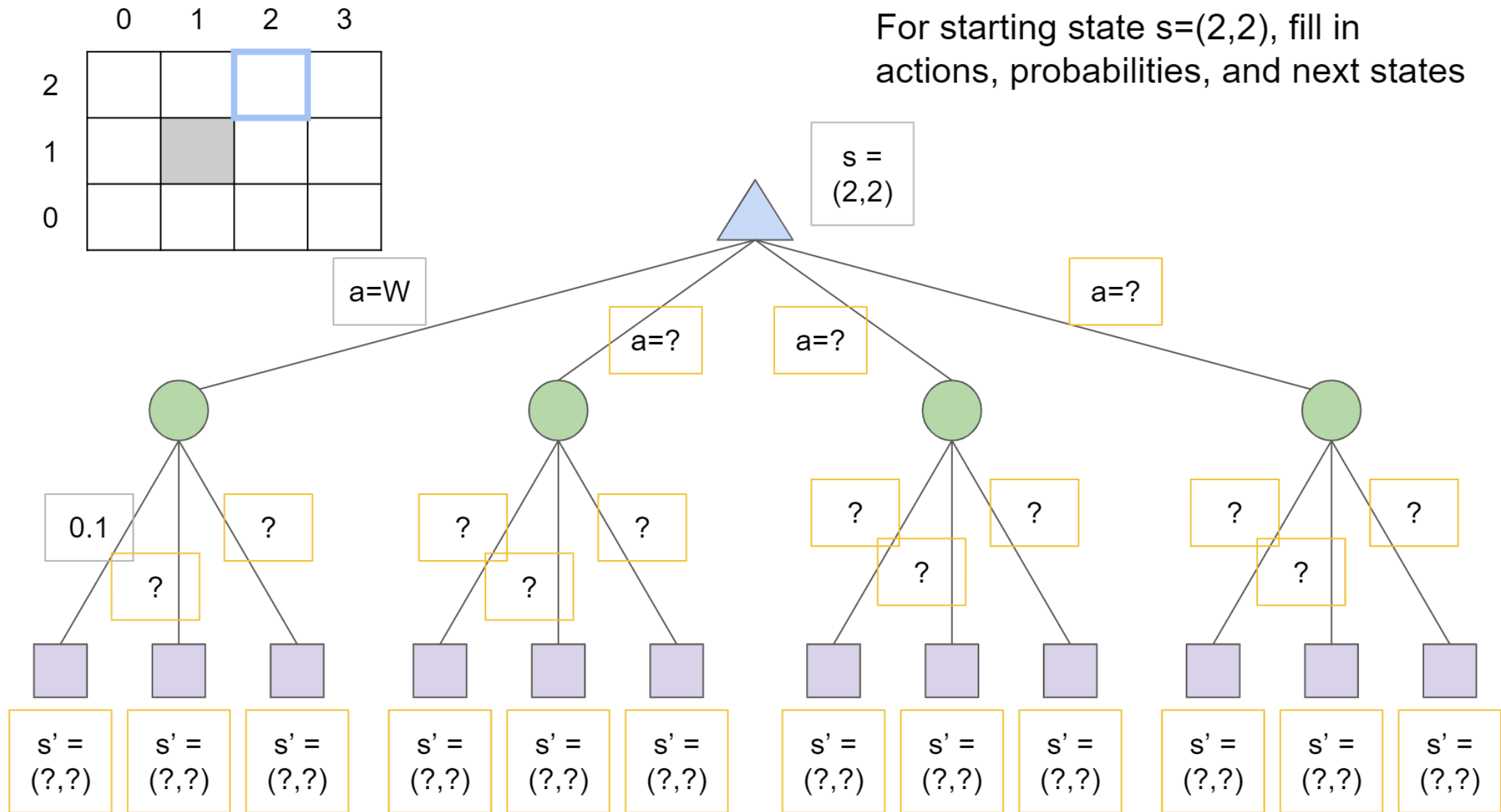
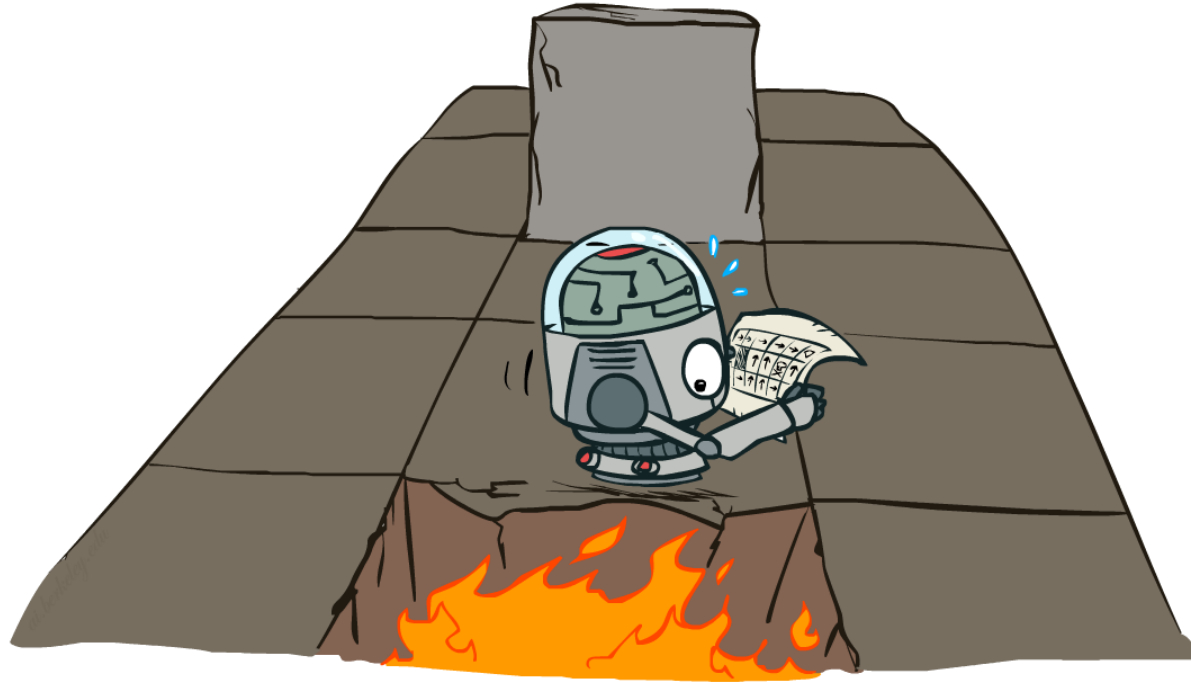


Warm-up as you walk in: Grid World



AI: Representation and Problem Solving

Markov Decision Processes II



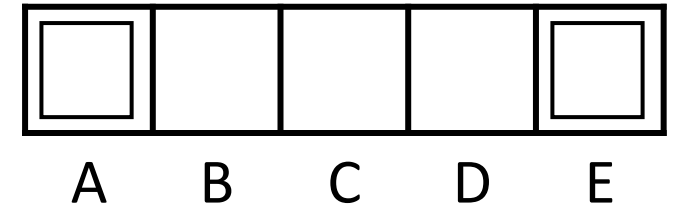
Instructor: Pat Virtue

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

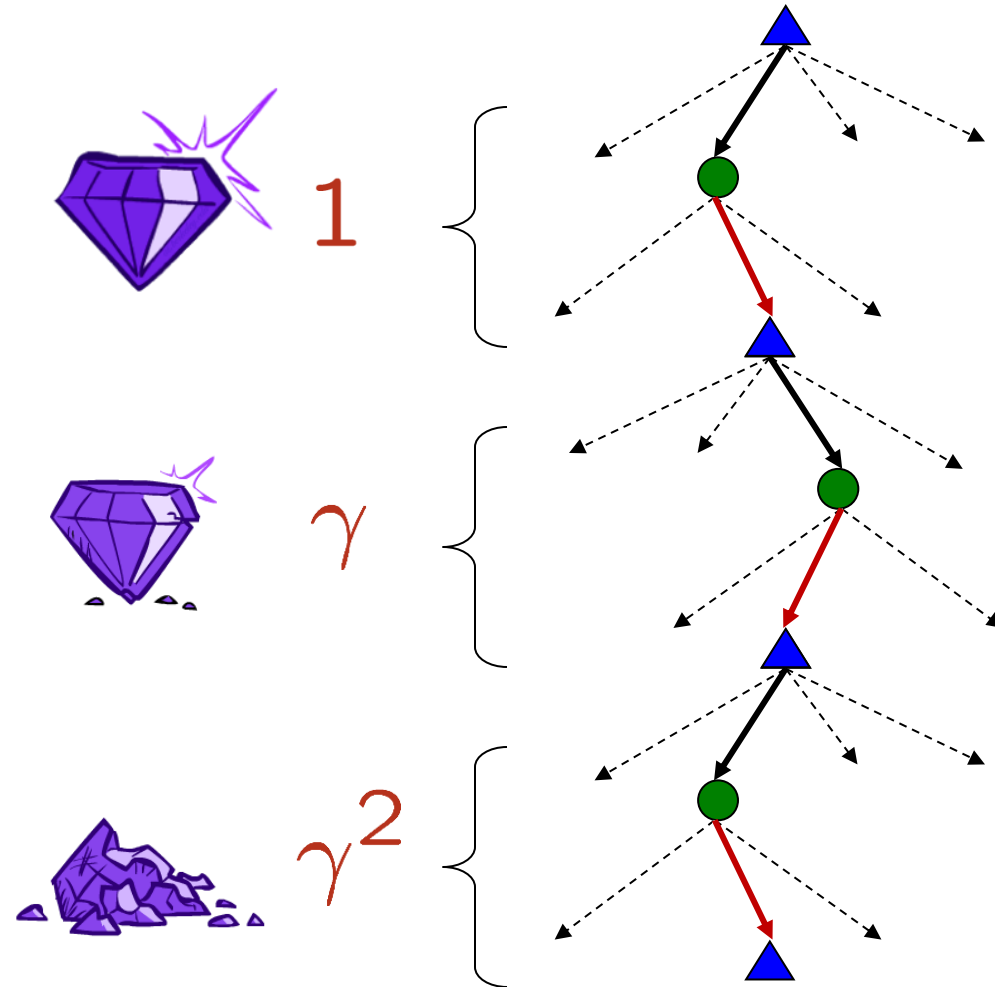
Discounting

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 $0 < \gamma < 1$



Recursive Expectimax

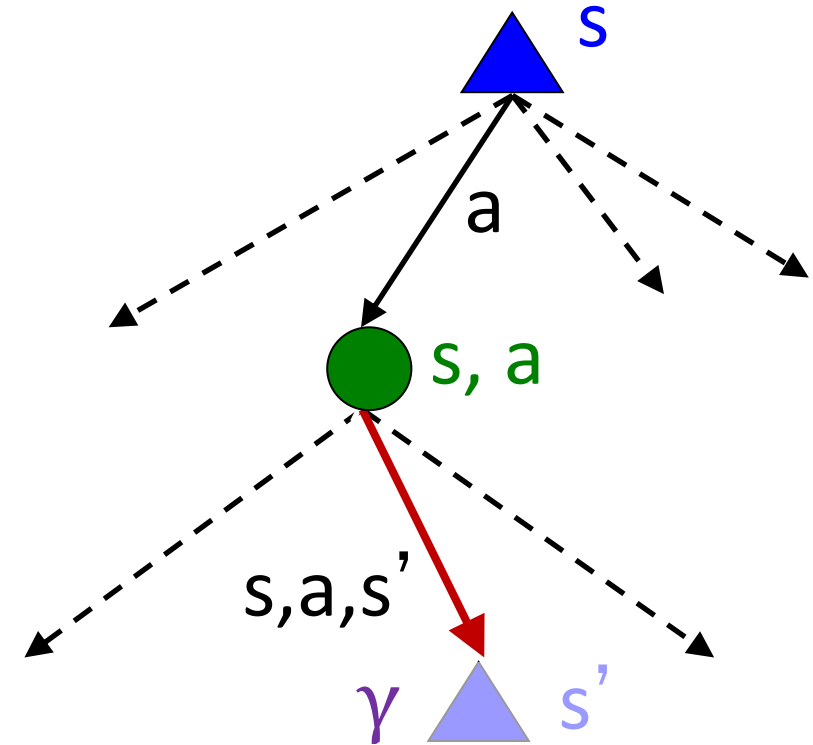
$$V(s) = \max_a \sum_{s'} P(s'|s, a) V(s')$$

With rewards:

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

With discount factor:

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



Discounting

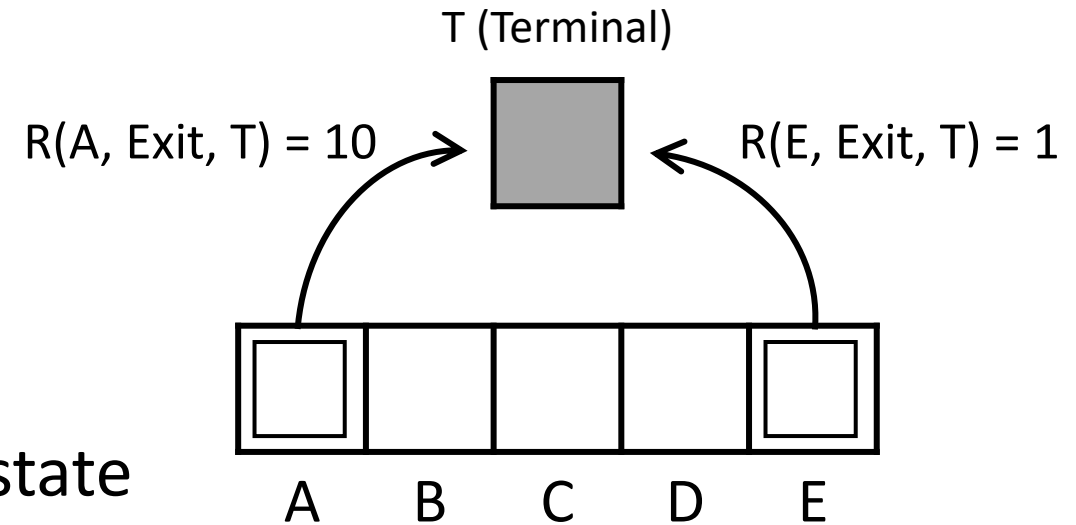
- Actions: B, C, D: East, West
- Actions: A, E: Exit
- Transitions: deterministic
- Rewards only for transitioning to terminal state

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

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

For $\gamma = 0.1$, what is the optimal policy?

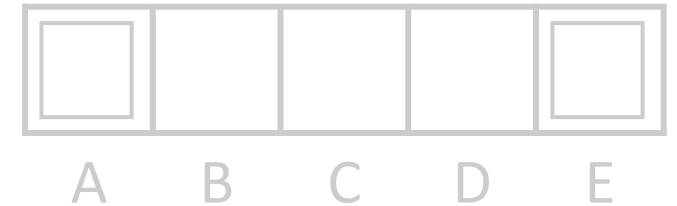
For which γ are West and East equally good when in state D?



Outline

MDP Setup

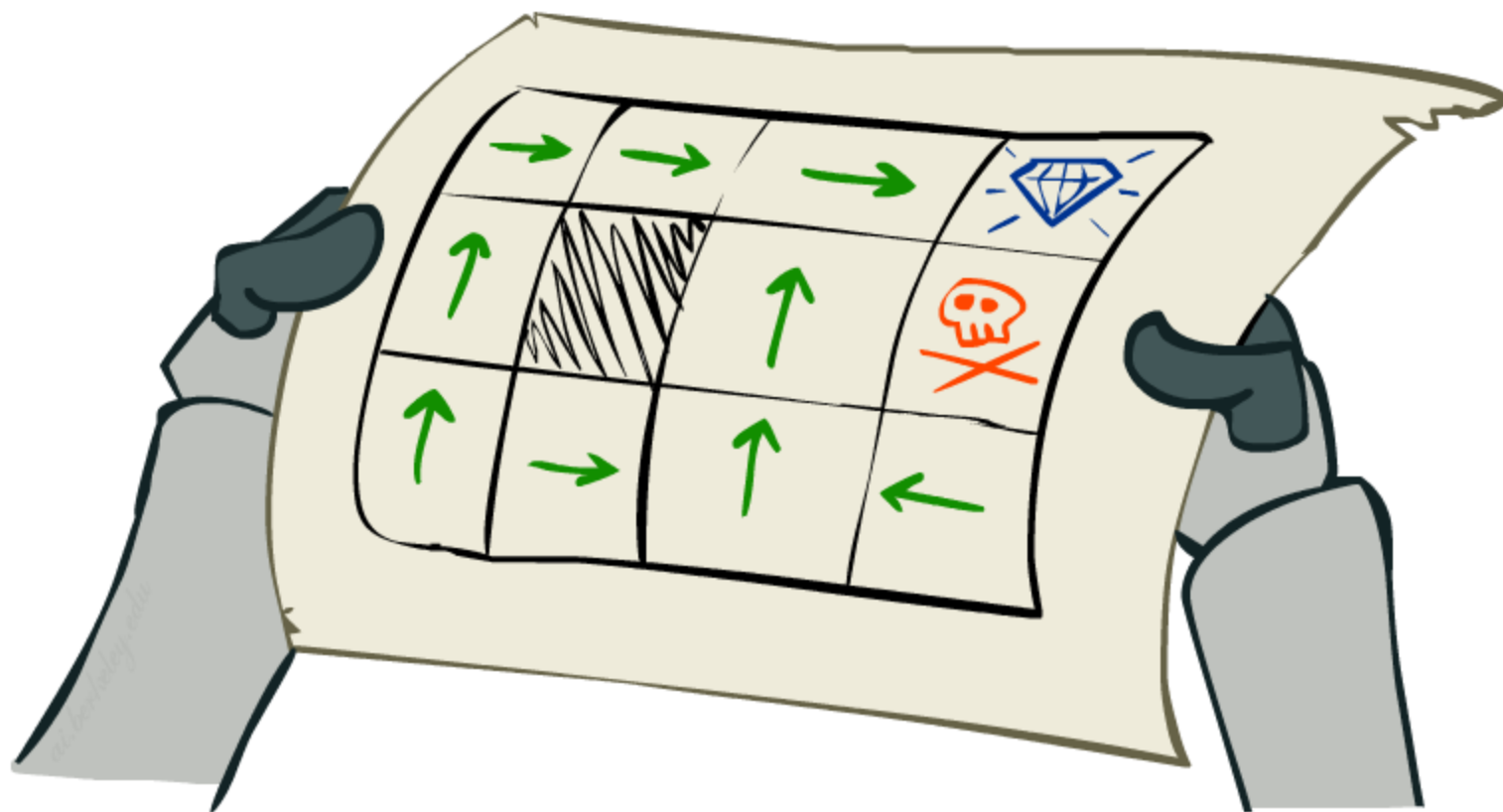
- Expectimax: State, actions, non-deterministic transition functions
- Rewards
 - Walk-through of super-simple value iteration
- Discounting, γ



Solving MDPs

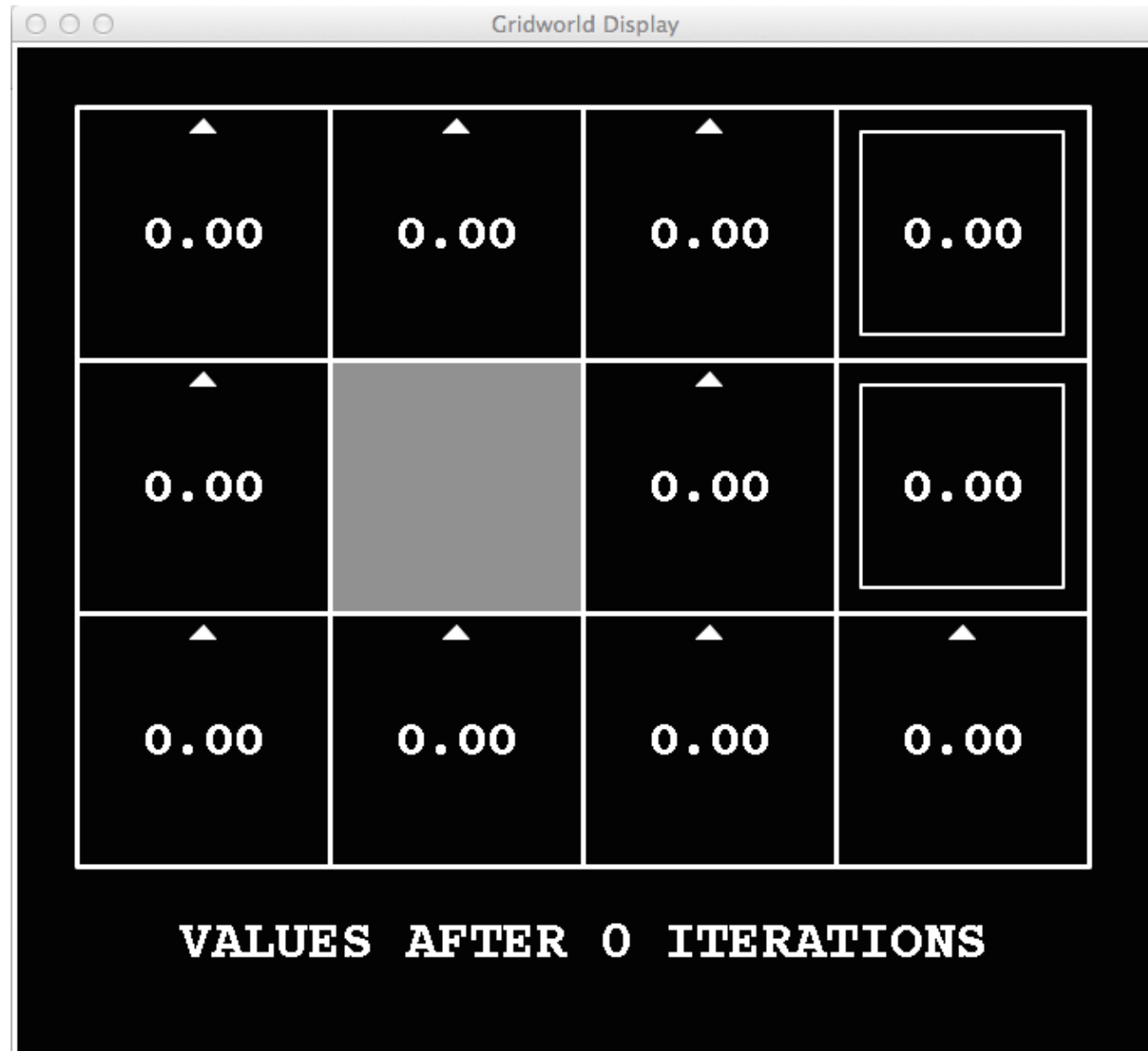
- Method 1) Value iteration
 - Value iteration convergence
- Bellman equations
- Policy Extraction
- Method 2) Policy Iteration

Solving MDPs



Value Iteration

$k=0$



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

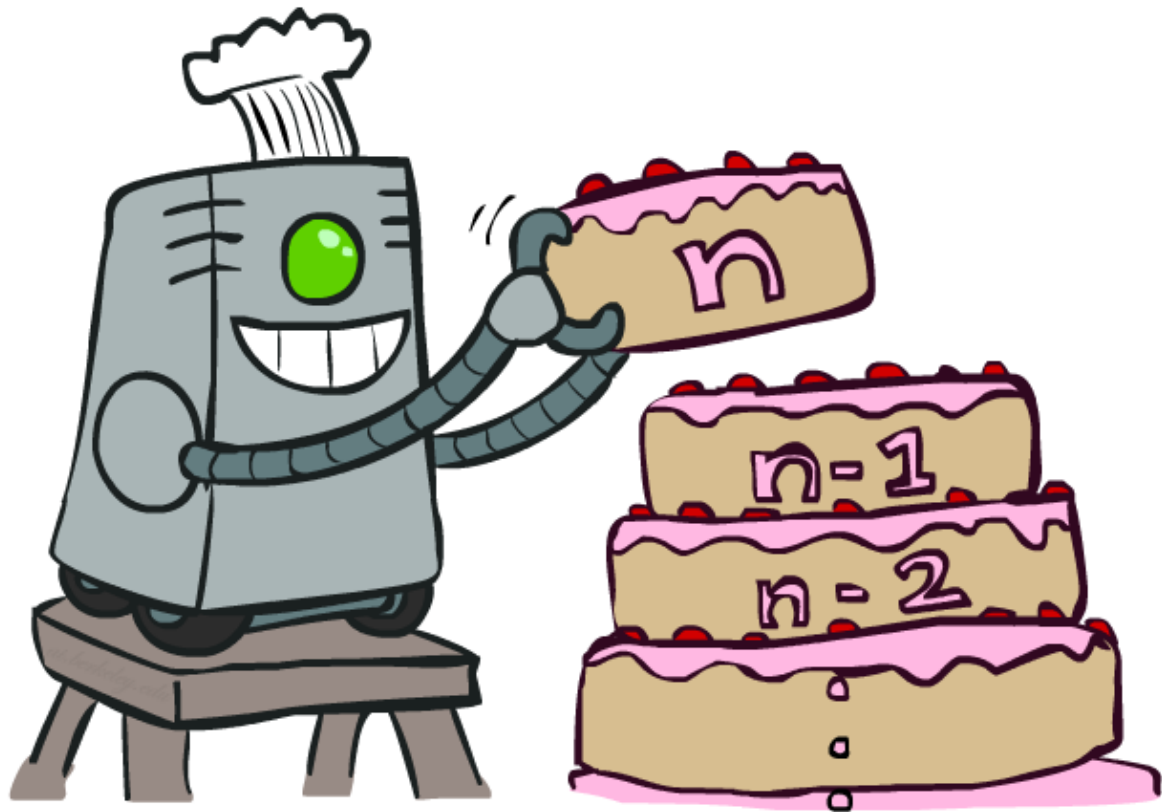


Noise = 0.2
Discount = 0.9
Living reward = 0

k=100



Noise = 0.2
Discount = 0.9
Living reward = 0



Value Iteration

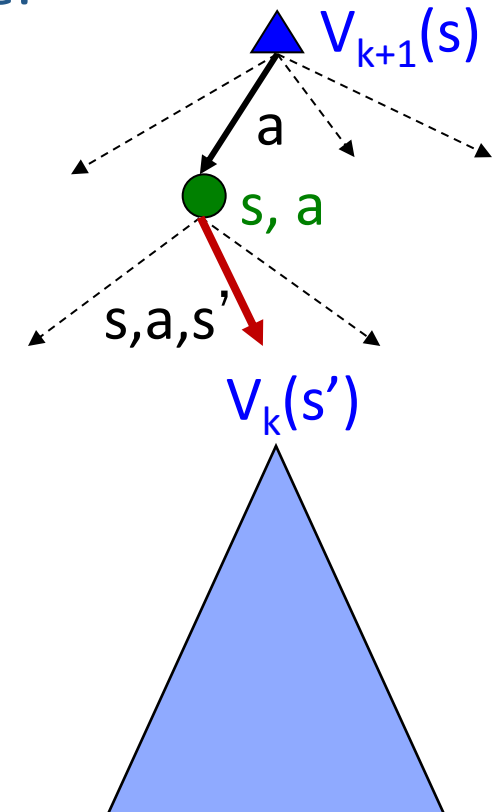
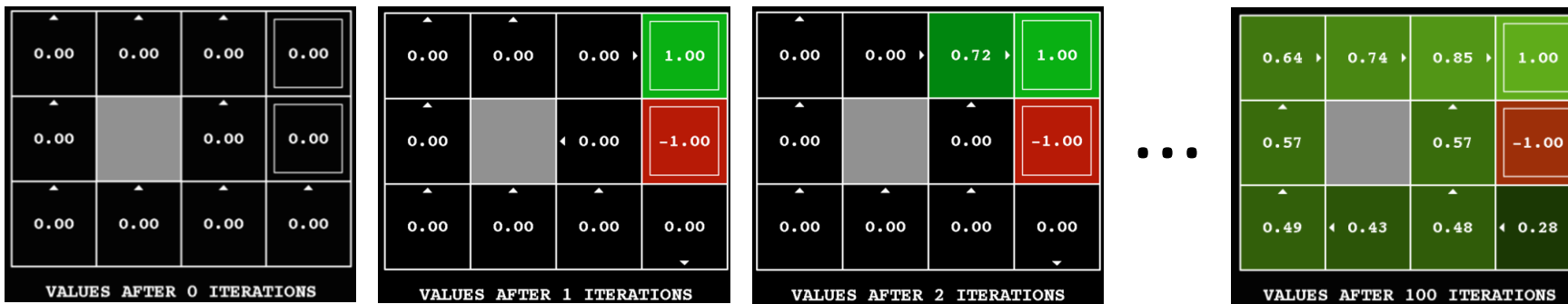
Value Iteration

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:

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

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

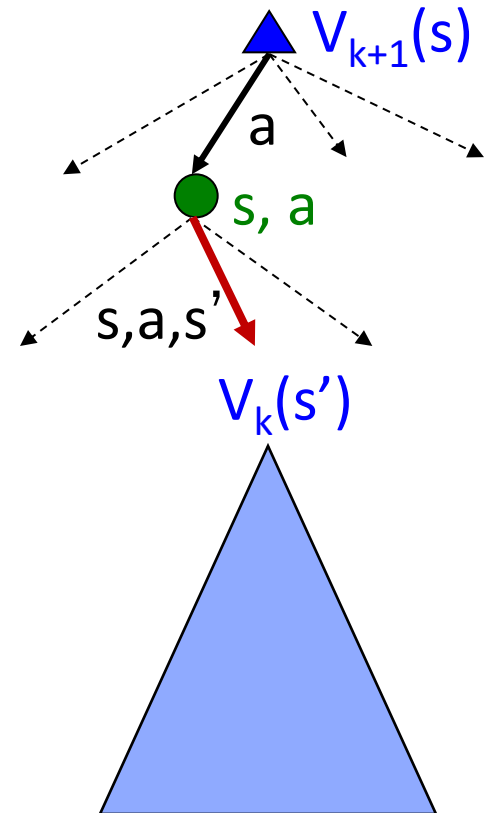
V: $O(|S|^2)$

0.00	0.00	0.00	1.00
0.00		0.00	-1.00
0.00	0.00	0.00	0.00

VALUES AFTER 1 ITERATIONS

0.00	0.00	0.72	1.00
0.00		0.00	-1.00
0.00	0.00	0.00	0.00

VALUES AFTER 2 ITERATIONS



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

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

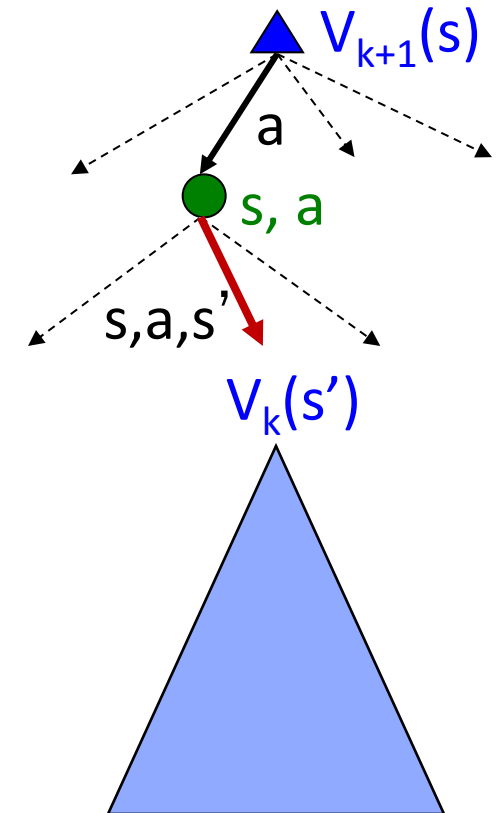
IV: $O(|S|^2|A|^2)$

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

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

0.00	0.00	0.72	1.00
0.00		0.00	-1.00
0.00	0.00	0.00	0.00

VALUES AFTER 2 ITERATIONS



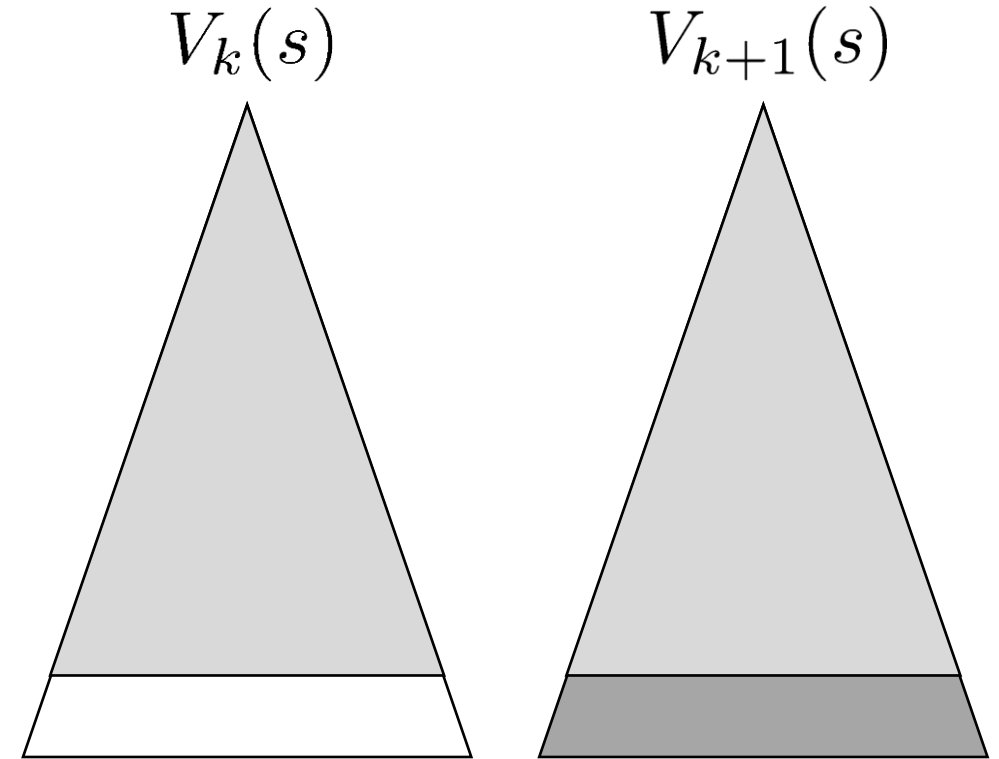
Value Iteration Convergence

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 $\gamma^k \max |R|$ different
- So as k increases, the values converge



Summary: 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:

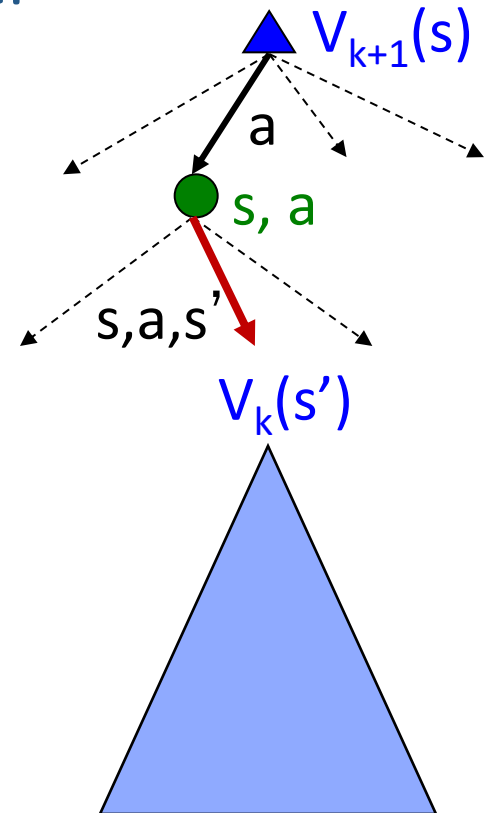
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_k(s')]$$

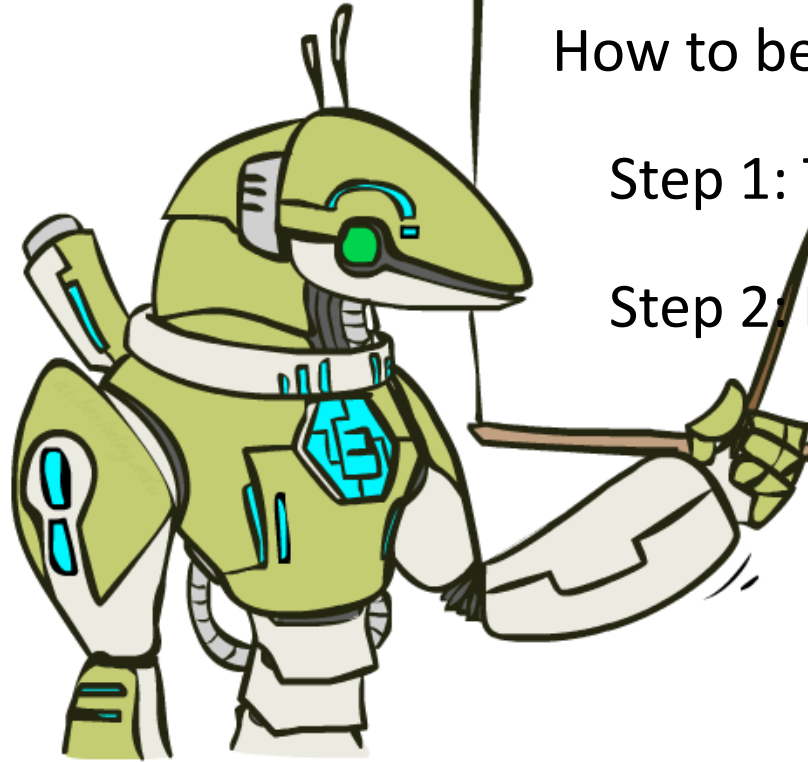
Repeat until convergence

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





How to be optimal:

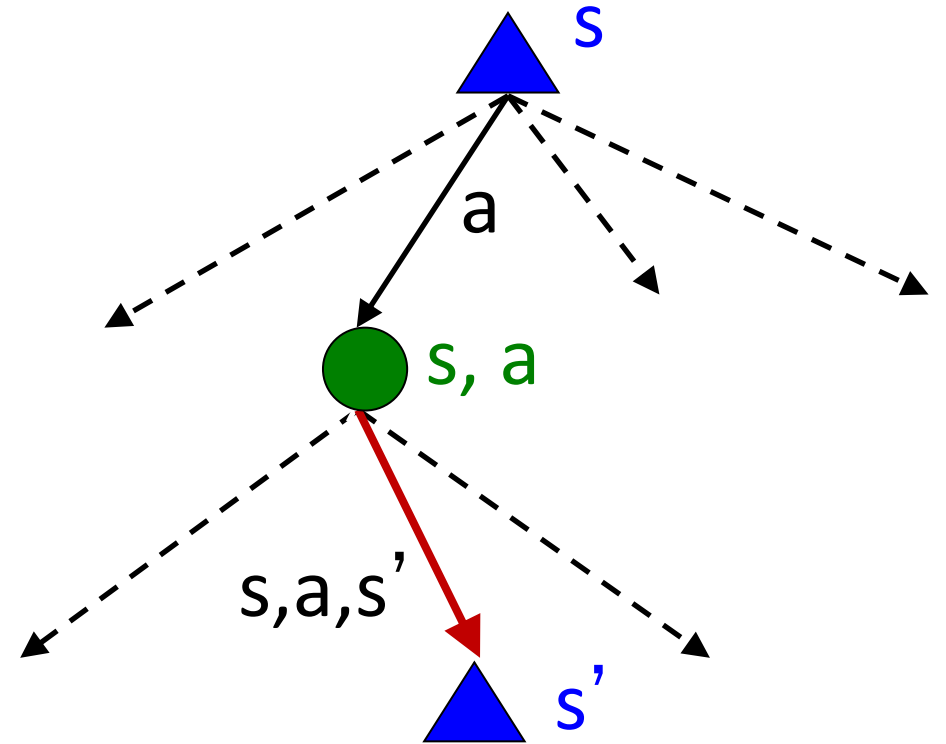
Step 1: Take correct first action

Step 2: Keep being optimal

Optimal Values & Bellman Equations

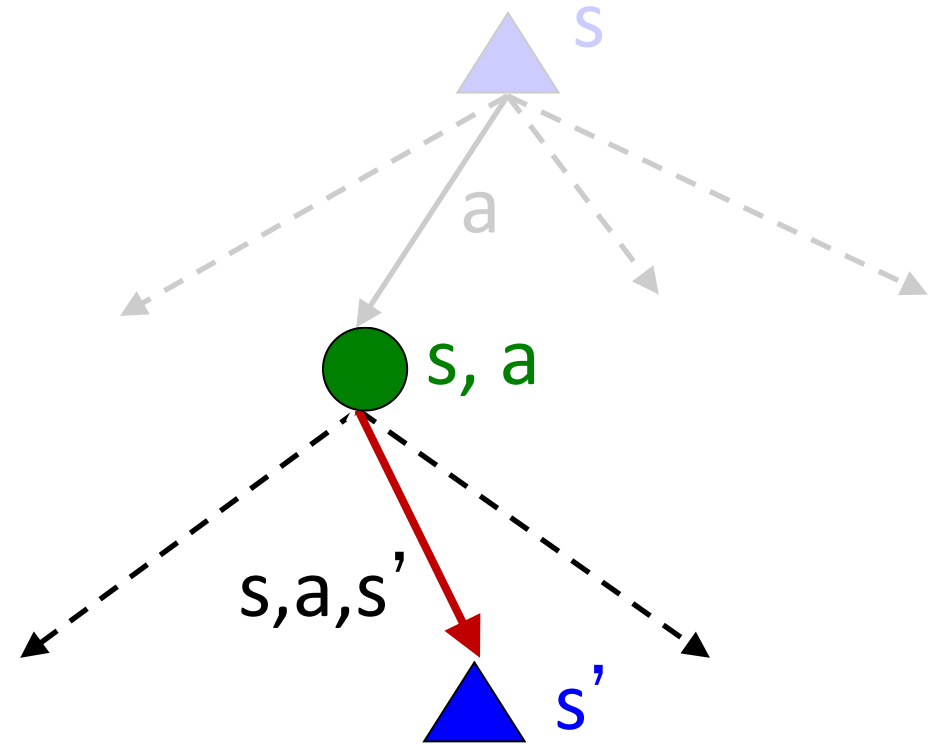
Optimal Quantities

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



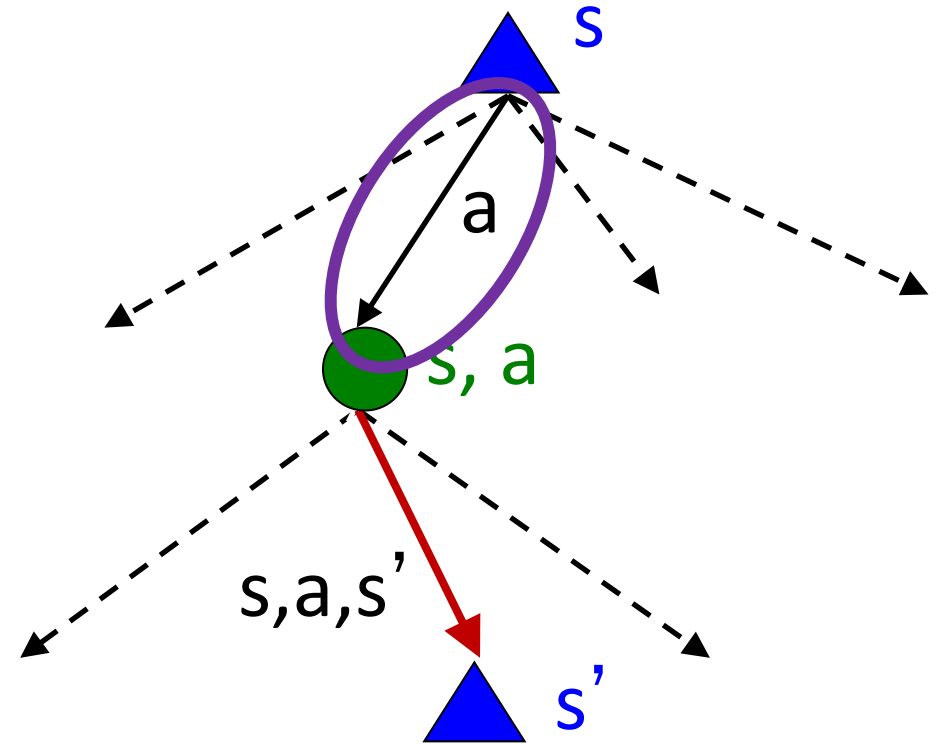
Optimal Quantities

- 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



Optimal Quantities

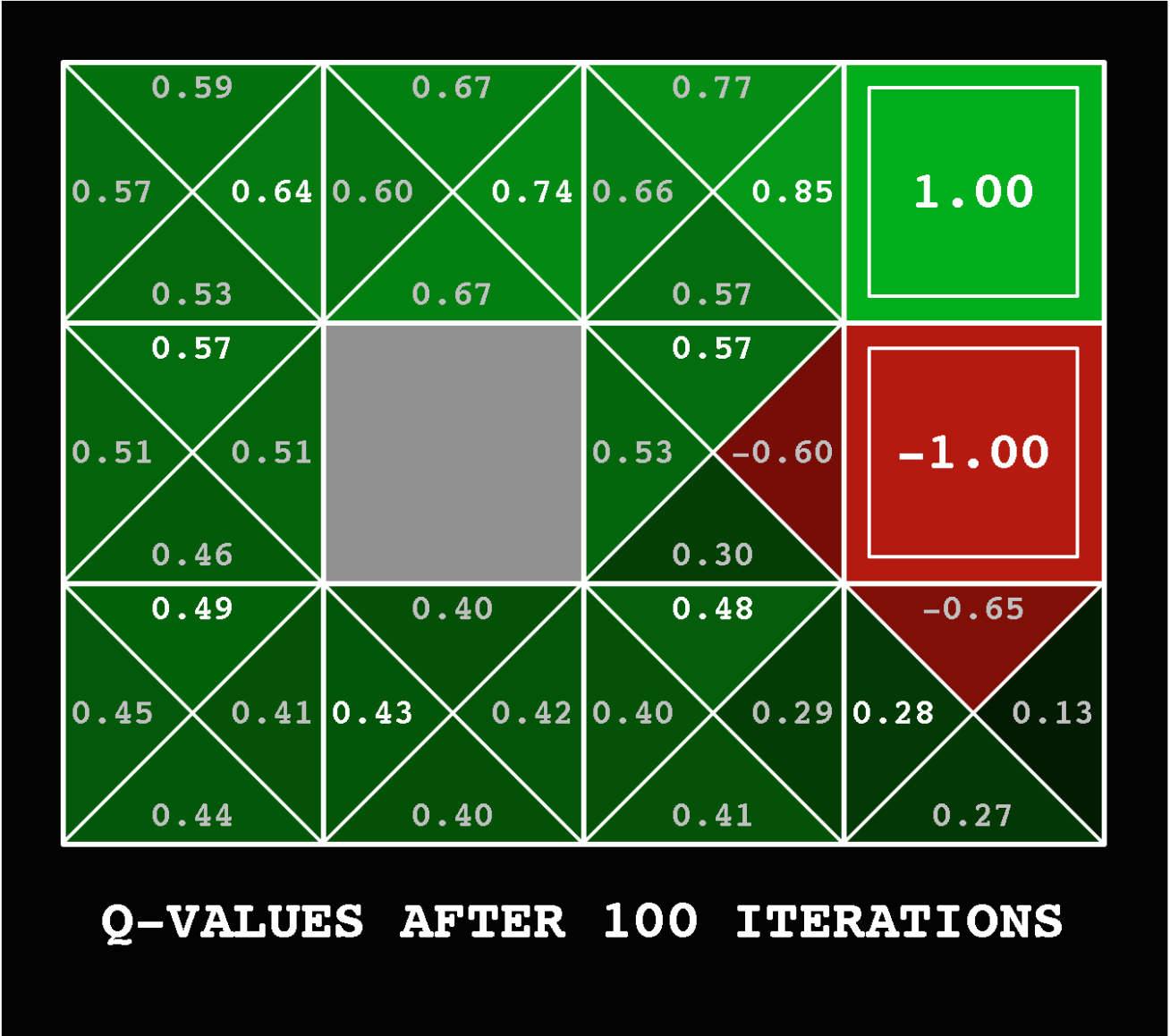
- The value (utility) of a state s :
 $V^*(s)$ = expected utility starting in s and acting optimally
- The value (utility) of a q-state (s,a) :
 $Q^*(s,a)$ = expected utility starting out having taken action a from state s and (thereafter) acting optimally
- The optimal policy:
 $\pi^*(s)$ = optimal action from state s



Gridworld Values V^*

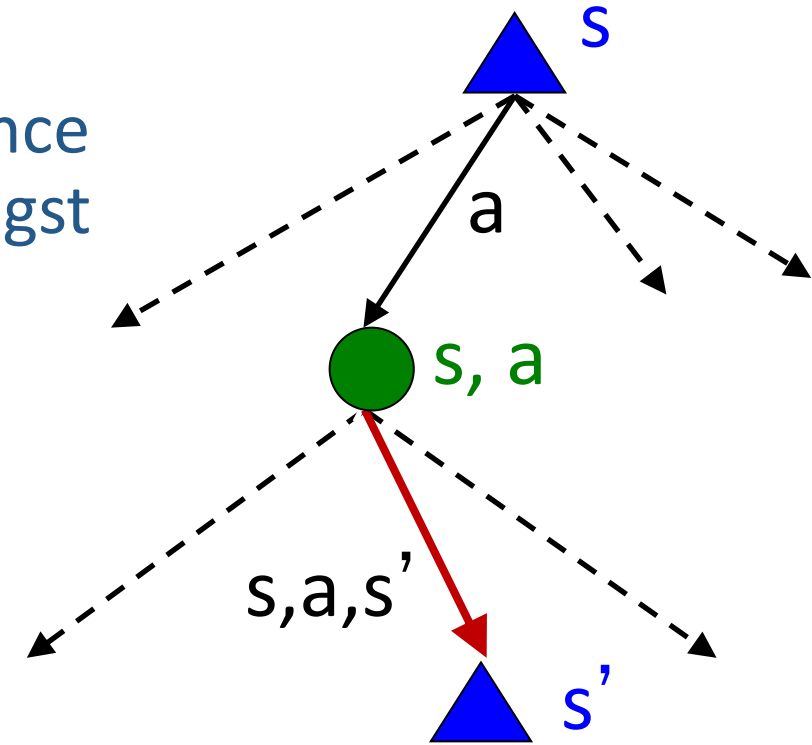


Gridworld: Q^*



The Bellman Equations

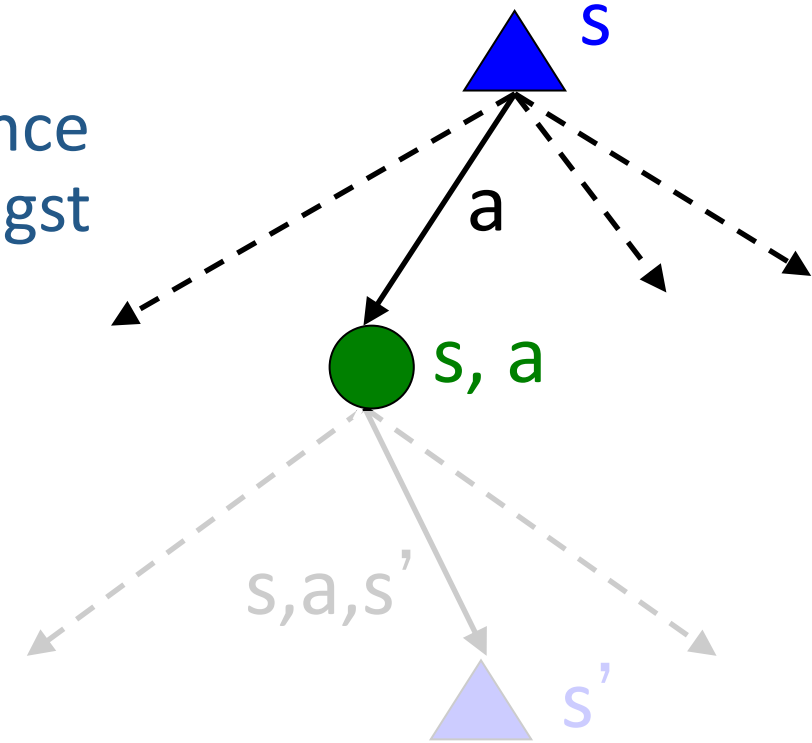
Definition of “optimal utility” via expectimax recurrence gives a simple one-step lookahead relationship amongst optimal utility values



The Bellman Equations

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

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

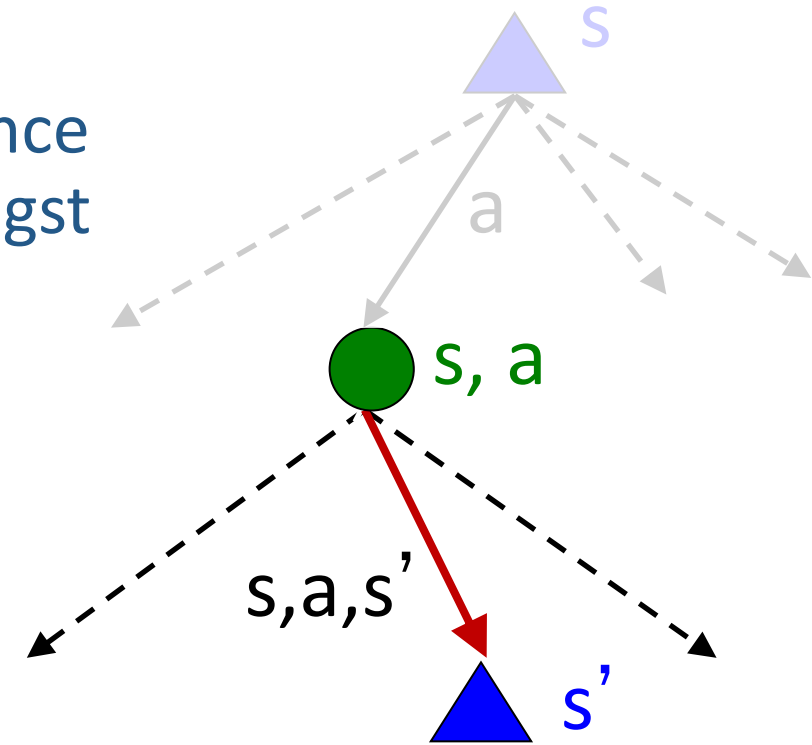


The Bellman Equations

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

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

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



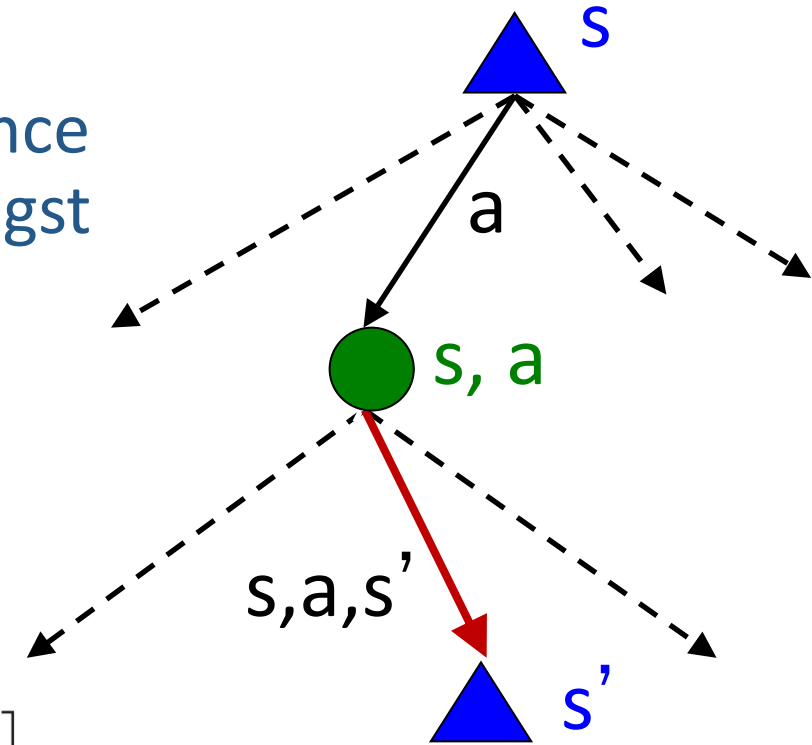
The Bellman Equations

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

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

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

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



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

Summary: Bellman Equations vs Value Iteration

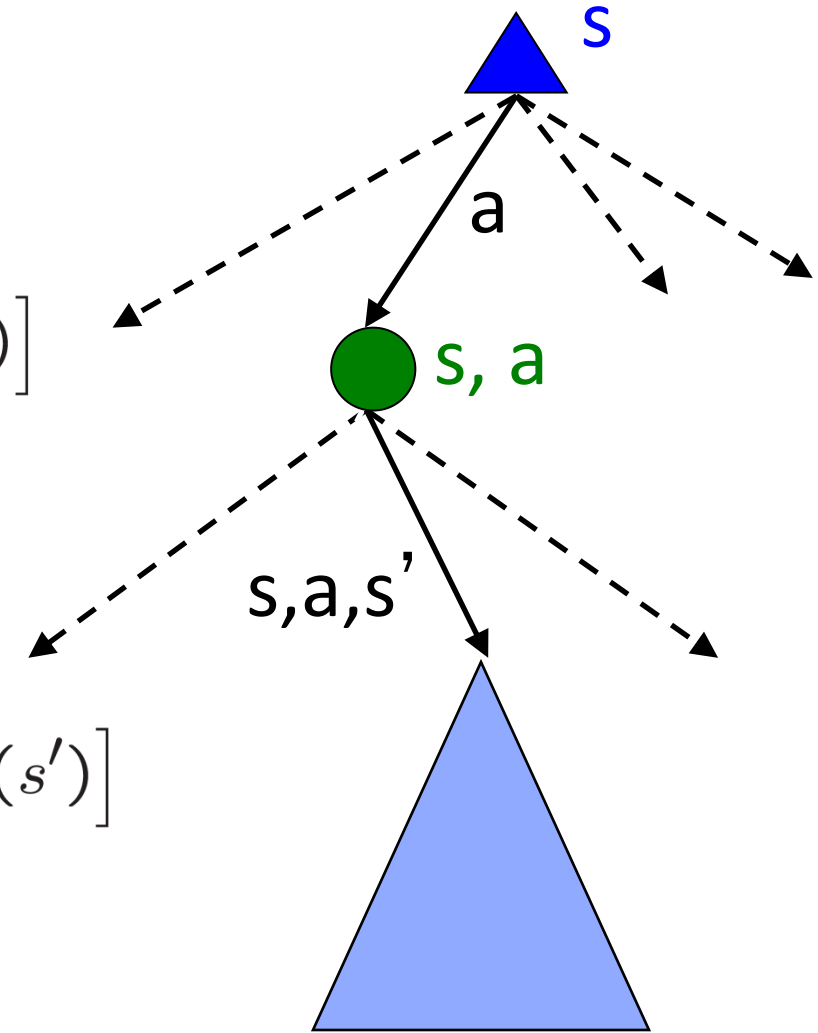
Bellman equations **characterize** the optimal values:

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

Value iteration **computes** them:

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

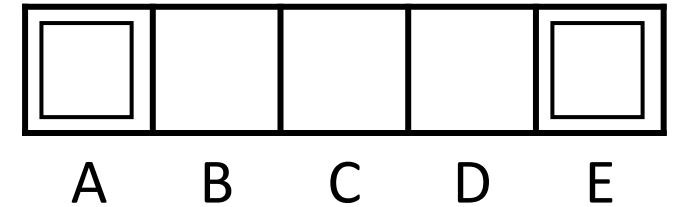
Value iteration is just a fixed point solution method



Outline

MDP Setup

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



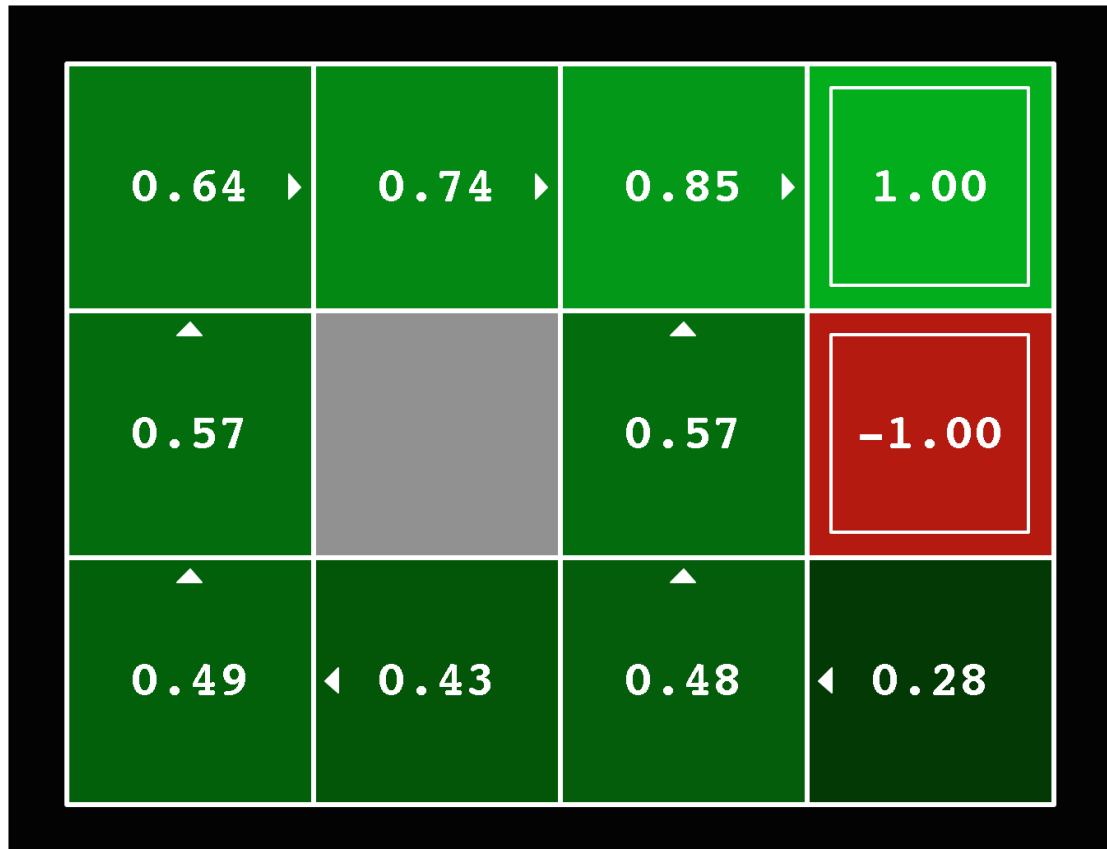
Solving MDPs

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

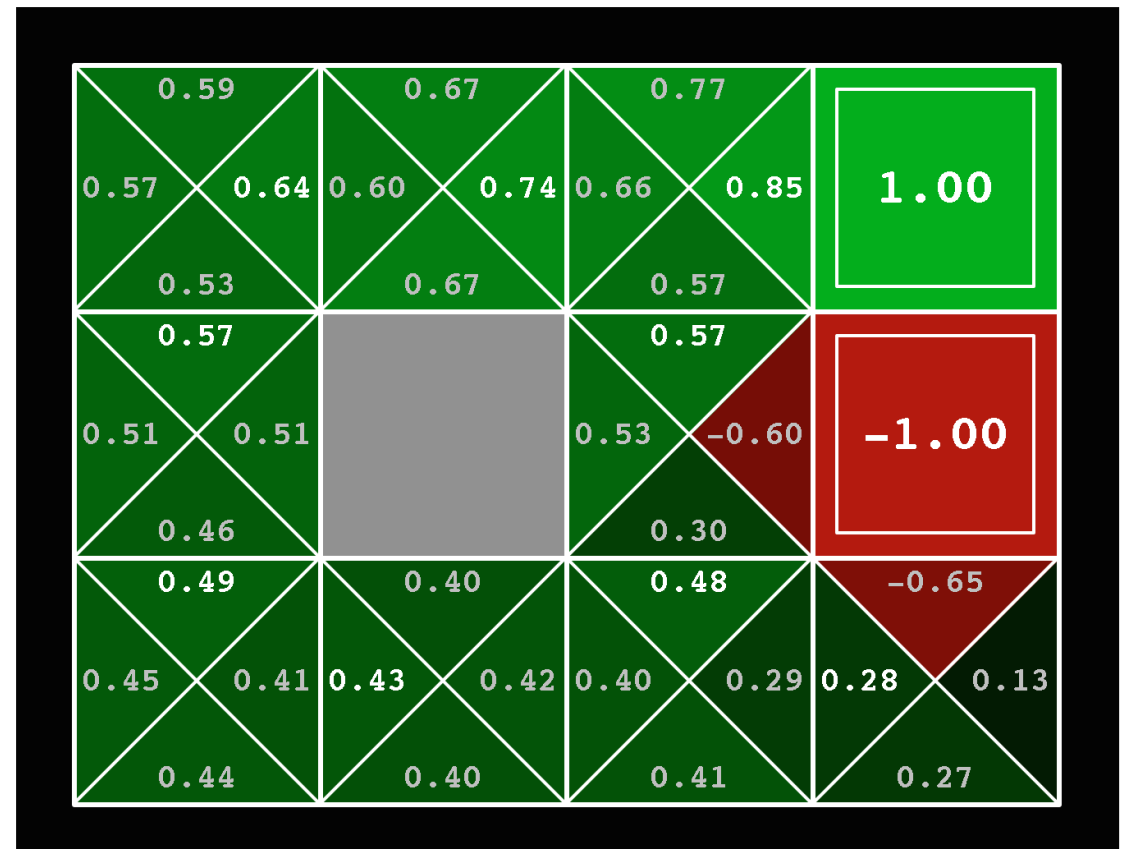
Solved MDP! Now what?

What are we going to do with these values??

$$V^*(s)$$



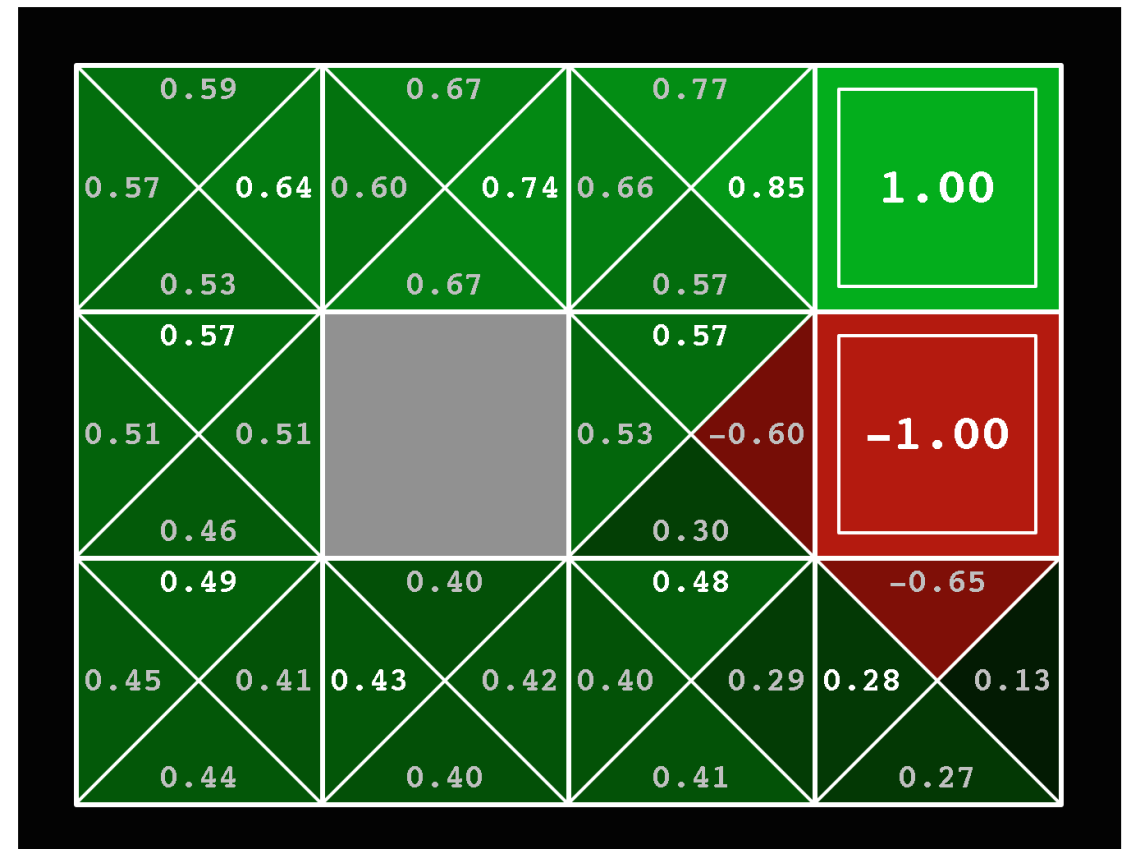
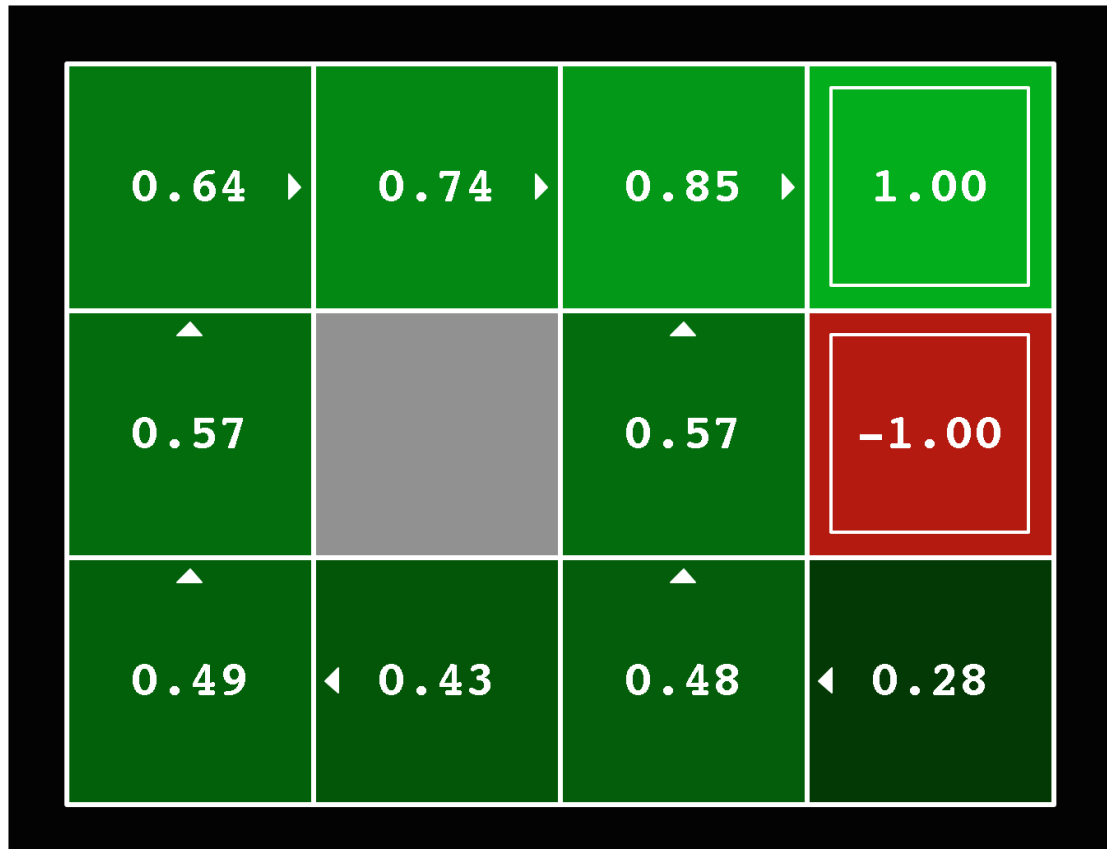
$$Q^*(s, a)$$



Poll 2

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

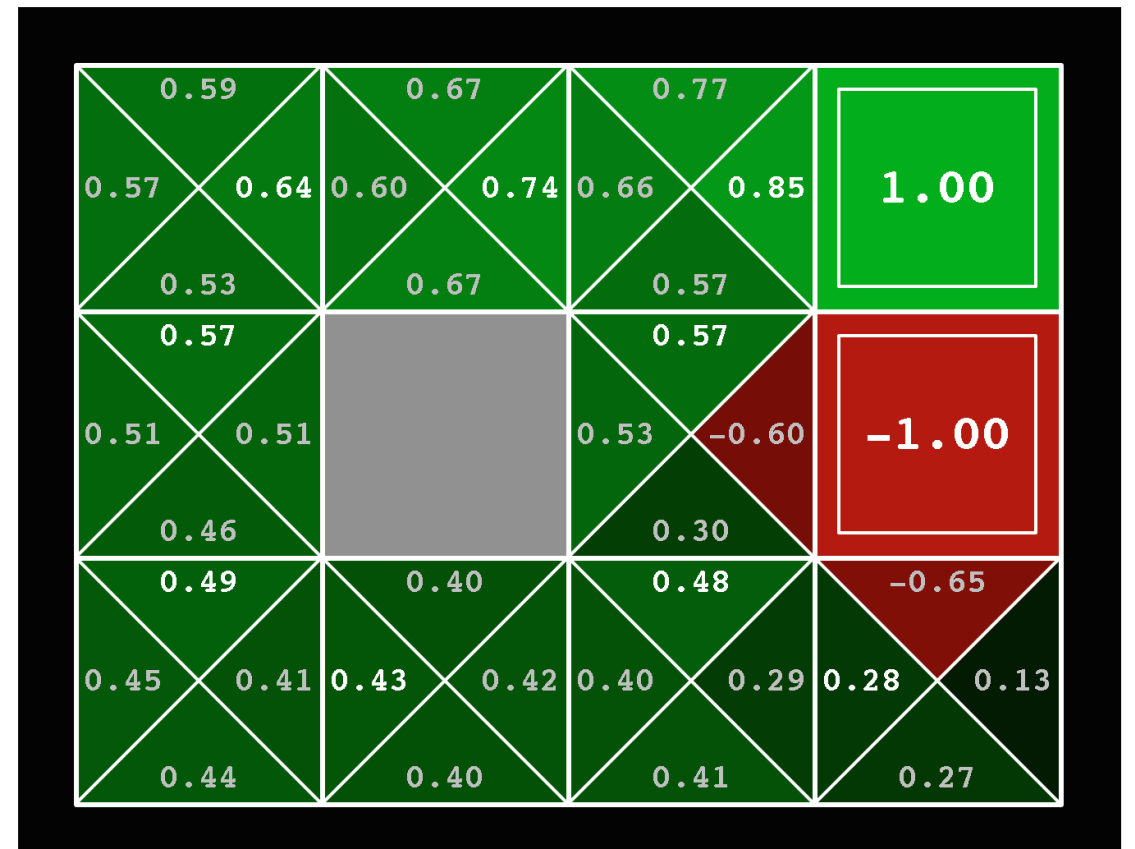
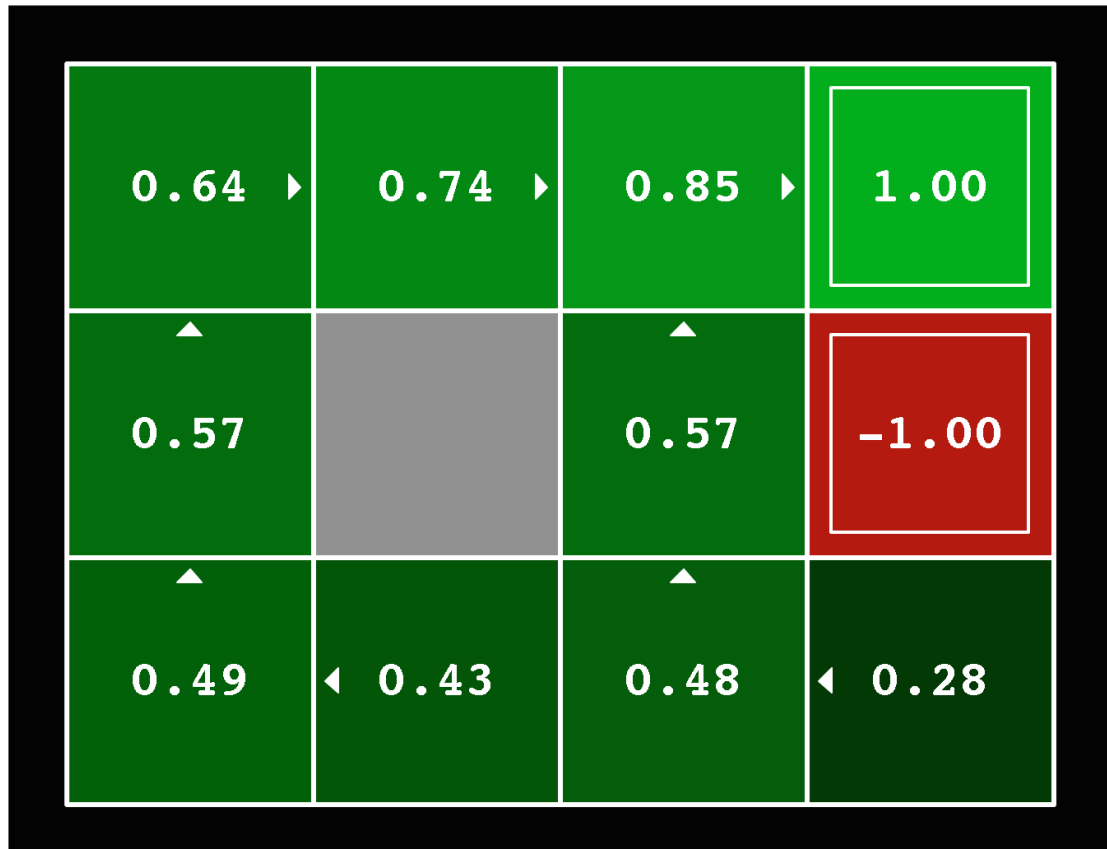
A) Values, B) Q-values?



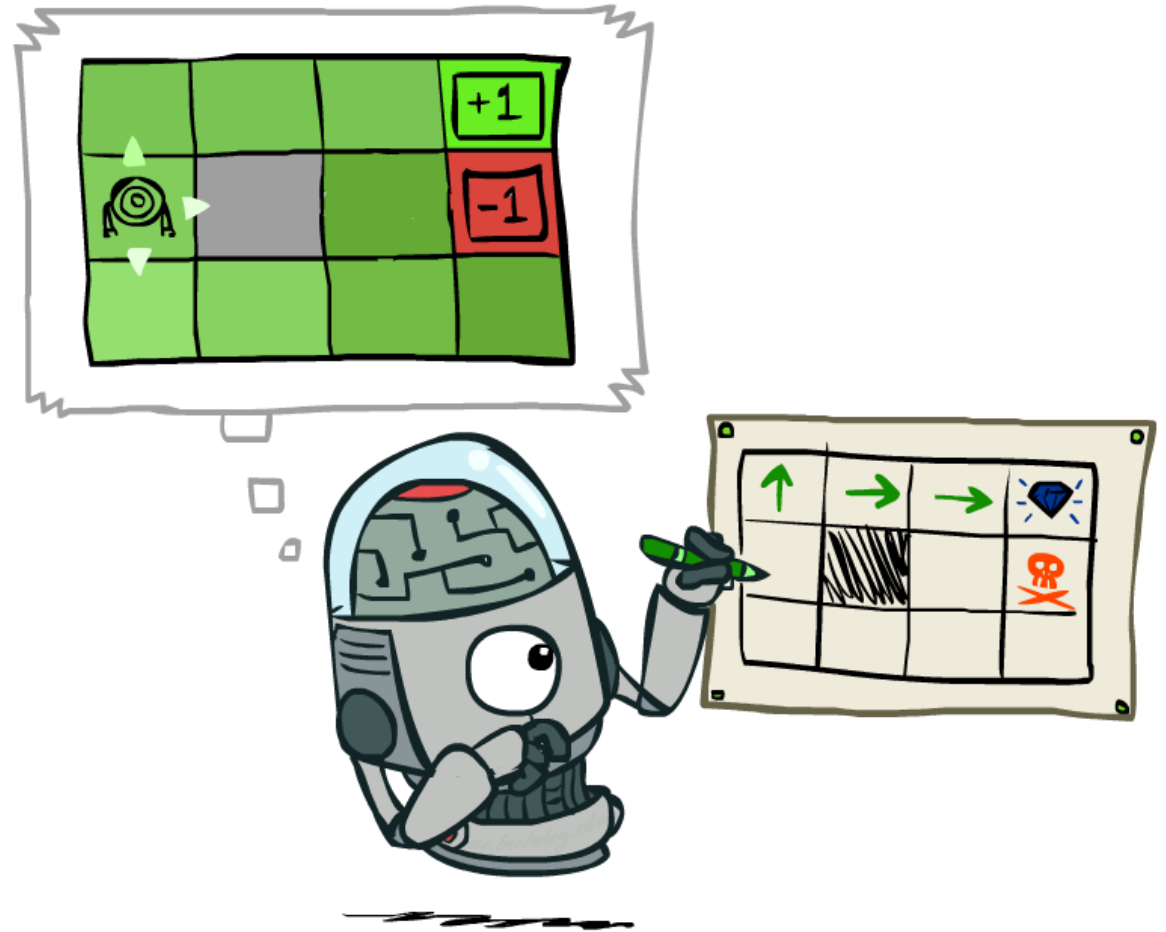
Poll 2

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

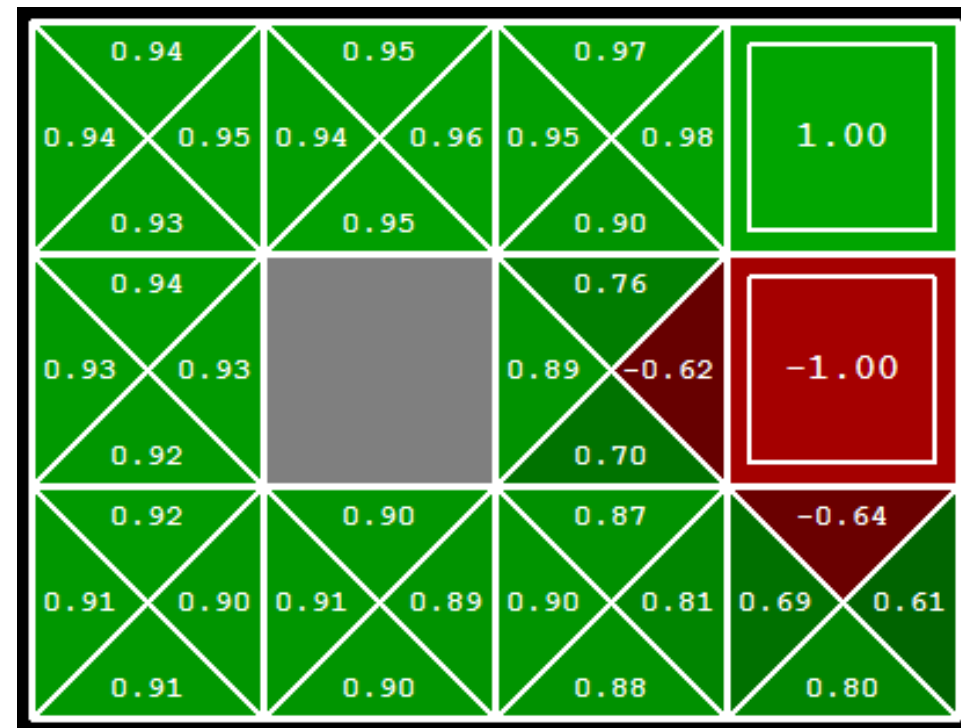
Computing Actions from Q-Values

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!

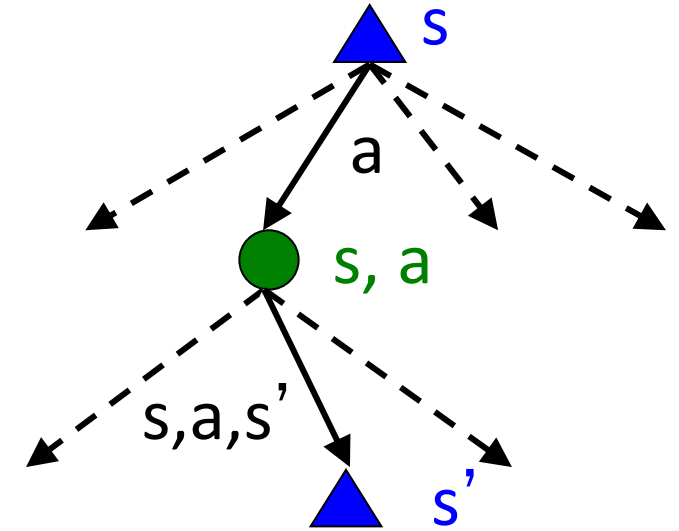
Value Iteration Notes

Value iteration repeats the Bellman updates:

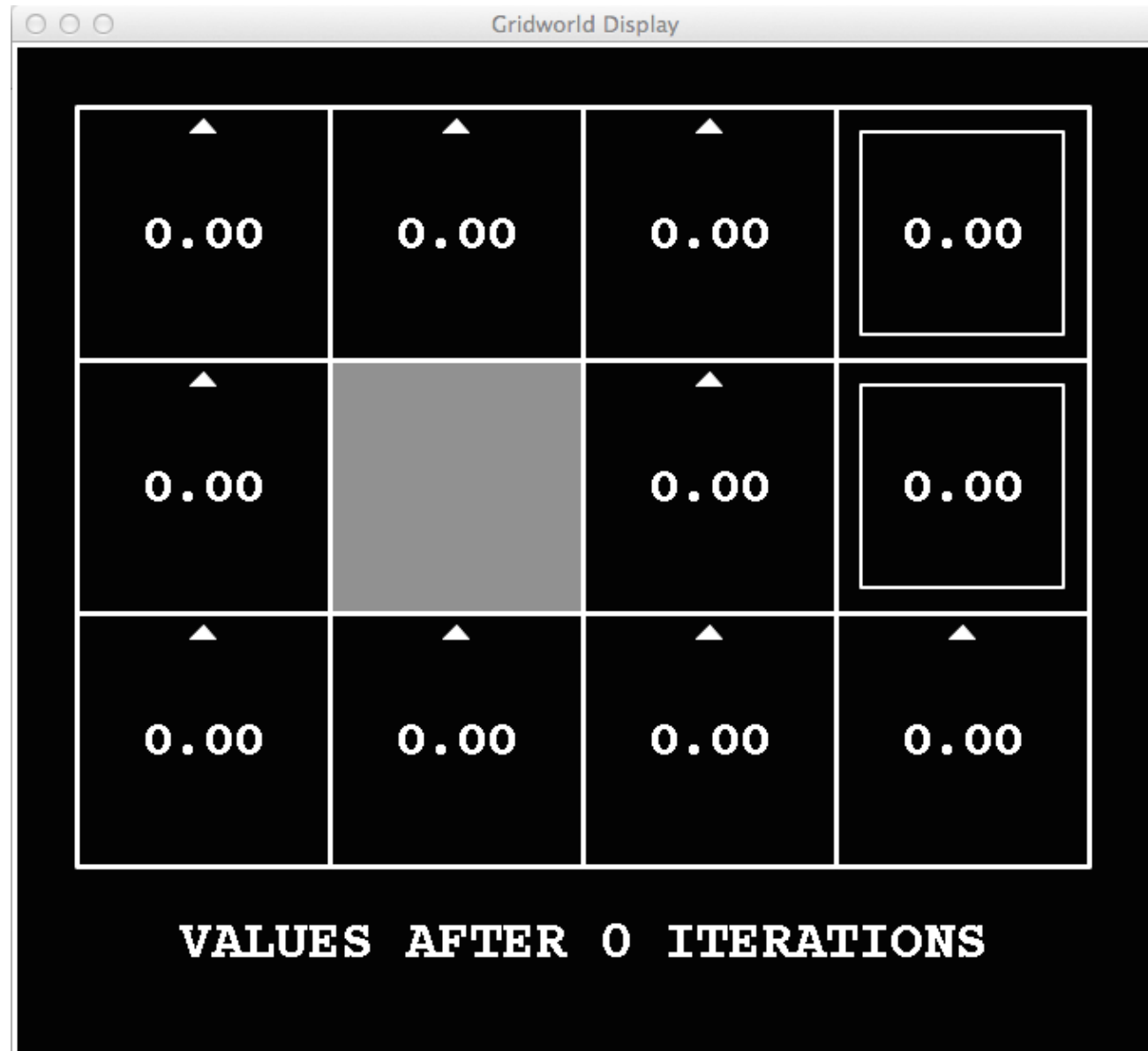
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_k(s')]$$

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)



$k=0$



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



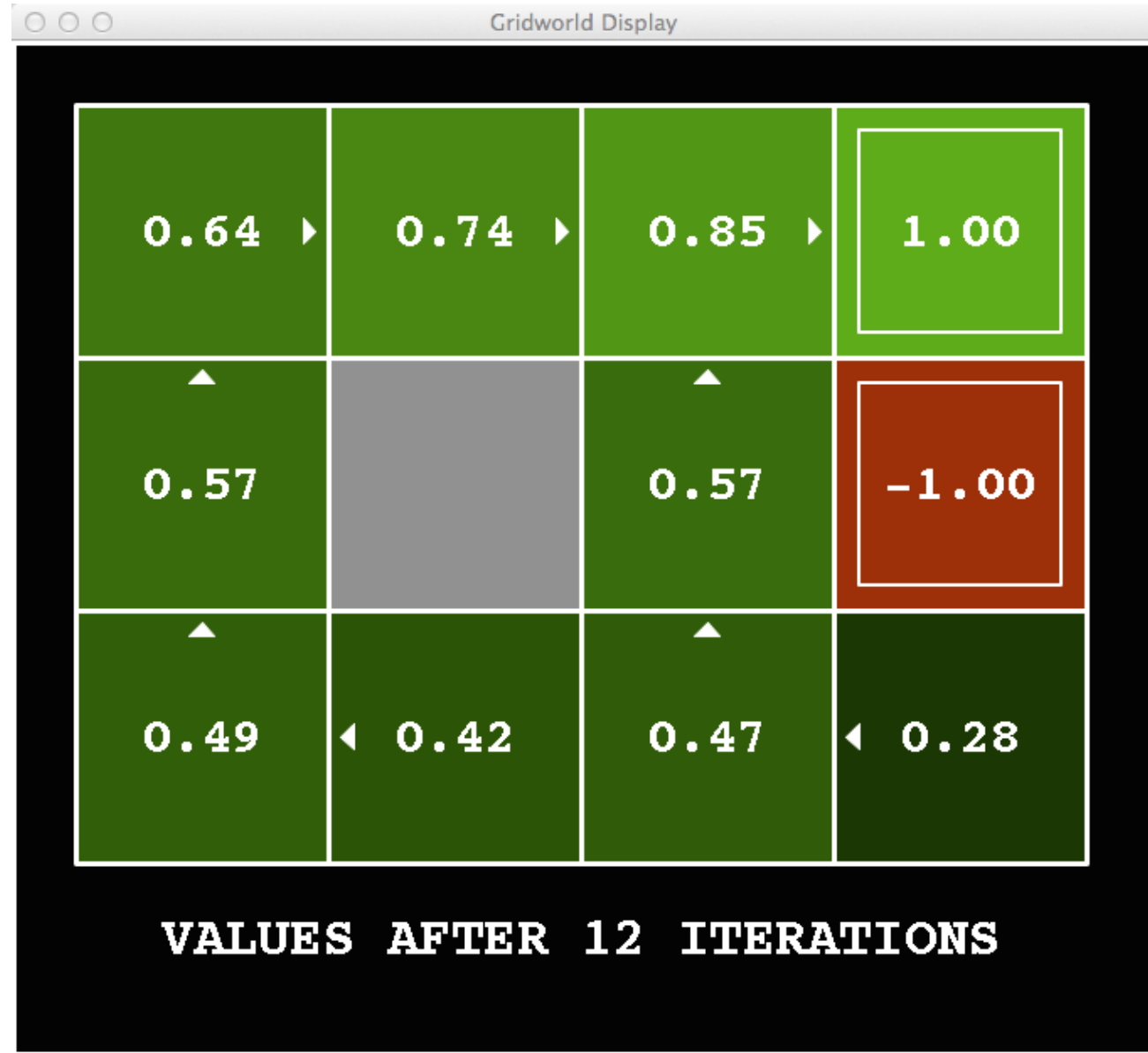
Noise = 0.2
Discount = 0.9
Living reward = 0

k=11



Noise = 0.2
Discount = 0.9
Living reward = 0

k=12



Noise = 0.2
Discount = 0.9
Living reward = 0

k=100

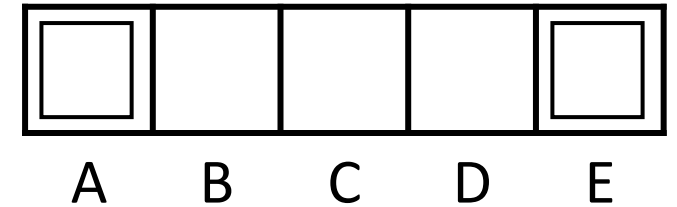


Noise = 0.2
Discount = 0.9
Living reward = 0

Outline

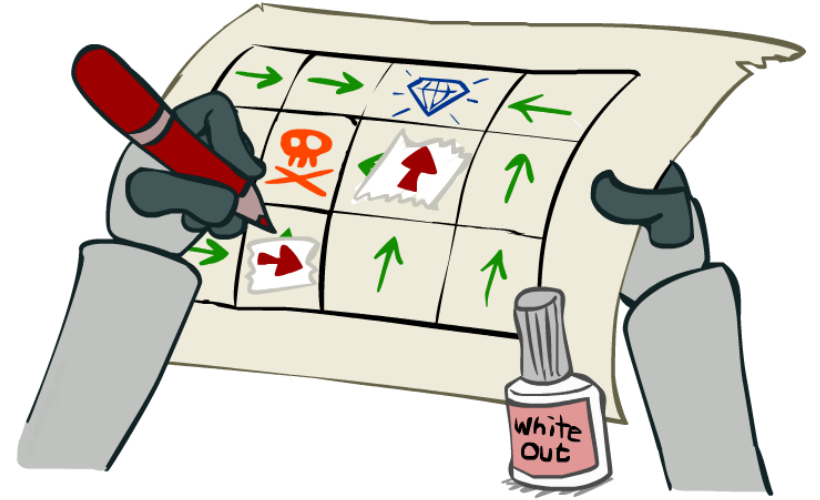
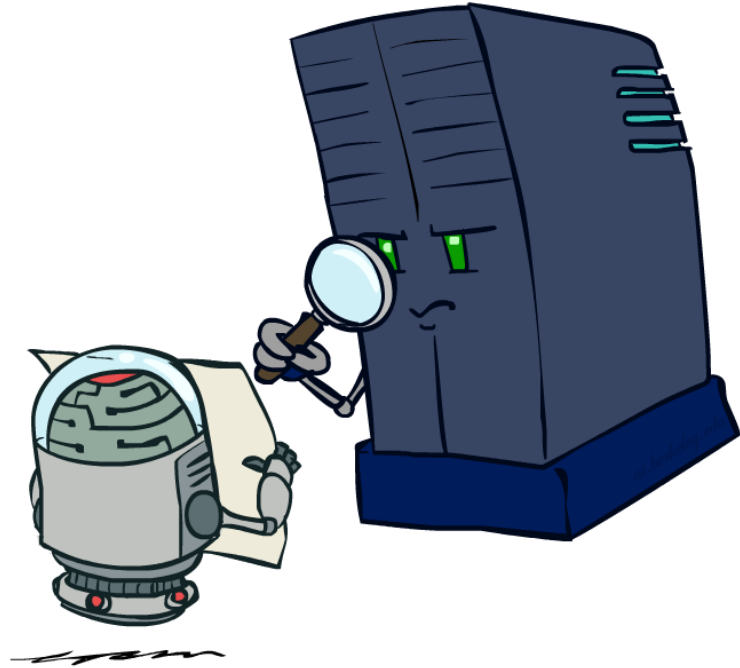
MDP Setup

- 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

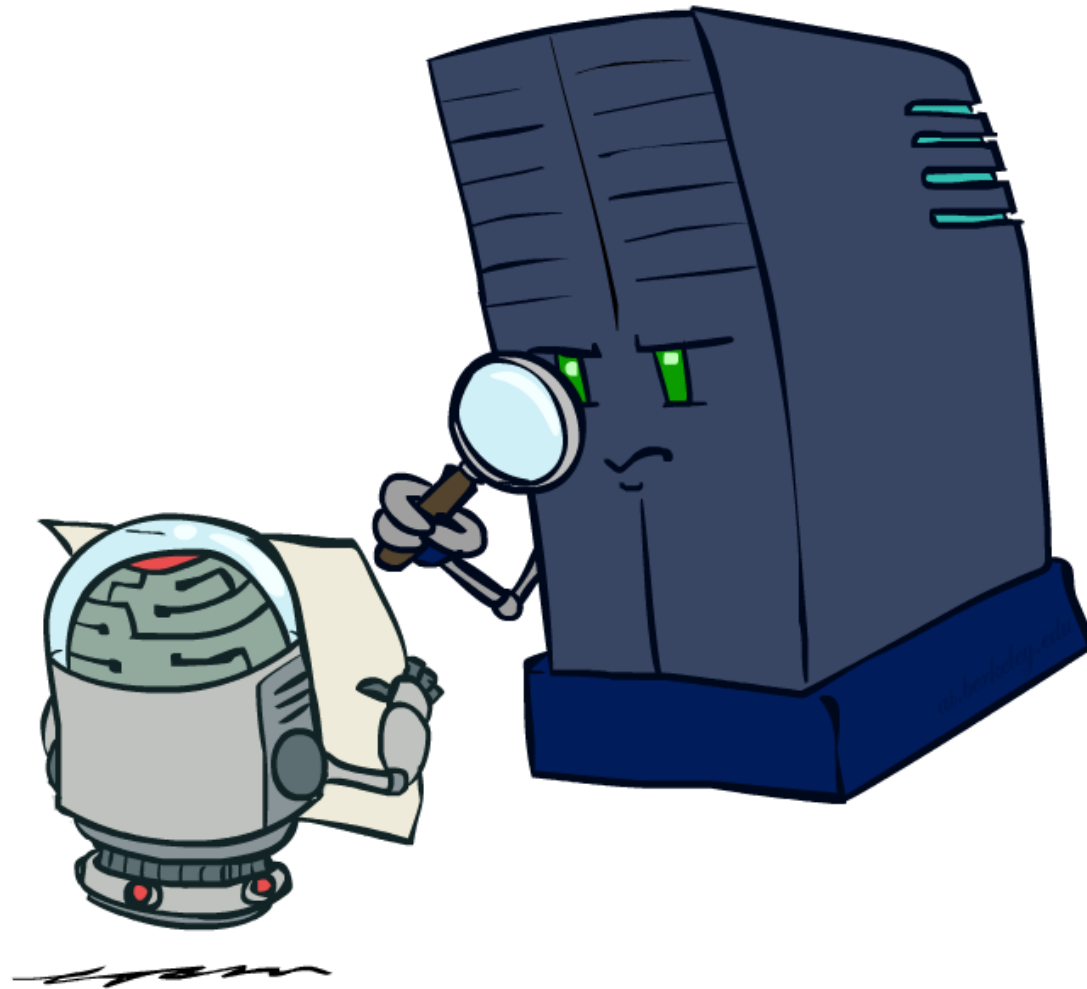
Value iteration + policy extraction

- **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

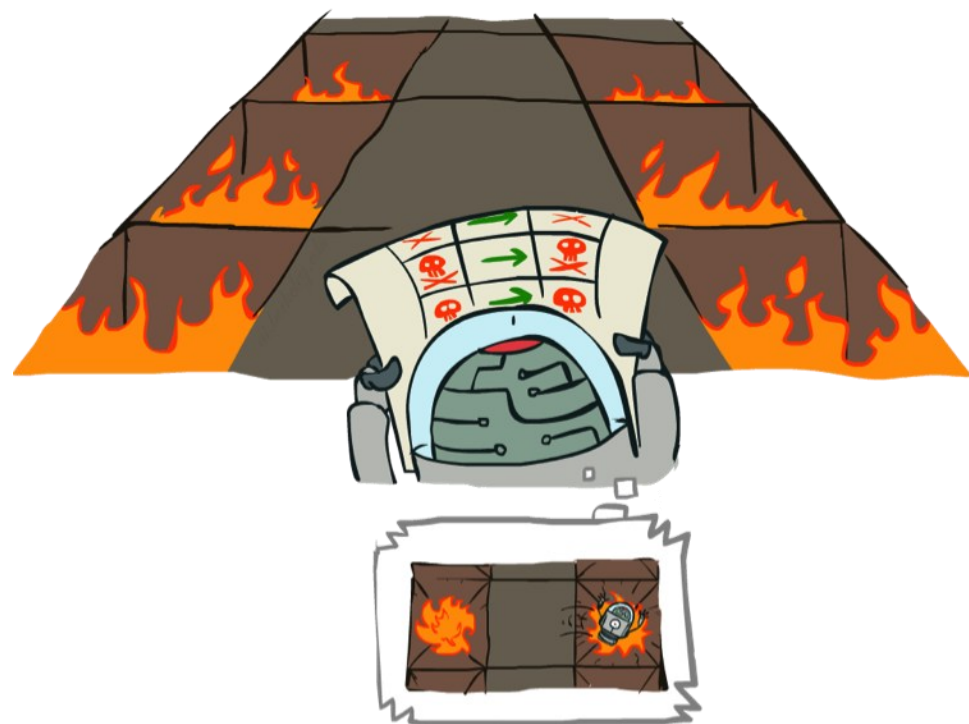
- **Step 1: Policy evaluation**: calculate values for some fixed policy (not optimal values!) **until convergence**
- **Step 2: Policy improvement**: update policy by running one ply of the Bellman equations using values from policy evaluation
- **Repeat** steps until policy converges

Policy Evaluation

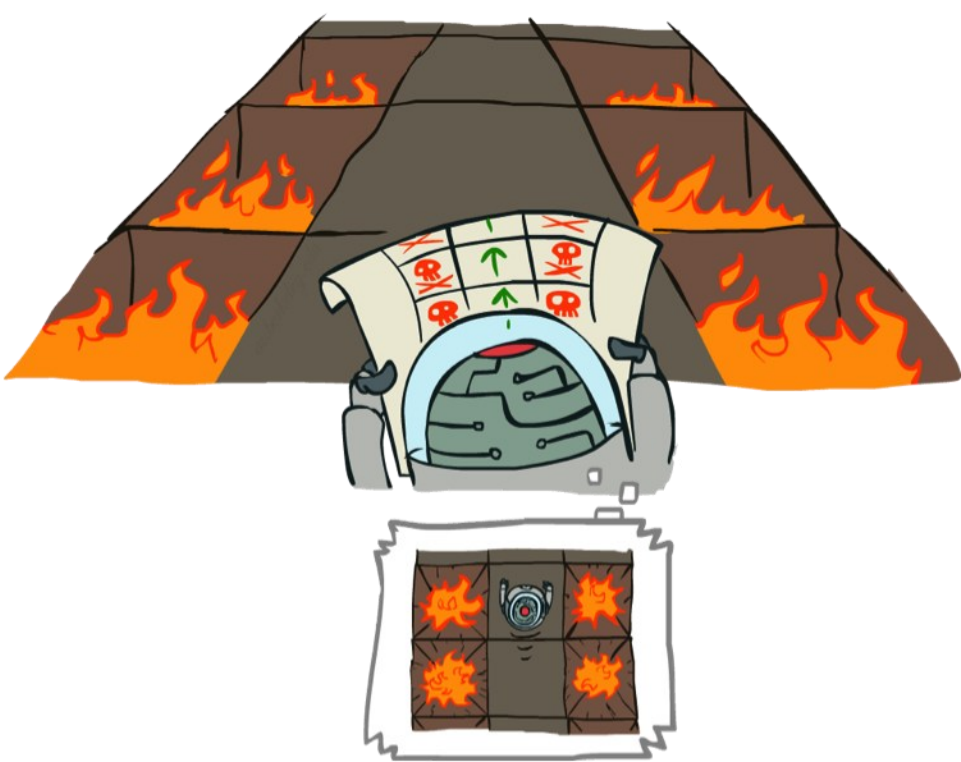


Example: Policy Evaluation

Always Go Right



Always Go Forward



Example: Policy Evaluation

Always Go Right

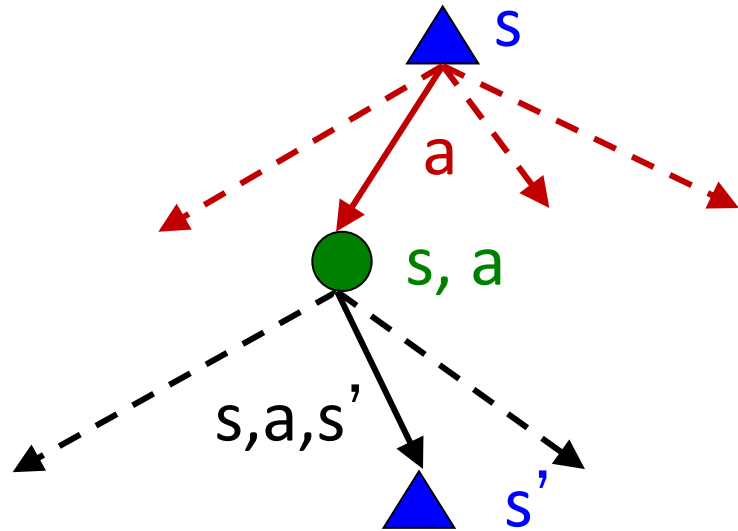


Always Go Forward

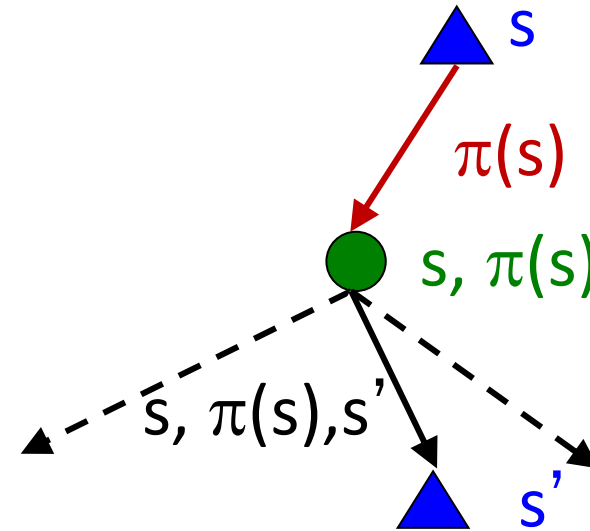


Policy Evaluation: Fixed Policies

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 $\pi(s)$, then the tree would be simpler
– only one action per state

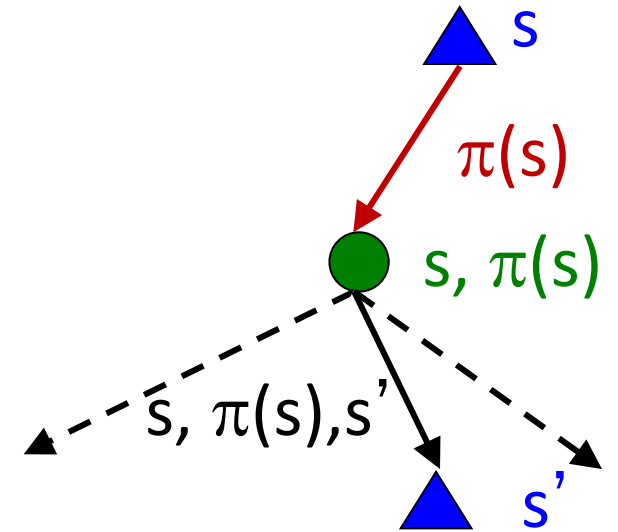
- ... though the tree's value would depend on which policy we fixed

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)

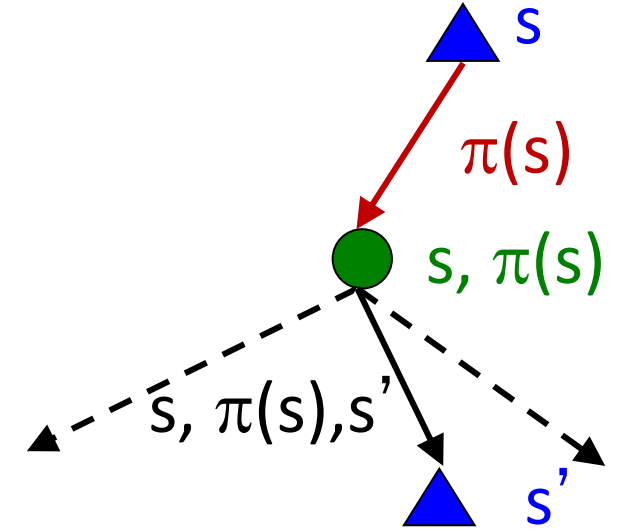
$$V_0^\pi(s) = 0$$

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

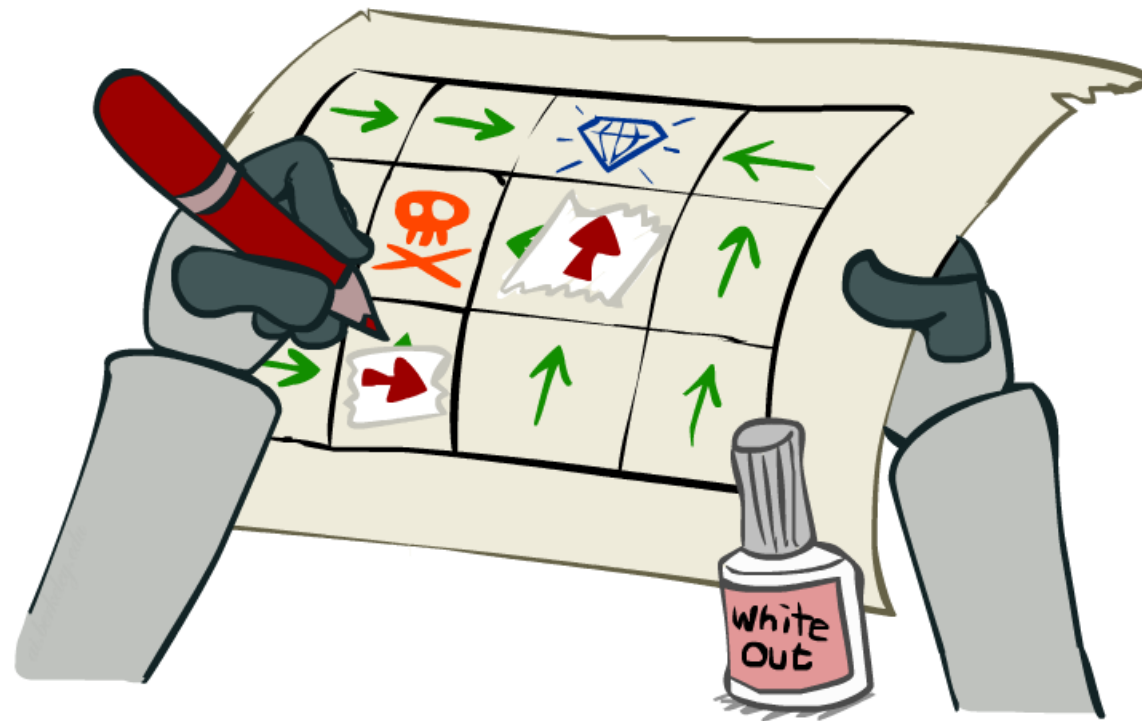
Efficiency: $O(S^2)$ per iteration

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:

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

Improvement: For fixed values, get a better policy using **policy extraction**

- One-step look-ahead:

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

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:**

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

- **Step 2: Policy extraction:**

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

Policy iteration

- **Step 1: Policy evaluation:**

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

- **Step 2: Policy improvement:**

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

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

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

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

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