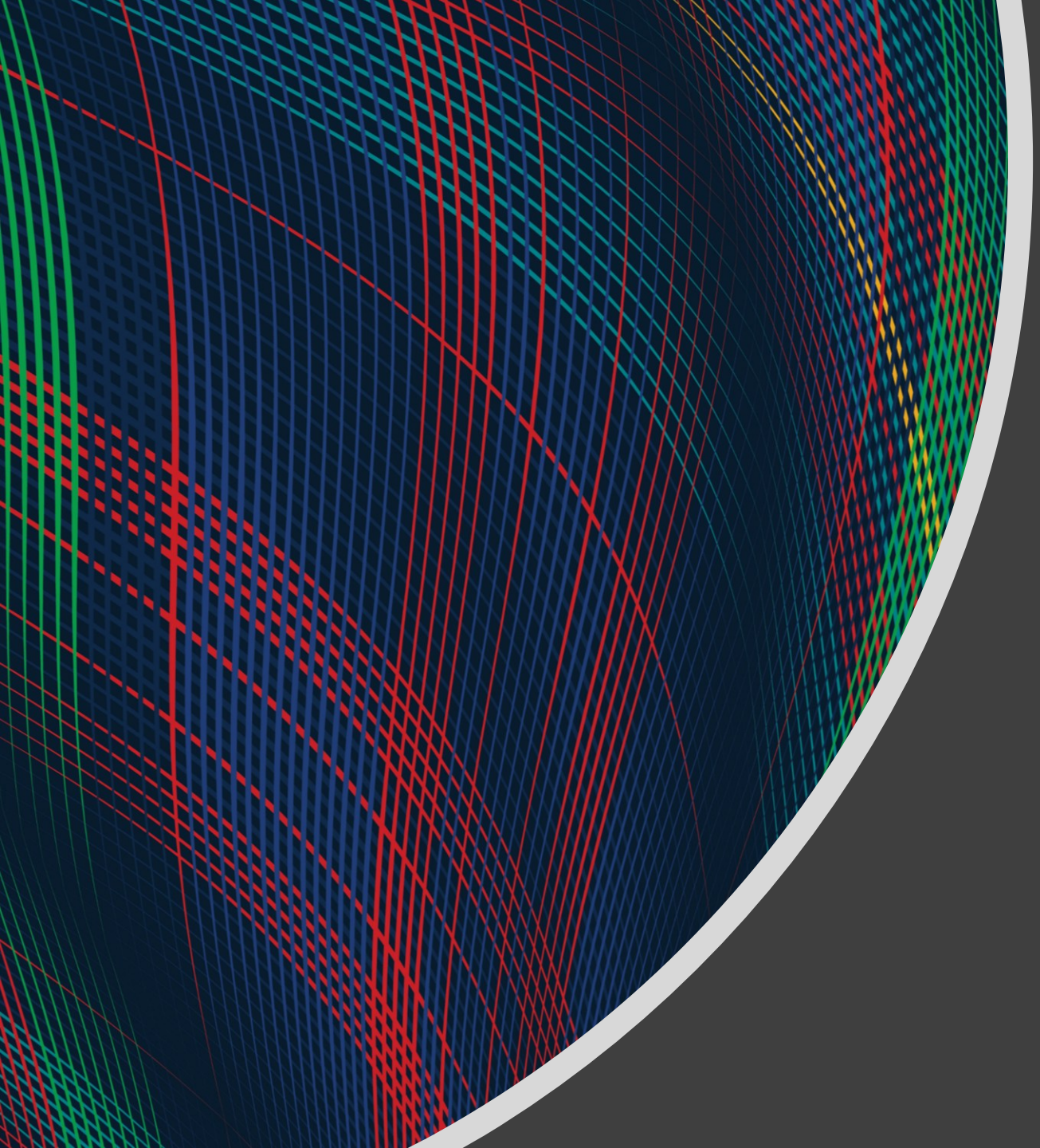


07-280
AI & ML I

Markov Decision Processes

Instructors:
Pat Virtue & Nihar Shah

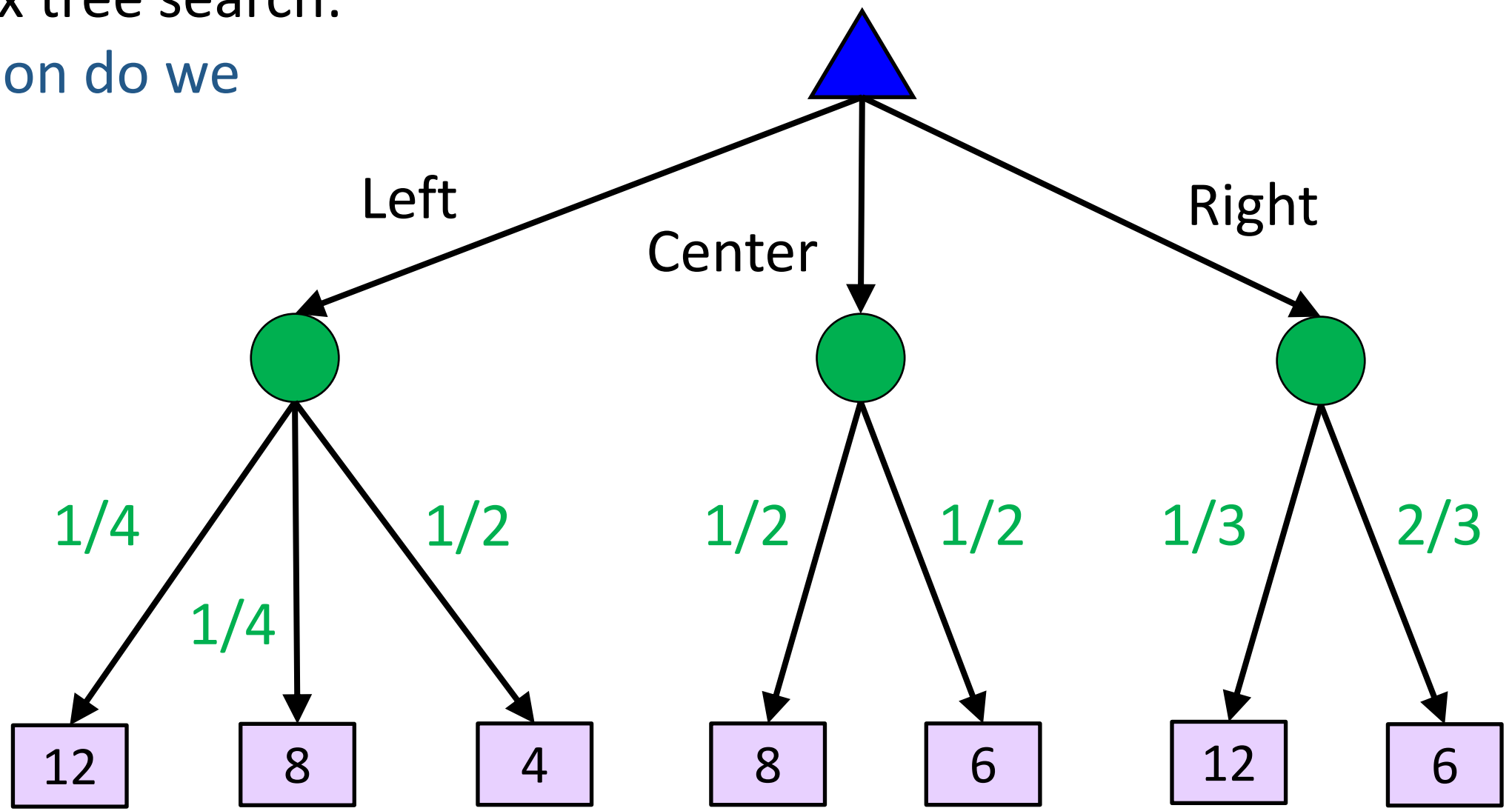
Pre-reading

Question

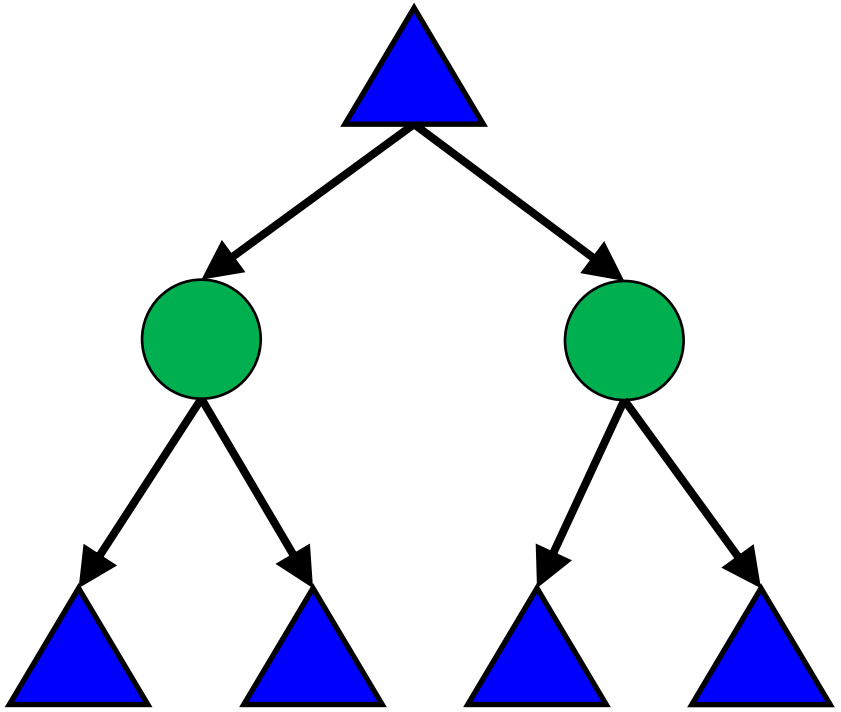
Expectimax tree search:

Which action do we choose?

A) Left
B) Center
C) Right
D) Eight



Expectimax Notation



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

Non-Deterministic Search

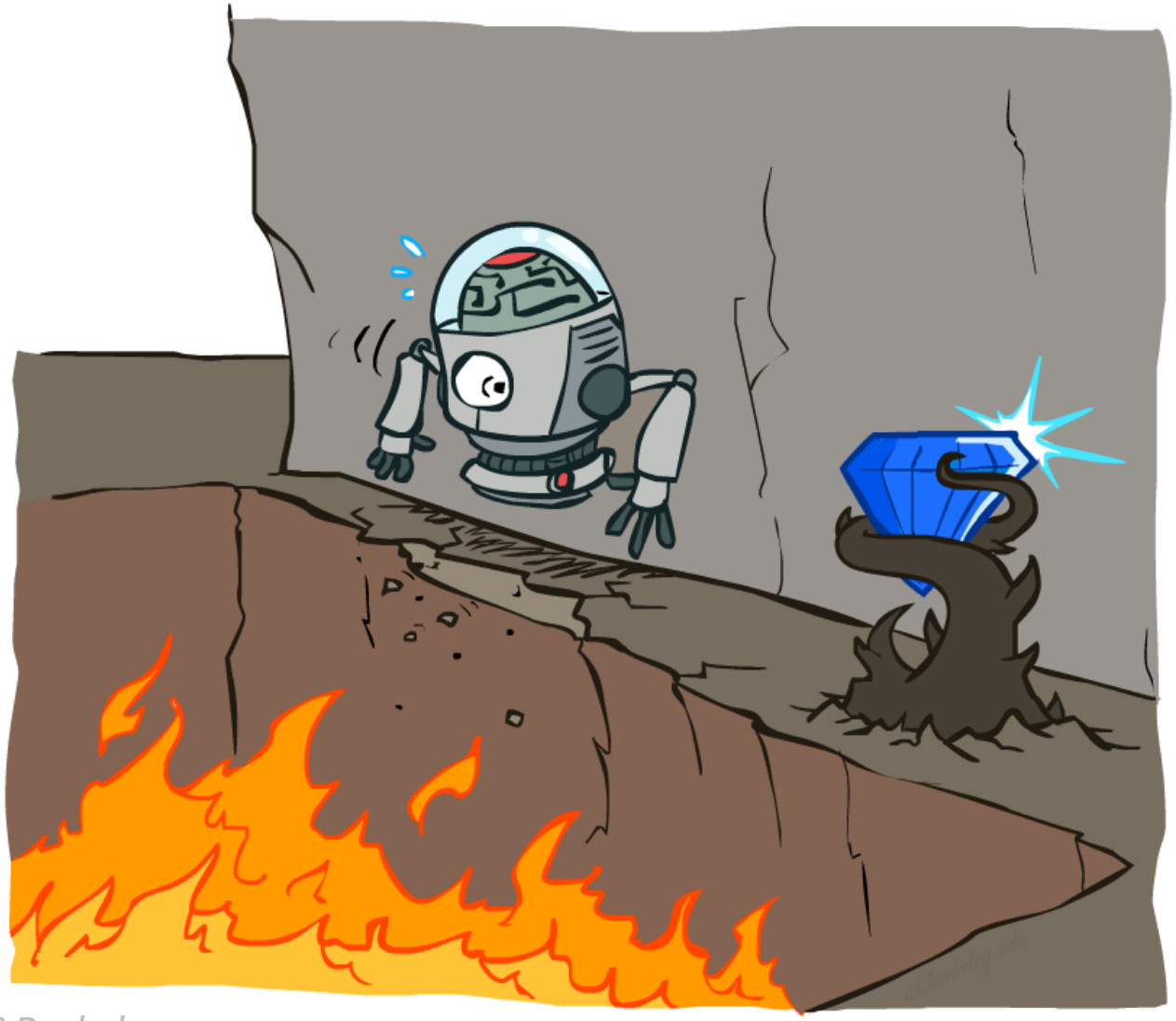
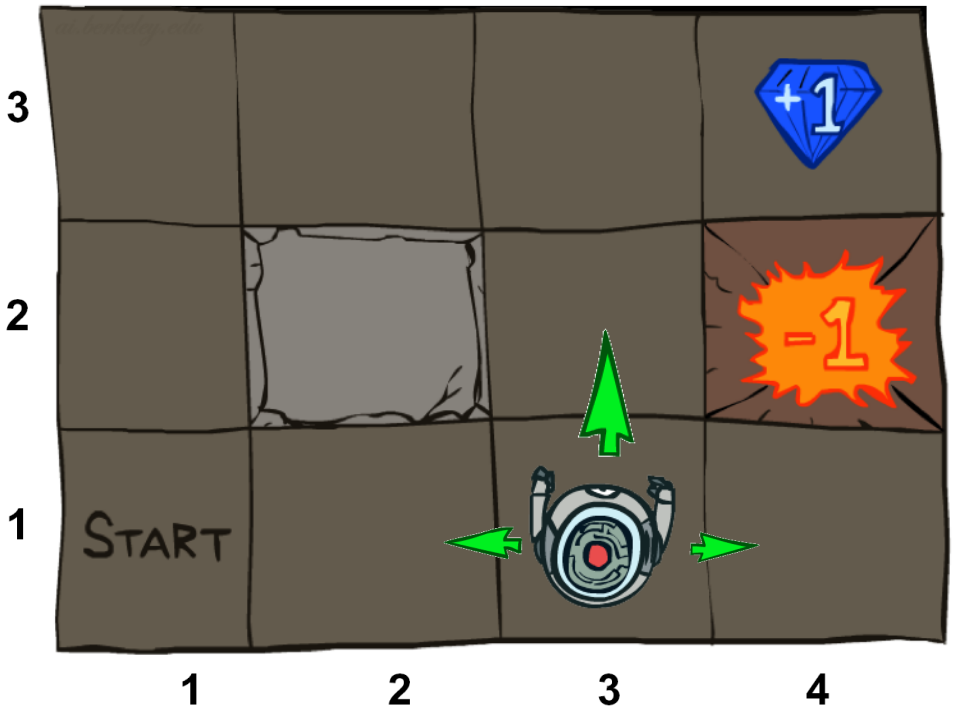


Image credit: *Ketrina Yim, UC Berkeley*

Example: Grid World

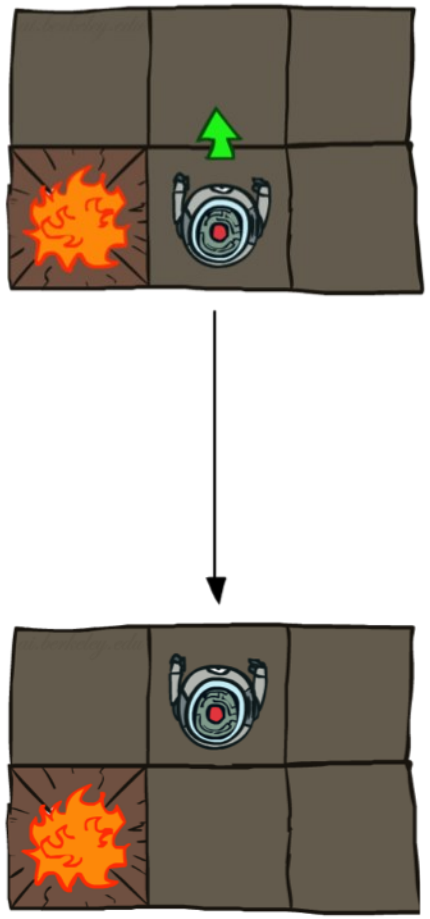
- A maze-like problem
  - The agent lives in a grid
  - Walls block the agent's path
- Noisy movement: actions do not always go as planned
  - 80% of the time, the action North takes the agent North (if there is no wall there)
  - 10% of the time, North takes the agent West; 10% East
  - If there is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



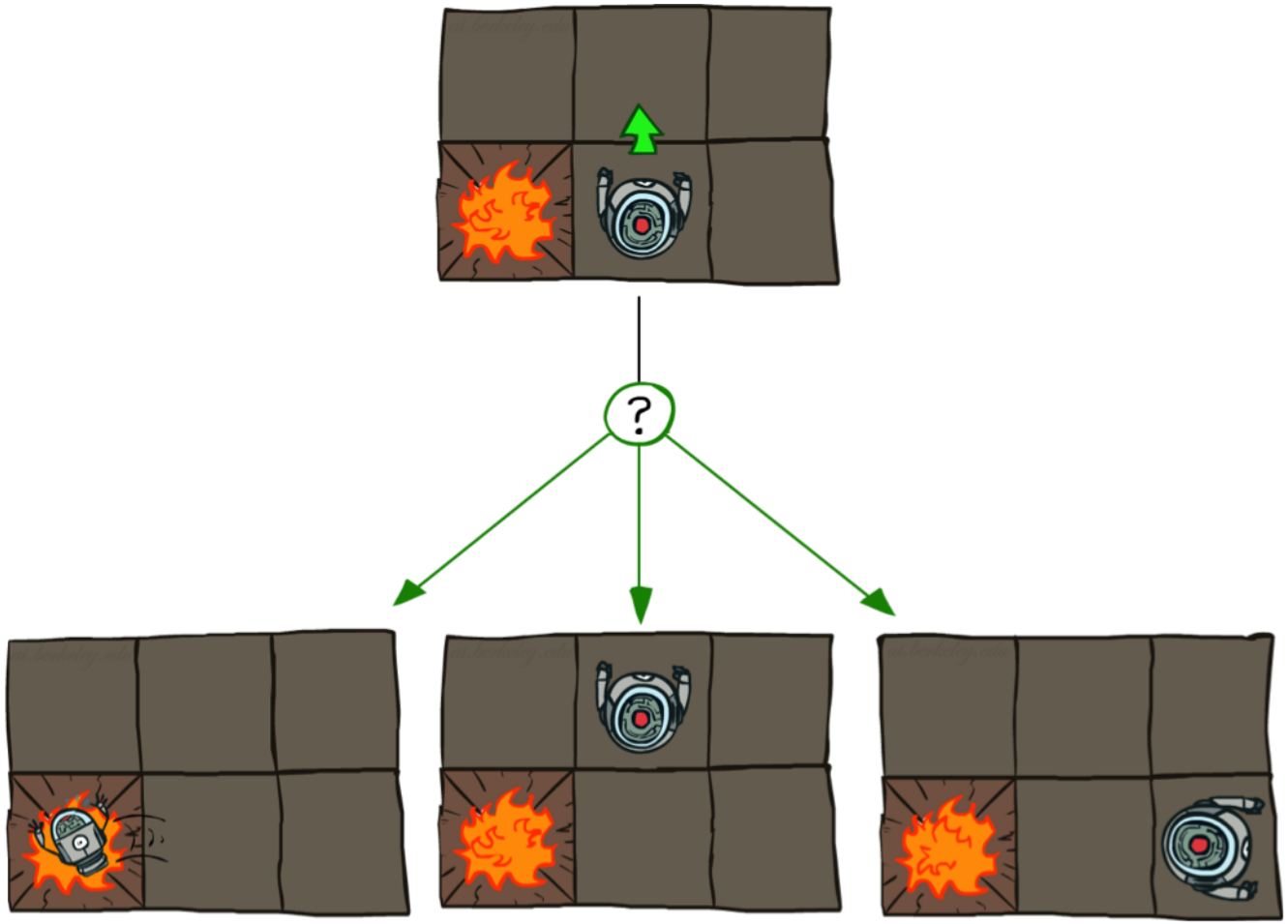
Slide credit: UC Berkeley AI

Grid World Actions

Deterministic Grid World



Stochastic Grid World

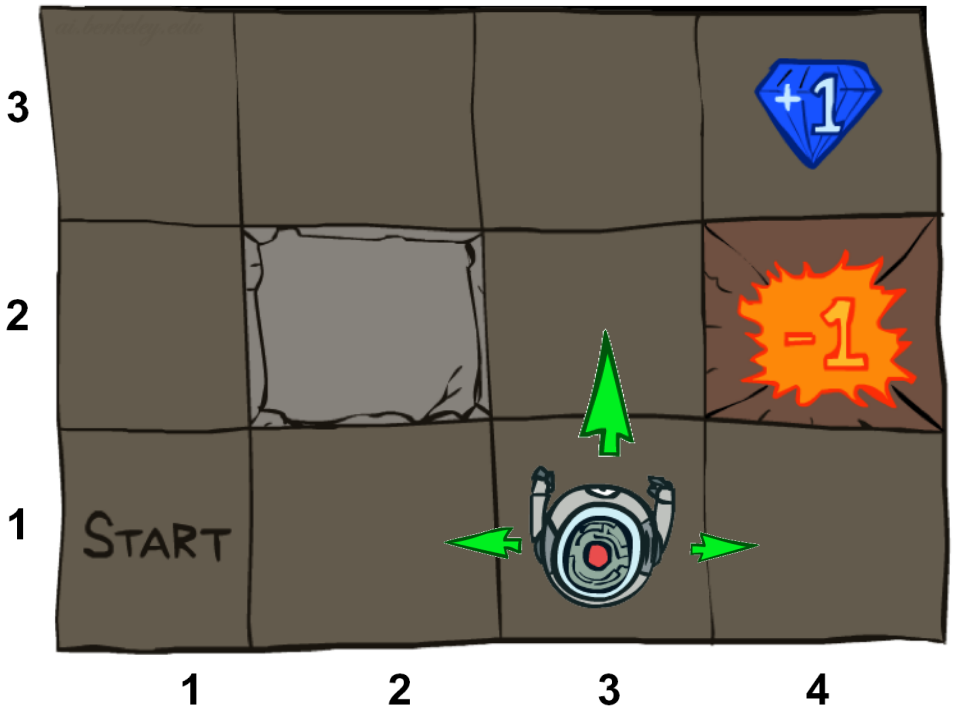


Slide credit: UC Berkeley AI

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 $P(s' | s, a)$
  - 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') (negative if it is a penalty)
- 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

Slide credit: UC Berkeley AI

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)

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

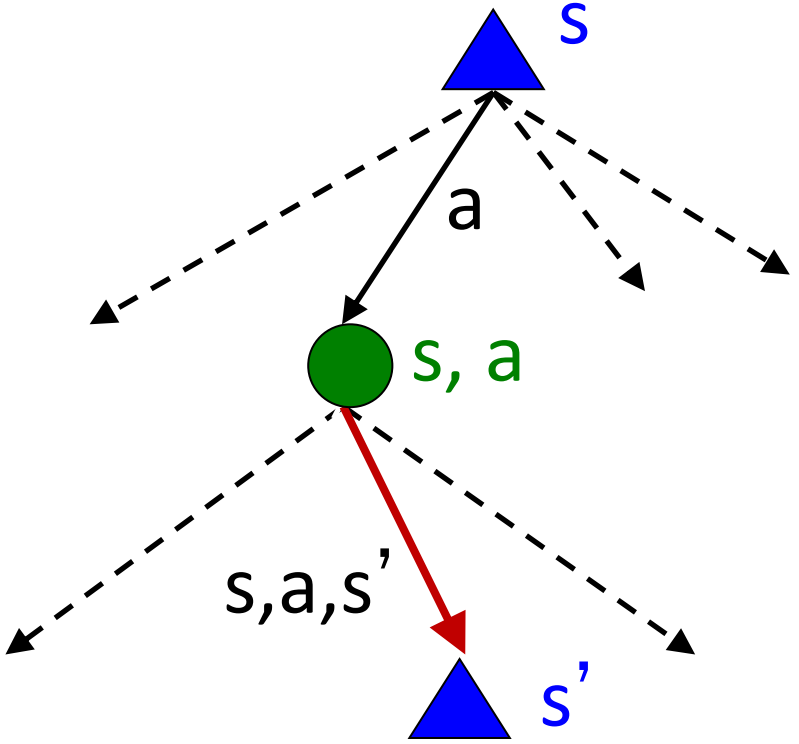
Slide credit: UC Berkeley AI

Recursive Expectimax (with rewards!)

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

With rewards:

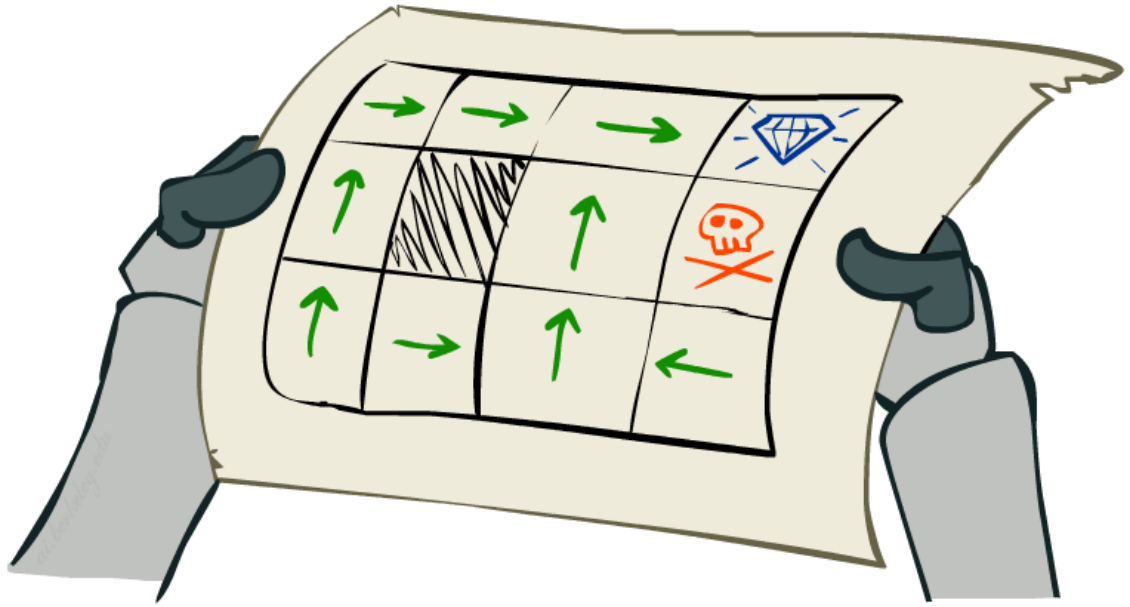
$$V(s) = \max_a \sum_{s'} P(s'|s,a)\, [R(s,a,s') + V(s')]$$



Policies

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



Example policy

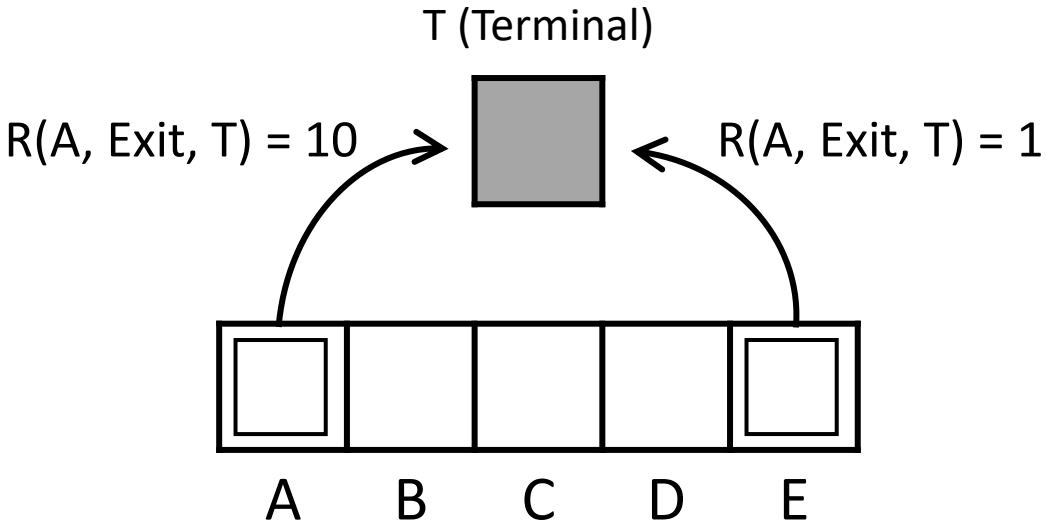
(a map of states to actions)

Slide credit: UC Berkeley AI

Simple Deterministic Example

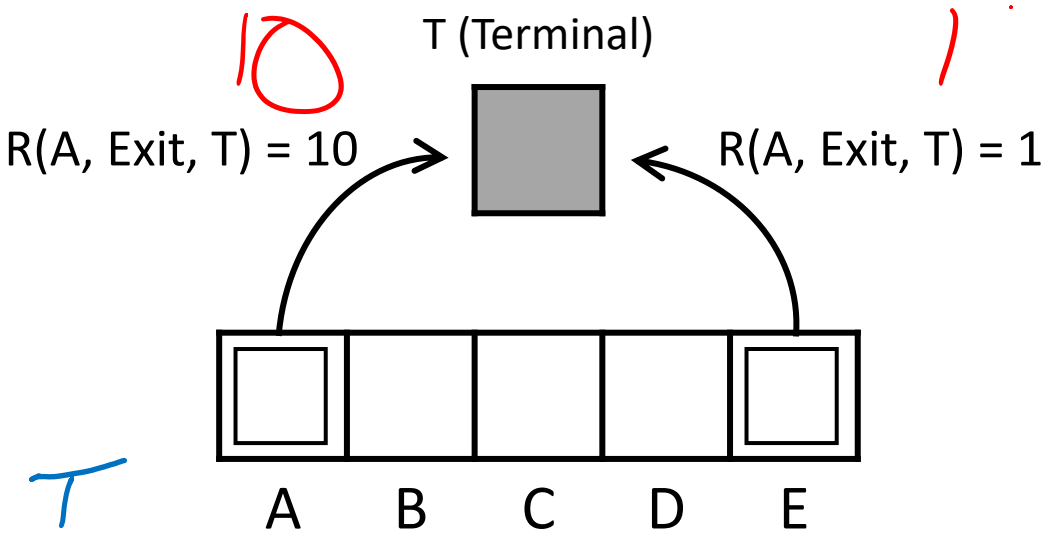
- Actions: B, C, D: East, West
- Actions: A, E: Exit
- Transitions: deterministic
- Rewards only for transitioning to terminal state

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

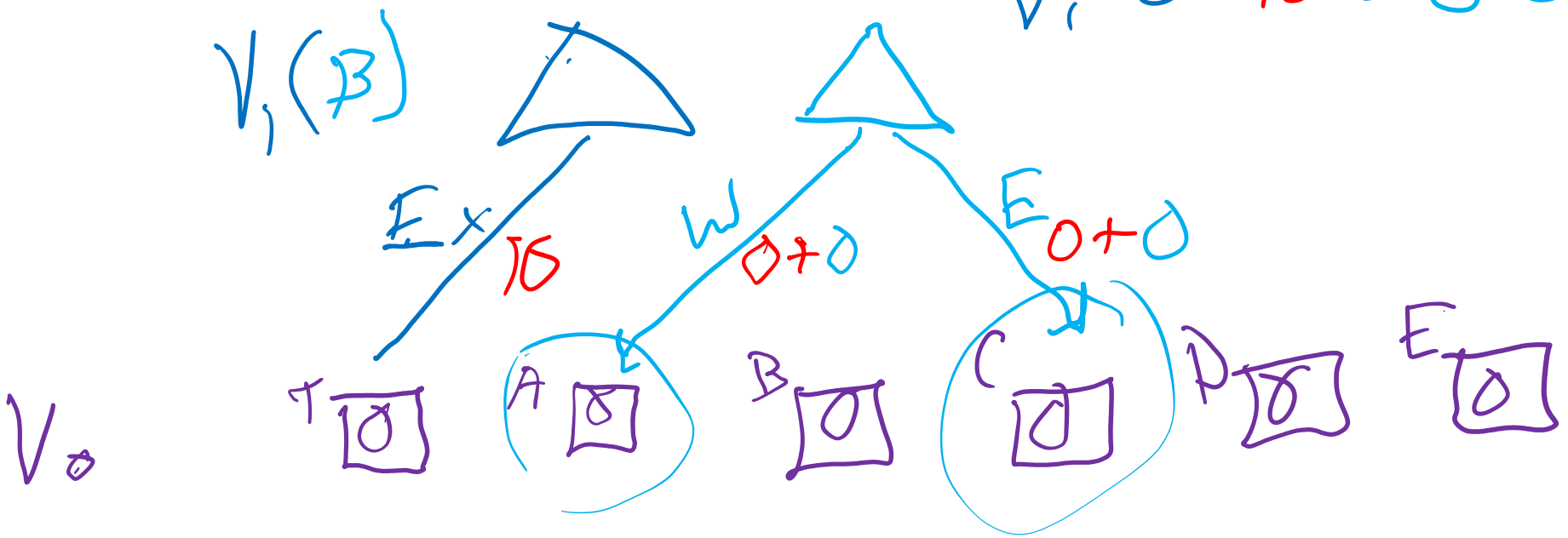


Simple Deterministic Example

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



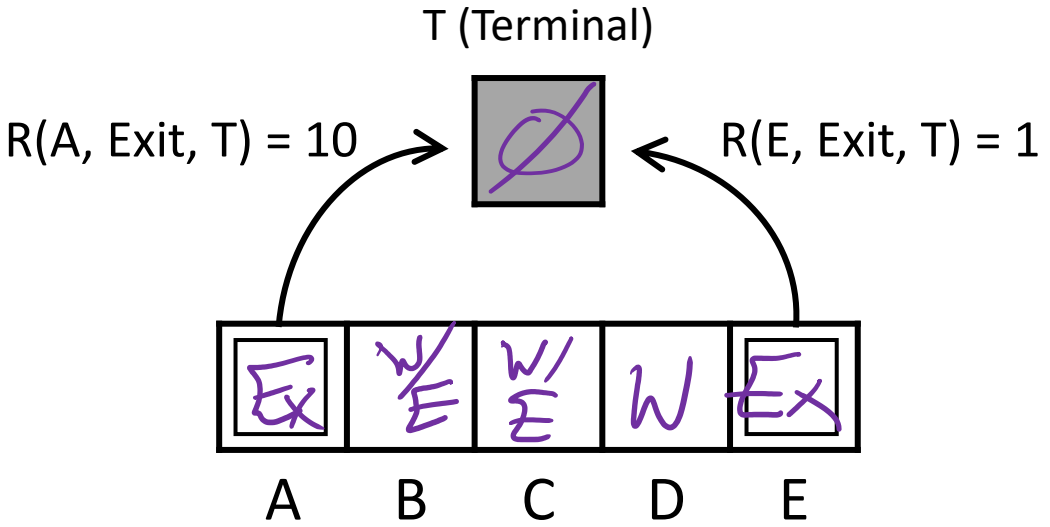
$$V_{k+1}(s) = \max_{a}[R(s,a,s') + V_k(s')]$$



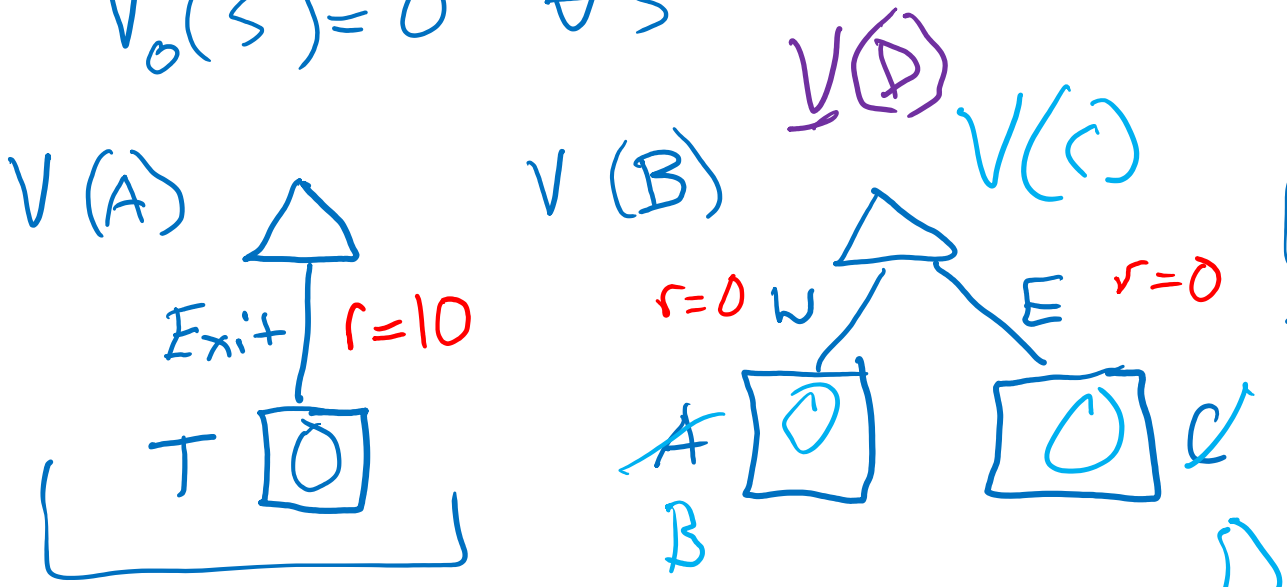
Simple Deterministic Example

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

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



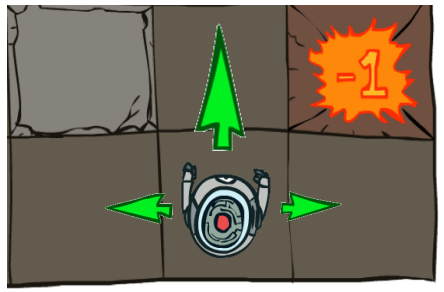
$V_0(s) = 0 \quad \forall s$



	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	10	10	0	1	1
V_3	0	10	10	10	1	1
V_4	0	10	10	10	10	1

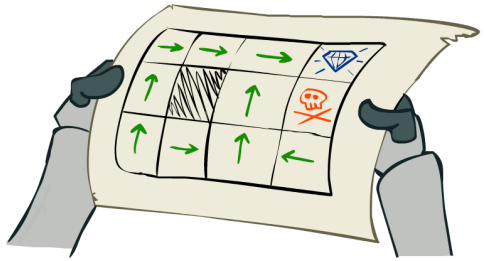
[End Pre-reading]

MDP Outline



Pre-reading: MDP Setup

- Expectimax: State, actions, non-deterministic transition functions
  - Example: GridWorld
- Policies: Mapping states → actions
- Rewards
- Simple Value Iteration



Discounting, γ

Solving MDPs

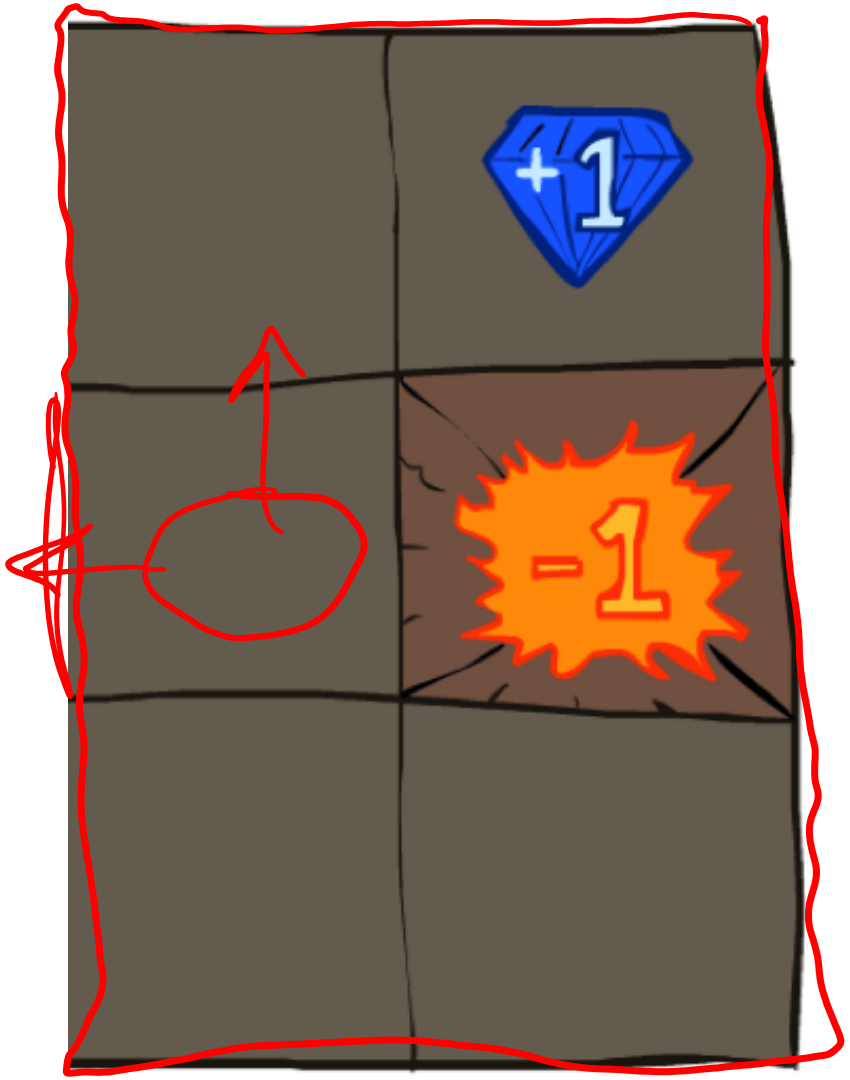
- Value iteration
- Policy iteration

Image credit: *Ketrina Yim, UC Berkeley*

Question

Which is the best move?

A. North
B. East
C. South
D. West



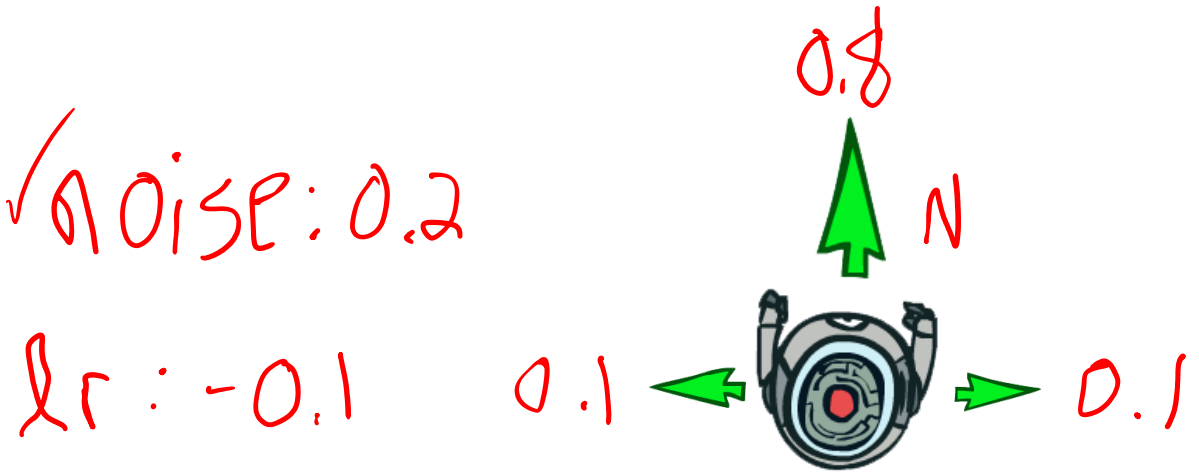
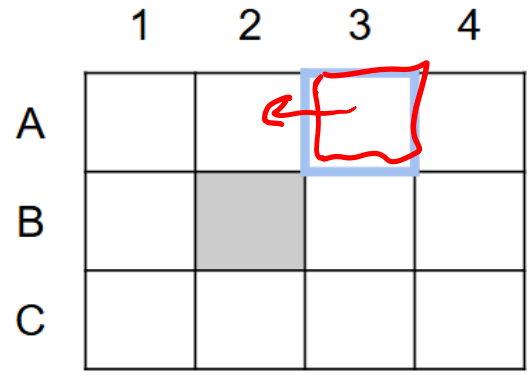


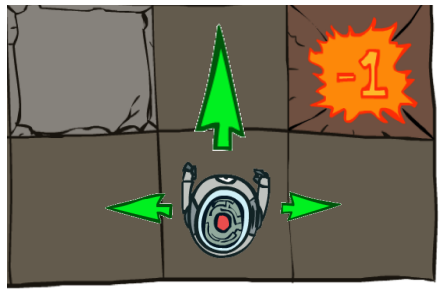
Image credit: *Ketrina Yim, UC Berkeley*

Grid World: Expectimax tree



For starting state s=A3, fill in actions, probabilities, and next states

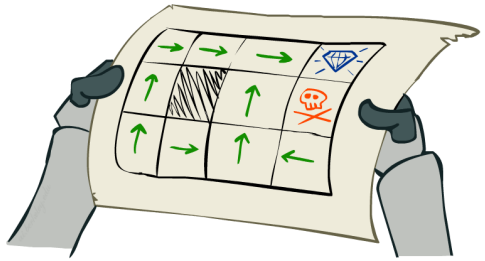
MDP Outline



Pre-reading: MDP Setup

- Expectimax: State, actions, non-deterministic transition functions
  - Example: GridWorld
- Policies: Mapping states → actions
- Rewards
- Simple Value Iteration

Discounting, γ

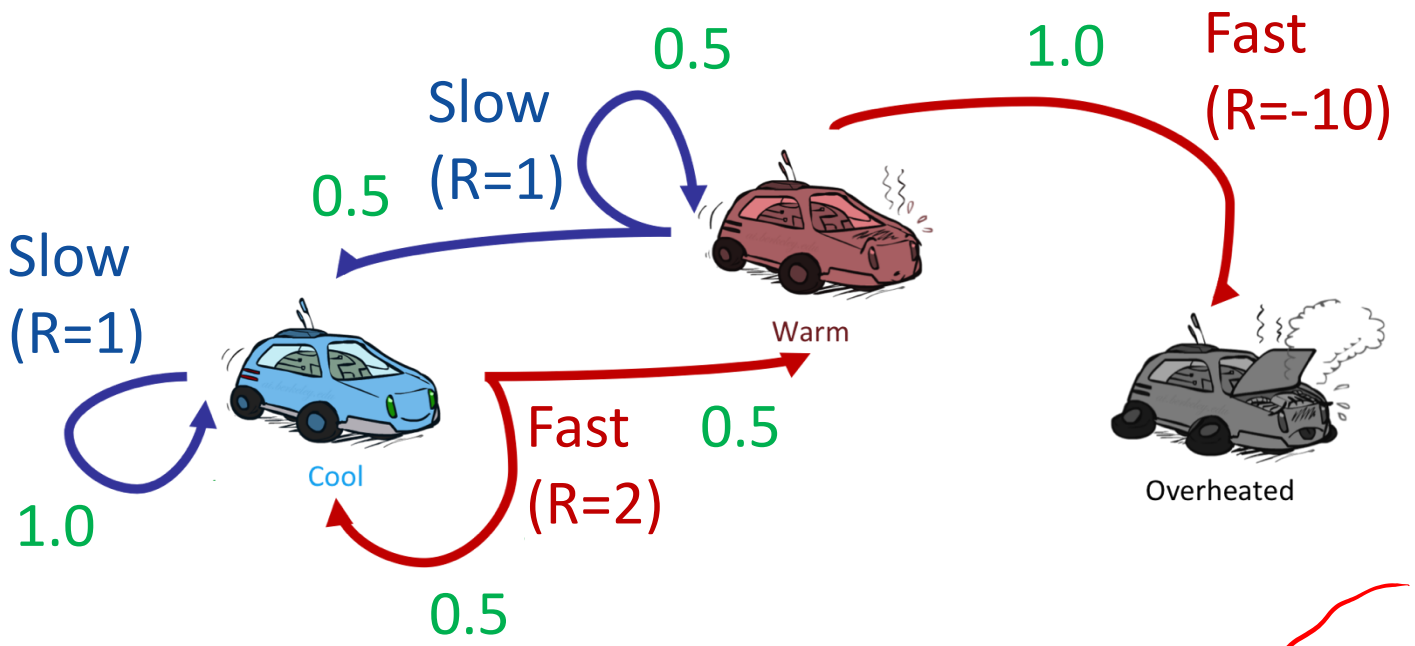


Solving MDPs

- Value iteration
- Policy iteration

Image credit: *Ketrina Yim, UC Berkeley*

Example MDP: Racing

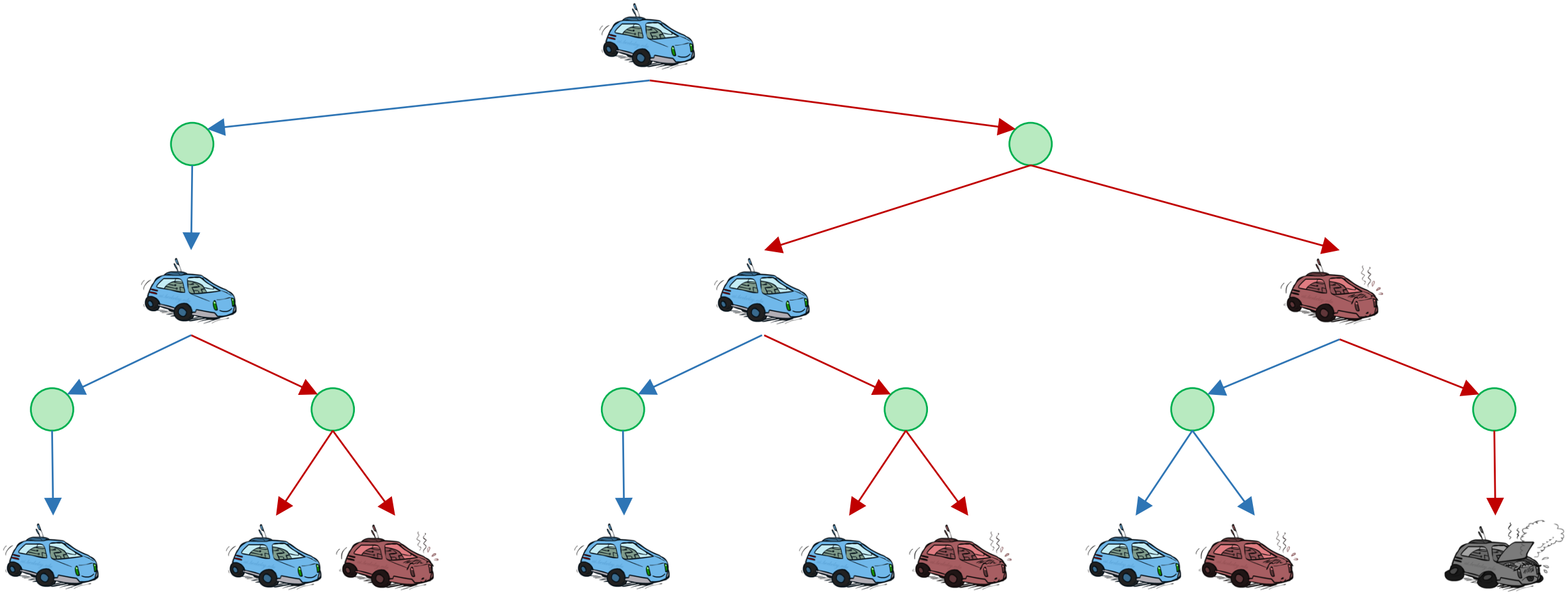


Assume no discount!

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

Slide credit: UC Berkeley AI

Racing Search Tree



Slide credit: UC Berkeley AI

Racing Search Tree

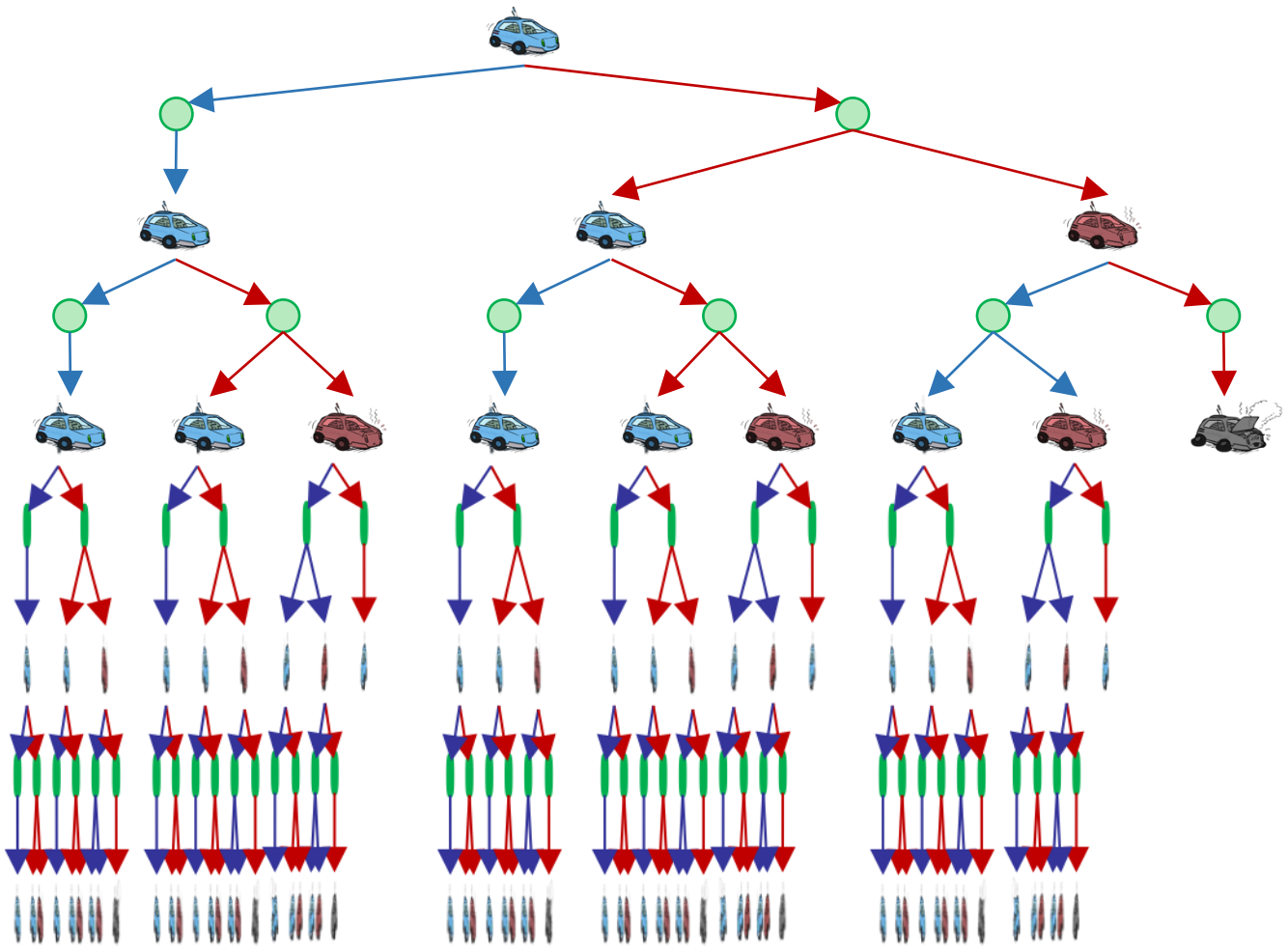
We're doing way too much work with expectimax!

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 $\gamma < 1$



Slide credit: UC Berkeley AI

Utilities of Sequences

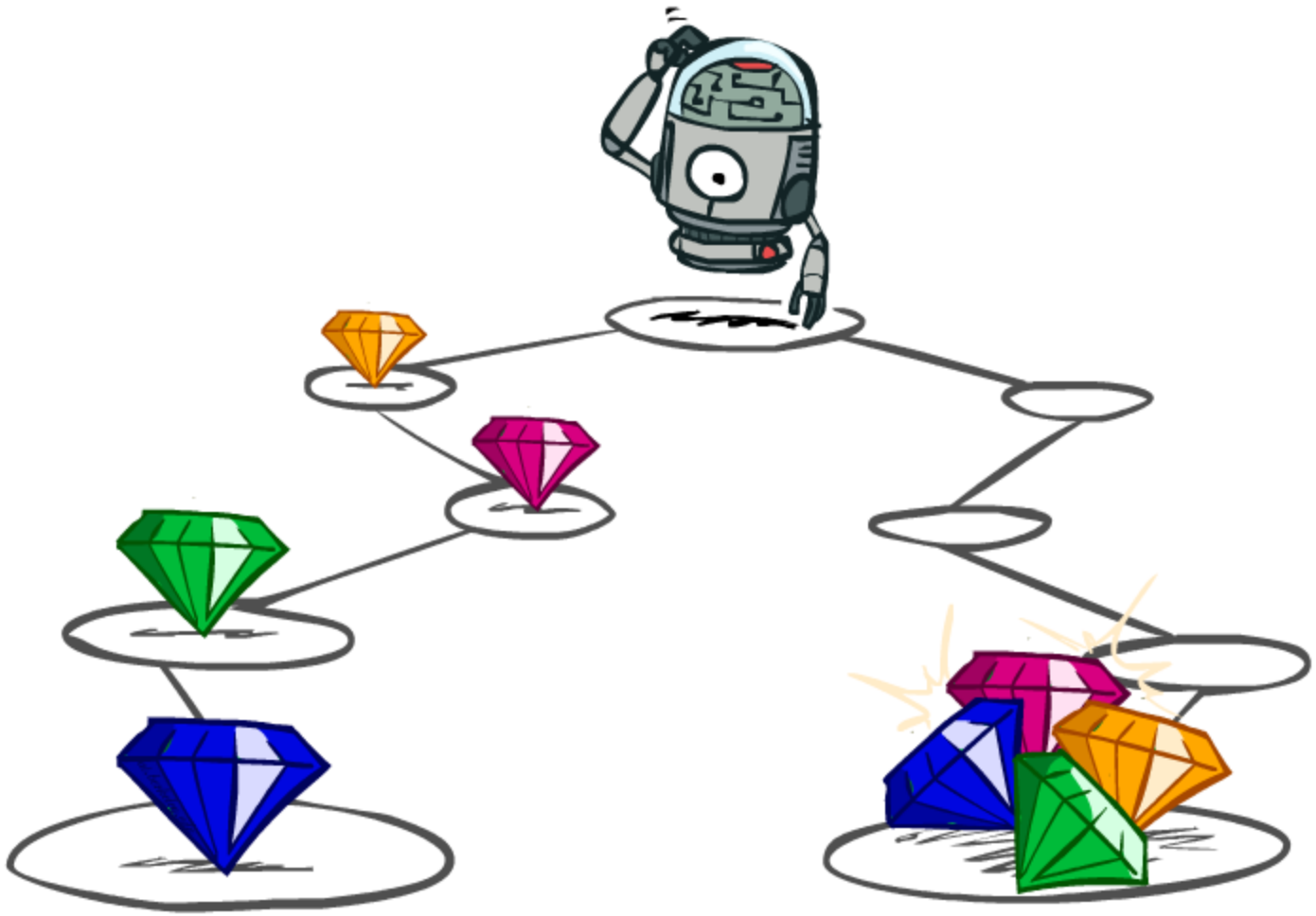


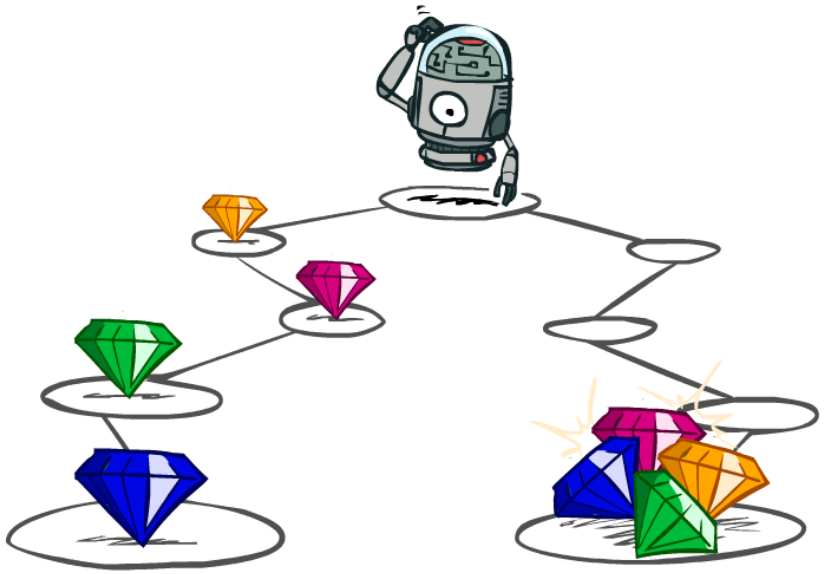
Image credit: *Ketrina Yim, UC Berkeley*

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]



Slide credit: UC Berkeley AI

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

100

$\gamma = 0.9$

$\gamma^0 = 1$

Worth Now

90

$\gamma = 0.9$

Worth Next Step

81

$\gamma^2 = 0.81$

Worth In Two Steps

Slide credit: UC Berkeley AI

Discounting

$10 + \gamma 3 + \gamma^2 7$

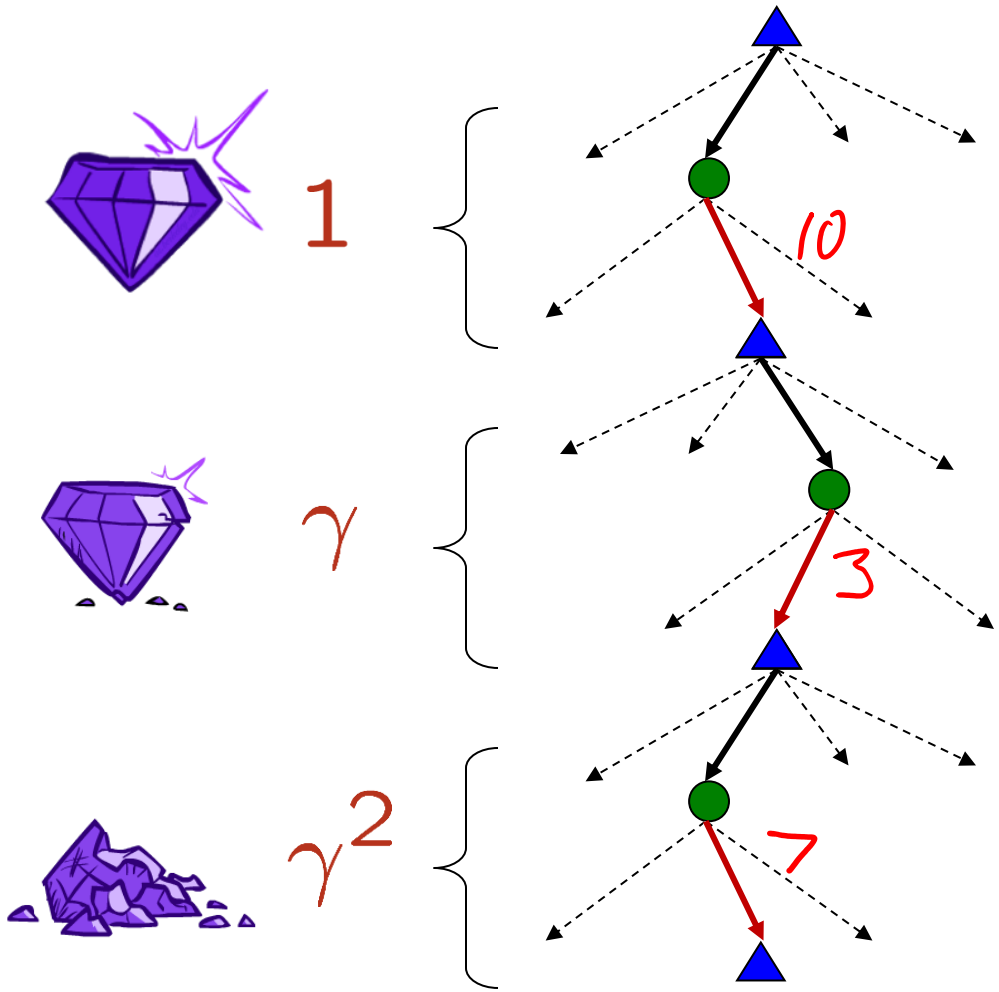
$= 10 + \gamma[3 + \gamma[7]]$

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 $0 < \gamma < 1$



Recursive Expectimax

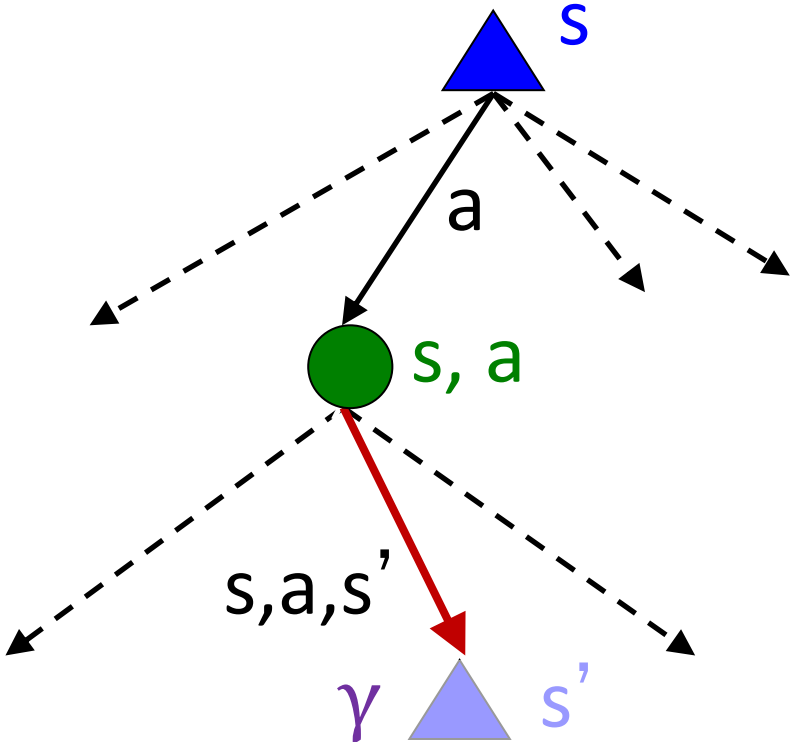
$$V(s) = \max_a \sum_{s'} P(s'|s,a)\, V(s')$$

With rewards:

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

With discount factor:

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



Recursive Expectimax

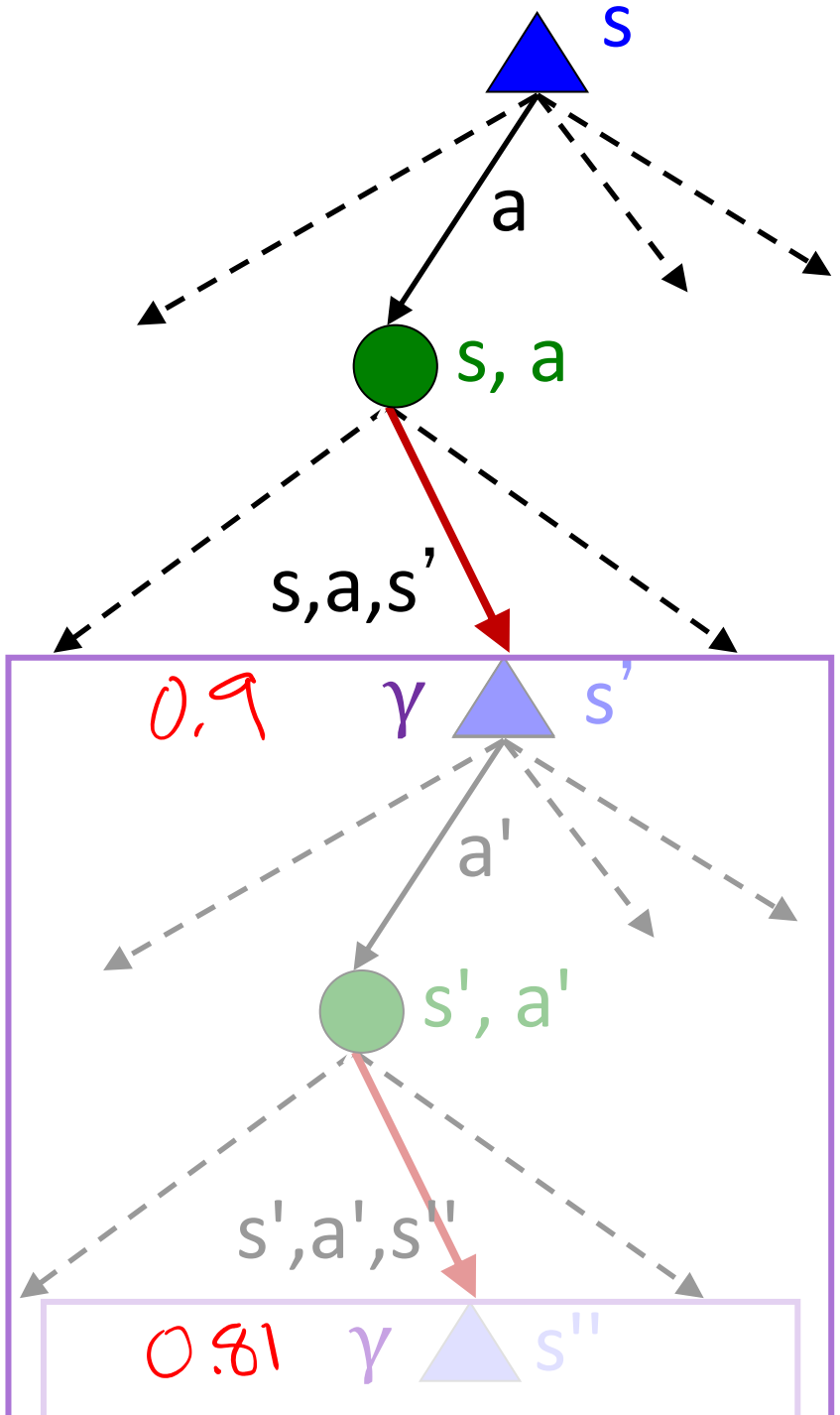
$$V(s) = \max_a \sum_{s'} P(s'|s,a)\, V(s')$$

With rewards:

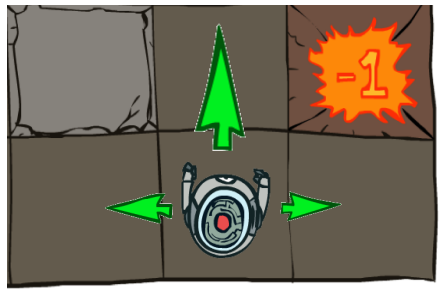
$$V(s) = \max_a \sum_{s'} P(s'|s,a)\, [R(s,a,s') + V(s')]$$

With discount factor:

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

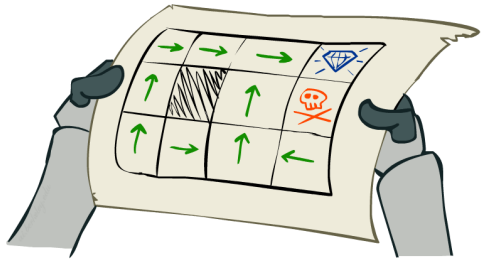


MDP Outline



MDP Setup

- Expectimax: State, actions, non-deterministic transition functions
  - Example: GridWorld
- Policies: Mapping states → actions
- Rewards
- Discounting, γ



Solving MDPs

- Value iteration
- Policy iteration

Image credit: *Ketrina Yim, UC Berkeley*

Solving MDPs

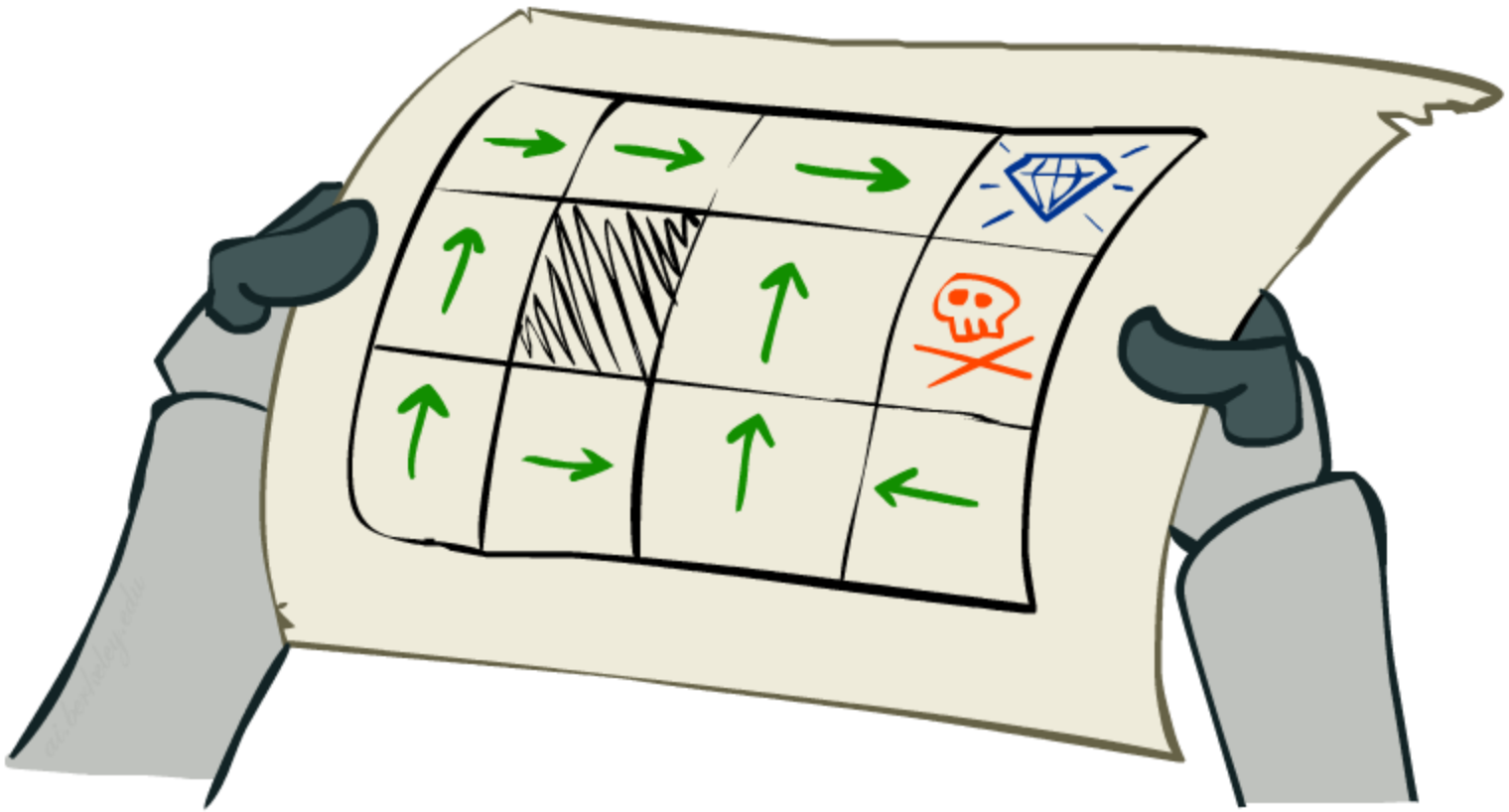
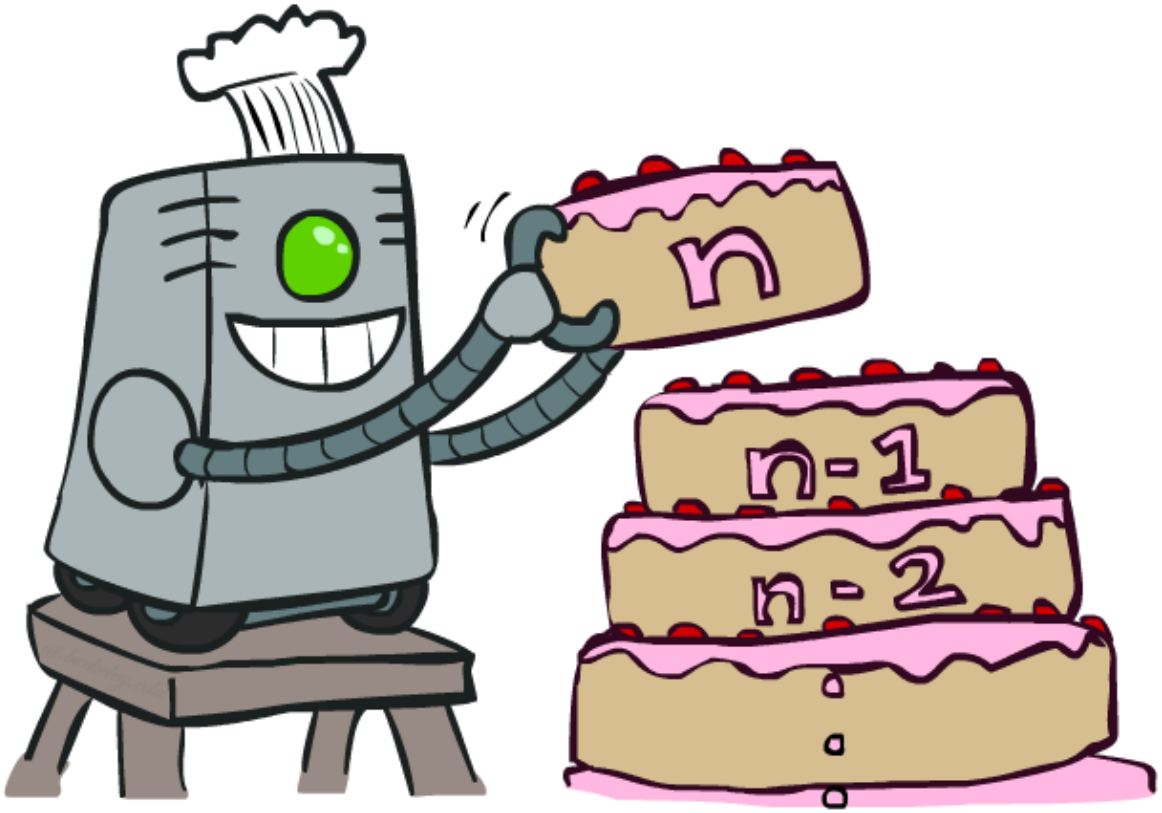


Image credit: *Ketrina Yim, UC Berkeley*



Value Iteration

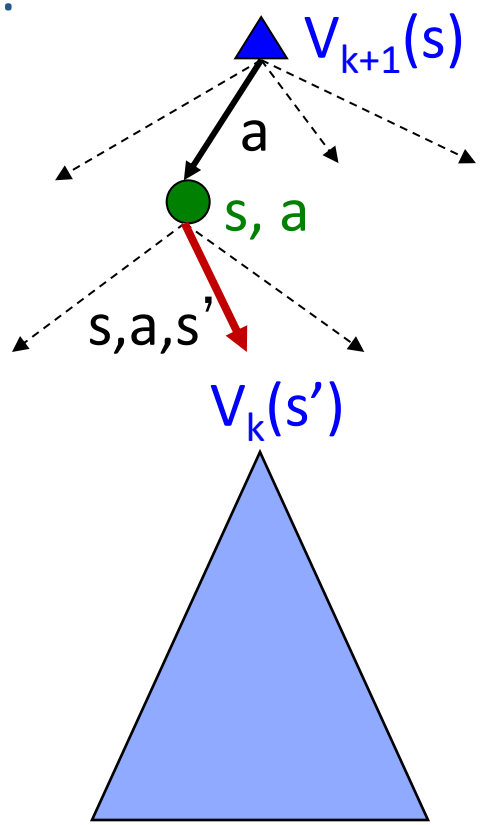
Image credit: *Ketrina Yim, UC Berkeley*

Value Iteration

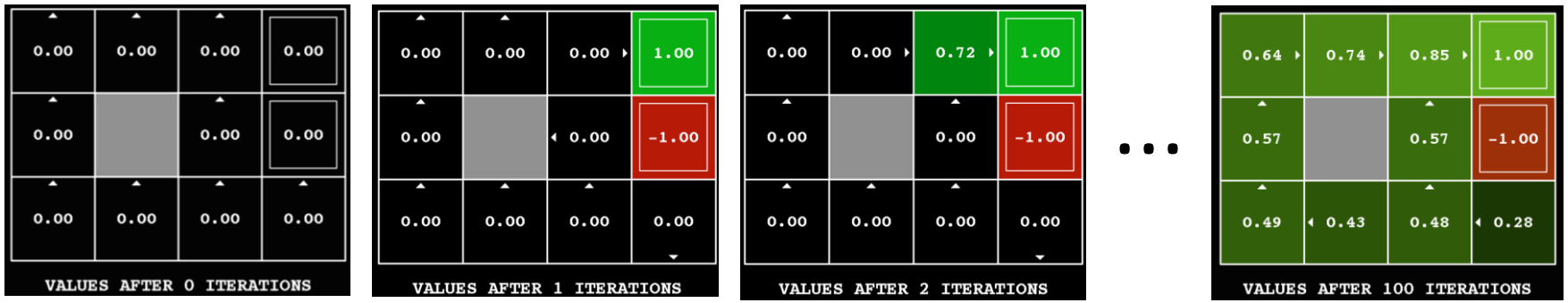
Start with $V_0(s) = 0$: no time steps left means an expected reward sum of zero

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



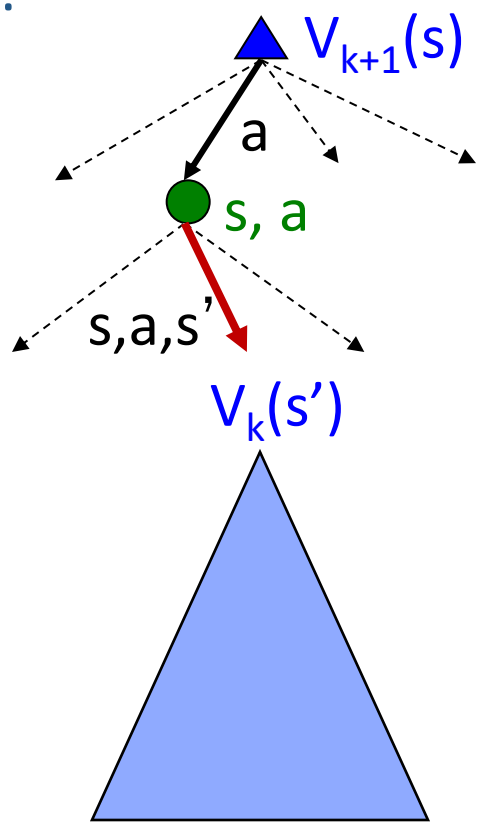
Slide credit: UC Berkeley AI

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

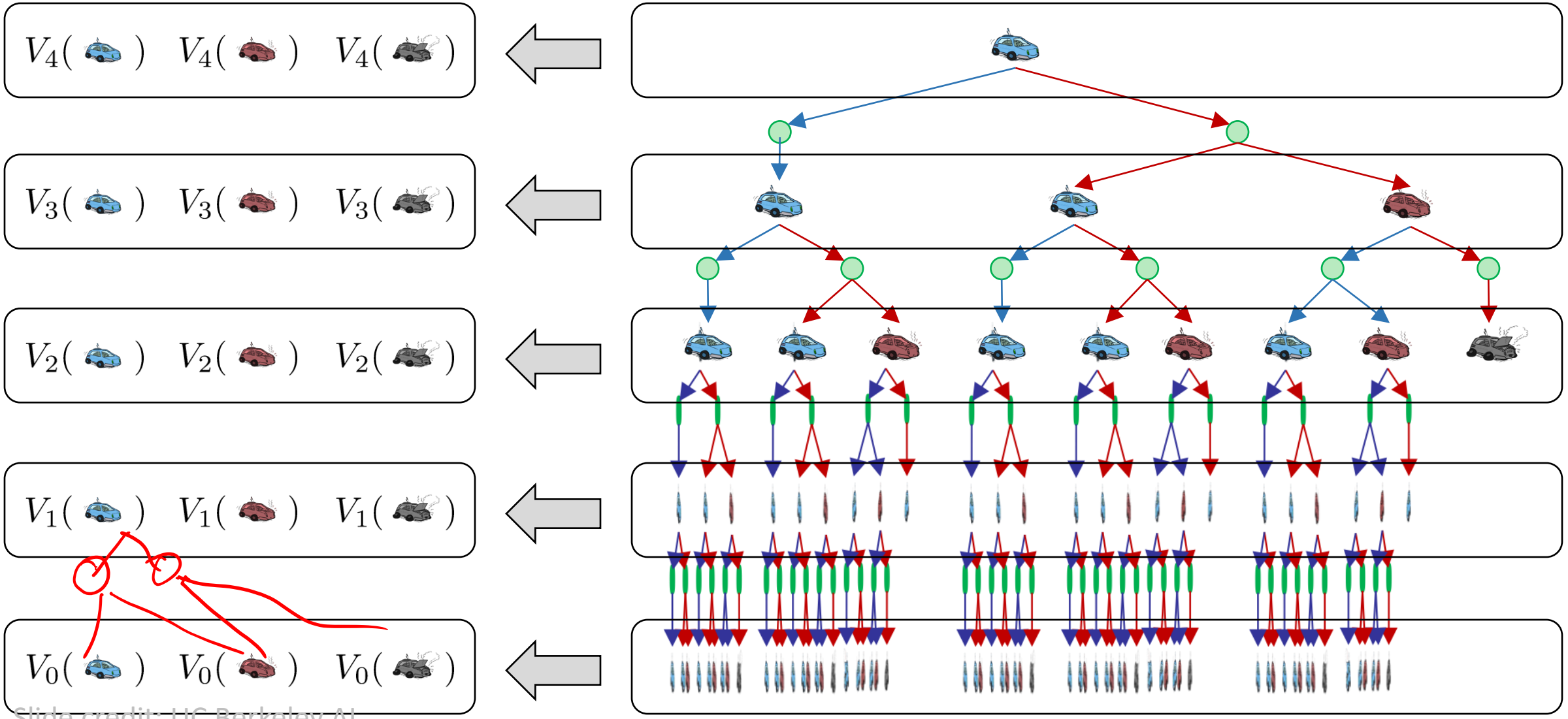
Theorem: will converge to unique optimal values

- Basic idea: approximations get refined towards optimal values

$\gamma < 1$

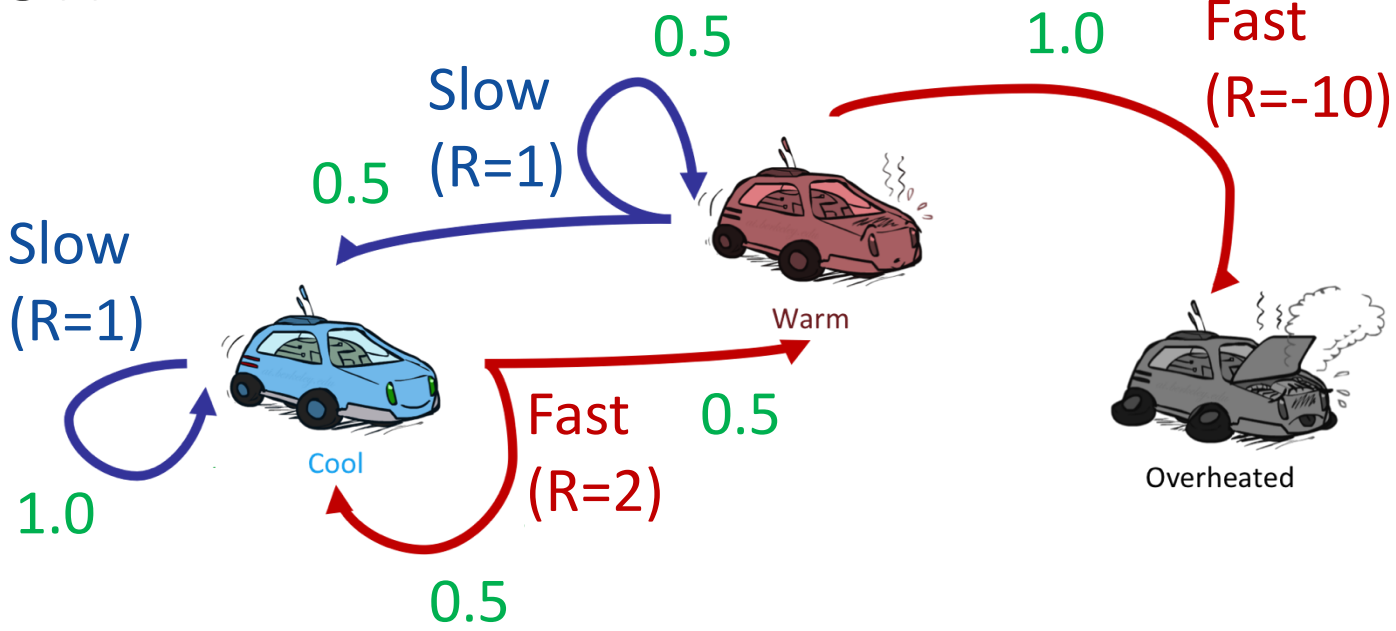
Slide credit: UC Berkeley AI

Computing Time-Limited Values



Example: Value Iteration

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



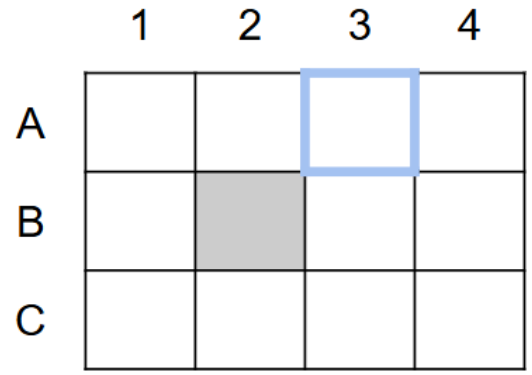
Assume no discount!

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

Slide credit: UC Berkeley AI

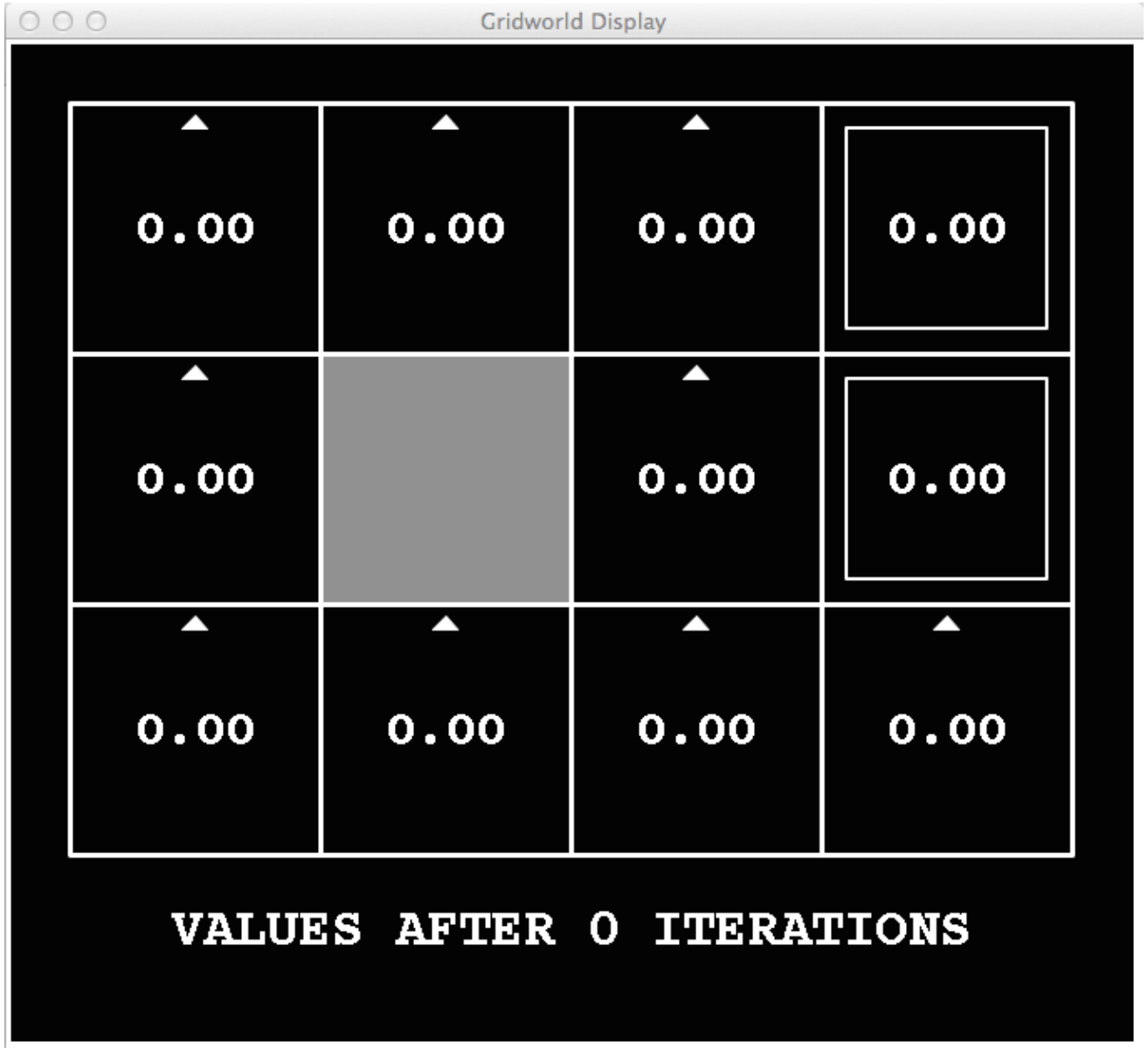
GridWorld Value Iteration

Grid World: Expectimax tree



For starting state s=A3, fill in actions, probabilities, and next states

k=0



0.1 0.8 0.1

noise: 0.2

γ: 0.9

lr: 0

Noise = 0.2
Discount = 0.9
Living reward = 0

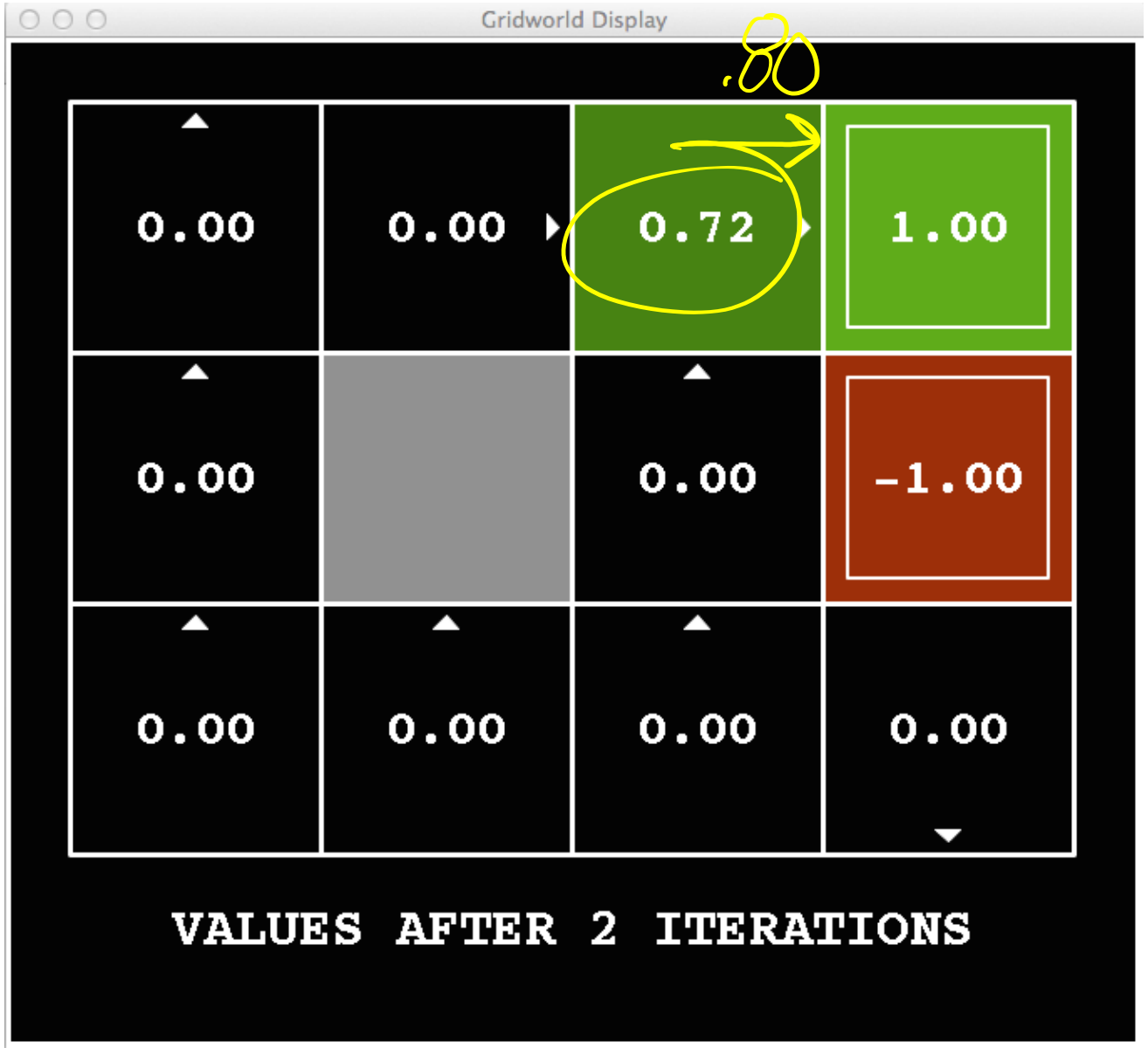
Slide credit: UC Berkeley AI

k=1

Noise = 0.2
Discount = 0.9
Living reward = 0

Slide credit: UC Berkeley AI

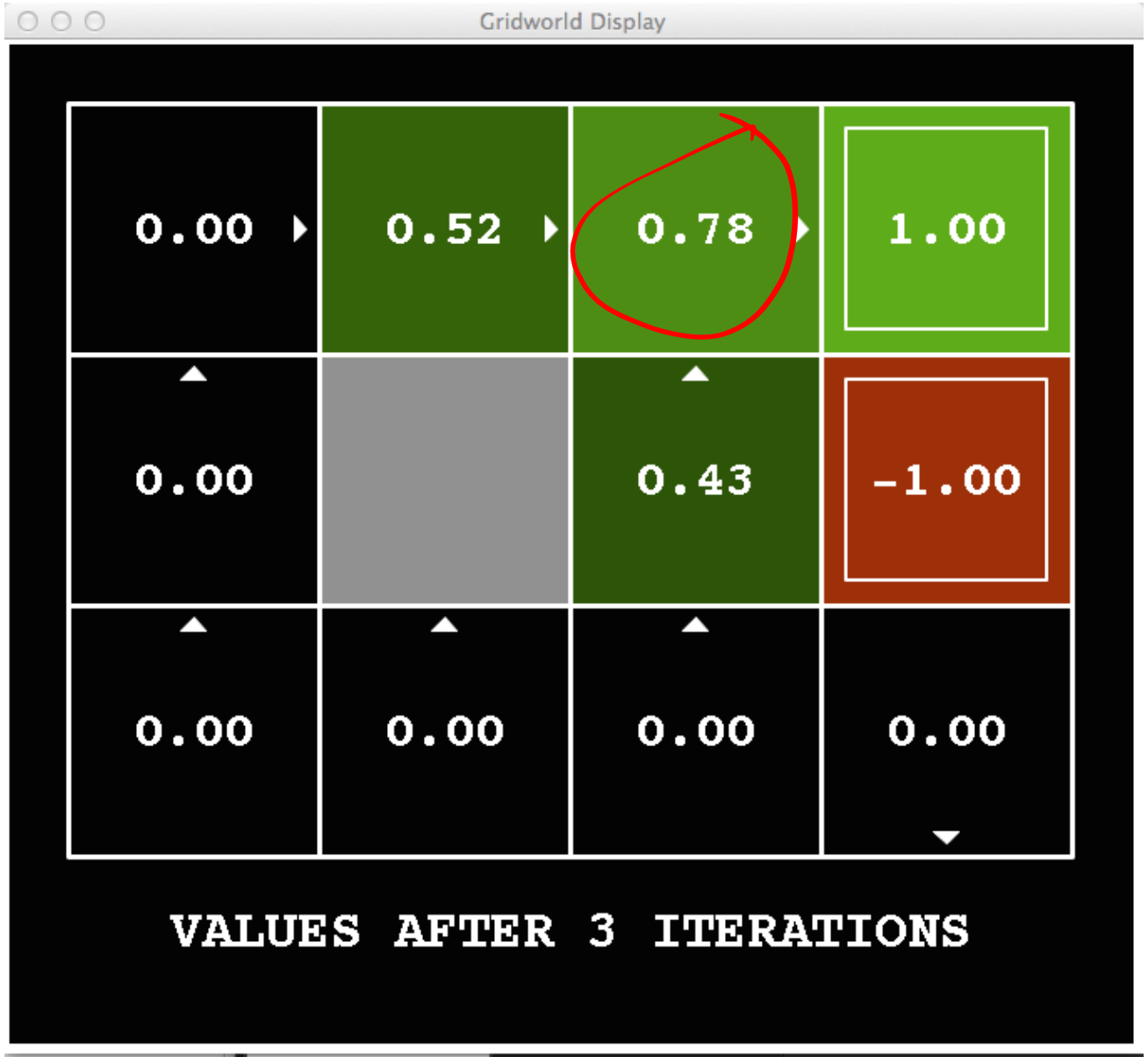
k=2



Noise = 0.2
Discount = 0.9
Living reward = 0

Slide credit: UC Berkeley AI

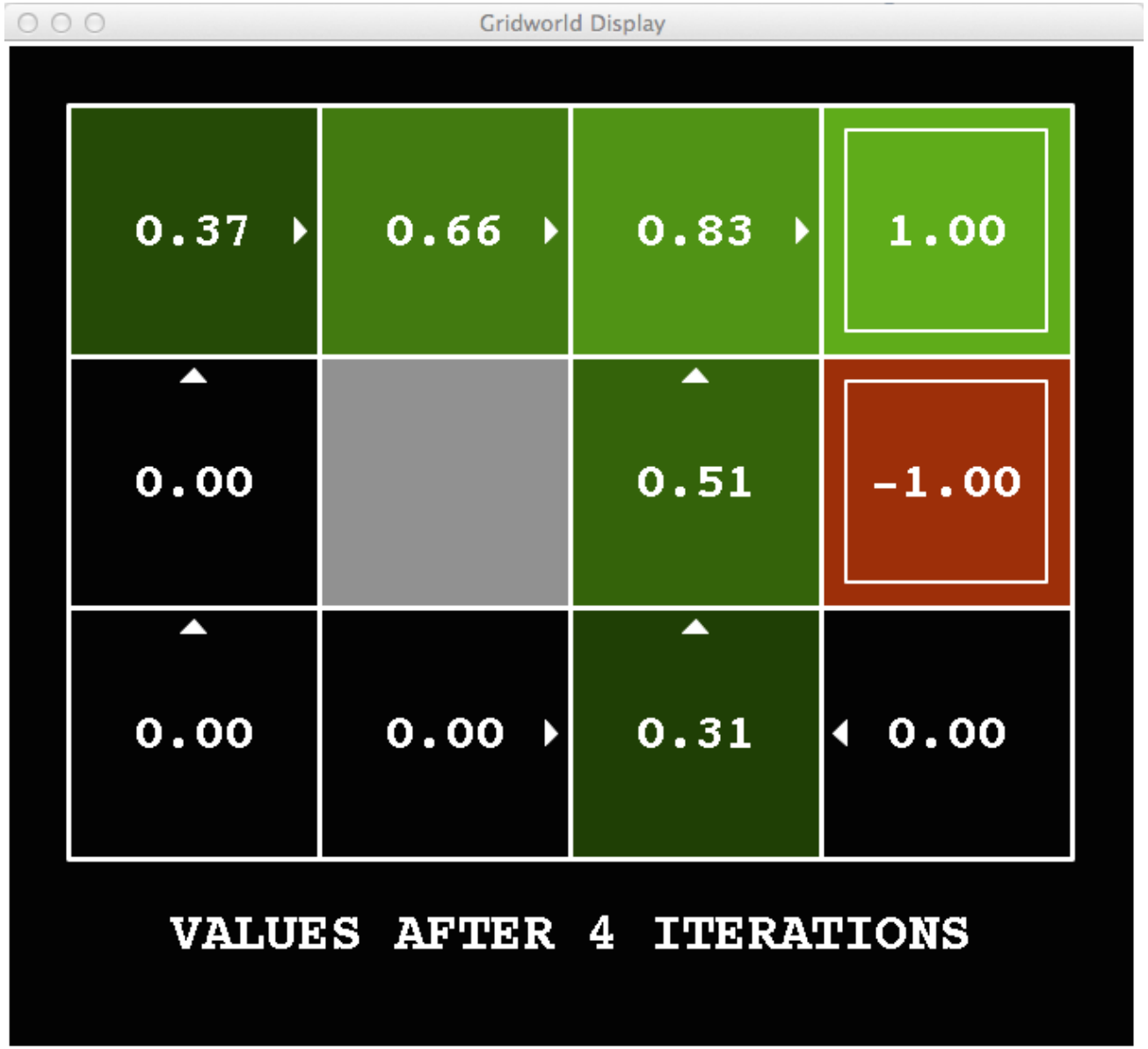
k=3



Noise = 0.2
Discount = 0.9
Living reward = 0

Slide credit: UC Berkeley AI

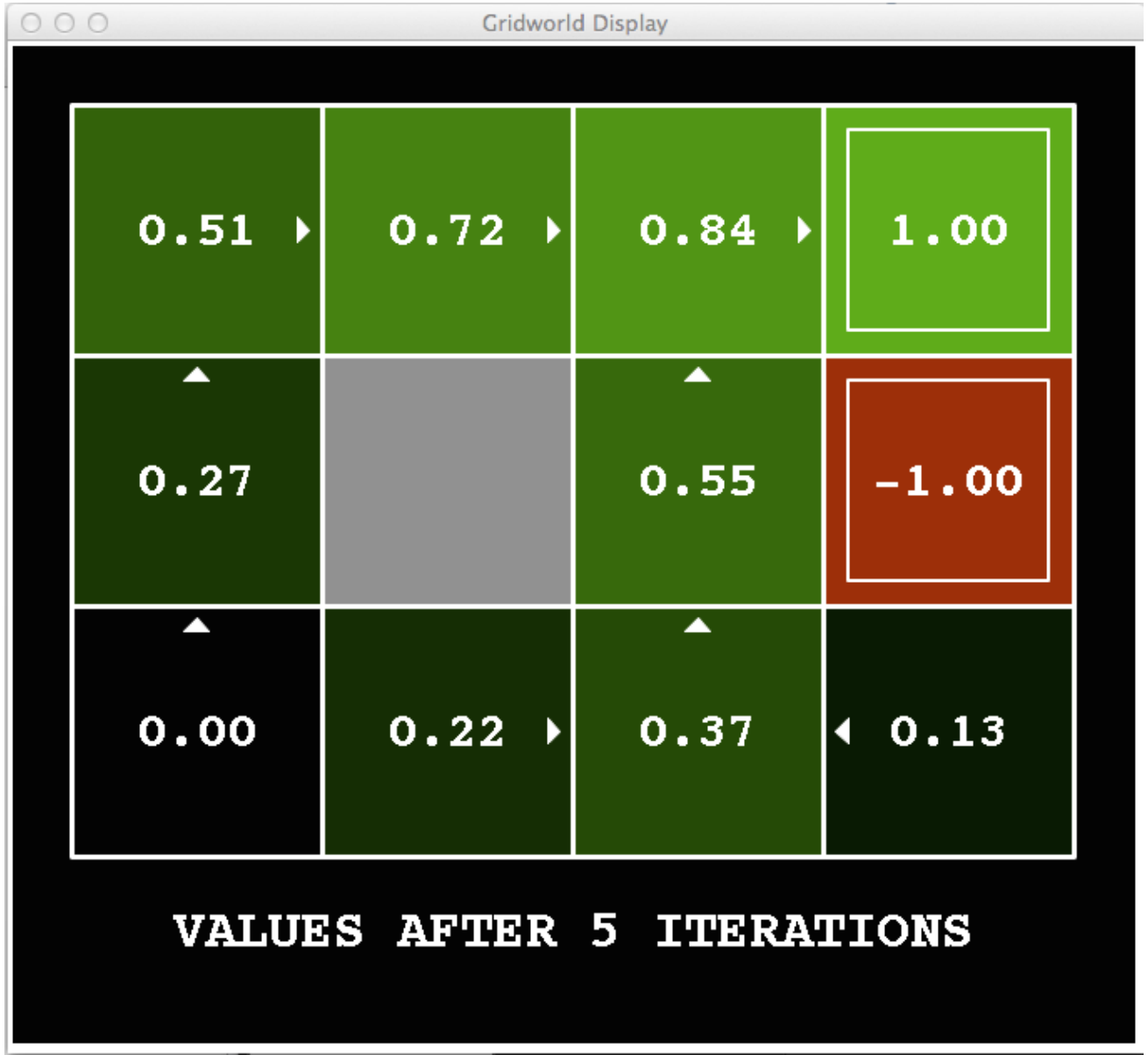
k=4



Noise = 0.2
Discount = 0.9
Living reward = 0

Slide credit: UC Berkeley AI

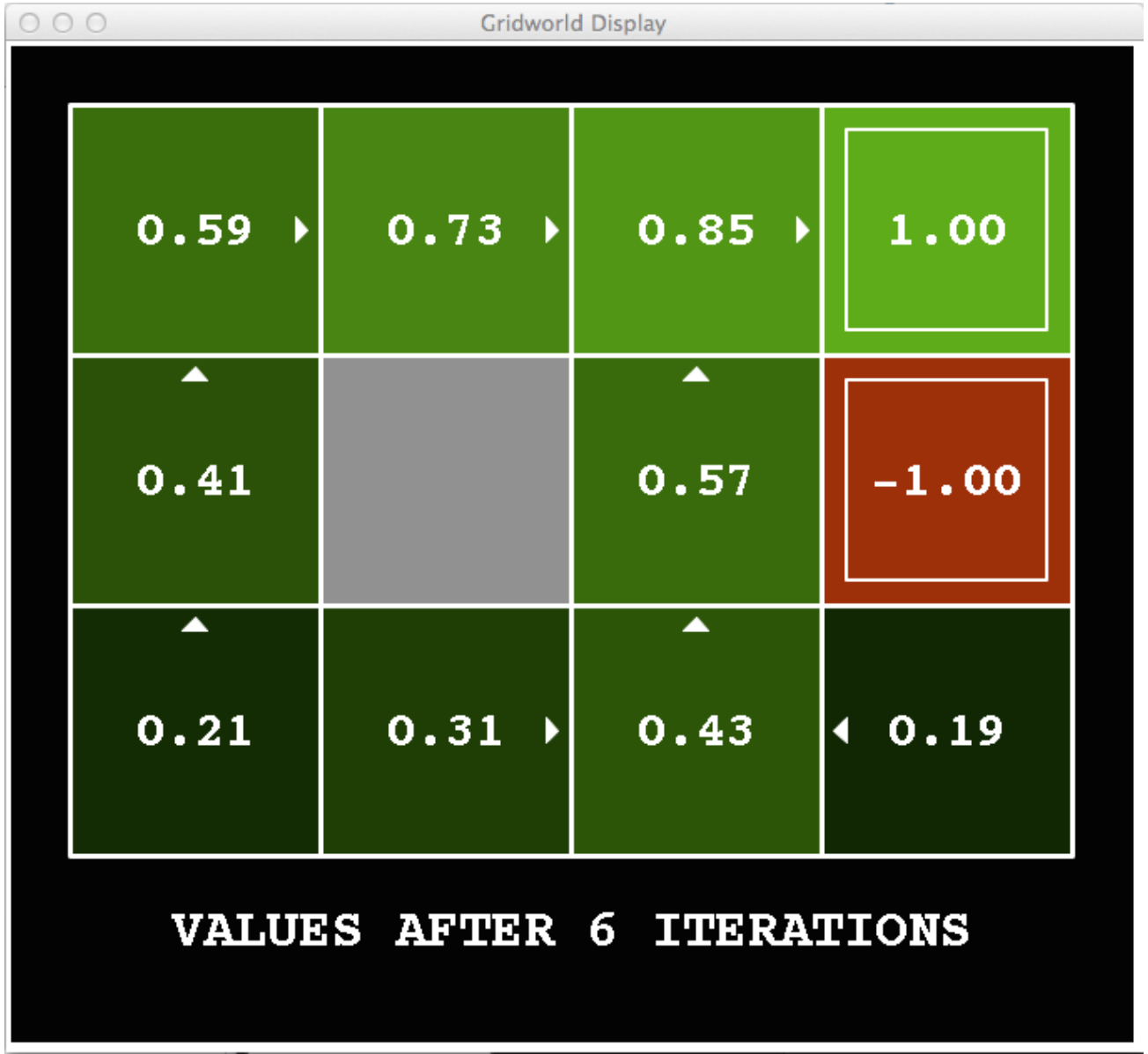
k=5



Noise = 0.2
Discount = 0.9
Living reward = 0

Slide credit: UC Berkeley AI

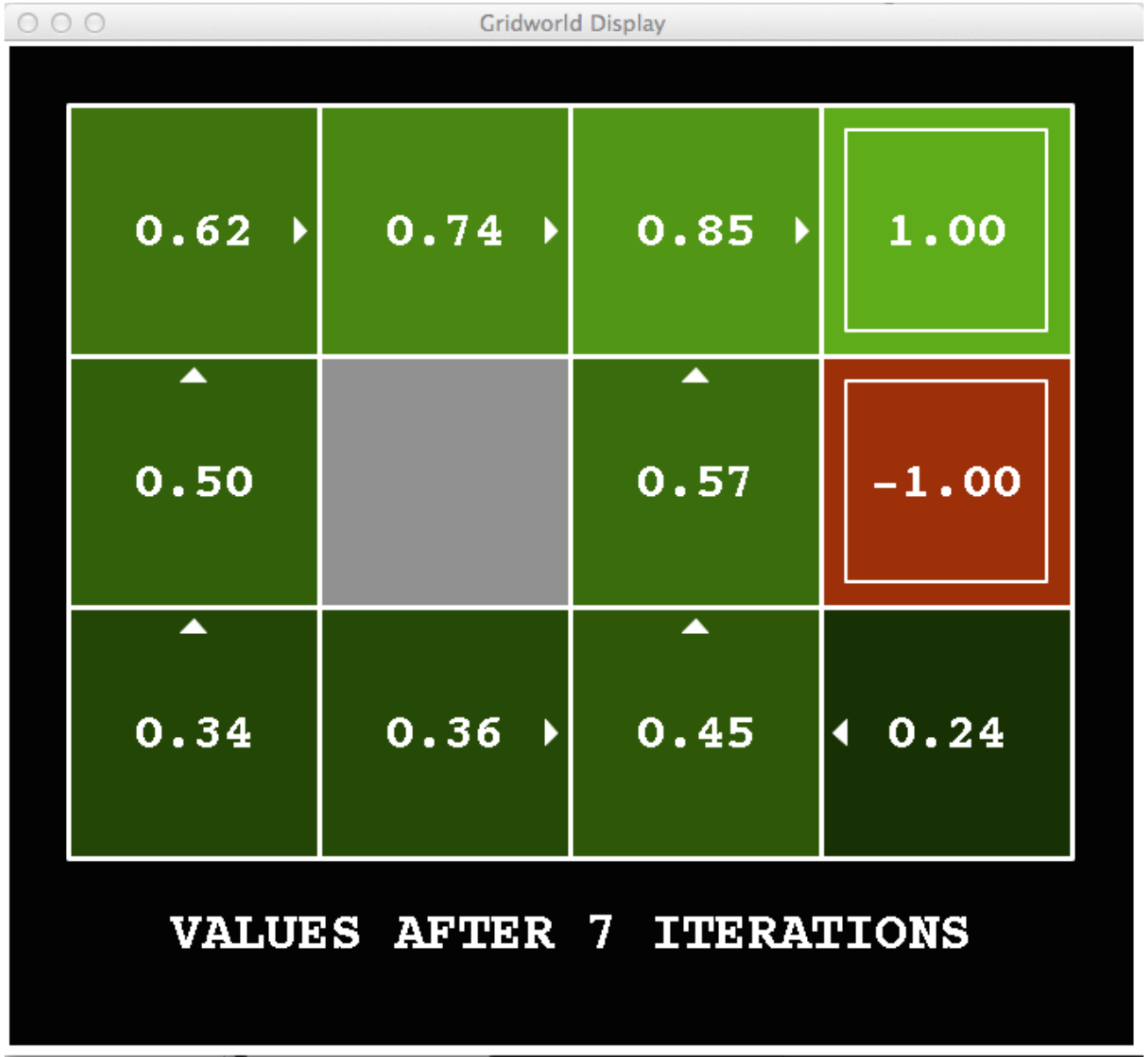
k=6



Noise = 0.2
Discount = 0.9
Living reward = 0

Slide credit: UC Berkeley AI

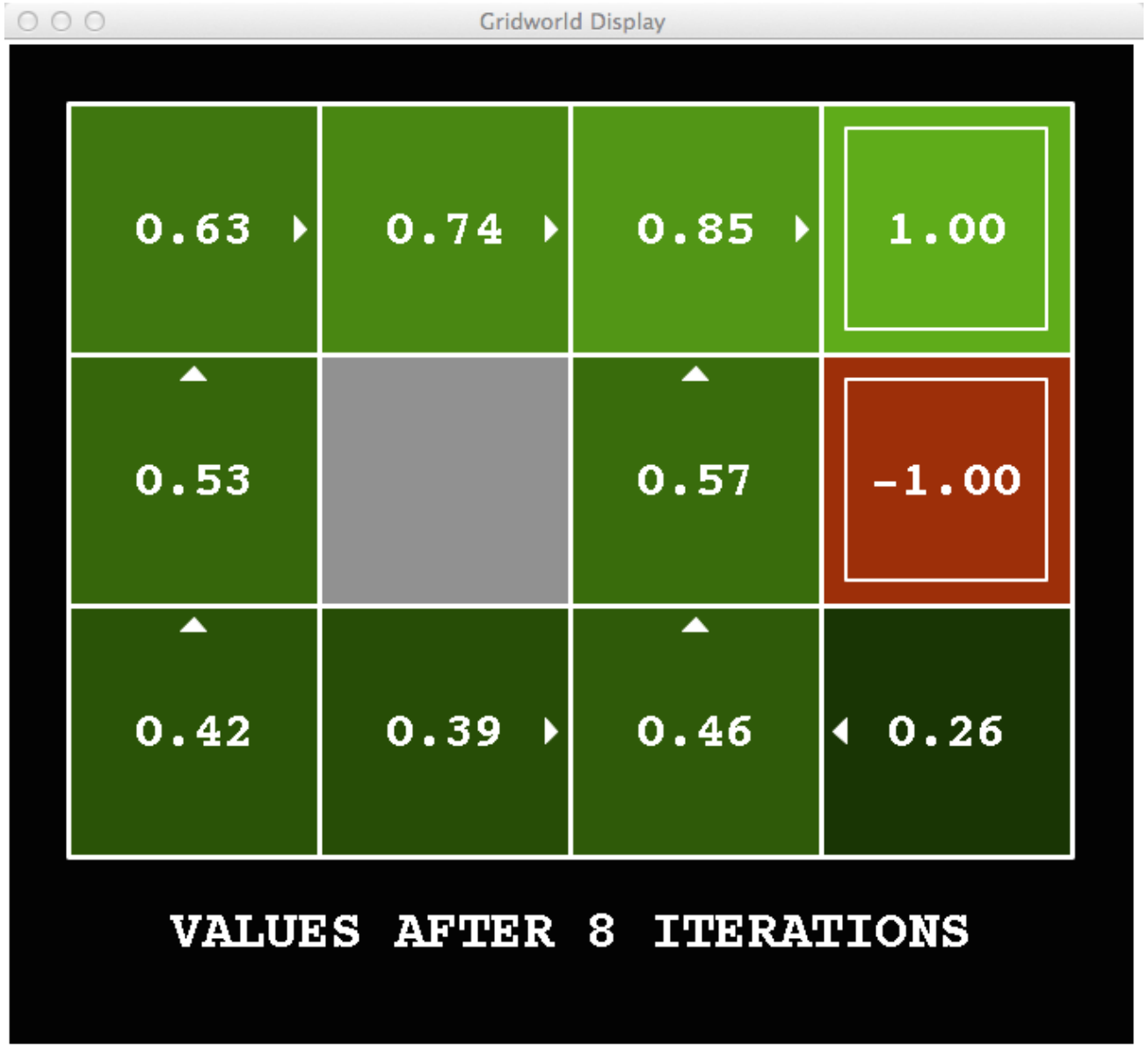
k=7



Noise = 0.2
Discount = 0.9
Living reward = 0

Slide credit: UC Berkeley AI

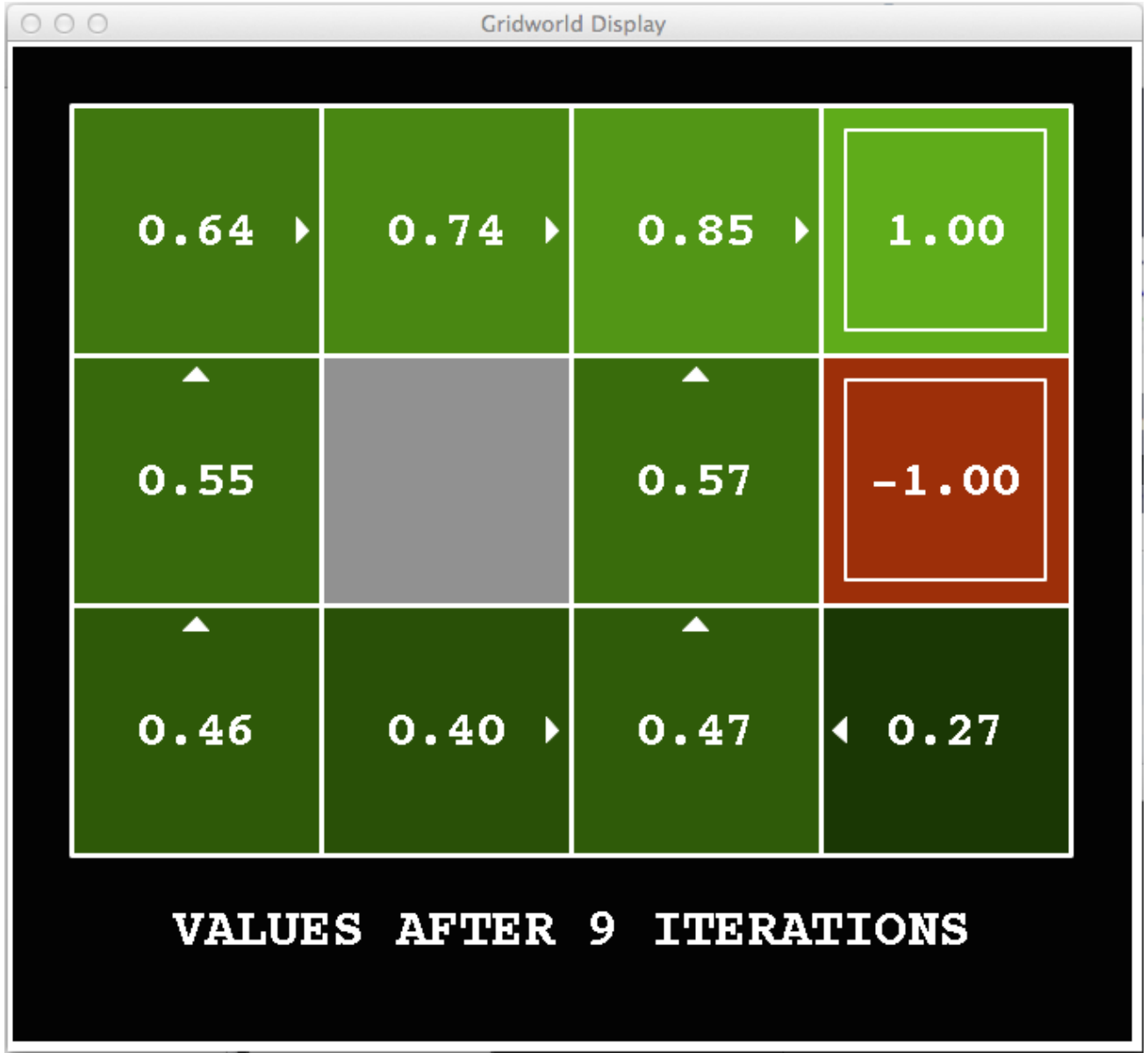
k=8



Noise = 0.2
Discount = 0.9
Living reward = 0

Slide credit: UC Berkeley AI

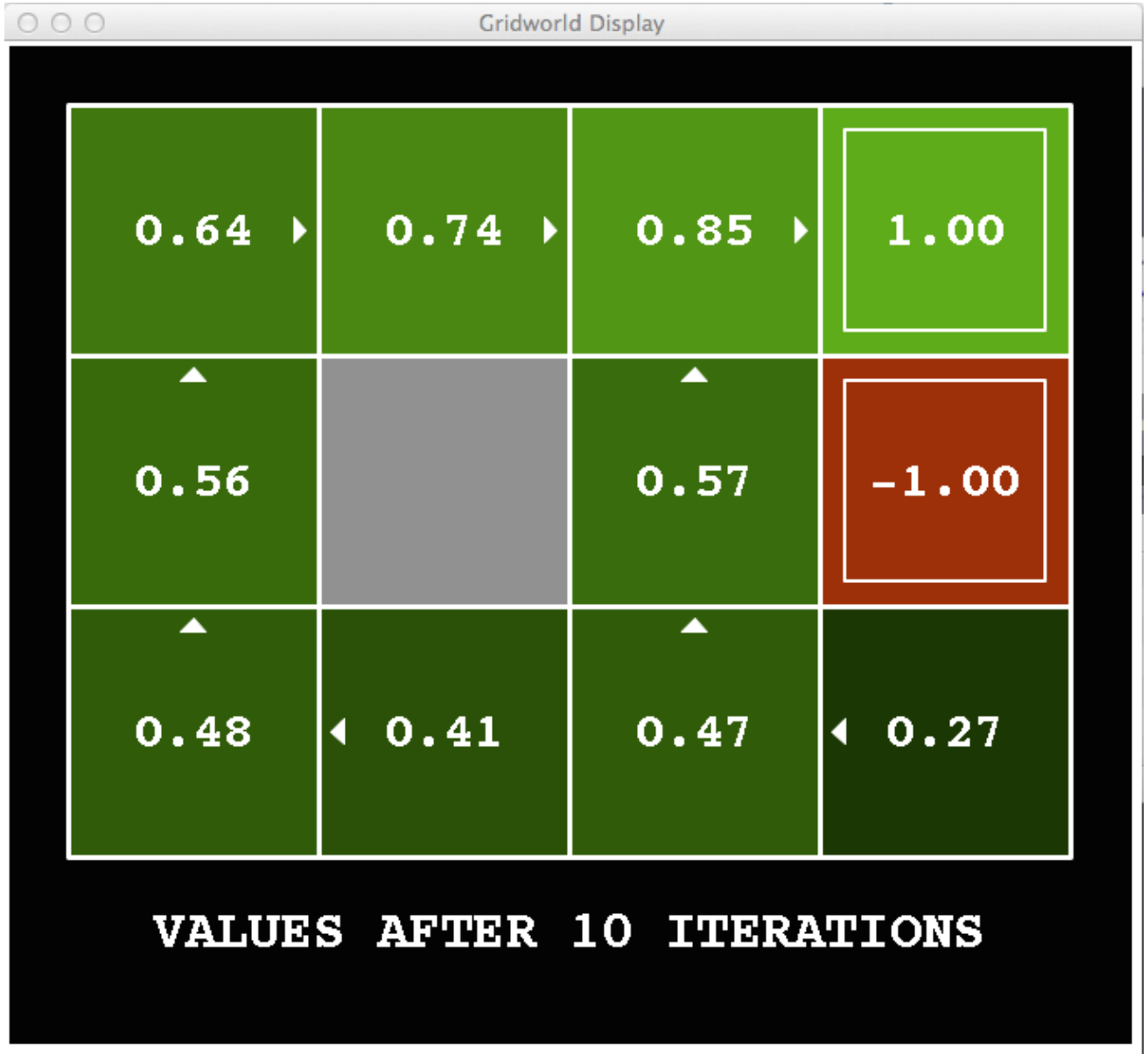
k=9



Noise = 0.2
Discount = 0.9
Living reward = 0

Slide credit: UC Berkeley AI

k=10



Noise = 0.2
Discount = 0.9
Living reward = 0

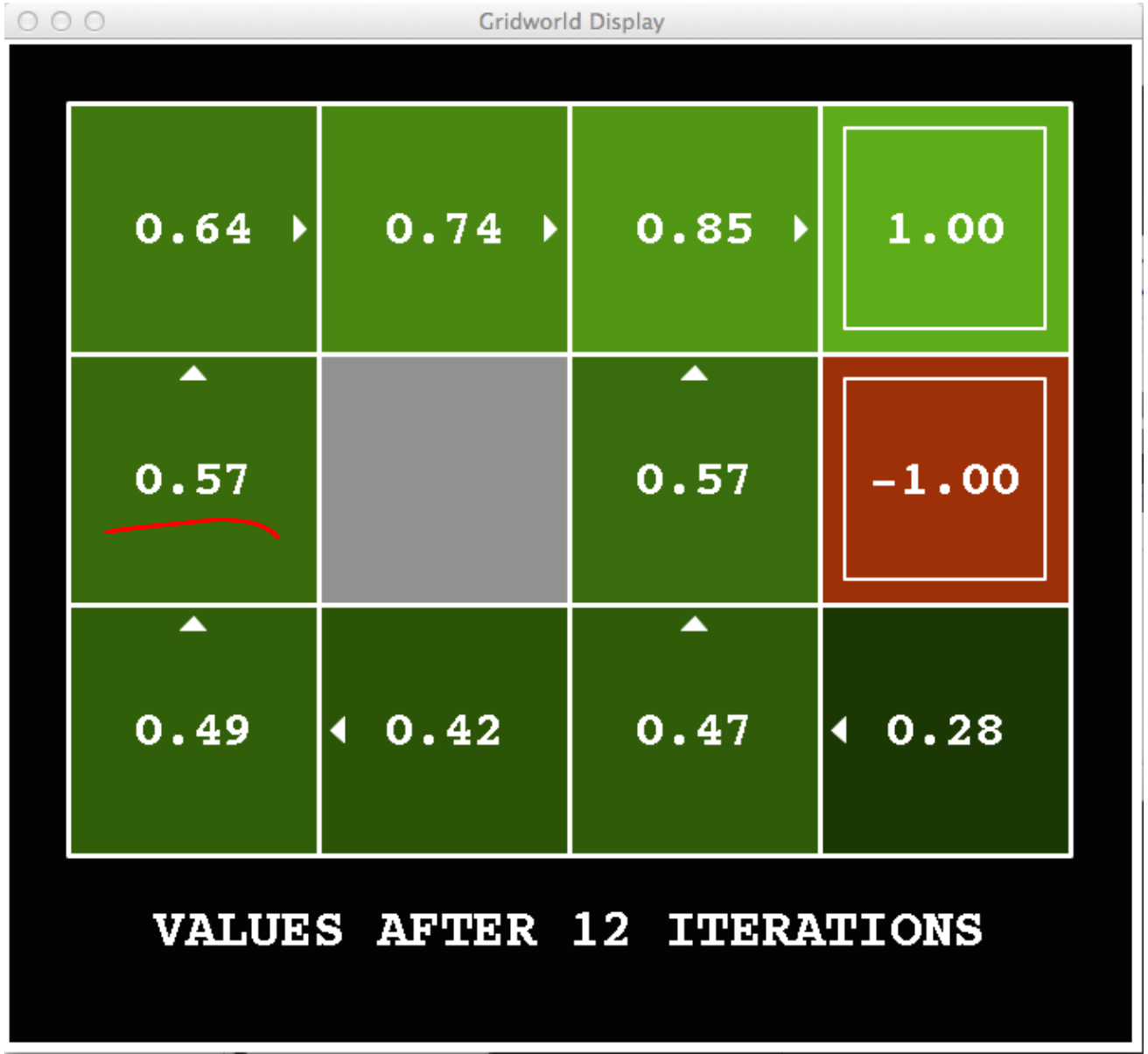
Slide credit: UC Berkeley AI

k=11

Noise = 0.2
Discount = 0.9
Living reward = 0

Slide credit: UC Berkeley AI

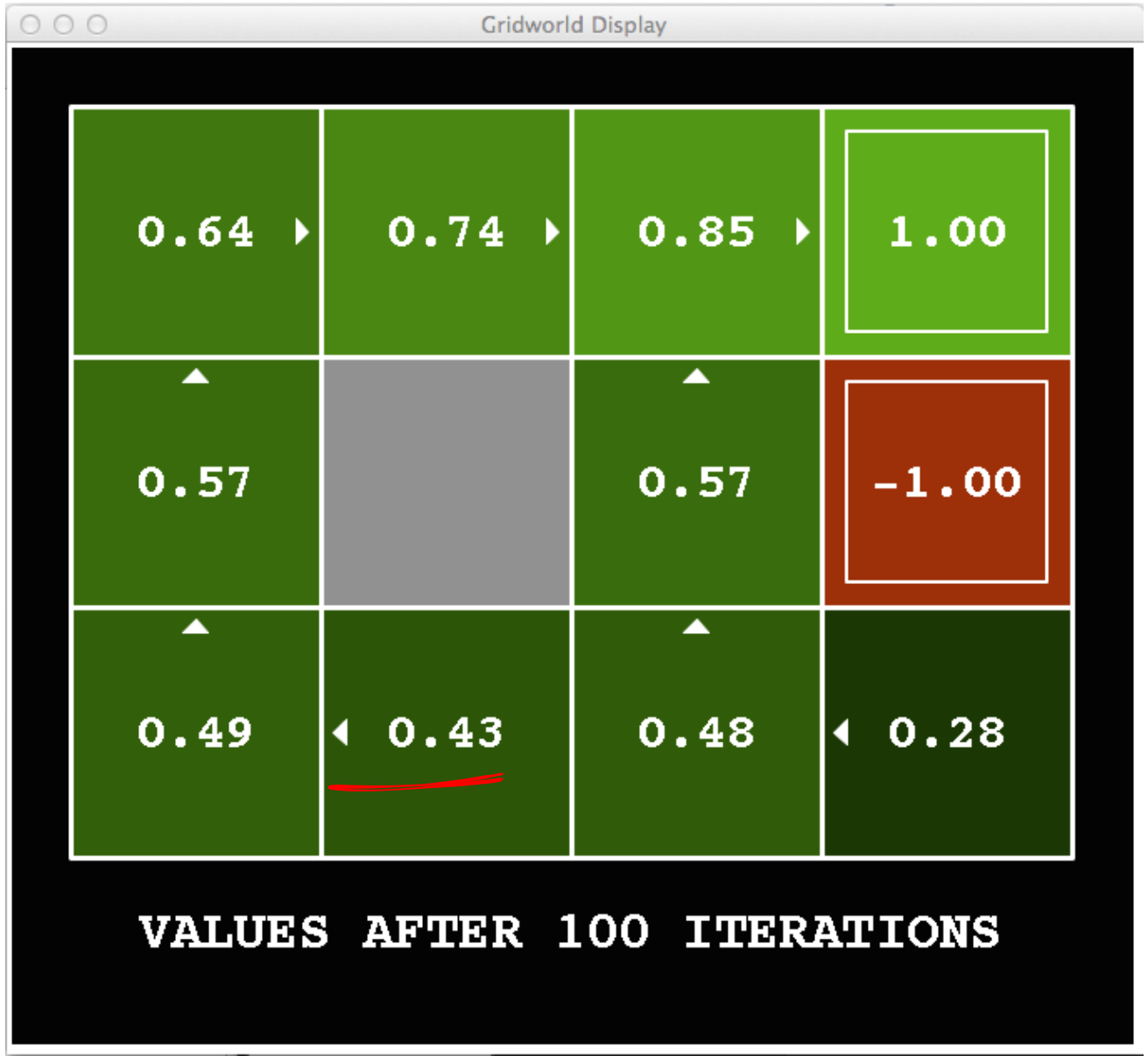
k=12



Noise = 0.2
Discount = 0.9
Living reward = 0

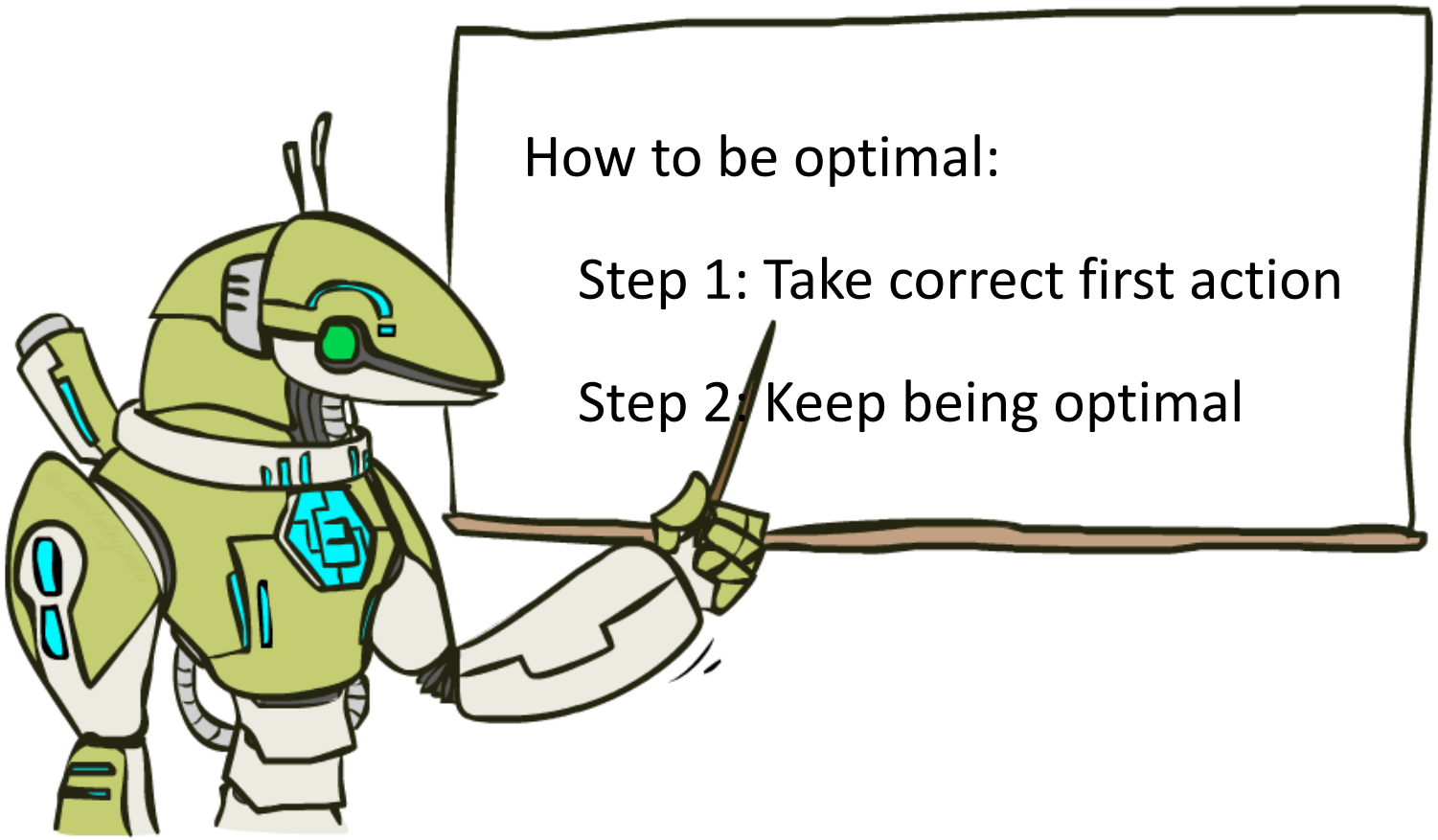
Slide credit: UC Berkeley AI

k=100



Noise = 0.2
Discount = 0.9
Living reward = 0

Slide credit: UC Berkeley AI



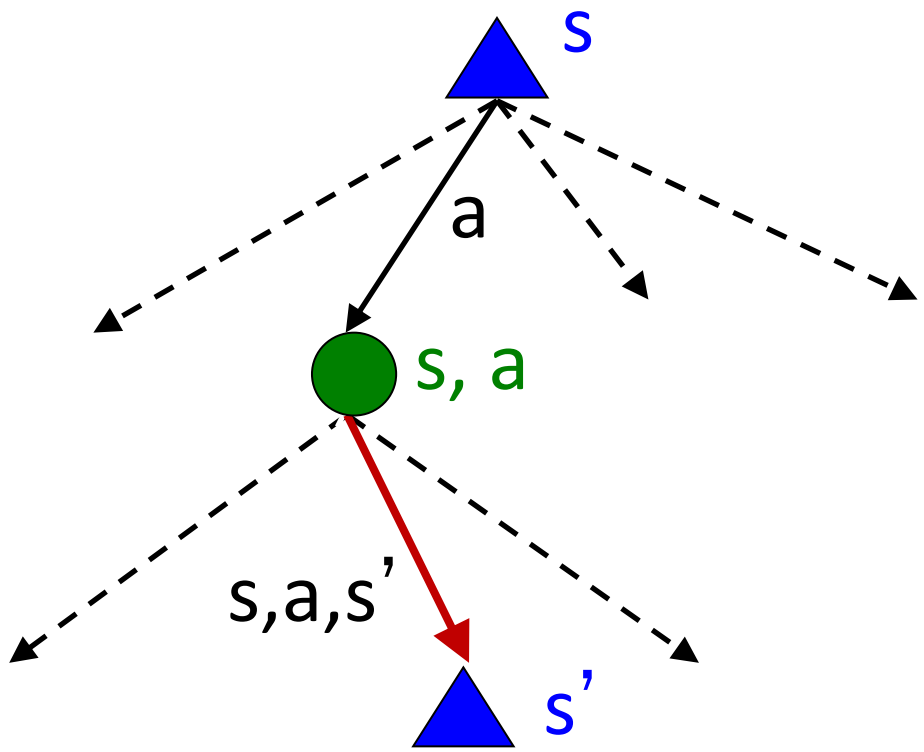
Optimal Values & Bellman Equations

Image credit: *Ketrina Yim, UC Berkeley*

Optimal Quantities

- The value (utility) of a state s:

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



Slide credit: UC Berkeley AI

[Demo: gridworld values (L9D1)]

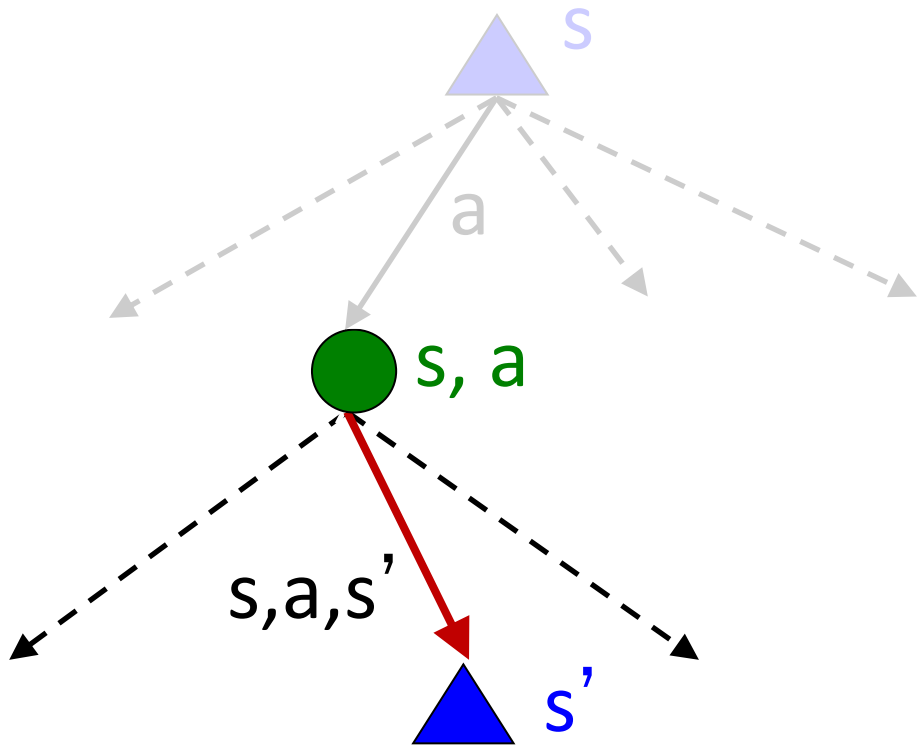
Optimal Quantities

- 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



Slide credit: UC Berkeley AI

[Demo: gridworld values (L9D1)]

Optimal Quantities

- The value (utility) of a state s:

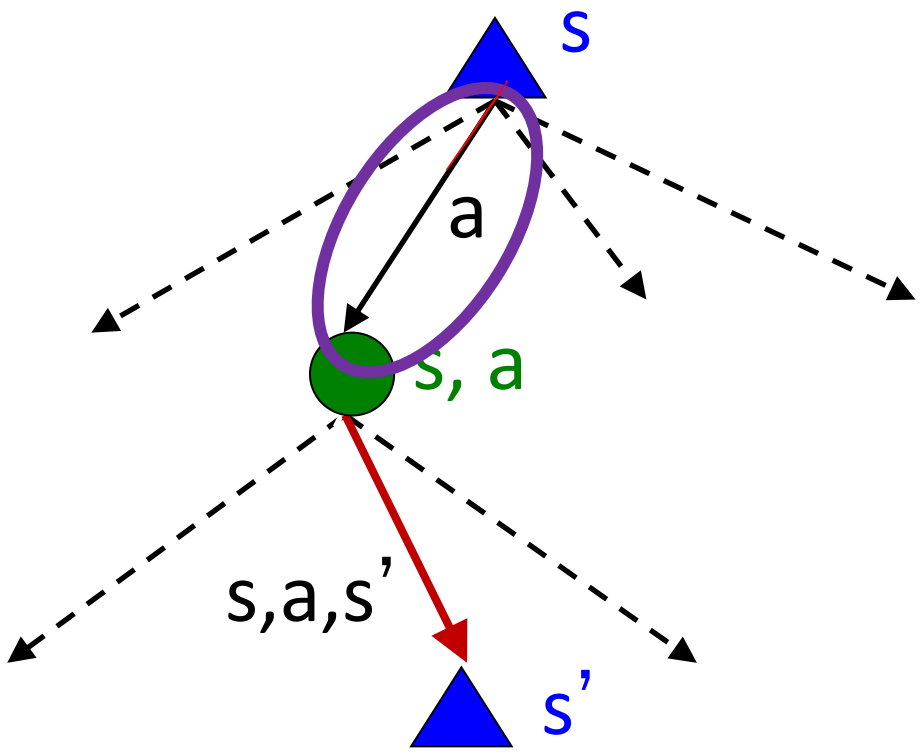
  $V^*(s)$ = expected utility starting in s and acting optimally

- 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



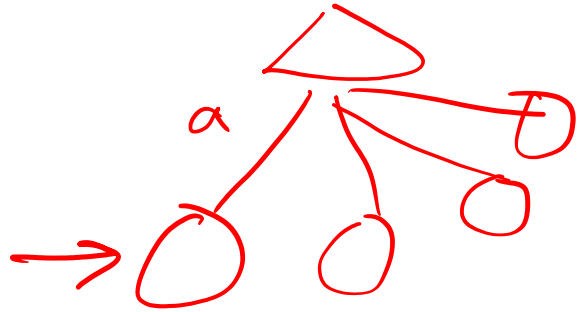
Slide credit: UC Berkeley AI

[Demo: gridworld values (L9D1)]

Gridworld Values V*

Slide credit: UC Berkeley AI

Gridworld: Q*



Slide credit: UC Berkeley AI

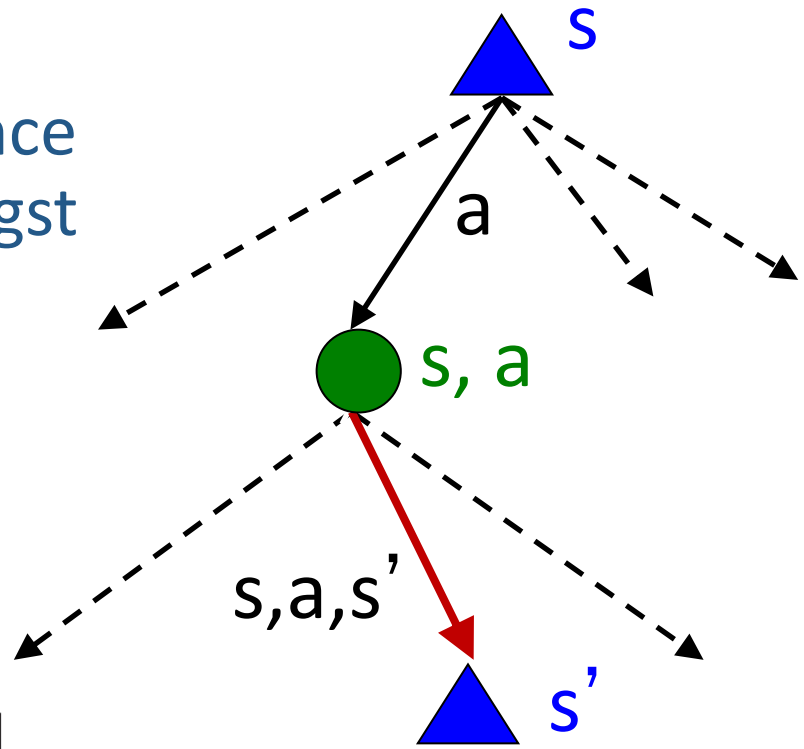
The Bellman Equations

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

Slide credit: UC Berkeley AI

Summary: Bellman Equations vs Value Iteration

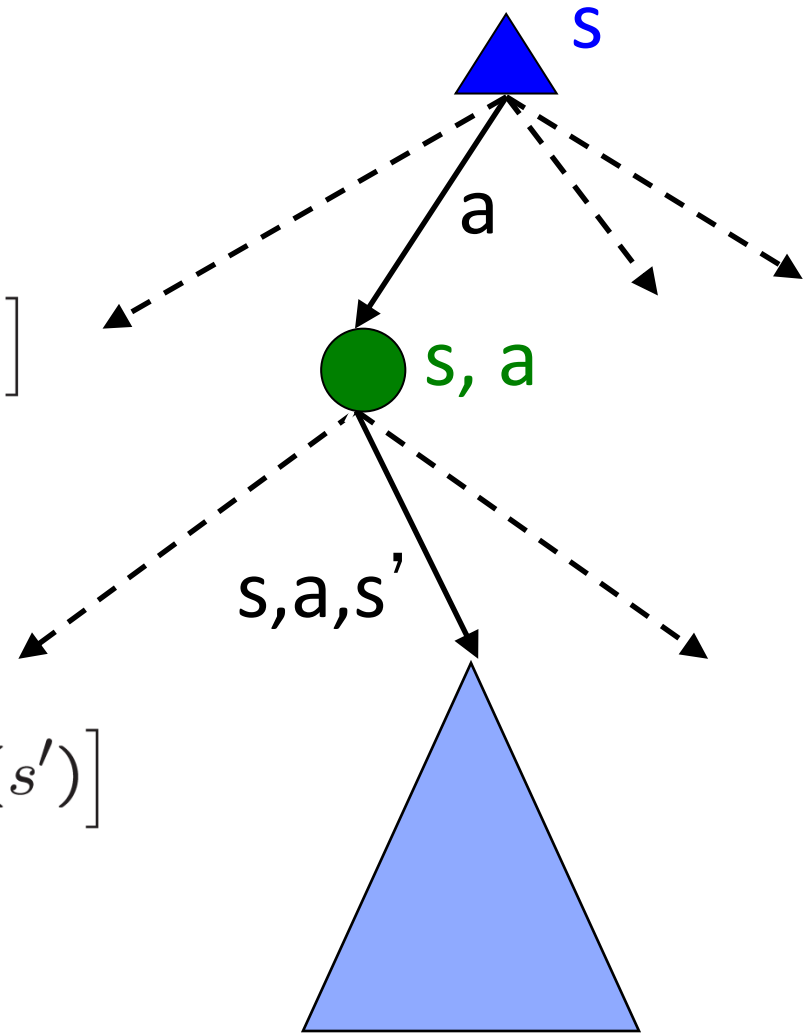
Bellman equations characterize the optimal values:

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

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



Slide credit: UC Berkeley AI

MDP Notation

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

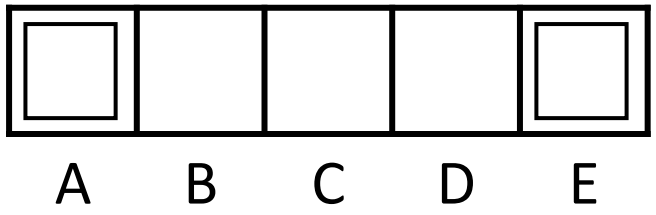
Bellman equations: $V^*(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^*(s')]$

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

Outline

MDP Setup

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



Solving MDPs

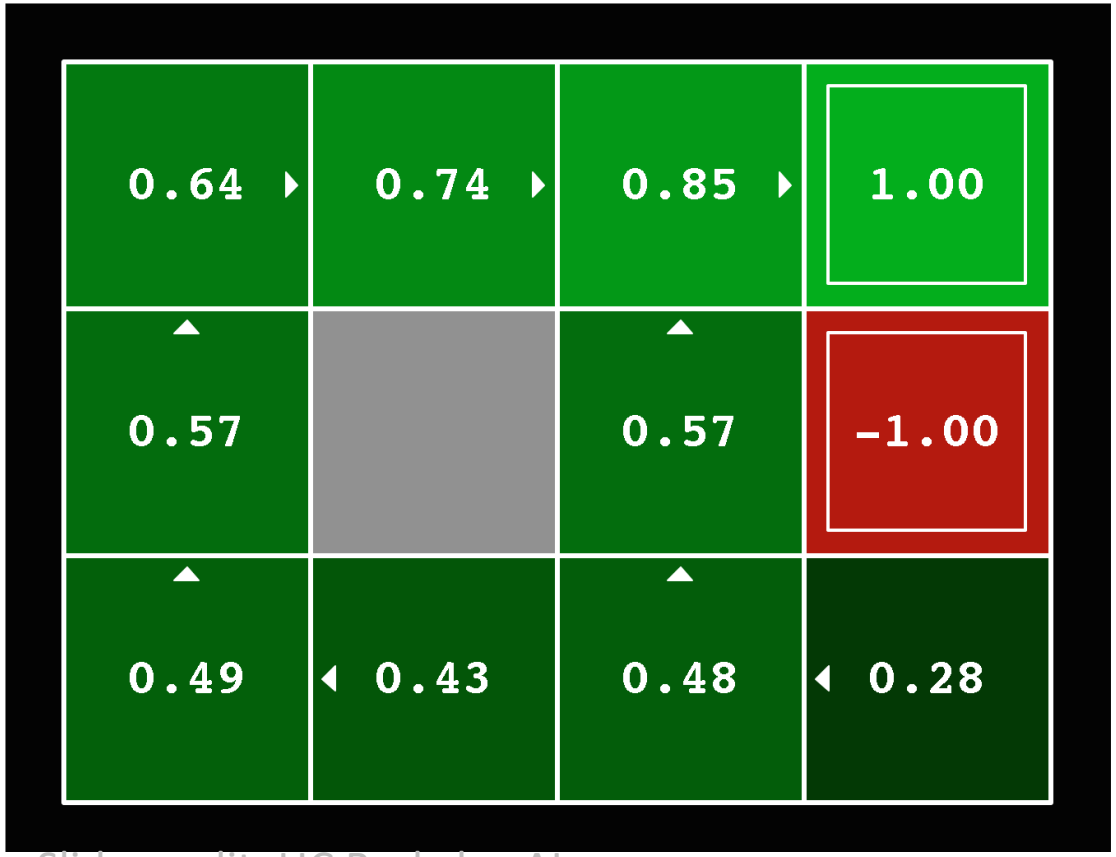
- Method 1) Value iteration
  - Value iteration convergence
- Optimal values & Bellman equations
- Policy Extraction
- Method 2) Policy Iteration

Image credit: *Ketrina Yim, UC Berkeley*

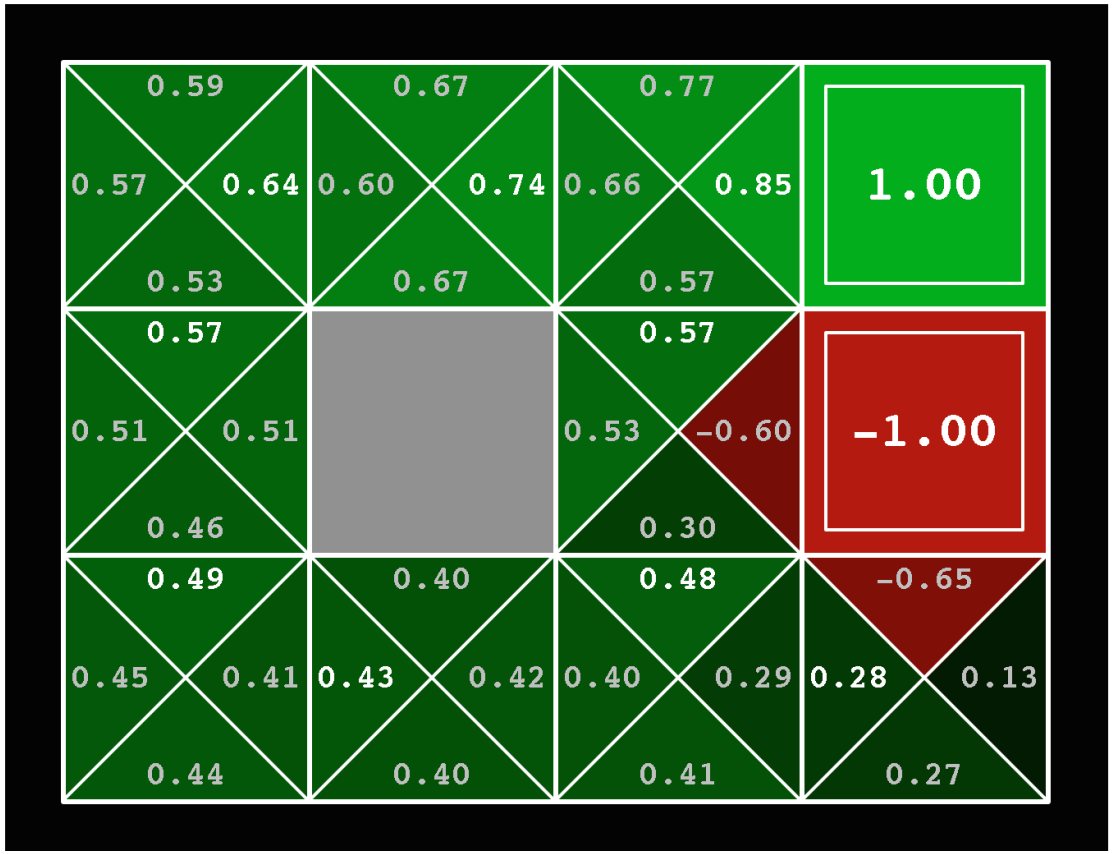
Solved MDP! Now what?

What are we going to do with these values??

$V^*(s)$



$Q^*(s,a)$

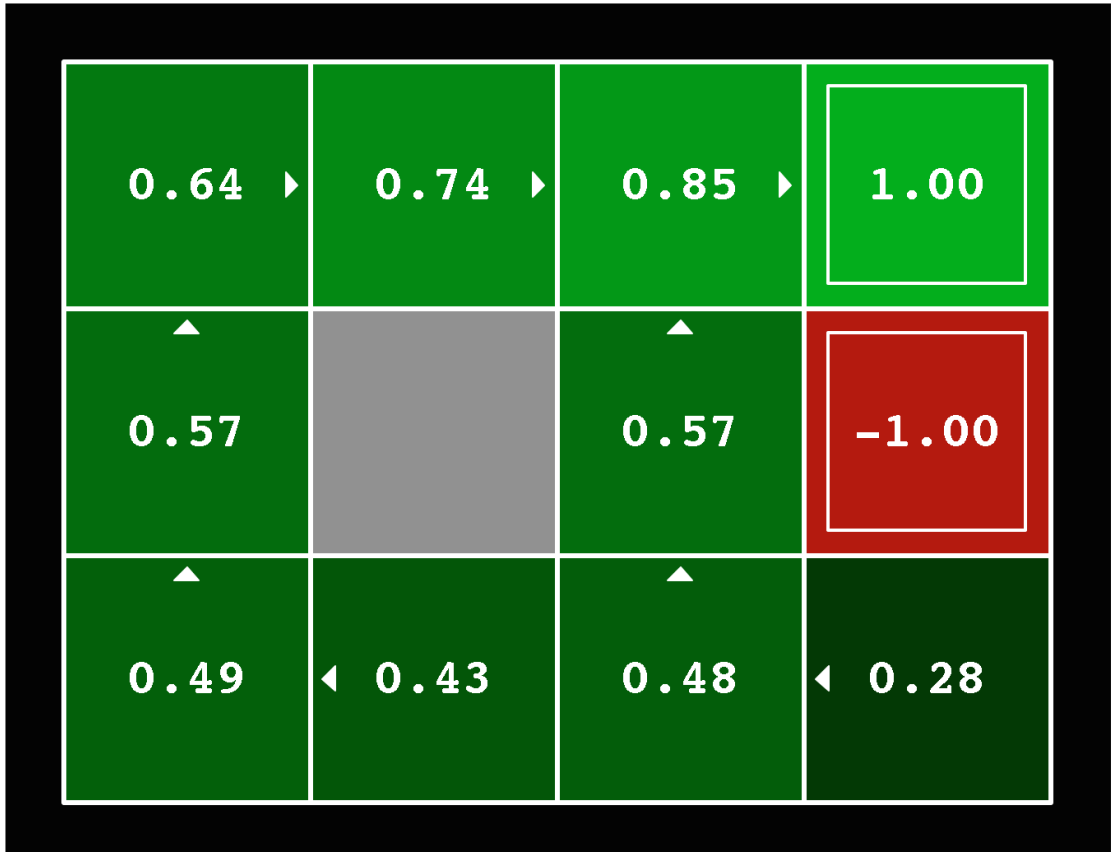


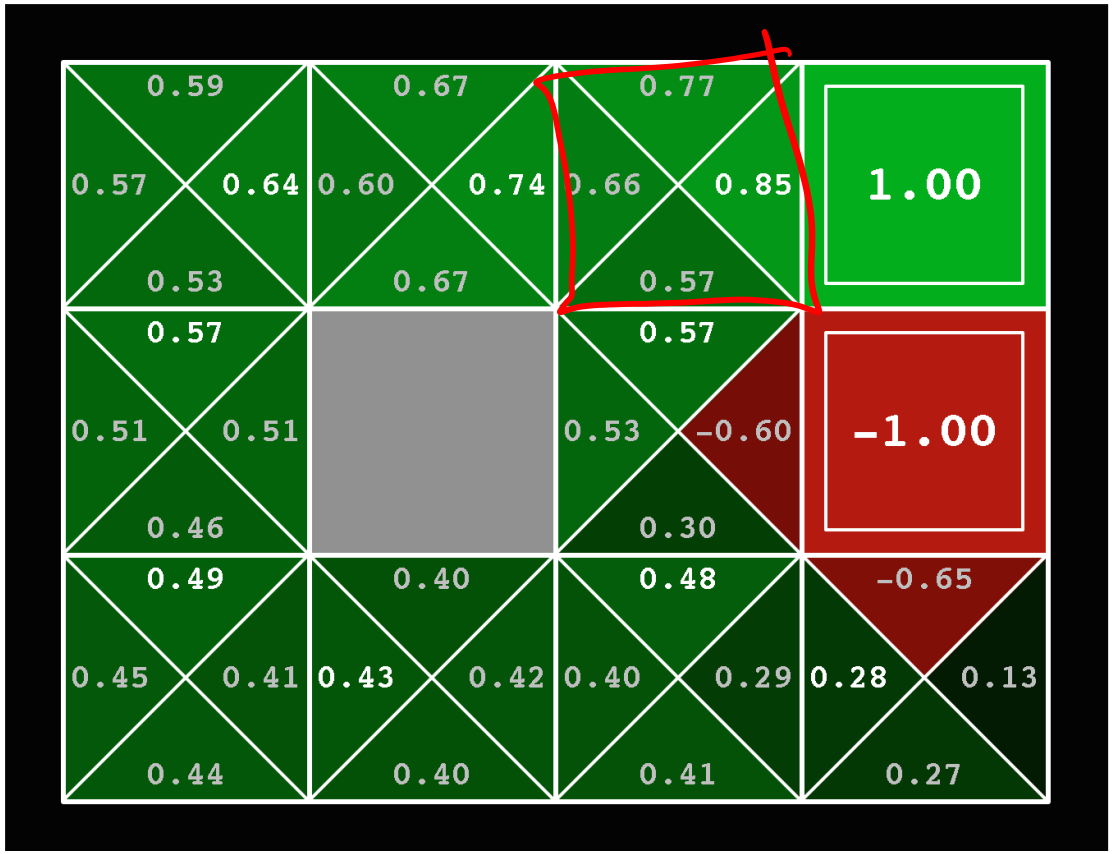
Slide credit: UC Berkeley AI

Poll 1

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

A) Values, B) Q-values?





Policy Extraction

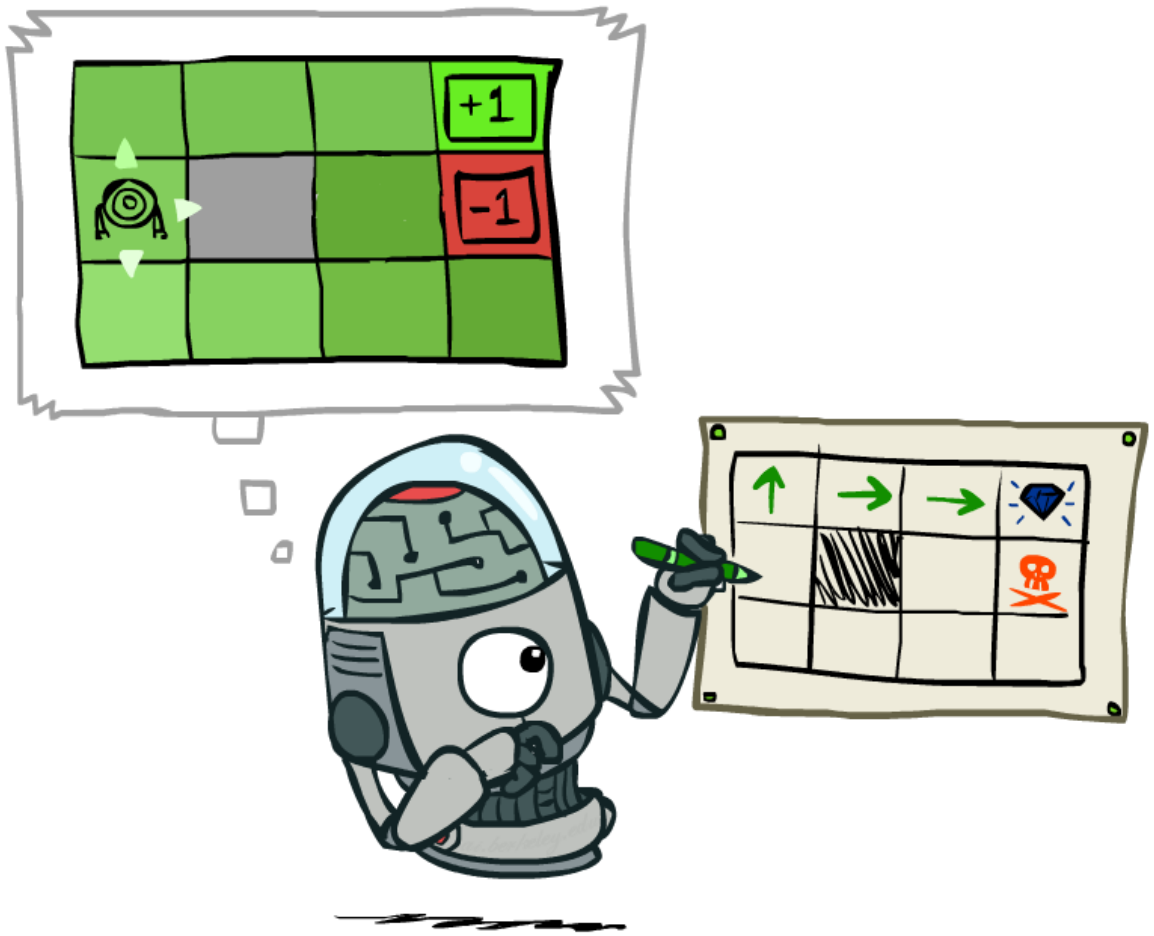
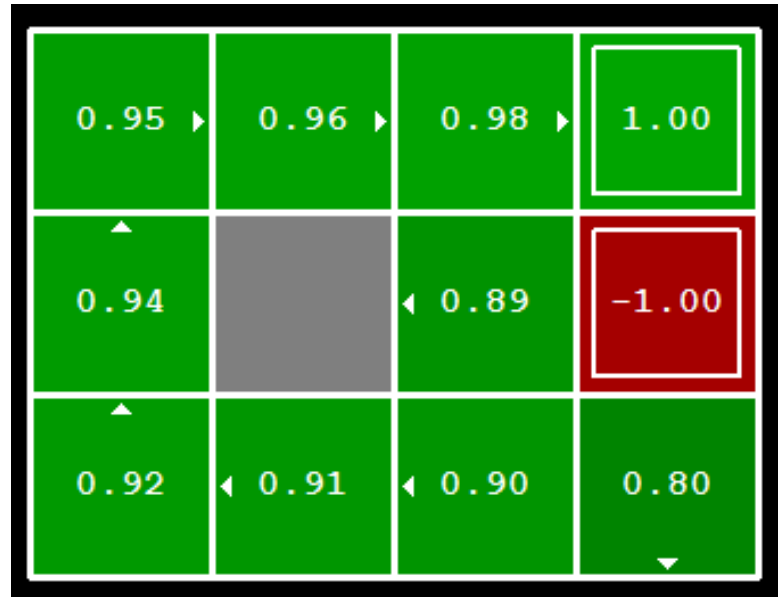


Image credit: *Ketrina Yim, UC Berkeley*

Computing Actions from Values

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



How should we act?

- It's not obvious!

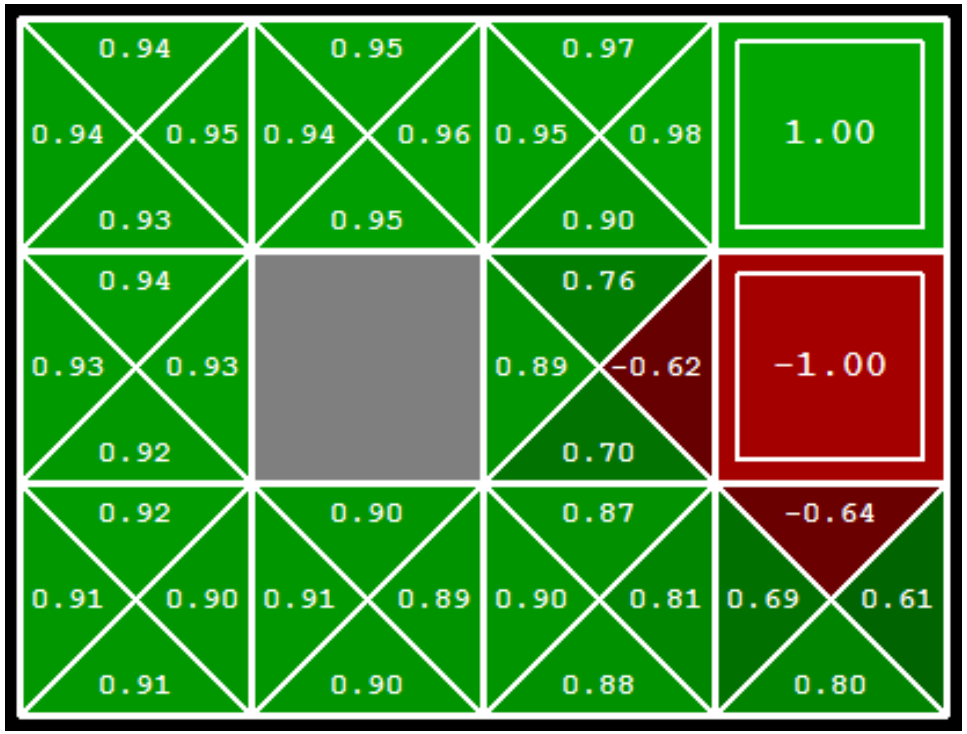
We need to do a mini-expectimax (one step)

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

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!

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

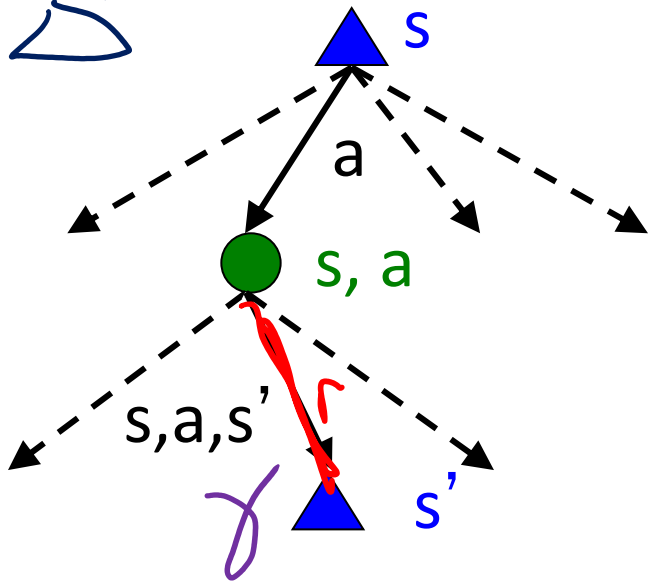
Important lesson: actions are easier to select from q-values than values!

Slide credit: UC Berkeley AI

Value Iteration Notes

Value iteration repeats the Bellman updates:

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



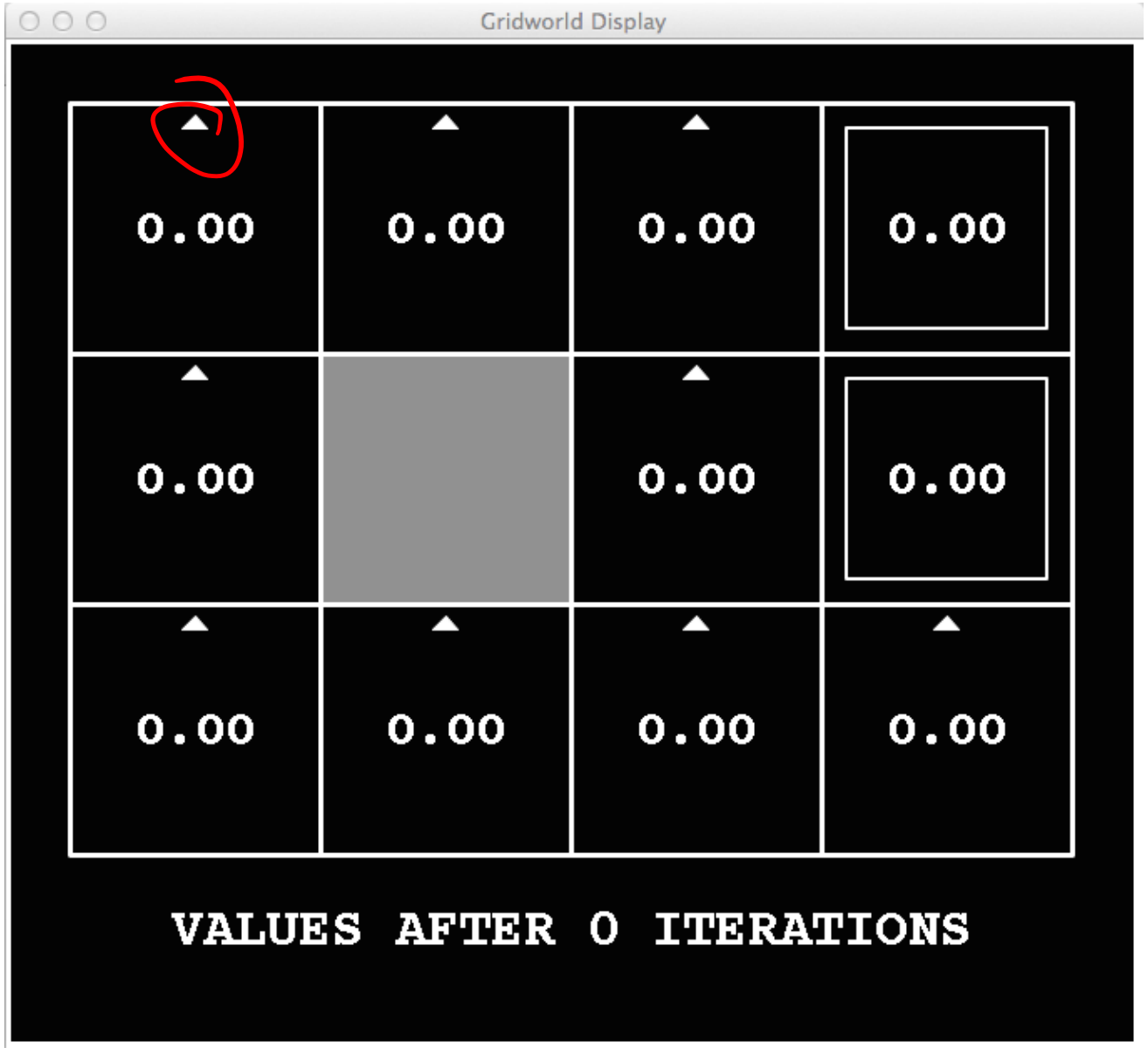
Things to notice when running value iteration:

- 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)

Slide credit: UC Berkeley AI

[Demo: value iteration (L9D2)]

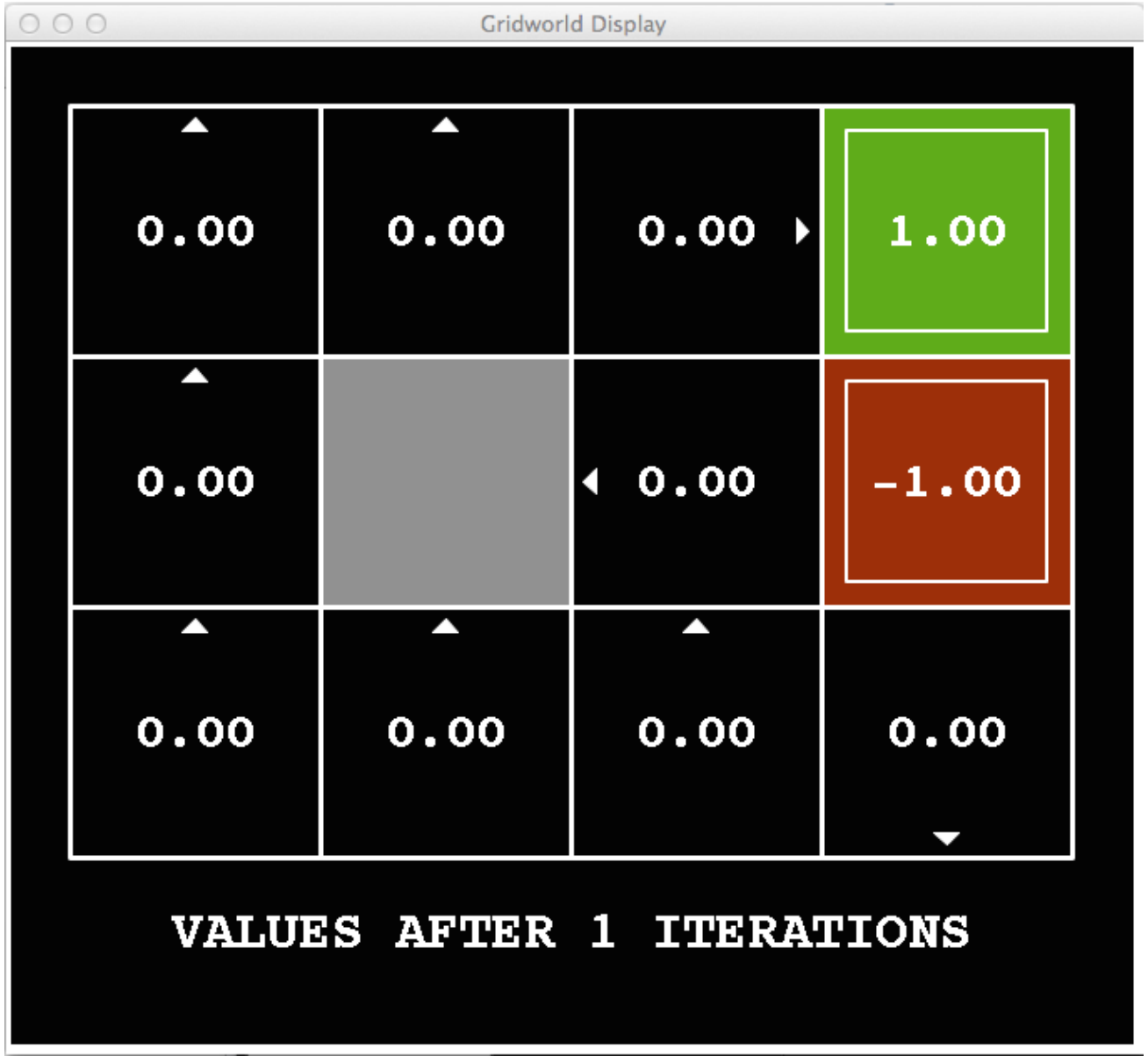
k=0



Noise = 0.2
Discount = 0.9
Living reward = 0

Slide credit: UC Berkeley AI

k=1



Noise = 0.2
Discount = 0.9
Living reward = 0

Slide credit: UC Berkeley AI

k=2

Noise = 0.2
Discount = 0.9
Living reward = 0

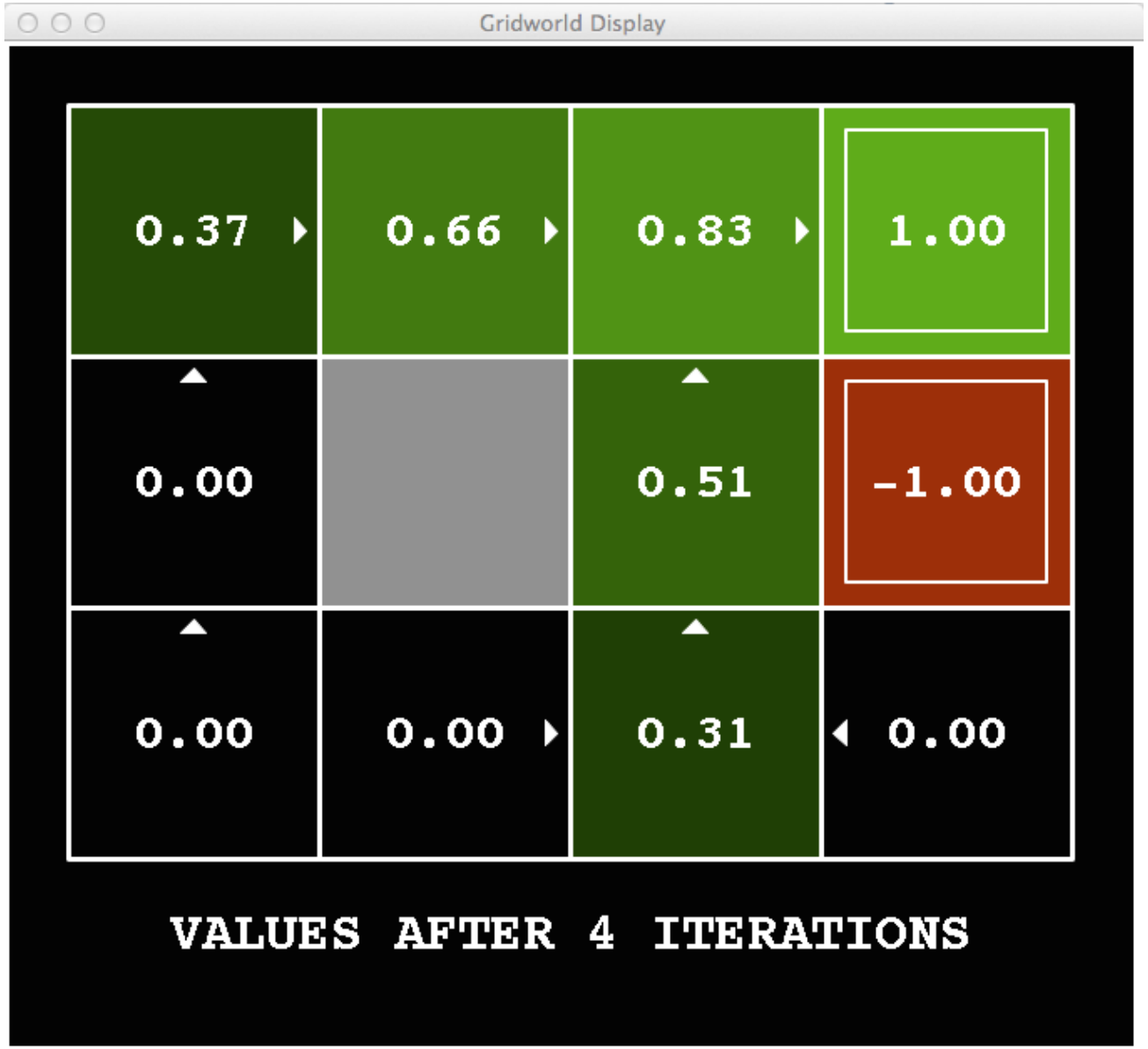
Slide credit: UC Berkeley AI

k=3

Noise = 0.2
Discount = 0.9
Living reward = 0

Slide credit: UC Berkeley AI

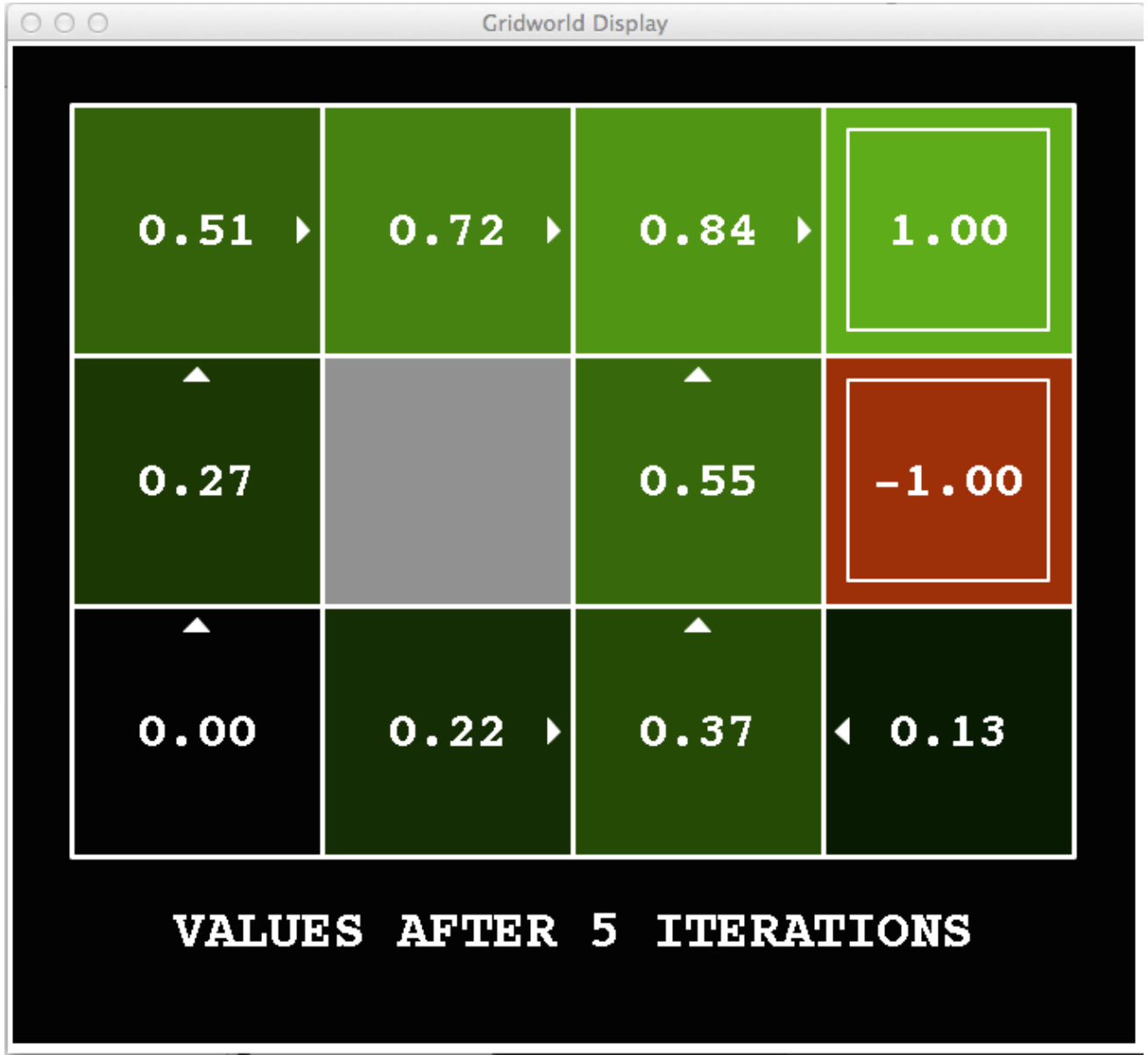
k=4



Noise = 0.2
Discount = 0.9
Living reward = 0

Slide credit: UC Berkeley AI

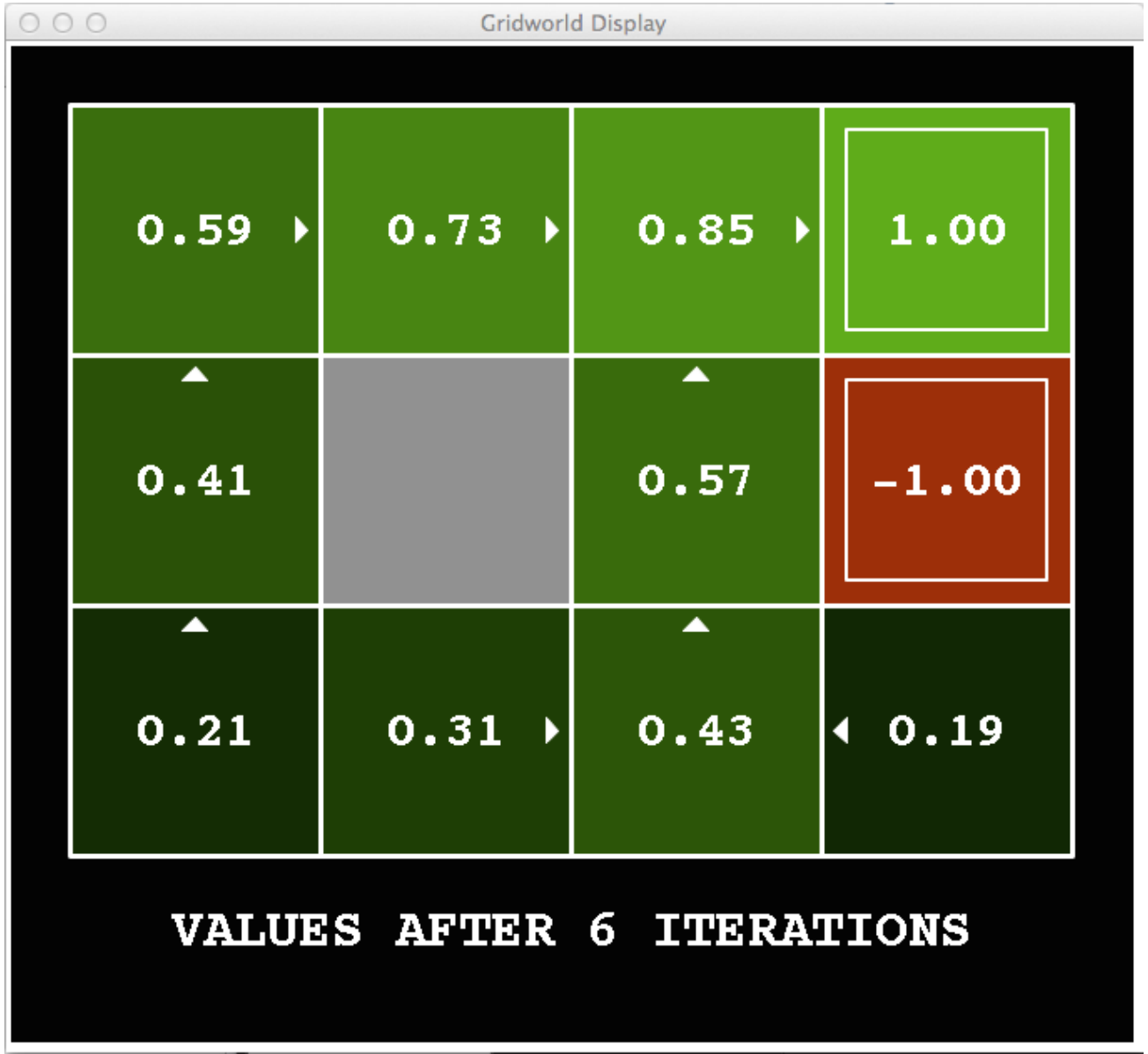
k=5



Noise = 0.2
Discount = 0.9
Living reward = 0

Slide credit: UC Berkeley AI

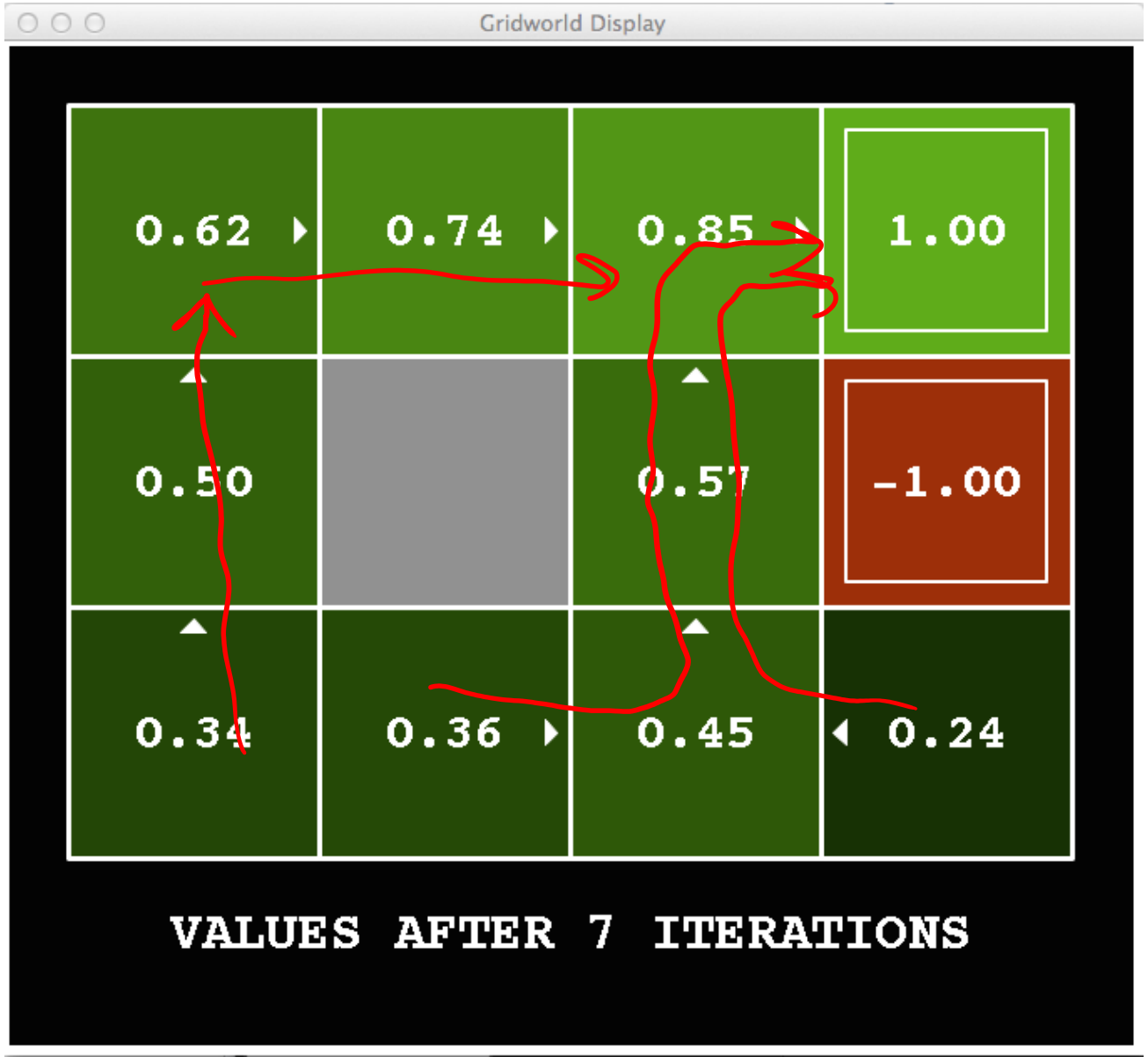
k=6



Noise = 0.2
Discount = 0.9
Living reward = 0

Slide credit: UC Berkeley AI

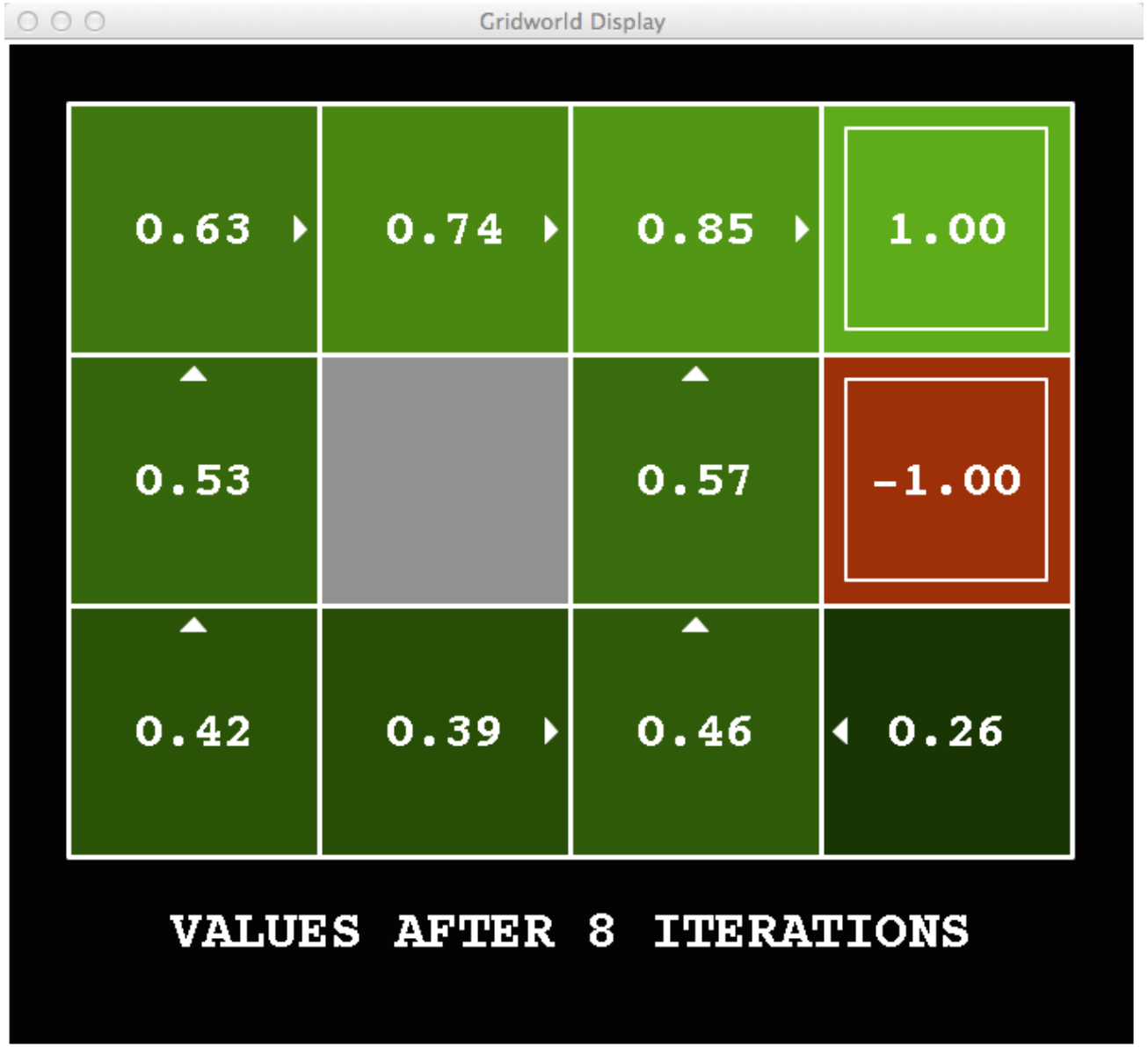
k=7



Noise = 0.2
Discount = 0.9
Living reward = 0

Slide credit: UC Berkeley AI

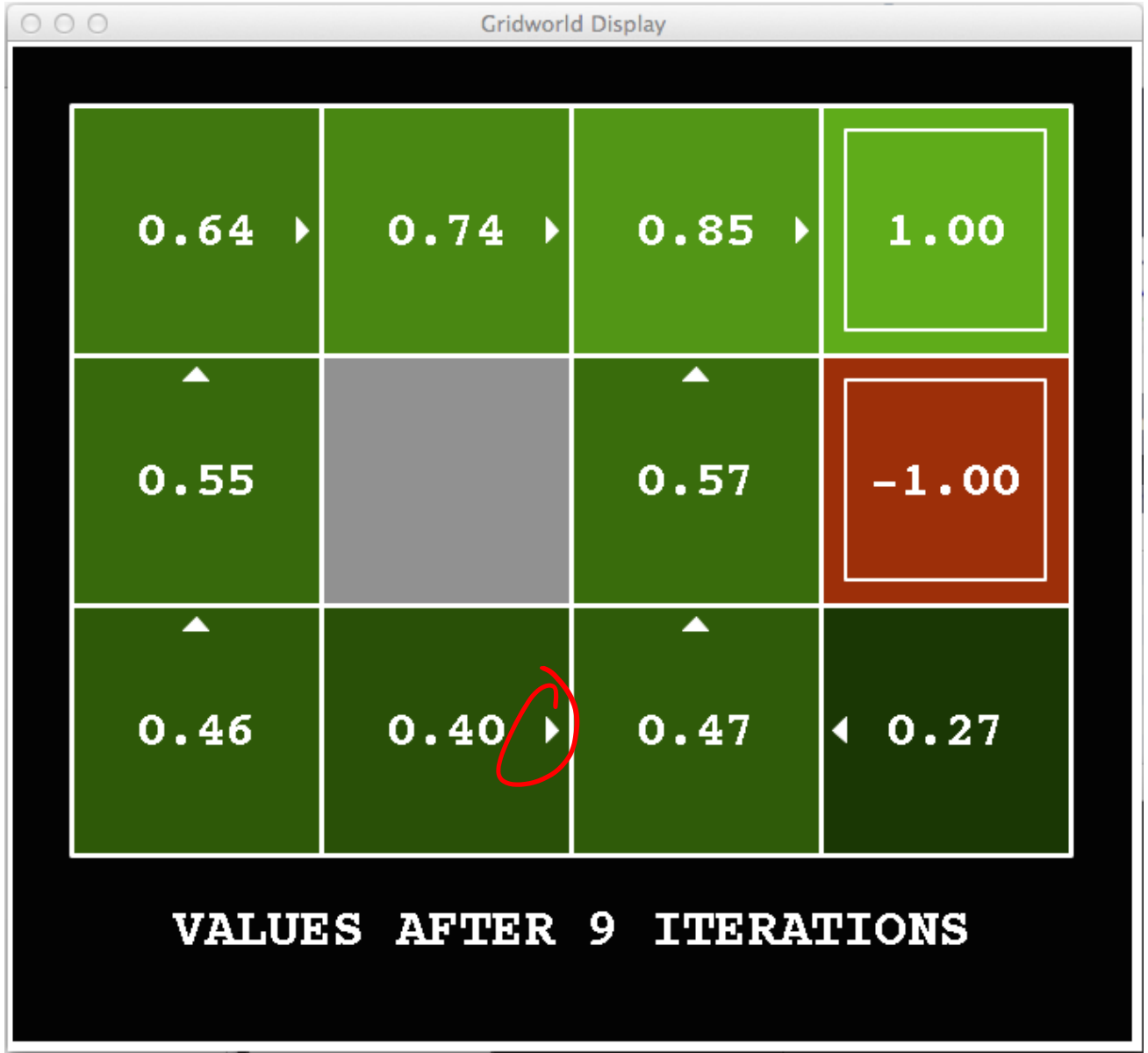
k=8



Noise = 0.2
Discount = 0.9
Living reward = 0

Slide credit: UC Berkeley AI

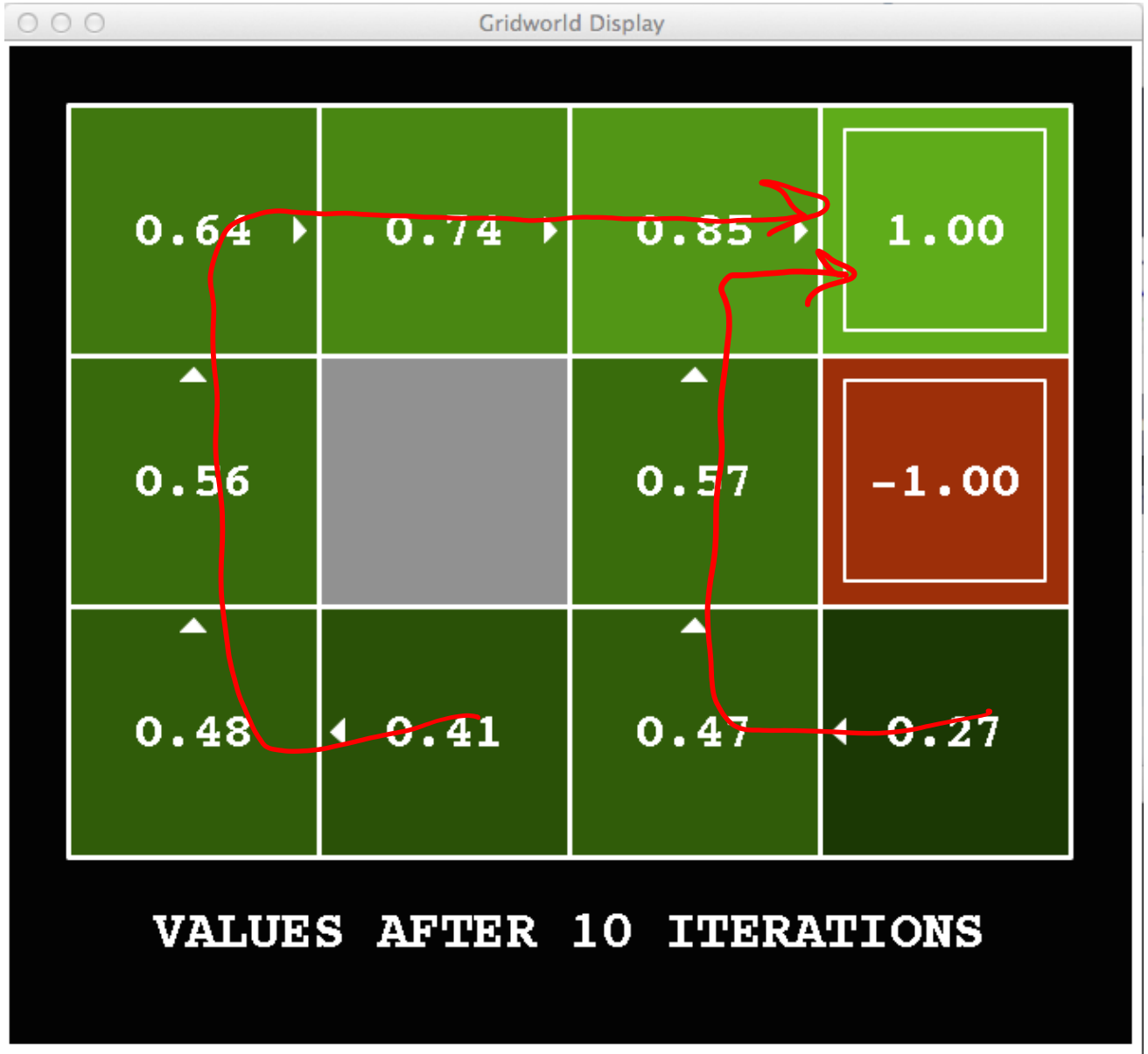
k=9



Noise = 0.2
Discount = 0.9
Living reward = 0

Slide credit: UC Berkeley AI

k=10



Noise = 0.2
Discount = 0.9
Living reward = 0

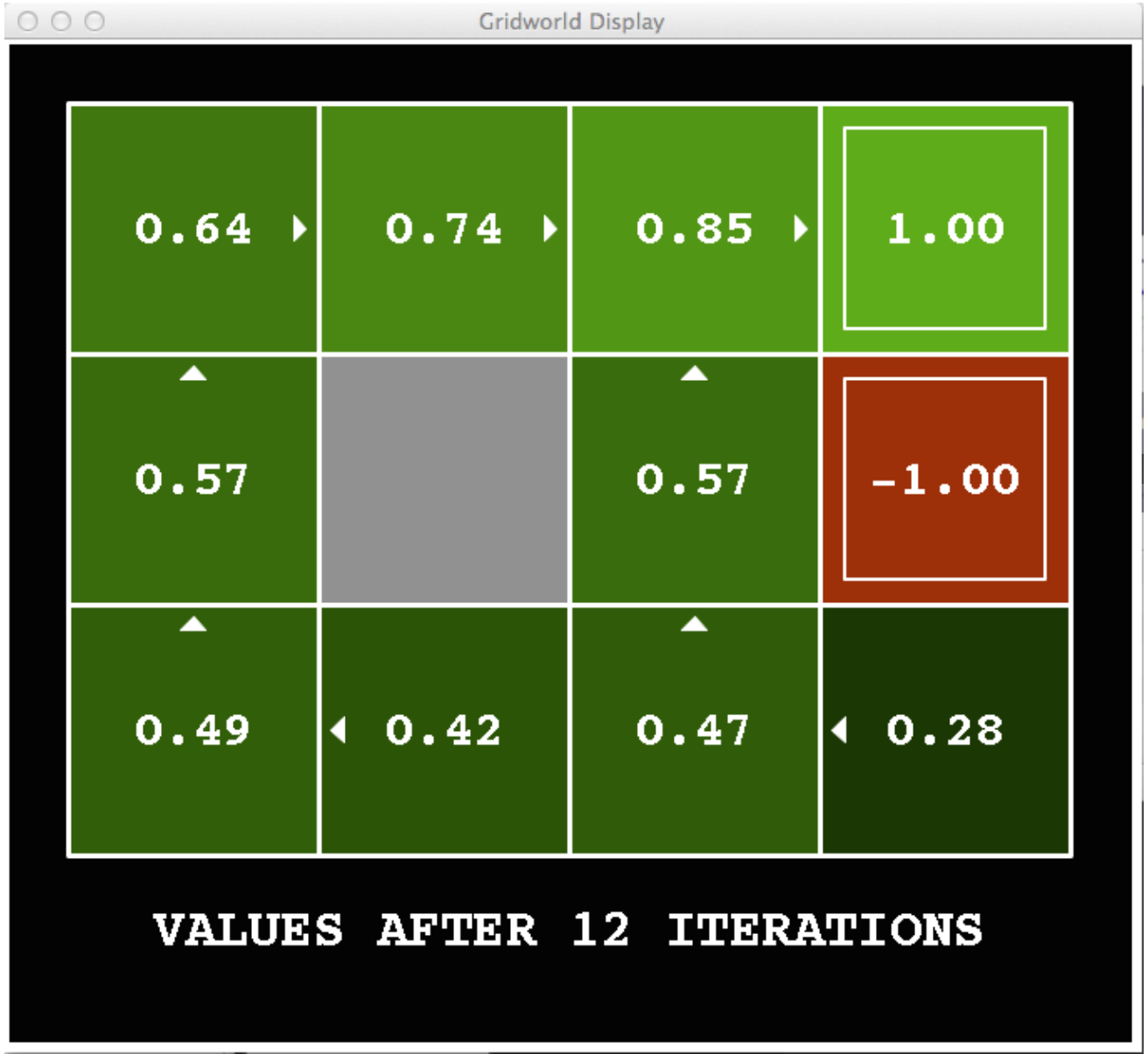
Slide credit: UC Berkeley AI

k=11

Noise = 0.2
Discount = 0.9
Living reward = 0

Slide credit: UC Berkeley AI

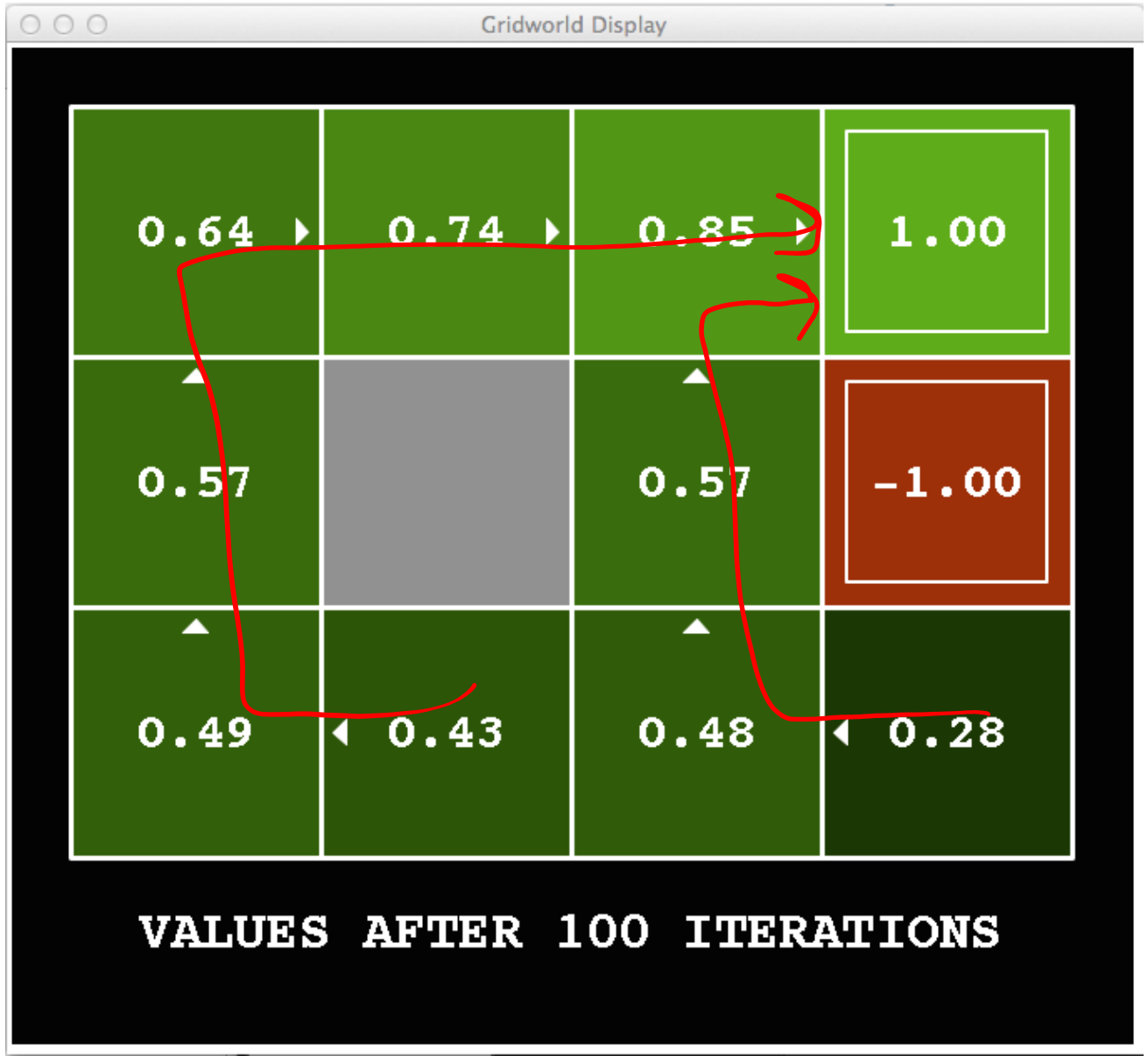
k=12



Noise = 0.2
Discount = 0.9
Living reward = 0

Slide credit: UC Berkeley AI

k=100



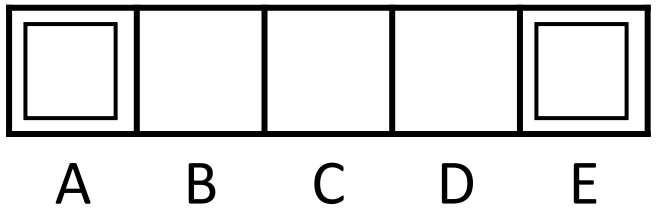
Noise = 0.2
Discount = 0.9
Living reward = 0

Slide credit: UC Berkeley AI

Outline

Pre-reading: MDP Setup

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

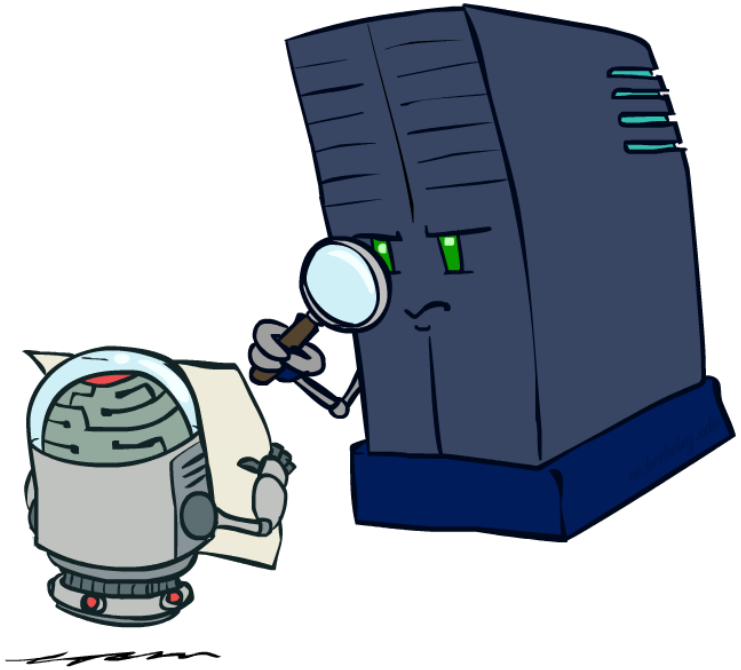


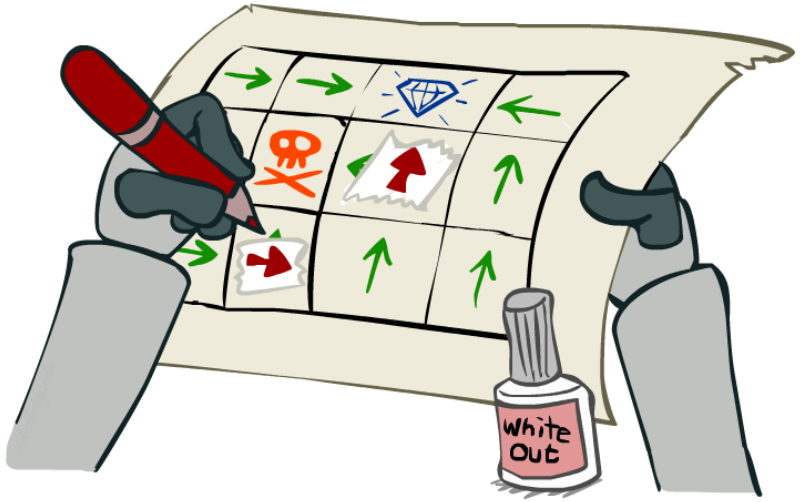
Discounting, γ 

Solving MDPs

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

Image credit: *Ketrina Yim, UC Berkeley*





Policy Iteration

Image credit: *Ketrina Yim, UC Berkeley*

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

π compute $V^{\pi}(s)$

fixed π

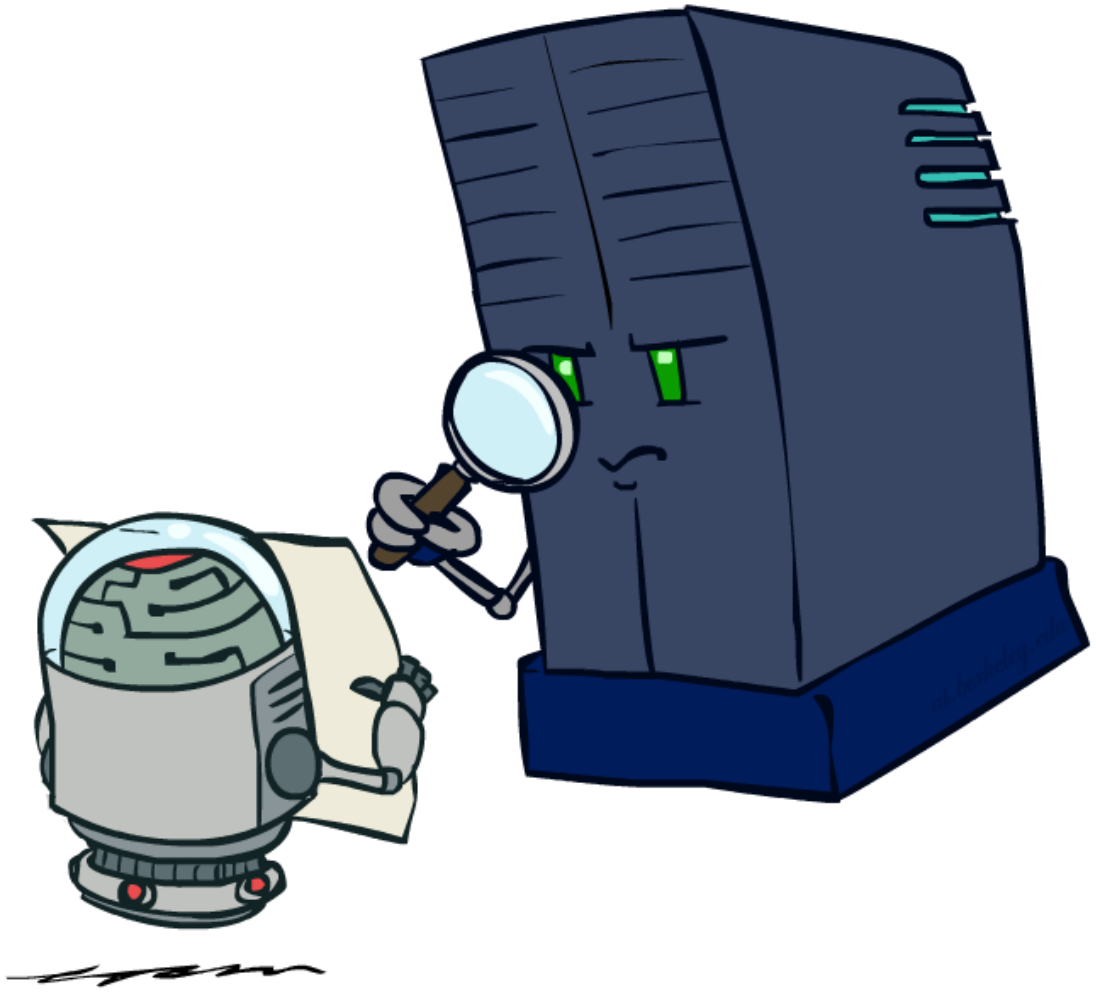
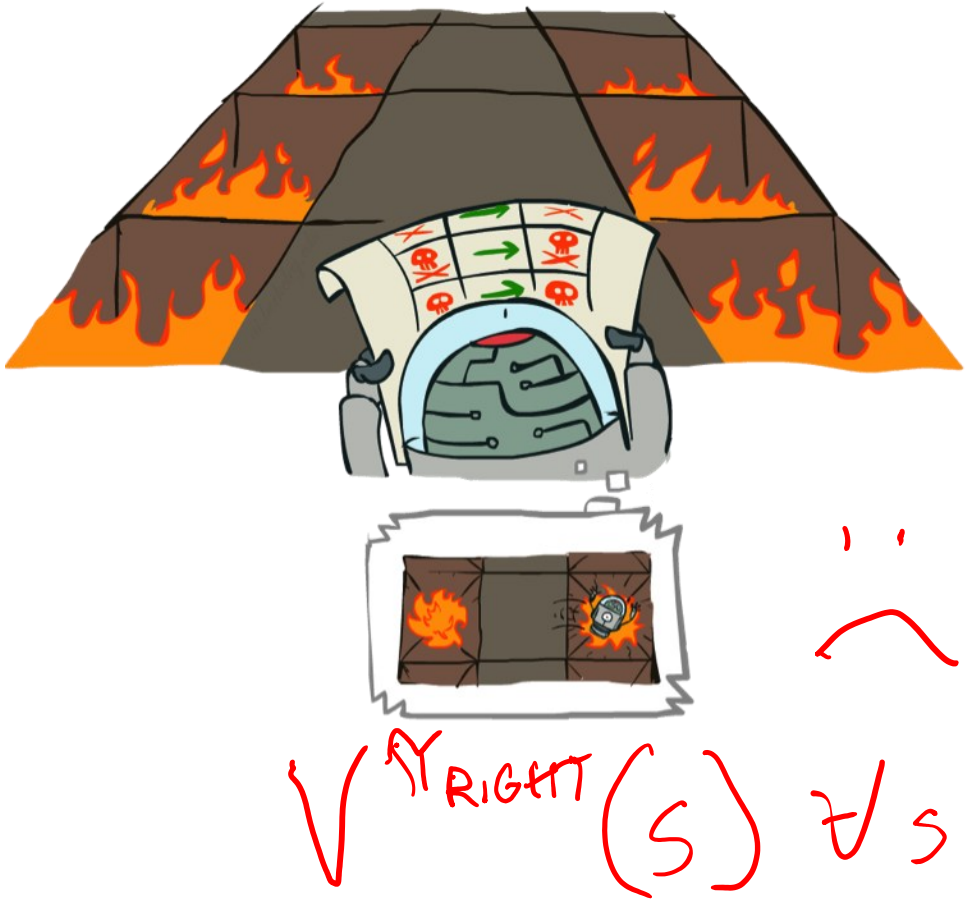


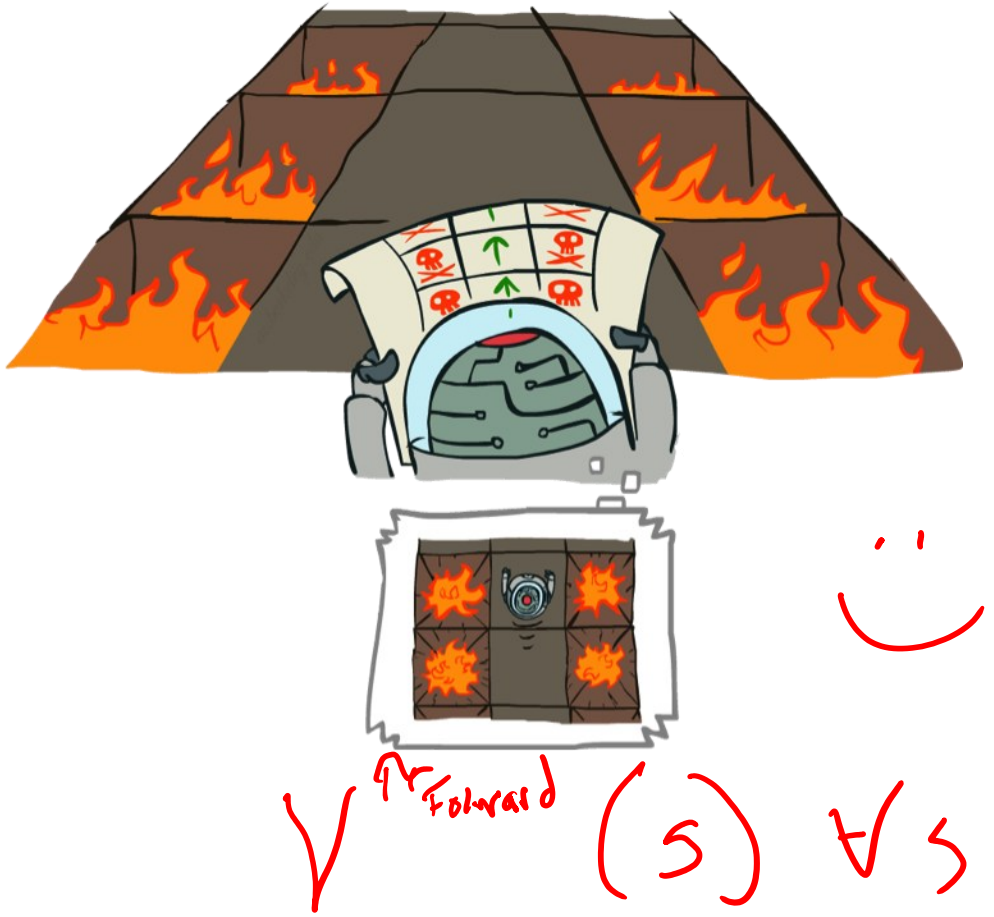
Image credit: *Ketrina Yim, UC Berkeley*

Example: Policy Evaluation

Always Go Right



Always Go Forward



Slide credit: UC Berkeley AI

Example: Policy Evaluation

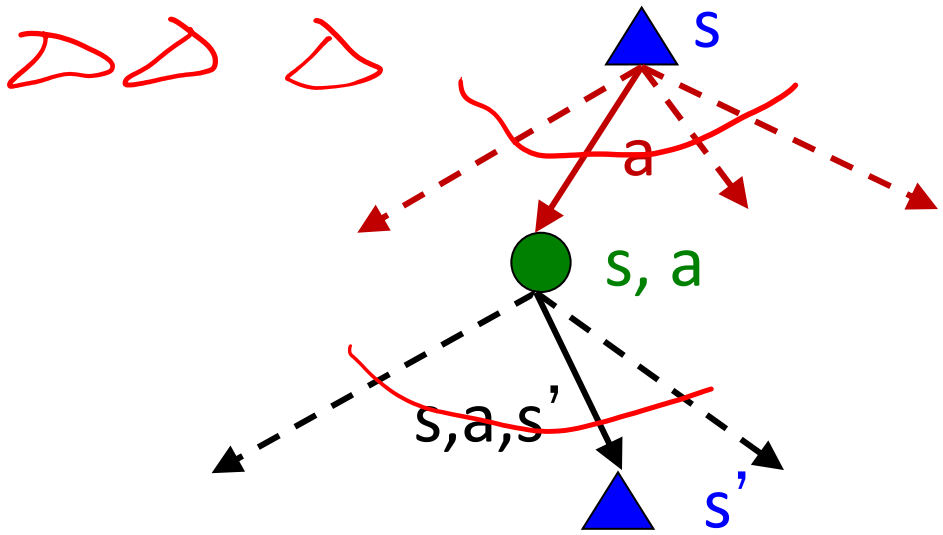
Always Go Right

Always Go Forward

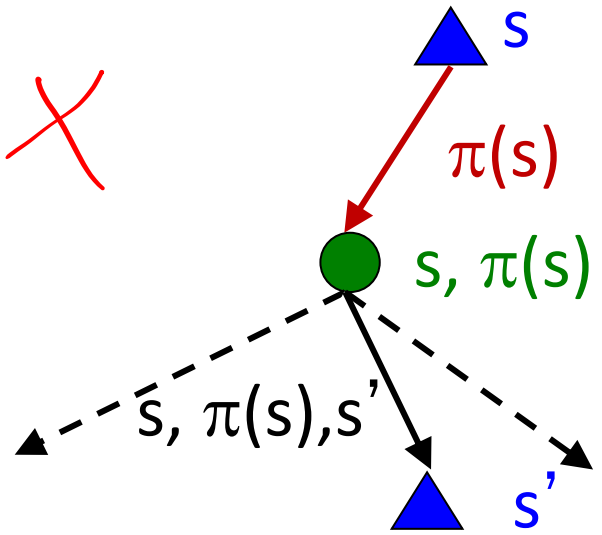
Slide credit: UC Berkeley AI

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

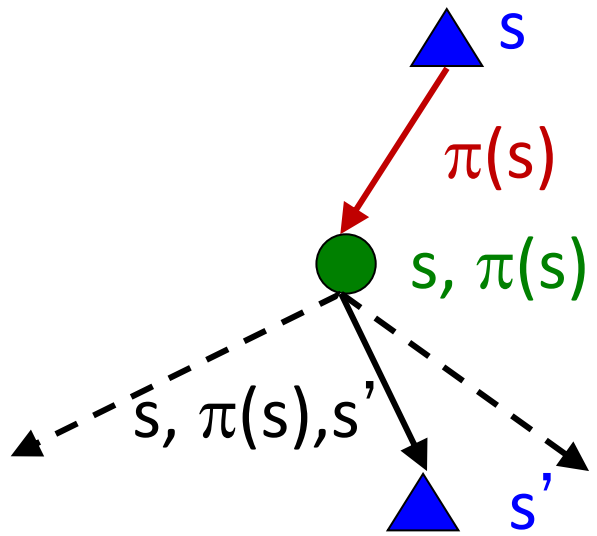
If we fixed some policy $\pi(s)$, then the tree would be simpler – only one action per state

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

Slide credit: UC Berkeley AI

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

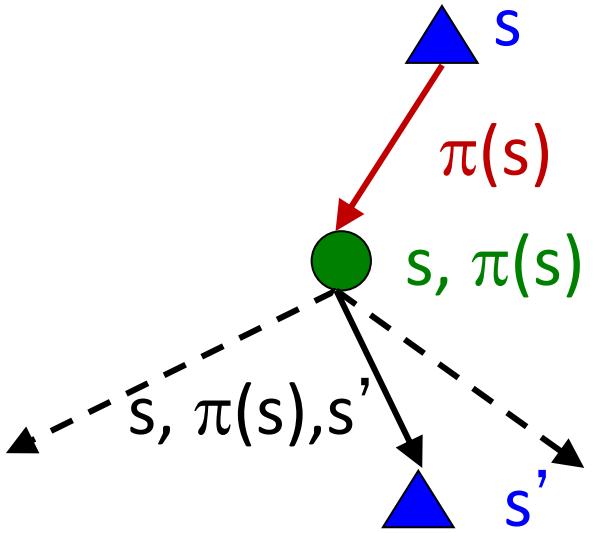
Recursive relation (one-step look-ahead / Bellman equation):

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

Slide credit: UC Berkeley AI

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

Slide credit: UC Berkeley AI

Policy Improvement

$\pi_k \rightarrow \pi_{k+1}$

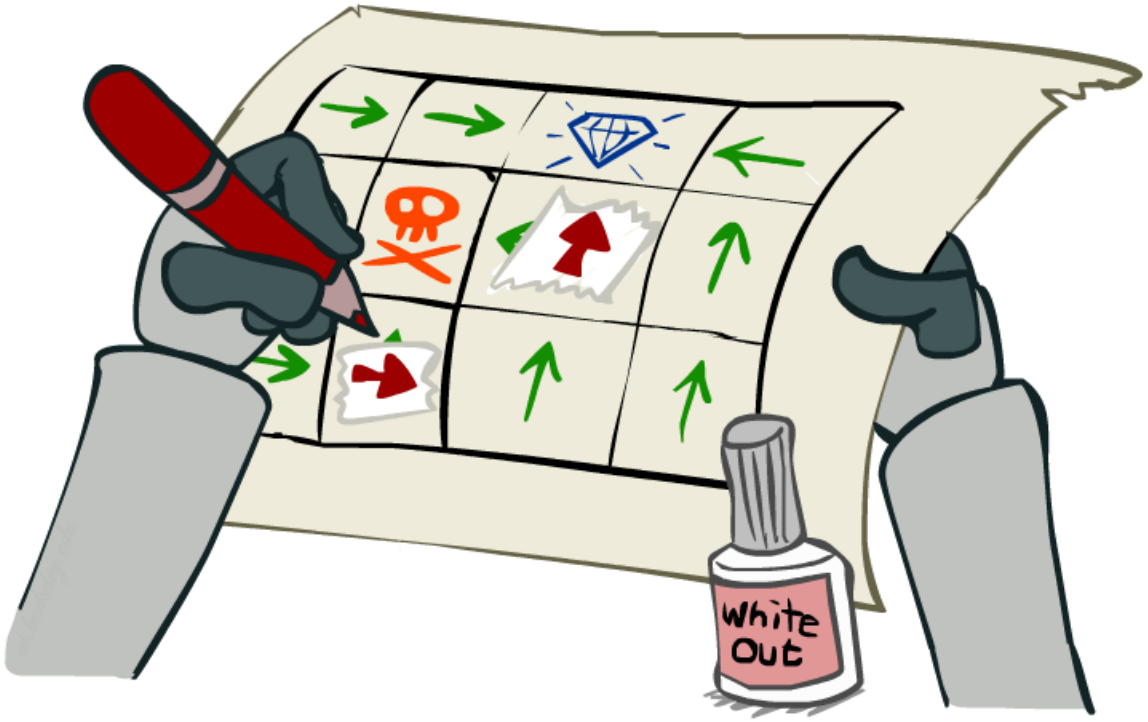


Image credit: *Ketrina Yim, UC Berkeley*

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

Slide credit: UC Berkeley AI

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:

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

Policy iteration

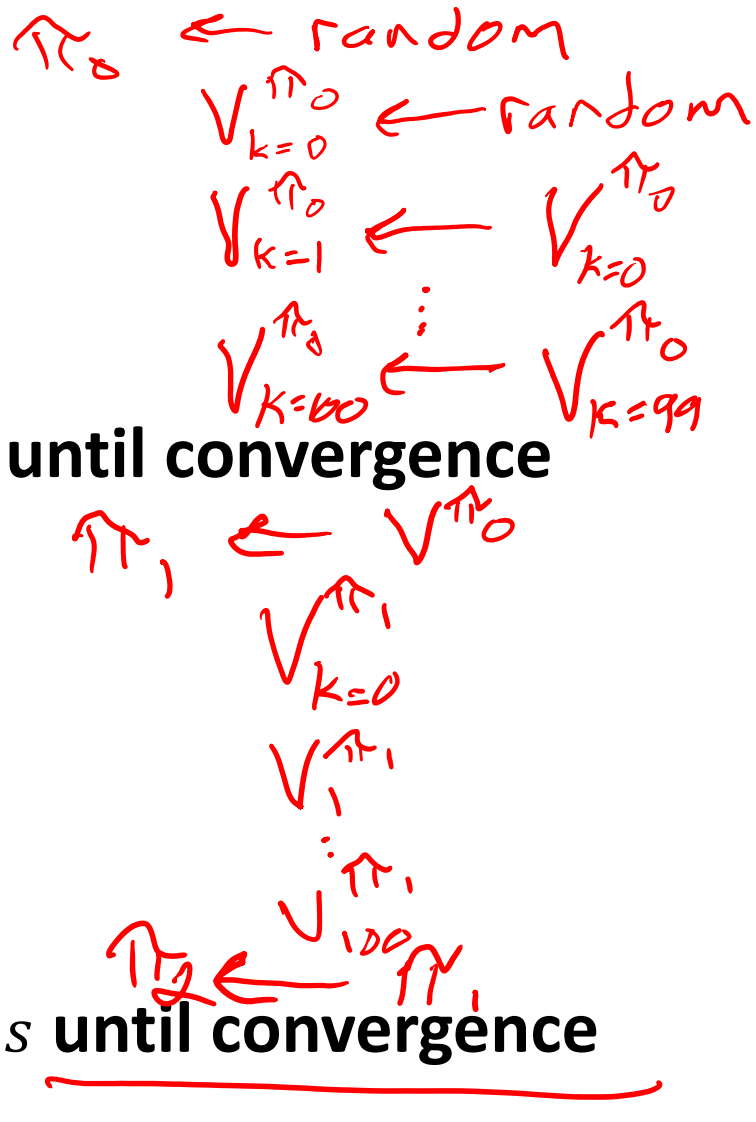
- Step 1: Policy evaluation:

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

- Step 2: Policy improvement:

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

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

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

Policy iteration
1) Policy evaluation
2) Policy improvement

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

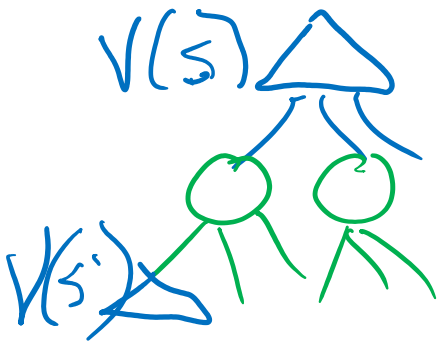
Q-iteration: $Q_{k+1}(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')], \quad \forall\, s,a$

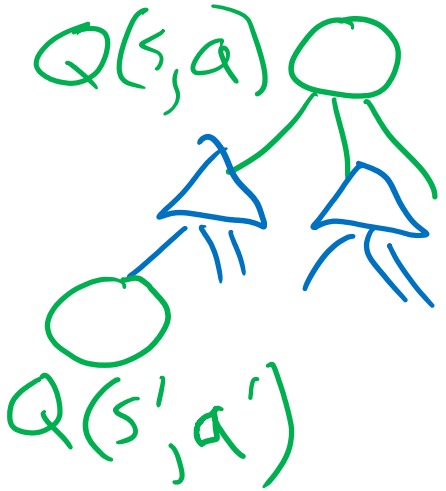
Policy extraction: $\pi_V(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')], \quad \forall\, s$

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

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