Announcements

Assignments

- HW7: Thu, 11/19, 11:59 pm

Schedule change

- Friday: Lecture in all three recitation slots
- Monday: Recitation in both lecture slots

Final exam scheduled

Study groups

Introduction to Machine Learning

Markov Decision Processes

Instructor: Pat Virtue

Plan

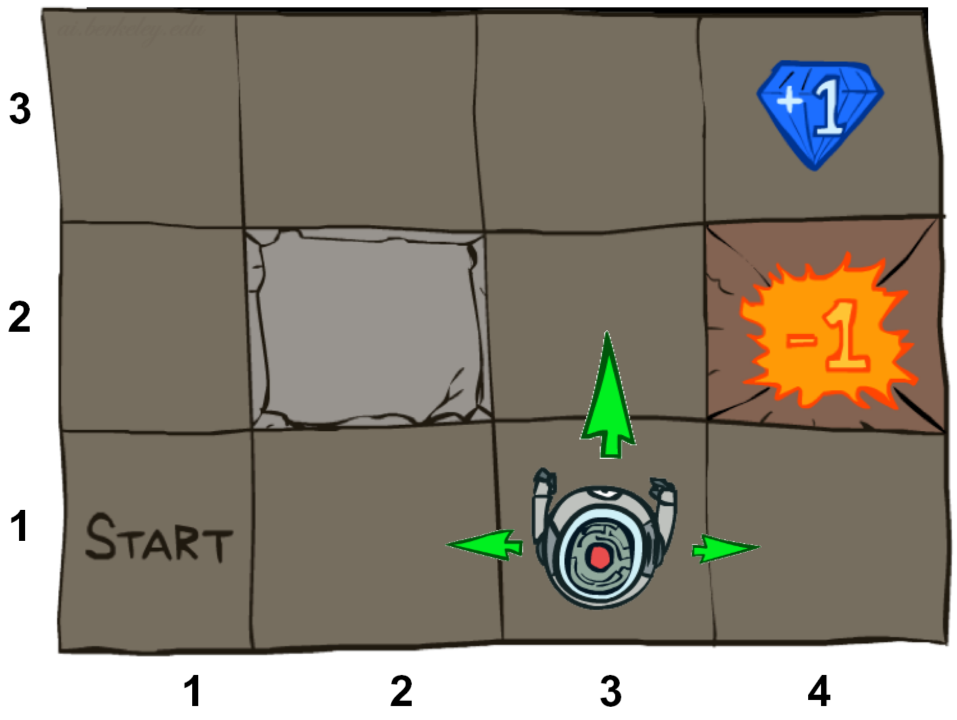
Last time

- Applications of sequential decision making (and Gridworld ☺)
- Minimax and expectimax trees
- MDP setup

Markov Decision Processes

An MDP is defined by:

- A set of states $s \in S$
- A set of actions $a \in A$
- A transition function T(s, a, s')
  - Probability that a from s leads to s', i.e., P(s' | s, a)
  - Also called the model or the dynamics
- A reward function R(s, a, s')
  - Sometimes just R(s) or R(s')
- A start state
- Maybe a terminal state



MDPs are non-deterministic search problems

- One way to solve them is with expectimax search
- We'll have a new tool soon

[Demo – gridworld manual intro (L8D1)]

What is Markov about MDPs?

“Markov” generally means that given the present state, the future and the past are independent

For Markov decision processes, “Markov” means action outcomes depend only on the current state

$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1}, \ldots S_0 = s_0)$$

$$=$$

$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t)$$

Andrey Markov
(1856-1922)

Policies

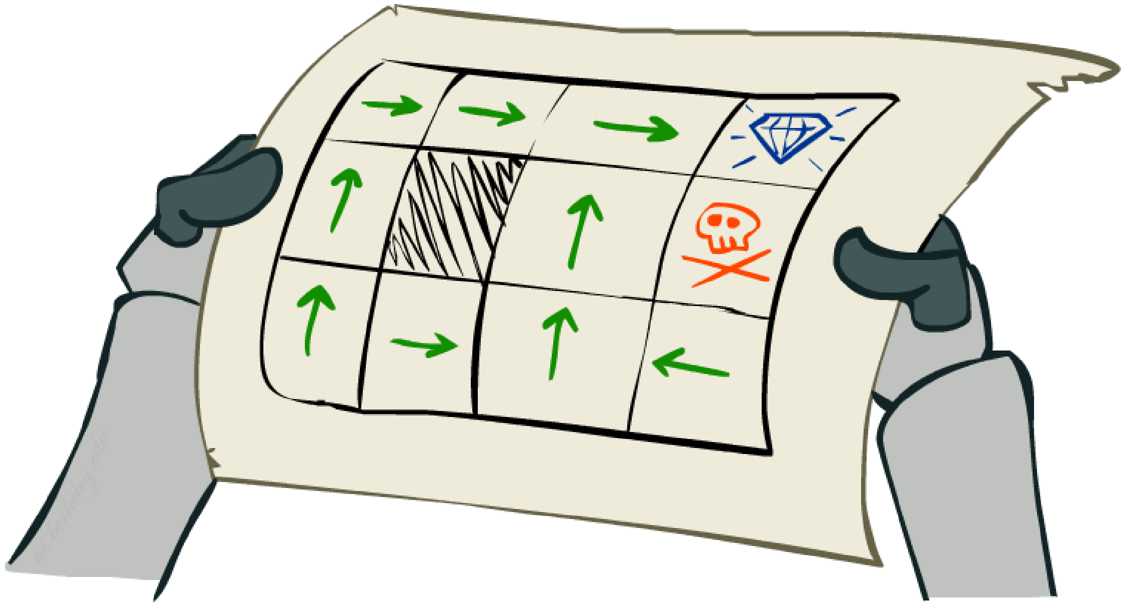
We don't just want an optimal plan, or sequence of actions, from start to a goal

For MDPs, we want an optimal policy $\pi^*: S \rightarrow A$

- A policy π gives an action for each state
- An optimal policy is one that maximizes expected utility if followed

Expectimax didn't compute entire policies

- It computed the action for a single state only



Optimal policy when R(s, a, s') = -0.03 for all non-terminals s

Plan

Last time

- MDP setup

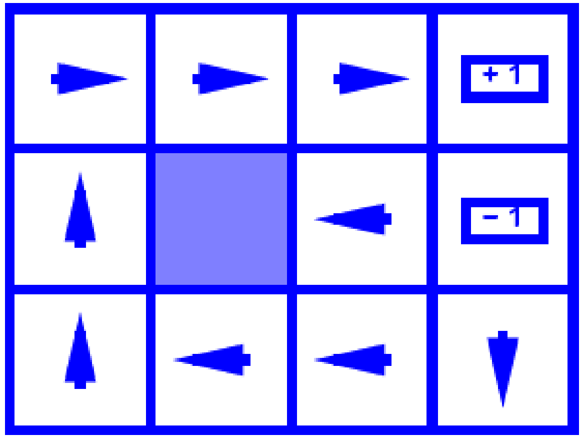
Today

- Rewards and Discounting
- Finding optimal policies: Value iteration and Bellman equations
- How to use optimal policies

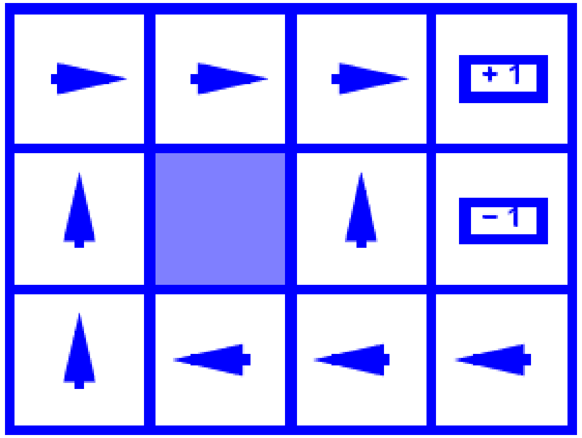
Next time

- What happens if we don't have $P(s' \mid s, a)$ and $R(s, a, s')$??

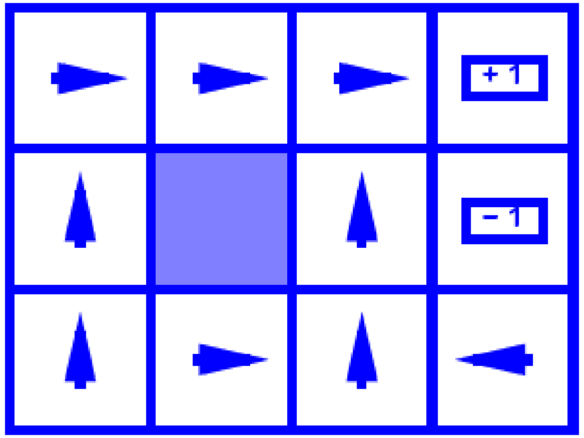
Optimal Policies



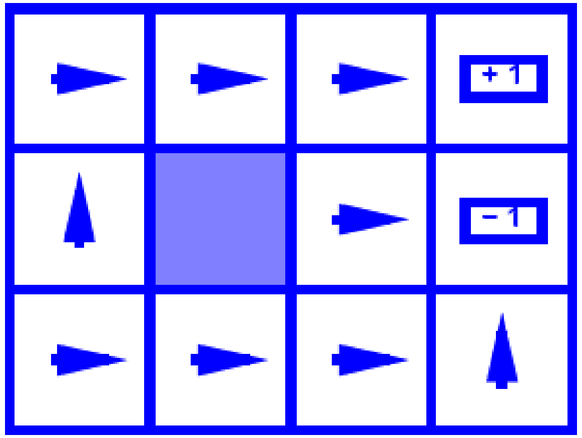
R(s) = -0.01



R(s) = -0.03



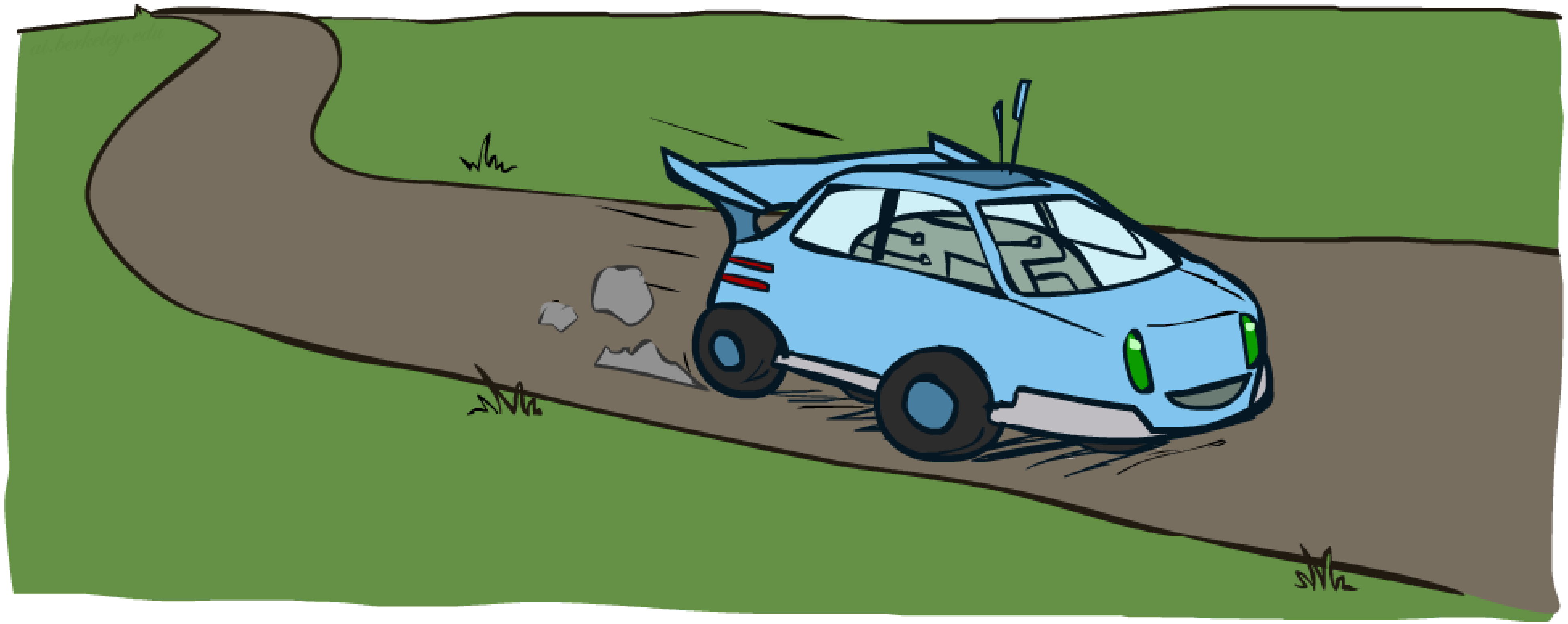
R(s) = -0.4



R(s) = -2.0

Slide: ai.berkeley.edu

Example: Racing



Slide: ai.berkeley.edu

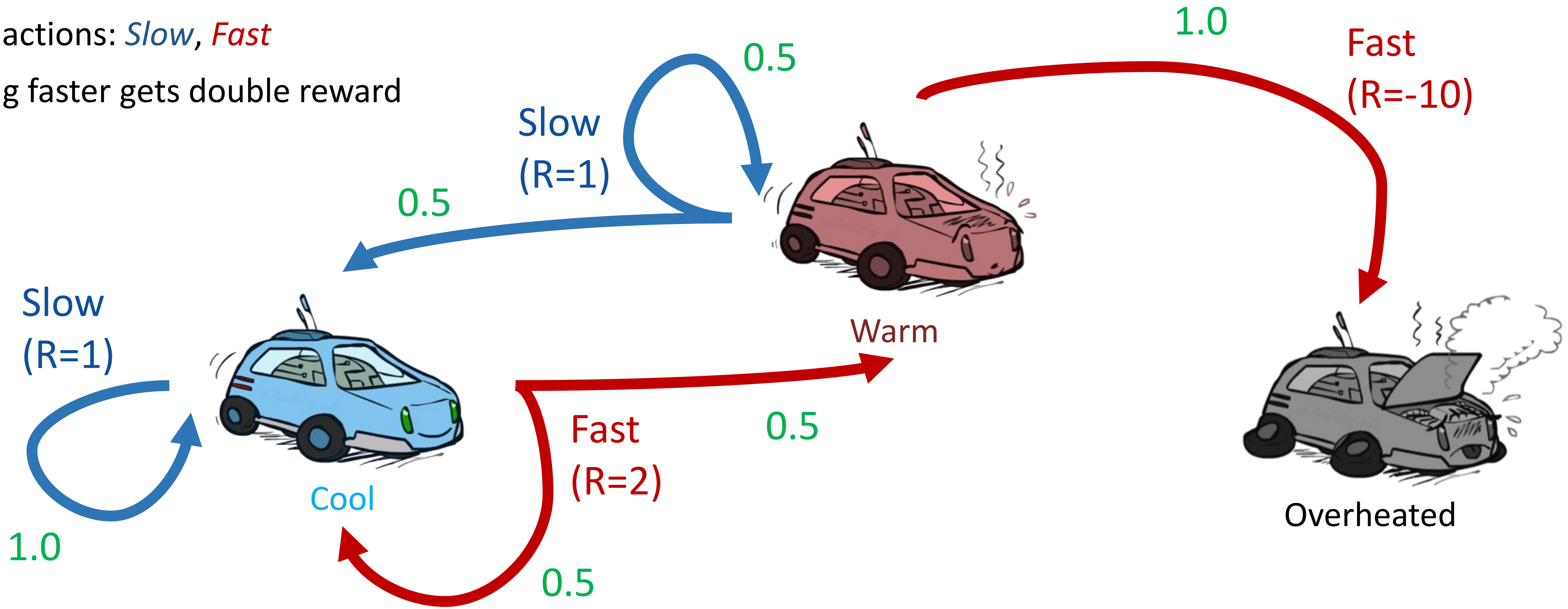
Example: Racing

A robot car wants to travel far, quickly

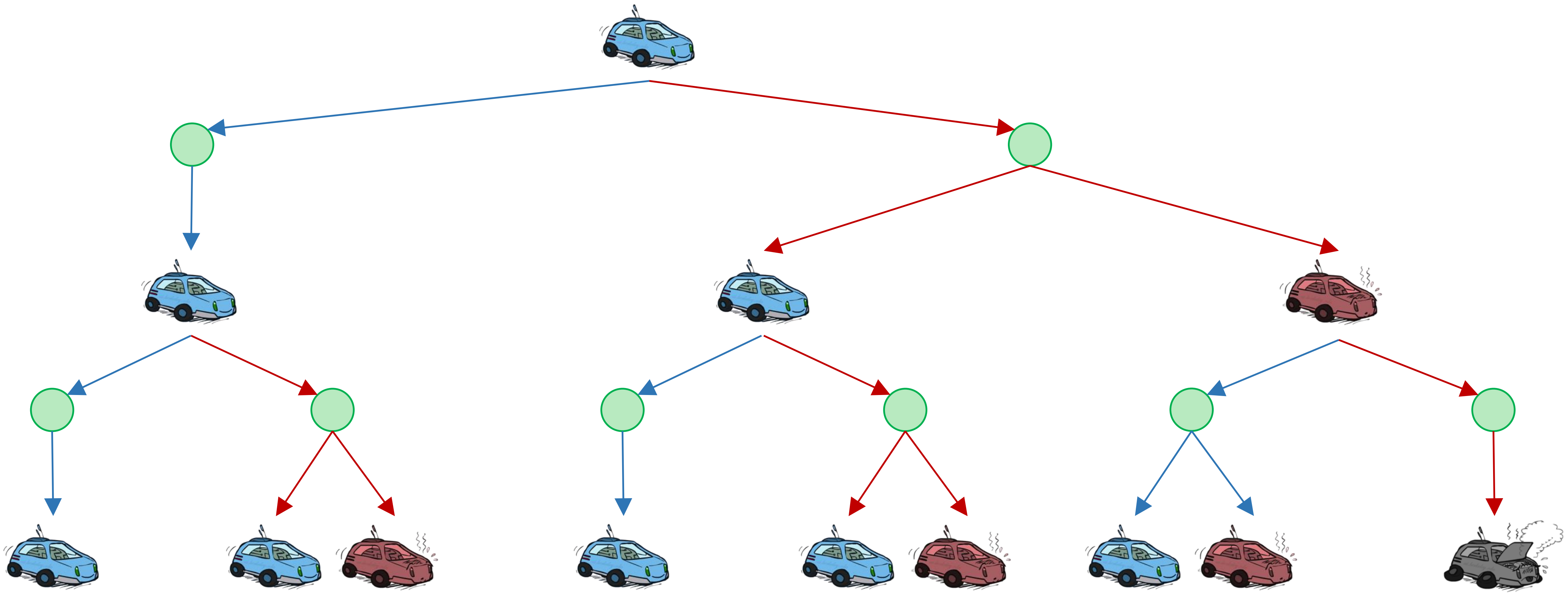
Three states: Cool, Warm, Overheated

Two actions: *Slow*, *Fast*

Going faster gets double reward



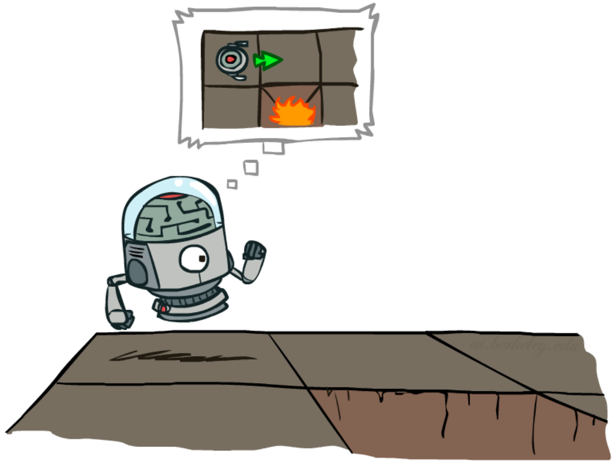
Racing Search Tree

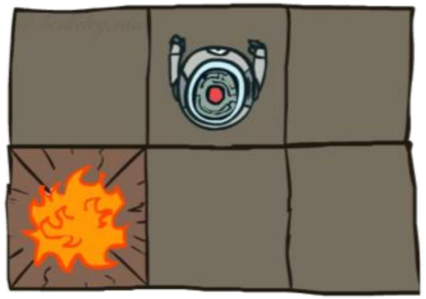


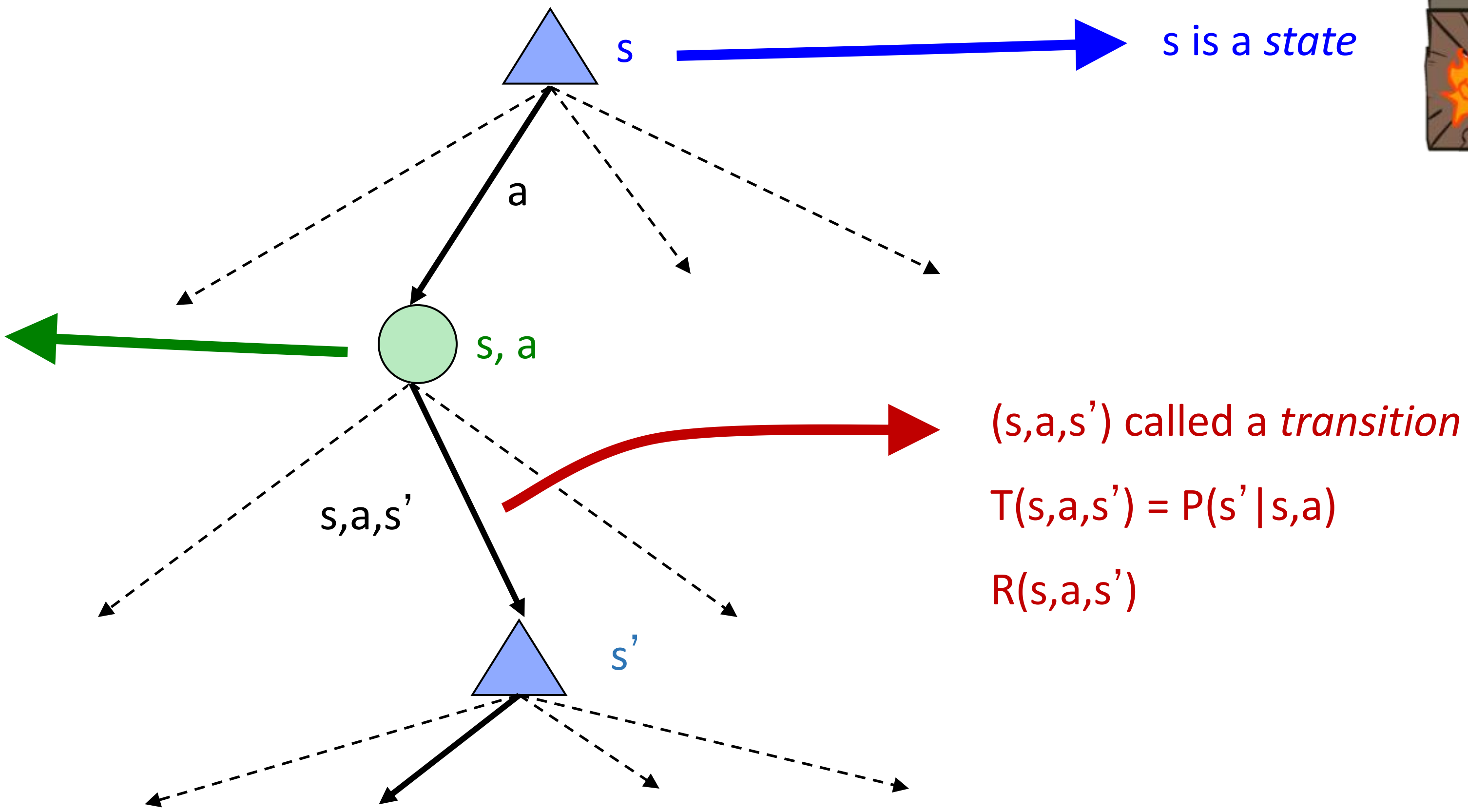
Slide: ai.berkeley.edu

MDP Search Trees

Each MDP state projects an expectimax-like search tree

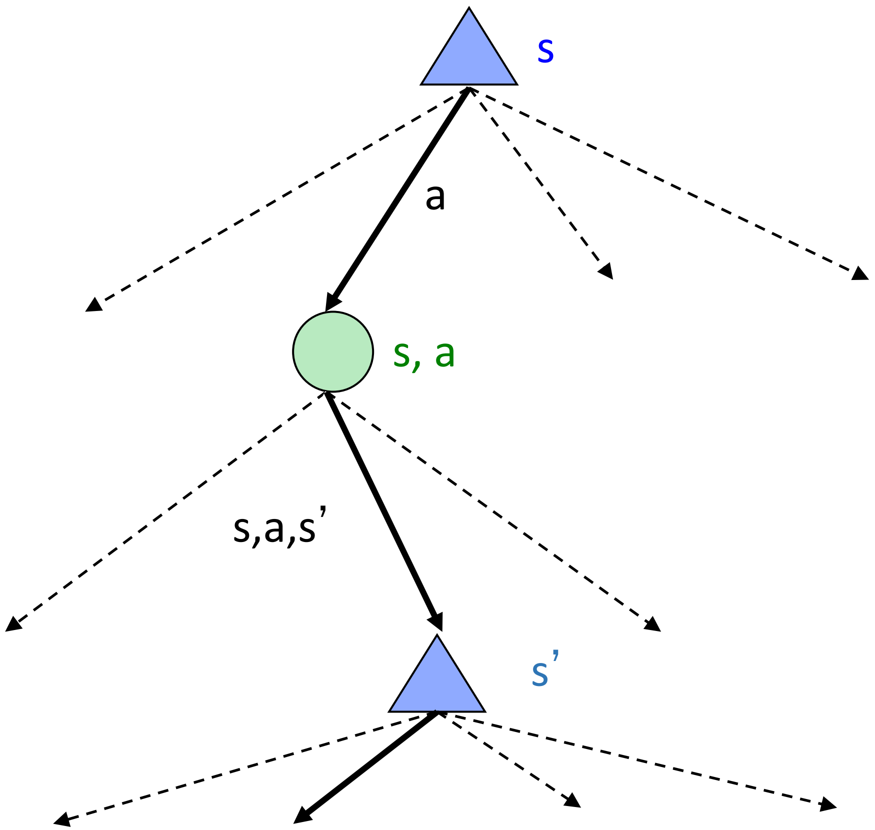






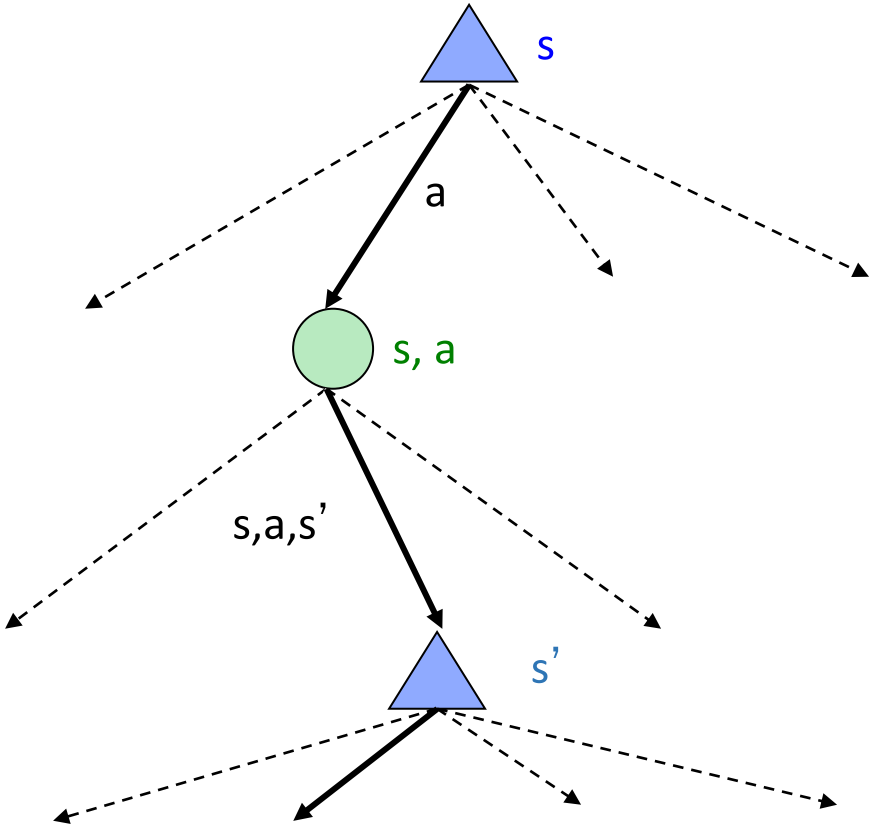
Recursive Expectimax

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



Recursive Expectimax

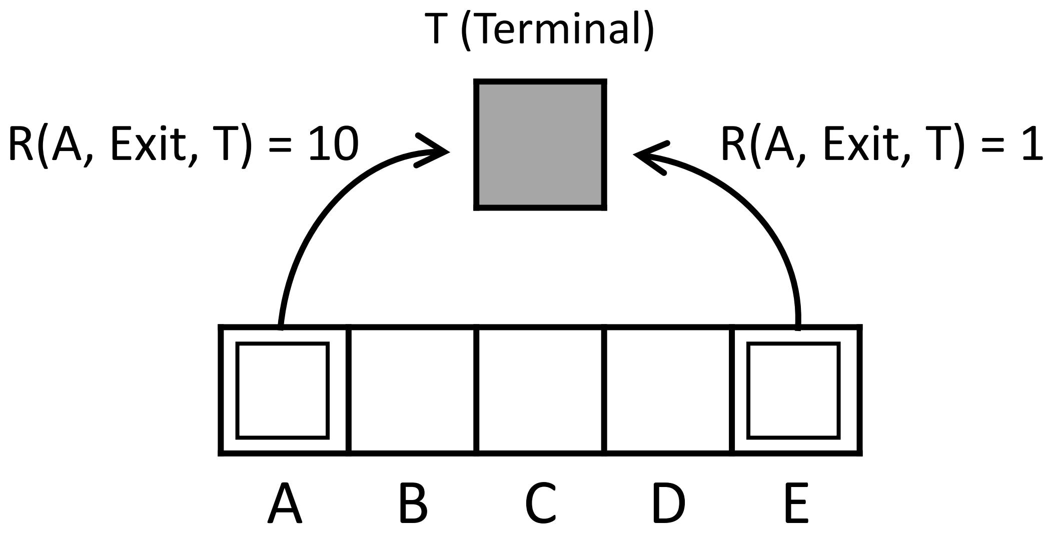
$$V(s) = \max_a \sum_{s'} P(s'|s,a)\,[R(s,a,s') + V(s')]$$



Simple Deterministic Example

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

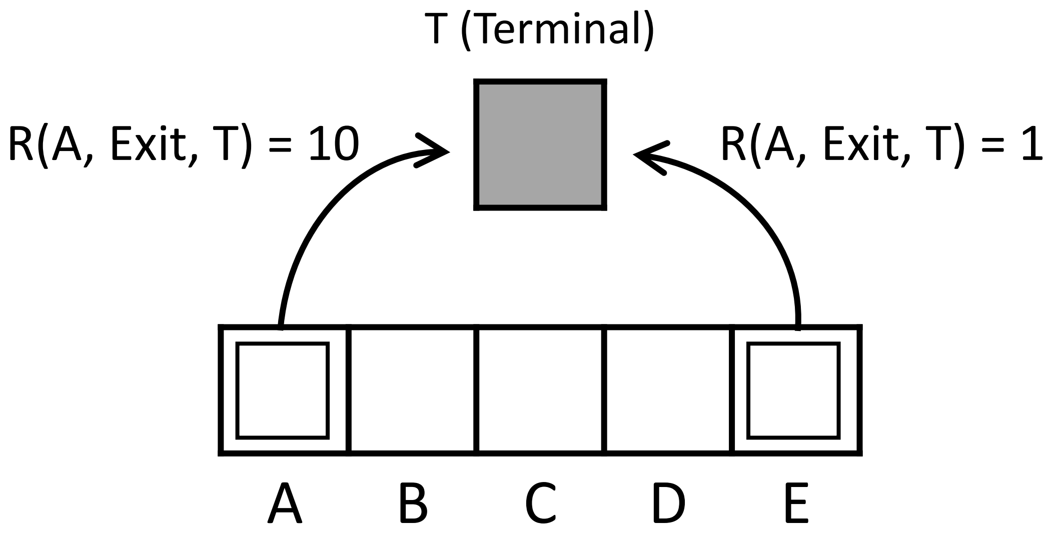
$$V(s) = \max_a [R(s,a,s') + V(s')]$$



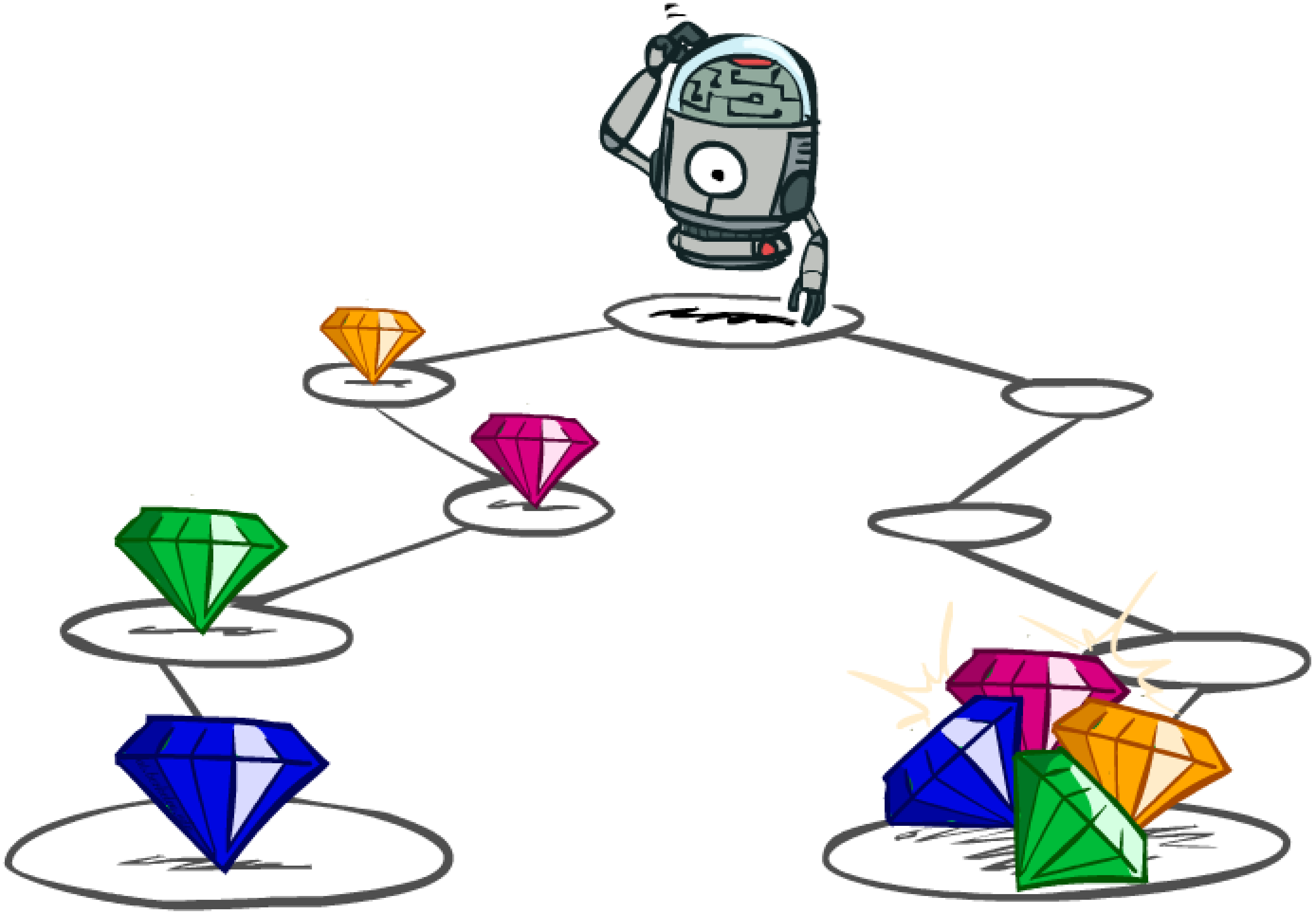
Simple Deterministic Example

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

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



Utilities of Sequences



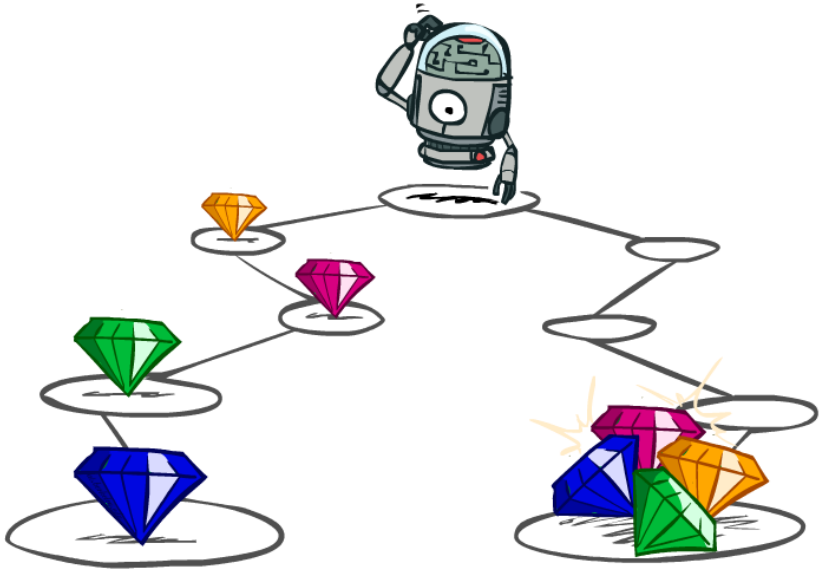
Slide: ai.berkeley.edu

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

Slide: ai.berkeley.edu

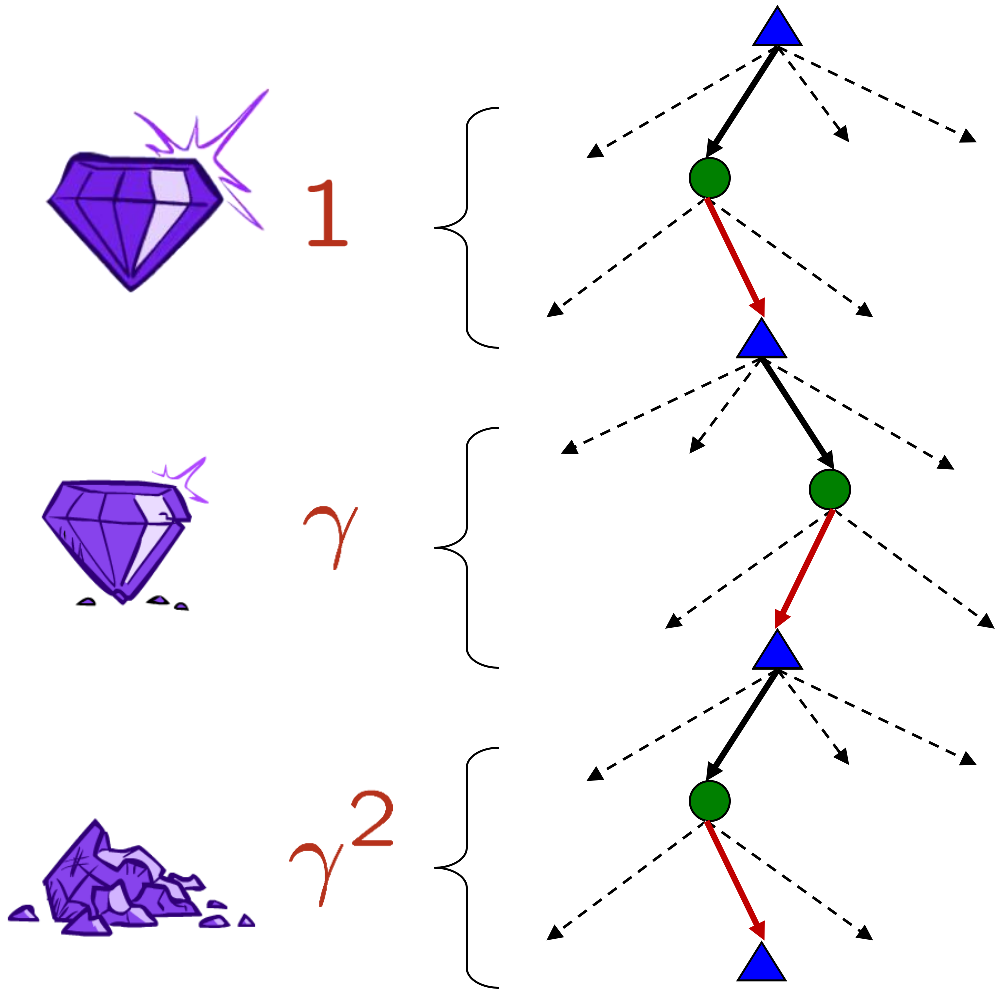
Discounting

How to discount?

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

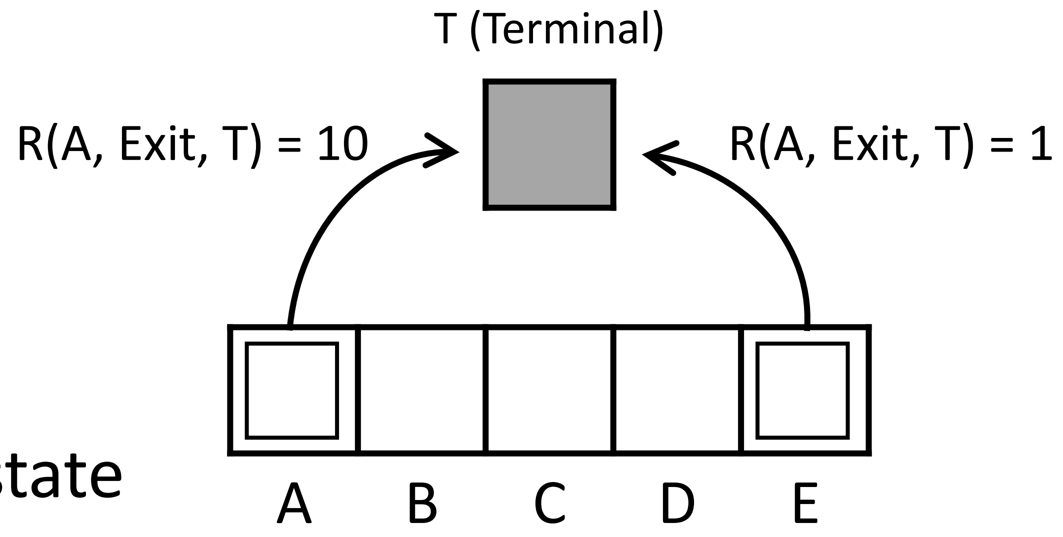
Why discount?

- Sooner rewards probably do have higher utility than later rewards
- Also helps our algorithms converge



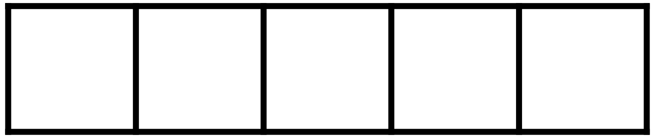
Discounting

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

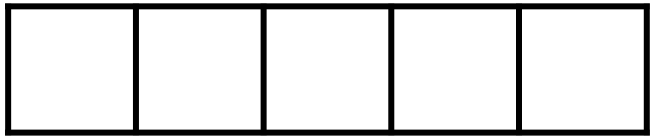


$$V_{k+1}(s) = \max_a [R(s,a,s') + \gamma\, V_k(s')]$$

For γ = 1, what is the optimal policy?



For γ = 0.1, what is the optimal policy?

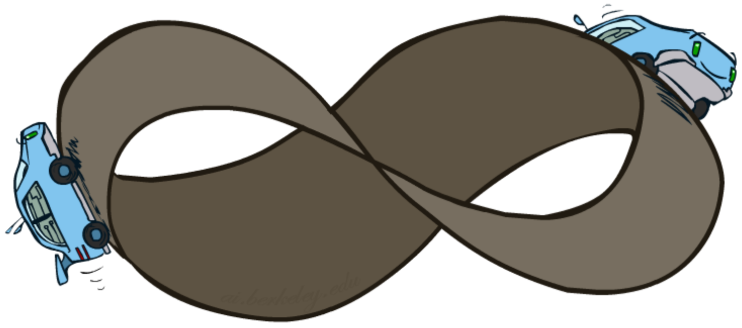


For which γ are West and East equally good when in state d?

Infinite Utilities?!

- Problem: What if the game lasts forever? Do we get infinite rewards?

- Solutions:
- Finite horizon: (similar to depth-limited search)
  - Terminate episodes after a fixed T steps (e.g. life)
  - Gives nonstationary policies (π depends on time left)



- Discounting: use $0 < \gamma < 1$

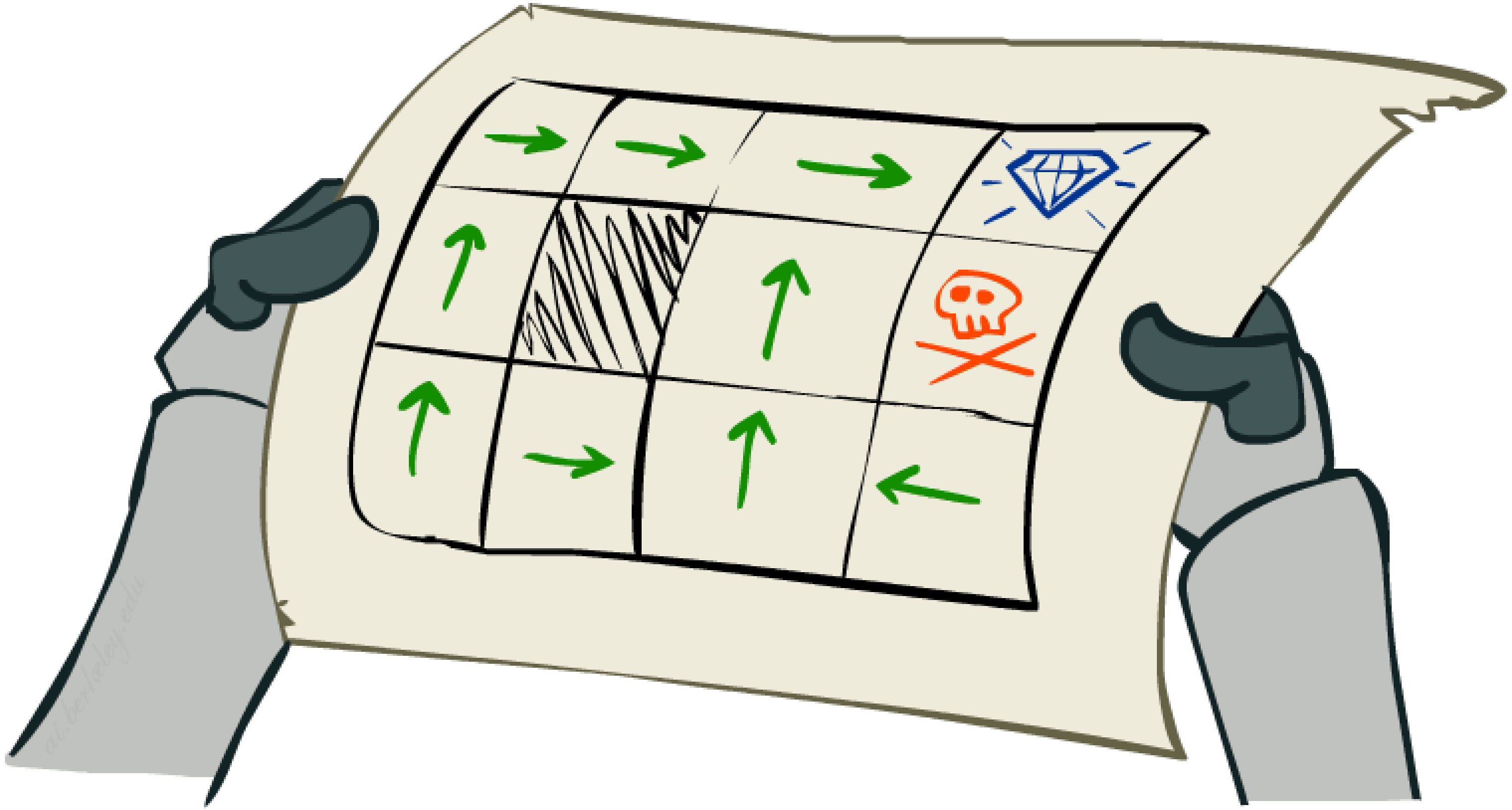
$$U([r_0, \ldots r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_t \le R_{\max}/(1-\gamma)$$

  - Smaller γ means smaller "horizon" – shorter term focus

- Absorbing state: guarantee that for every policy, a terminal state will eventually be reached (like "overheated" for racing)

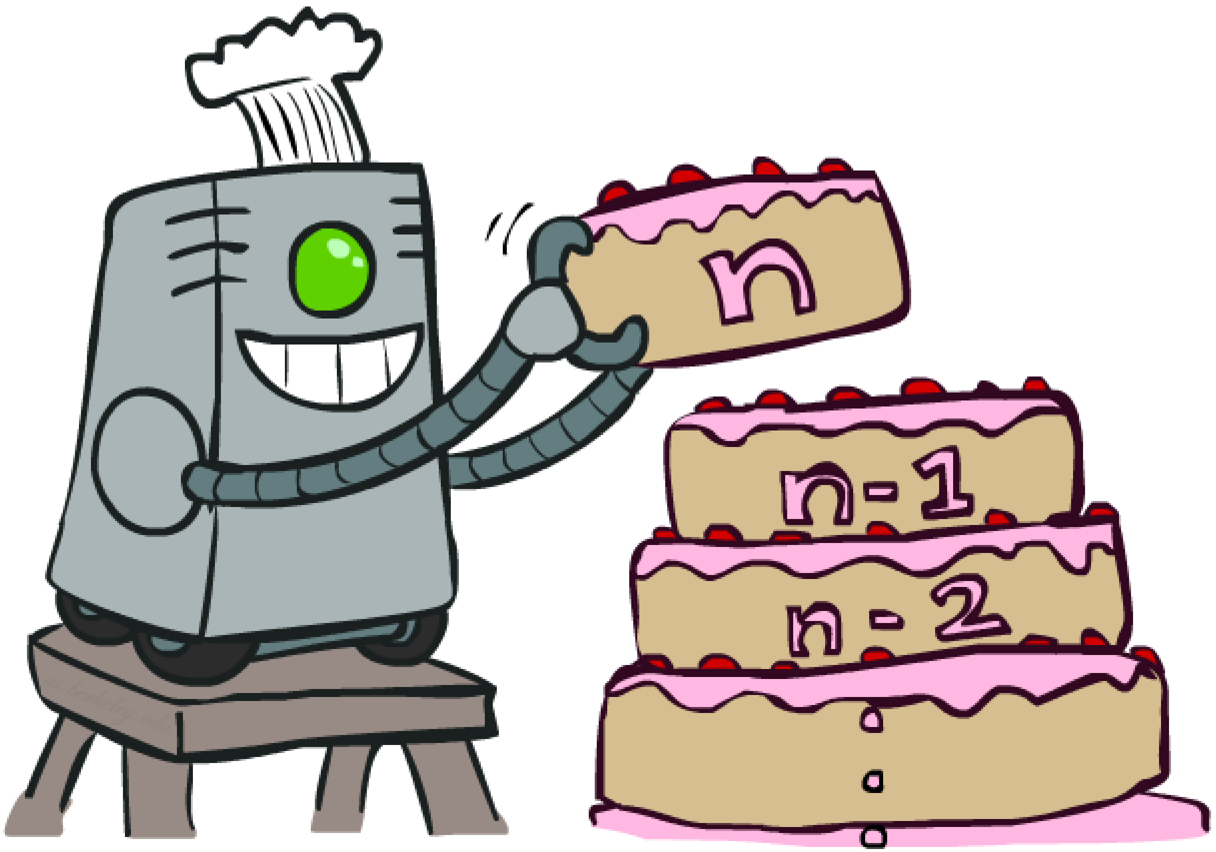
Slide: ai.berkeley.edu

Solving MDPs



Slide: ai.berkeley.edu

Value Iteration



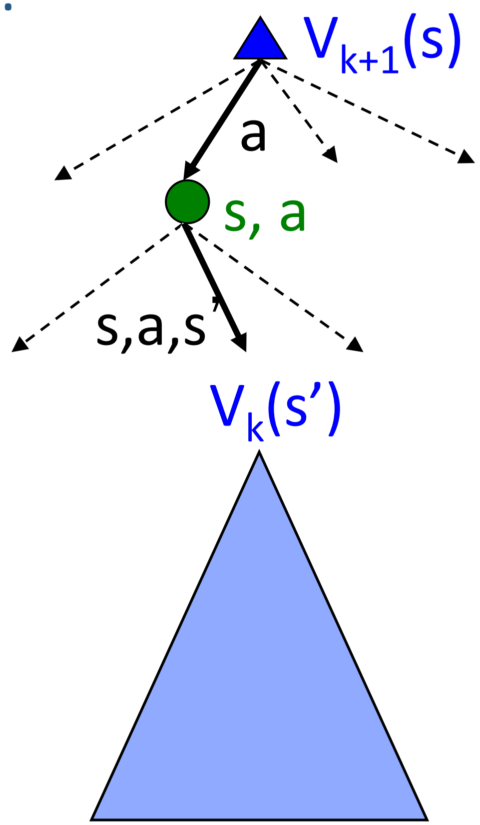
Slide: ai.berkeley.edu

Value Iteration

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

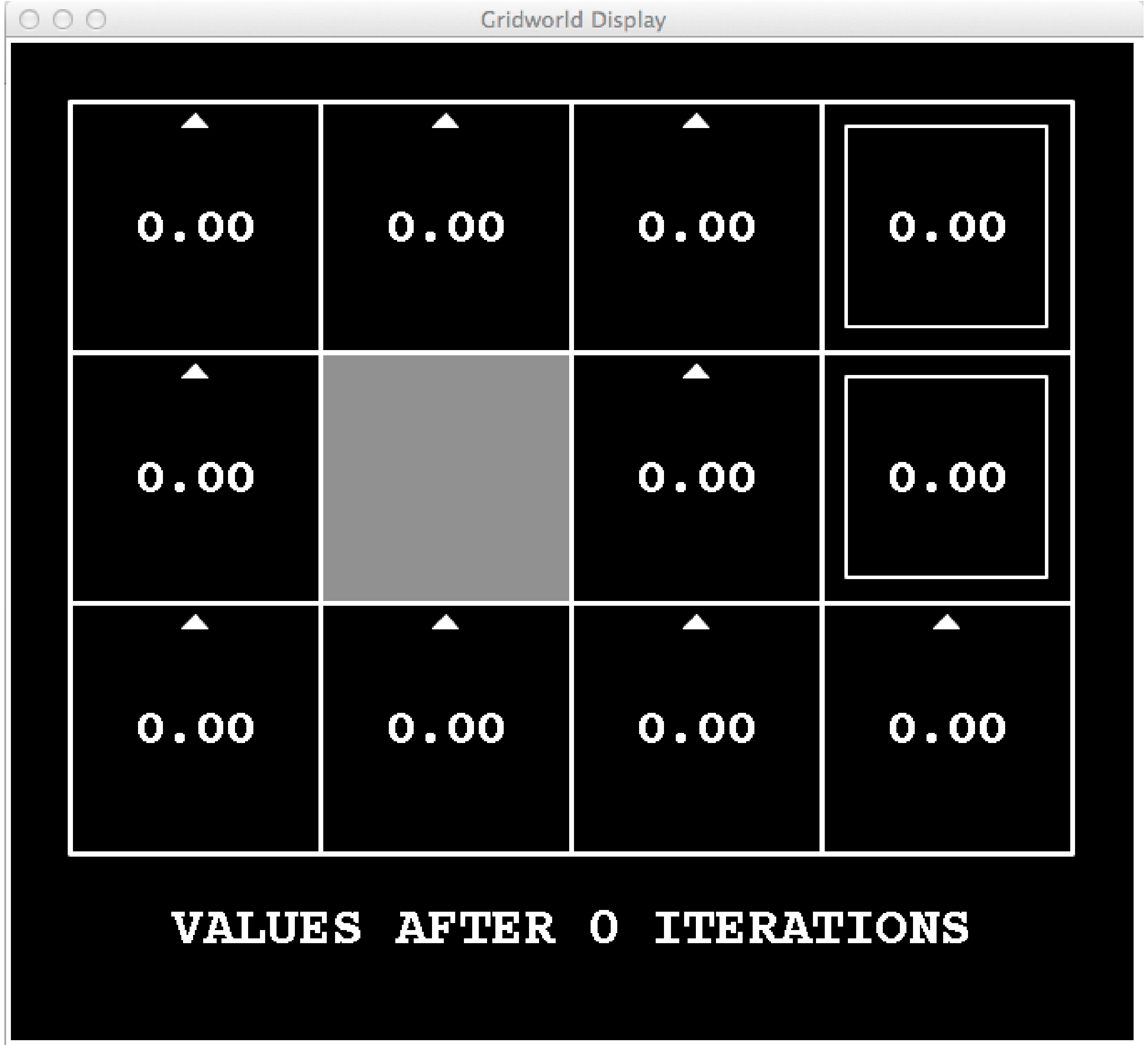
Given vector of $V_k(s)$ values, do one ply of expectimax from each state:

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



Repeat until convergence

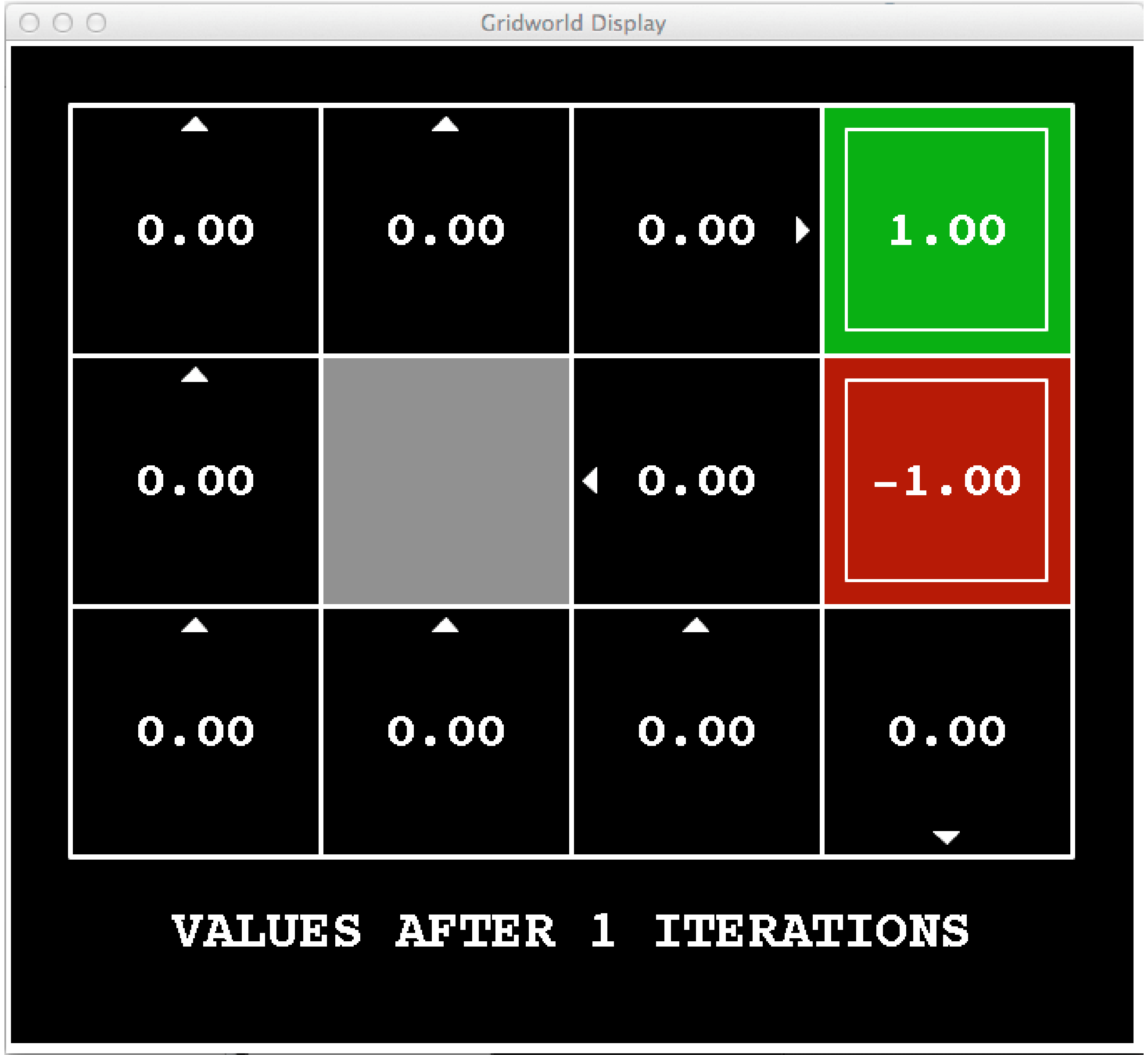
k=0



Noise = 0.2
Discount = 0.9
Living reward = 0

Slide: ai.berkeley.edu

k=1



Noise = 0.2
Discount = 0.9
Living reward = 0

Slide: ai.berkeley.edu

k=2

Noise = 0.2
Discount = 0.9
Living reward = 0

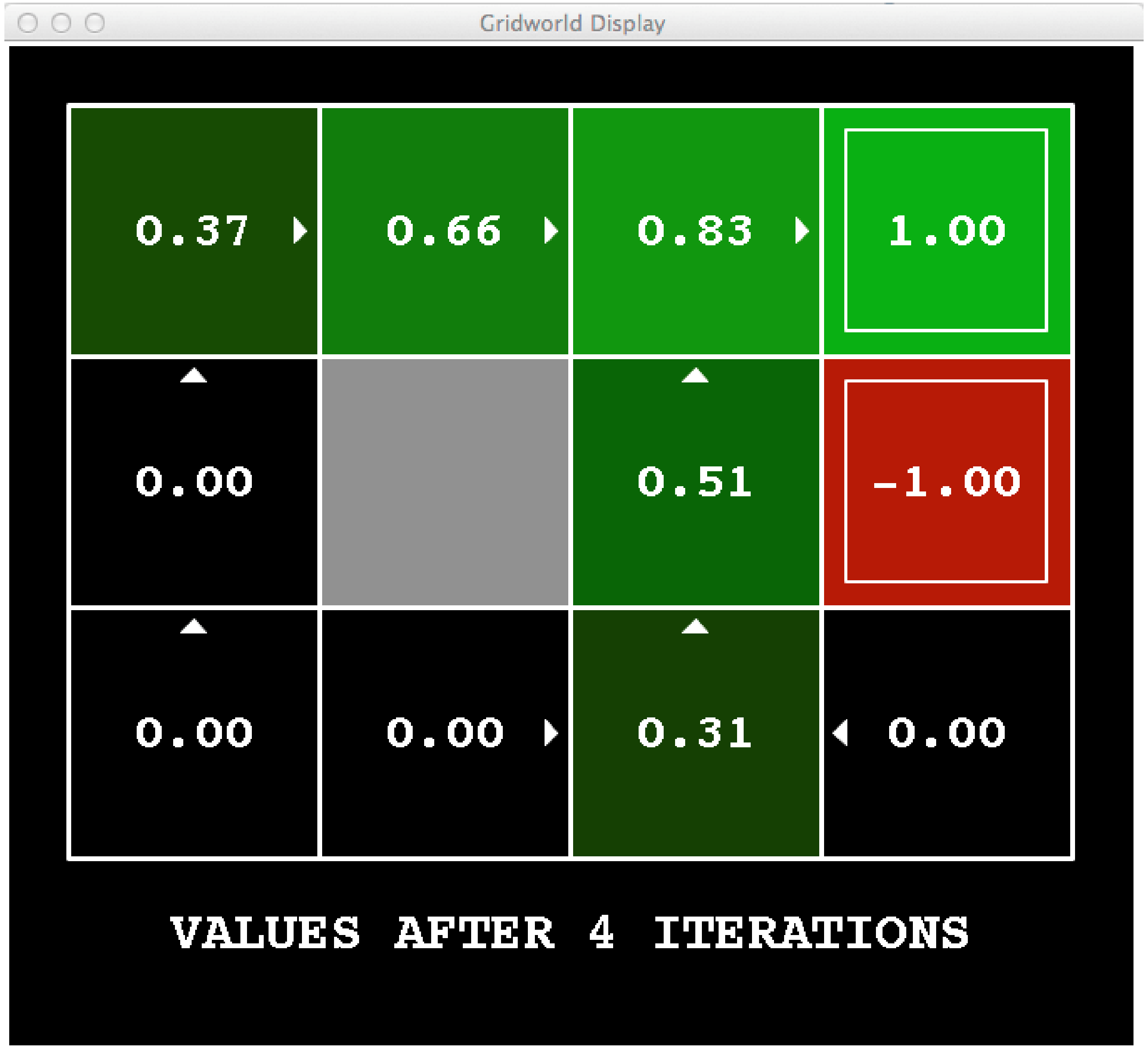
Slide: ai.berkeley.edu

k=3

Slide: ai.berkeley.edu

Noise = 0.2
Discount = 0.9
Living reward = 0

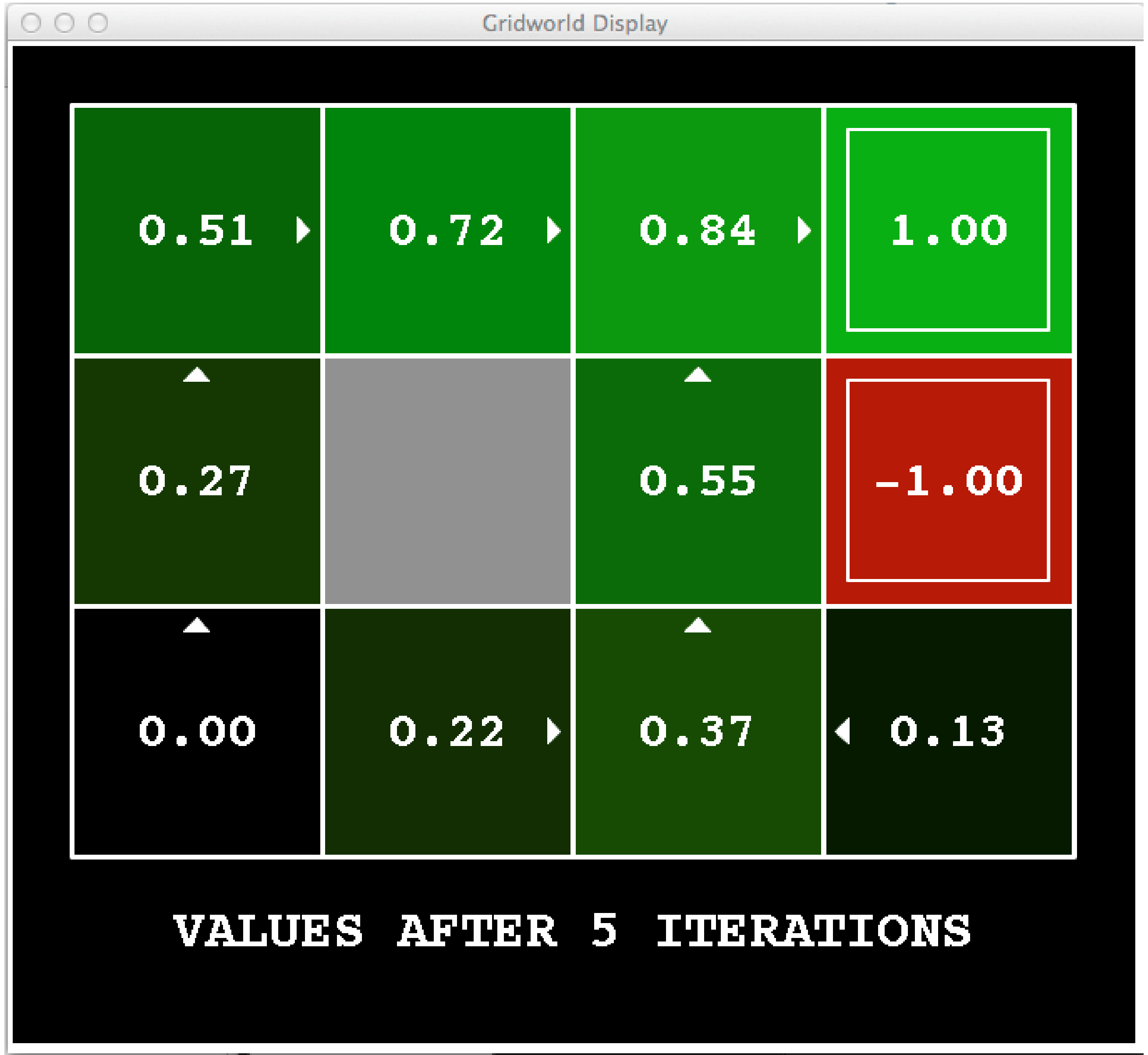
k=4



Noise = 0.2
Discount = 0.9
Living reward = 0

Slide: ai.berkeley.edu

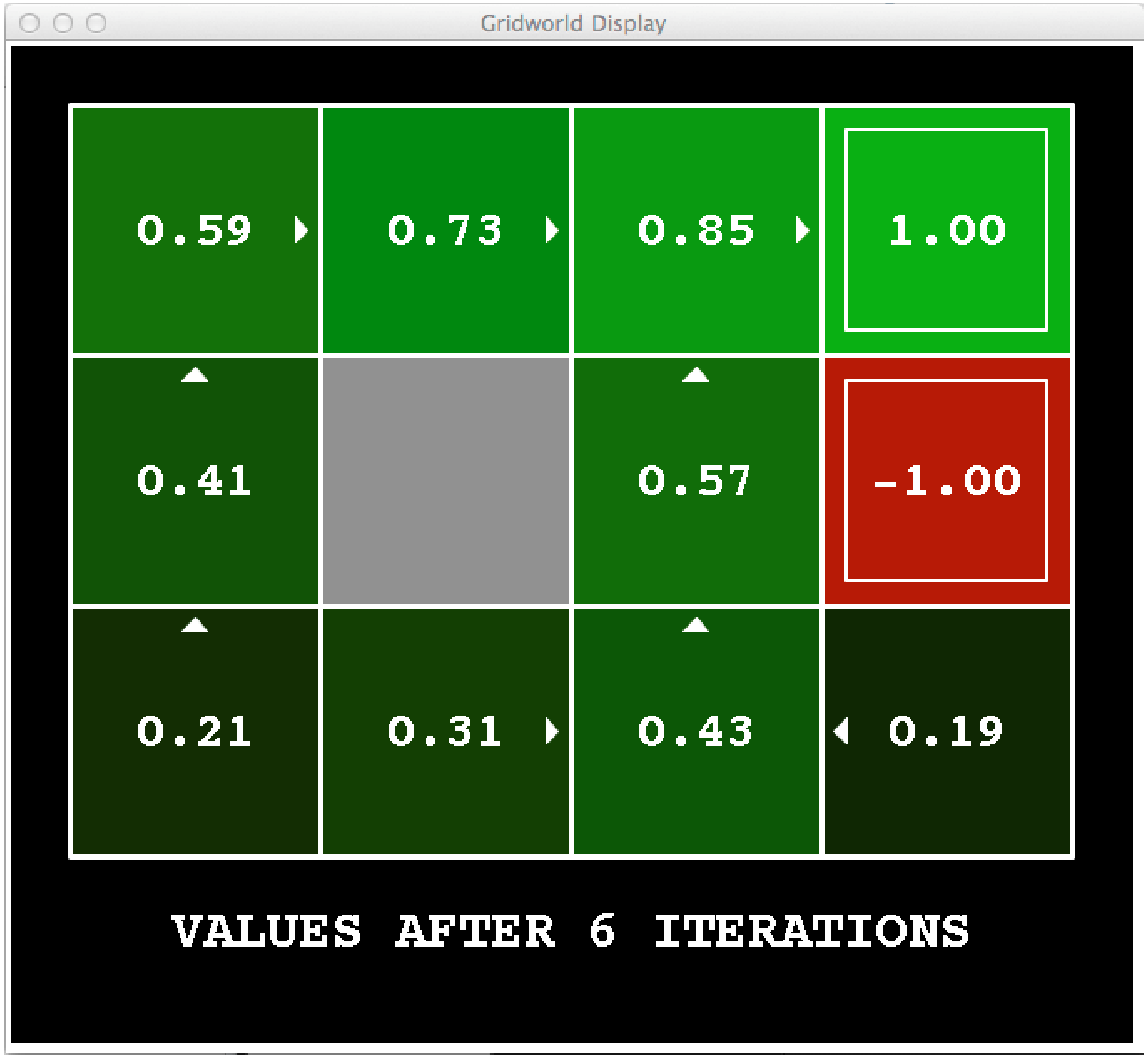
k=5



Noise = 0.2
Discount = 0.9
Living reward = 0

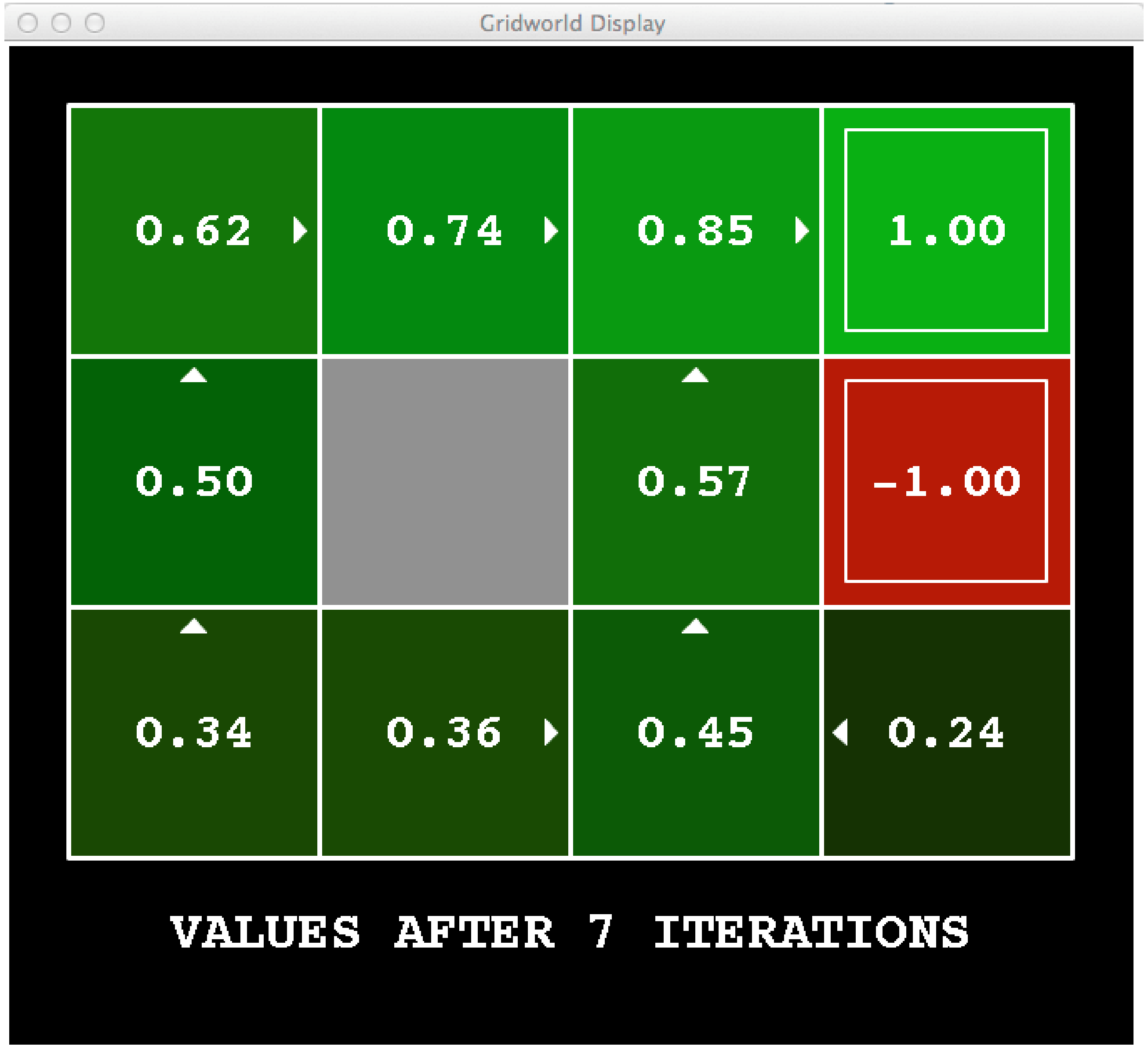
Slide: ai.berkeley.edu

k=6



Noise = 0.2
Discount = 0.9
Living reward = 0

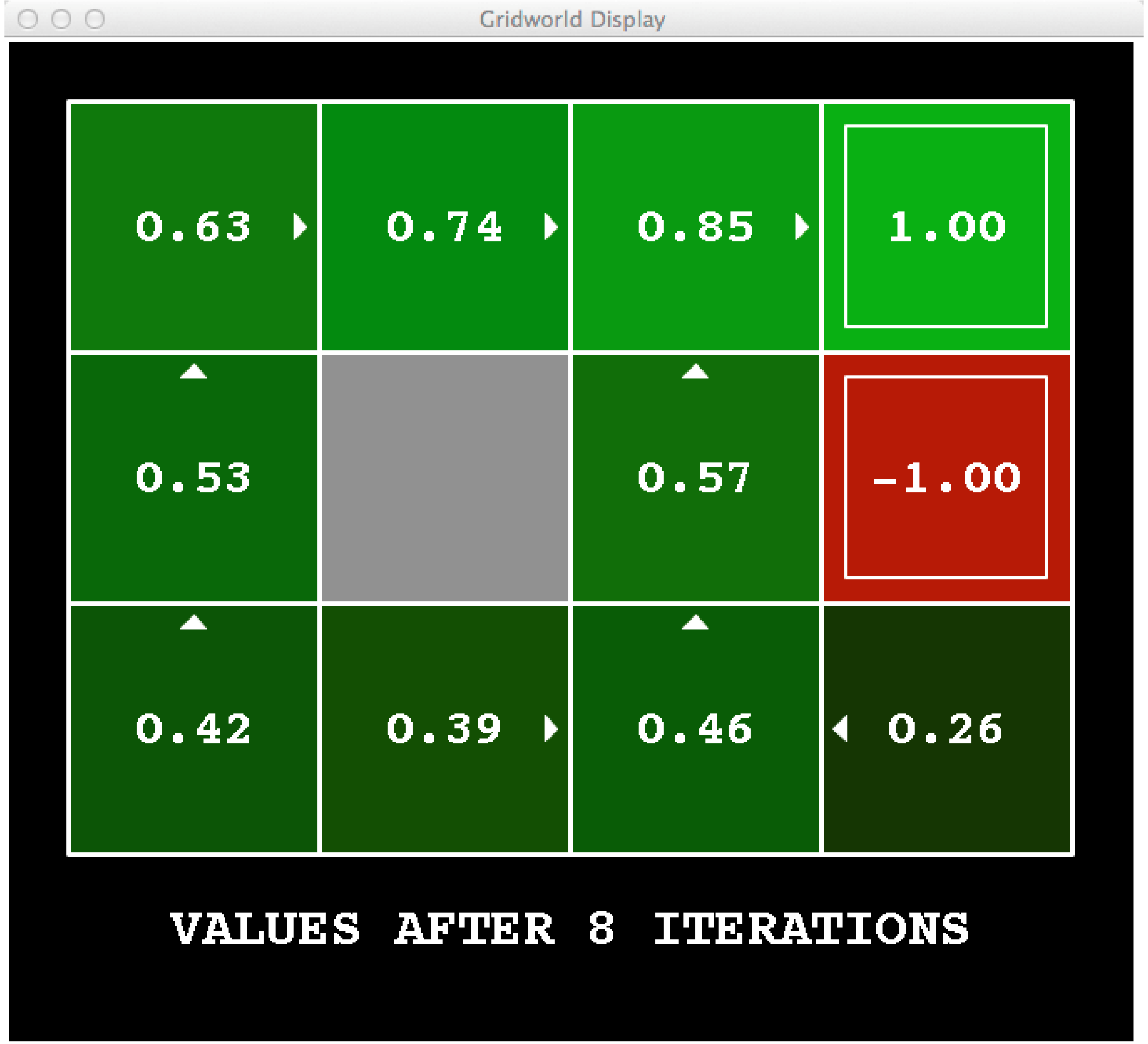
k=7



Noise = 0.2
Discount = 0.9
Living reward = 0

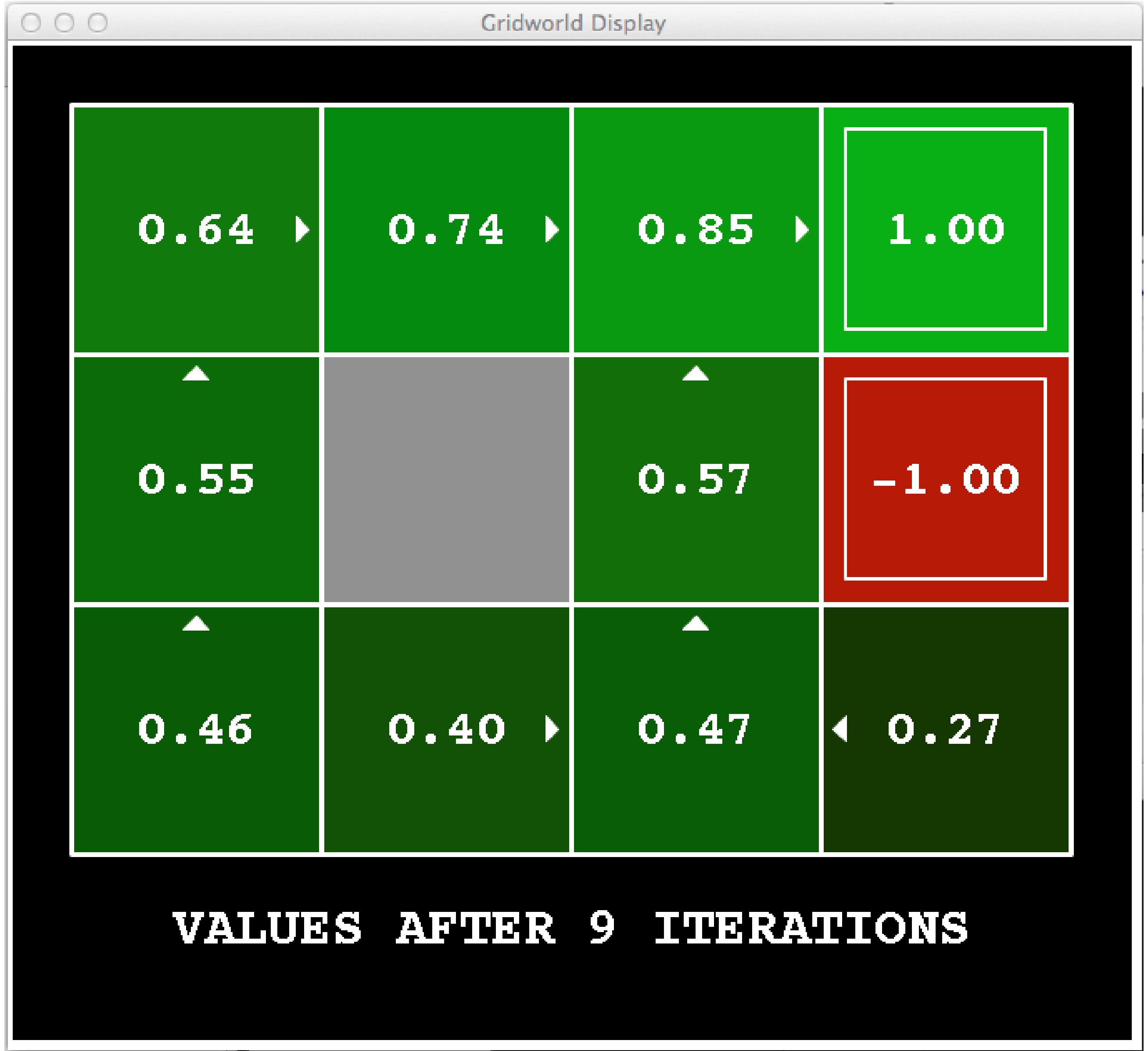
Slide: ai.berkeley.edu

k=8



Noise = 0.2
Discount = 0.9
Living reward = 0

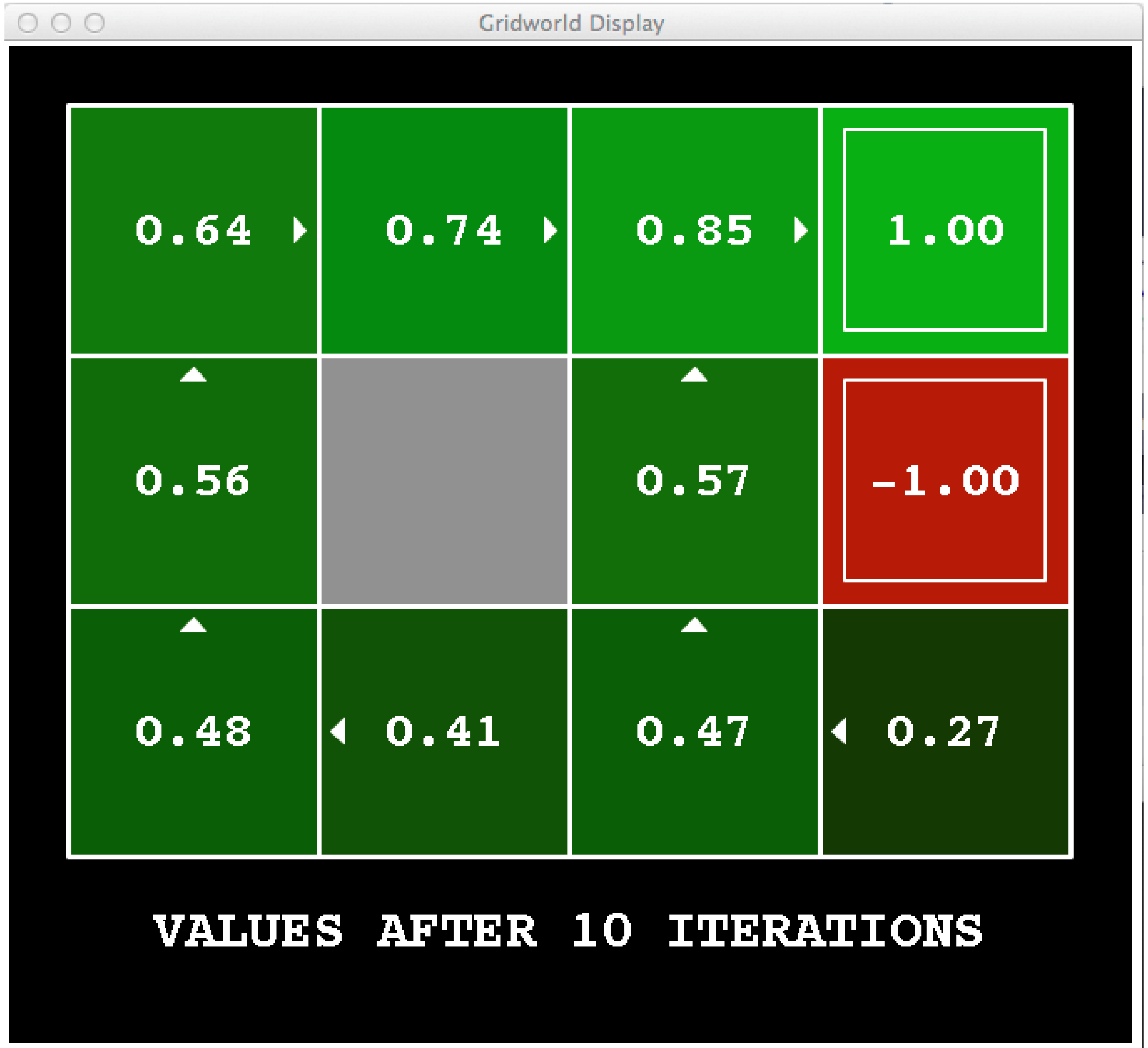
k=9



Slide: ai.berkeley.edu

Noise = 0.2
Discount = 0.9
Living reward = 0

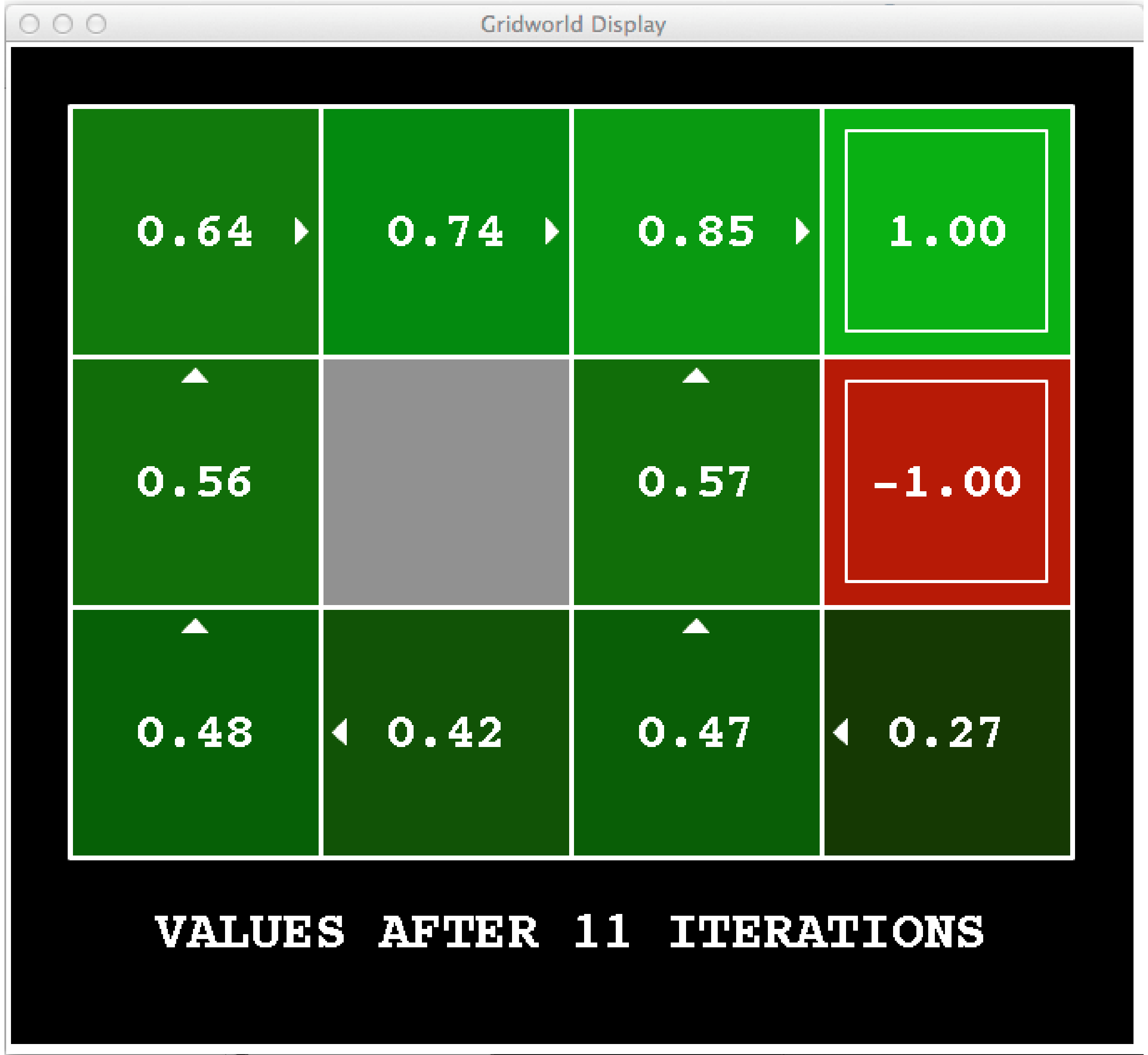
k=10



Noise = 0.2
Discount = 0.9
Living reward = 0

Slide: ai.berkeley.edu

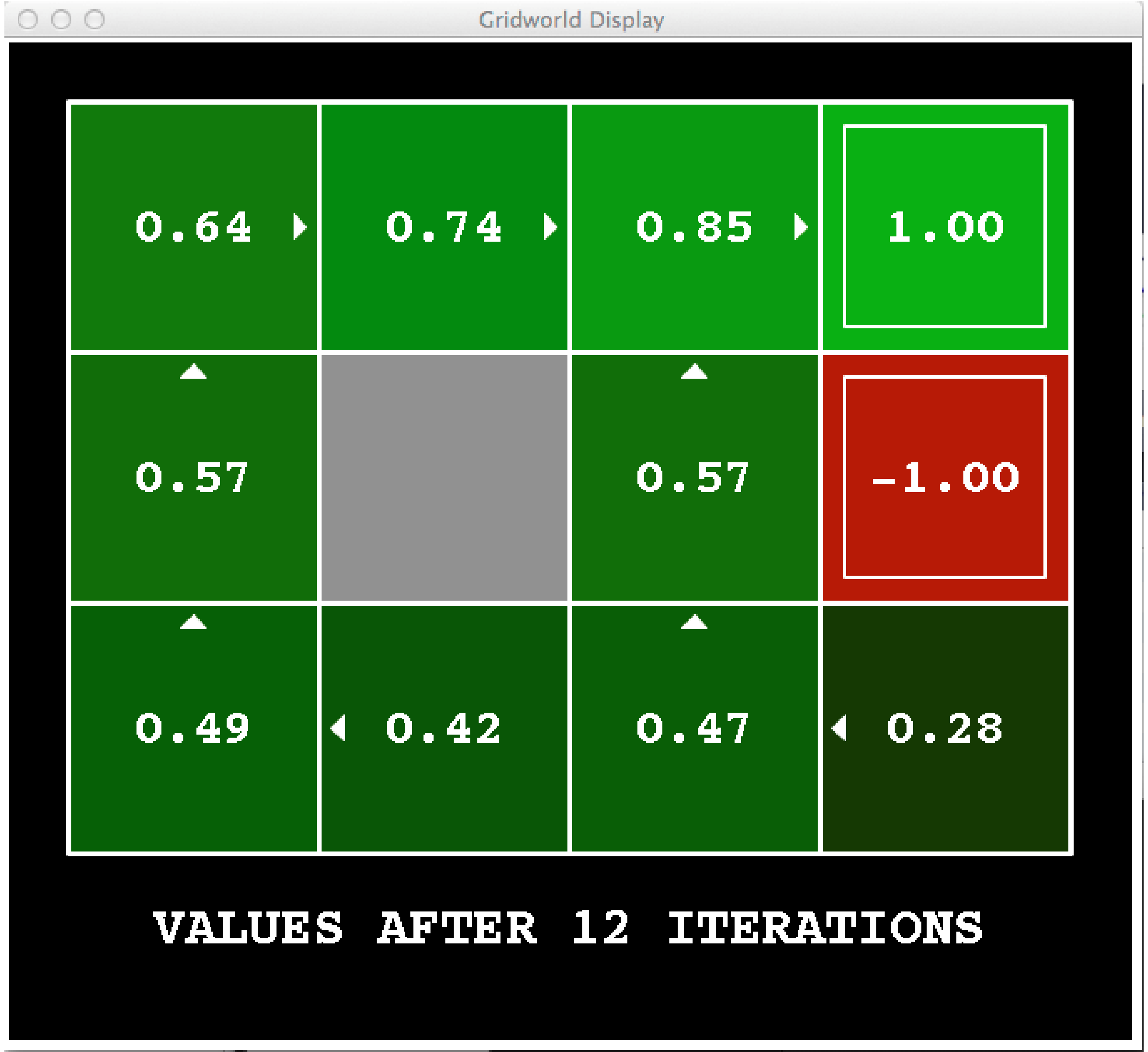
k=11



Noise = 0.2
Discount = 0.9
Living reward = 0

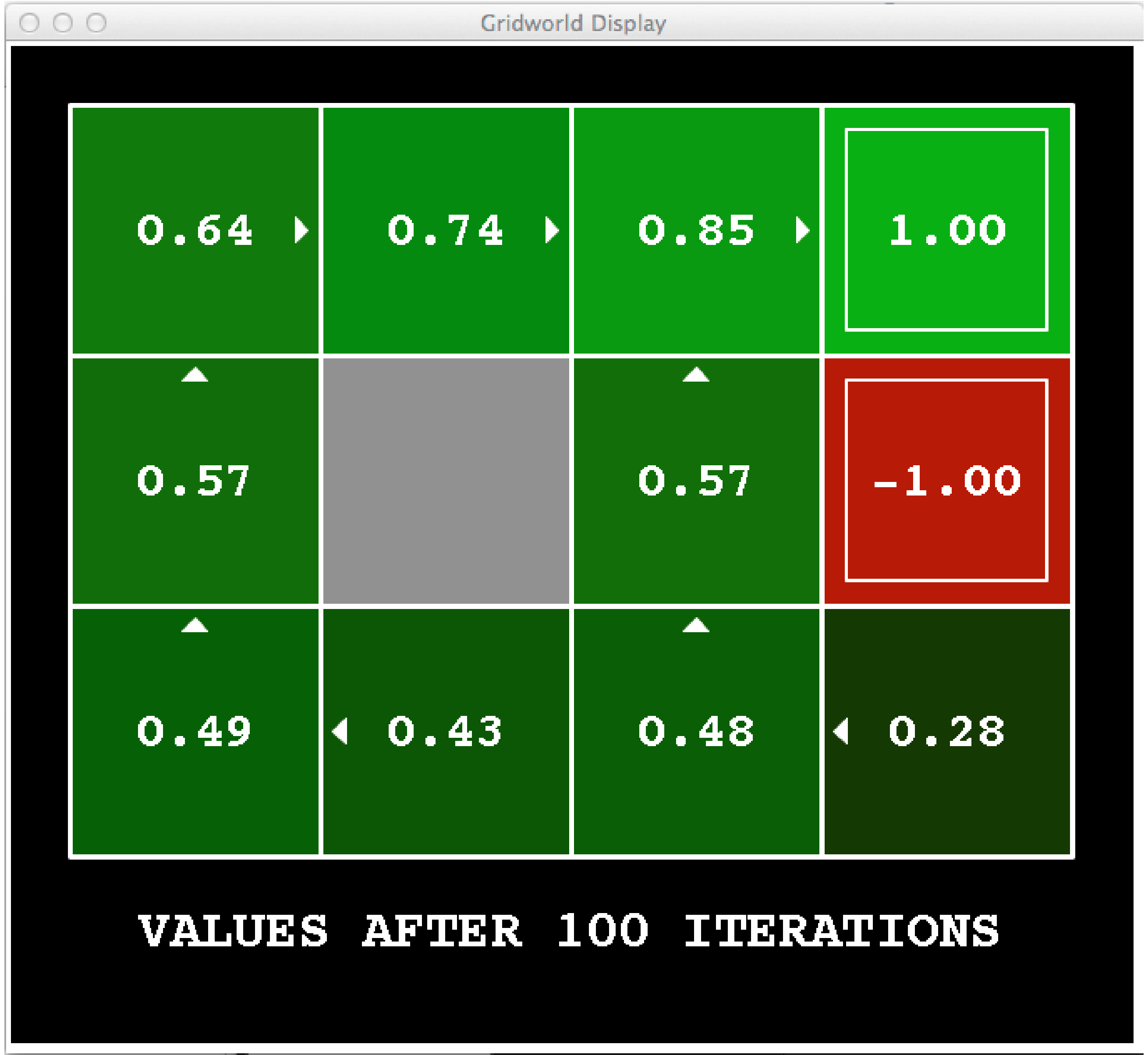
Slide: ai.berkeley.edu

k=12



Noise = 0.2
Discount = 0.9
Living reward = 0

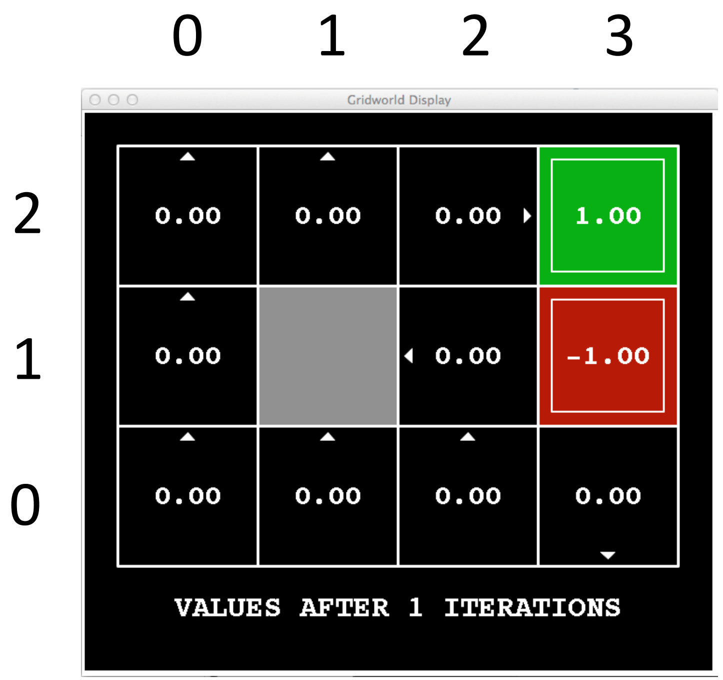
k=100

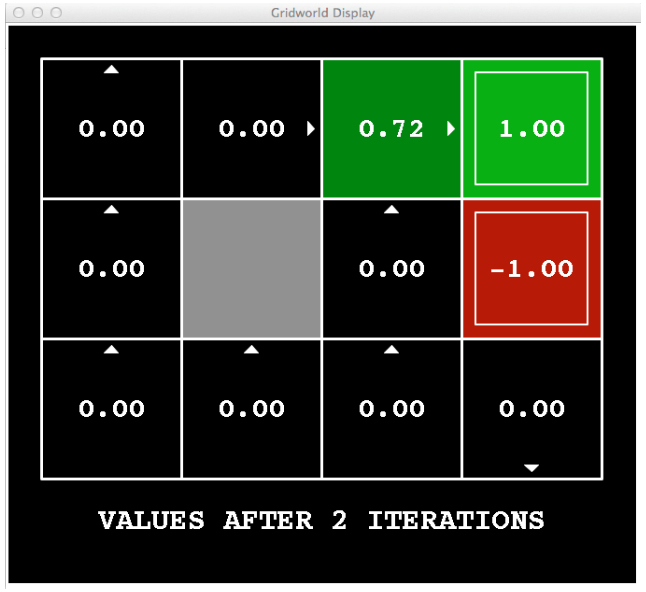


Noise = 0.2
Discount = 0.9
Living reward = 0

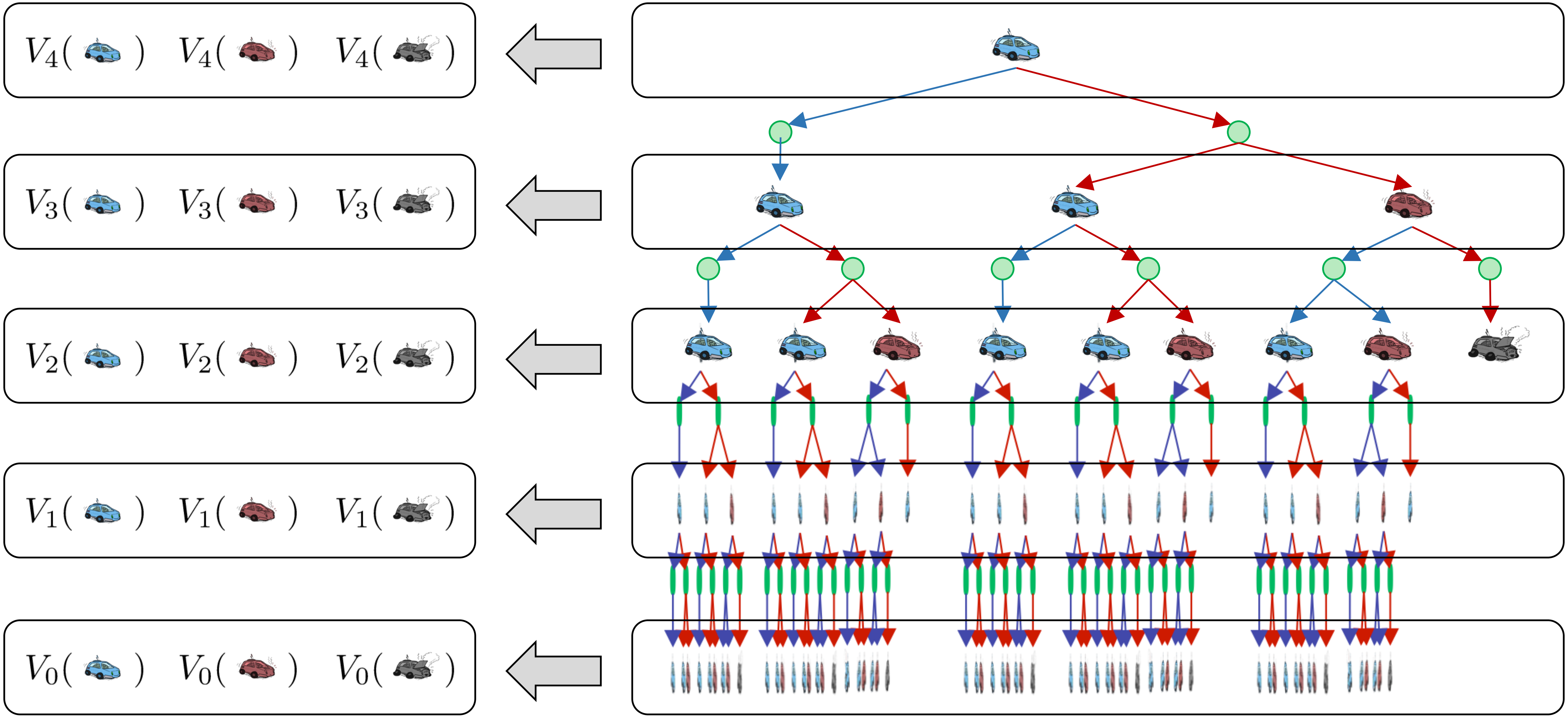
Exercise

As we moved from k=1 to k=2 to k=3, how did we get these specific values for s=(2,2)?



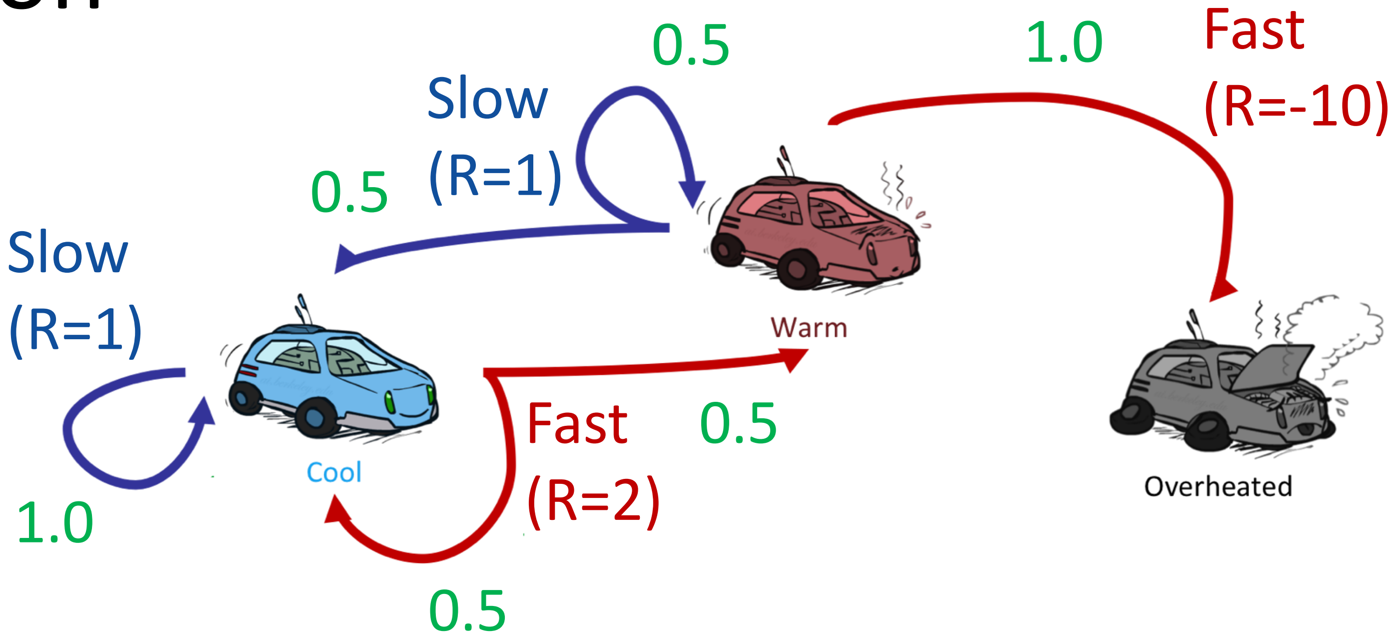


Racing Tree Example



Example: Value Iteration

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



Assume no discount! $\gamma = 1$

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

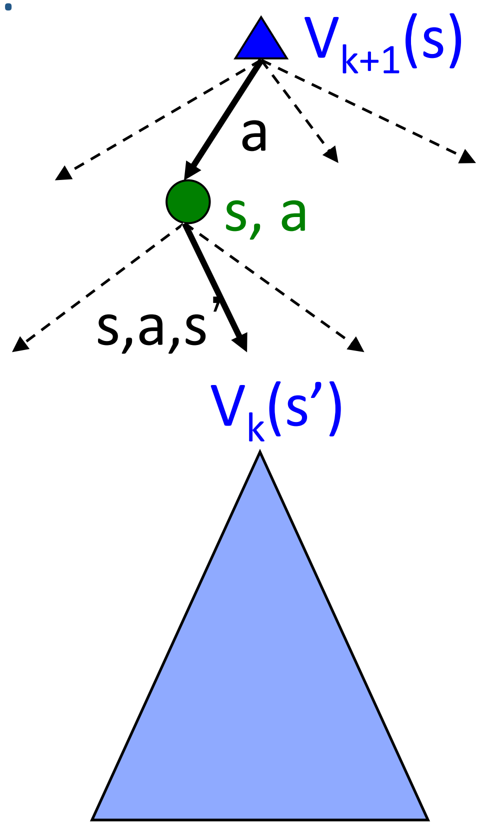
Value Iteration

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



Piazza Poll 1

What is the complexity of each iteration in Value Iteration?

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

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

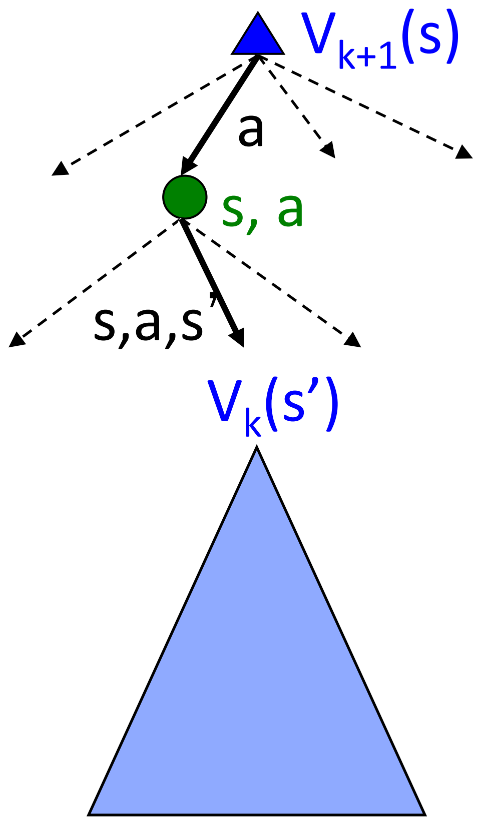
II: $O(|S|^2|A|)$

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

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

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

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



Piazza Poll 1

What is the complexity of each iteration in Value Iteration?

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

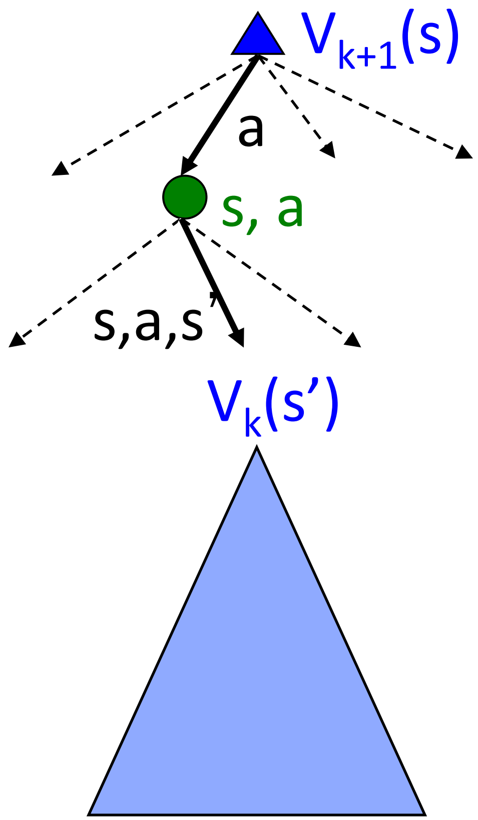
I: $O(|S||A|)$

$\boxed{\text{II: } O(|S|^2|A|)}$

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

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

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



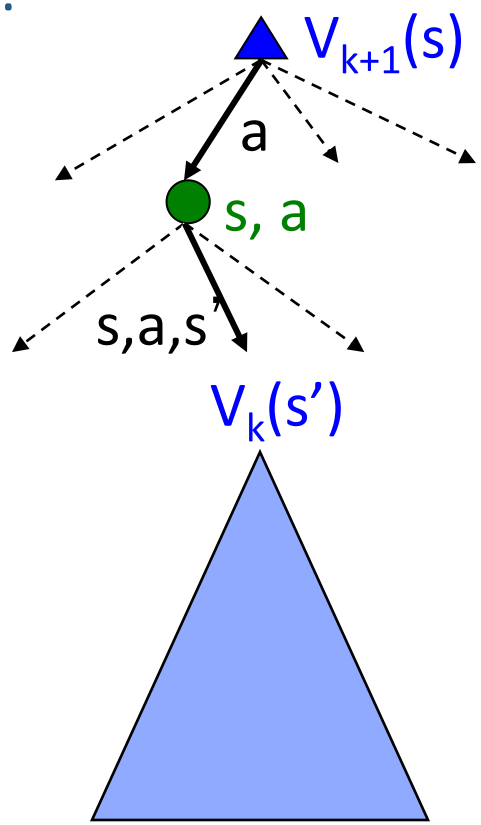
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s')\right]$$

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:

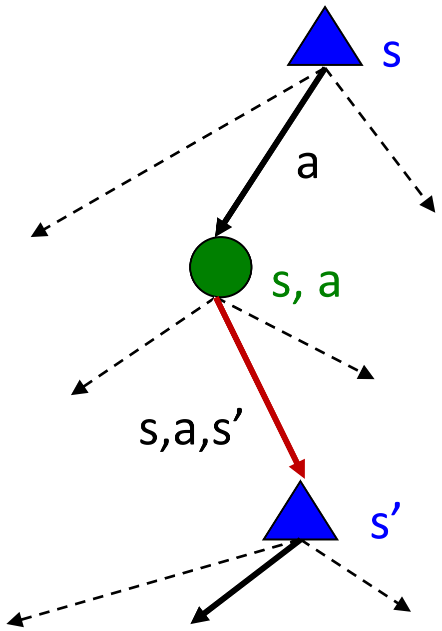
  $V^*(s)$ = expected utility starting in s and acting optimally

- 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



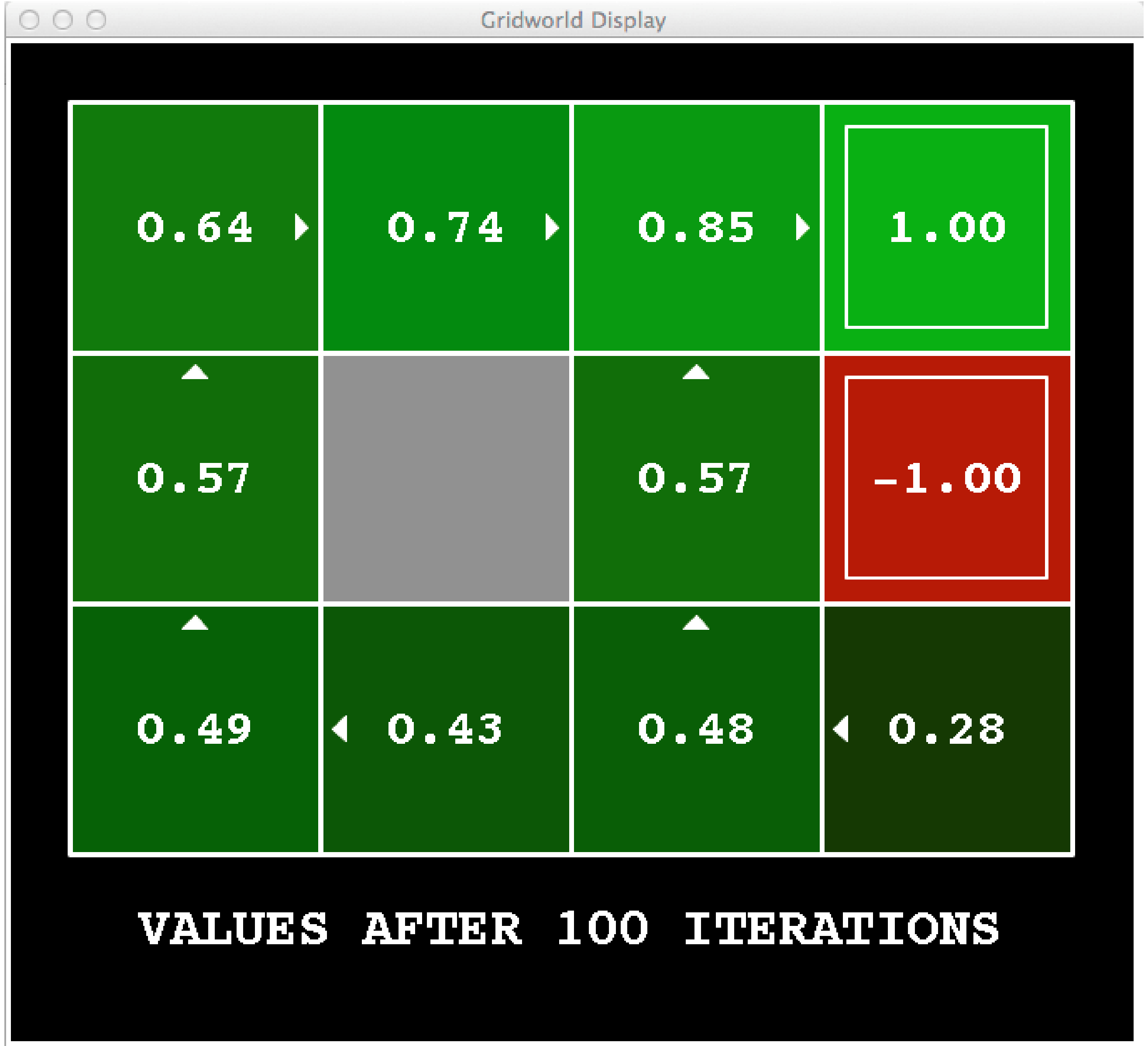
s is a *state*

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

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

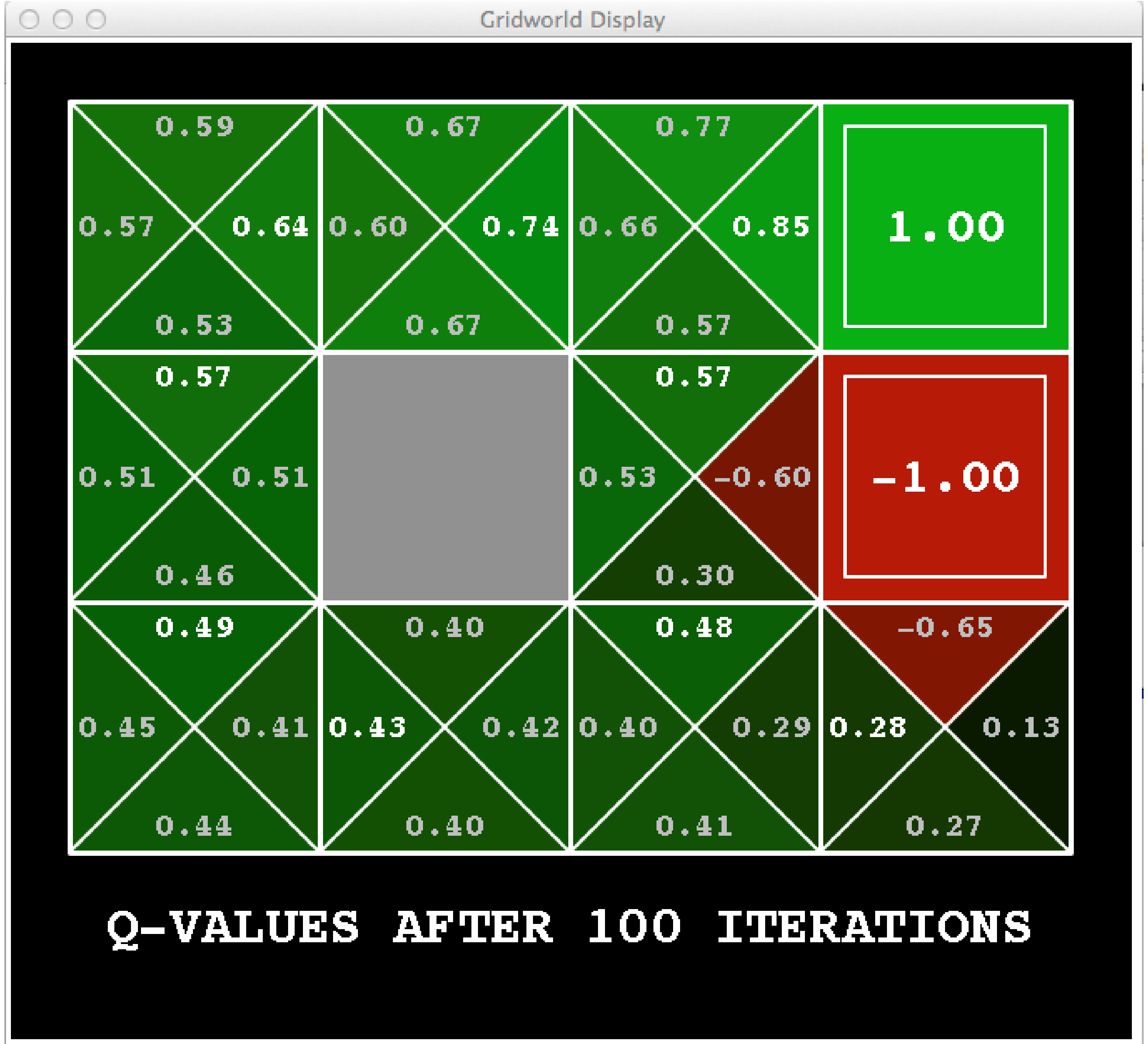
[Demo – gridworld values (L8D4)]

Snapshot of Demo – Gridworld V Values



Noise = 0.2
Discount = 0.9
Living reward = 0

Snapshot of Demo – Gridworld Q Values

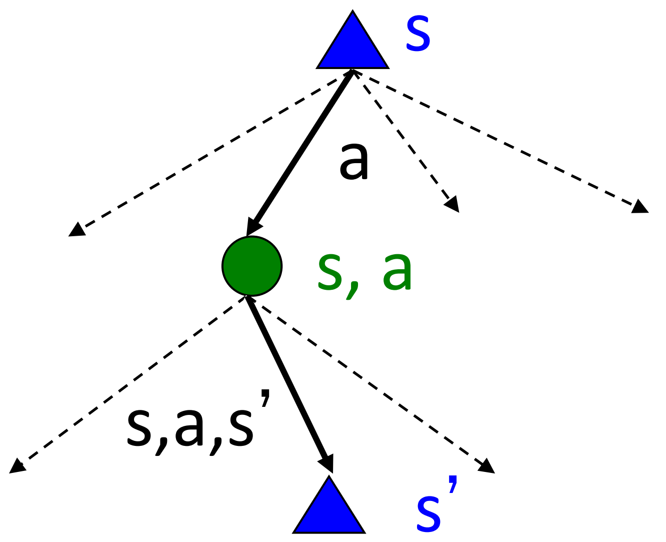


Noise = 0.2
Discount = 0.9
Living reward = 0

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:

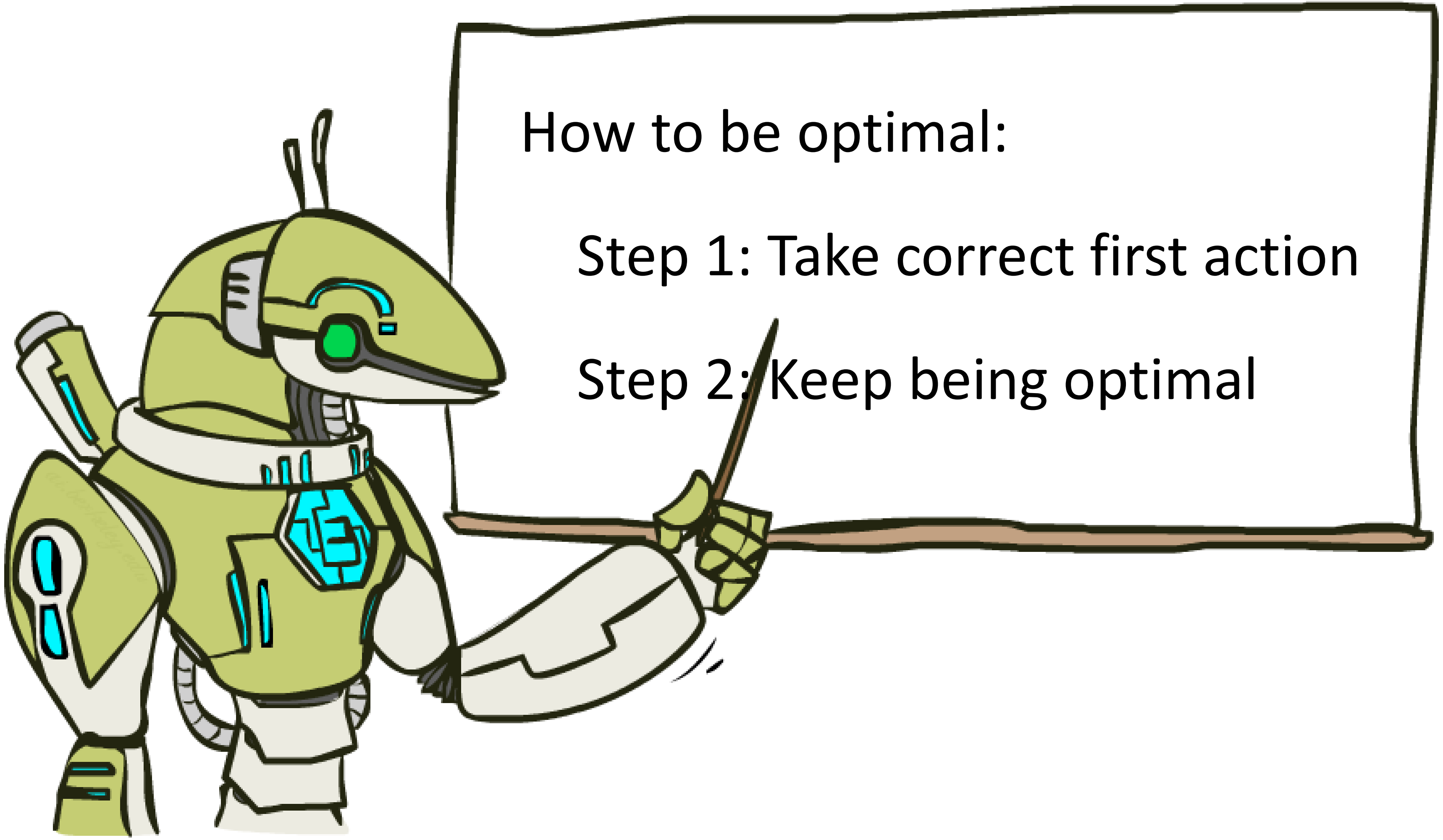
$$V^*(s) = \max_a Q^*(s,a)$$

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

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

Slide: ai.berkeley.edu

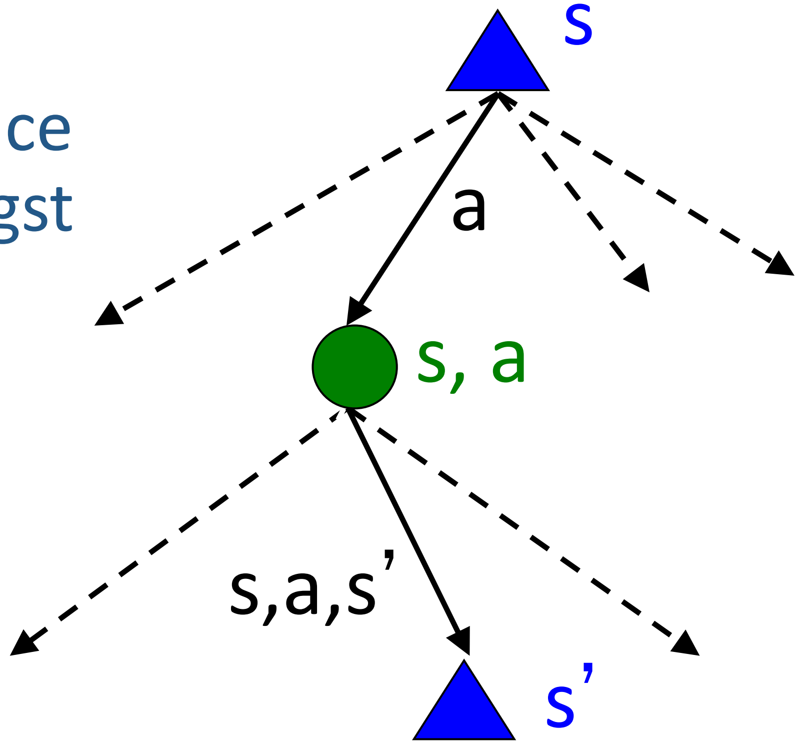
The Bellman Equations



Slide: ai.berkeley.edu

The Bellman Equations

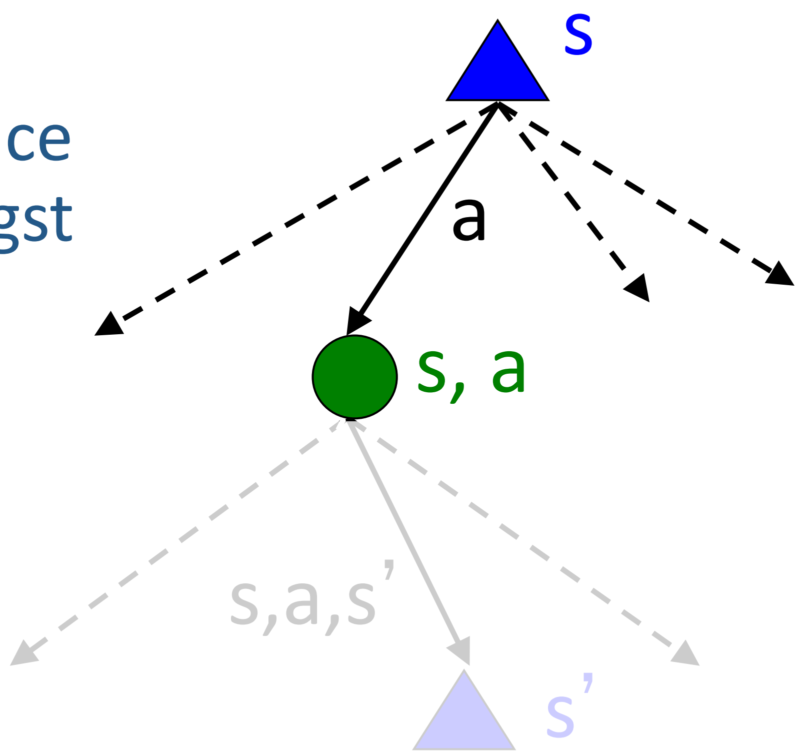
Definition of “optimal utility” via expectimax recurrence gives a simple one-step lookahead relationship amongst optimal utility values



The Bellman Equations

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

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

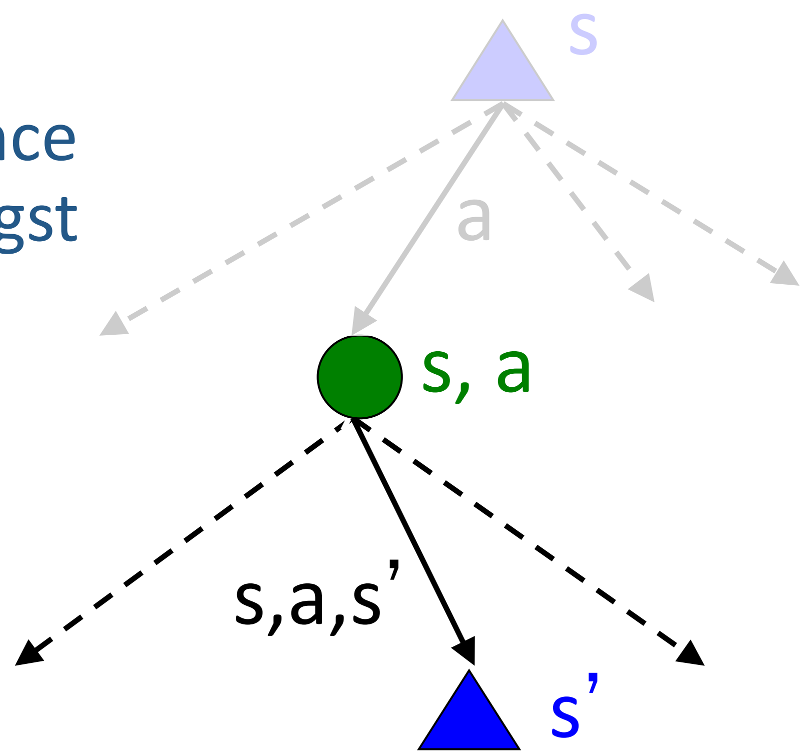


The Bellman Equations

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

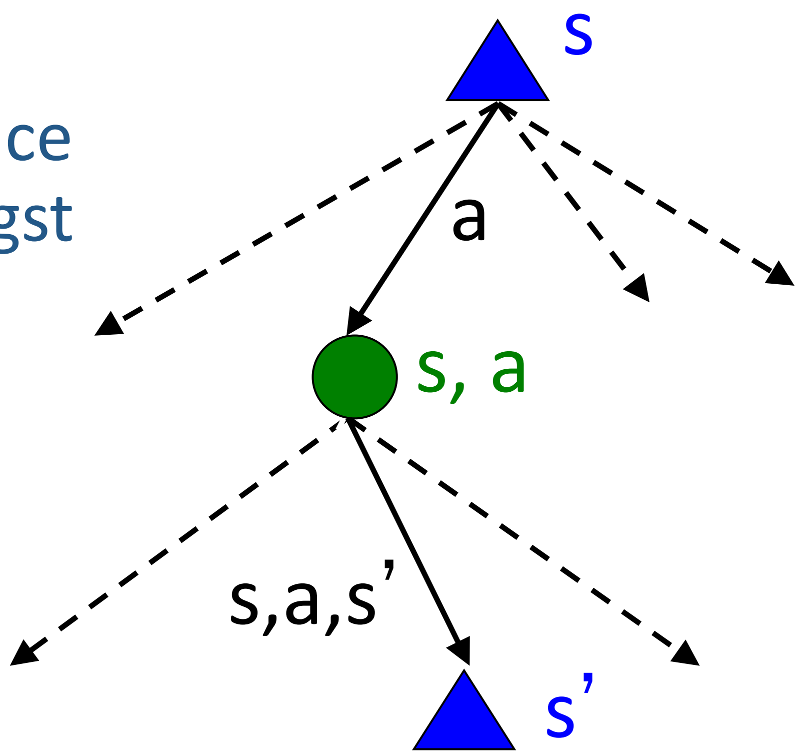
$$V^*(s) = \max_a Q^*(s,a)$$

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



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

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

Synchronous vs Asynchronous Value Iteration

Synchronous Value iteration

$k \leftarrow 0$

for s in $\mathcal{S}$

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

$k \leftarrow k + 1$

synchronous updates: compute all the fresh values of V(s) from all the stale values of V(s), then update V(s) with fresh values

Asynchronous Value iteration

$k \leftarrow 0$

for s in $\mathcal{S}$

$$V(s) \leftarrow \max_a \sum_{s'} P(s' \mid s, a)\,[R(s, a, s') + \gamma V(s')]$$

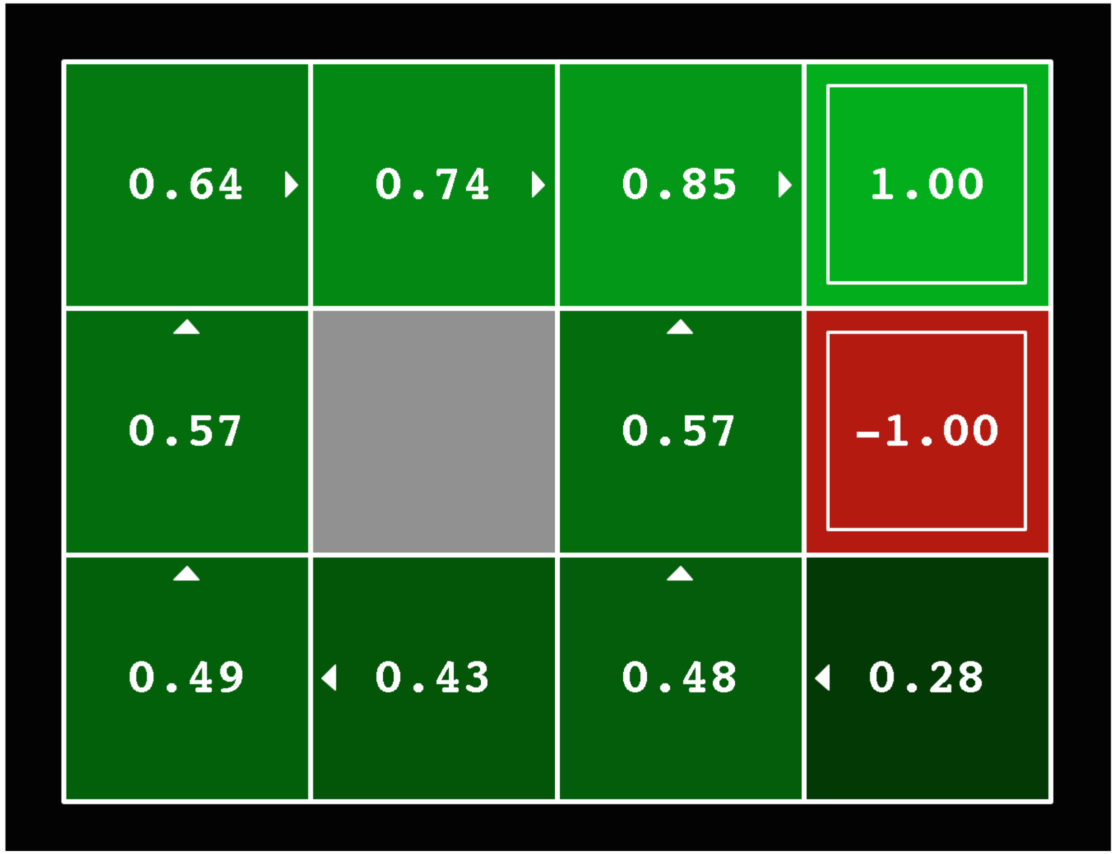
$k \leftarrow k + 1$

asynchronous updates: compute and update V(s) for each state one at a time

Solved MDP! Now what?

What are we going to do with these values??

$V^*(s)$



$Q^*(s,a)$