Markov Decision Processes

Lecture outline

- Stochastic domains (e.g. Grid World)
- MDP definitions
  - Transition functions
  - Policy
  - Reward functions and discounting
- Solving MDPs
  - Value iteration
    - Bellman equations
  - Policy iteration

Stochastic domains

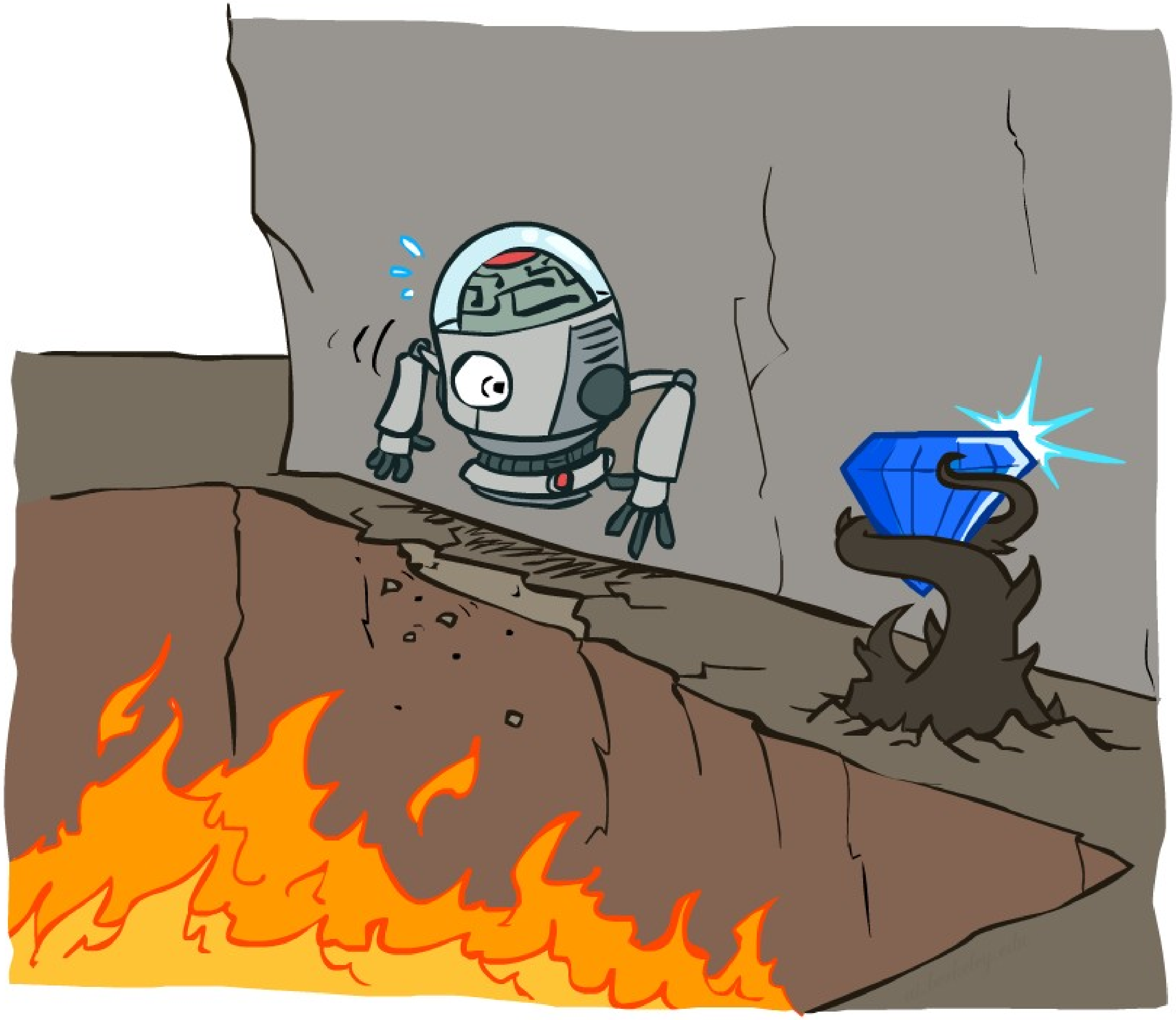
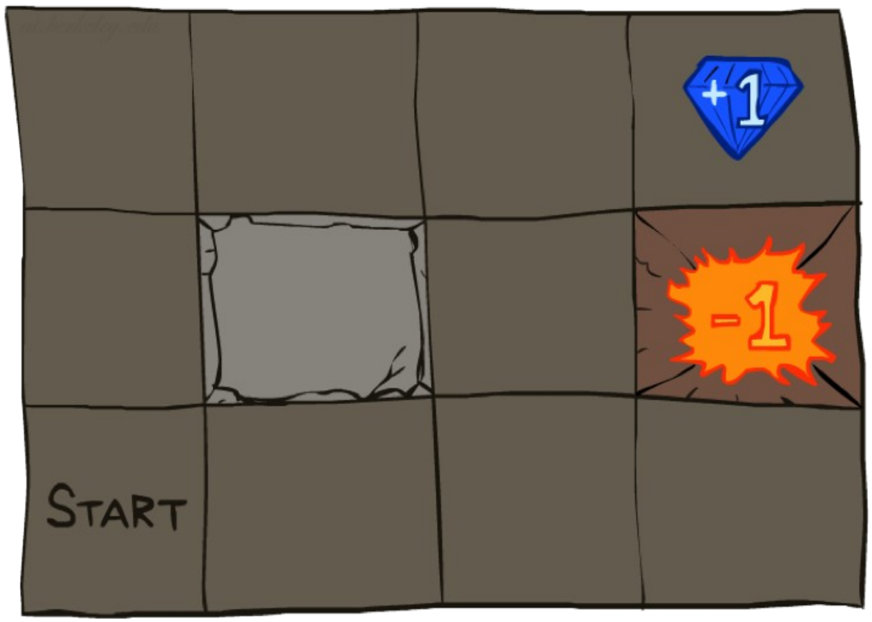


Image: Berkeley CS188 course notes (downloaded Summer 2015)

Example: stochastic grid world

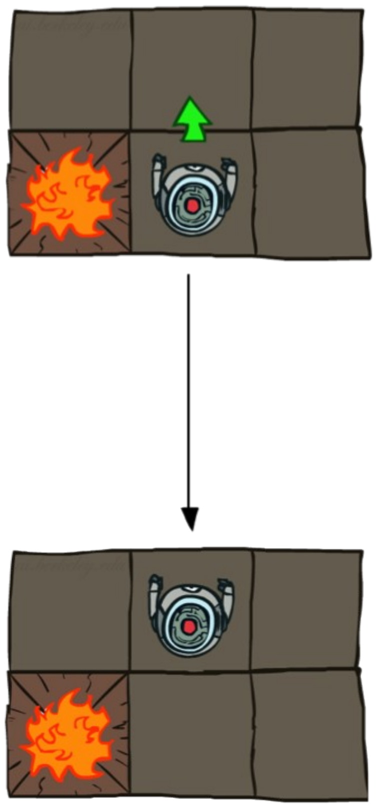
- A maze-like problem
  - The agent lives in a grid
  - Walls block the agent's path
- Noisy movement: actions do not always go as planned
  - 80% of the time, the action North takes the agent North
    (if there is no wall there)
  - 10% of the time, North takes the agent West; 10% East
  - If there is a wall in the direction the agent would have been taken, the agent stays put
- The agent receives rewards each time step
  - Reward function can be anything. For ex:
    - Small "living" reward each step (can be negative)
    - Big rewards come at the end (good or bad)
- Goal: maximize (discounted) sum of rewards



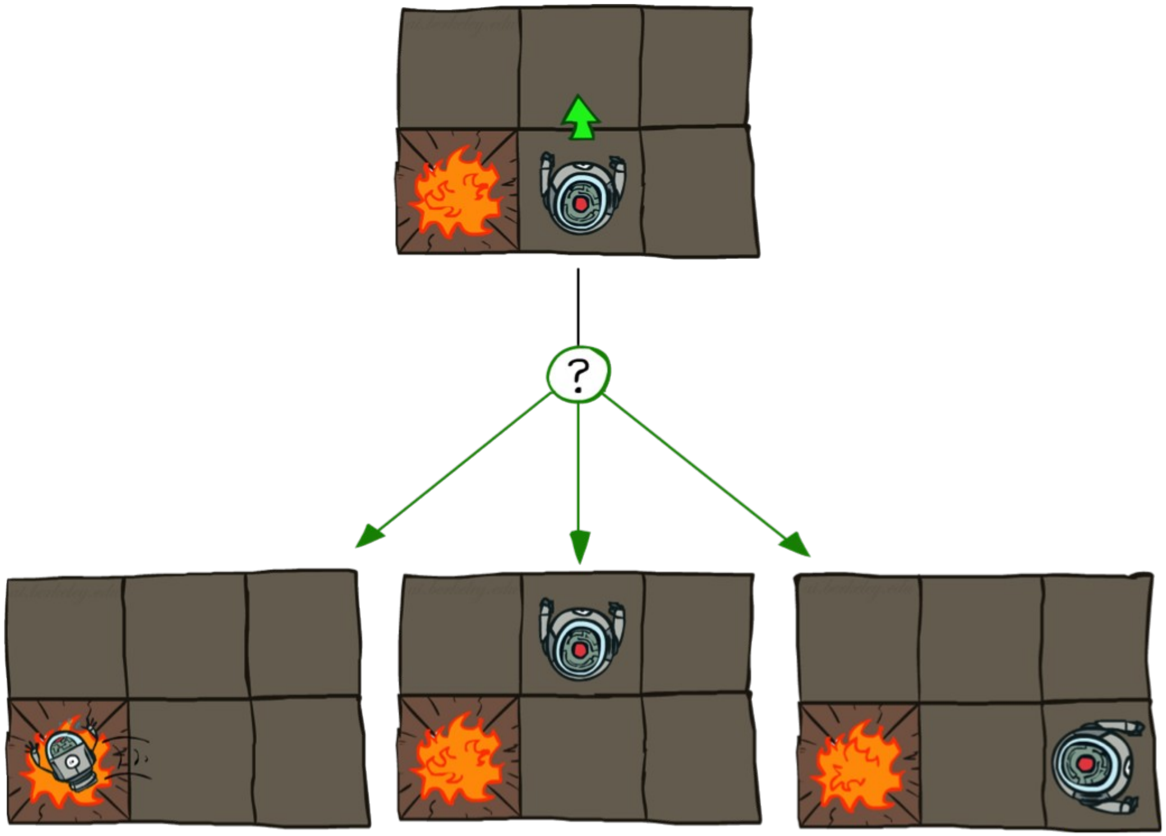
Slide: based on Berkeley CS188 course notes (downloaded Summer 2015)

Stochastic actions

Deterministic Grid World

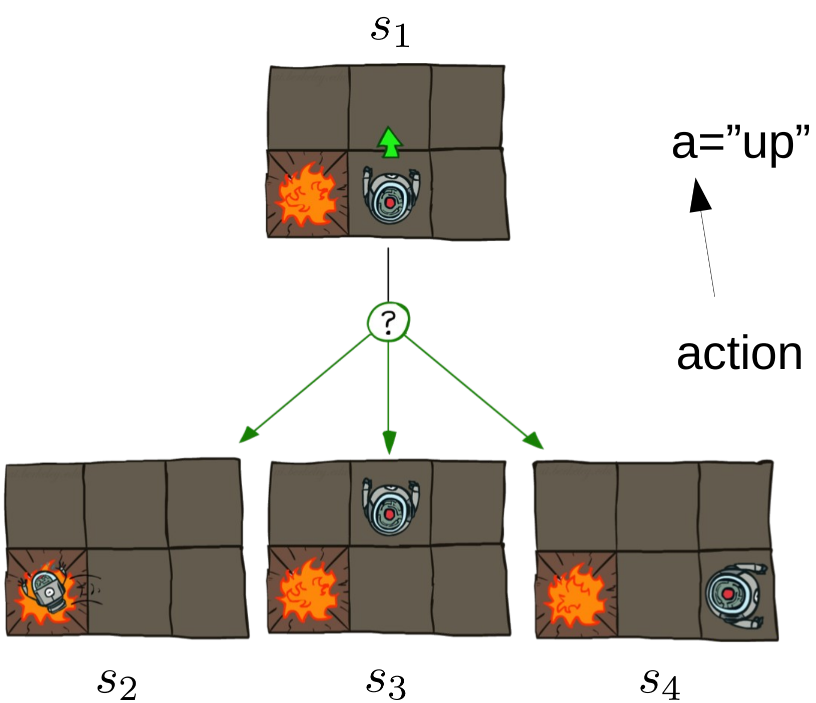


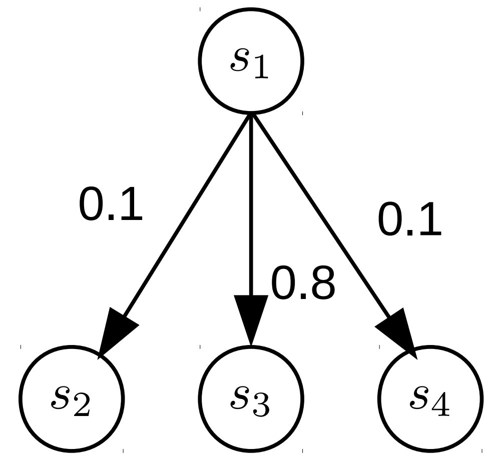
Stochastic Grid World



Slide: Berkeley CS188 course notes (downloaded Summer 2015)

The transition function



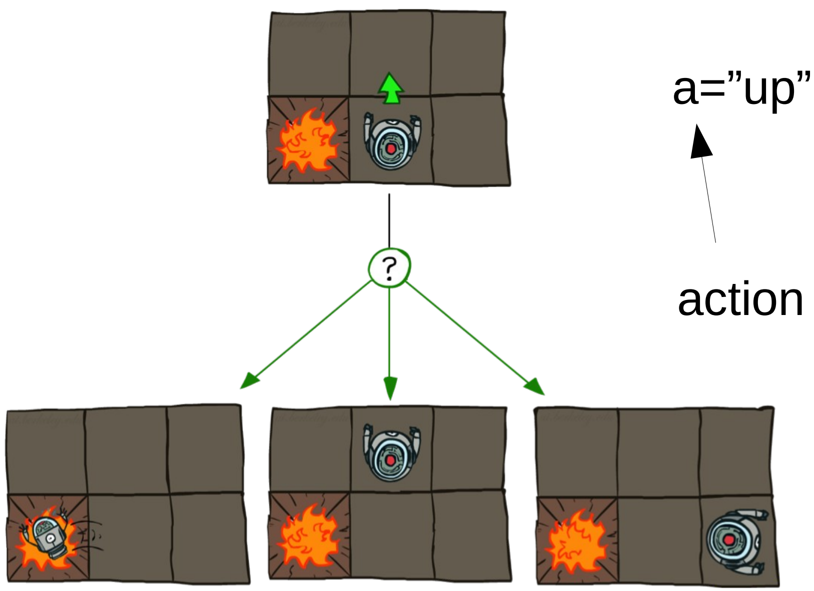


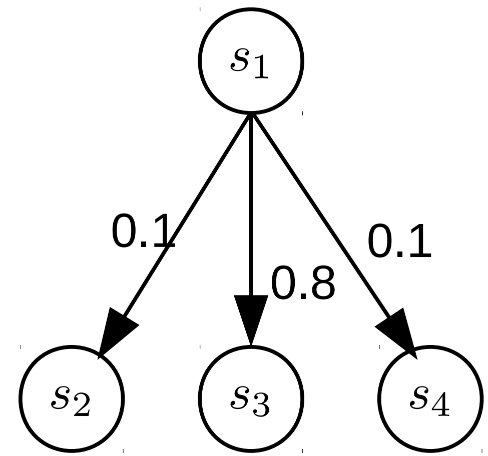
Transition probabilities:

s'	P(s' $\mid s_1, a$)
s_2	0.1
s_3	0.8
s_4	0.1

Image: Berkeley CS188 course notes (downloaded Summer 2015)

The transition function





Transition function: $T(s, a, s')$

– defines transition probabilities for each state,action pair

Transition probabilities:

s'	P(s' | s_1, a)
s_2	0.1
s_3	0.8
s_4	0.1

Image: Berkeley CS188 course notes (downloaded Summer 2015)

What is an *MDP*?

Technically, an MDP is a 4-tuple

An MDP (Markov Decision Process) defines a stochastic control problem: $M = (S, A, T, R)$

State set: $s \in S$

Action Set: $a \in A$

Transition function: $T : S \times A \times S \to \mathbb{R}_{\geq 0}$

Reward function: $R : S \times A \to \mathbb{R}_{\geq 0}$

What is an *MDP*?

Technically, an MDP is a 4-tuple

An MDP (Markov Decision Process) defines a stochastic control problem: $M = (S, A, T, R)$

State set: $s \in S$

Action Set: $a \in A$

Probability of going from *s* to *s'* when executing action *a*

Transition function: $T : S \times A \times S \to \mathbb{R}_{\geq 0}$

Reward function: $R : S \times A \to \mathbb{R}_{\geq 0}$

$$\sum_{s' \in S} T(s, a, s') = 1$$

What is an *MDP*?

Technically, an MDP is a 4-tuple

An MDP (Markov Decision Process) defines a stochastic control problem: $M = (S, A, T, R)$

State set: $s \in S$

Action Set: $a \in A$

Probability of going from *s* to *s'* when executing action *a*

Transition function: $T : S \times A \times S \to \mathbb{R}_{\geq 0}$

Reward function: $R : S \times A \to \mathbb{R}_{\geq 0}$

$$\sum_{s' \in S} T(s, a, s') = 1$$

But, what is the objective?

What is an *MDP*?

Technically, an MDP is a 4-tuple

An MDP (Markov Decision Process) defines a stochastic control problem: $M = (S, A, T, R)$

State set: $s \in S$

Action Set: $a \in A$

Probability of going from *s* to *s'* when executing action *a*

Transition function: $T : S \times A \times S \rightarrow \mathbb{R}_{\geq 0}$

Reward function: $R : S \times A \rightarrow \mathbb{R}_{\geq 0}$

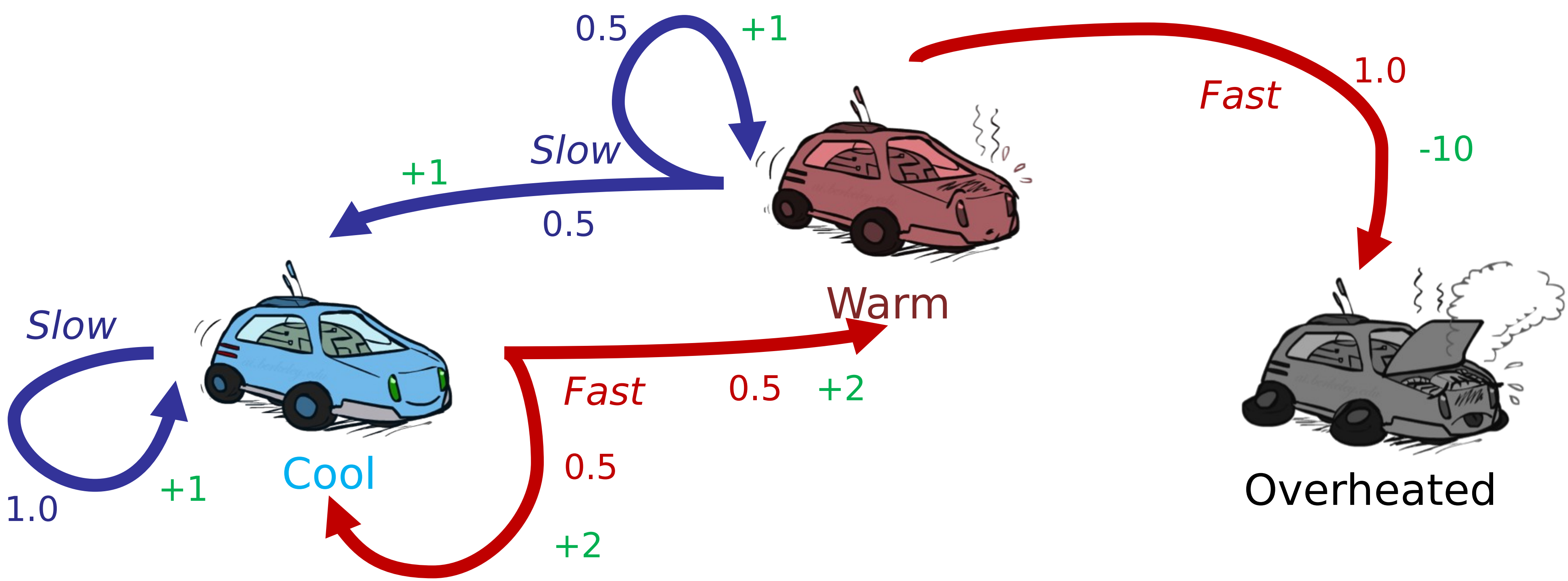
$$\sum_{s' \in S} T(s, a, s') = 1$$

<u>Objective</u>: calculate a strategy for acting so as to maximize the (discounted) sum of future rewards.

– we will calculate a *policy* that will tell us how to act

Example

- A robot car wants to travel far, quickly
- Three states: Cool, Warm, Overheated
- Two actions: *Slow*, *Fast*
- Going faster gets double reward



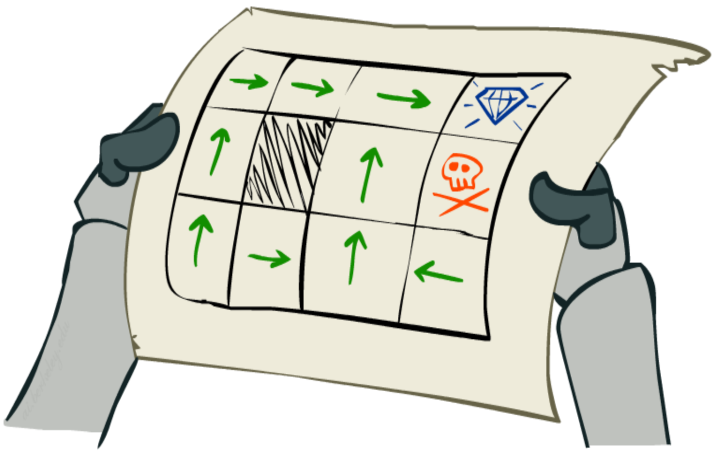
Slide: Berkeley CS188 course notes (downloaded Summer 2015)

What is a *policy*?

- In deterministic single-agent search problems, we wanted 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
  - An explicit policy defines a reflex agent

- Expectimax didn't compute entire policies
  - It computed the action for a single state only



This policy is optimal when R(s, a, s') = -0.03 for all non-terminal states

Slide: Berkeley CS188 course notes (downloaded Summer 2015)

Why is it Markov?

- “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

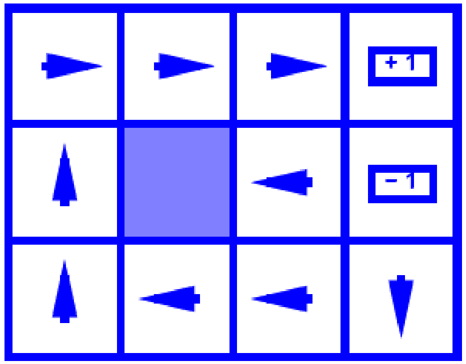
$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1}, \ldots S_0 = s_0)$$

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

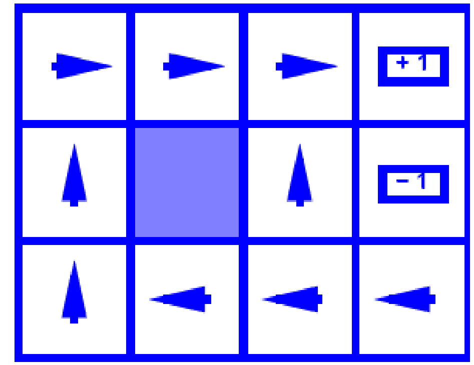
- This is just like search, where the successor function could only depend on the current state (not the history)

Andrey Markov
(1856-1922)

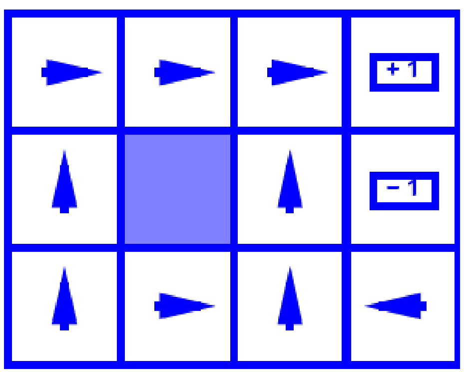
Examples of optimal policies



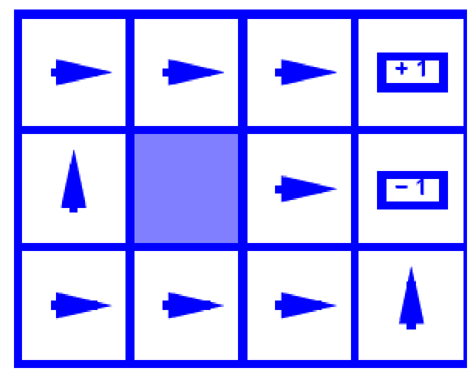
R(s) = -0.01



R(s) = -0.03



R(s) = -0.4



R(s) = -2.0

Slide: Berkeley CS188 course notes (downloaded Summer 2015)

How would we solve this using expectimax?

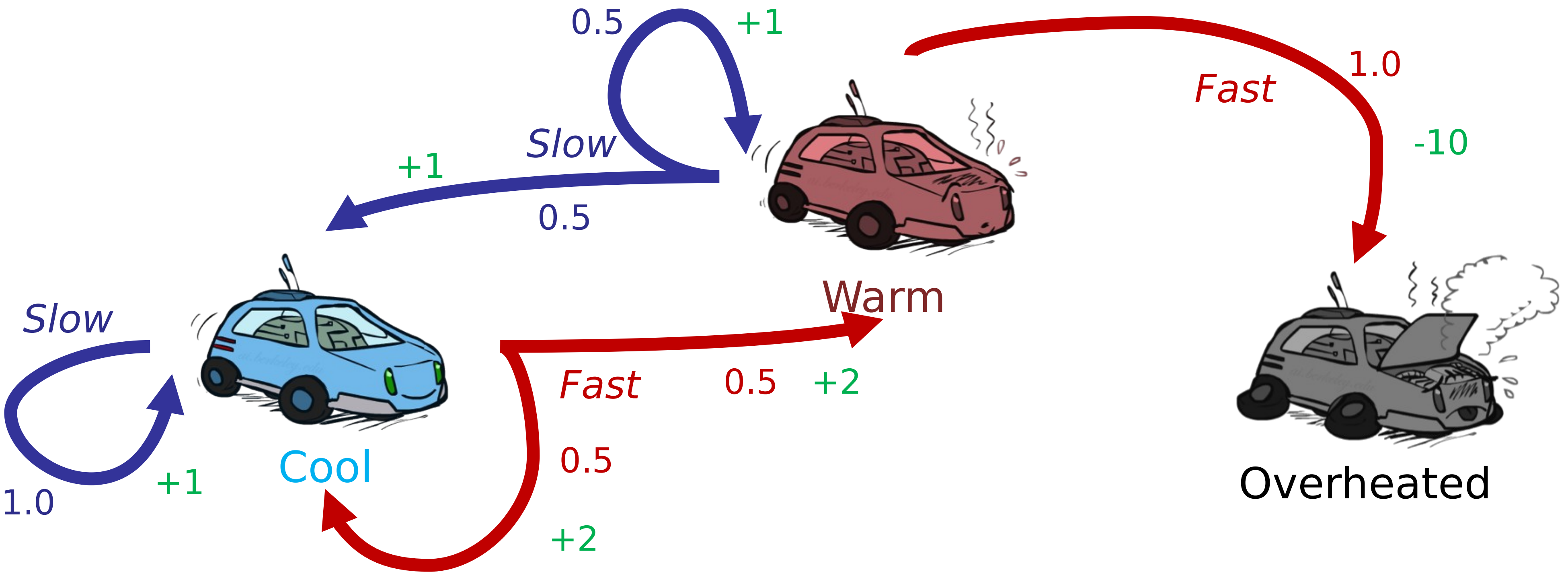
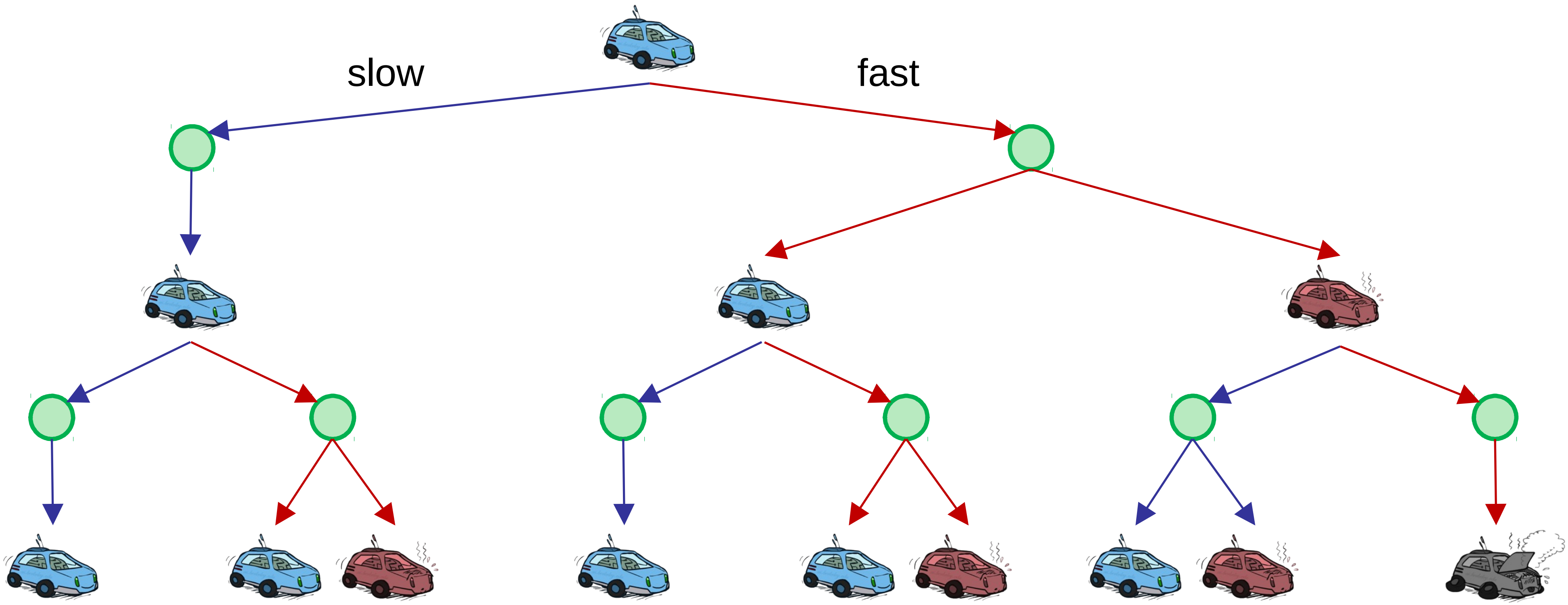


Image: Berkeley CS188 course notes (downloaded Summer 2015)

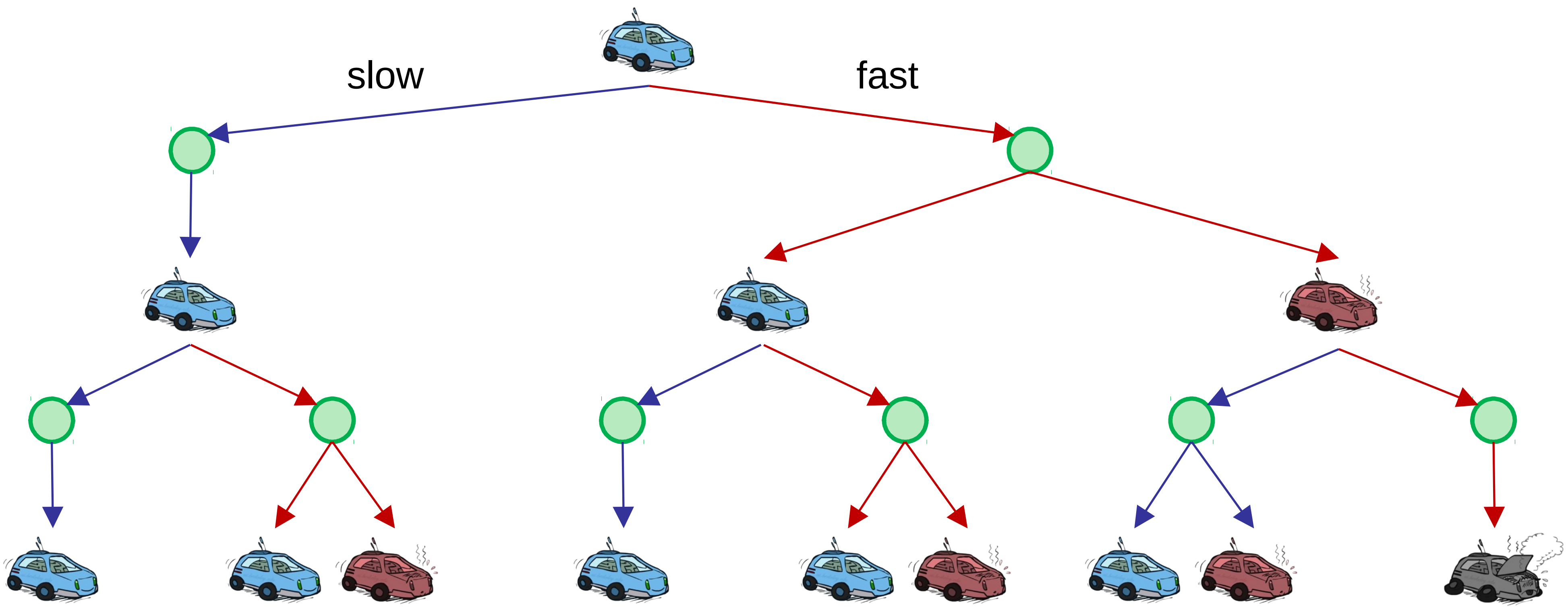
How would we solve this using expectimax?



Problems w/ this approach:
– how deep do we search?
– how do we deal w/ loops?

Image: Berkeley CS188 course notes (downloaded Summer 2015)

How would we solve this using expectimax?

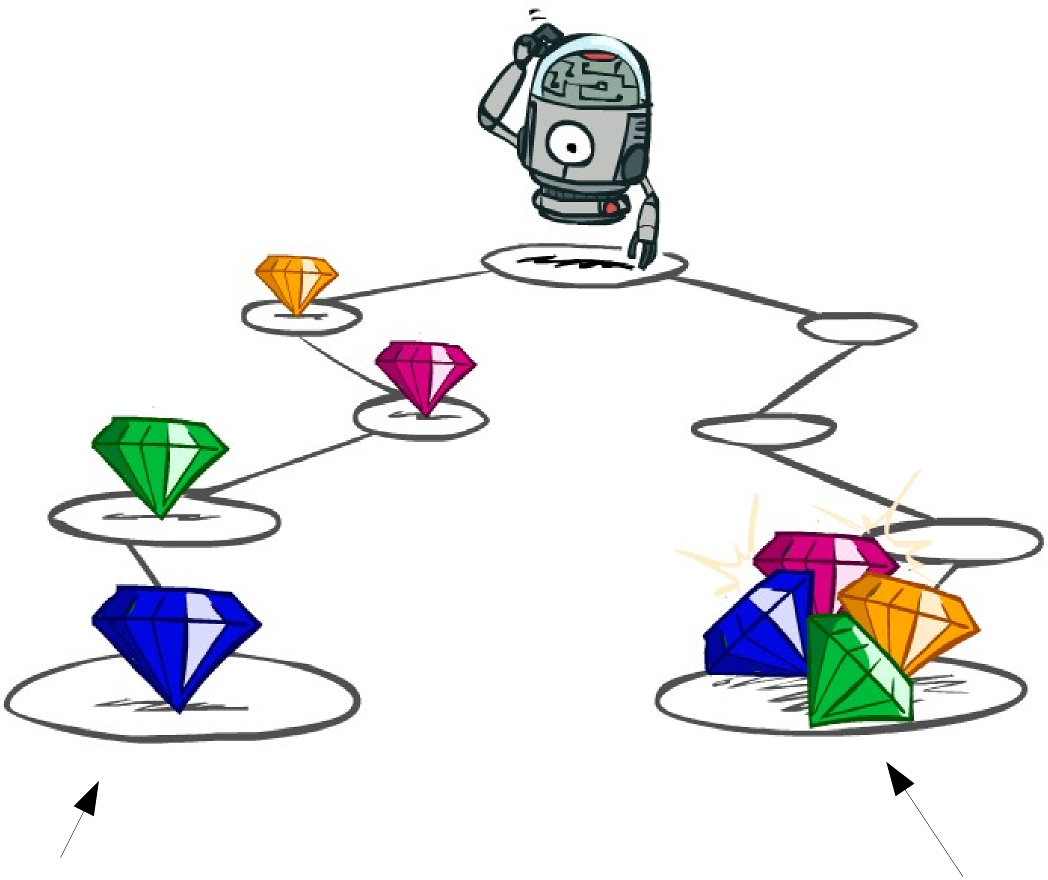


<u>Problems w/ this approach:</u>
– how deep do we search?
– how do we deal w/ loops?

Is there a better way?

Image: Berkeley CS188 course notes (downloaded Summer 2015)

Discounting rewards



Is this better?

Or is this better?

In general: how should we balance amount of reward vs how soon it is obtained?

Image: Berkeley CS188 course notes (downloaded Summer 2015)

Discounting rewards

- 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

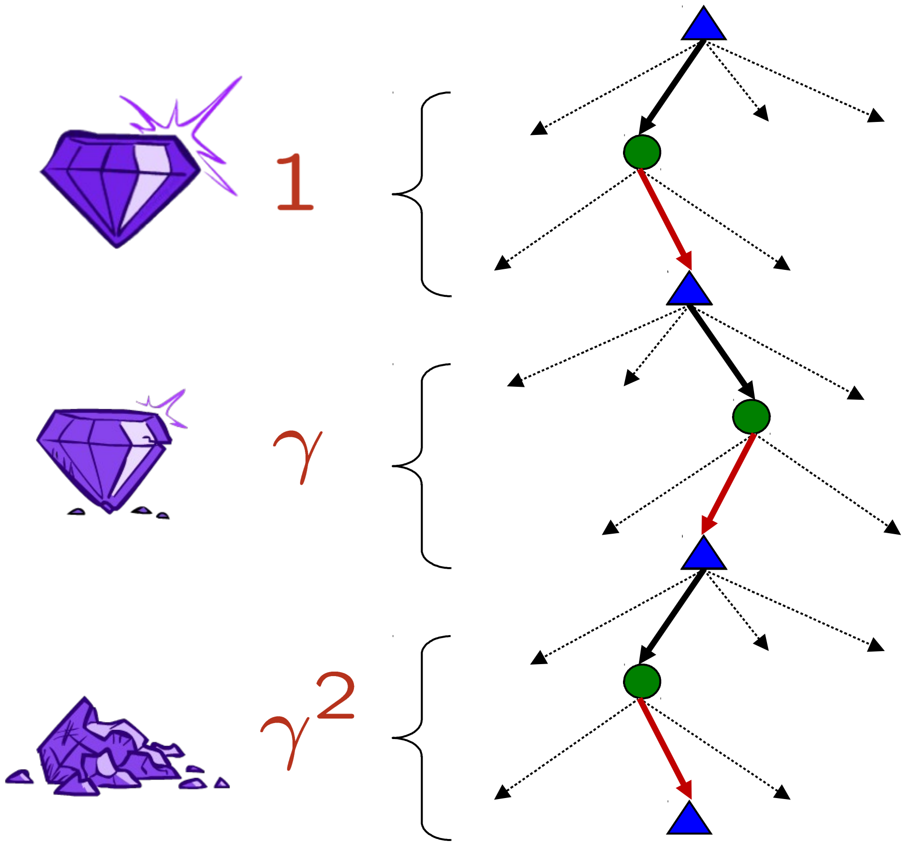
γ^2

Worth In Two Steps

Where, for example: $\gamma \approx 0.9$

Discounting rewards

- 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
- Example: discount of 0.5
  - U([1,2,3]) = 1*1 + 0.5*2 + 0.25*3
  - U([1,2,3]) < U([3,2,1])



Slide: Berkeley CS188 course notes (downloaded Summer 2015)

Discounting rewards

In general: $U_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots$

$$= \sum_{k=0}^{\infty} \gamma^k r_{t+1+k}$$

Utility

Choosing a reward function

<u>A few possibilities:</u>

– all reward on goal/firepit
– negative reward everywhere
  except terminal states
– gradually increasing reward
  as you approach the goal

<u>In general:</u>

– reward can be whatever you
  want

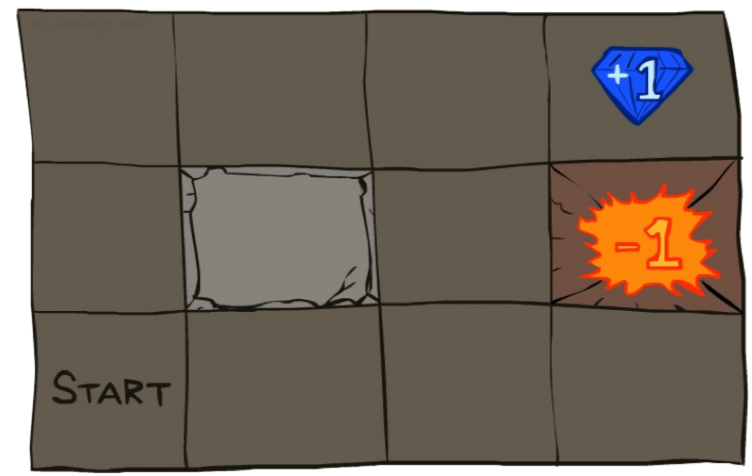


Image: Berkeley CS188 course notes (downloaded Summer 2015)

Discounting example

- Given:

10				1
a	b	c	d	e

  - Actions: East, West, and Exit (only available in exit states a, e)
  - Transitions: deterministic

- Quiz 1: For $\gamma = 1$, what is the optimal policy?

10				1

- Quiz 2: For $\gamma = 0.1$, what is the optimal policy?

10				1

- Quiz 3: For which γ are West and East equally good when in state d?

Slide: Berkeley CS188 course notes (downloaded Summer 2015)

Solving MDPs

- The value (utility) of a state s:

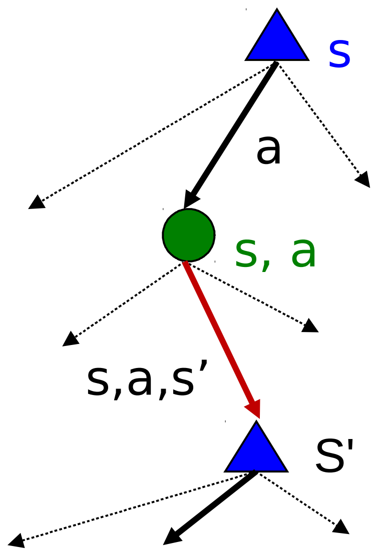
  $V^*(s)$ = expected utility starting in s and acting optimally

- 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

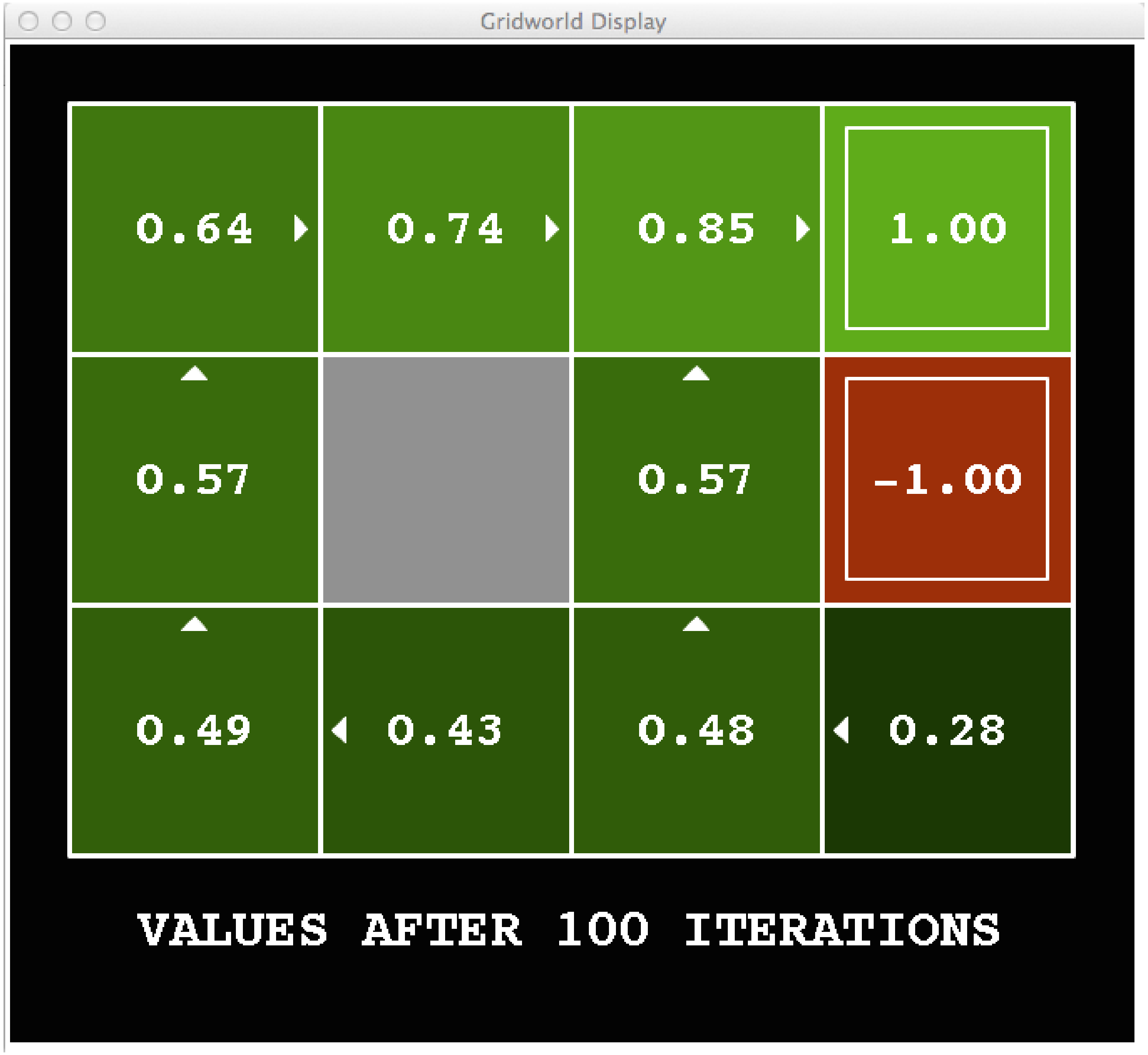


s is a *state*

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

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

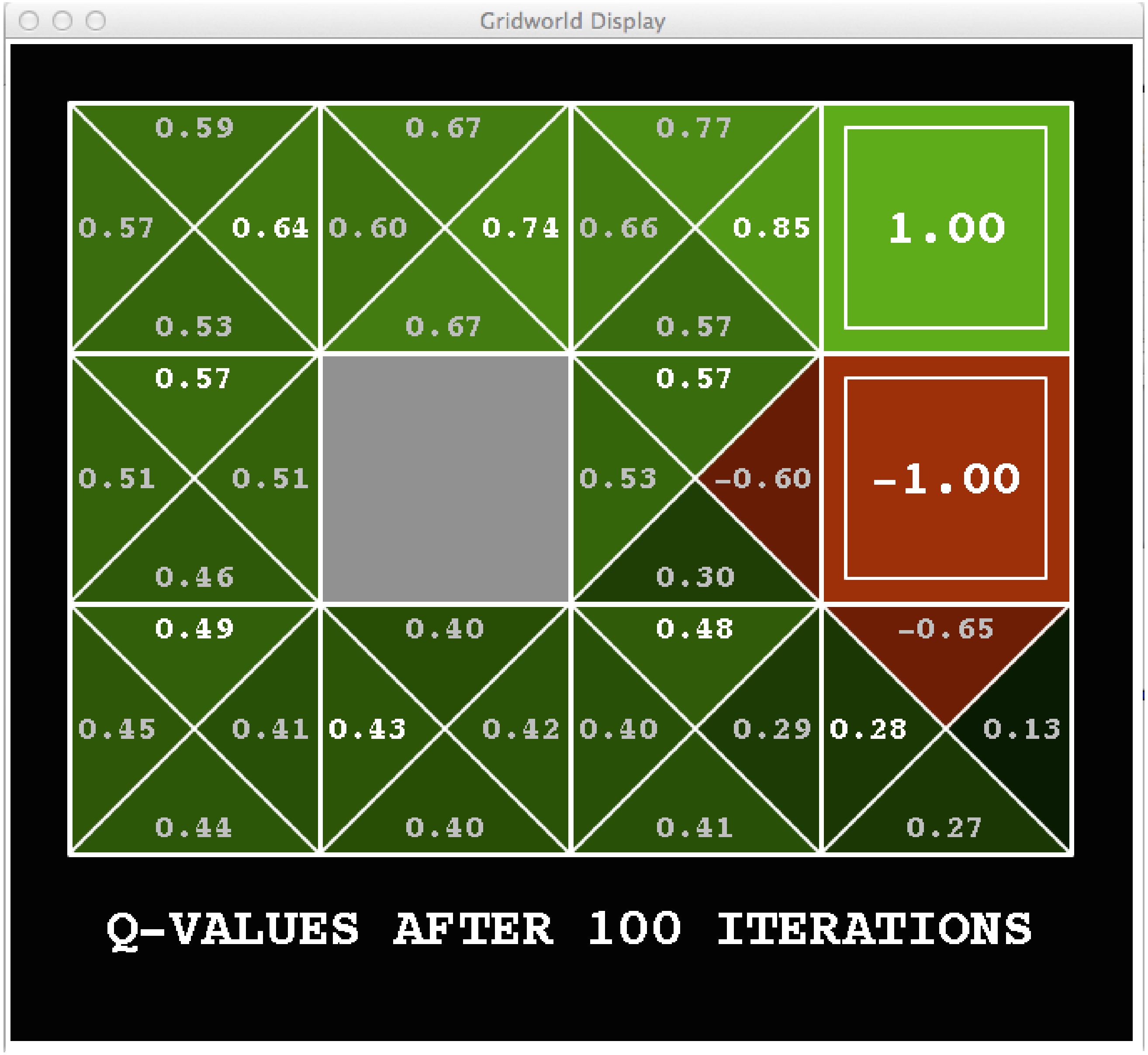
Snapshot of Demo – Gridworld V Values



Noise = 0.2
Discount = 0.9
Living reward = 0

Slide: Berkeley CS188 course notes (downloaded Summer 2015)

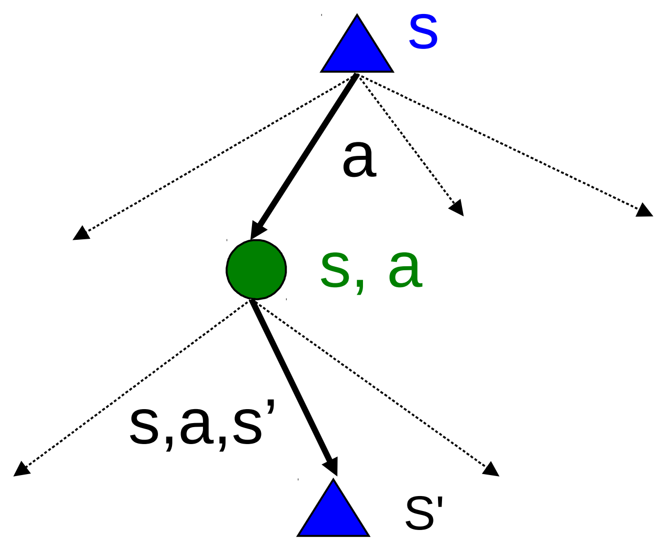
Snapshot of Demo – Gridworld V Values



Noise = 0.2
Discount = 0.9
Living reward = 0

Value iteration

We're going to calculate V* and/or Q* by repeatedly doing one-step expectimax.



Notice that the V* and Q* can be defined recursively:

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

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

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

Called Bellman equations

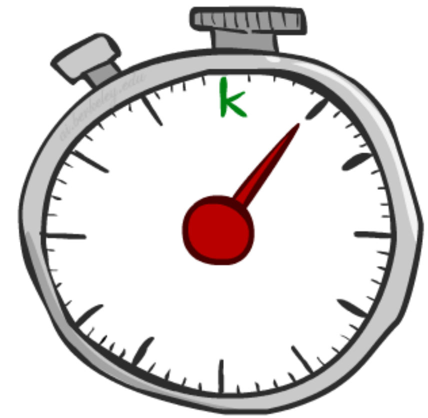
– note that the above do not reference the optimal policy, π^*

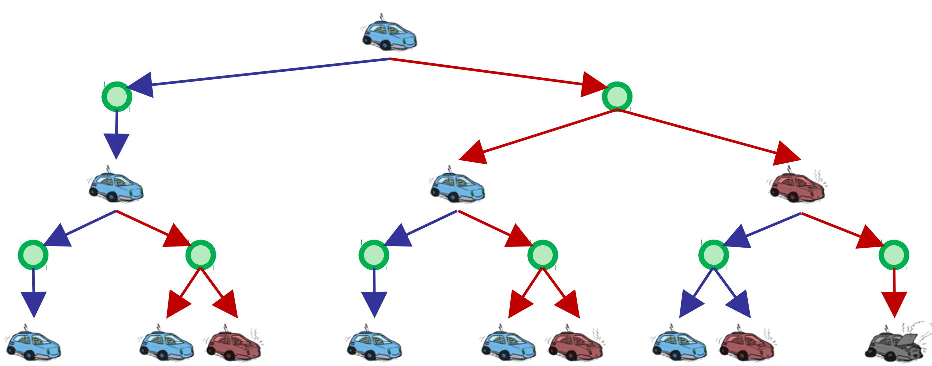
Slide: Derived from Berkeley CS188 course notes (downloaded Summer 2015)

Value iteration

- 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





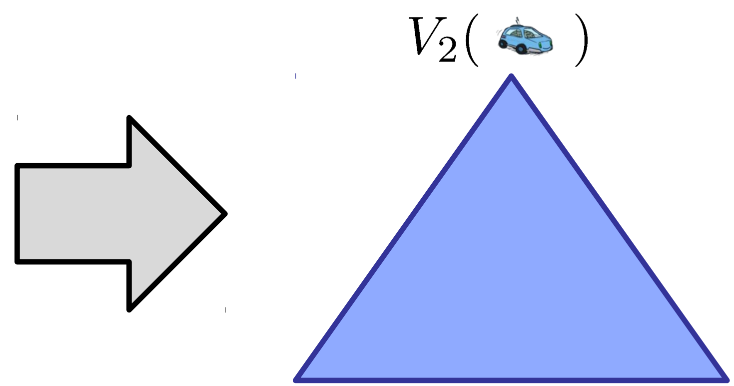


Image: Berkeley CS188 course notes (downloaded Summer 2015)

Value iteration

Value of *s* at *k* timesteps to go: $V_k(s)$

<u>Value iteration:</u>

1. initialize $V_0(s) = 0$
2. $V_1(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_0(s') \right]$
3. $V_2(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_1(s') \right]$
4. ....
5. $V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right]$

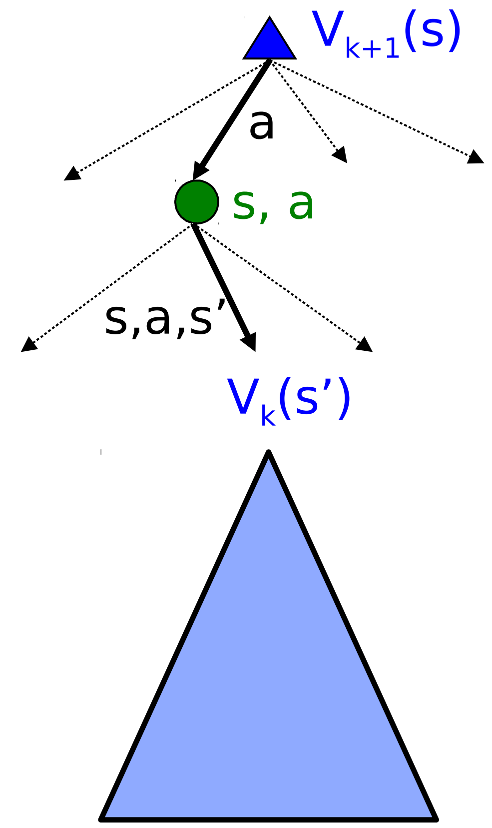
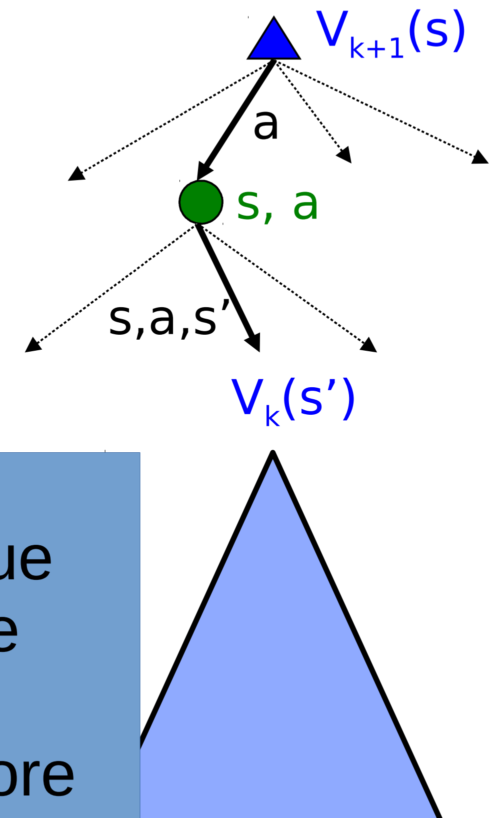


Image: Berkeley CS188 course notes (downloaded Summer 2015)

Value iteration

Value of *s* at *k* timesteps to go: $V_k(s)$



<u>Value iteration:</u>

1. initialize $V_0(s) = 0$

2. $V_1(s) \leftarrow$

3. $V_2(s) \leftarrow$

4. ....

5. $V_{k+1}(s)$

– This iteration converges! The value of each state converges to a unique optimal value.

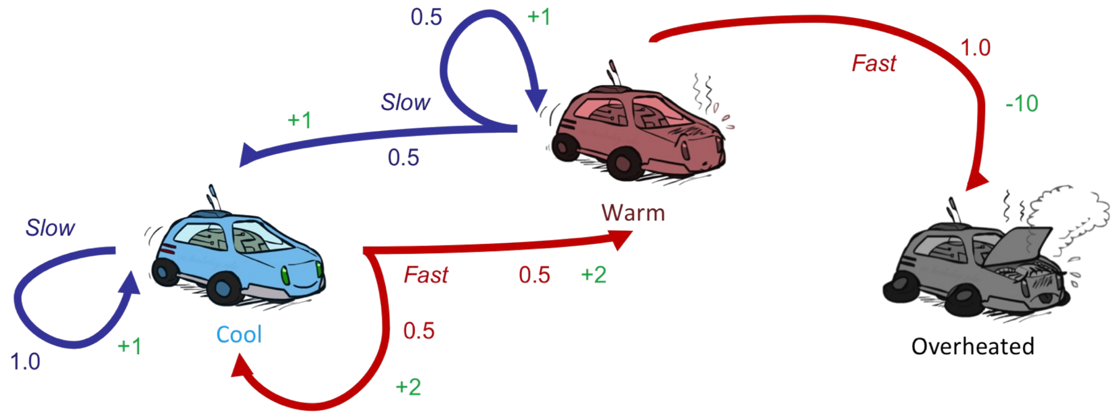
– policy typically converges before value function converges...

– time complexity = O(S^2 A)

Image: Berkeley CS188 course notes (downloaded Summer 2015)

Value iteration example

V_2			
V_1			
V_0	0	0	0



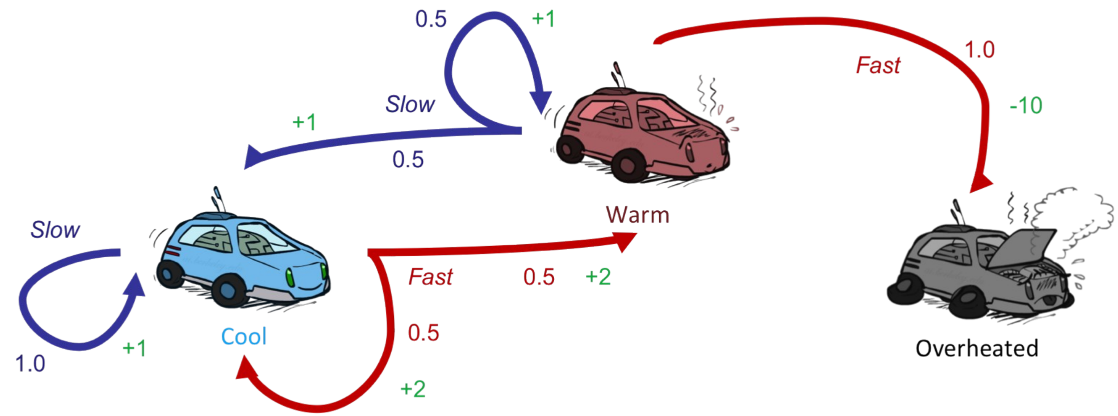
Assume no discount

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

Slide: Berkeley CS188 course notes (downloaded Summer 2015)

Value iteration example

V_2			
V_1	2	1	0
V_0	0	0	0



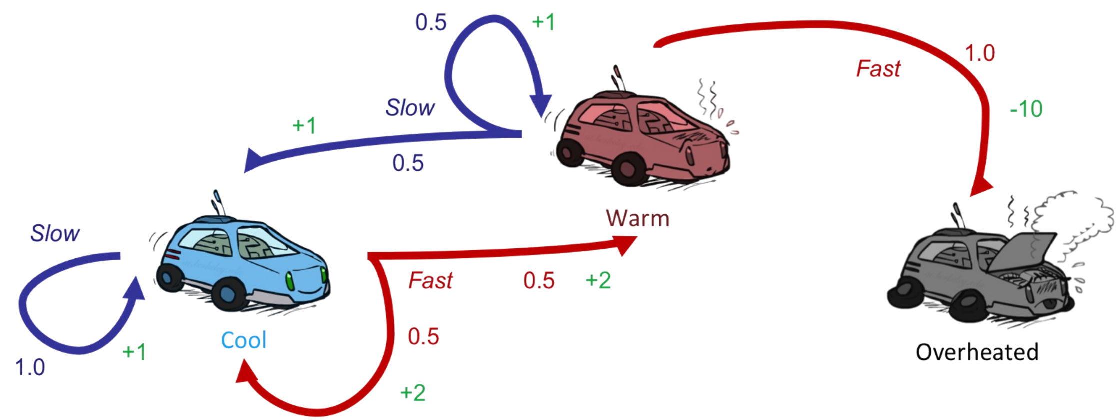
Assume no discount

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

Slide: Berkeley CS188 course notes (downloaded Summer 2015)

Value iteration example

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



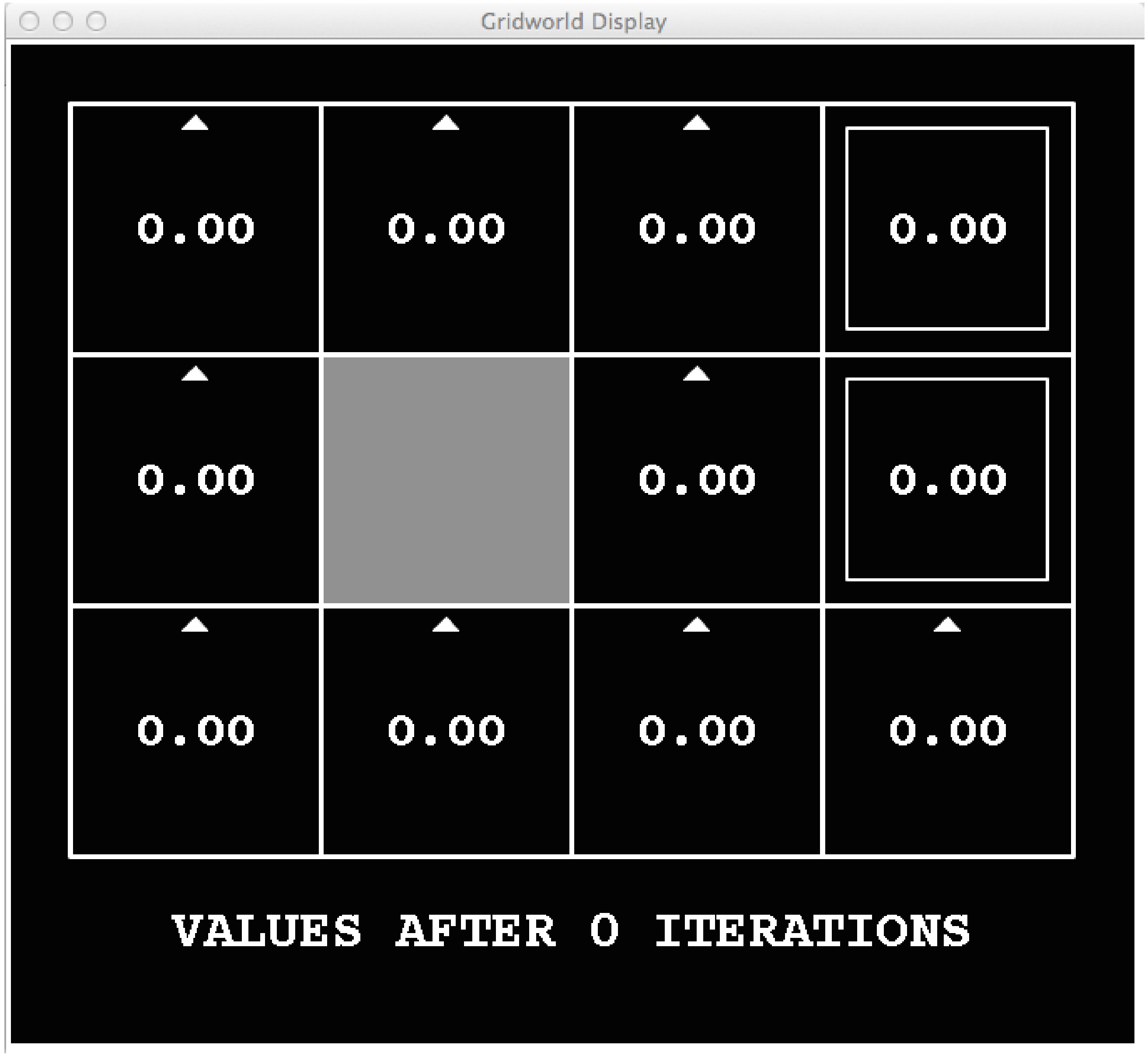
Assume no discount

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

Slide: Berkeley CS188 course notes (downloaded Summer 2015)

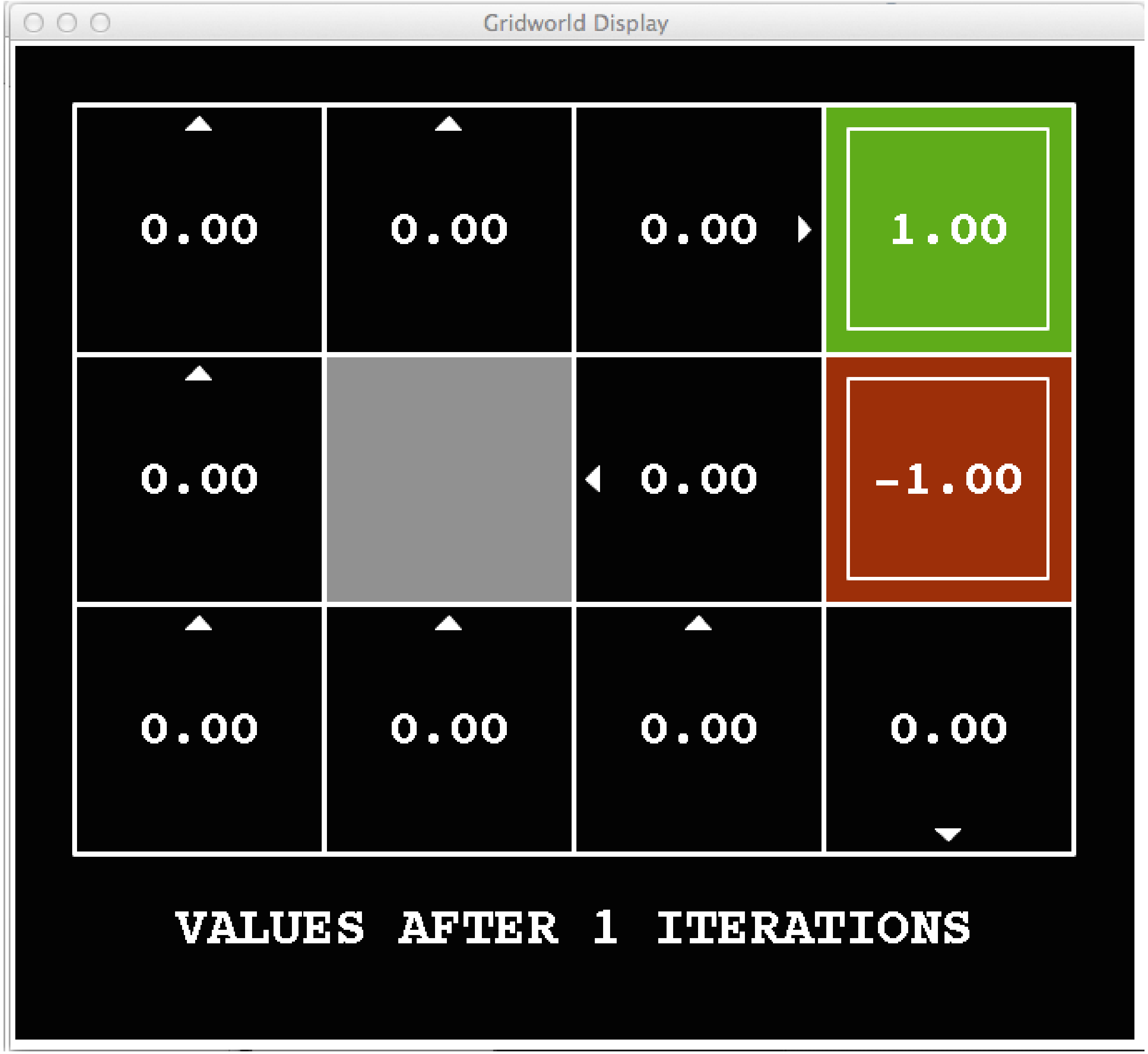
Value iteration example

Noise = 0.2
Discount = 0.9
Living reward = 0



Slide: Berkeley CS188 course notes (downloaded Summer 2015)

Value iteration example



Slide: Berkeley CS188 course notes (downloaded Summer 2015)

Value iteration example

Slide: Berkeley CS188 course notes (downloaded Summer 2015)

Value iteration example

Slide: Berkeley CS188 course notes (downloaded Summer 2015)

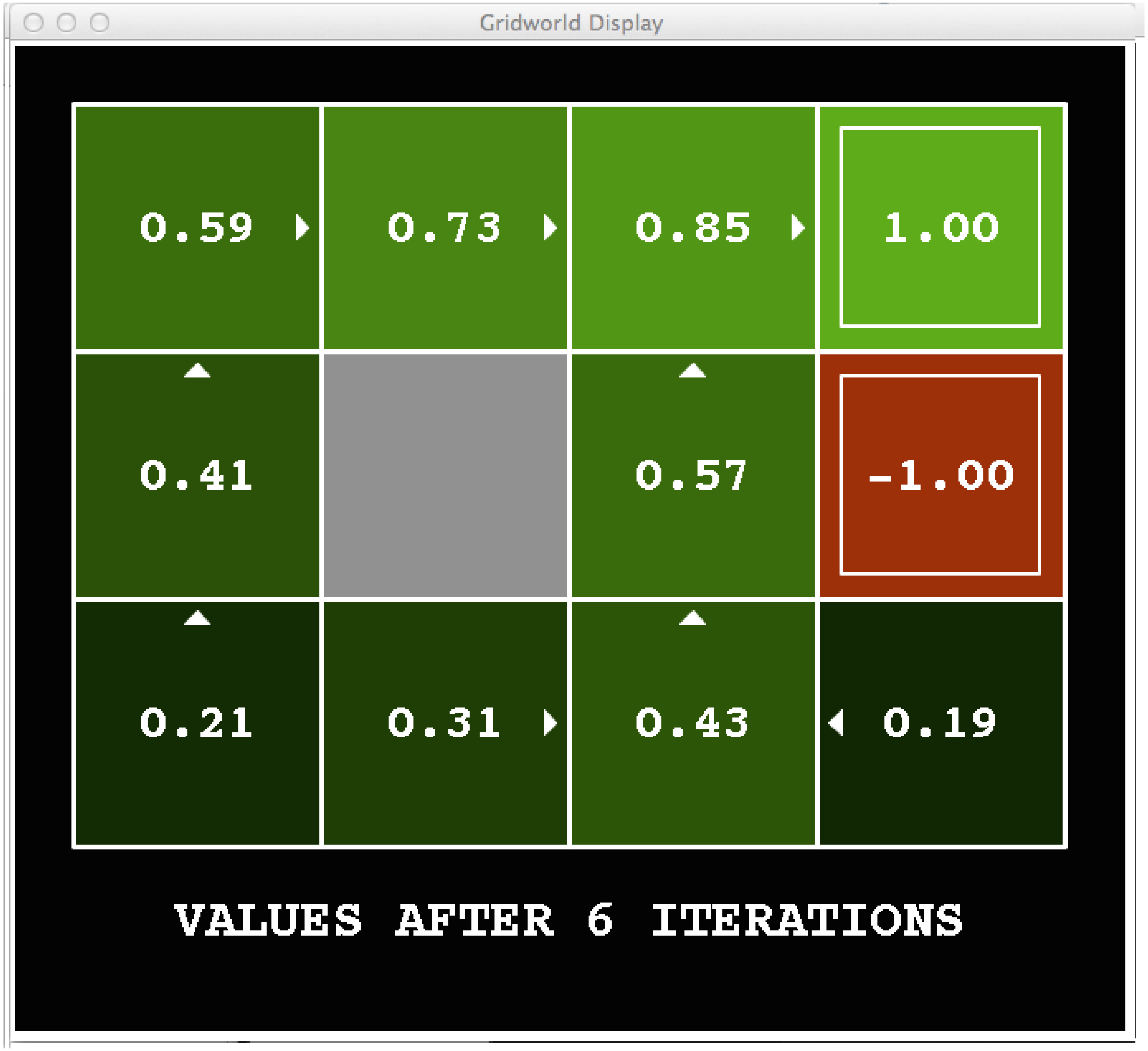
Value iteration example

Slide: Berkeley CS188 course notes (downloaded Summer 2015)

Value iteration example

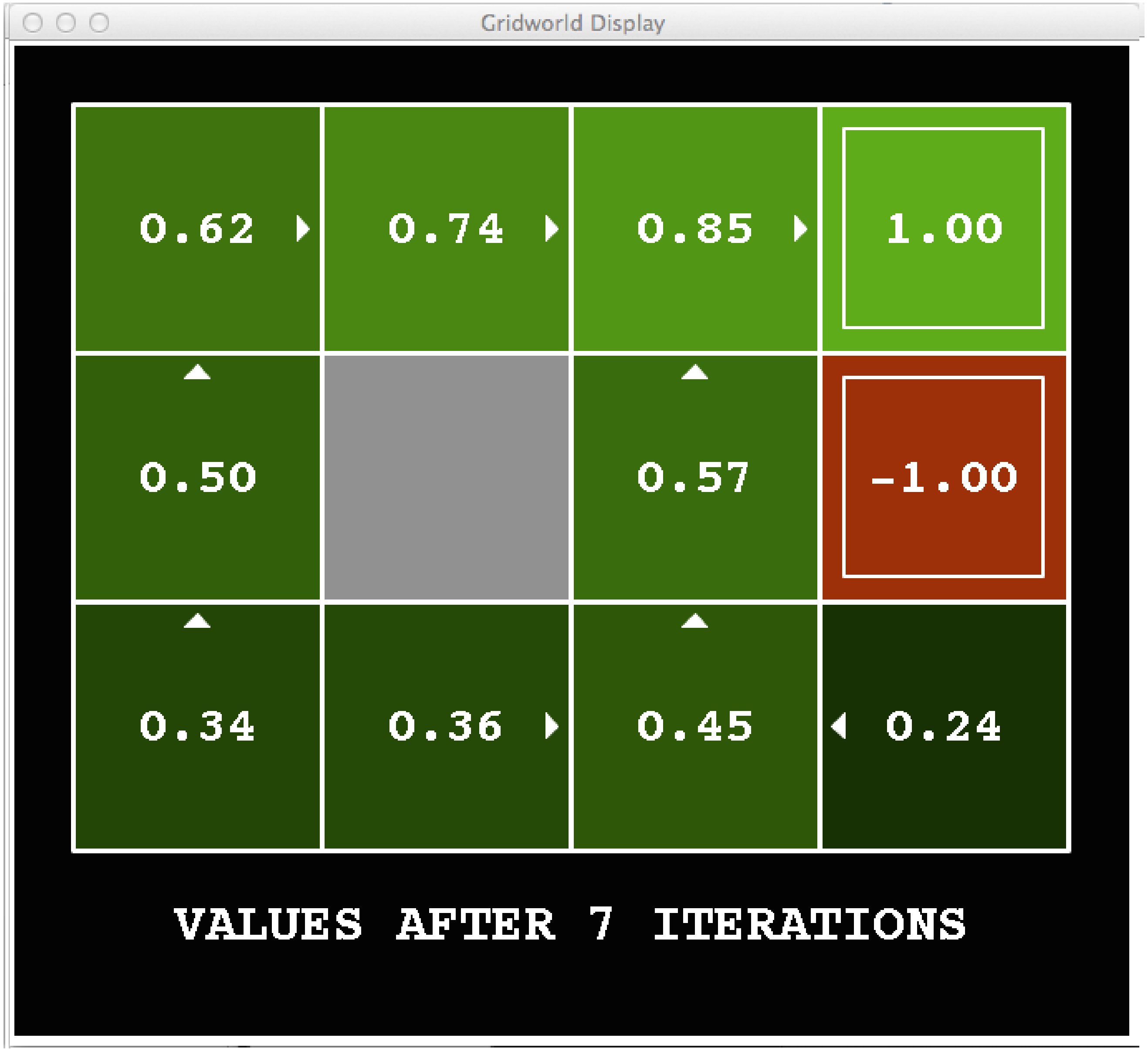
Slide: Berkeley CS188 course notes (downloaded Summer 2015)

Value iteration example



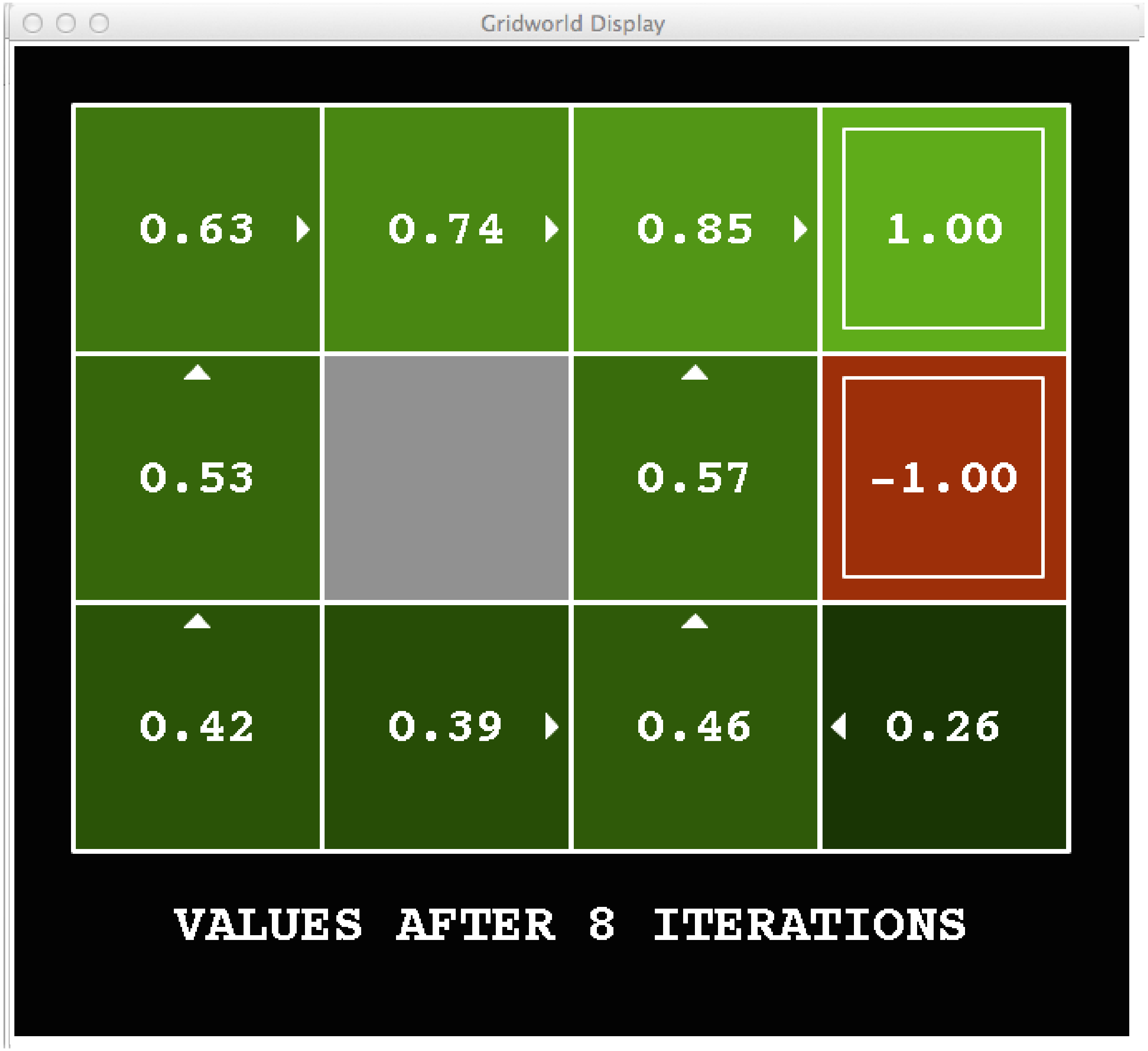
Slide: Berkeley CS188 course notes (downloaded Summer 2015)

Value iteration example



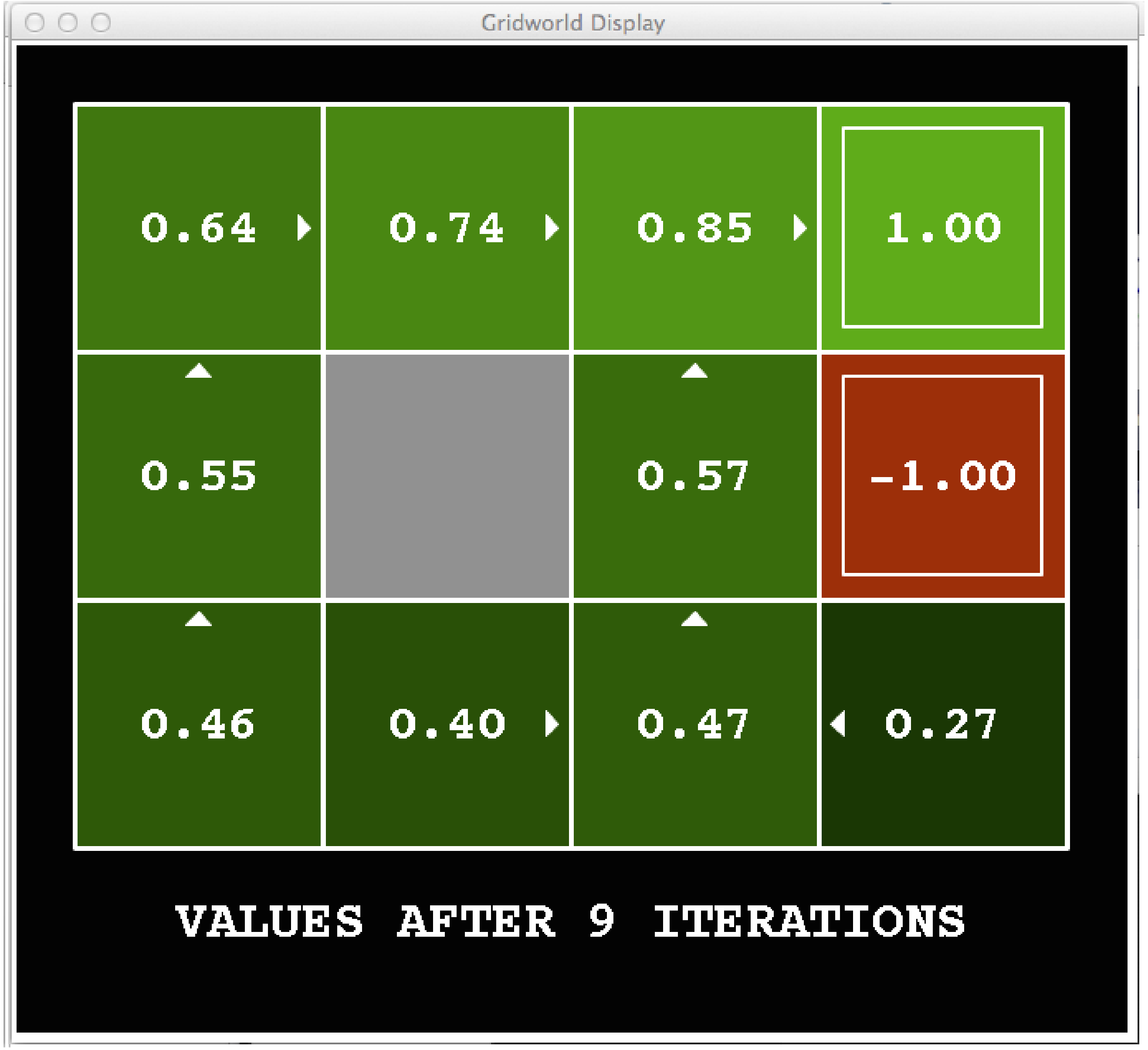
Slide: Berkeley CS188 course notes (downloaded Summer 2015)

Value iteration example



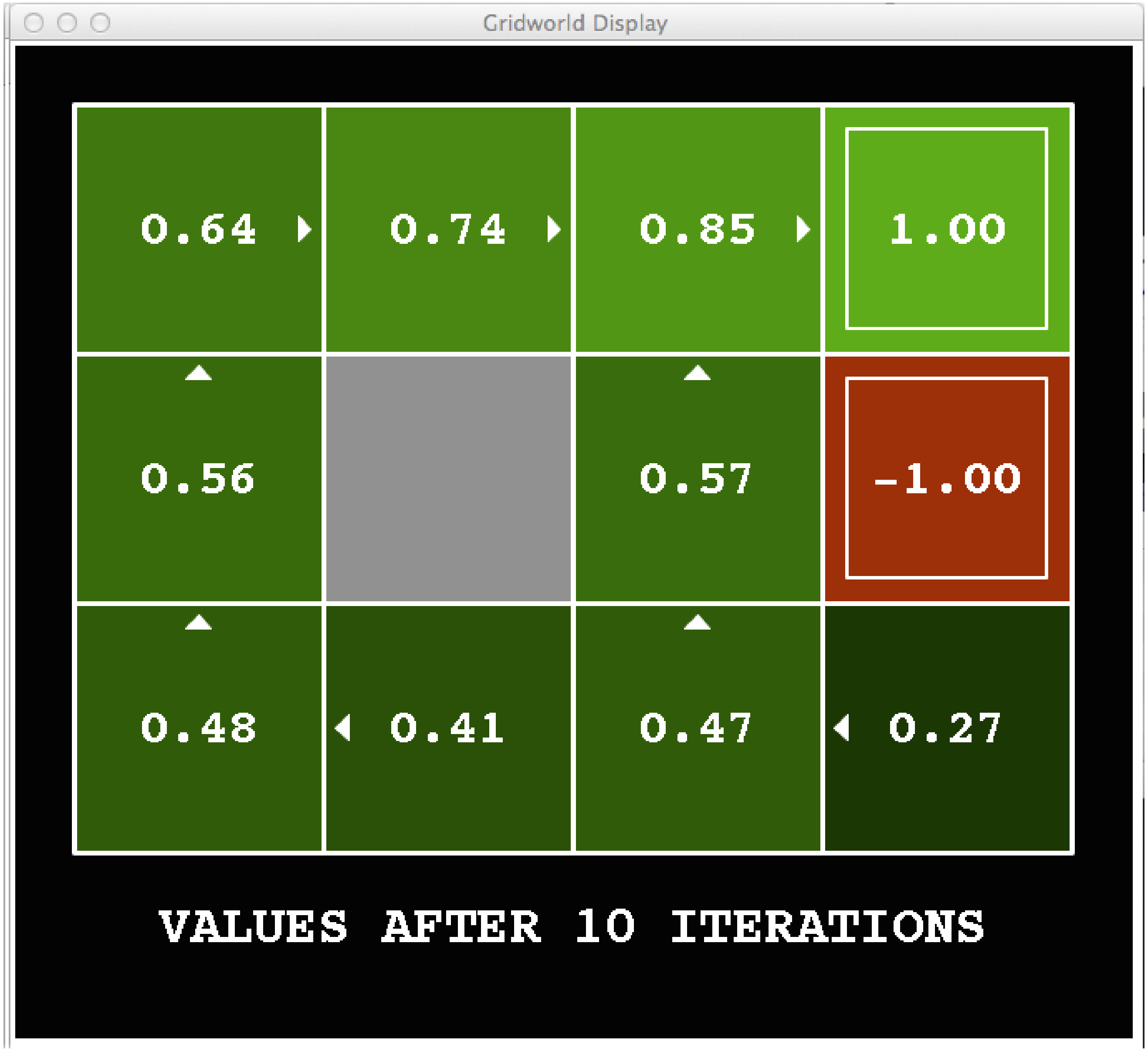
Slide: Berkeley CS188 course notes (downloaded Summer 2015)

Value iteration example



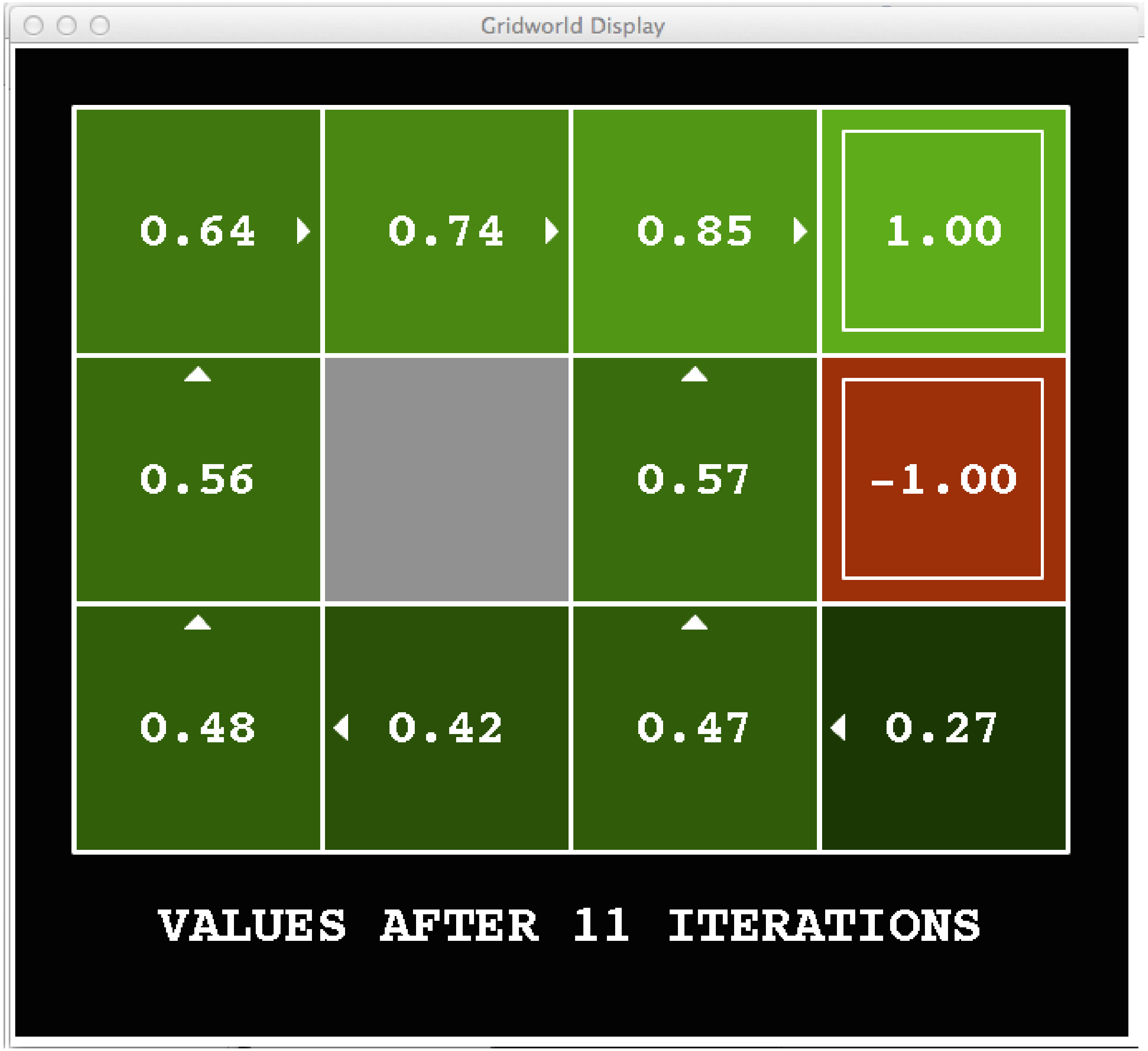
Slide: Berkeley CS188 course notes (downloaded Summer 2015)

Value iteration example



Slide: Berkeley CS188 course notes (downloaded Summer 2015)

Value iteration example

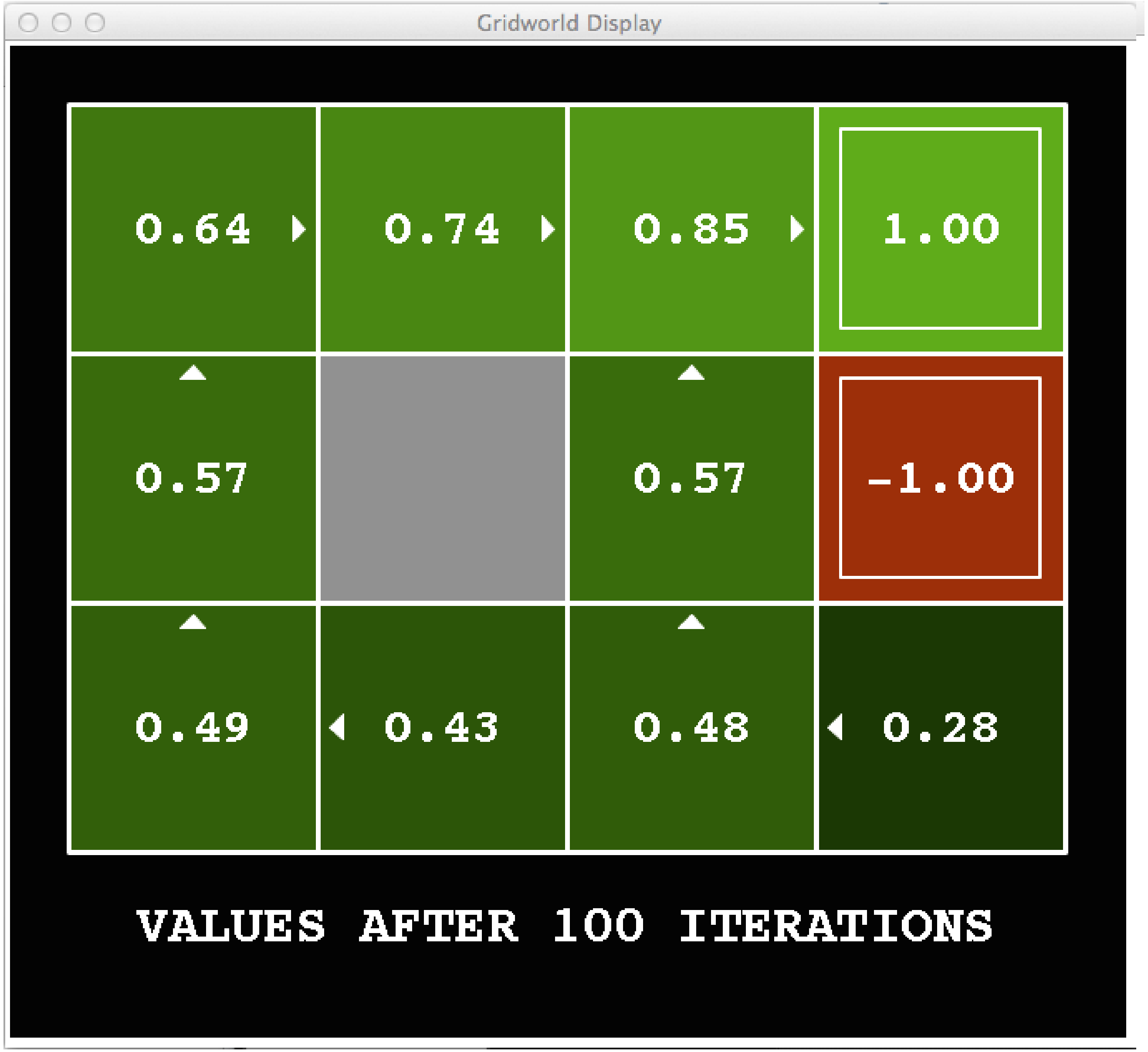


Slide: Berkeley CS188 course notes (downloaded Summer 2015)

Value iteration example

Slide: Berkeley CS188 course notes (downloaded Summer 2015)

Value iteration example



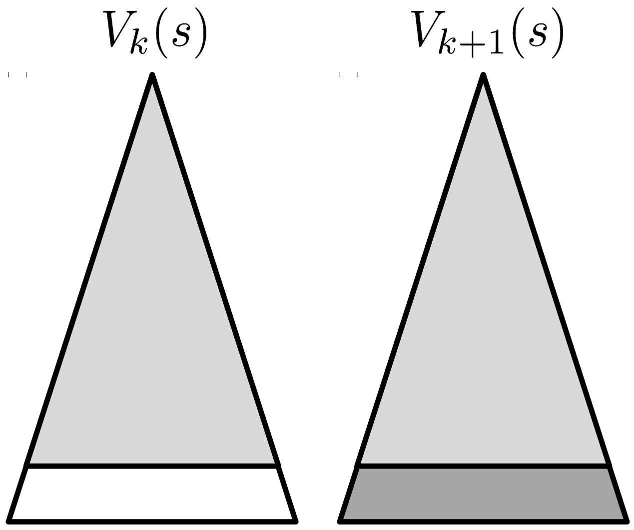
Slide: Berkeley CS188 course notes (downloaded Summer 2015)

Proof sketch: convergence of value iteration

- 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



Slide: Berkeley CS188 course notes (downloaded Summer 2015)

Bellman Equations and 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

Slide: Berkeley CS188 course notes (downloaded Summer 2015)

But, how do you compute a policy?

Suppose that we have run value iteration and now have a pretty good approximation of V* …

How do we compute the optimal policy?

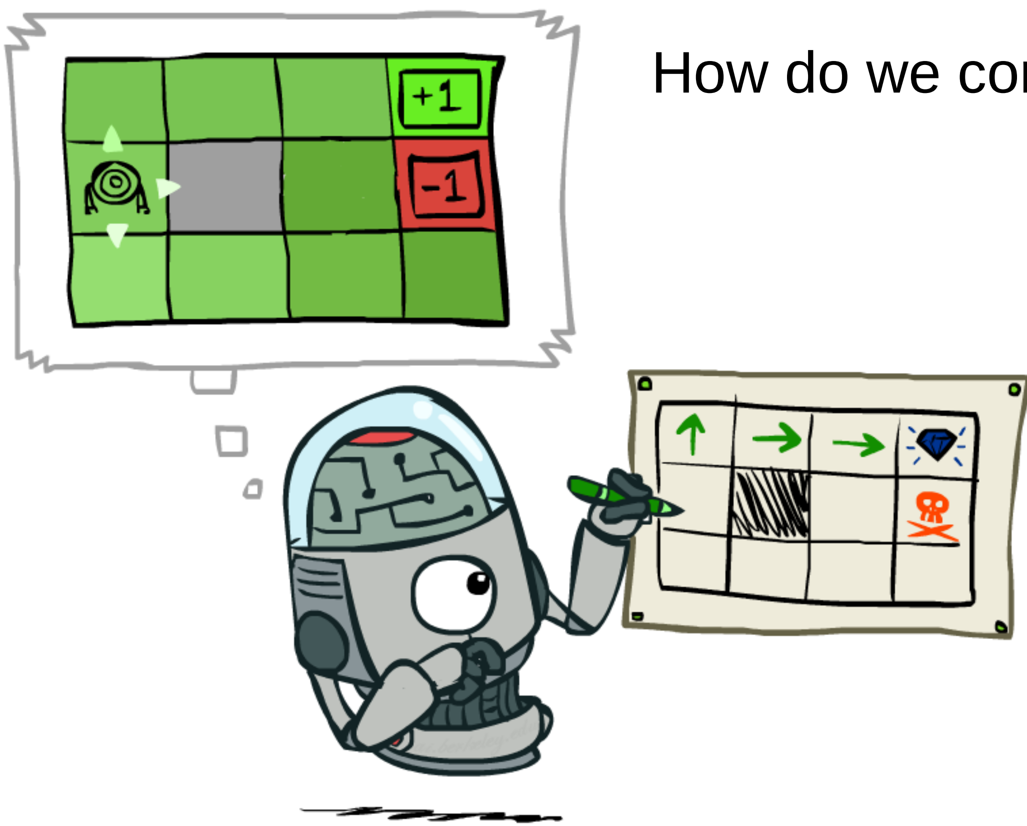
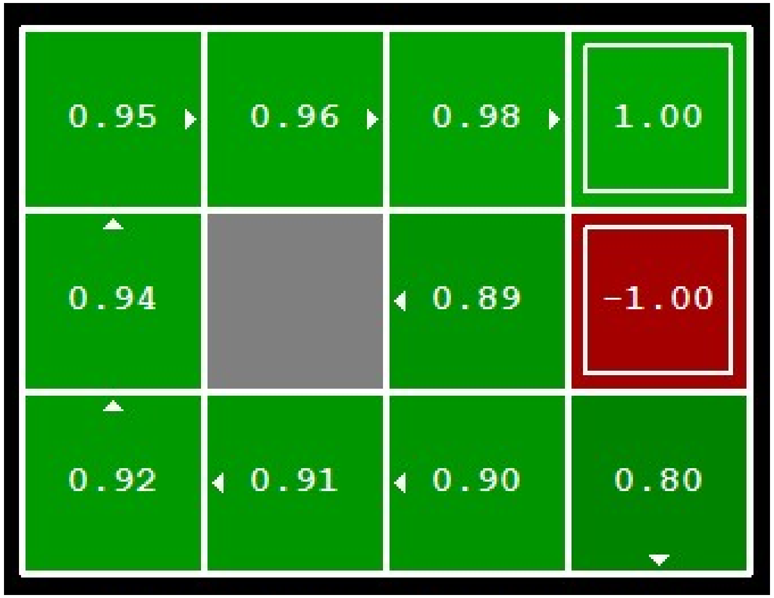


Image: Berkeley CS188 course notes (downloaded Summer 2015)

But, how do you compute a policy?



Given values calculated using value iteration, do one step of expectimax:

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

The optimal policy is implied by the optimal value function...

Image: Berkeley CS188 course notes (downloaded Summer 2015)