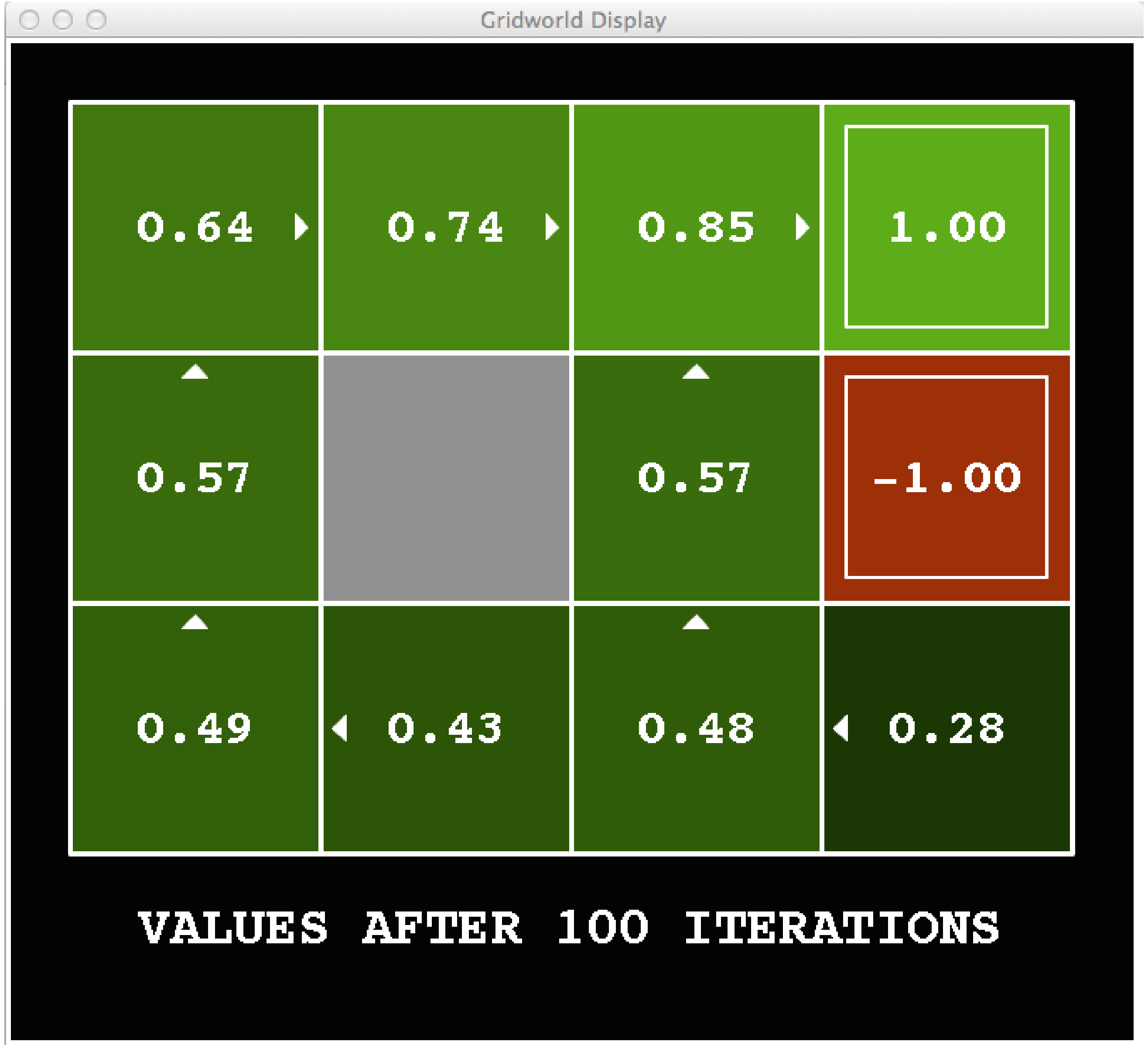
s is a *state*

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

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

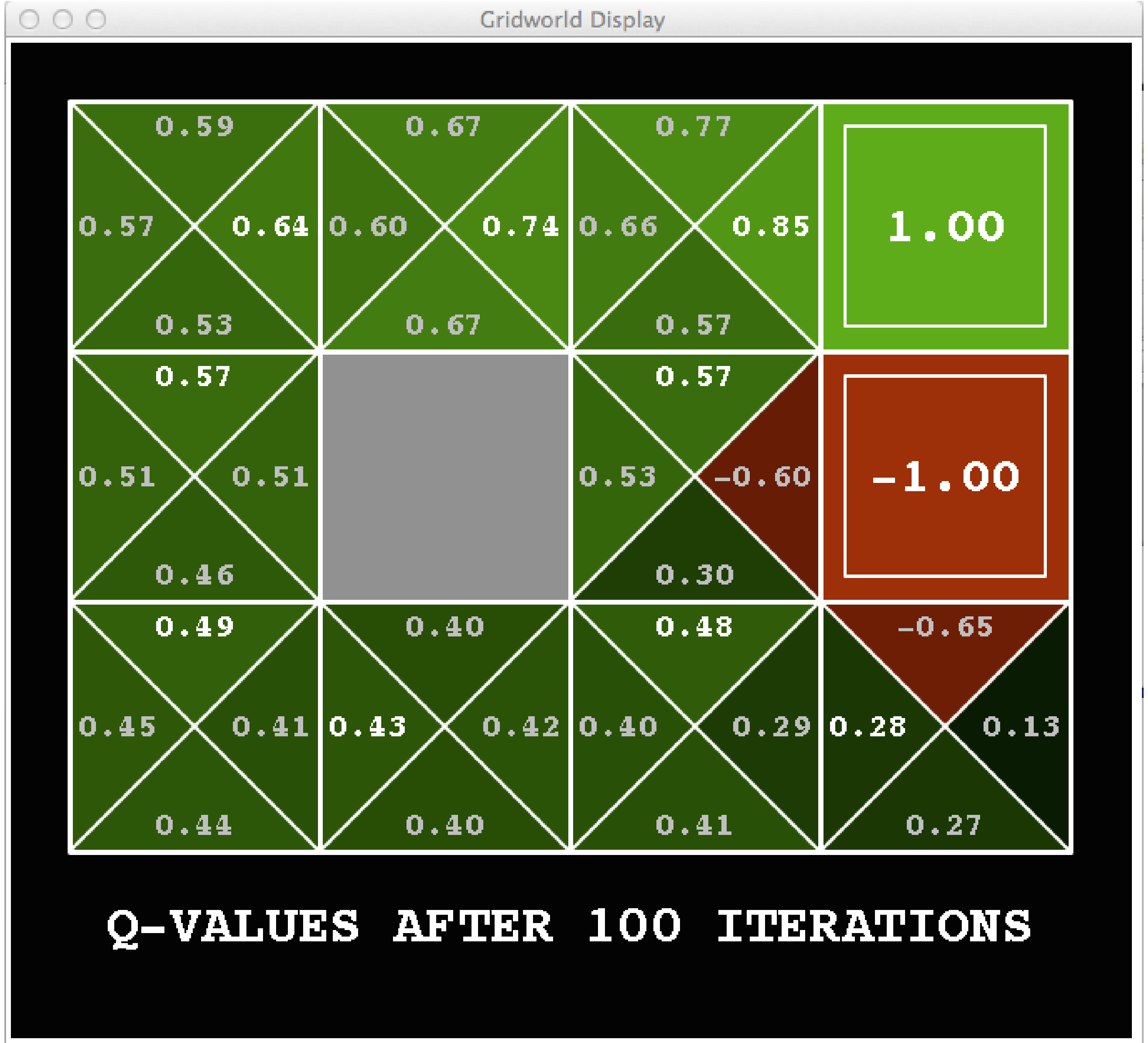
[Demo – gridworld values (L8D4)]

Snapshot of Demo – Gridworld V Values



Noise = 0.2
Discount = 0.9
Living reward = 0

Snapshot of Demo – Gridworld Q Values

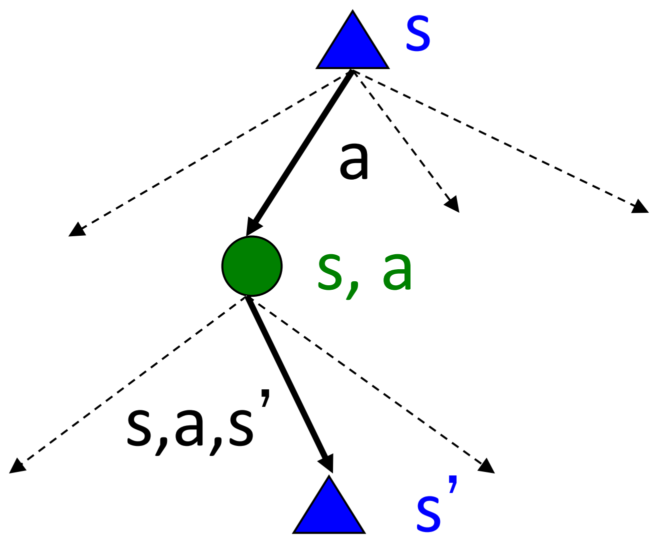


Noise = 0.2
Discount = 0.9
Living reward = 0

Values of States

Fundamental operation: compute the (expectimax) value of a state

- Expected utility under optimal action
- Average sum of (discounted) rewards
- This is just what expectimax computed!



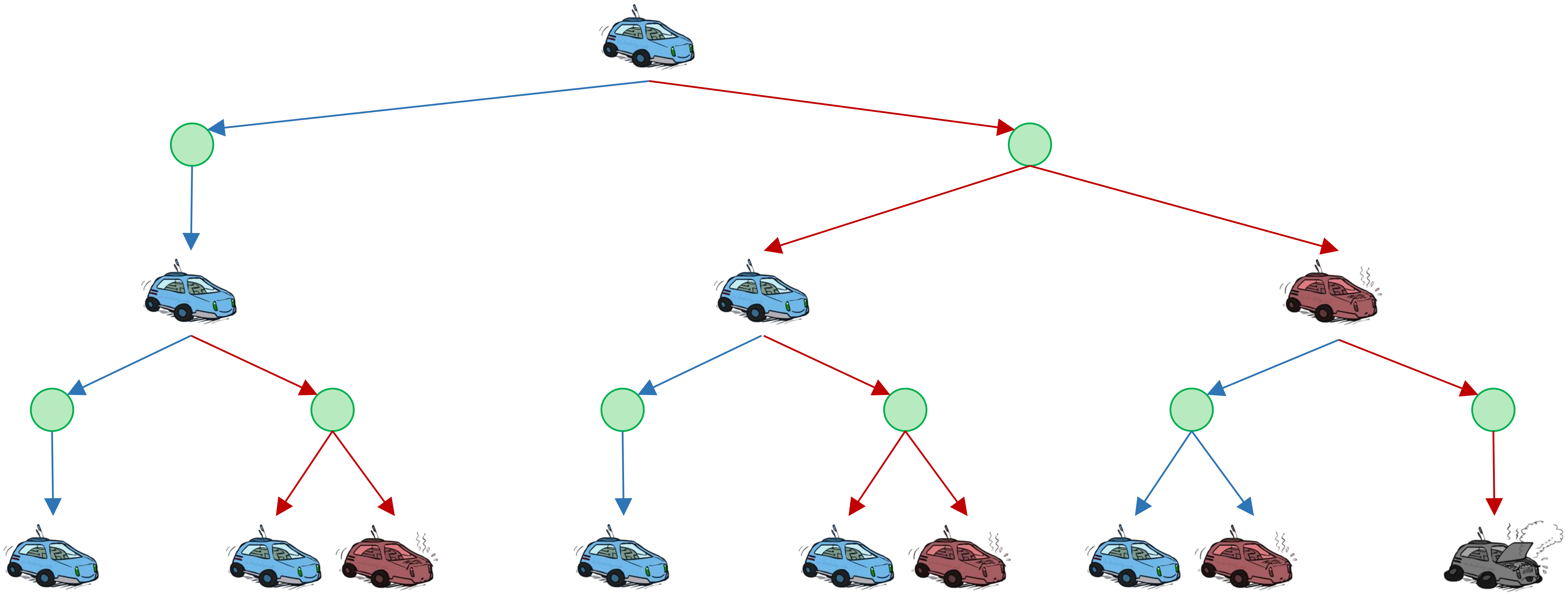
Recursive definition of value:

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

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

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

Racing Search Tree



Racing Search Tree

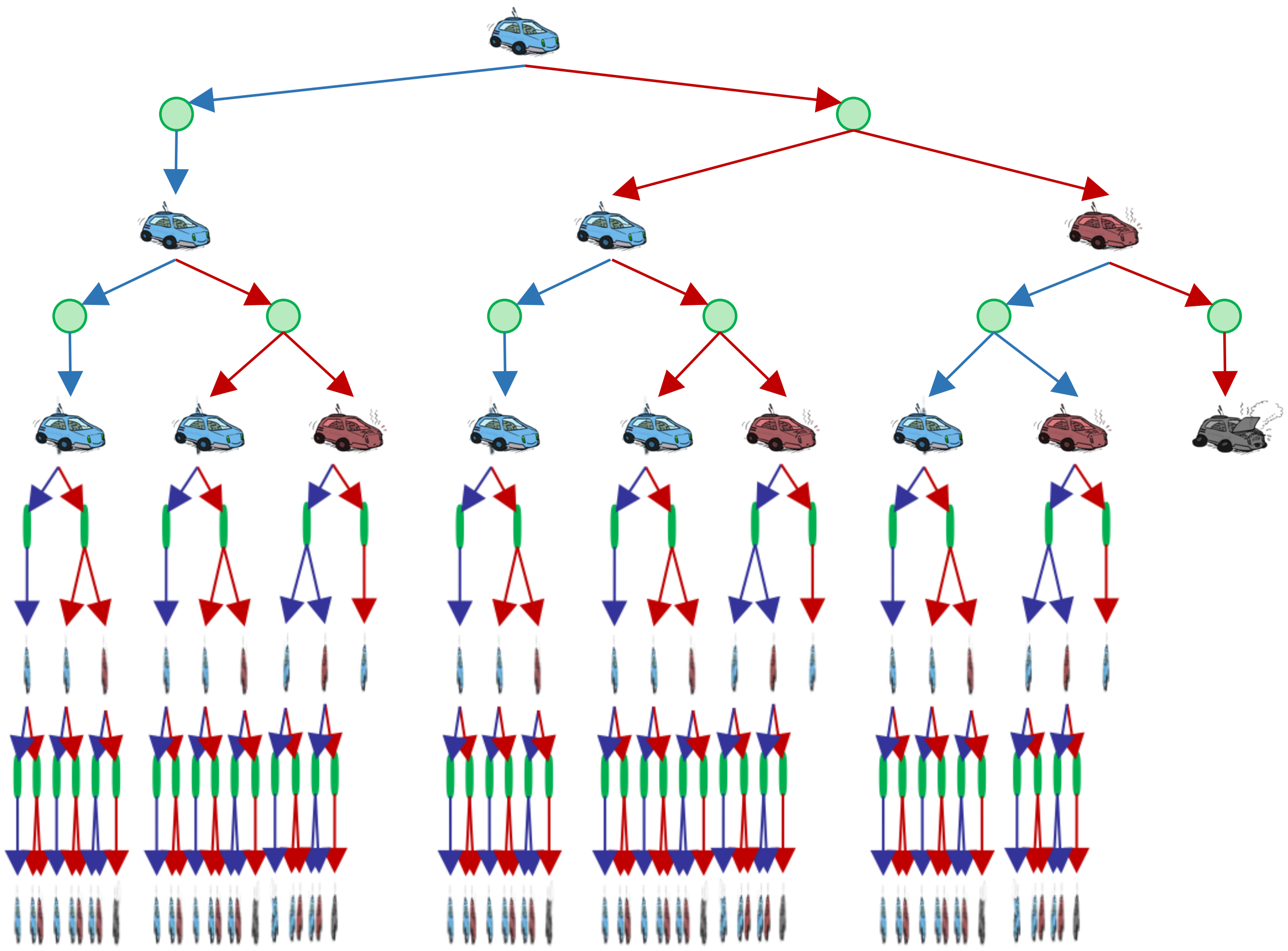
We're doing way too much work with expectimax!

Problem: States are repeated

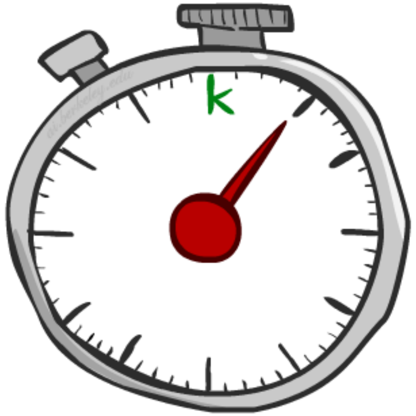
- Idea: Only compute needed quantities once

Problem: Tree goes on forever

- Idea: Do a depth-limited computation, but with increasing depths until change is small
- Note: deep parts of the tree eventually don't matter if $\gamma < 1$



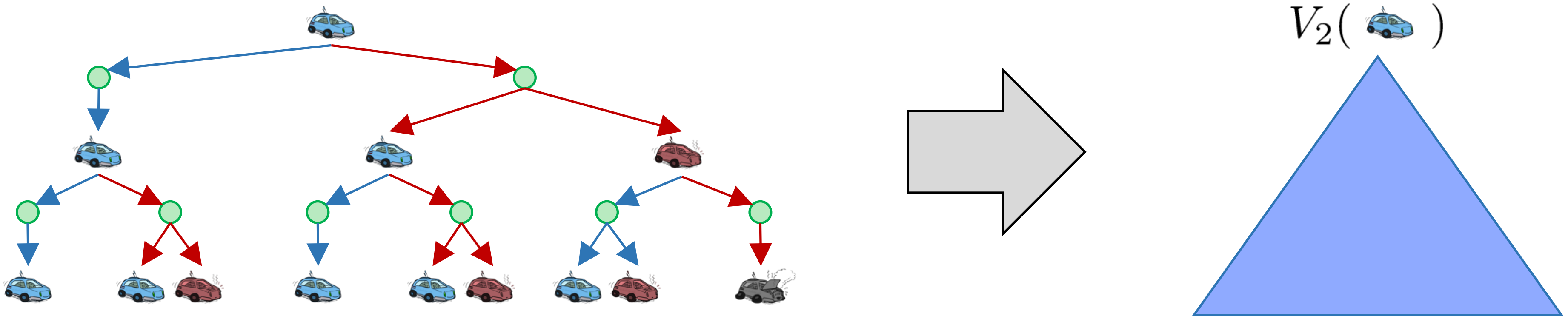
Time-Limited Values



Key idea: time-limited values

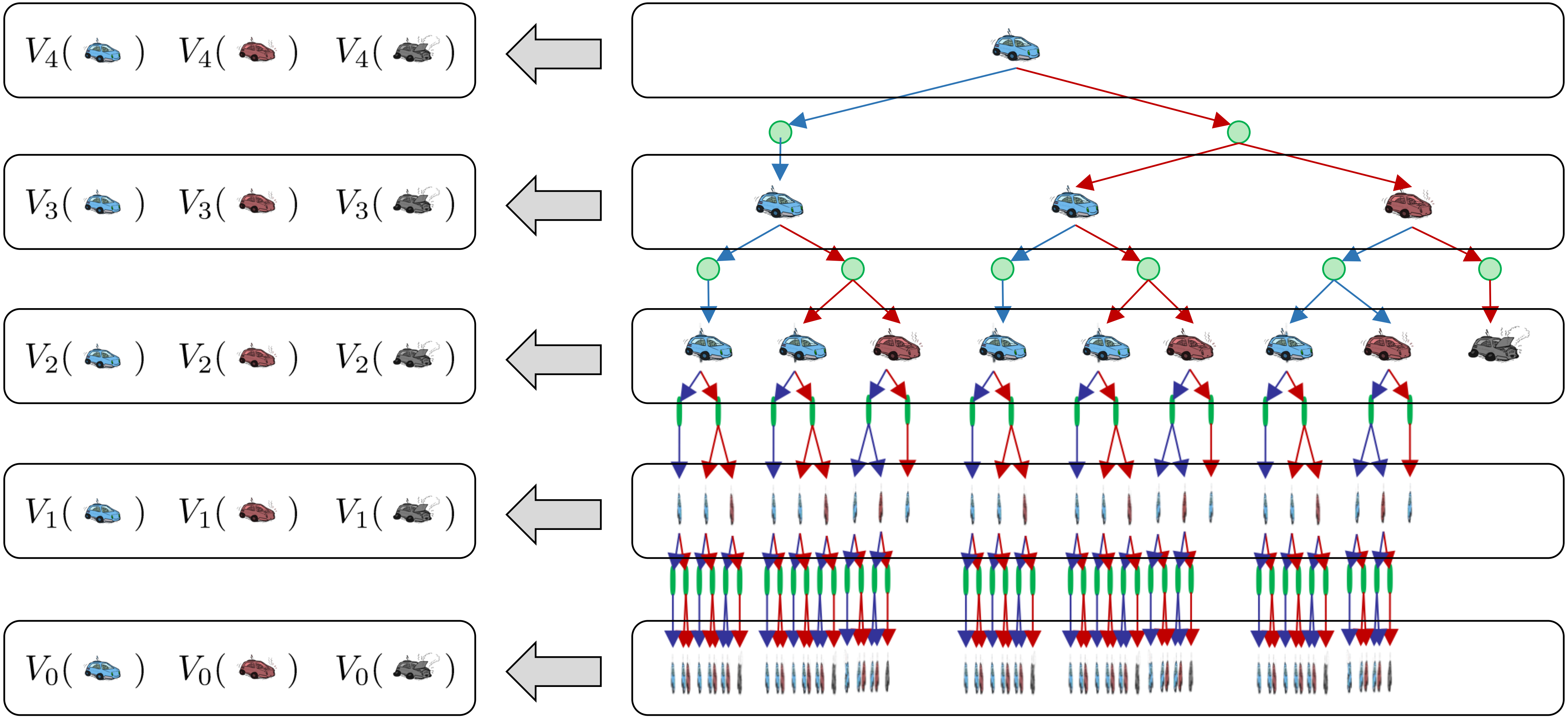
Define $V_k(s)$ to be the optimal value of s if the game ends in k more time steps

- Equivalently, it's what a depth-k expectimax would give from s



[Demo – time-limited values (L8D6)]

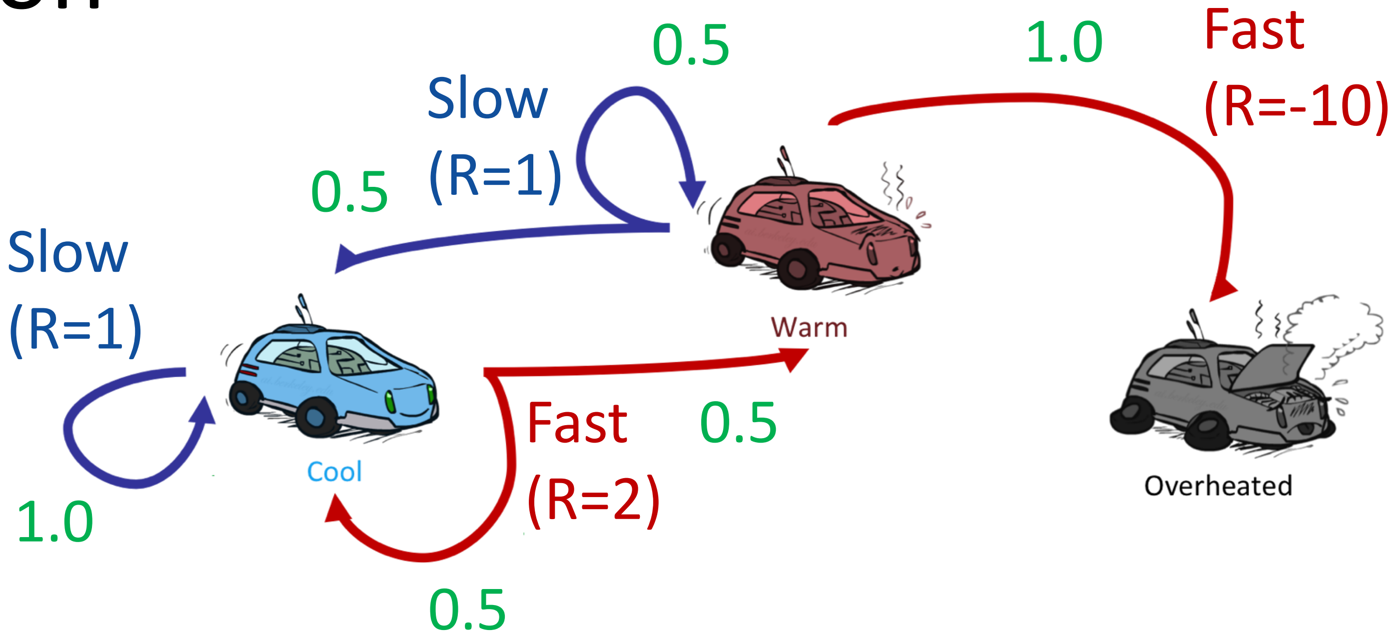
Computing Time-Limited Values



Value Iteration

Example: Value Iteration

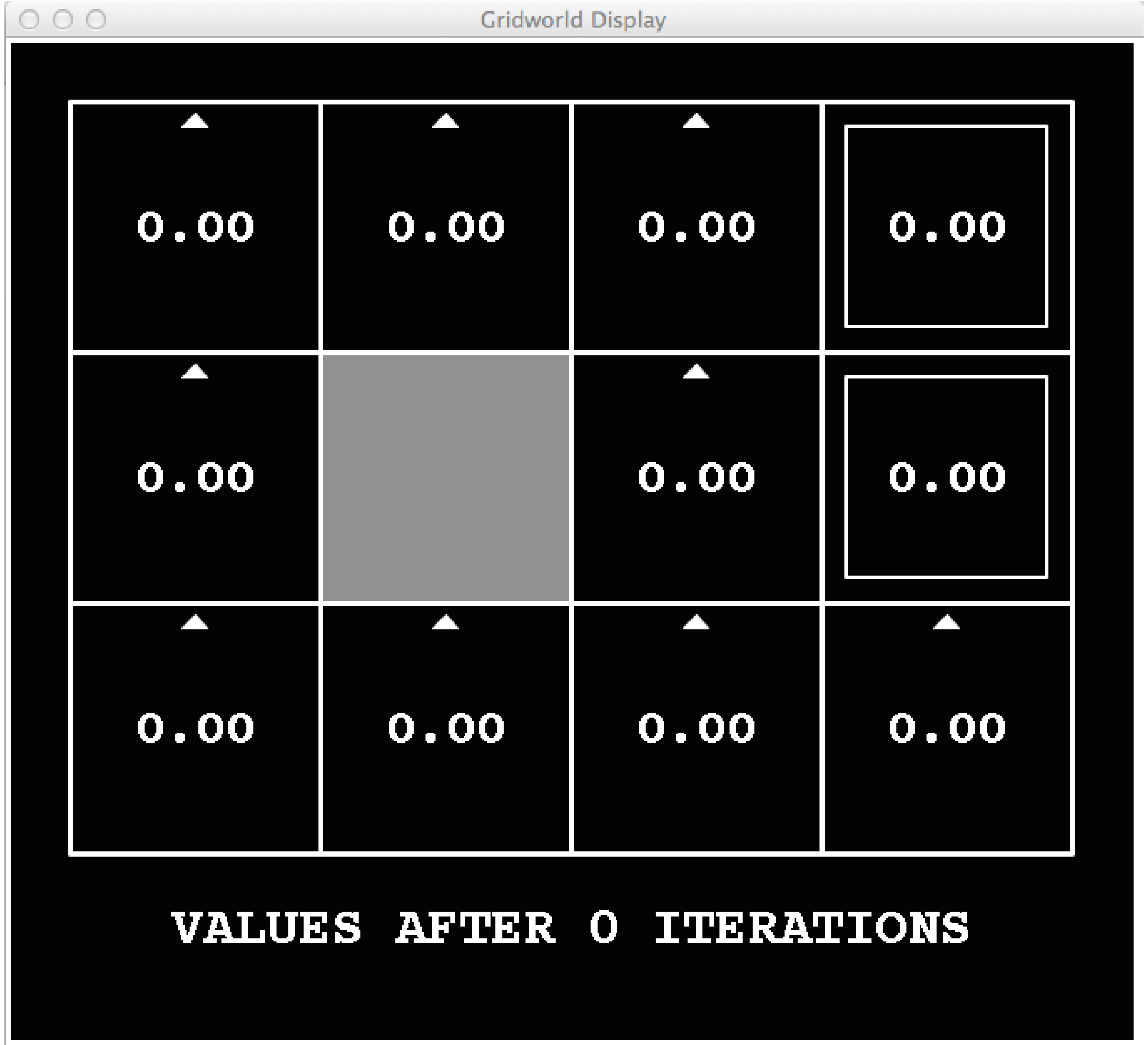
V_2	3.5	2.5	0
V_1	2	1	0
V_0	0	0	0



Assume no discount!

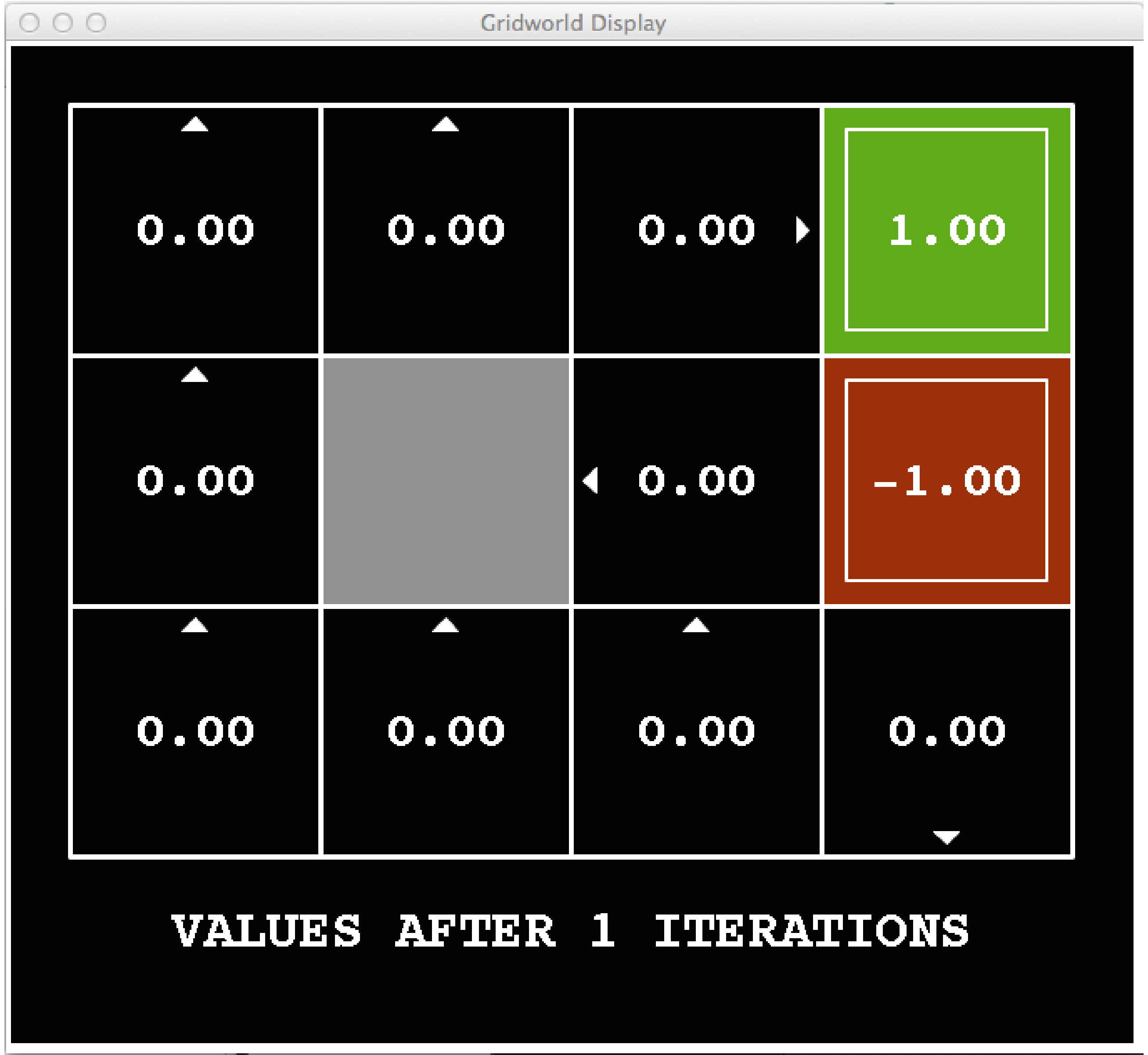
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right]$$

k=0



Noise = 0.2
Discount = 0.9
Living reward = 0

k=1



Noise = 0.2
Discount = 0.9
Living reward = 0

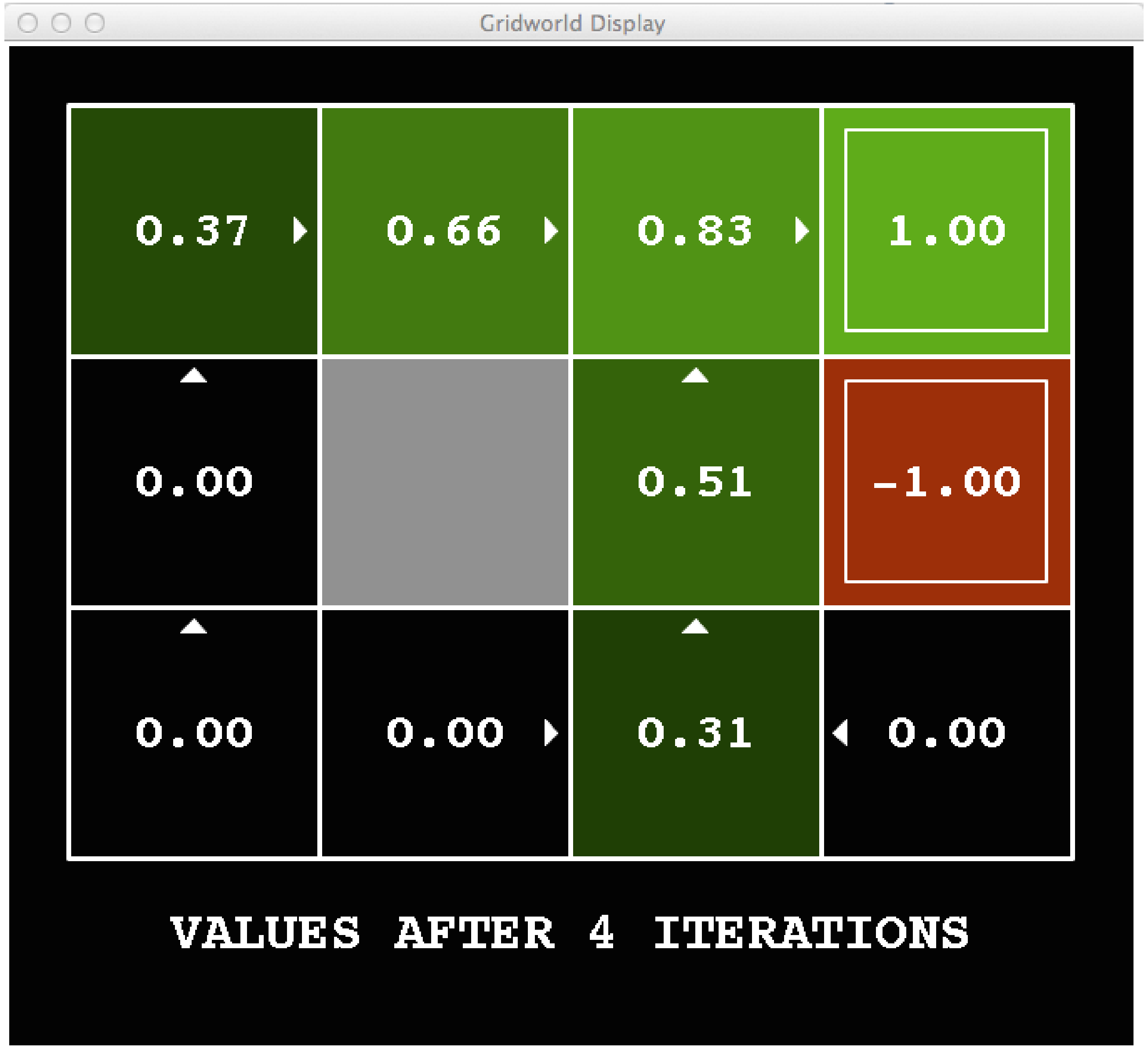
k=2

Noise = 0.2
Discount = 0.9
Living reward = 0

k=3

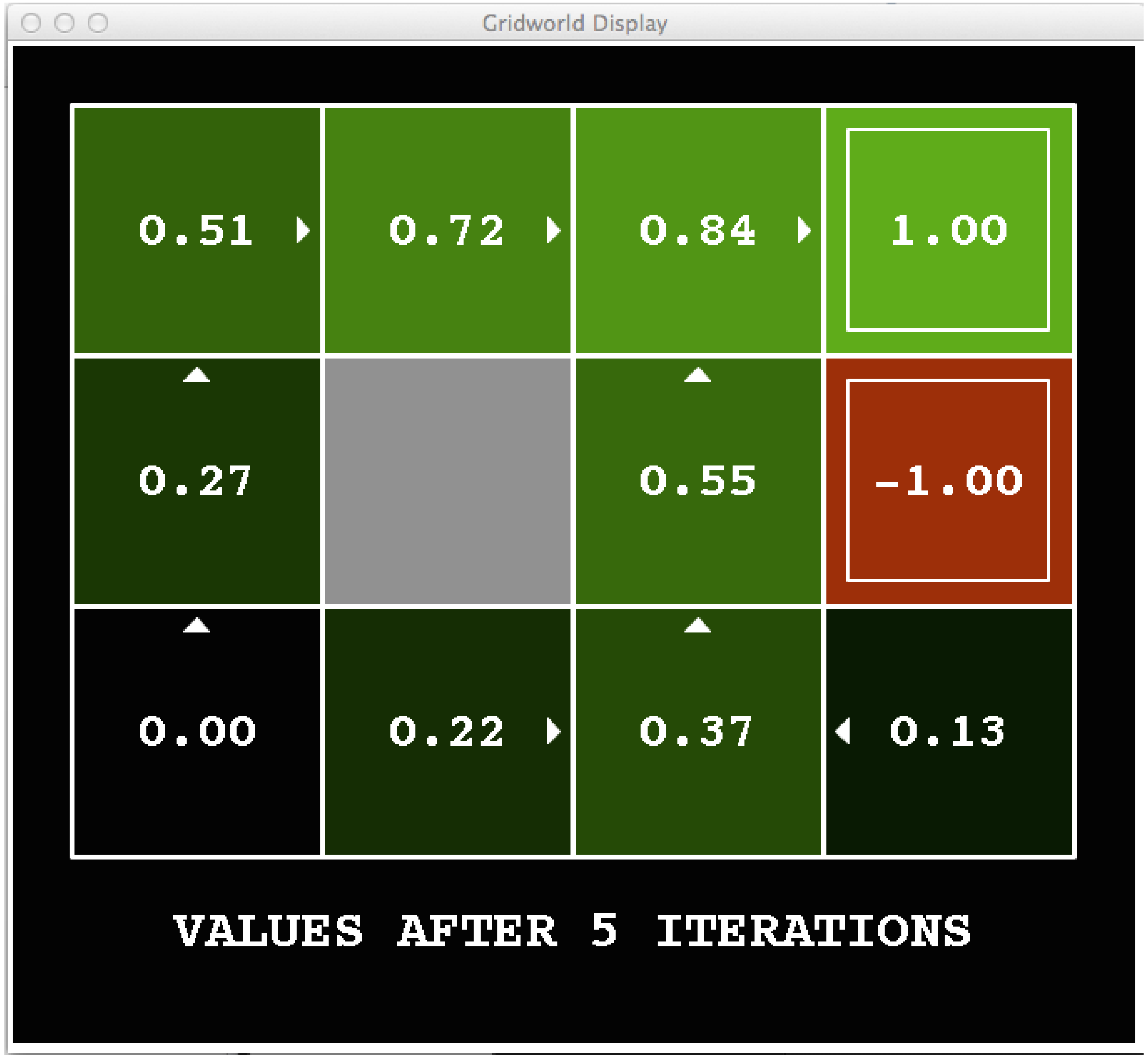
Noise = 0.2
Discount = 0.9
Living reward = 0

k=4



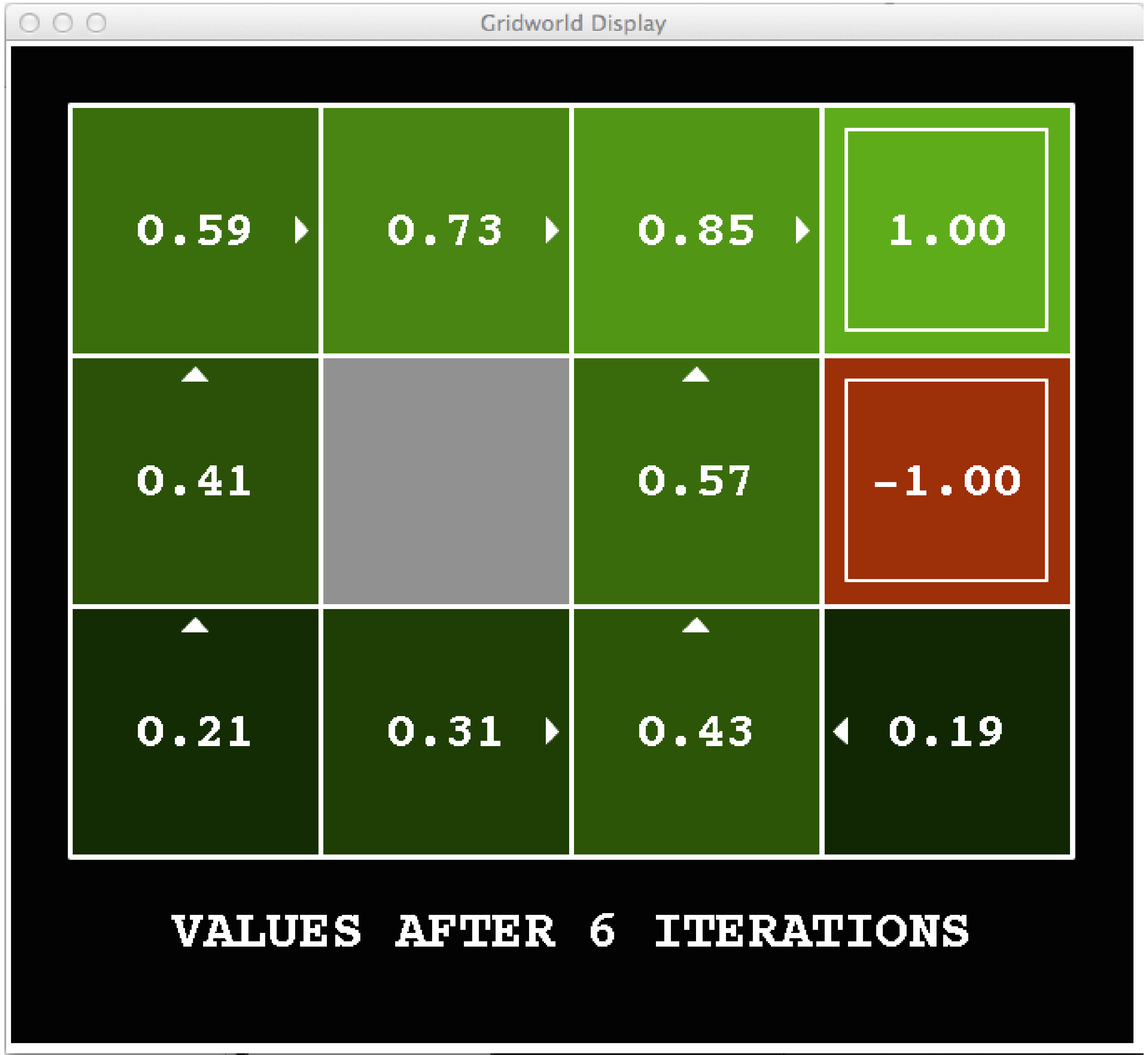
Noise = 0.2
Discount = 0.9
Living reward = 0

k=5



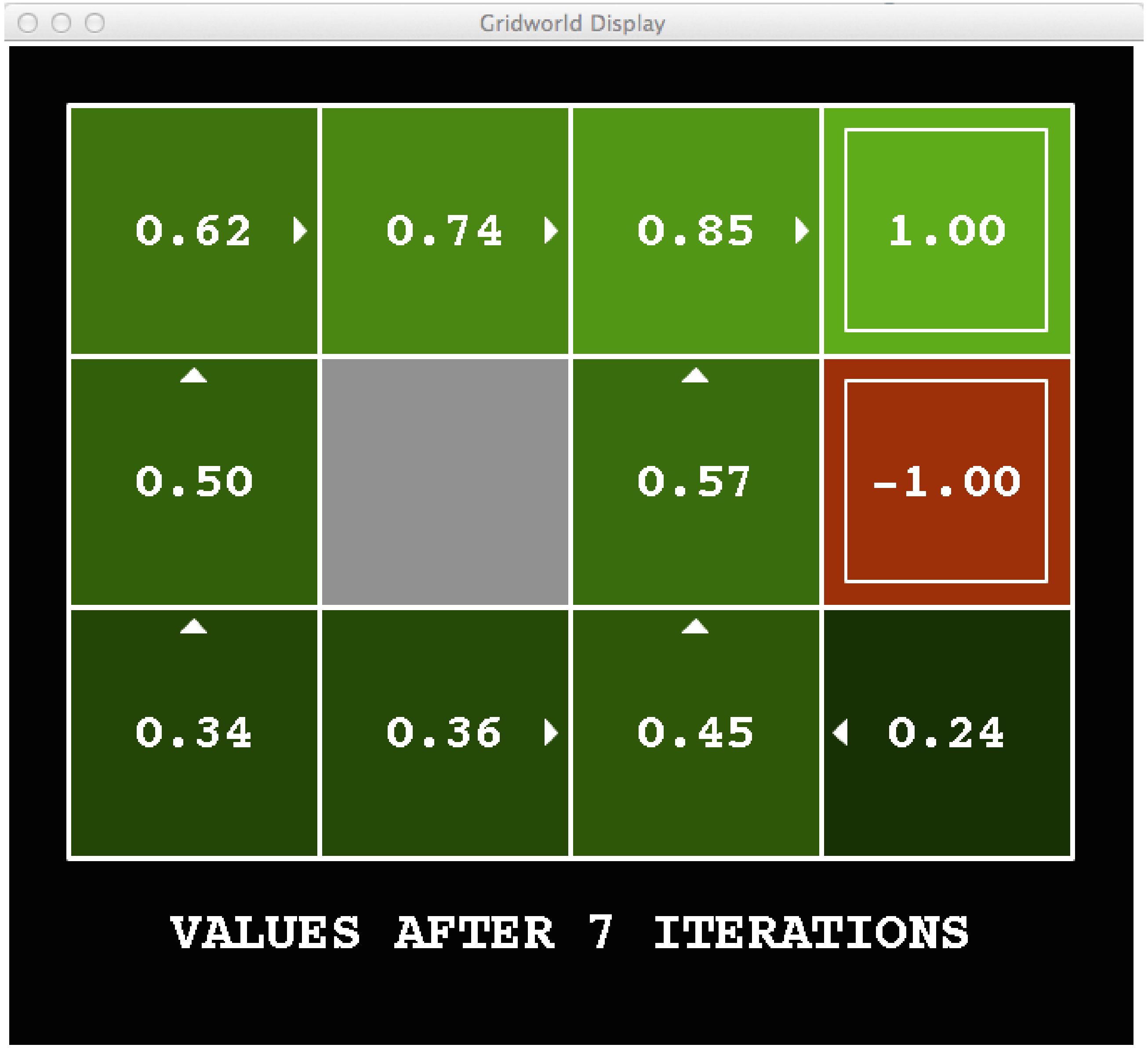
Noise = 0.2
Discount = 0.9
Living reward = 0

k=6



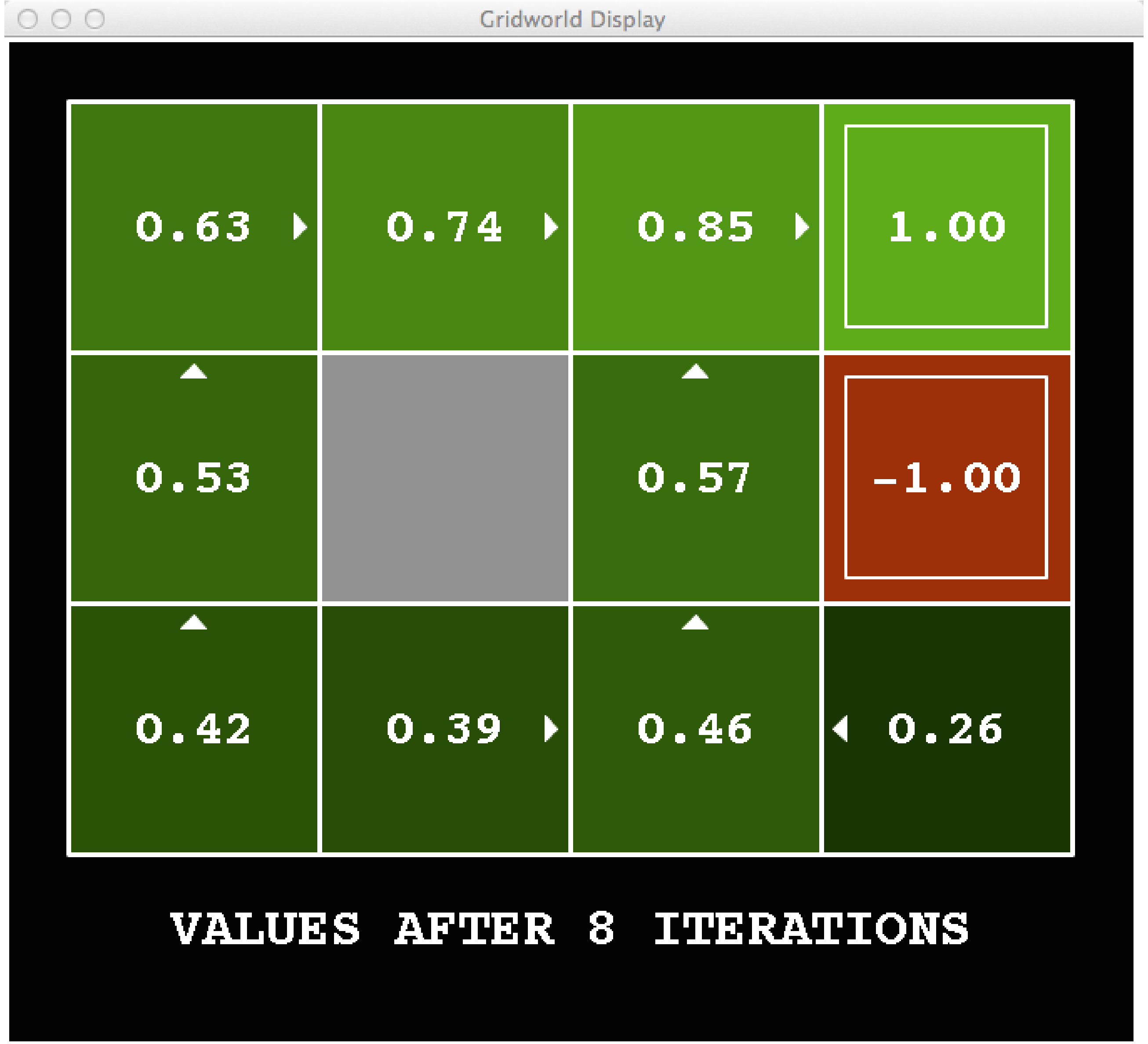
Noise = 0.2
Discount = 0.9
Living reward = 0

k=7



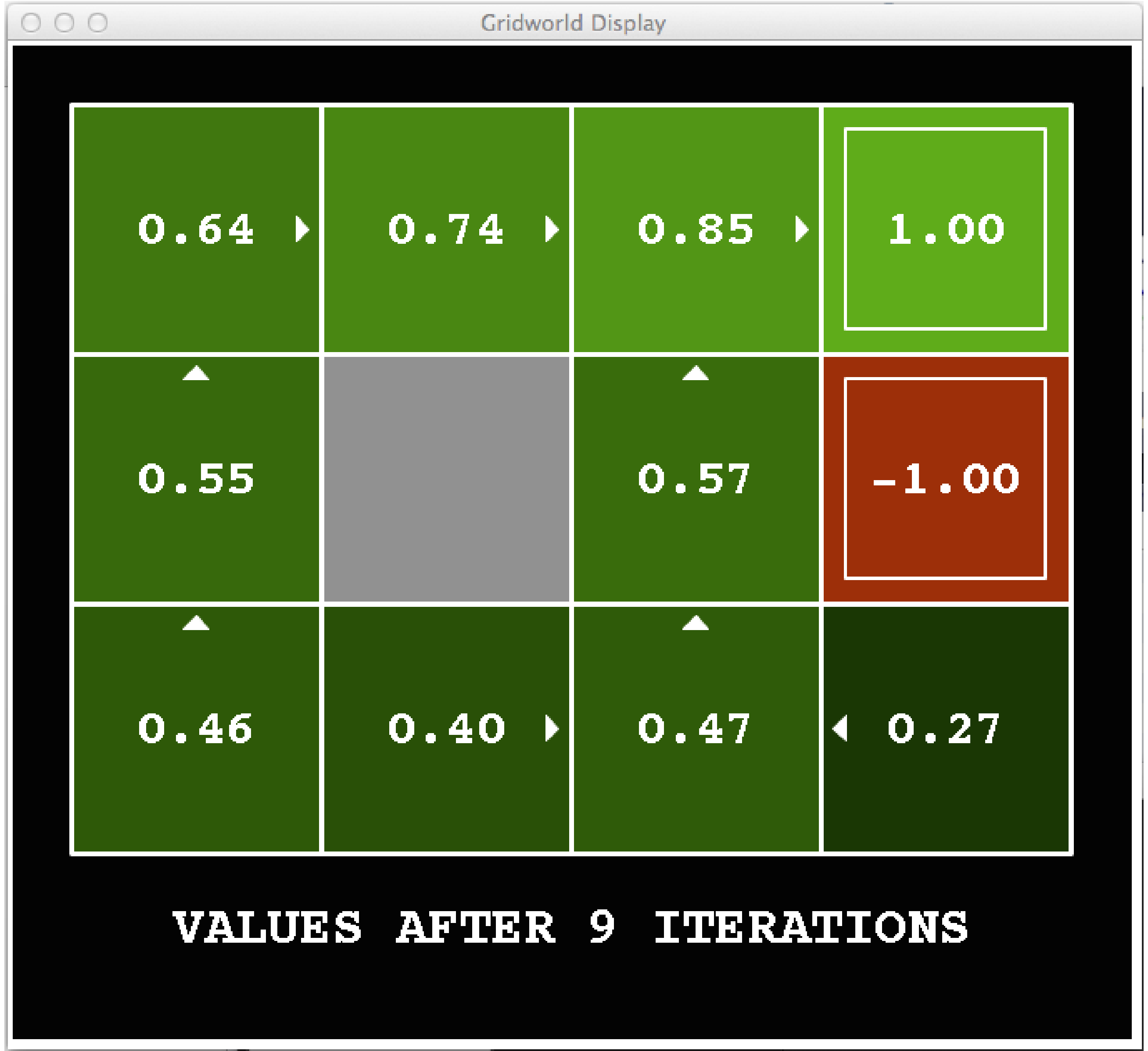
Noise = 0.2
Discount = 0.9
Living reward = 0

k=8



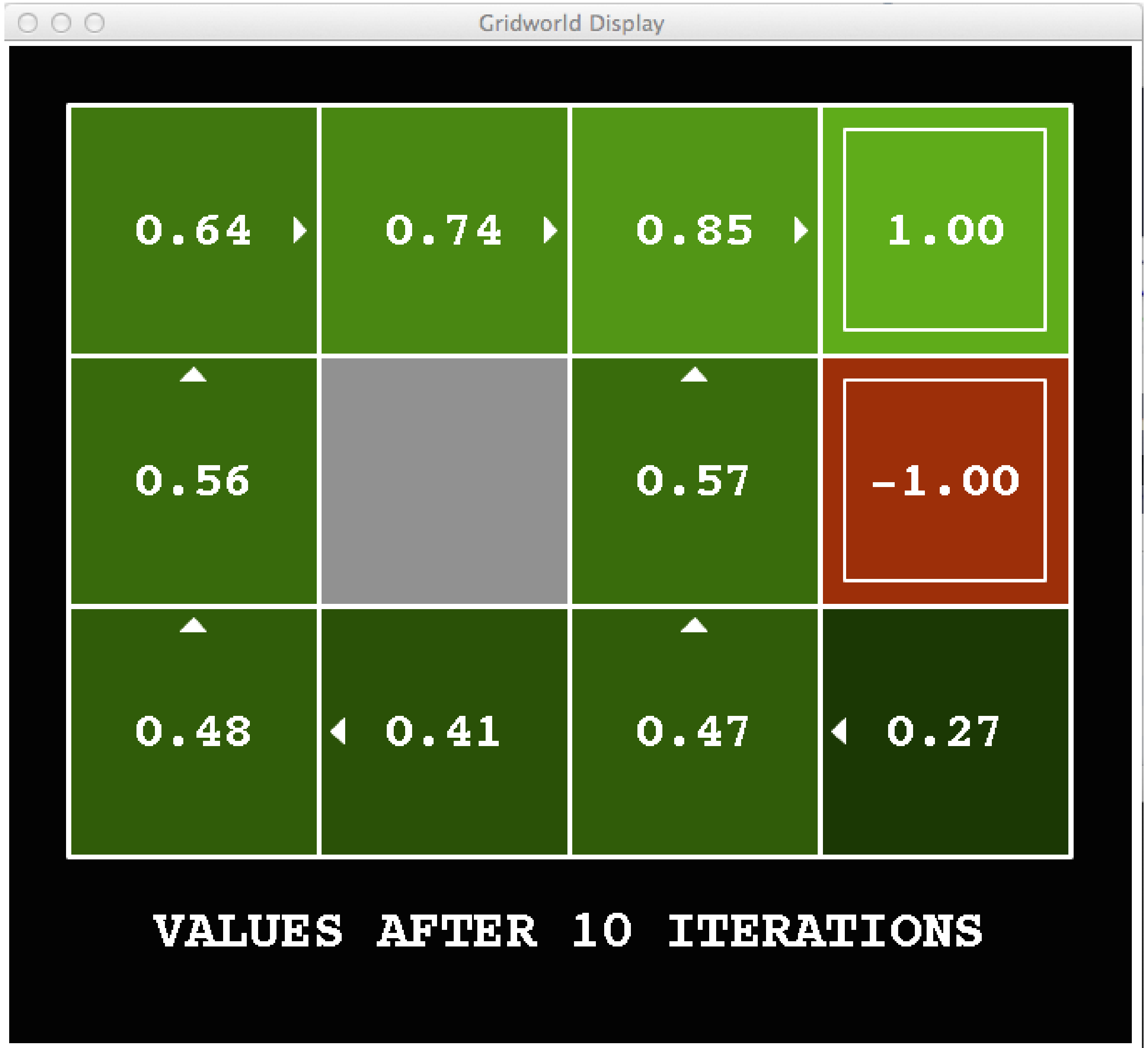
Noise = 0.2
Discount = 0.9
Living reward = 0

k=9



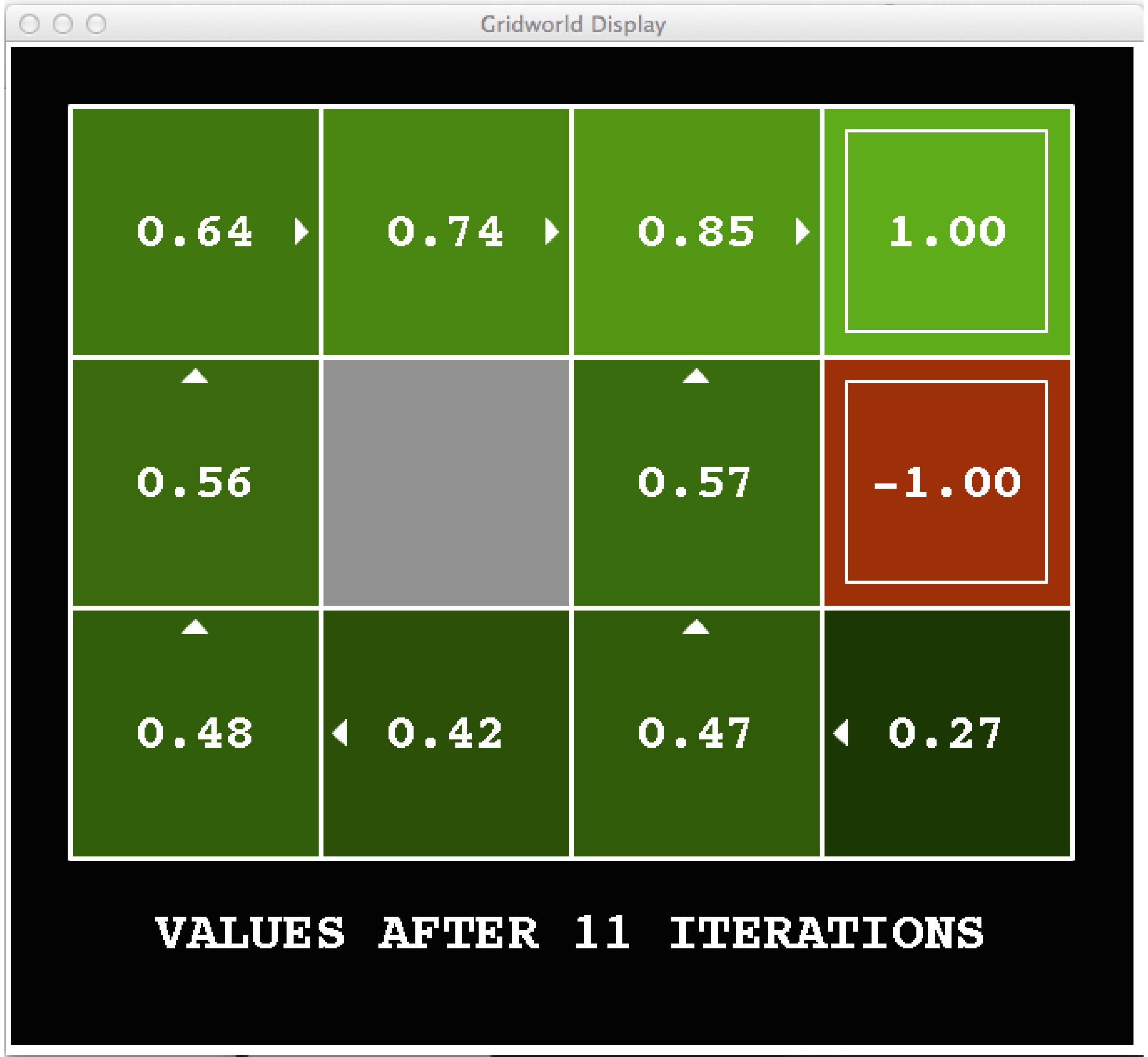
Noise = 0.2
Discount = 0.9
Living reward = 0

k=10



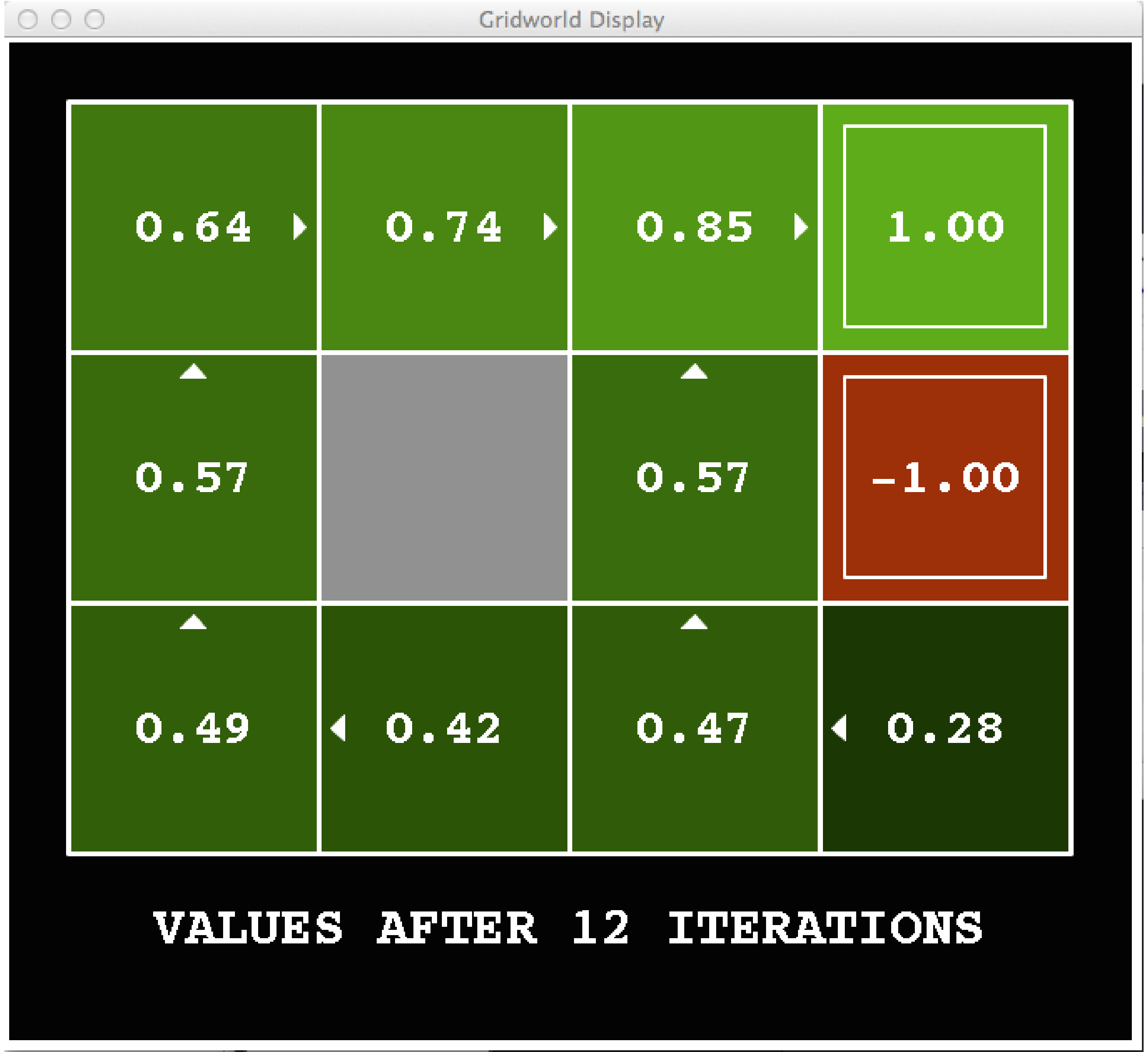
Noise = 0.2
Discount = 0.9
Living reward = 0

k=11



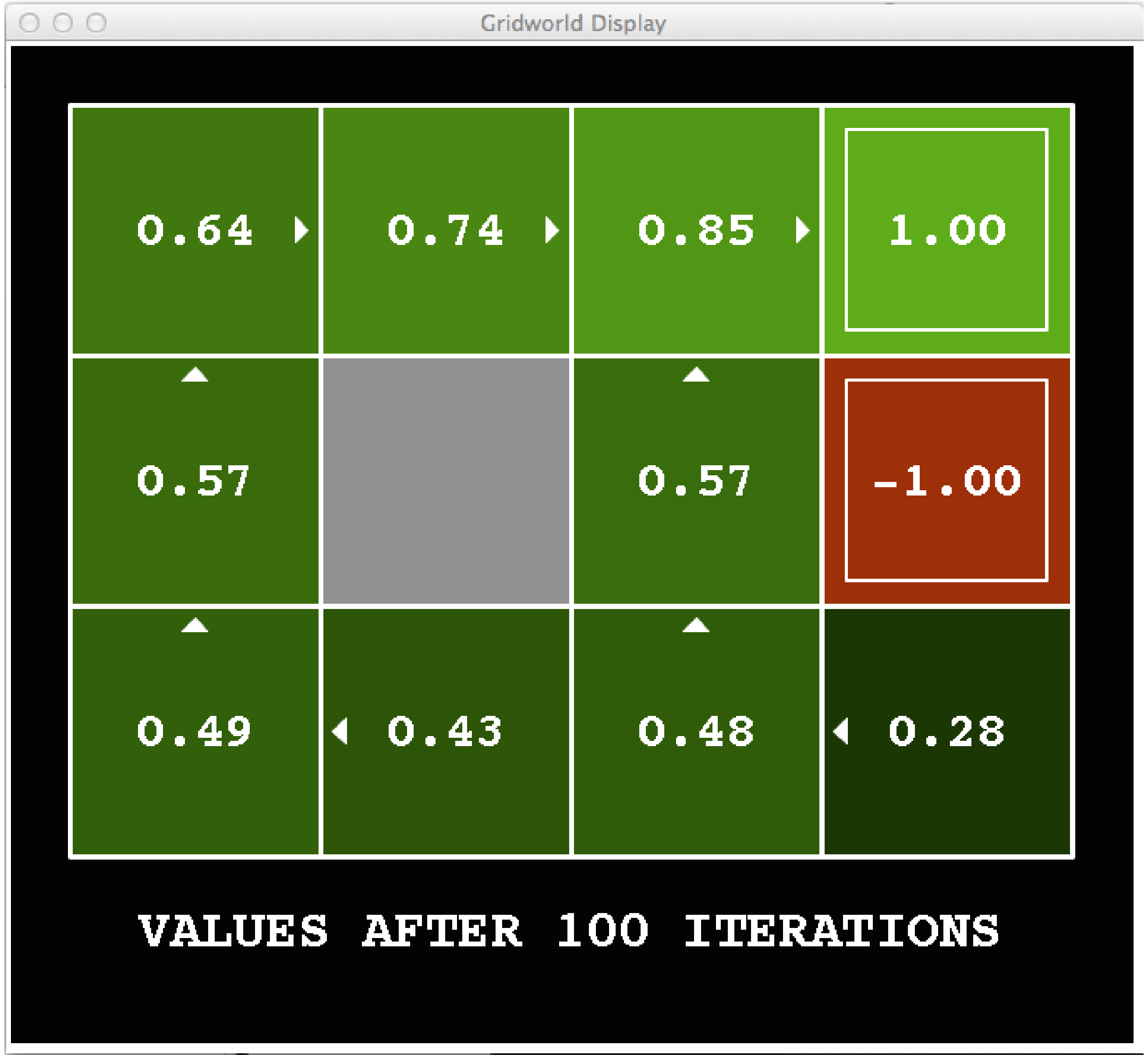
Noise = 0.2
Discount = 0.9
Living reward = 0

k=12

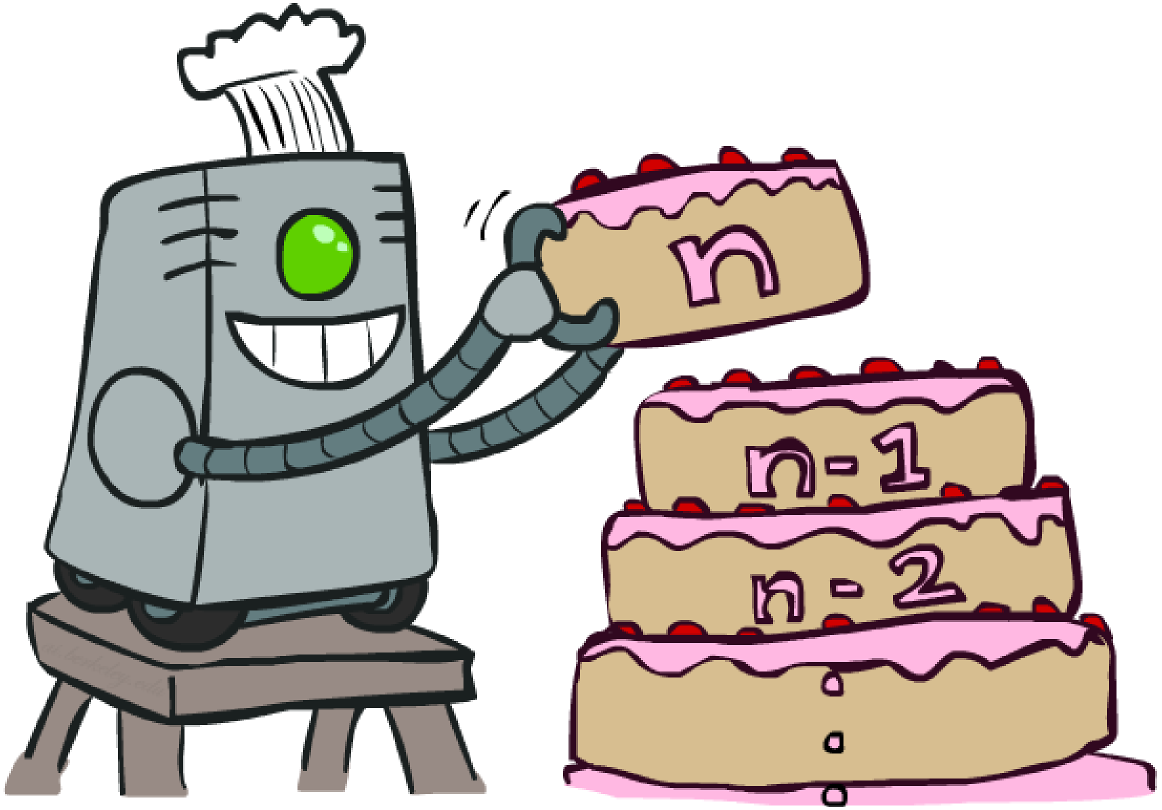


Noise = 0.2
Discount = 0.9
Living reward = 0

k=100



Noise = 0.2
Discount = 0.9
Living reward = 0



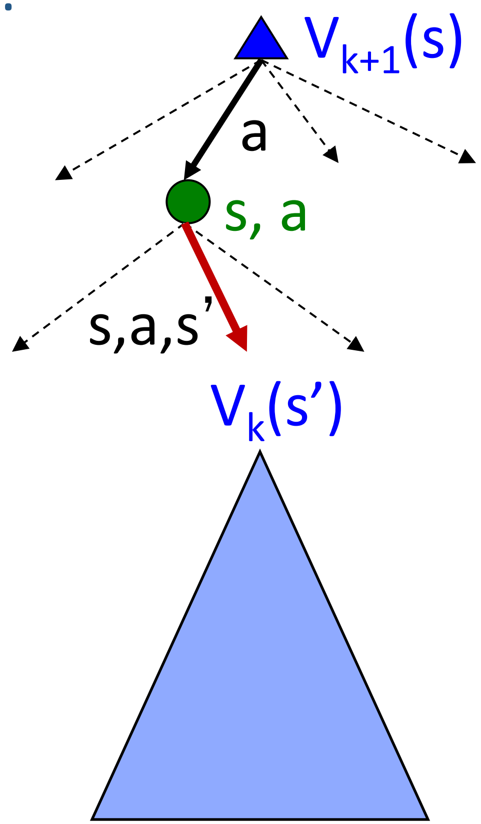
Value Iteration

Value Iteration

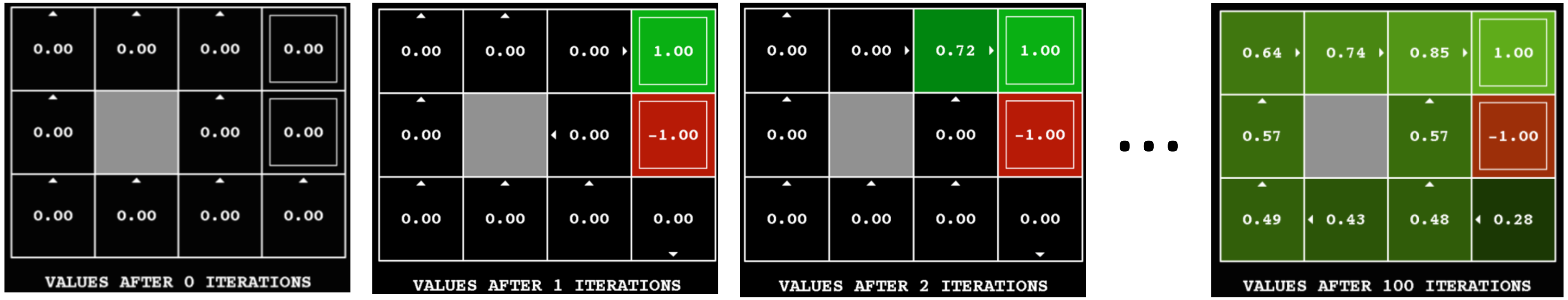
Start with $V_0(s) = 0$: no time steps left means an expected reward sum of zero

Given vector of $V_k(s)$ values, do one ply 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



Poll 1

What is the complexity of each iteration in Value Iteration?

S -- set of states; A -- set of actions

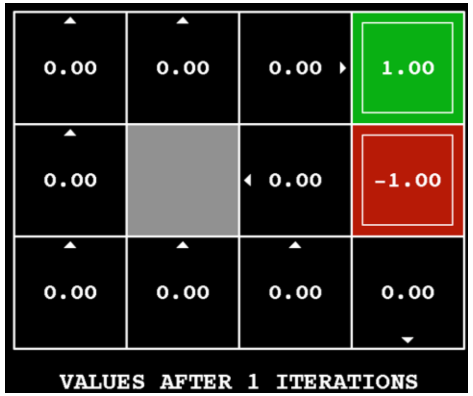
I: $O(|S||A|)$

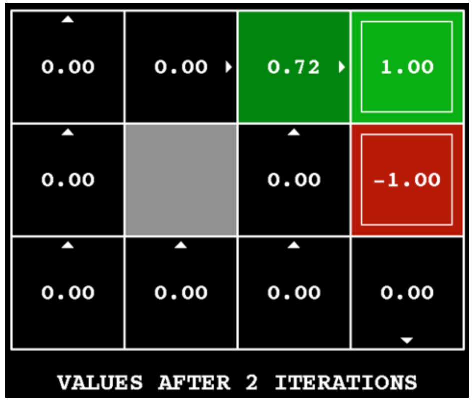
II: $O(|S|^2|A|)$

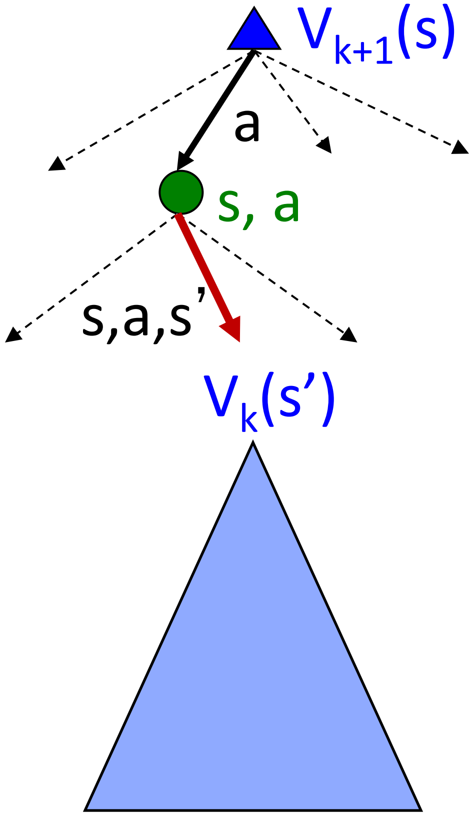
III: $O(|S||A|^2)$

IV: $O(|S|^2|A|^2)$

V: $O(|S|^2)$





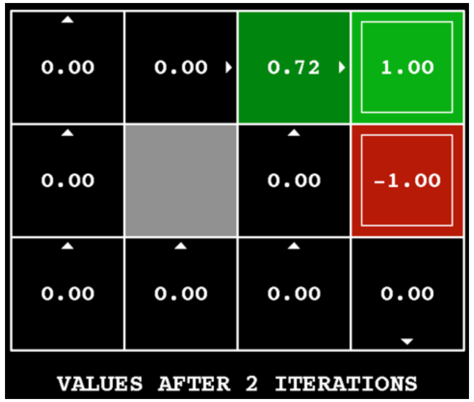


$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right]$$

Poll 1

What is the complexity of each iteration in Value Iteration?

S -- set of states; A -- set of actions



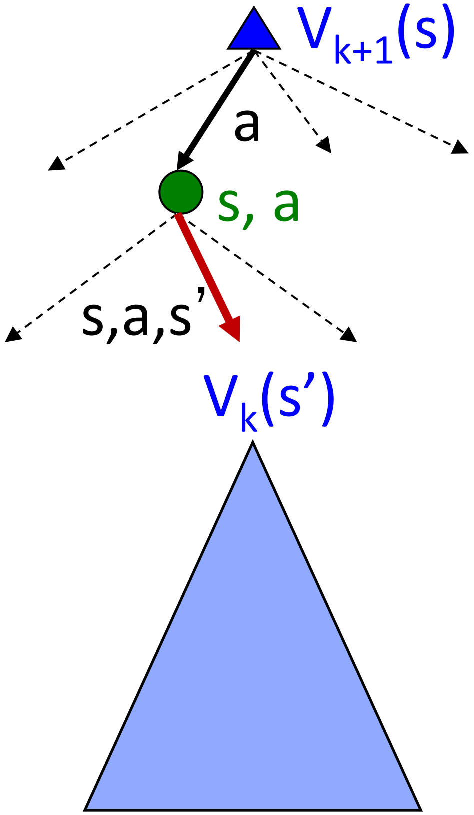
I: $O(|S||A|)$

II: $O(|S|^2|A|)$

III: $O(|S||A|^2)$

IV: $O(|S|^2|A|^2)$

V: $O(|S|^2)$



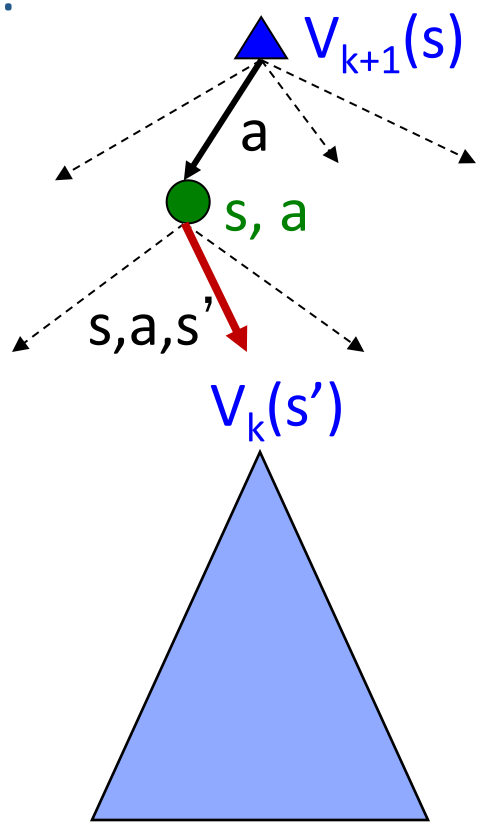
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right]$$

Value Iteration

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 ply 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

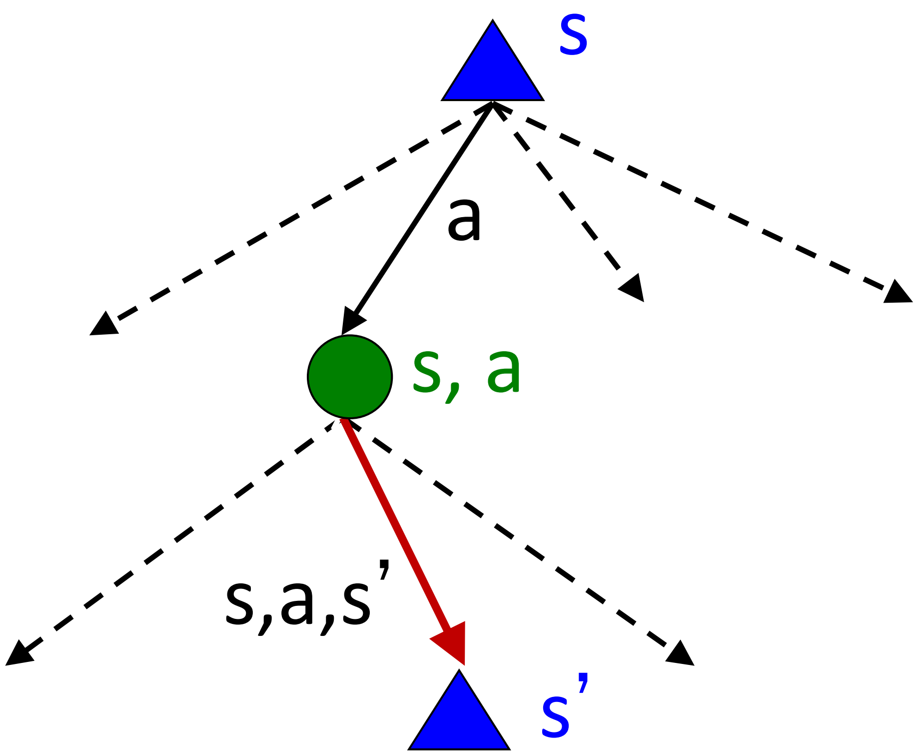
Complexity of each iteration: $O(S^2A)$

Theorem: will converge to unique optimal values

- Basic idea: approximations get refined towards optimal values
- Policy may converge long before values do

Optimal Quantities

- The value (utility) of a state s:
  $V^*(s)$ = expected utility starting in s and acting optimally



[Demo: gridworld values (L9D1)]

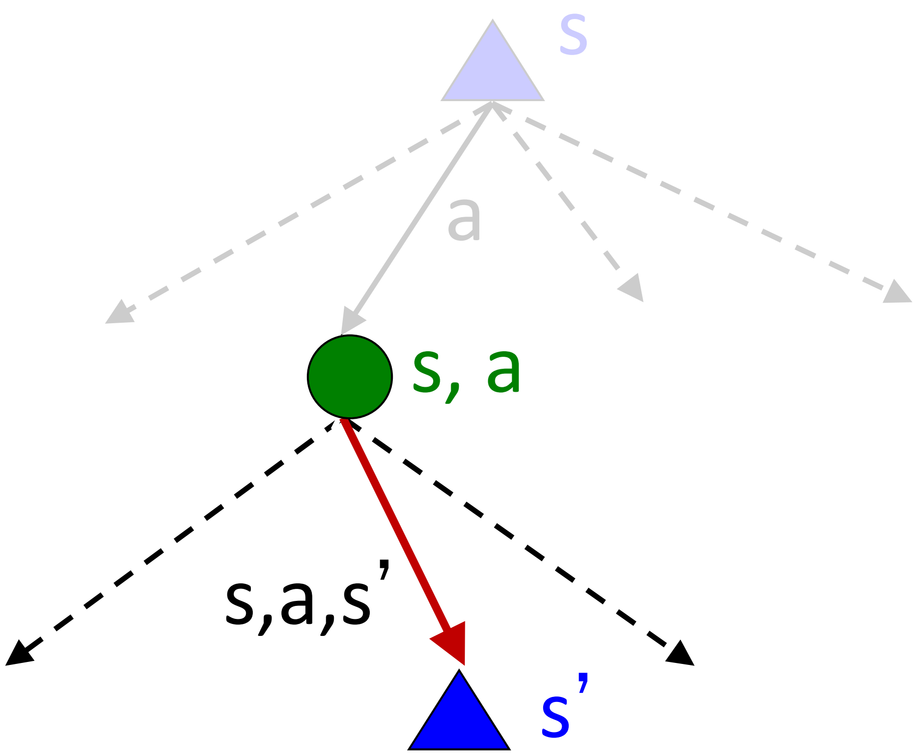
Optimal Quantities

- 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



[Demo: gridworld values (L9D1)]

Optimal Quantities

- The value (utility) of a state s:

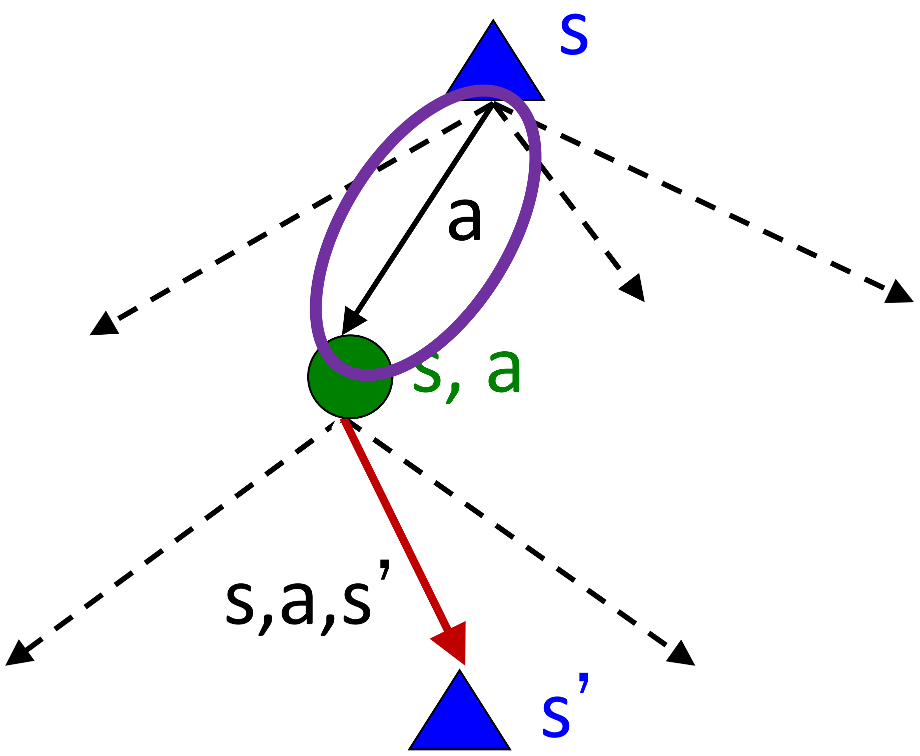
  $V^*(s)$ = expected utility starting in s and acting optimally

- 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:

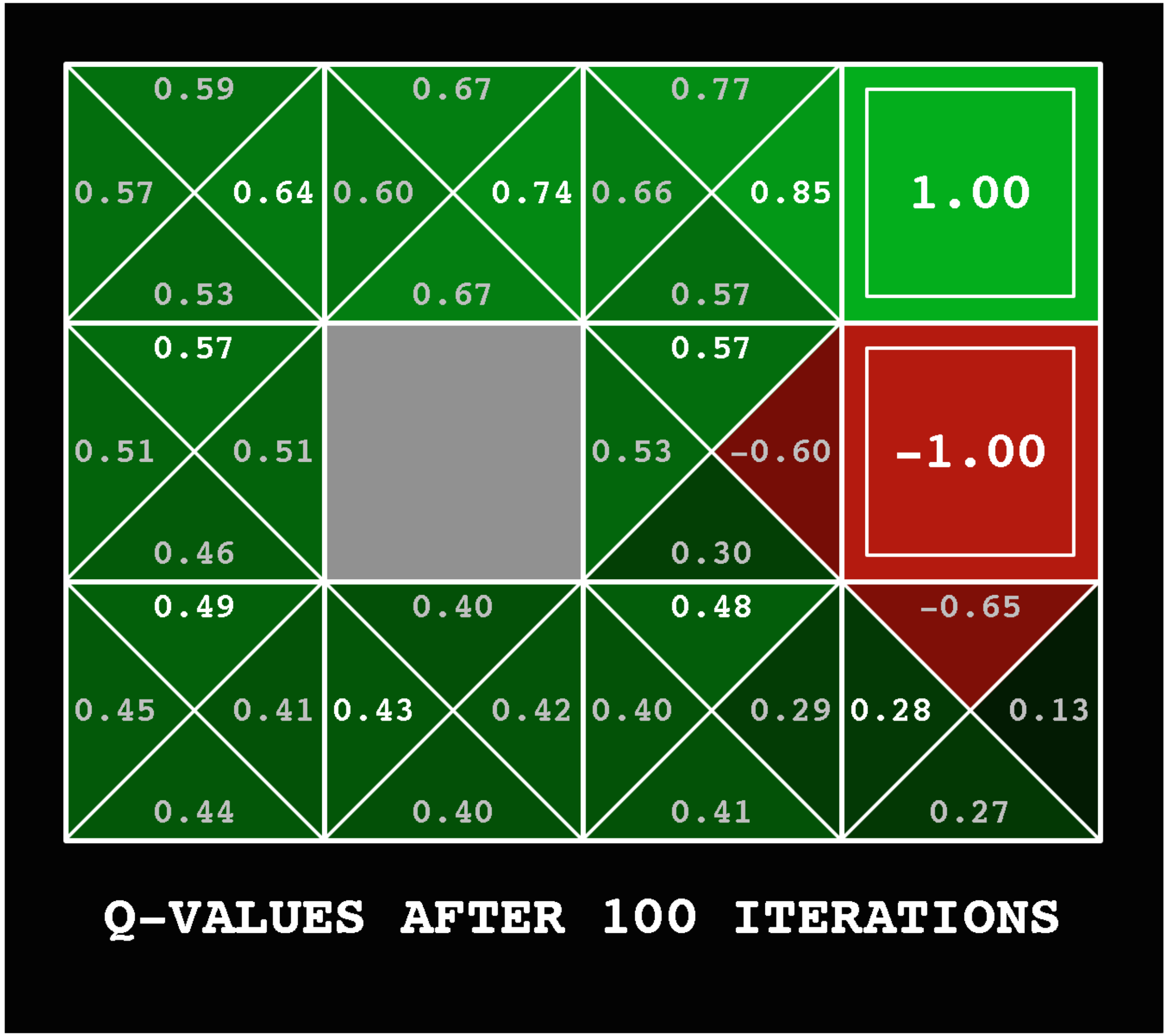
  $\pi^*(s)$ = optimal action from state s



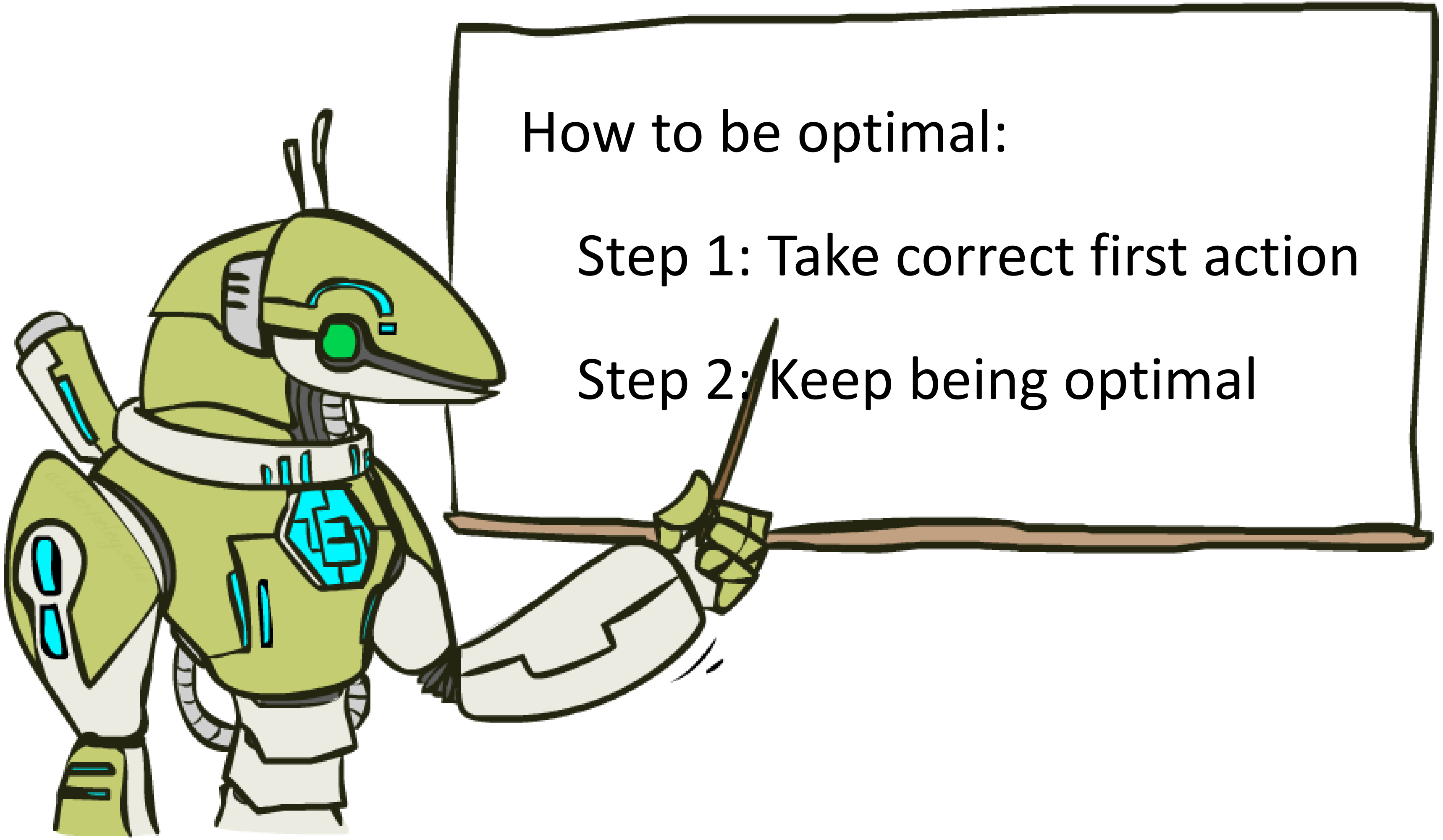
[Demo: gridworld values (L9D1)]

Gridworld Values V*

Gridworld: Q*

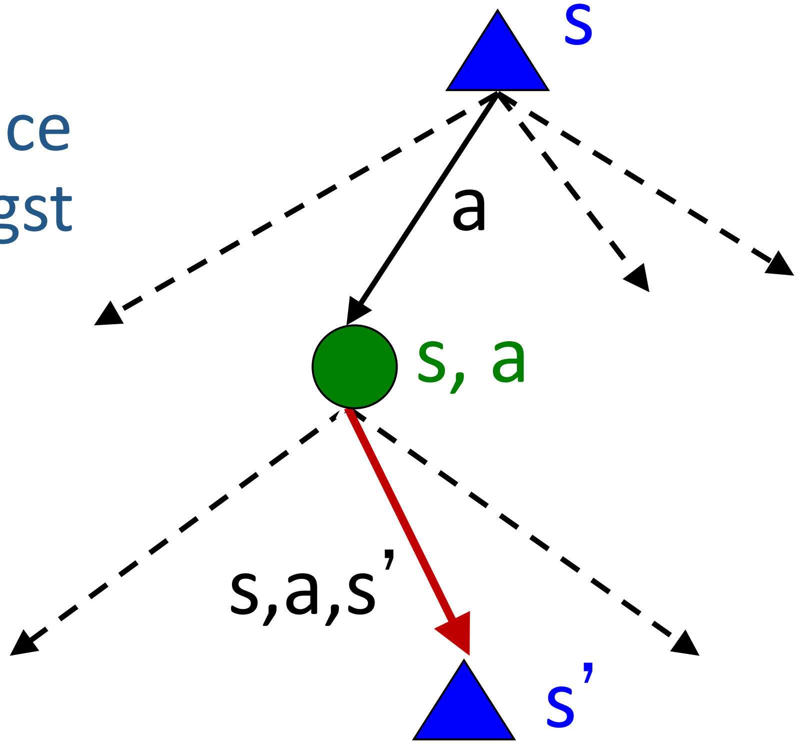

0.59
0.57
0.64
0.53
0.67
0.60
0.74
0.67
0.77
0.66
0.85
0.57
1.00
0.57
0.51
0.51
0.46
0.57
0.53
-0.60
0.30
-1.00
0.49
0.45
0.41
0.44
0.40
0.43
0.42
0.40
0.48
0.40
0.29
0.41
-0.65
0.28
0.13
0.27
Q-VALUES AFTER 100 ITERATIONS

The Bellman Equations



The Bellman Equations

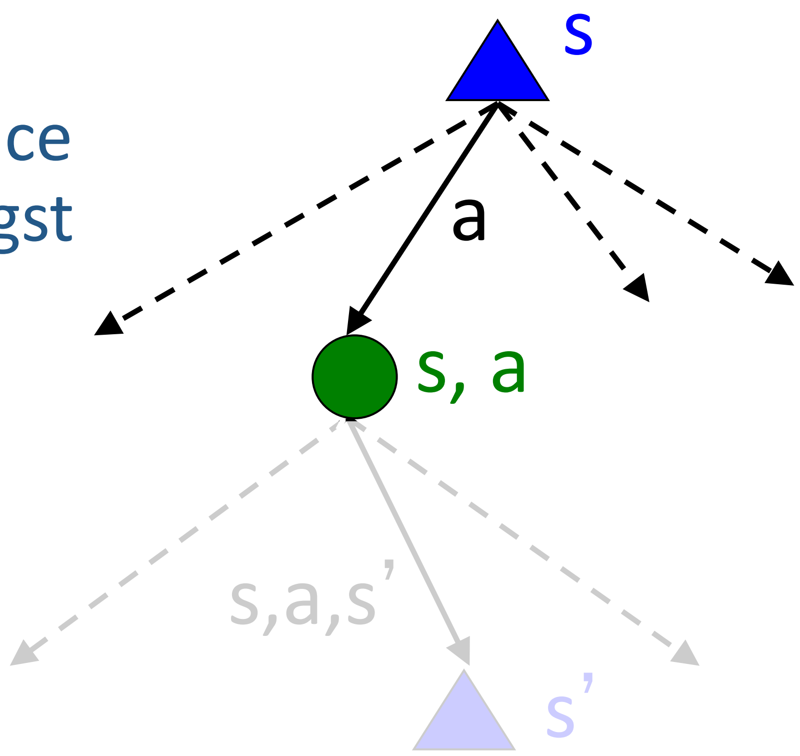
Definition of "optimal utility" via expectimax recurrence gives a simple one-step lookahead relationship amongst optimal utility values



The Bellman Equations

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

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

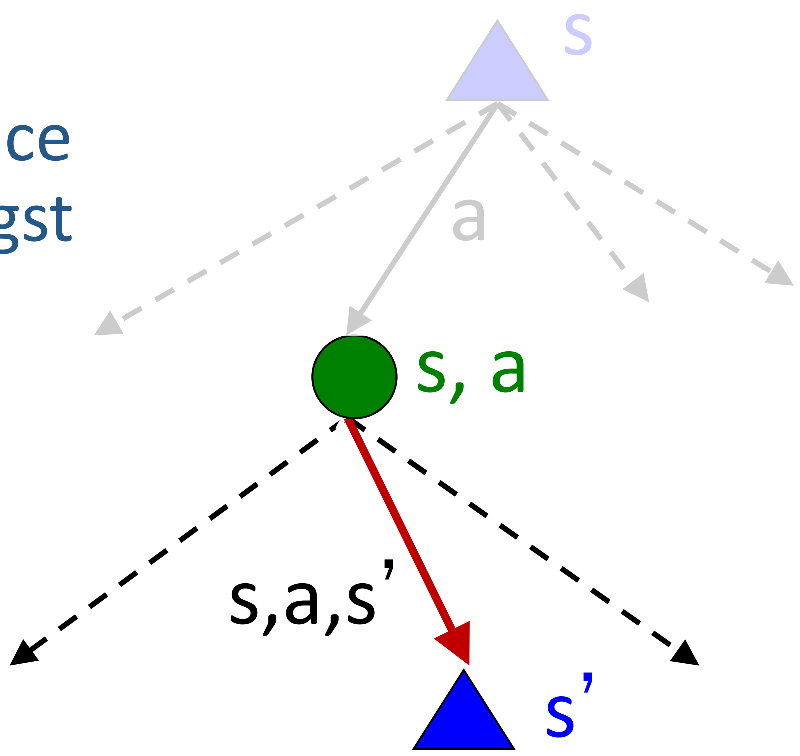


The Bellman Equations

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

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

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



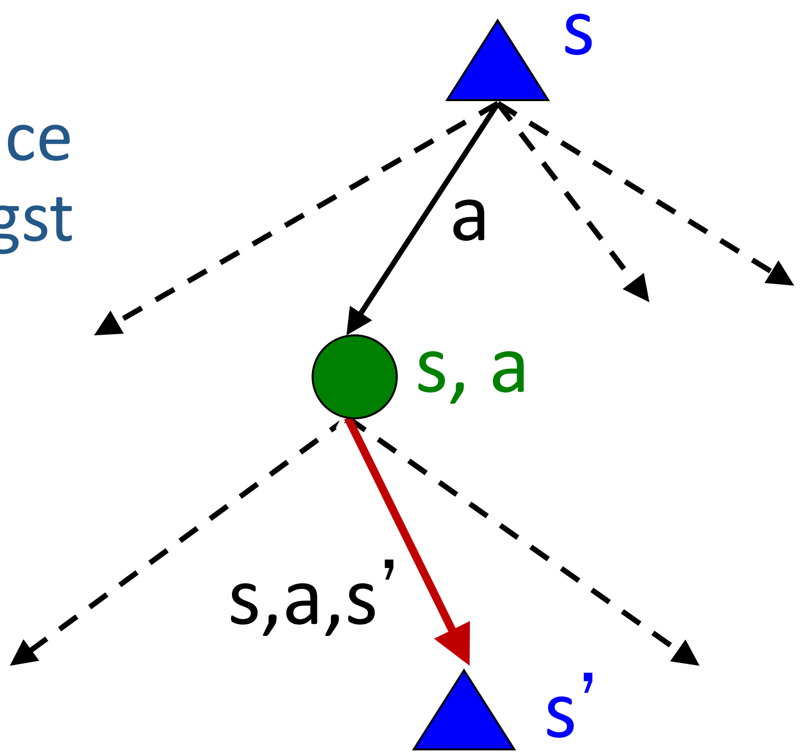
The Bellman Equations

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

MDP Notation

Standard expectimax: $V(s) = \max_a \sum_{s'} P(s'|s,a)V(s')$

Bellman equations: $V^*(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^*(s')]$

Value iteration: $V_{k+1}(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V_k(s')], \quad \forall s$

Value Iteration

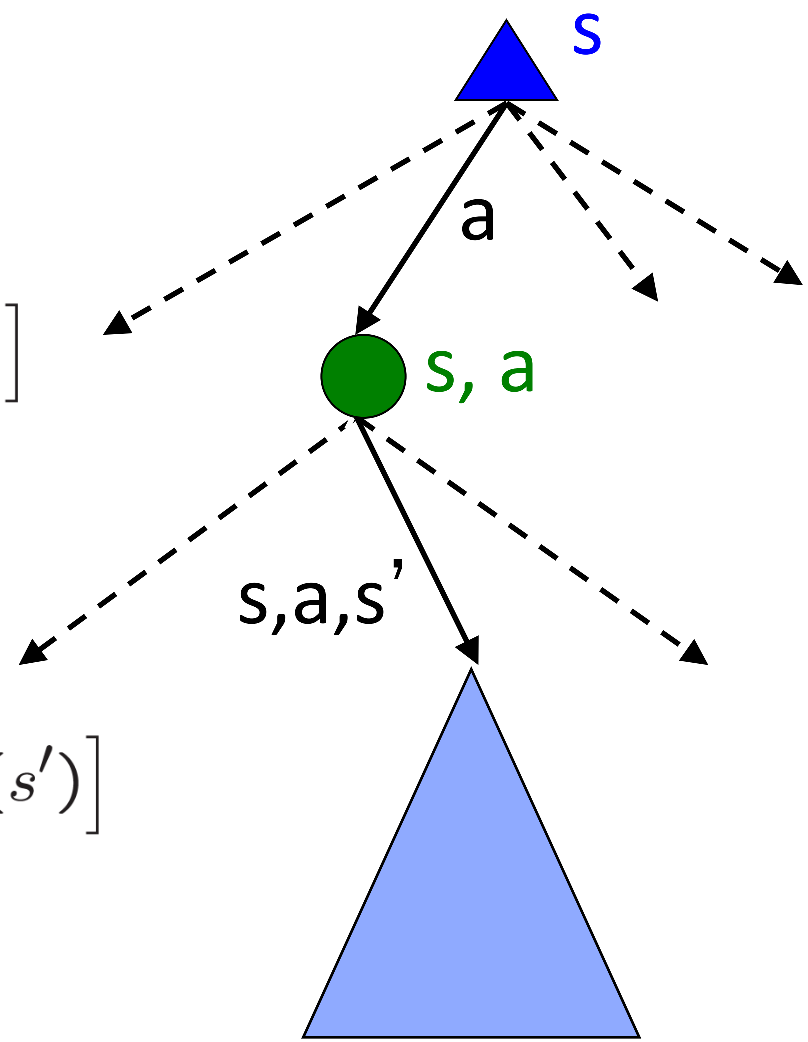
Bellman equations characterize the optimal values:

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

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



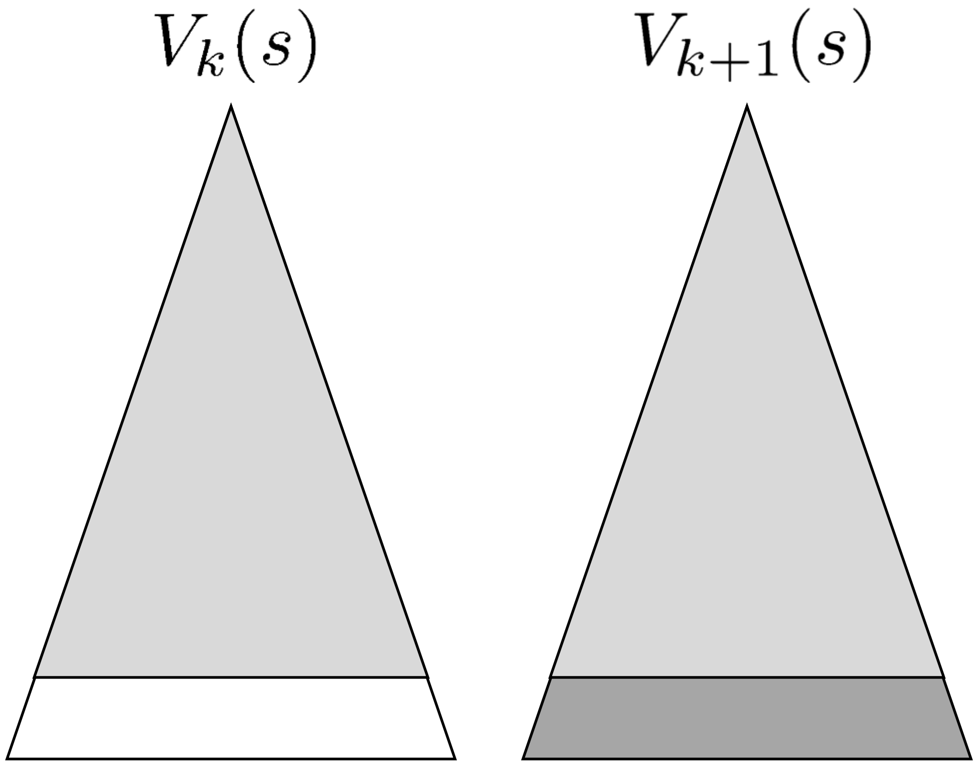
Value Iteration Convergence

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

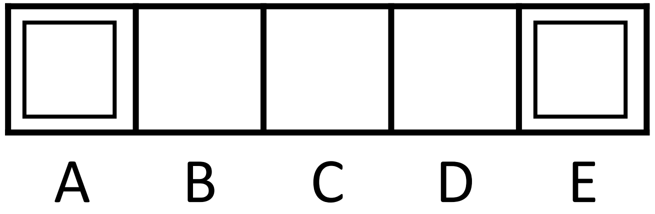
- Sketch: For any state V_k and V_{k+1} can be viewed as depth k+1 expectimax results in nearly identical search trees
- The difference is that on the bottom layer, V_{k+1} has actual rewards while V_k has zeros
- That last layer is at best all R_{MAX}
- It is at worst R_{MIN}
- But everything is discounted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



Outline

MDP Setup

- Expectimax: State, actions, non-deterministic transition functions
- Rewards
  - Walk-through of super-simple value iteration
- Discounting, γ



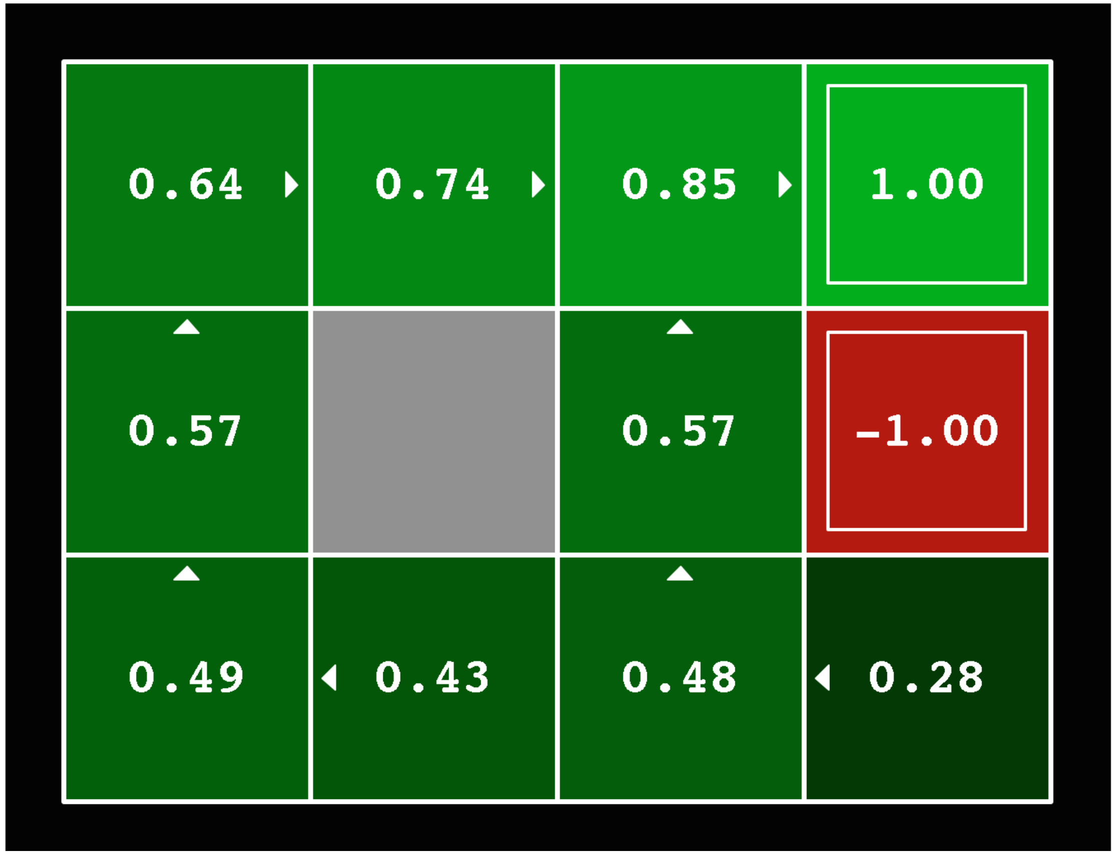
Solving MDPs

- Method 1) Value iteration
  - Value iteration convergence
- Bellman equations
- Policy Extraction
- Method 2) Policy Iteration

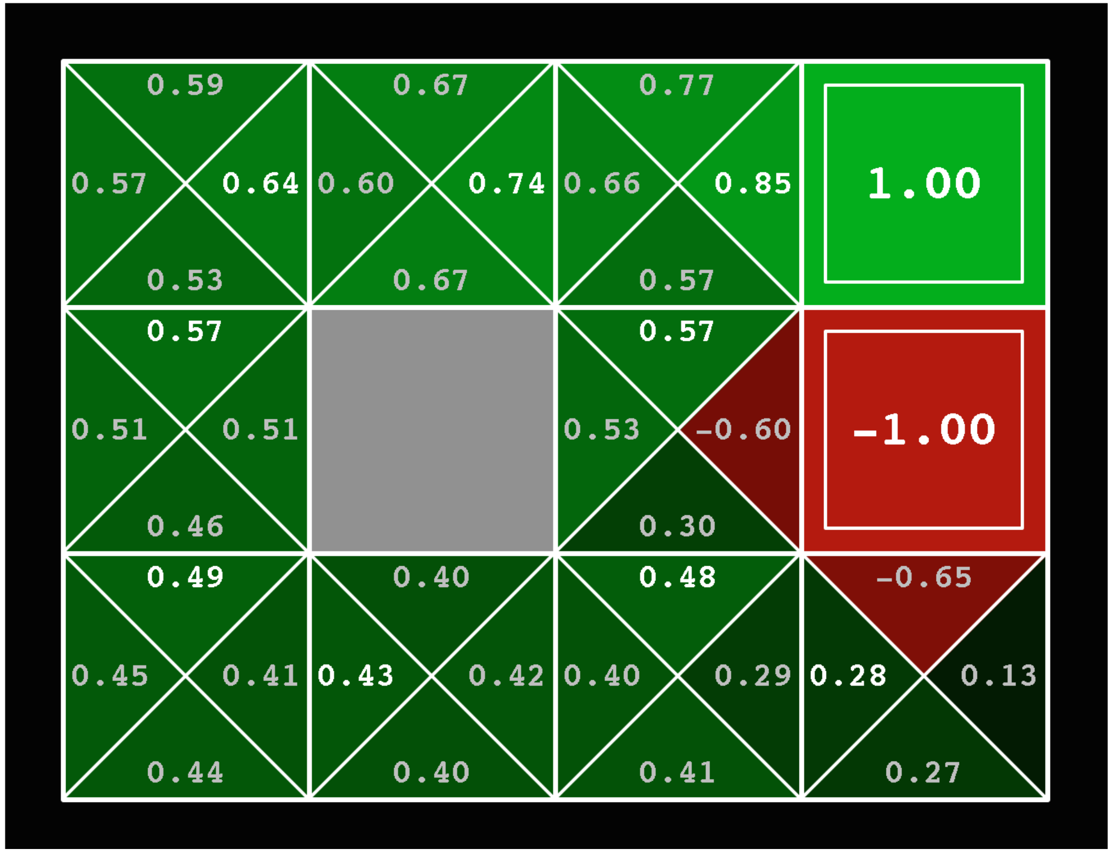
Solved MDP! Now what?

What are we going to do with these values??

$V^*(s)$



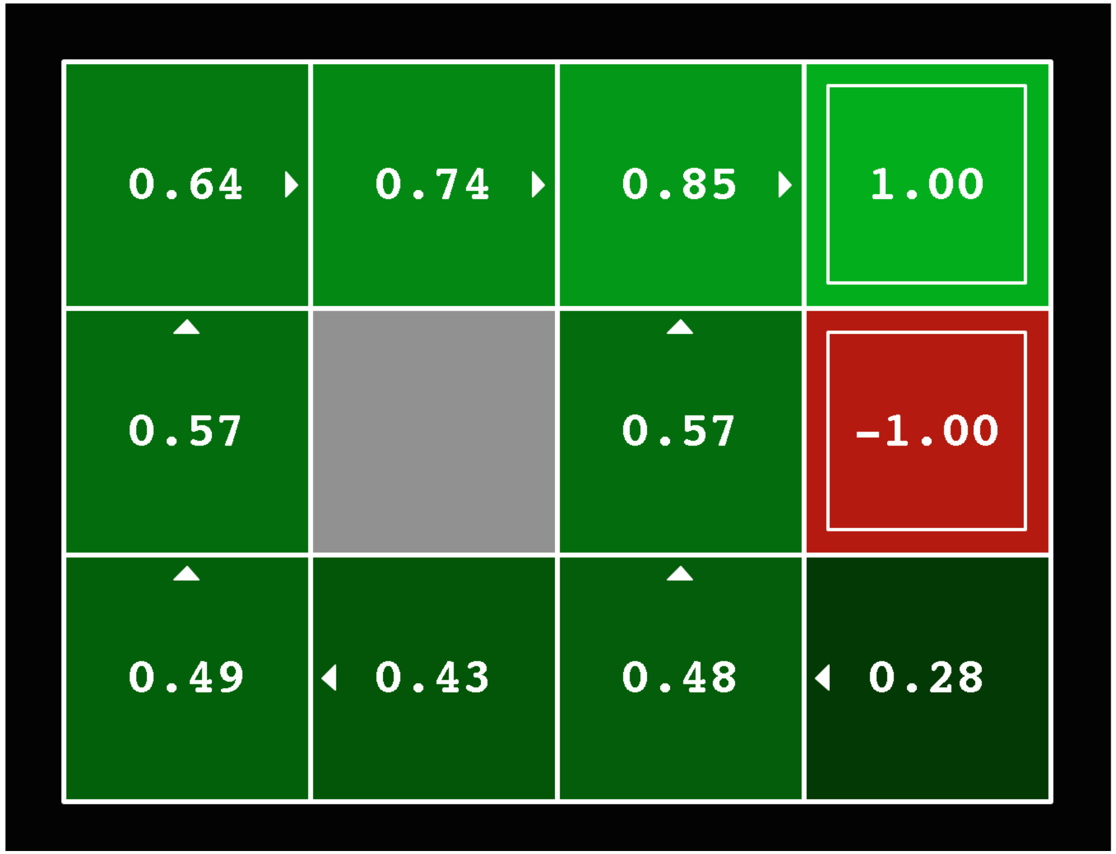
$Q^*(s,a)$

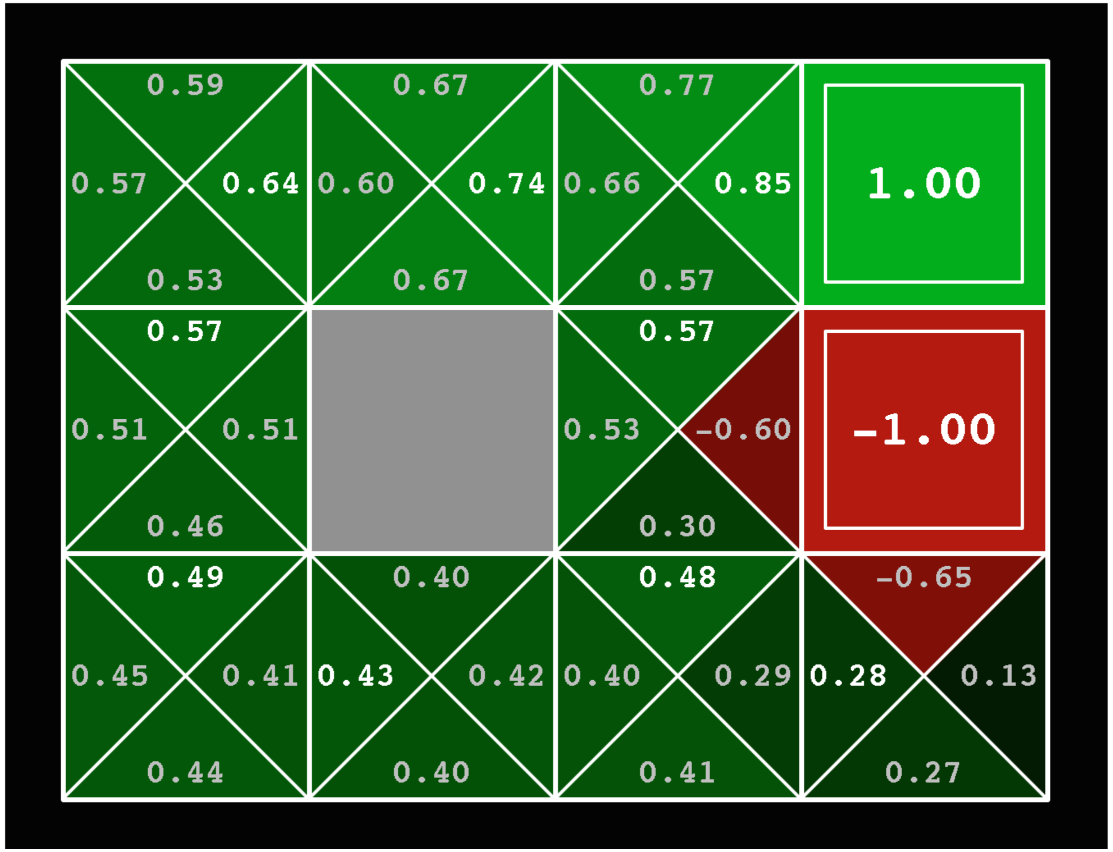


Poll 2

If you need to extract a policy, would you rather have

A) Values, B) Q-values?

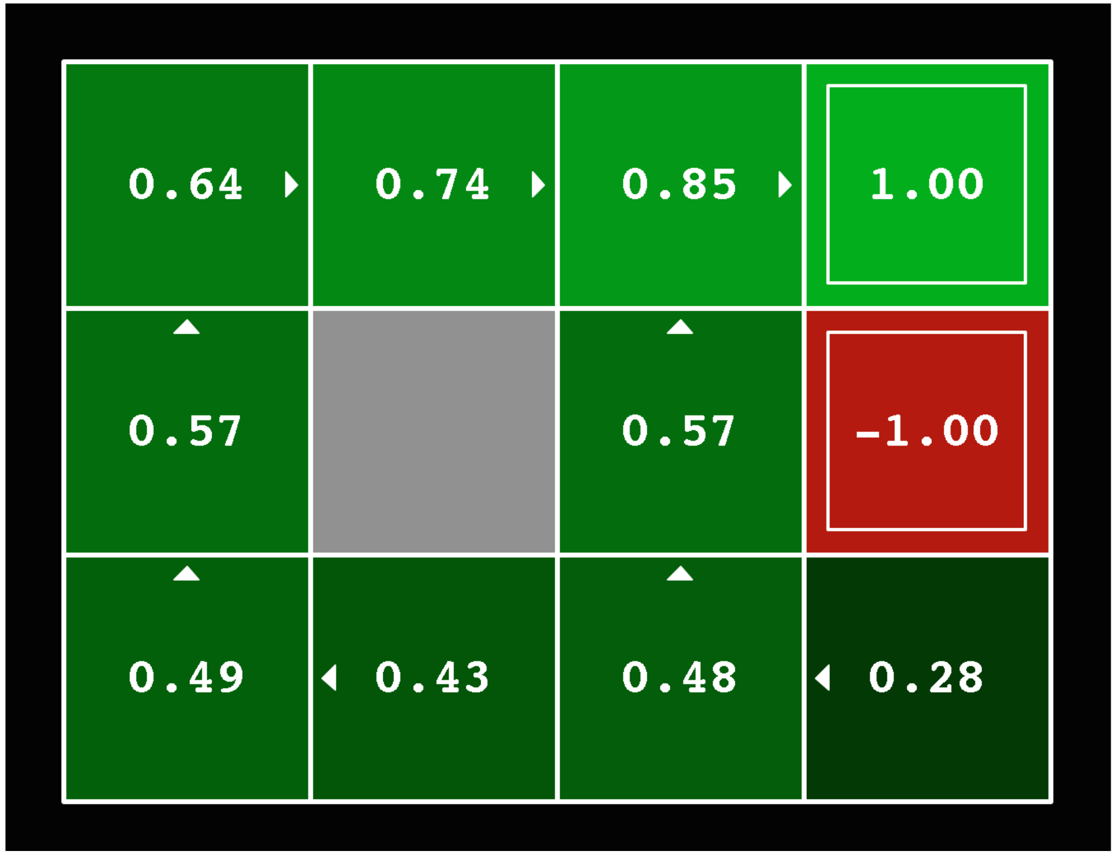


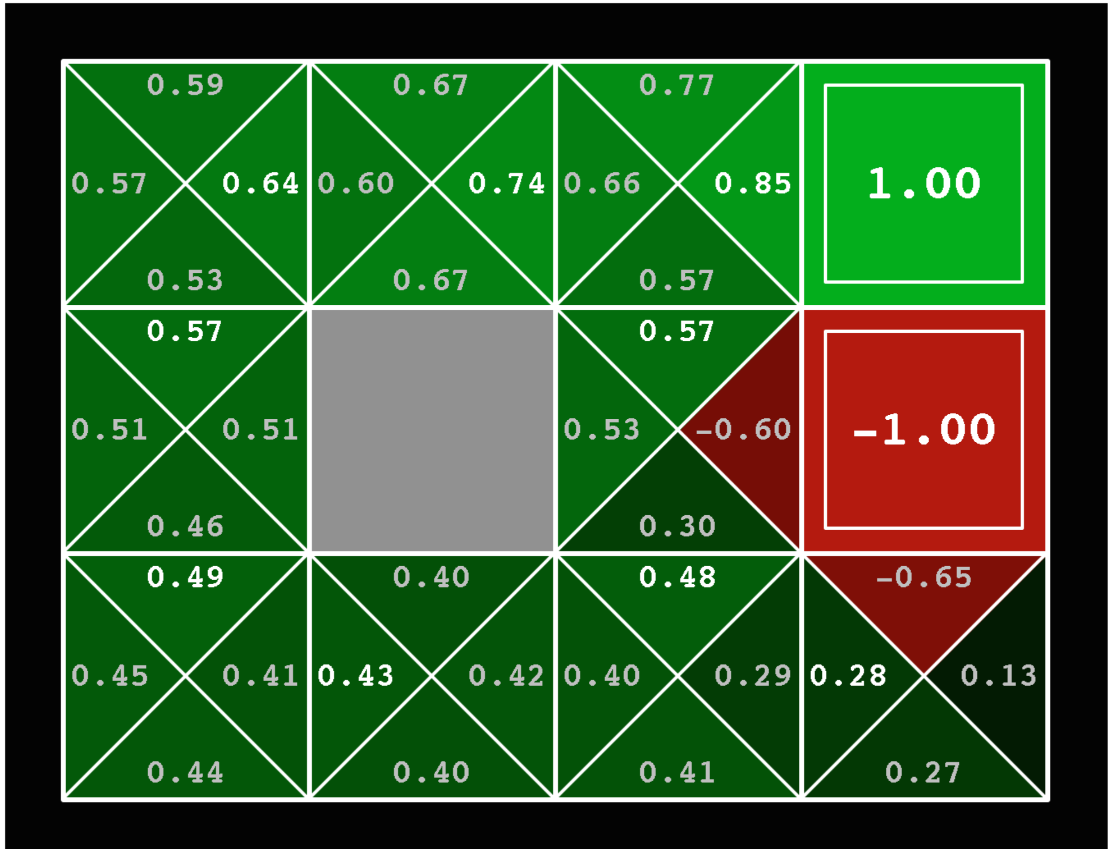


Poll 2

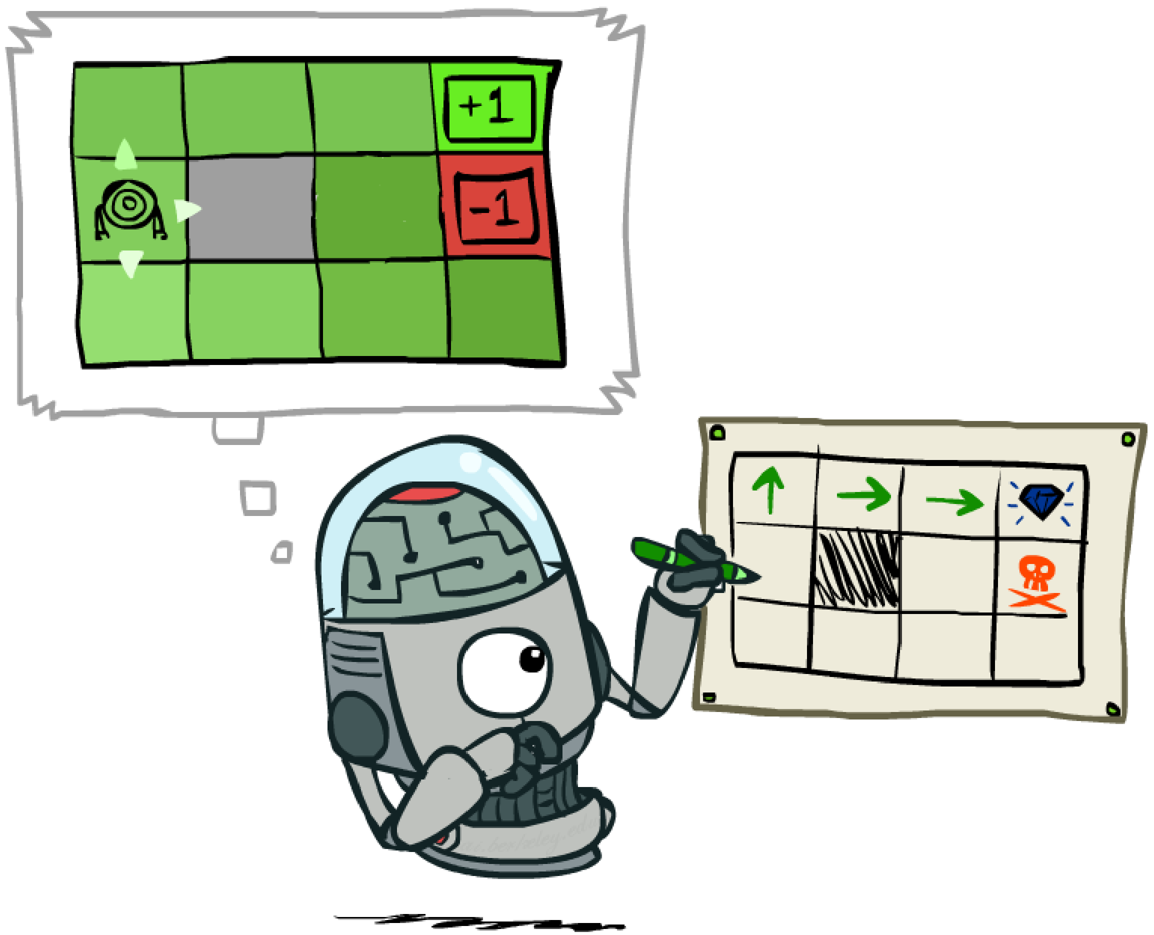
If you need to extract a policy, would you rather have

A) Values, B) Q-values ?



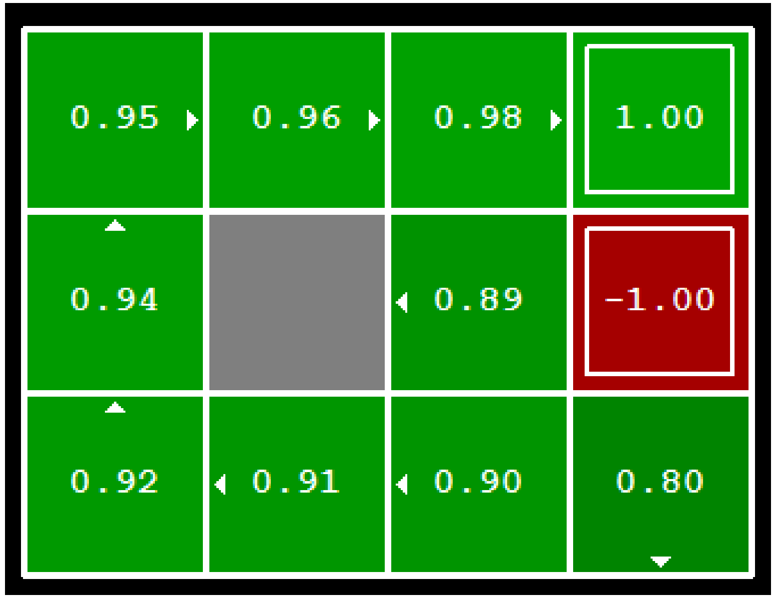


Policy Extraction



Computing Actions from Values

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



How should we act?

- It's not obvious!

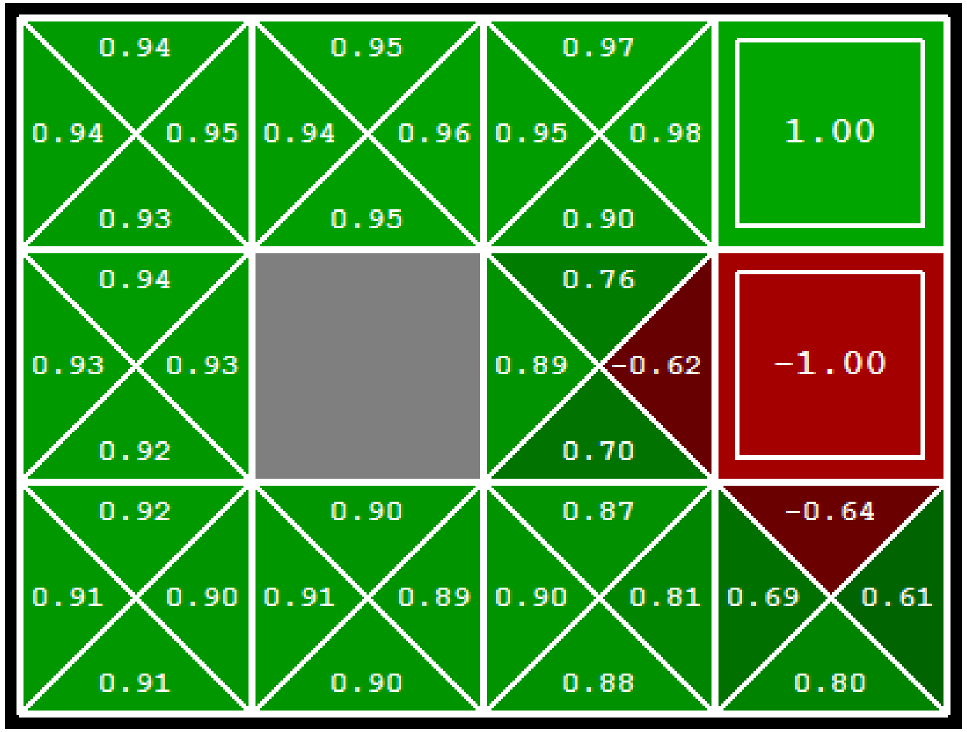
We need to do a mini-expectimax (one step)

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

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!

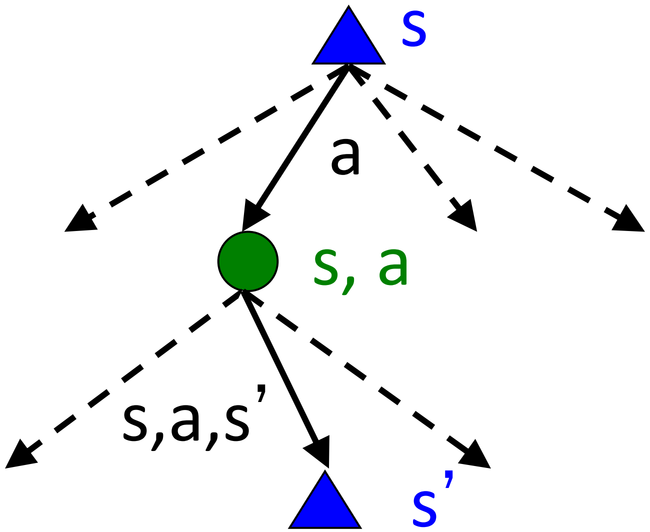
$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

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

Value Iteration Notes

Value iteration repeats the Bellman updates:

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

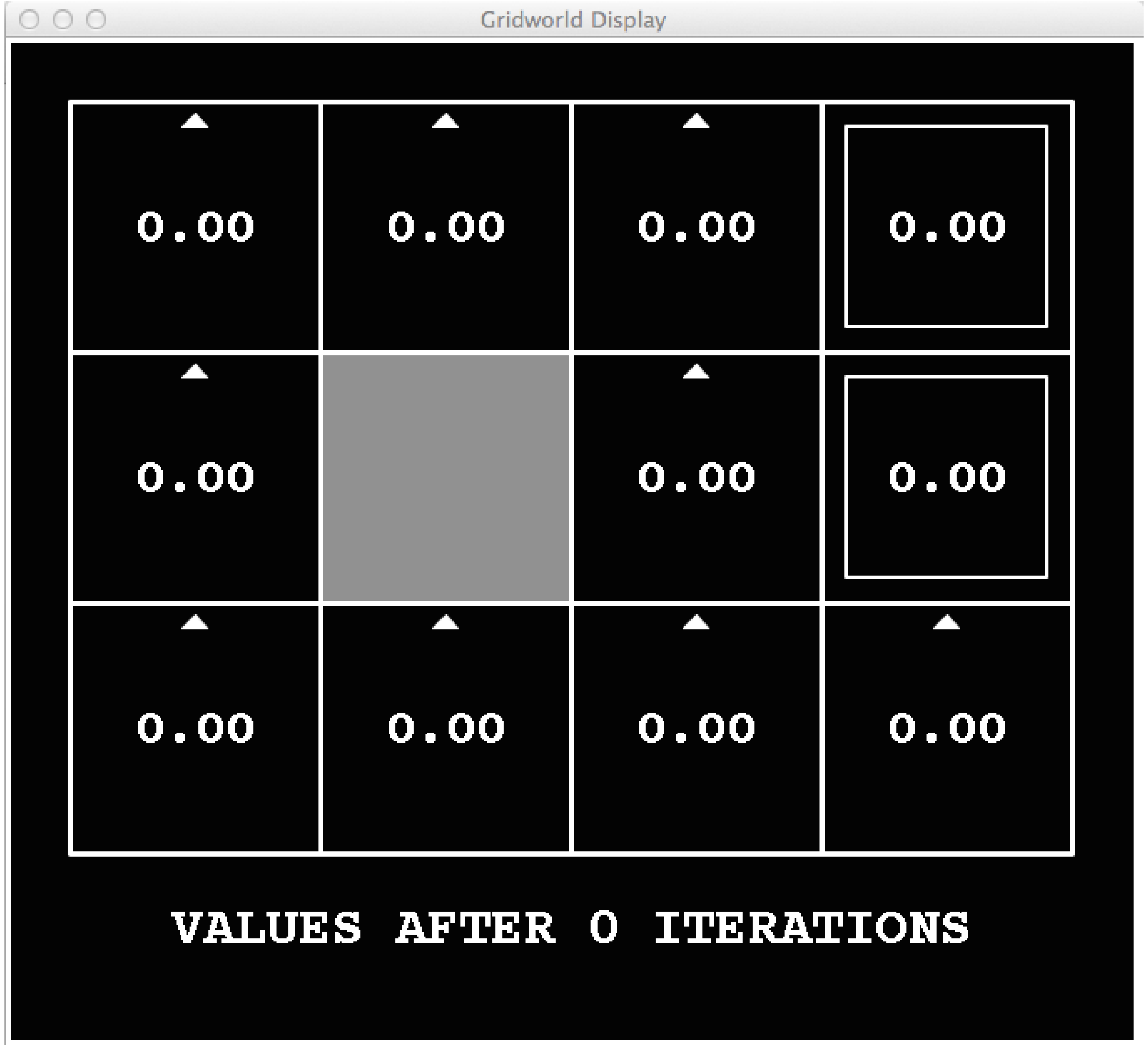


Things to notice when running value iteration:

- It's slow – $O(S^2A)$ per iteration
- The "max" at each state rarely changes
- The optimal policy appears before the values converge (but we don't know that the policy is optimal until the values converge)

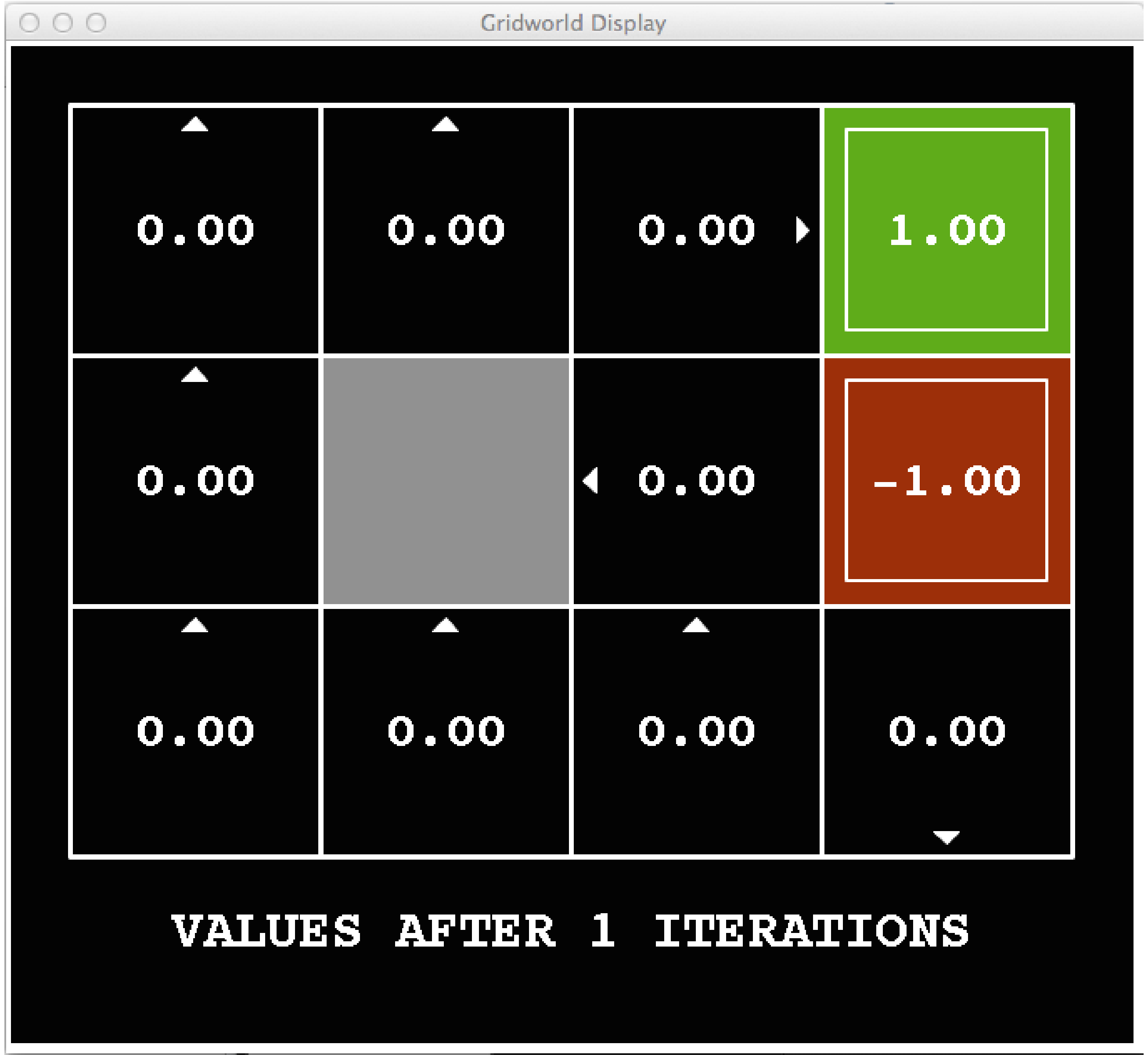
[Demo: value iteration (L9D2)]

k=0



Noise = 0.2
Discount = 0.9
Living reward = 0

k=1



Noise = 0.2
Discount = 0.9
Living reward = 0

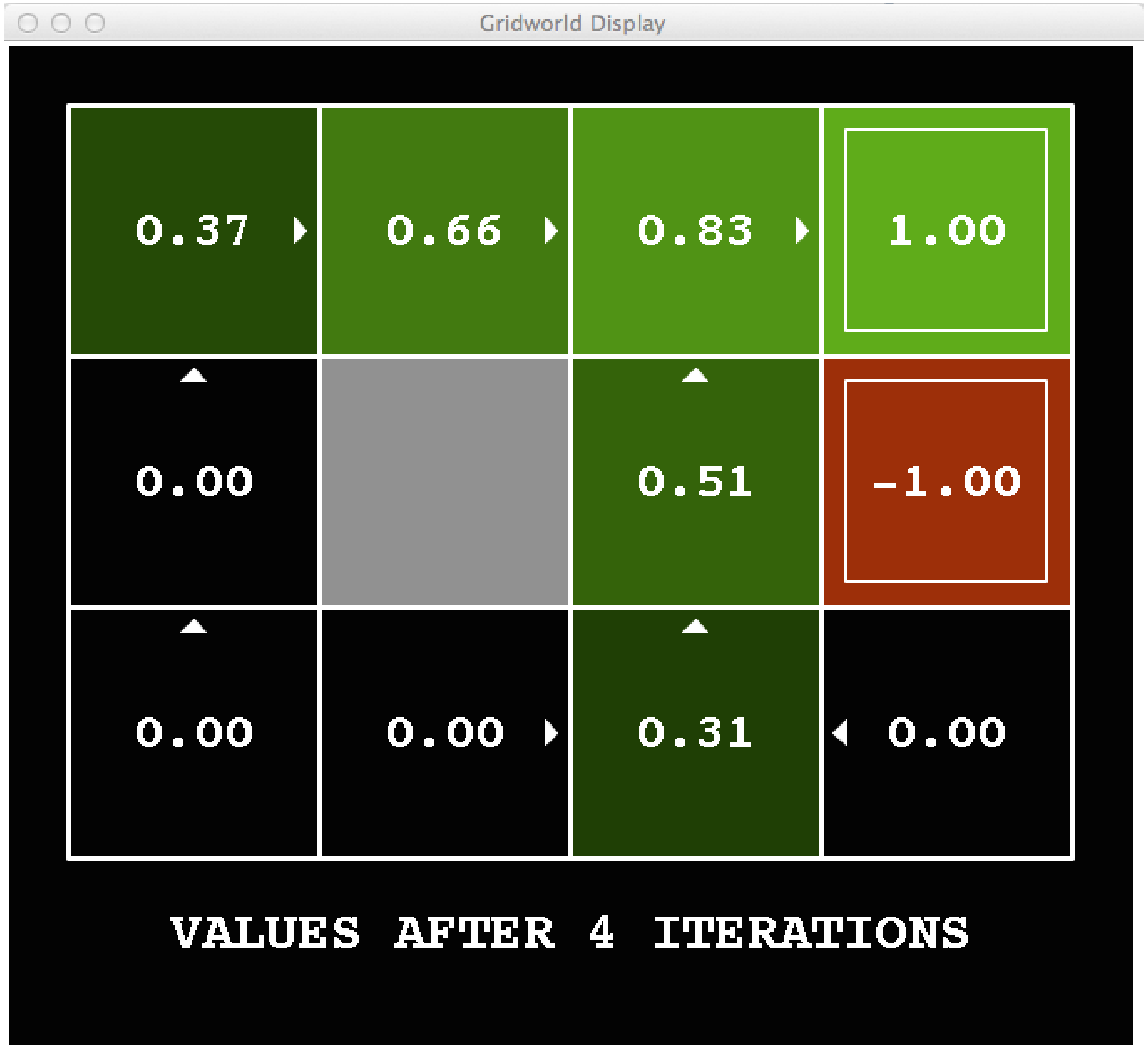
k=2

Noise = 0.2
Discount = 0.9
Living reward = 0

k=3

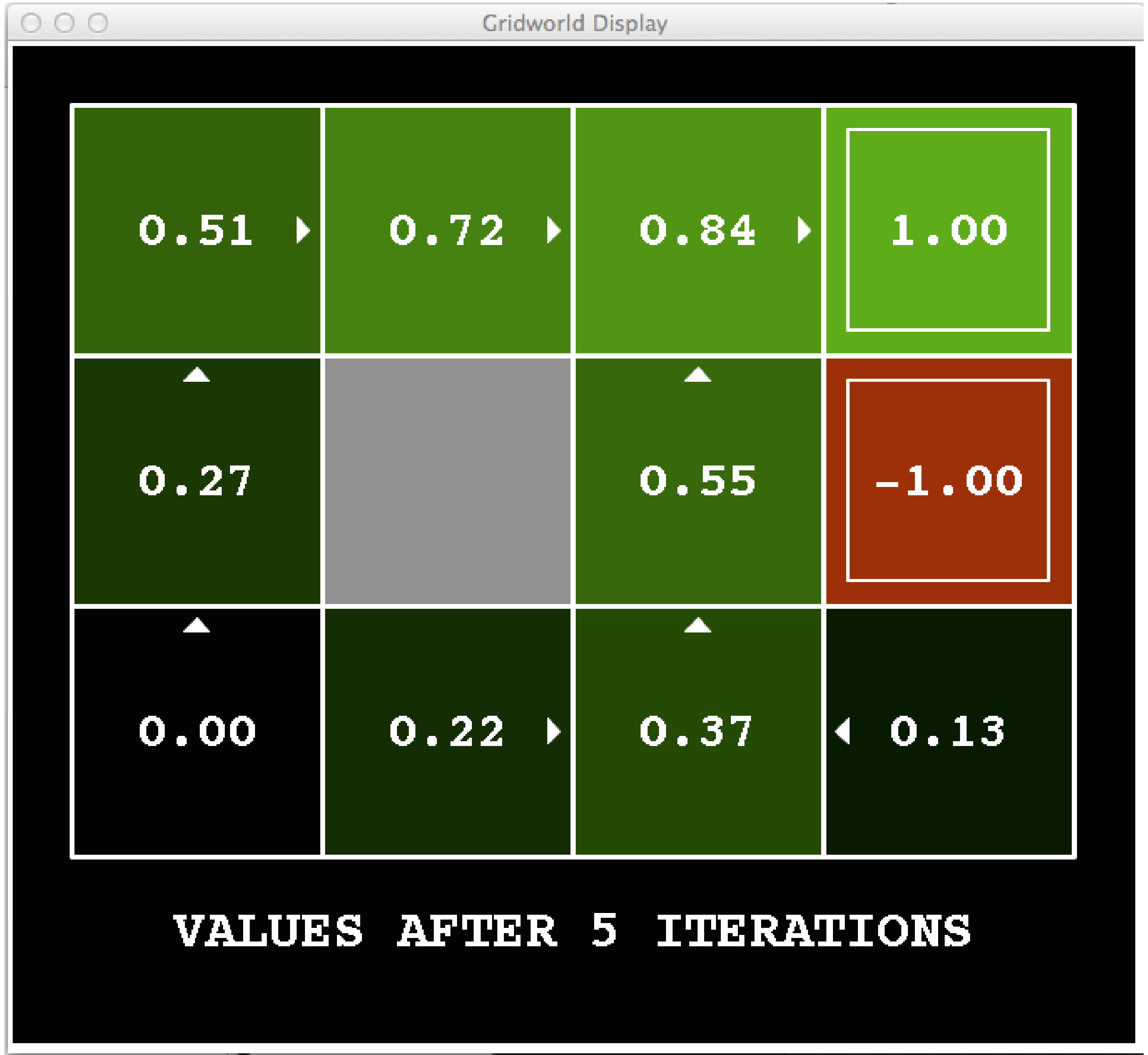
Noise = 0.2
Discount = 0.9
Living reward = 0

k=4



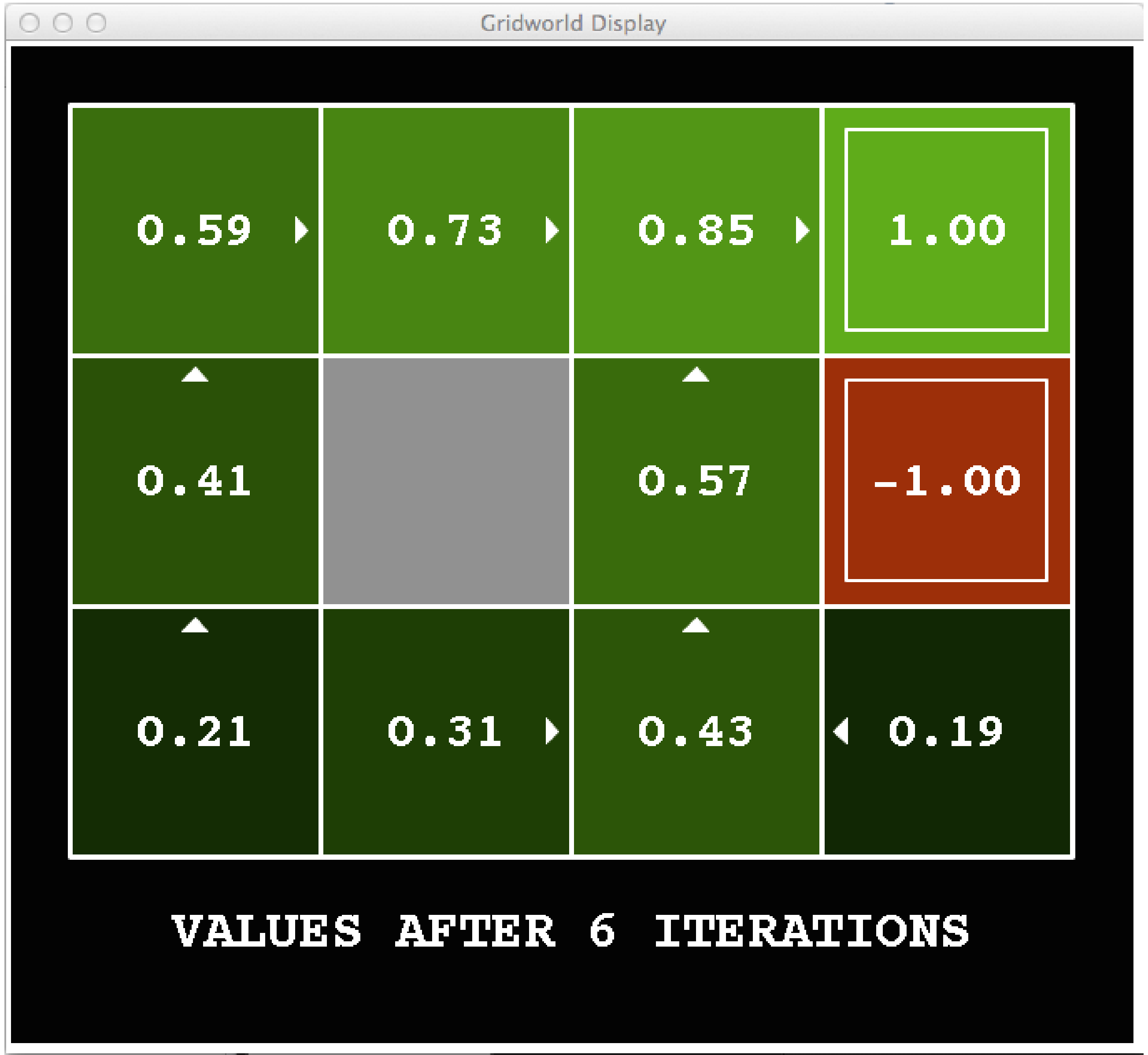
Noise = 0.2
Discount = 0.9
Living reward = 0

k=5



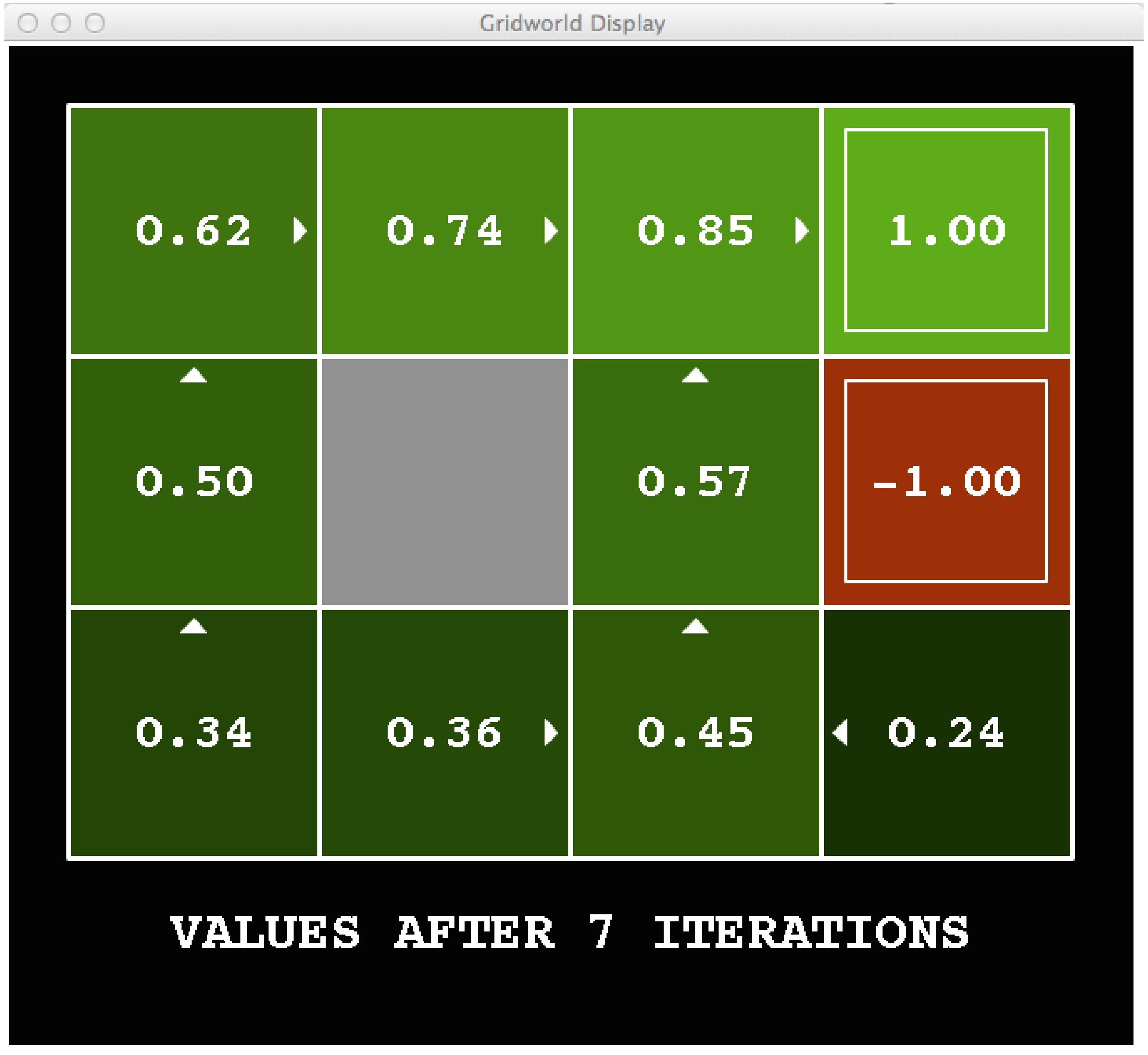
Noise = 0.2
Discount = 0.9
Living reward = 0

k=6



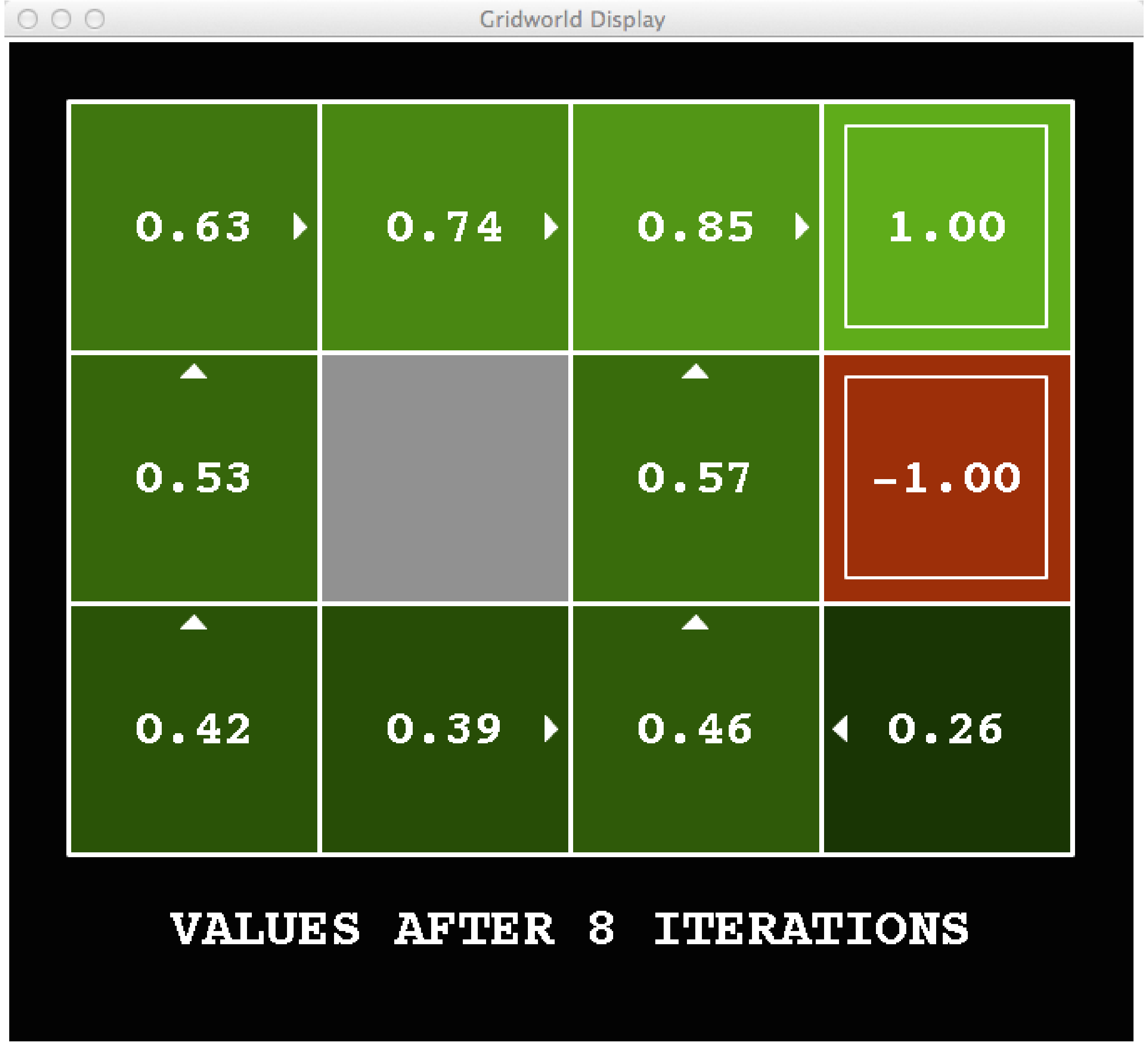
Noise = 0.2
Discount = 0.9
Living reward = 0

k=7



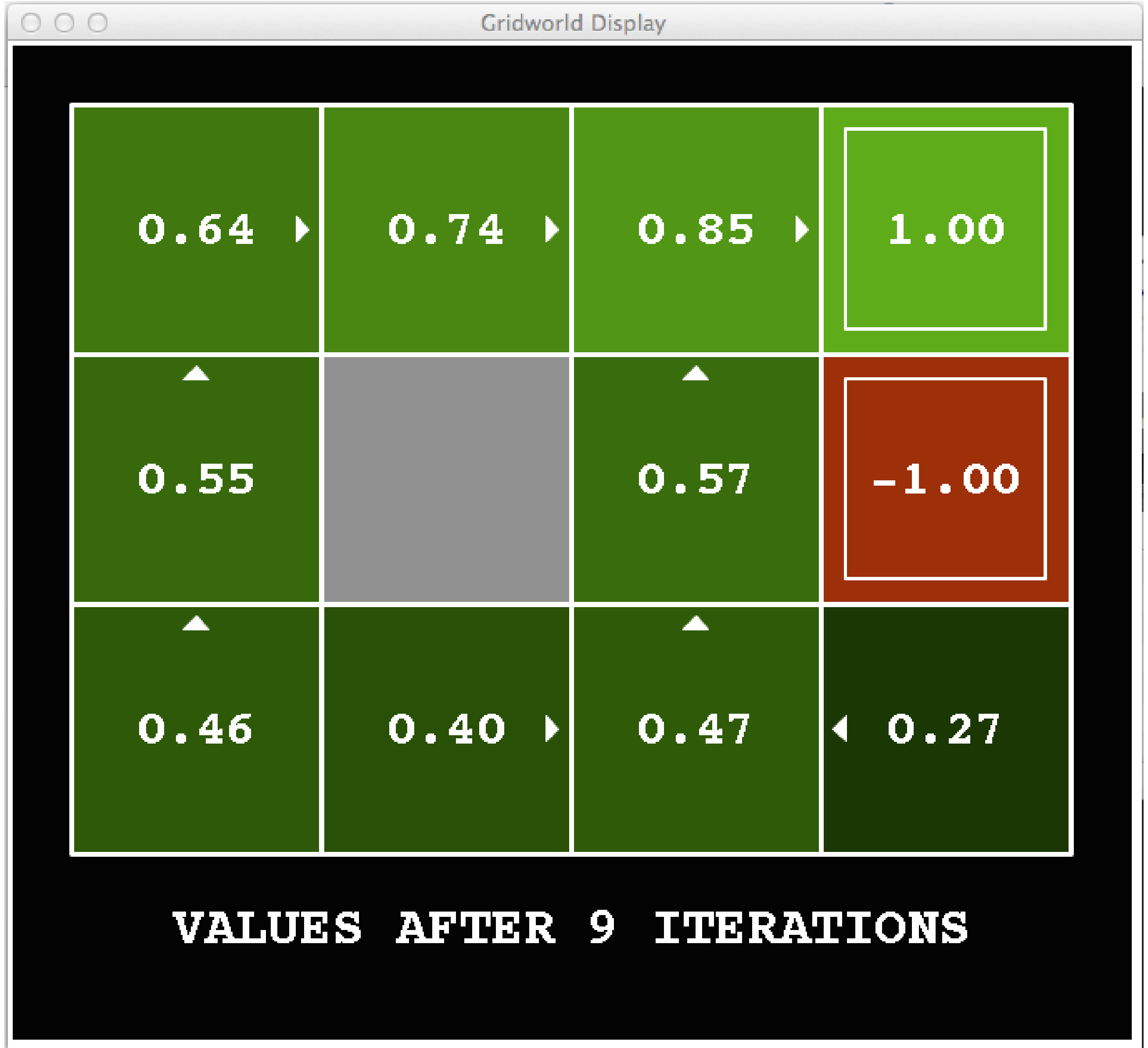
Noise = 0.2
Discount = 0.9
Living reward = 0

k=8



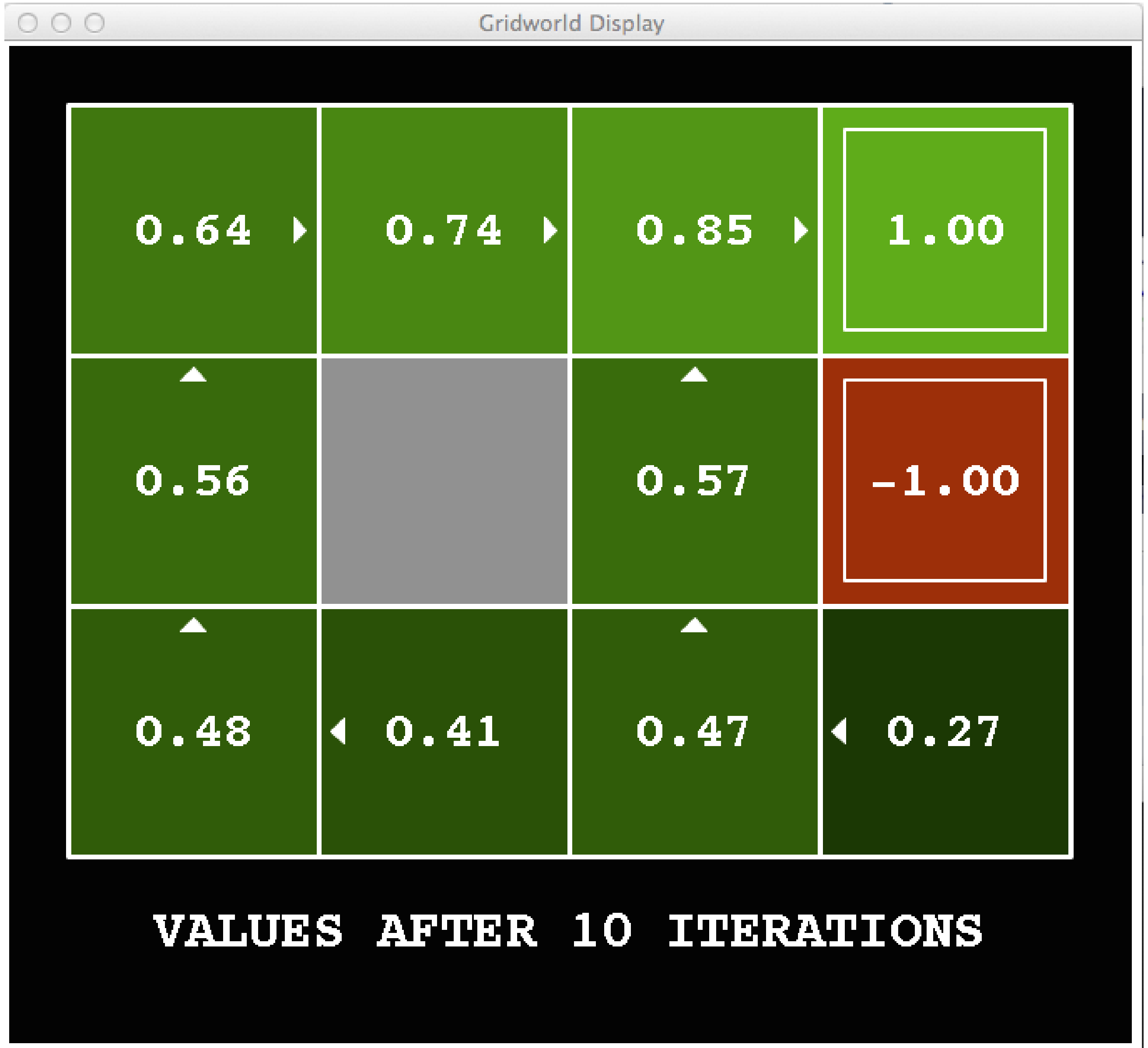
Noise = 0.2
Discount = 0.9
Living reward = 0

k=9



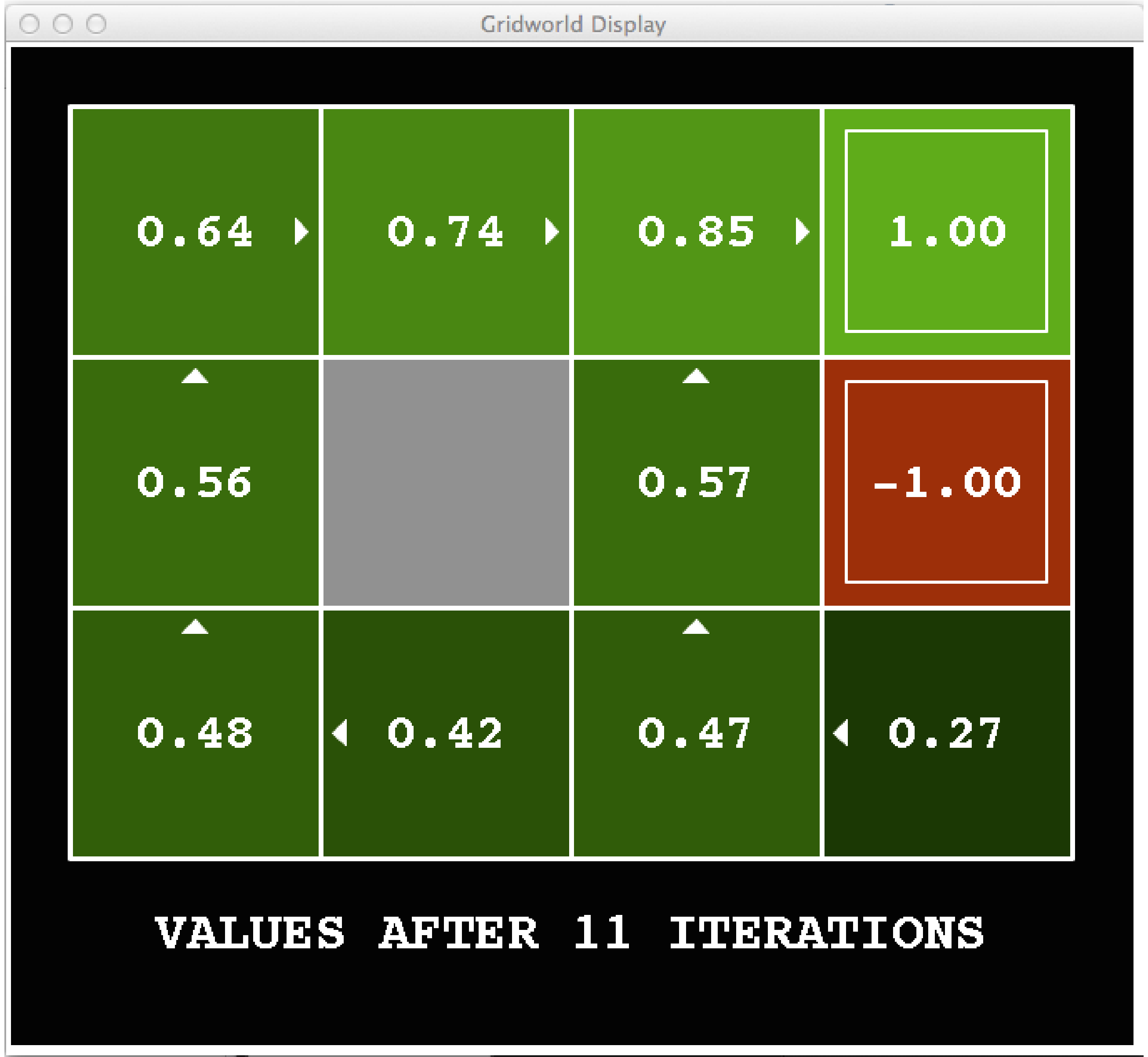
Noise = 0.2
Discount = 0.9
Living reward = 0

k=10



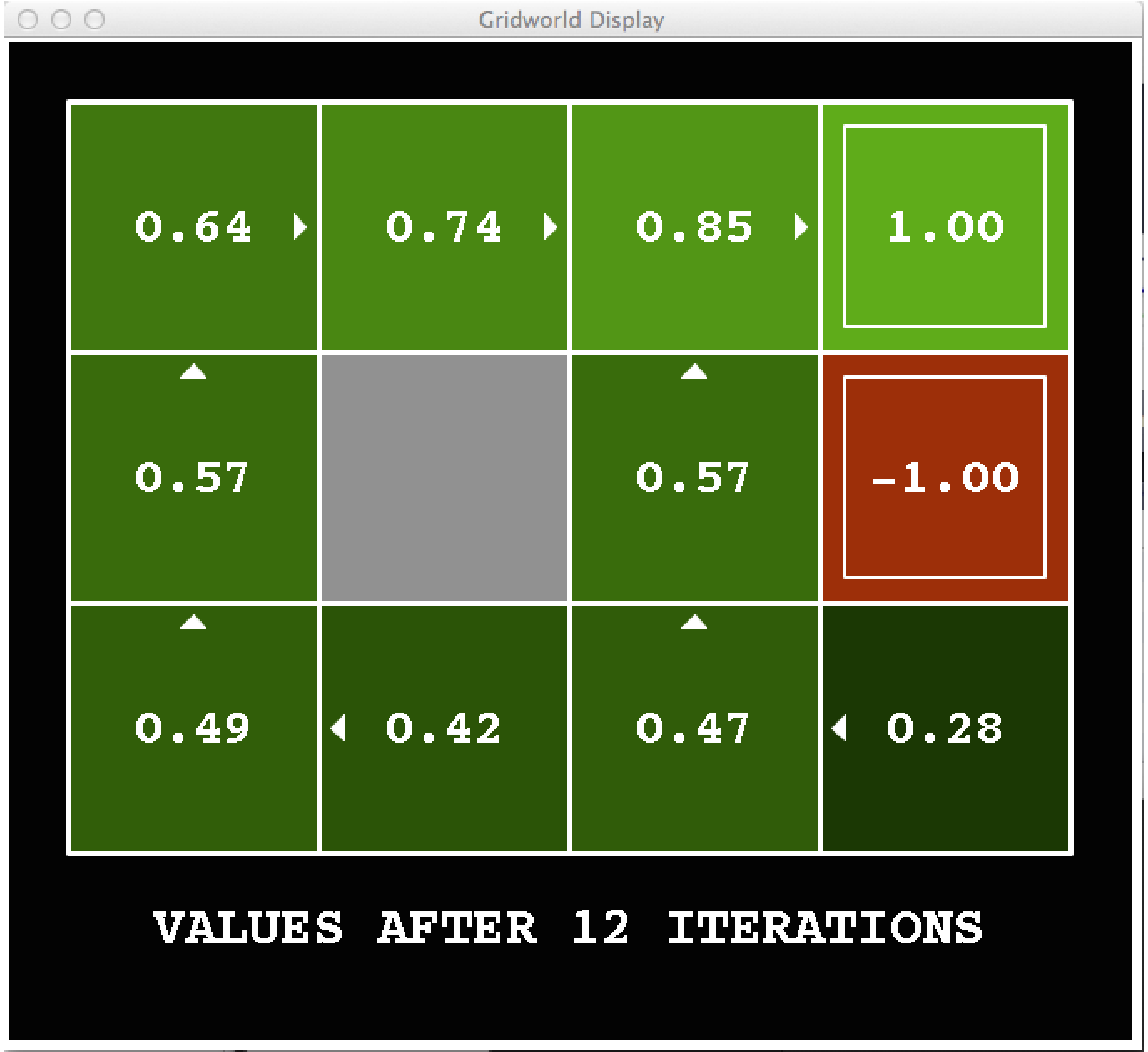
Noise = 0.2
Discount = 0.9
Living reward = 0

k=11



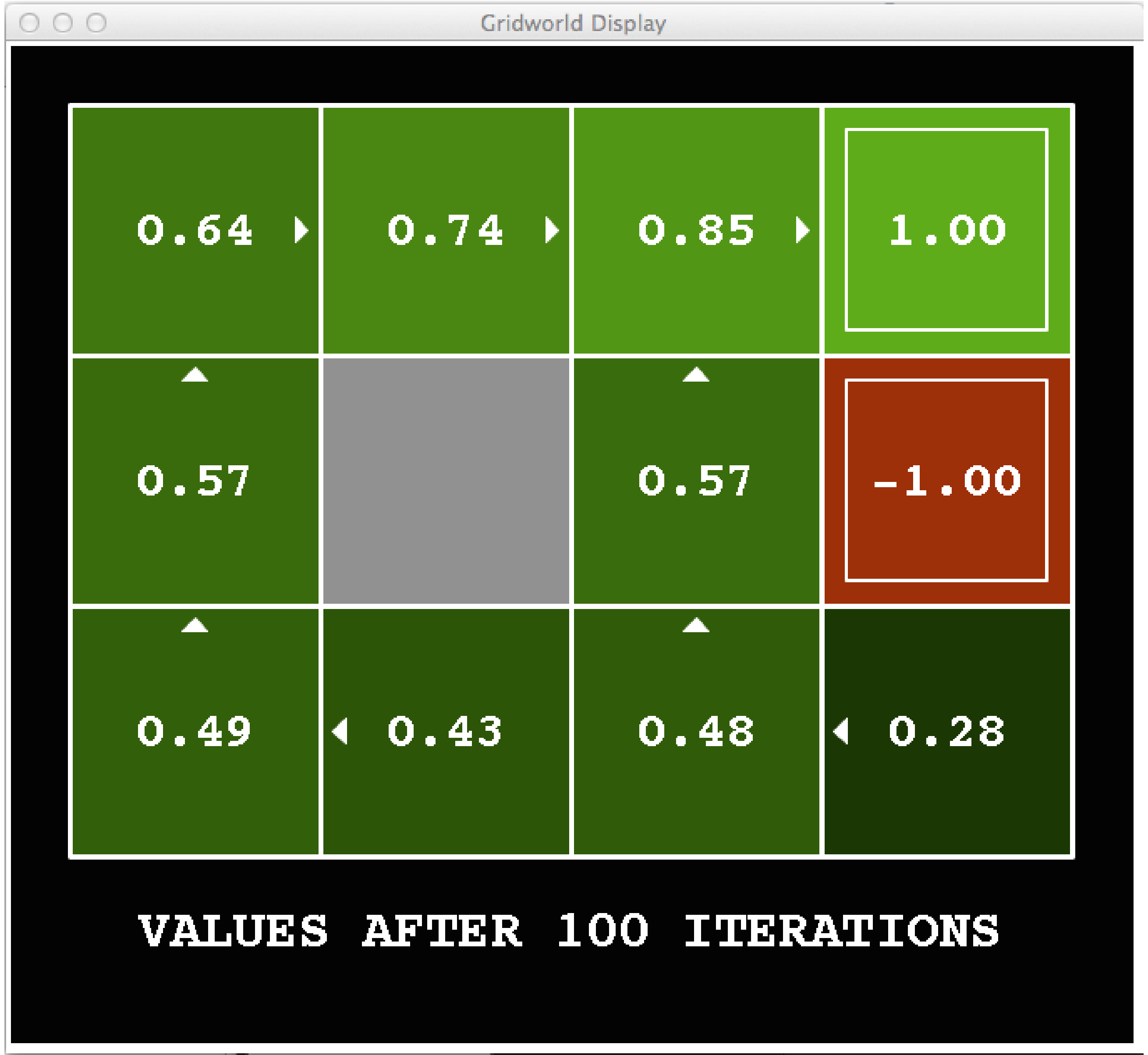
Noise = 0.2
Discount = 0.9
Living reward = 0

k=12



Noise = 0.2
Discount = 0.9
Living reward = 0

k=100

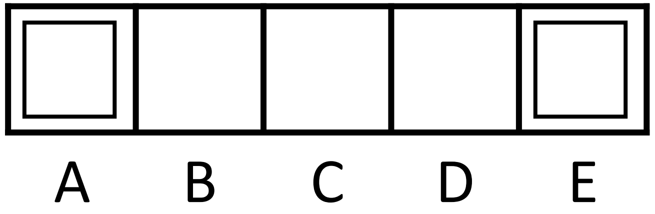


Noise = 0.2
Discount = 0.9
Living reward = 0

Outline

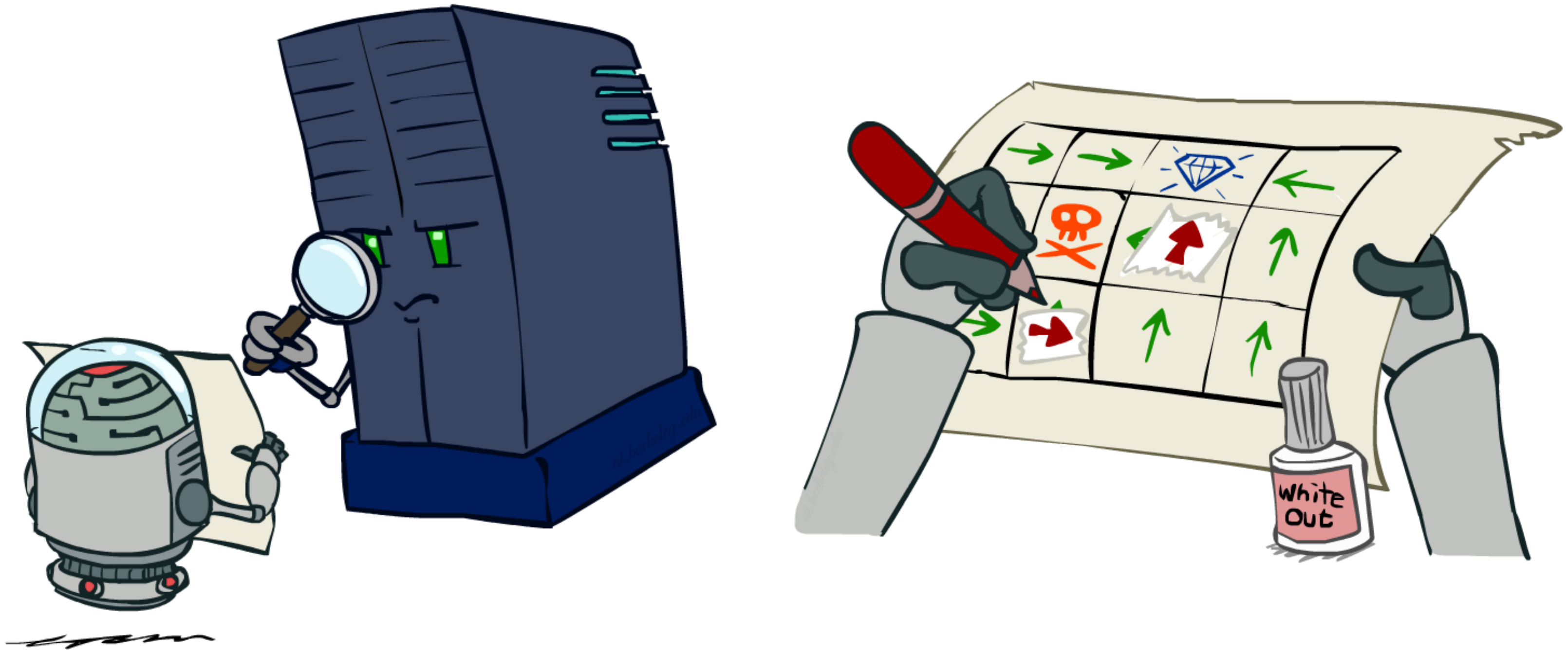
MDP Setup

- Expectimax: State, actions, non-deterministic transition functions
- Rewards
  - Walk-through of super-simple value iteration
- Discounting, γ



Solving MDPs

- Method 1) Value iteration
  - Value iteration convergence
- Bellman equations
- Policy Extraction
- Method 2) Policy Iteration



Policy Iteration

Two Methods for Solving MDPs

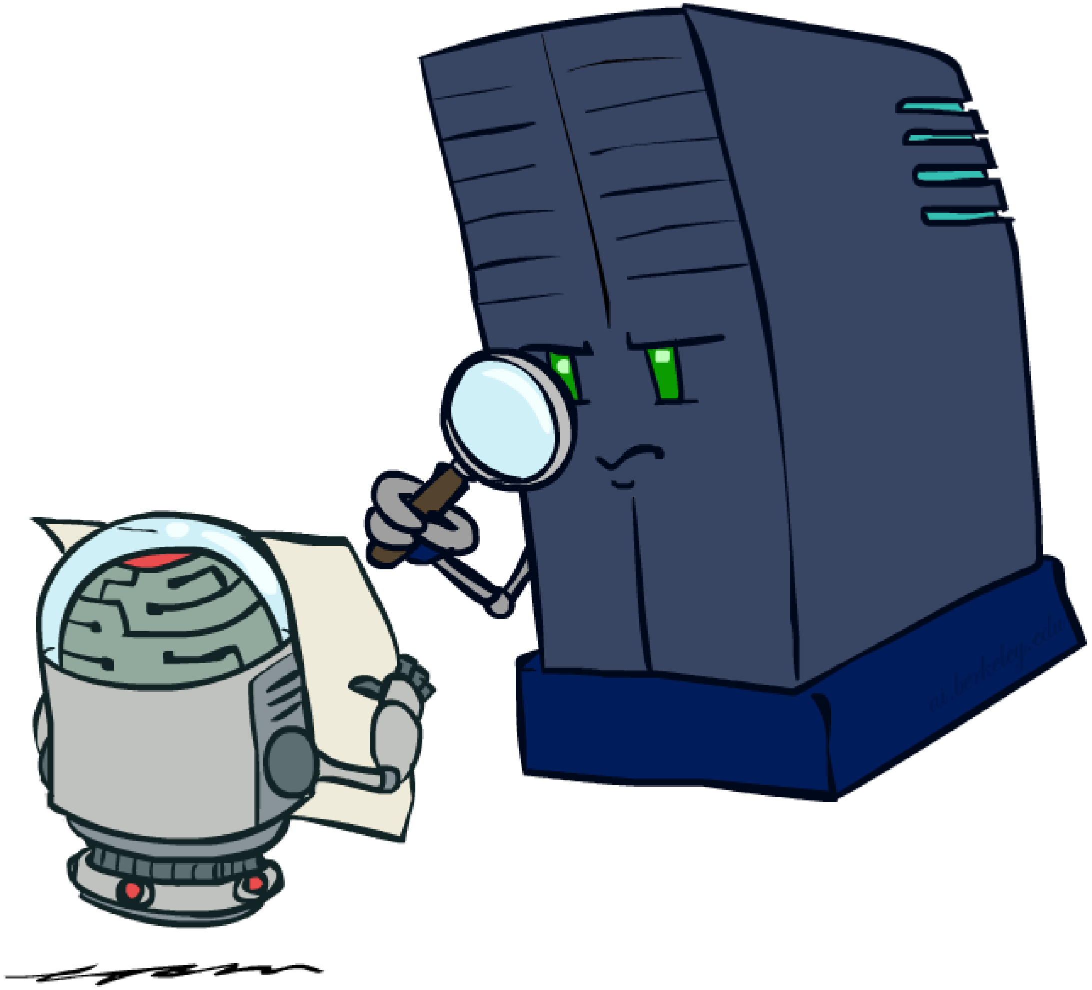
Value iteration + policy extraction

- Step 1: Value iteration: calculate values for all states by running one ply of the Bellman equations using values from previous iteration **until convergence**
- Step 2: Policy extraction: compute policy by running one ply of the Bellman equations using values from value iteration

Policy iteration

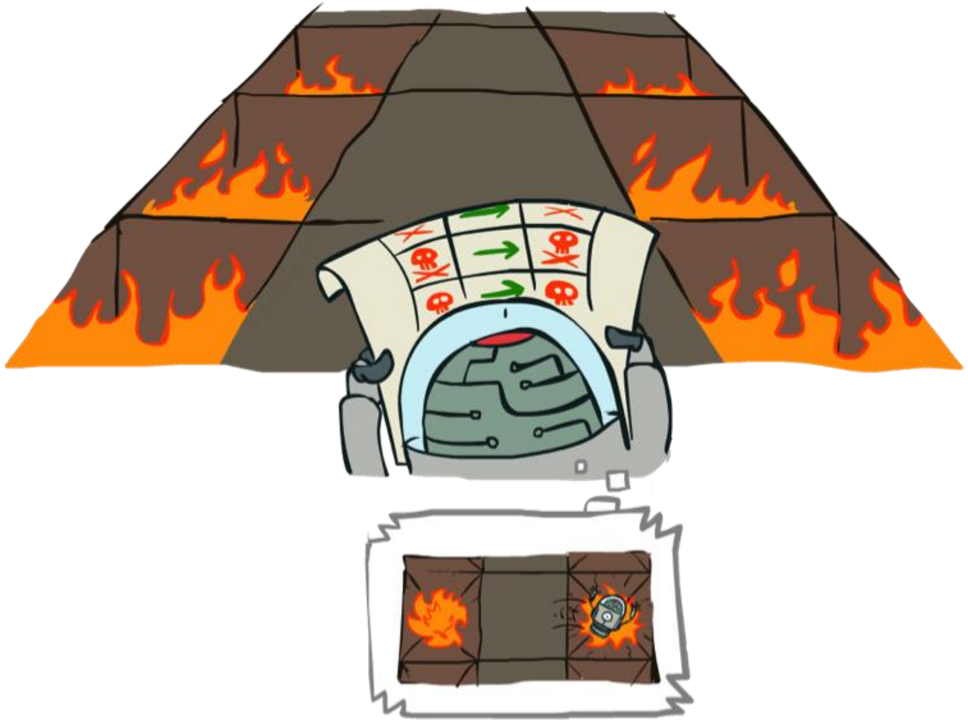
- Step 1: Policy evaluation: calculate values for some fixed policy (not optimal values!) **until convergence**
- Step 2: Policy improvement: update policy by running one ply of the Bellman equations using values from policy evaluation
- Repeat steps until policy converges

Policy Evaluation

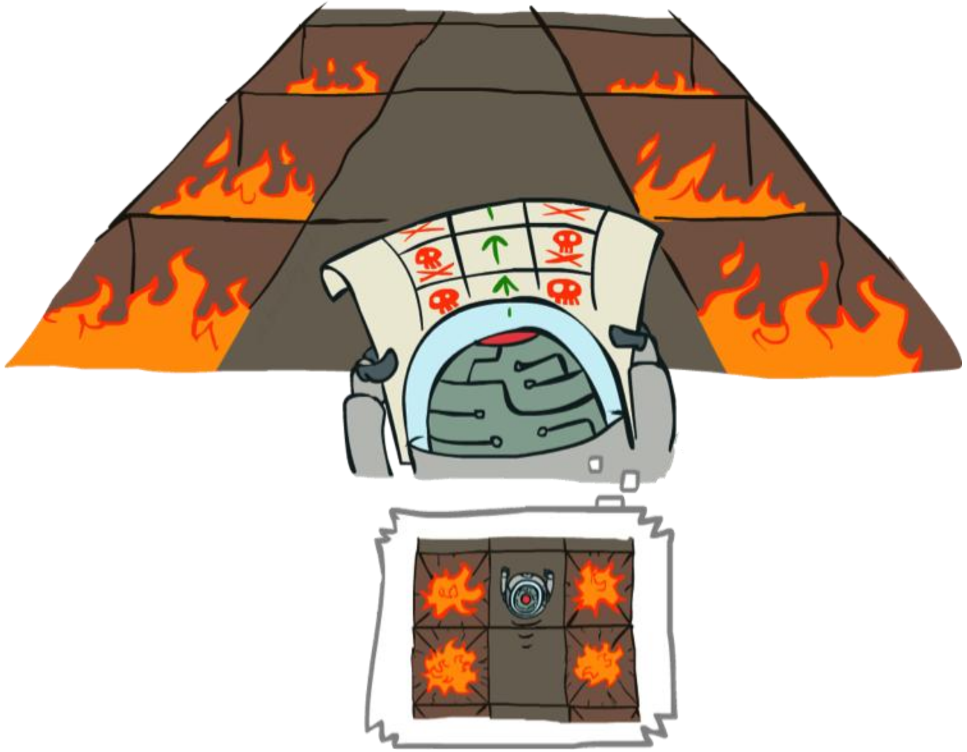


Example: Policy Evaluation

Always Go Right



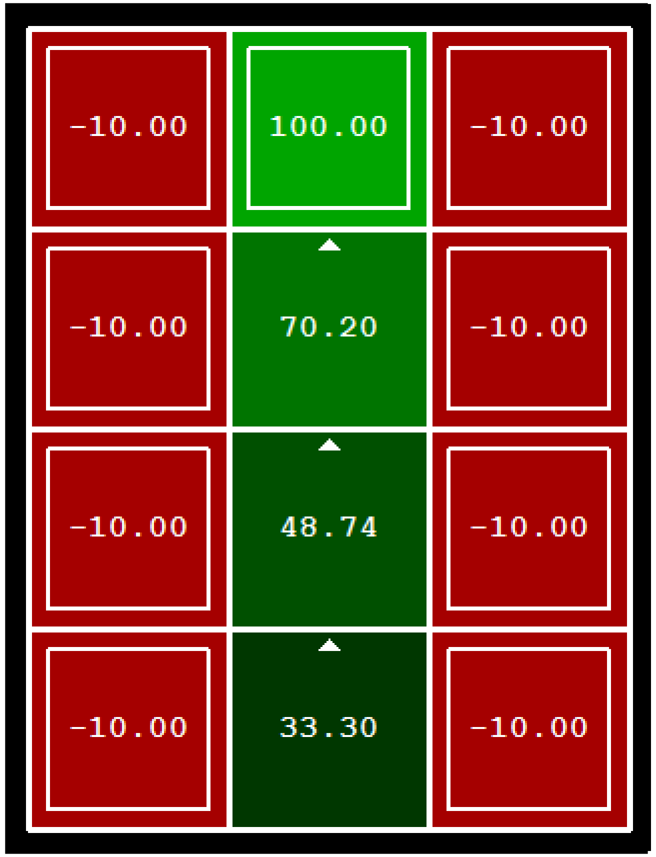
Always Go Forward



Example: Policy Evaluation

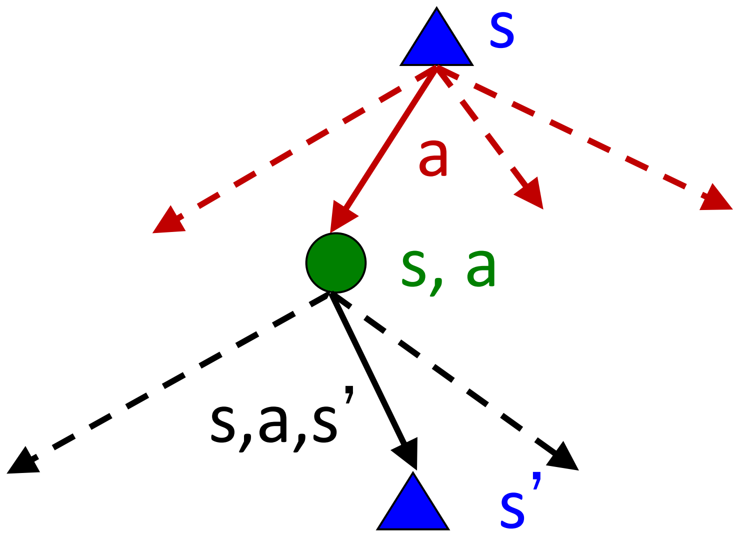
Always Go Right

Always Go Forward

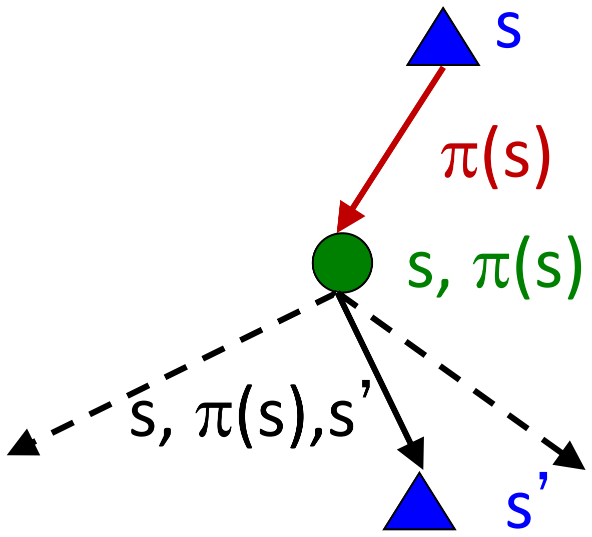


Policy Evaluation: Fixed Policies

Normally: Do the optimal action



Fixed policy: Do what π says to do



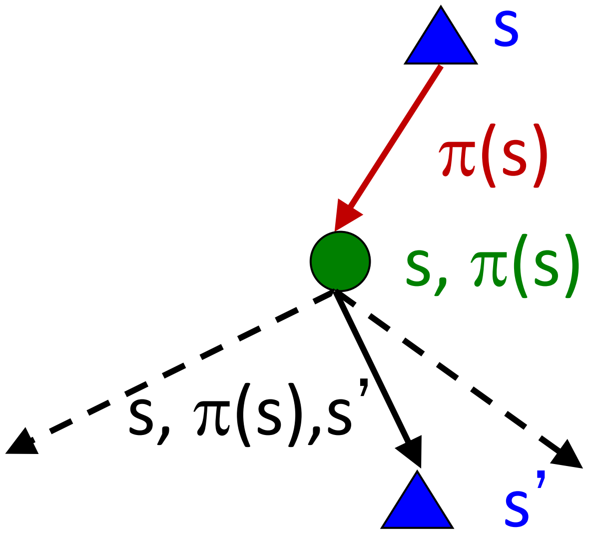
Expectimax trees max over all actions to compute the optimal values

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

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

Policy Evaluation: Utilities for a Fixed Policy

Another basic operation: compute the utility value 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 sum of discounted rewards starting in s and following π

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

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

Policy Evaluation

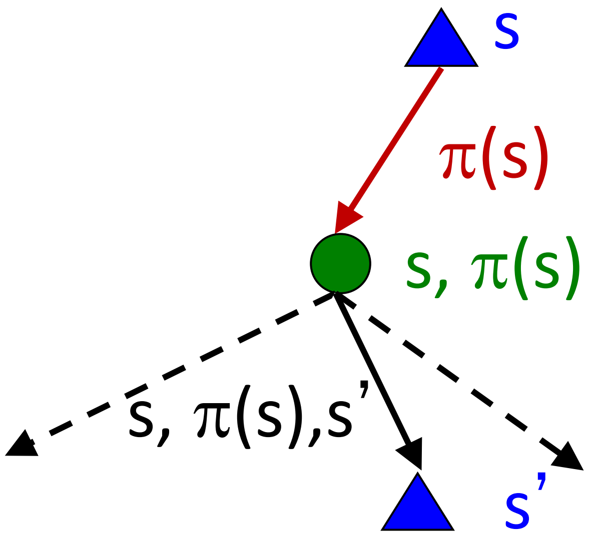
How do we calculate the V's for a fixed policy π?

Idea 1: Turn recursive Bellman equations into updates

(like value iteration)

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

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

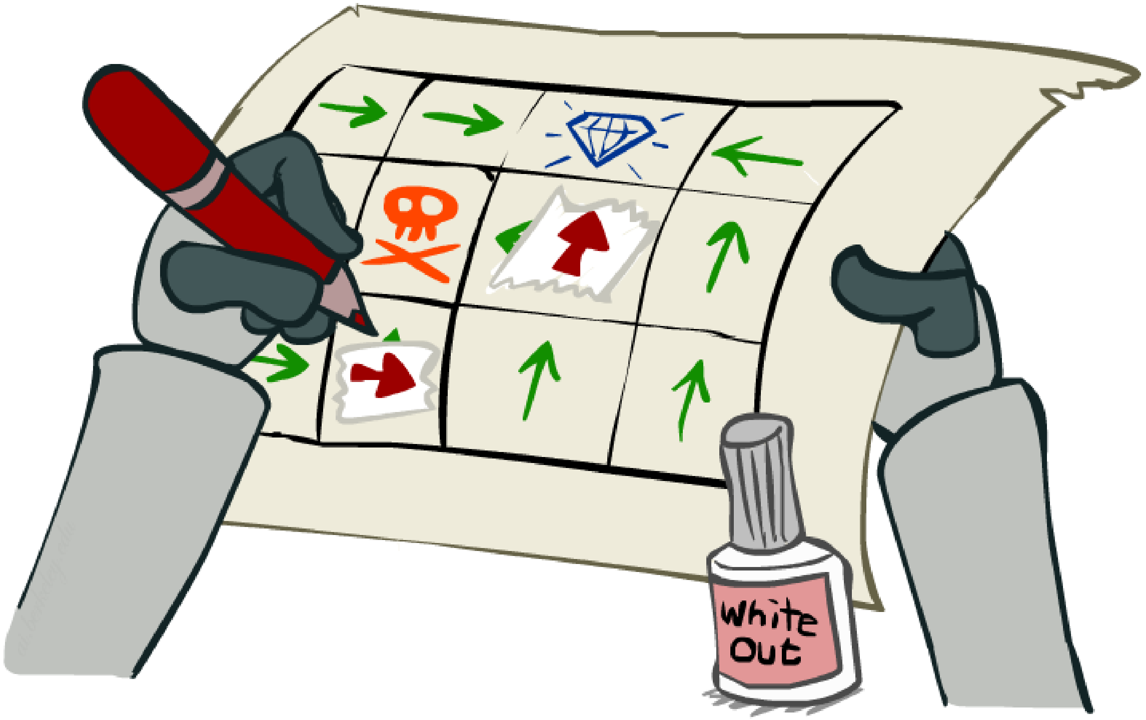


Efficiency: $O(S^2)$ per iteration

Idea 2: Without the maxes, the Bellman equations are just a linear system

- Solve with your favorite linear system solver

Policy Improvement



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:

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

Policy iteration

- It's still optimal!
- Can converge faster under some conditions

Two Methods for Solving MDPs

Value iteration + policy extraction

- Step 1: Value iteration:

$$V_{k+1}(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V_k(s')], \ \forall\, s$$ **until convergence**

- Step 2: Policy extraction:

$$\pi_V(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')], \ \forall\, s$$

Policy iteration

- Step 1: Policy evaluation:

$$V_{k+1}^{\pi}(s) = \sum_{s'} P(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma V_k^{\pi}(s')], \ \forall\, s$$ **until convergence**

- Step 2: Policy improvement:

$$\pi_{new}(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^{\pi_{old}}(s')], \ \forall\, s$$

- 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 values with fixed policy (each pass is fast because we consider only one action, not all of them; however we do **many** passes)
- After the policy is evaluated, a new policy is chosen (with (arg)max like value iteration)
- 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

MDP Notation

Standard expectimax: $V(s) = \max_a \sum_{s'} P(s'|s,a)V(s')$

Bellman equations: $V^*(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^*(s')]$

Value iteration: $V_{k+1}(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V_k(s')], \quad \forall s$

Q-iteration: $Q_{k+1}(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')], \quad \forall s,a$

Policy extraction: $\pi_V(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')], \quad \forall s$

Policy evaluation: $V_{k+1}^{\pi}(s) = \sum_{s'} P(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma V_k^{\pi}(s')], \quad \forall s$

Policy improvement: $\pi_{new}(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^{\pi_{old}}(s')], \quad \forall s$

MDP Notation

Standard expectimax: $V(s) = \max_a \sum_{s'} P(s'|s,a)V(s')$

Bellman equations: $V^*(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^*(s')]$

Value iteration: $V_{k+1}(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V_k(s')], \quad \forall s$

Q-iteration: $Q_{k+1}(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')], \quad \forall s,a$

Policy extraction: $\pi_V(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')], \quad \forall s$

Policy evaluation: $V^{\pi}_{k+1}(s) = \sum_{s'} P(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma V^{\pi}_k(s')], \quad \forall s$

Policy improvement: $\pi_{new}(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^{\pi_{old}}(s')], \quad \forall s$

MDP Notation

Standard expectimax: $$V(s) = \max_a \sum_{s'} P(s'|s,a)V(s')$$

Bellman equations: $$V^*(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^*(s')]$$

Value iteration: $$V_{k+1}(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V_k(s')], \quad \forall\, s$$

Q-iteration: $$Q_{k+1}(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')], \quad \forall\, s,a$$

Policy extraction: $$\pi_V(s) = \operatorname*{argmax}_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')], \quad \forall\, s$$

Policy evaluation: $$V_{k+1}^{\pi}(s) = \sum_{s'} P(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma V_k^{\pi}(s')], \quad \forall\, s$$

Policy improvement: $$\pi_{new}(s) = \operatorname*{argmax}_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^{\pi_{old}}(s')], \quad \forall\, s$$

MDP Notation

Standard expectimax: $V(s) = \max_a \sum_{s'} P(s'|s,a)V(s')$

Bellman equations: $V^*(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^*(s')]$

Value iteration: $V_{k+1}(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V_k(s')], \quad \forall s$

Q-iteration: $Q_{k+1}(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')], \quad \forall s,a$

Policy extraction: $\pi_V(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')], \quad \forall s$

Policy evaluation: $V_{k+1}^{\pi}(s) = \sum_{s'} P(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma V_k^{\pi}(s')], \quad \forall s$

Policy improvement: $\pi_{new}(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^{\pi_{old}}(s')], \quad \forall s$