Announcements

Assignments:

- P3: Optimization; Due today, 10 pm
- HW7 (online) 10/22 Tue, 10 pm

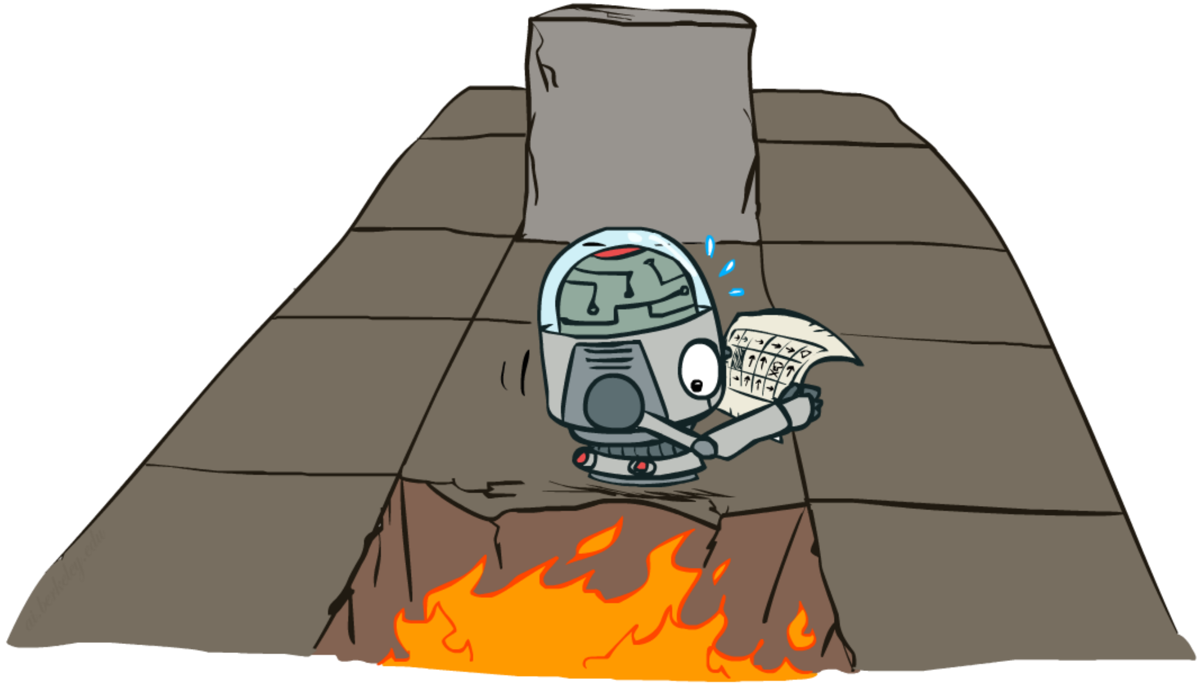
Recitations canceled on: October 18 (Mid-Semester Break) and October 25 (Day for Community Engagement)

We will provide recitation worksheet (reference for midterm/final)

Piazza post for In-class Questions

AI: Representation and Problem Solving

Markov Decision Processes II



Instructors: Fei Fang & Pat Virtue

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

Learning Objectives

- Write Bellman Equation for state-value and Q-value for optimal policy and a given policy
- Describe and implement value iteration algorithm (through Bellman update) for solving MDPs
- Describe and implement policy iteration algorithm (through policy evaluation and policy improvement) for solving MDPs
- Understand convergence for value iteration and policy iteration
- Understand concept of exploration, exploitation, regret

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

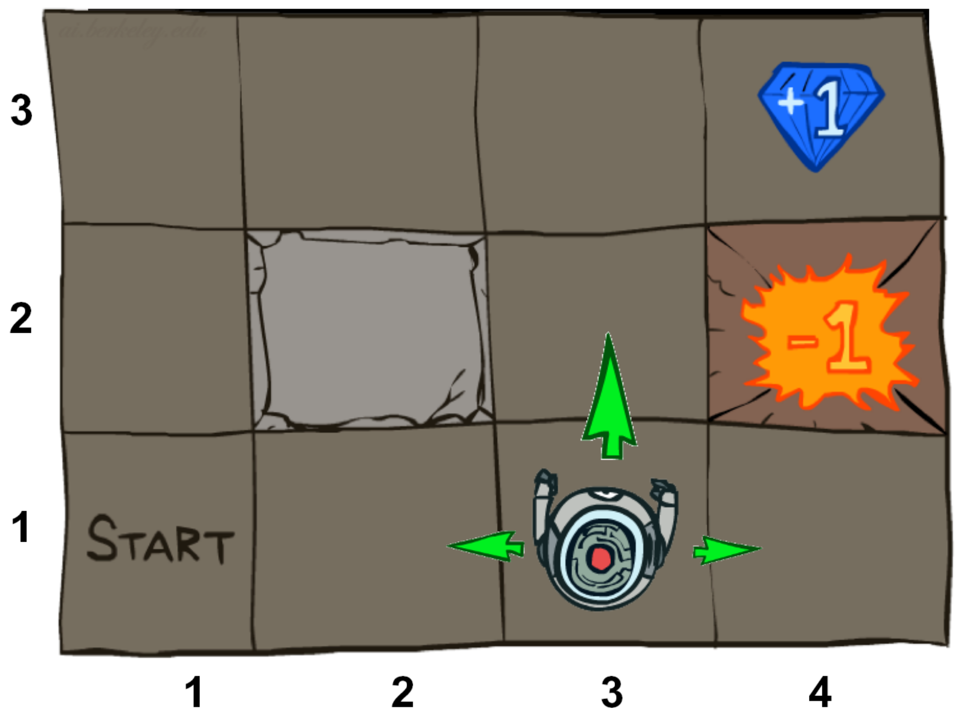
Q-iteration: $Q_{k+1}(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')], \quad \forall\, s,a$

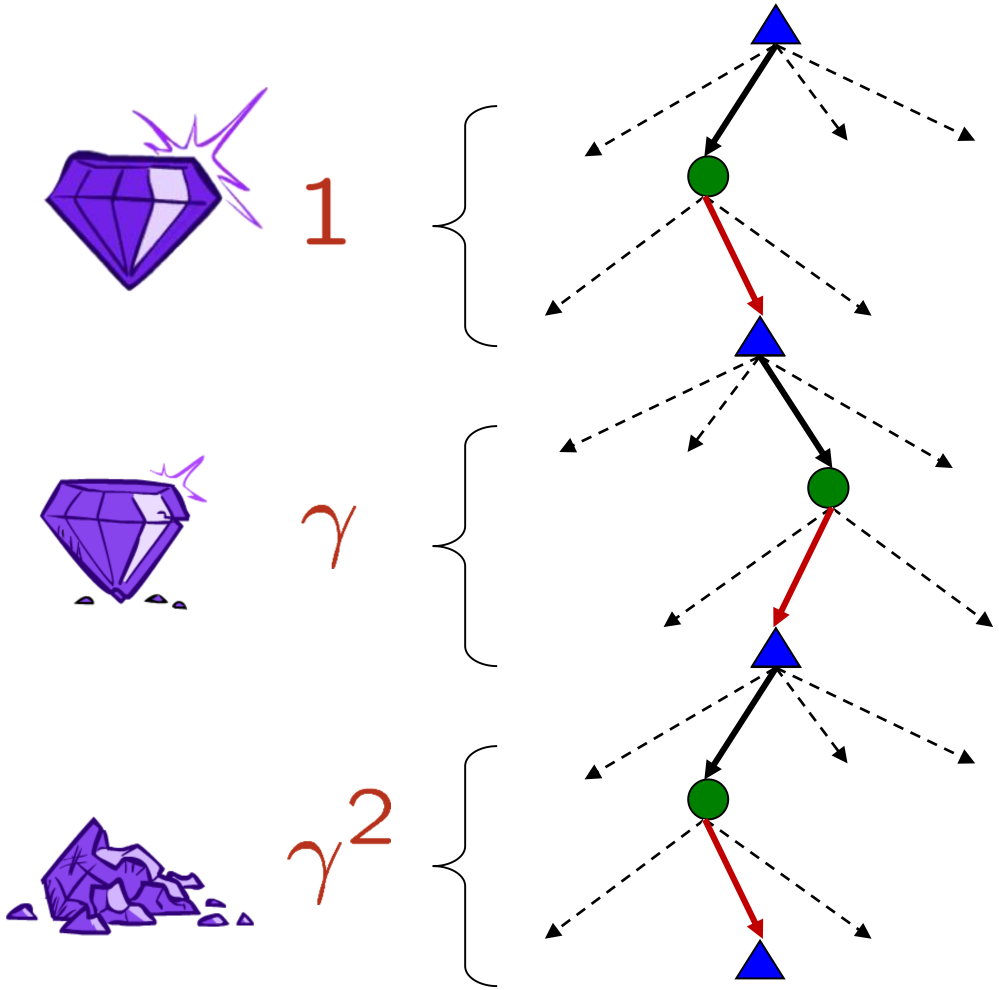
Policy extraction: $\pi_V(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')], \quad \forall\, s$

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

Example: Grid World



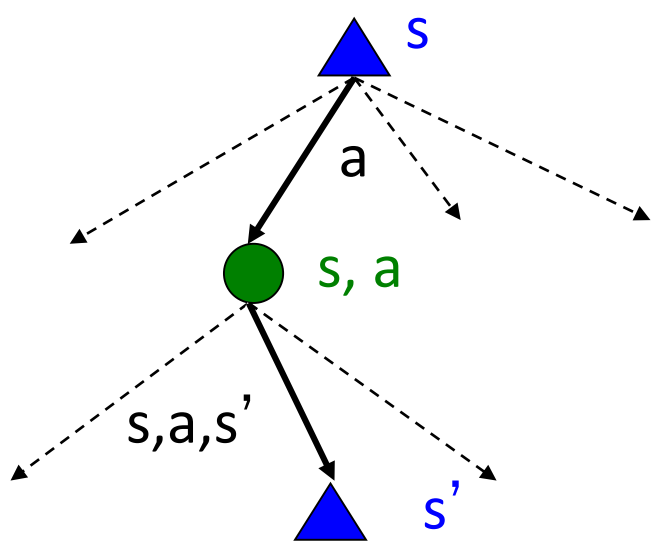


- Goal: maximize sum of (discounted) rewards

MDP Quantities

Markov decision processes:

- States S
- Actions A
- Transitions P(s'|s,a) (or T(s,a,s'))
- Rewards R(s,a,s') (and discount γ)
- Start state s_0



MDP quantities:

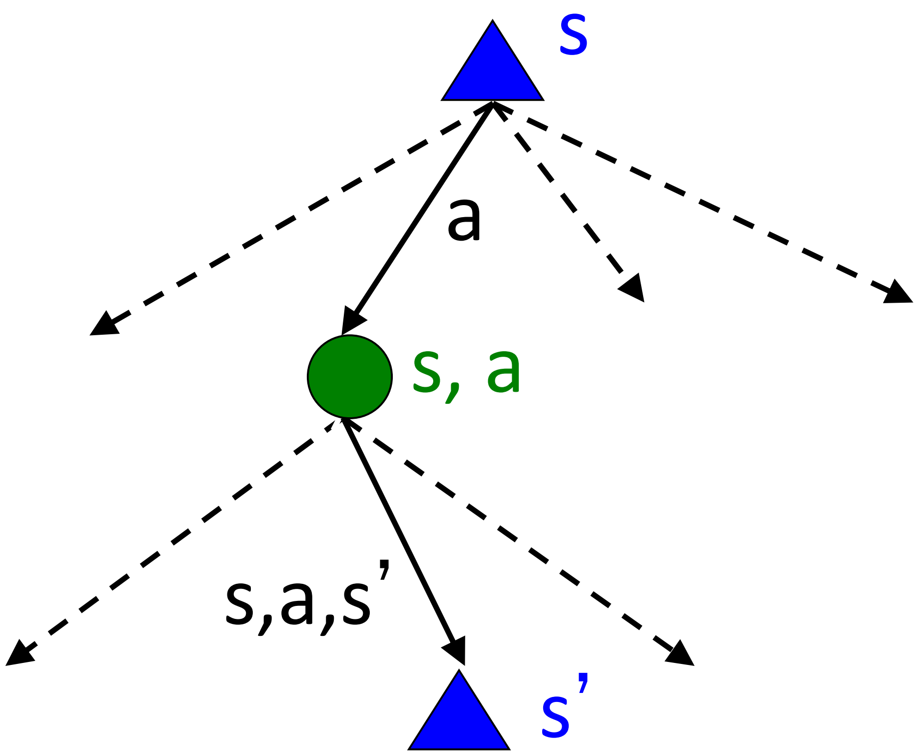
- Policy = map of states to actions
- Utility = sum of (discounted) rewards
- (State) Value = expected utility starting from a state (max node)
- Q-Value = expected utility starting from a state-action pair, i.e., q-state (chance node)

MDP Optimal Quantities

- The optimal policy:
  $\pi^*(s)$ = optimal action from state s

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

- The (true) value (or 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



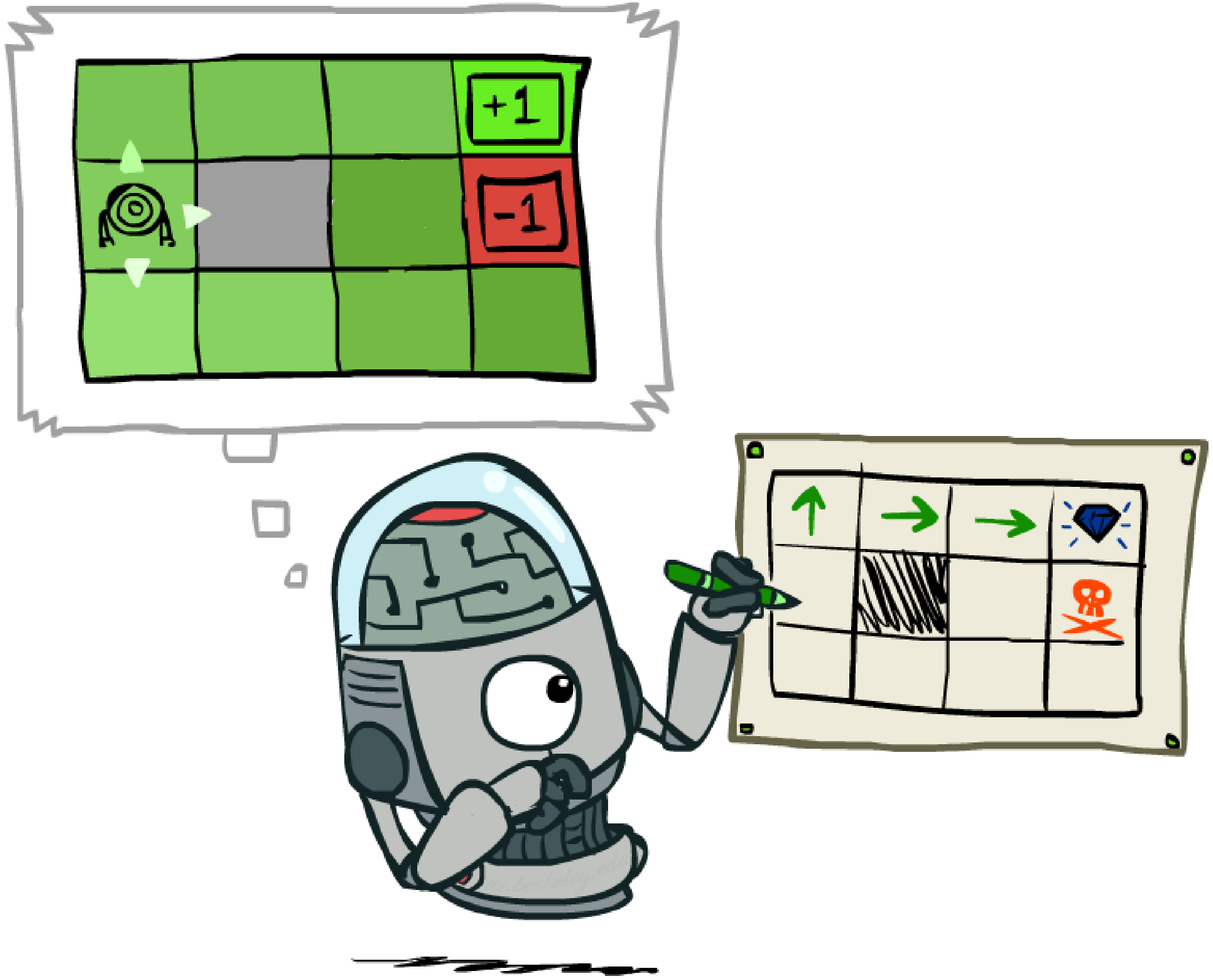
$V^*(s) < +\infty$ if $\gamma < 1$ and $R(s,a,s') < \infty$

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

Solve MDP: Find π^*, V^* and/or Q^*

[Demo: gridworld values (L9D1)]

Computing Optimal Policy from Values



Computing Optimal Policy from Values

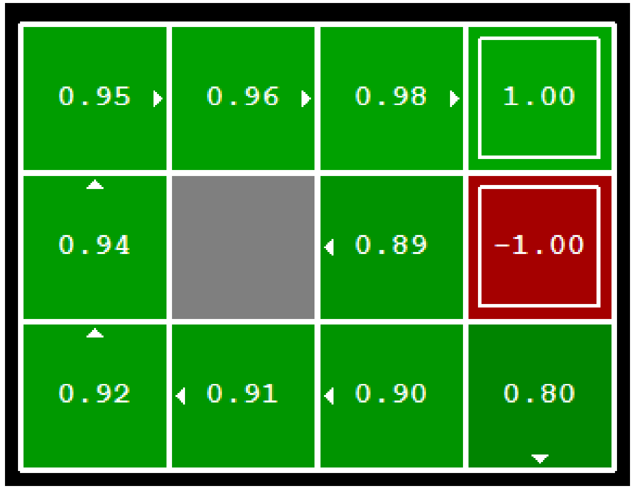
Let's imagine we have the optimal values V*(s)

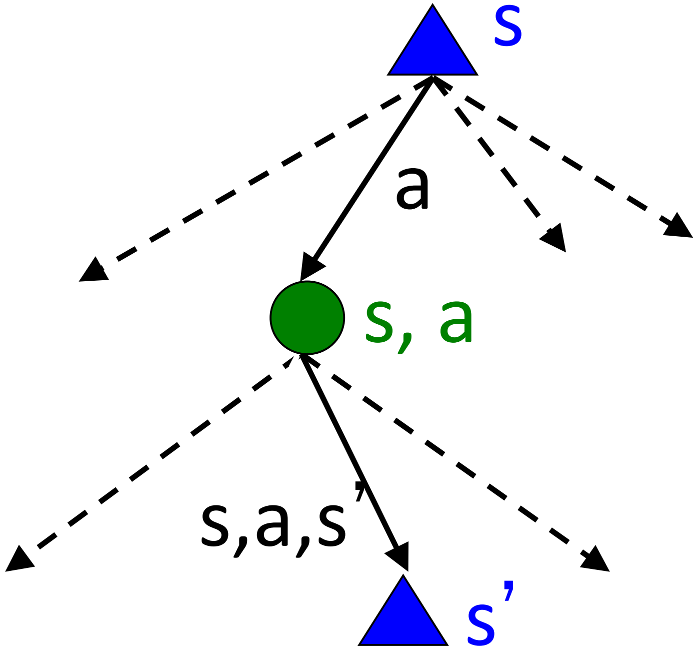
How should we act?

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

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

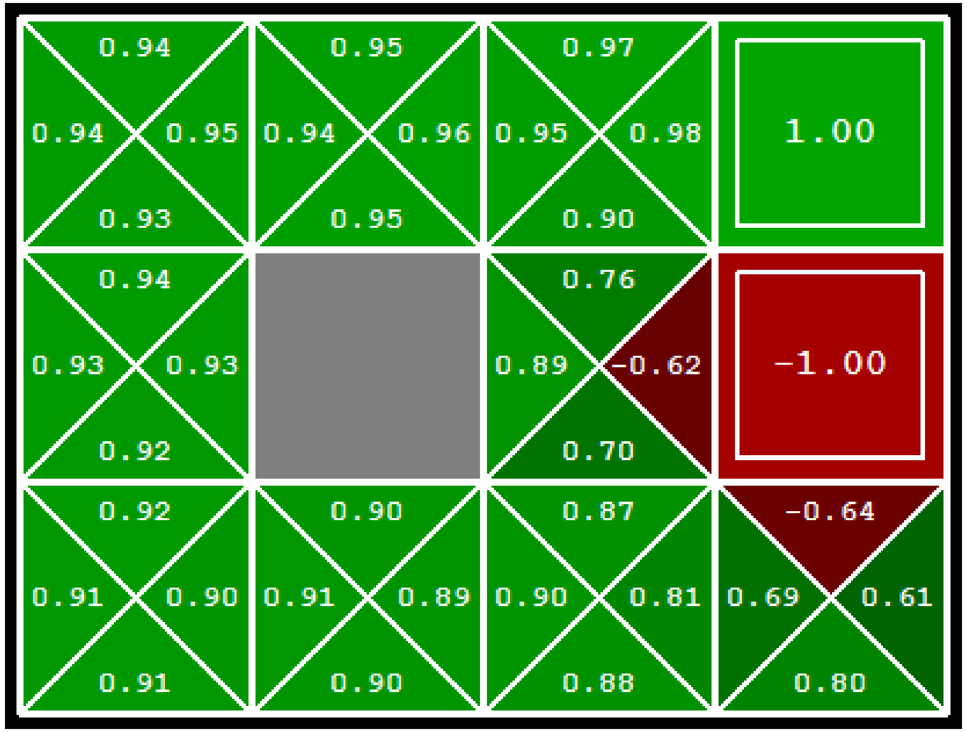
Sometimes this is called policy extraction, since it gets the policy implied by the values





Computing Optimal Policy from Q-Values

Let's imagine we have the optimal q-values:



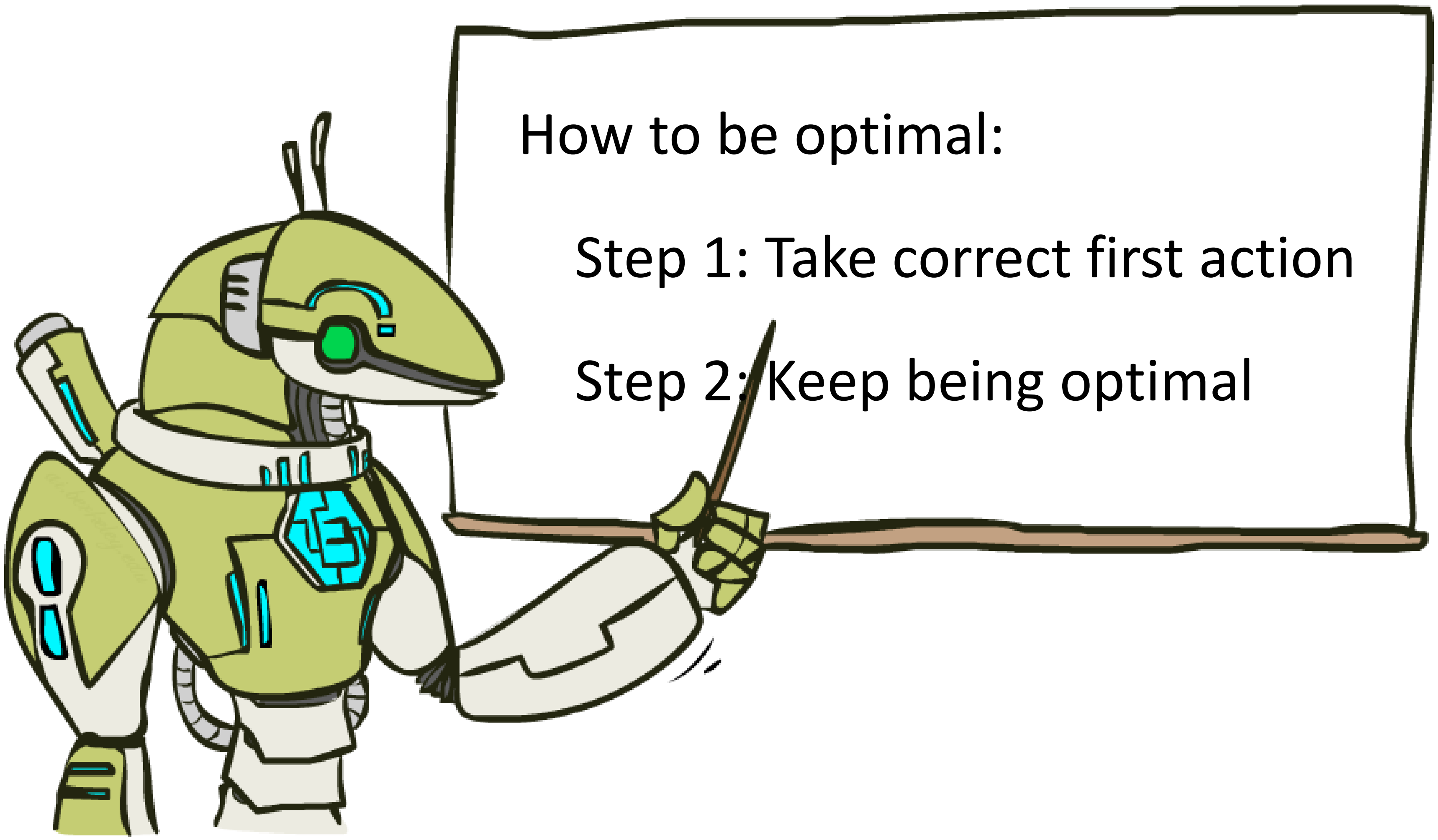
How should we act?

- Completely trivial to decide!

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

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

The Bellman Equations



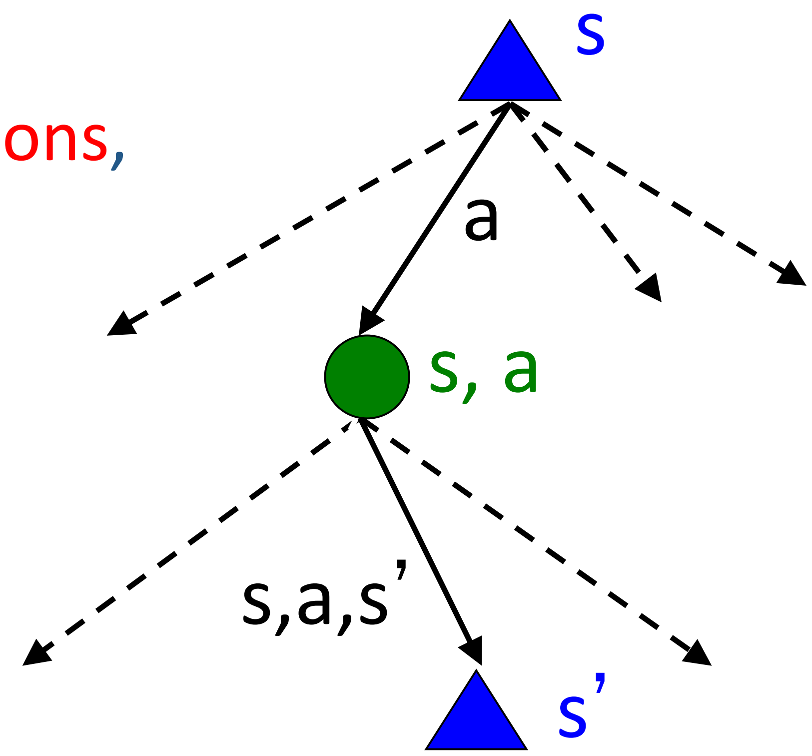
The Bellman Equations

Definition of "optimal utility" leads to Bellman Equations, which characterize the relationship amongst optimal utility values

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

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

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



Expectimax-like computation

Necessary and sufficient conditions for optimality

Solution is unique

The Bellman Equations

Definition of "optimal utility" leads to Bellman Equations, which characterize the relationship amongst optimal utility values

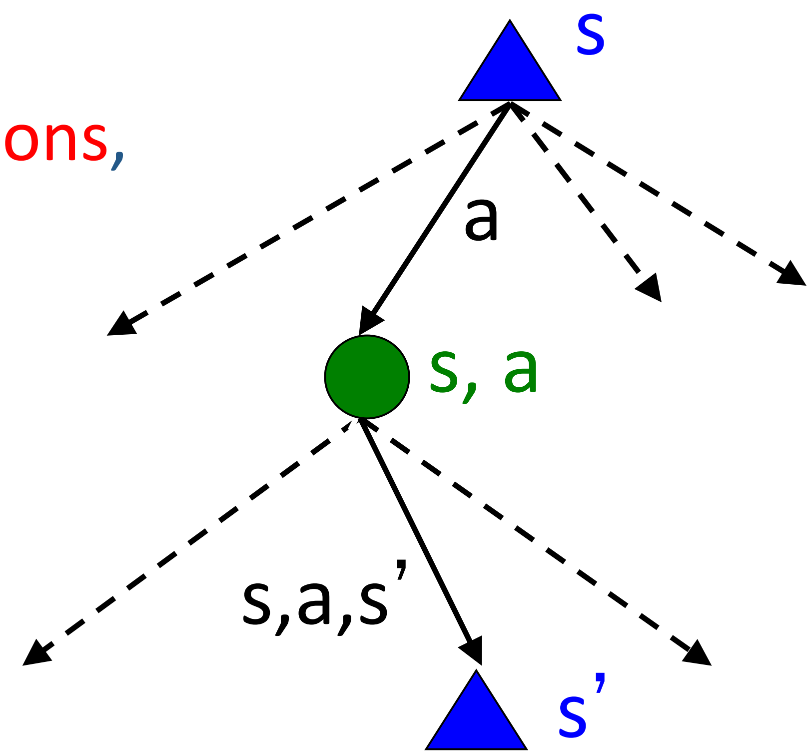
$$V^*(s) = \max_a Q^*(s,a)$$

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

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

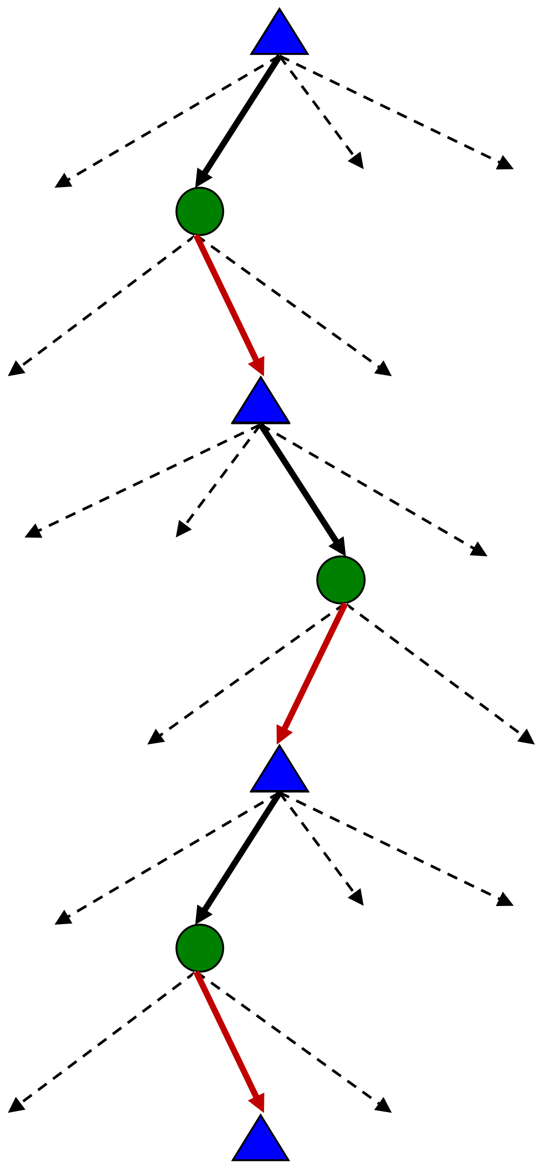
Necessary and sufficient conditions for optimality

Solution is unique



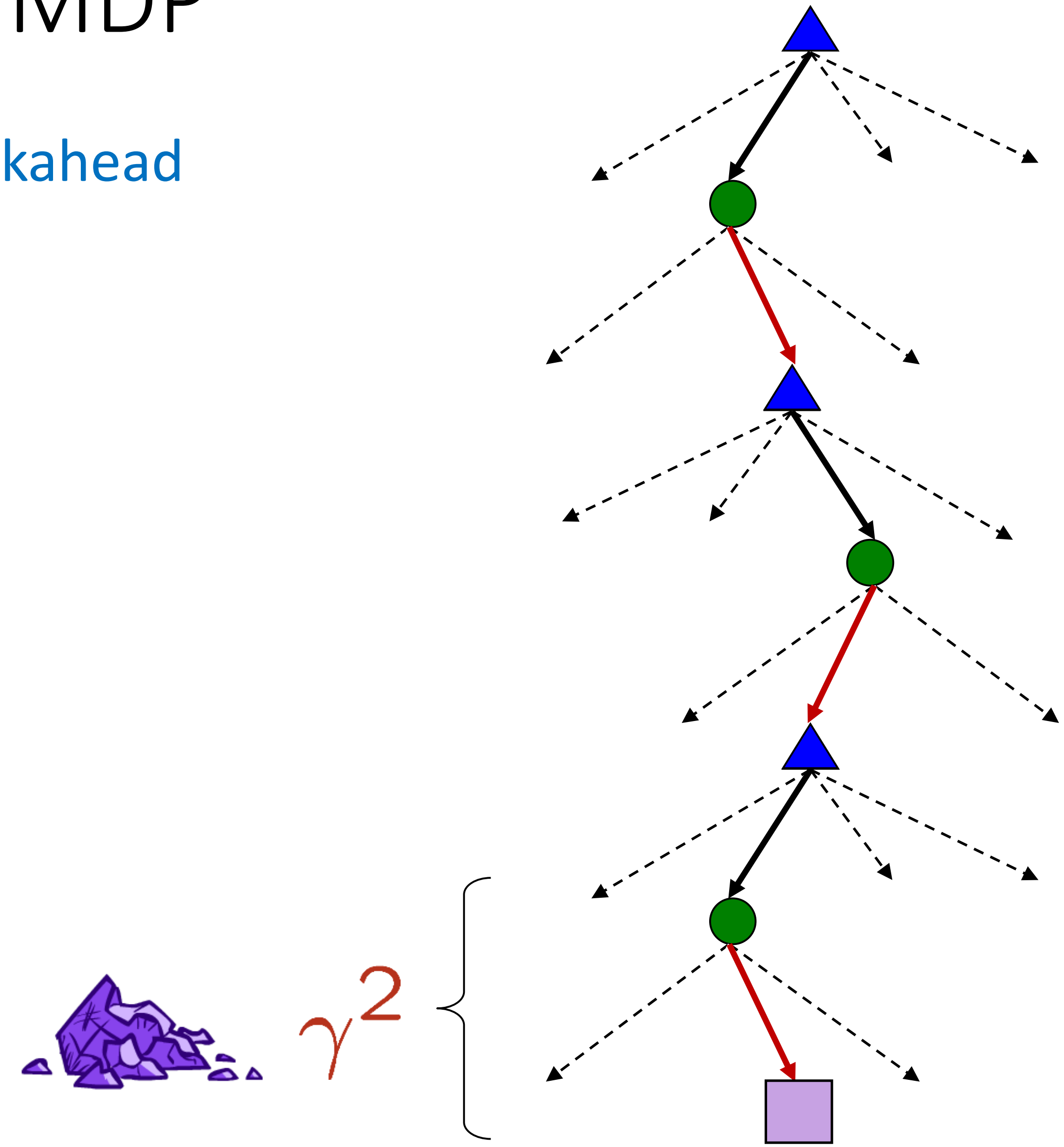
Expectimax-like computation with one-step lookahead and a "perfect" heuristic at leaf nodes

Solving Expectimax



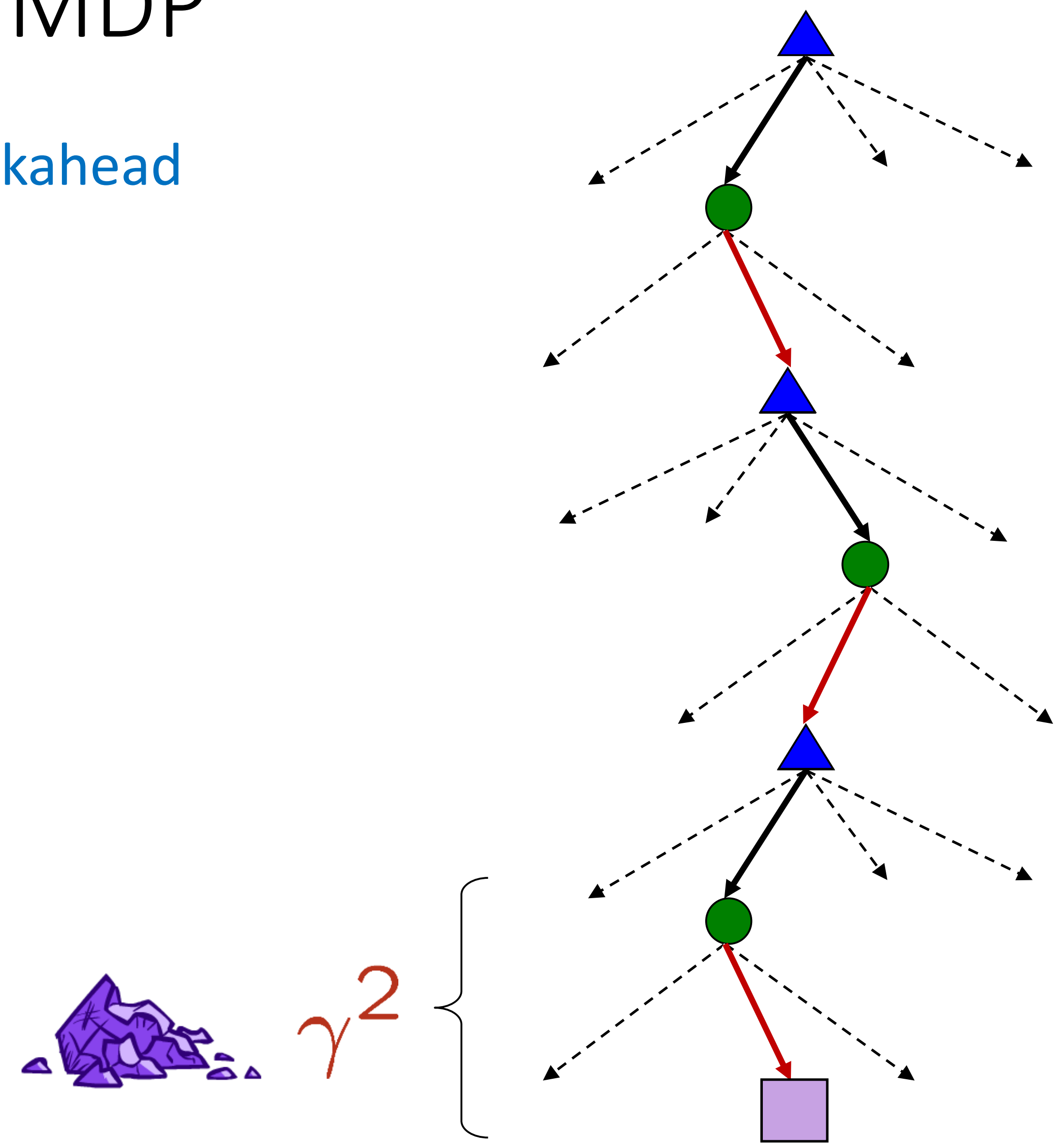
Solving MDP

Limited Lookahead

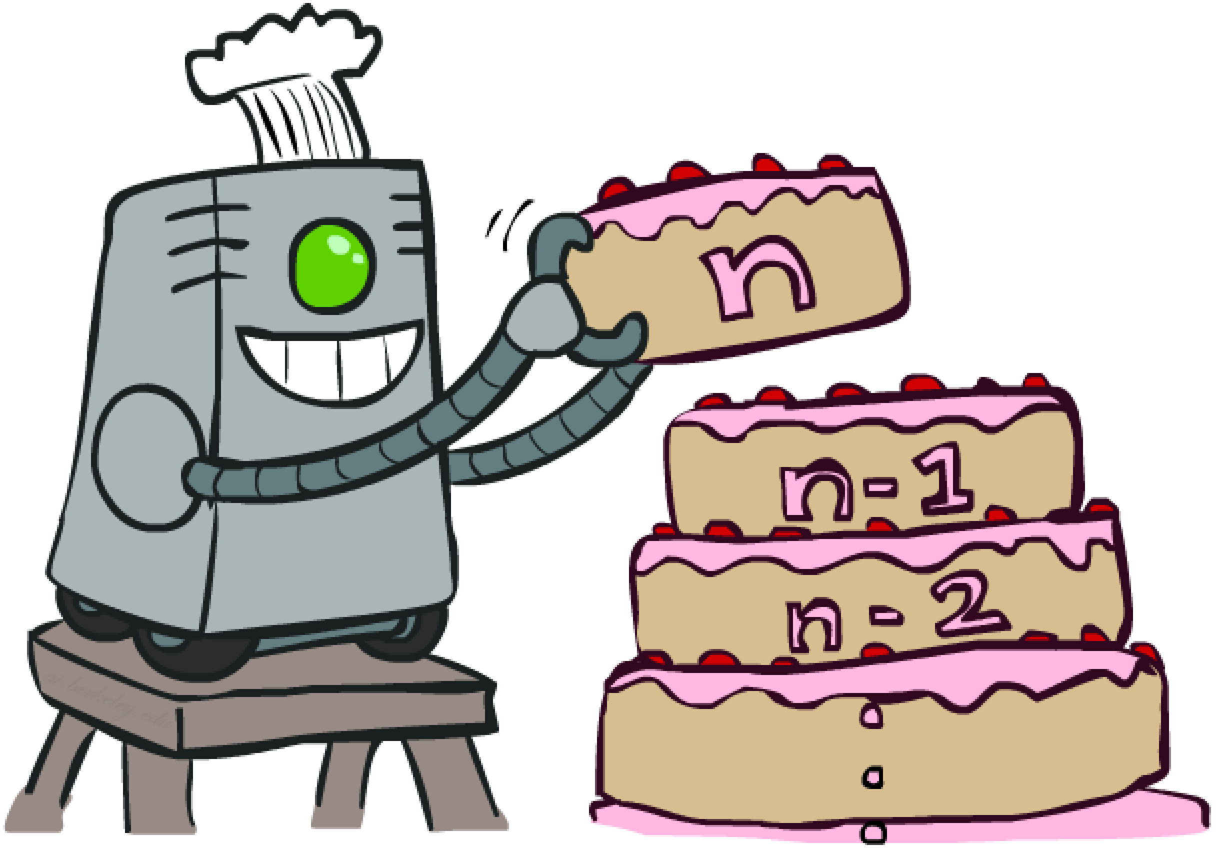


Solving MDP

Limited Lookahead



Value Iteration



Demo Value Iteration

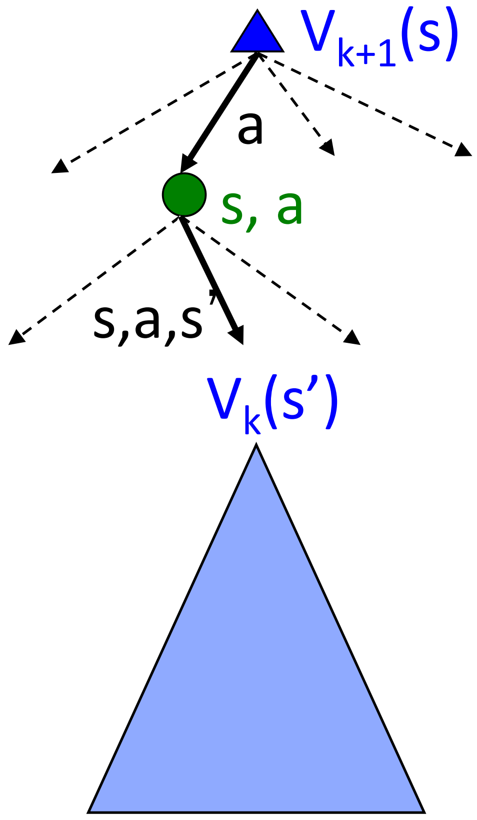
[Demo: value iteration (L8D6)]

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, apply Bellman update once (do one ply of expectimax with R and γ 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

Will this process converge?

Yes!

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

function VALUE-ITERATION(MDP=(S,A,T,R,γ), $threshold$) returns a state value function

```
for s in S
    V_0(s) ← 0
k ← 0
repeat
    δ ← 0
    for s in S
        V_{k+1}(s) ← −∞
        for a in A
            v ← 0
            for s' in S
                v ← v + T(s,a,s')(R(s,a,s') + γV_k(s))
            V_{k+1}(s) ← max{V_{k+1}(s), v}
        δ ← max{δ, |V_{k+1}(s) − V_k(s)|}
    k ← k + 1
until δ < treshold
return V_{k−1}
```

Do we really need to store the value of V_k for each k?

Does $V_{k+1}(s) \geq V_k(s)$ always hold?

Value Iteration

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

function VALUE-ITERATION(MDP=(S,A,T,R,γ), $threshold$) returns a state value function

```
for s in S
    V_0(s) ← 0
k ← 0
repeat
    δ ← 0
    for s in S
        V_{k+1}(s) ← −∞
        for a in A
            v ← 0
            for s' in S
                v ← v + T(s,a,s')(R(s,a,s') + γV_k(s))
            V_{k+1}(s) ← max{V_{k+1}(s), v}
        δ ← max{δ, |V_{k+1}(s) − V_k(s)|}
    k ← k + 1
until δ < treshold
return V_{k−1}
```

Do we really need to store the value of V_k for each k?

No. Use $V = V_{last}$ and $V' = V_{current}$

Does $V_{k+1}(s) \geq V_k(s)$ always hold?

No. If $T(s,a,s') = 1$ and $R(s,a,s') < 0$, then $V_1(s) = R(s,a,s') < 0$

Bellman Equation vs Value Iteration vs Bellman Update

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 by applying Bellman update repeatedly

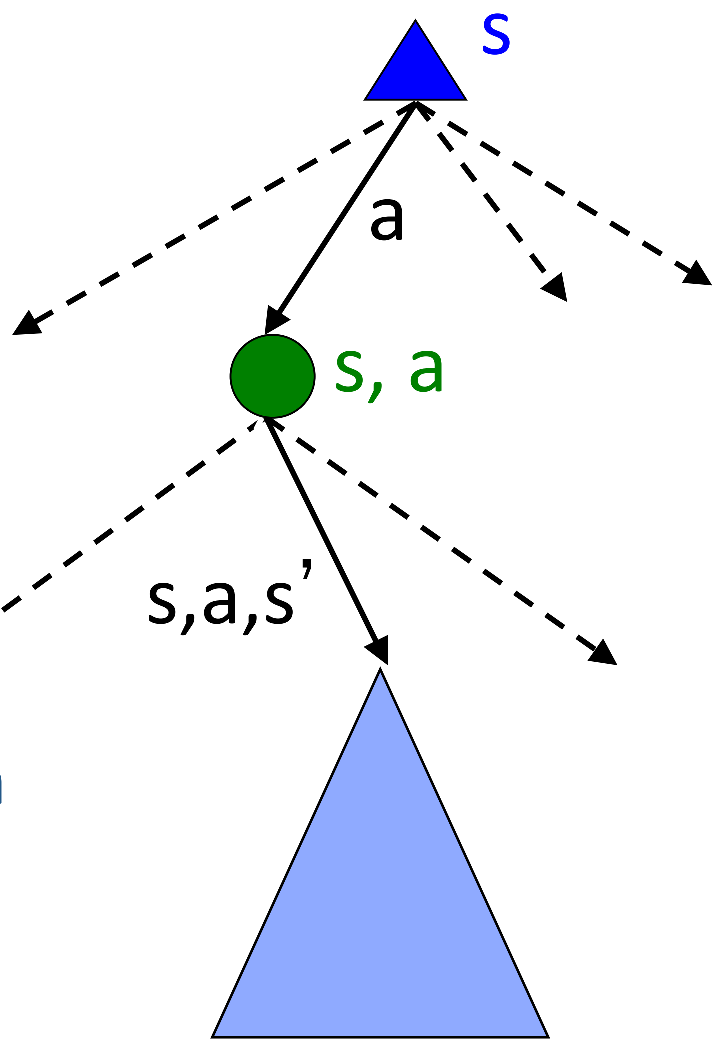
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s')\right]$$

Value iteration is a method for solving Bellman Equation

V_k vectors are also interpretable as time-limited values

Value iteration finds the fixed point of the function

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



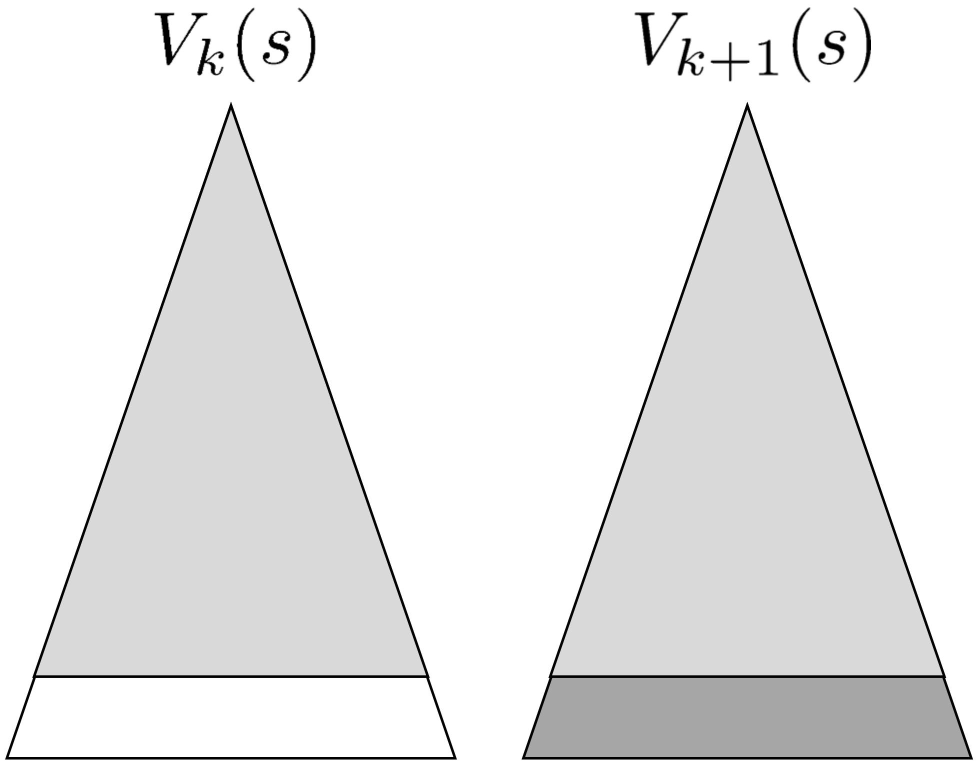
Value Iteration Convergence

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

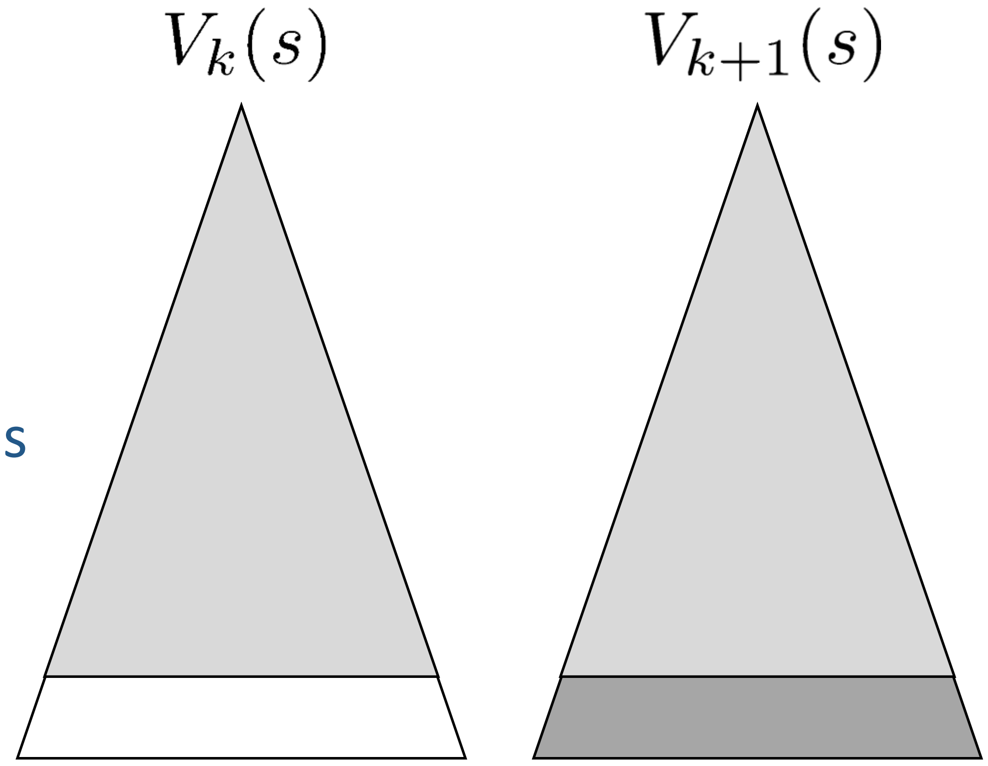
Case 2: If $\gamma < 1$ and $|R(s,a,s')| \leq R_{max} < \infty$

- Intuition: For any state V_k and V_{k+1} can be viewed as depth k+1 expectimax results (with R and γ) in nearly identical search trees
- The difference is that on the bottom layer, V_{k+1} has actual rewards while V_k has zeros
- $|R(s,a,s')| \leq R_{max}$
- $|V_1(s) - V_0(s)| = |V_1(s) - 0| \leq R_{max}$
- $|V_{k+1}(s) - V_k(s)| \leq \gamma^k R_{max}$
- So $|V_{k+1}(s) - V_k(s)| \to 0$ as $k \to \infty$



If we initialized $V_0(s)$ differently, what would happen?

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 $\gamma < 1$ and $|R(s,a,s')| \leq R_{max} < \infty$

- Intuition: For any state V_k and V_{k+1} can be viewed as depth k+1 expectimax results (with R and γ) in nearly identical search trees
- The difference is that on the bottom layer, V_{k+1} has actual rewards while V_k has zeros
- Each value at last layer of V_{k+1} tree is at most R_{max} in magnitude
- But everything is discounted by γ^k that far out
- So V_k and V_{k+1} are at most $\gamma^k R_{max}$ different
- So as k increases, the values converge

If we initialized $V_0(s)$ differently, what would happen?

Still converge to $V^*(s)$ as long as $|V_0(s)| < +\infty$, but may be slower

Other ways to solve Bellman Equation?

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

Treat $V^*(s)$ as variables

Solve Bellman Equation through Linear Programming

Other ways to solve Bellman Equation?

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

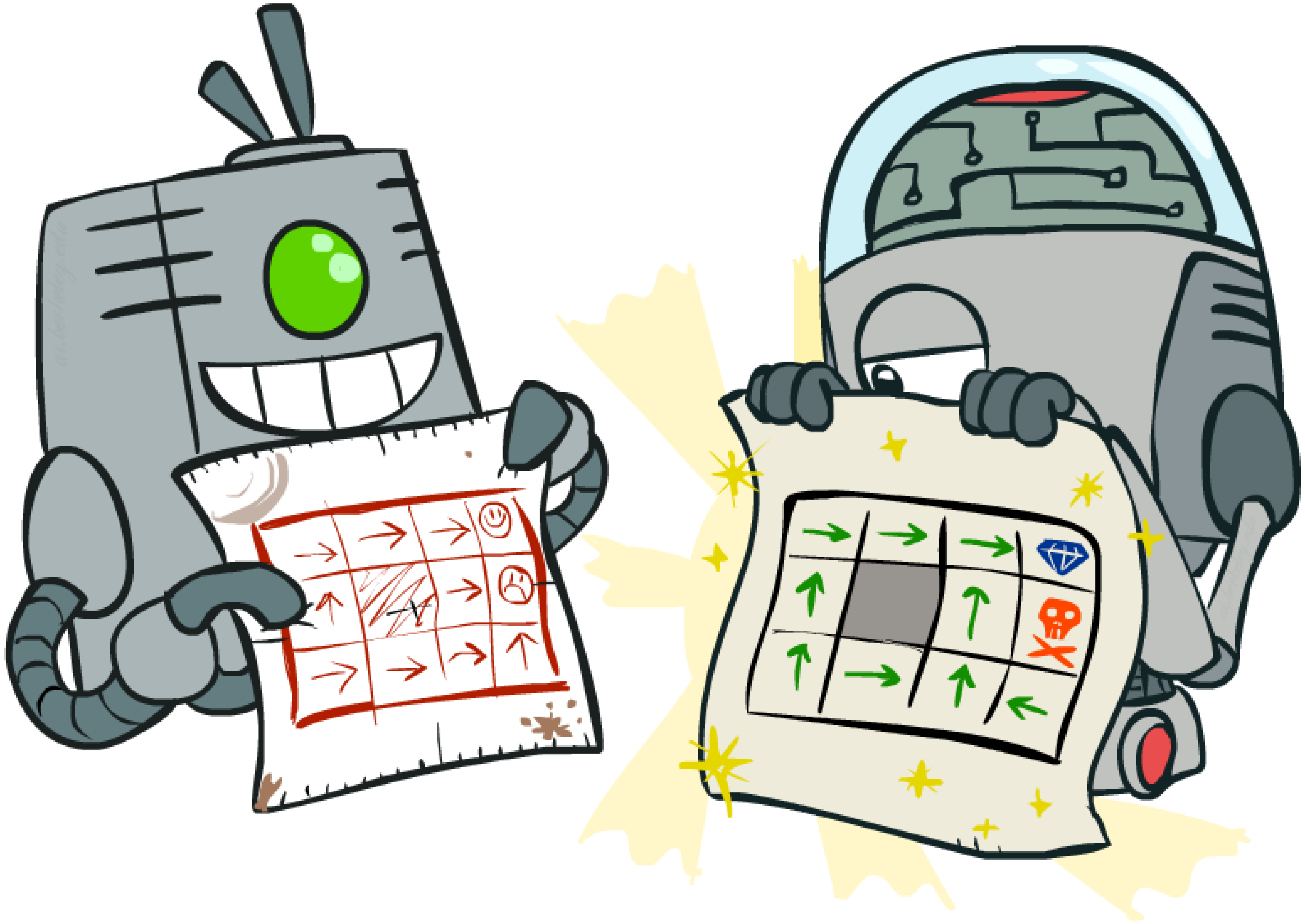
Treat $V^*(s)$ as variables

Solve Bellman Equation through Linear Programming

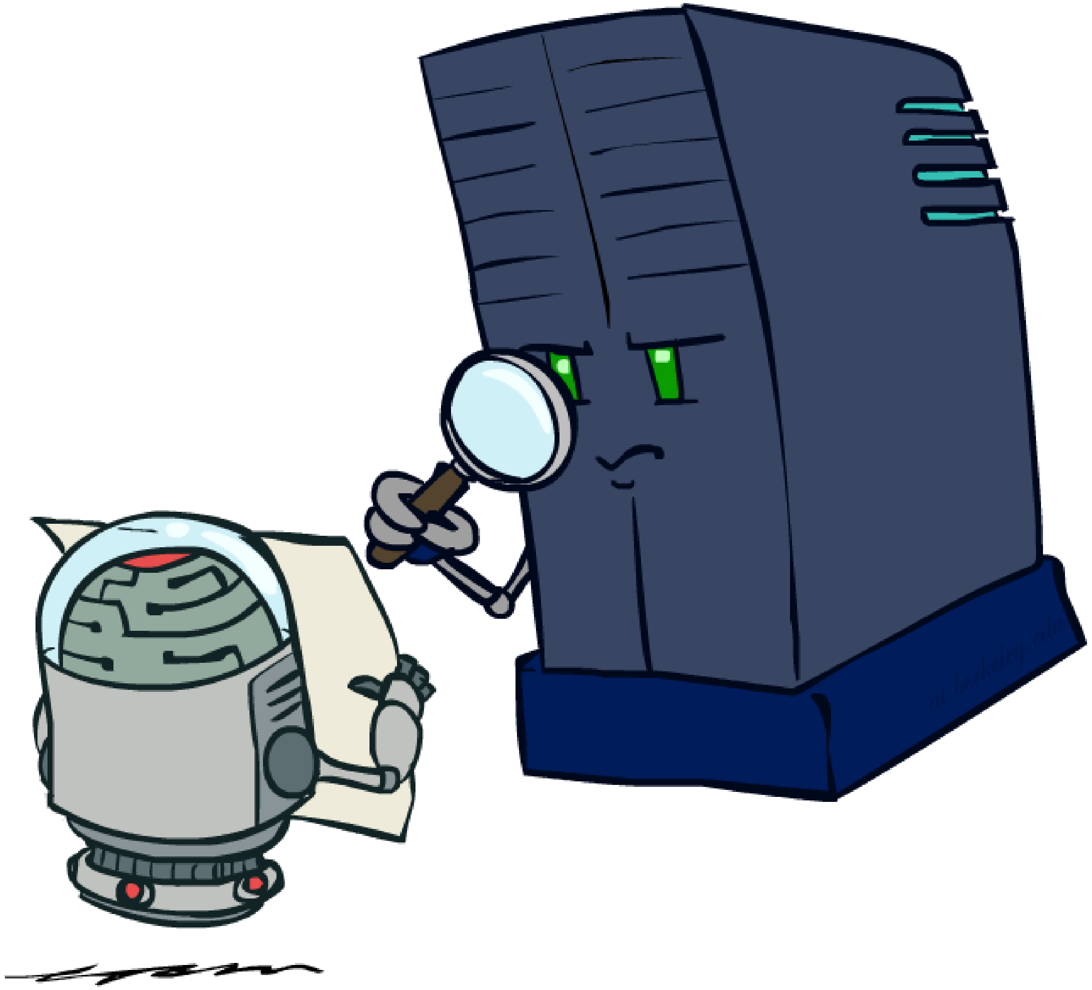
$$\min_{V^*} \sum_s V^*(s)$$

$$\text{s.t. } V^*(s) \geq \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')], \forall s, a$$

Policy Iteration for Solving MDPs

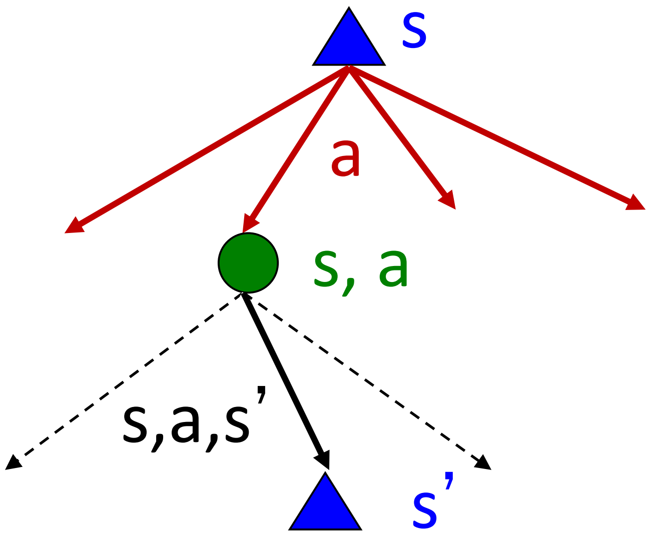


Policy Evaluation

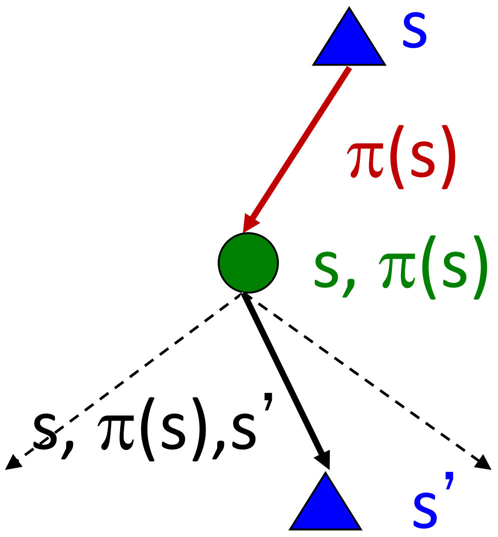


Fixed Policies

Do the optimal action



Do what π says to do



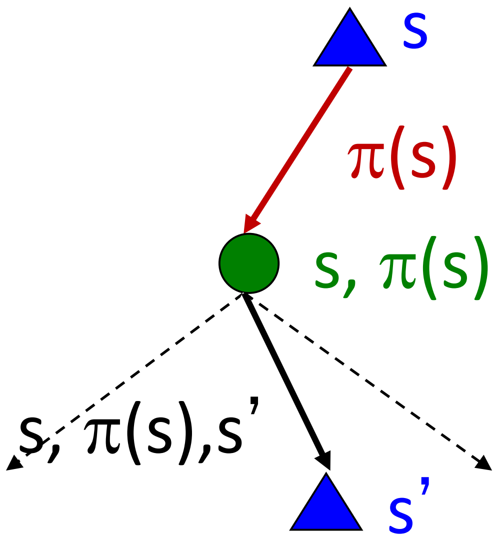
Expectimax trees max over all actions to compute the optimal values

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

Utilities for a Fixed Policy

Another basic operation: compute the utility 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 total discounted rewards starting in s and following π

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

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

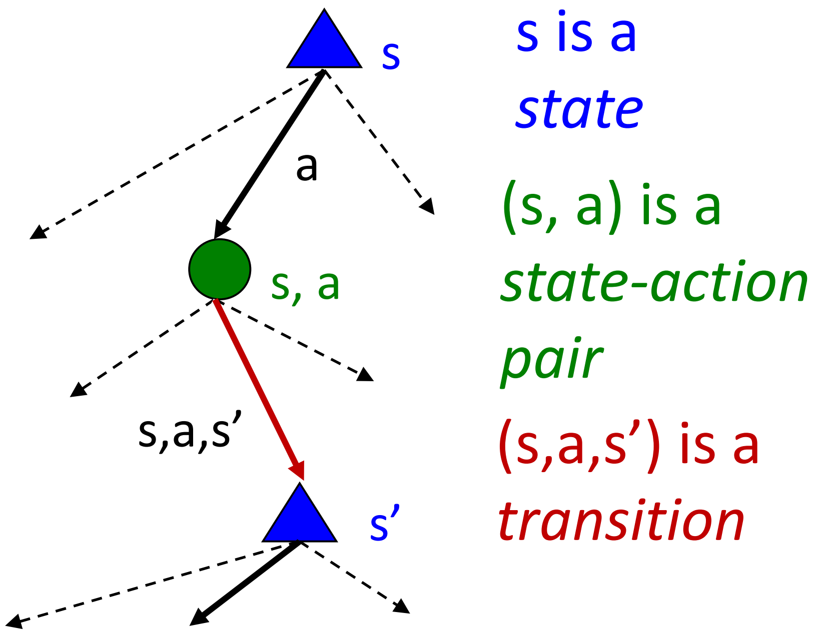
Compare

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

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

MDP Quantities

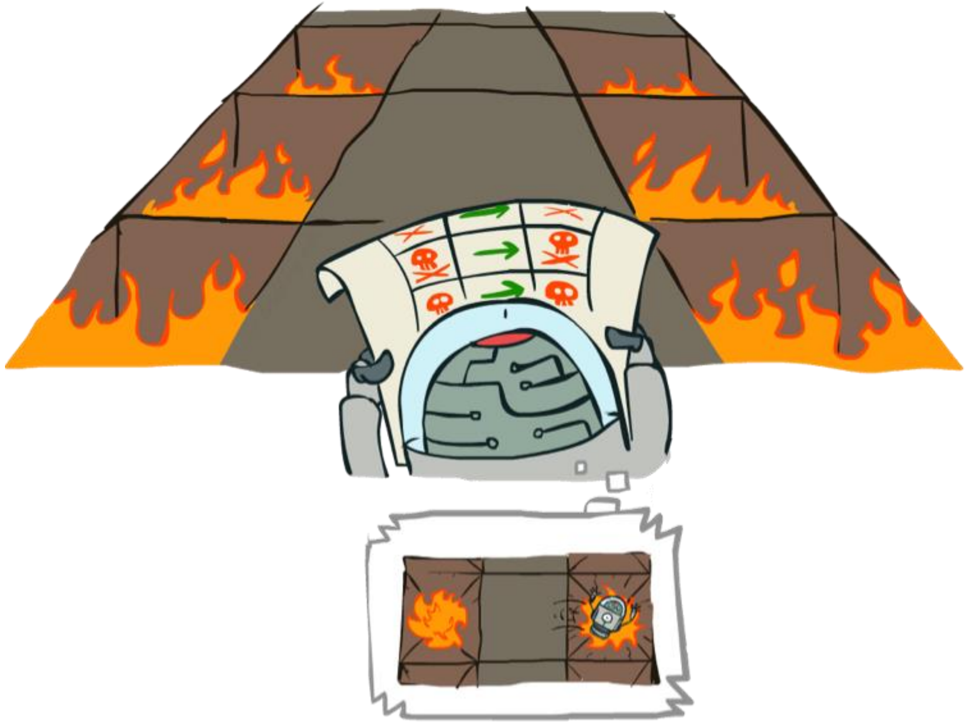
- A policy π: map of states to actions
- The optimal policy π^*: $\pi^*(s)$ = optimal action from state s
- Value function of a policy $V^{\pi}(s)$: expected utility starting in s and acting according to π
- Optimal value function V*: V*(s) = $V^{\pi^*}(s)$
- Q function of a policy $Q^{\pi}(s)$: expected utility starting out having taken action a from state s and (thereafter) acting according to π
- Optimal Q function Q* : Q*(s,a) = $Q^{\pi^*}(s)$



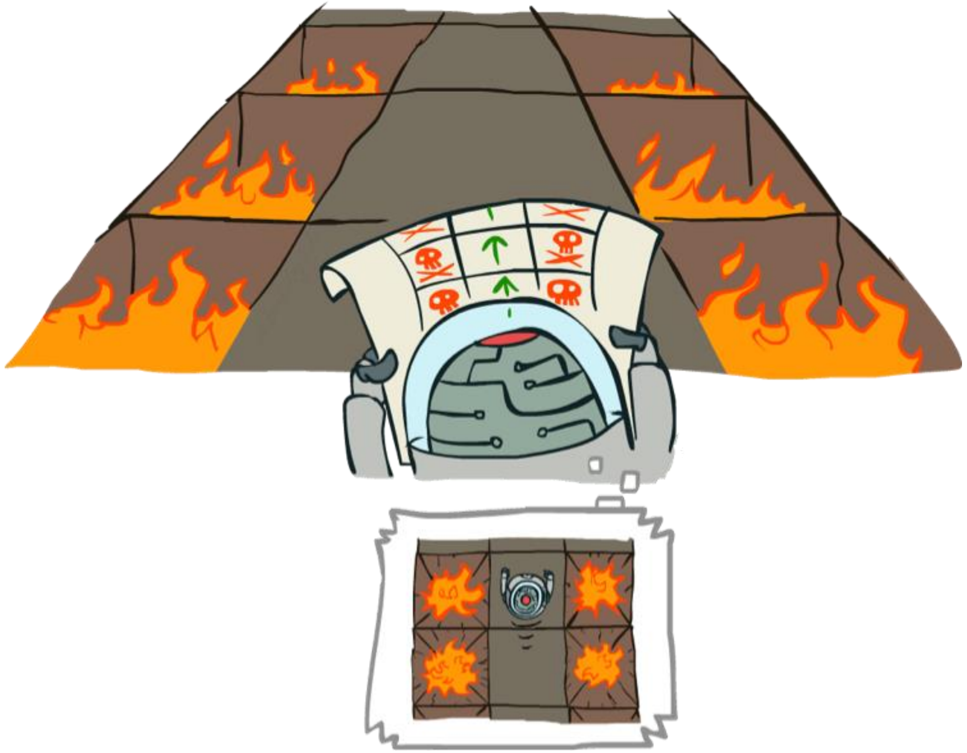
Solve MDP: Find π^*, V^* and/or Q^*

Example: Policy Evaluation

Always Go Right



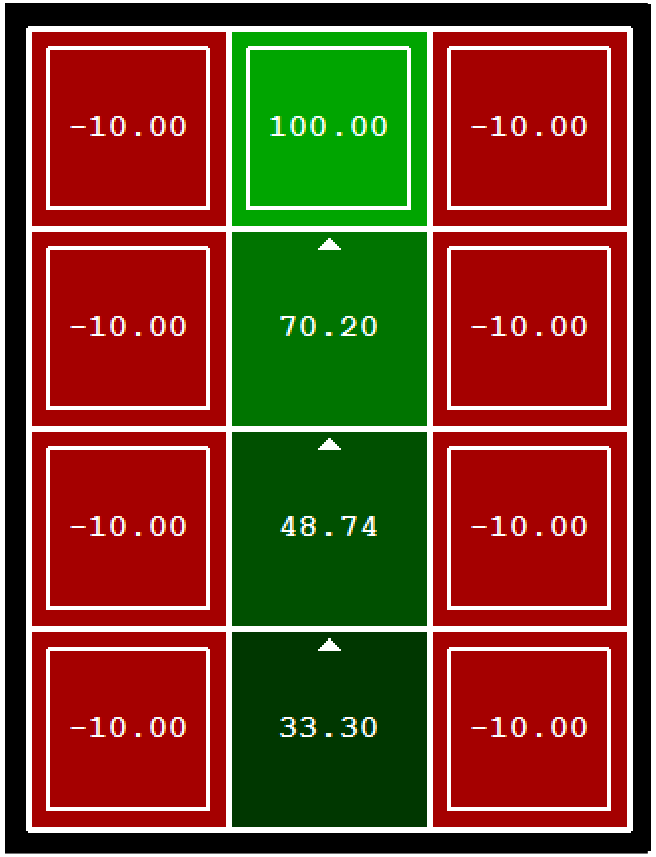
Always Go Forward



Example: Policy Evaluation

Always Go Right

Always Go Forward



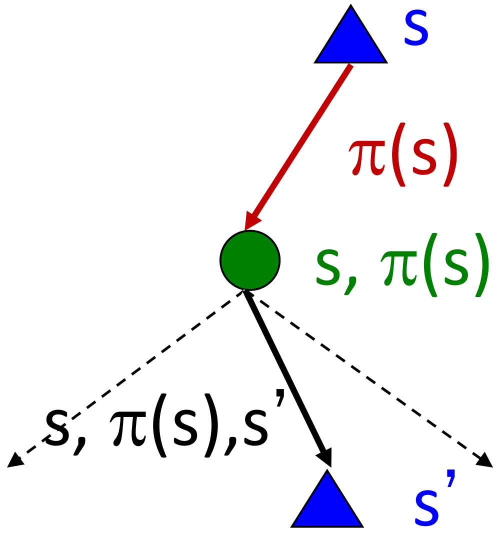
Policy Evaluation

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

Idea 1: Turn recursive Bellman equations into updates
(like value iteration)

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

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



Policy Evaluation

Idea 2: Bellman Equation w.r.t. a given policy π defines a linear system

- Solve with your favorite linear system solver

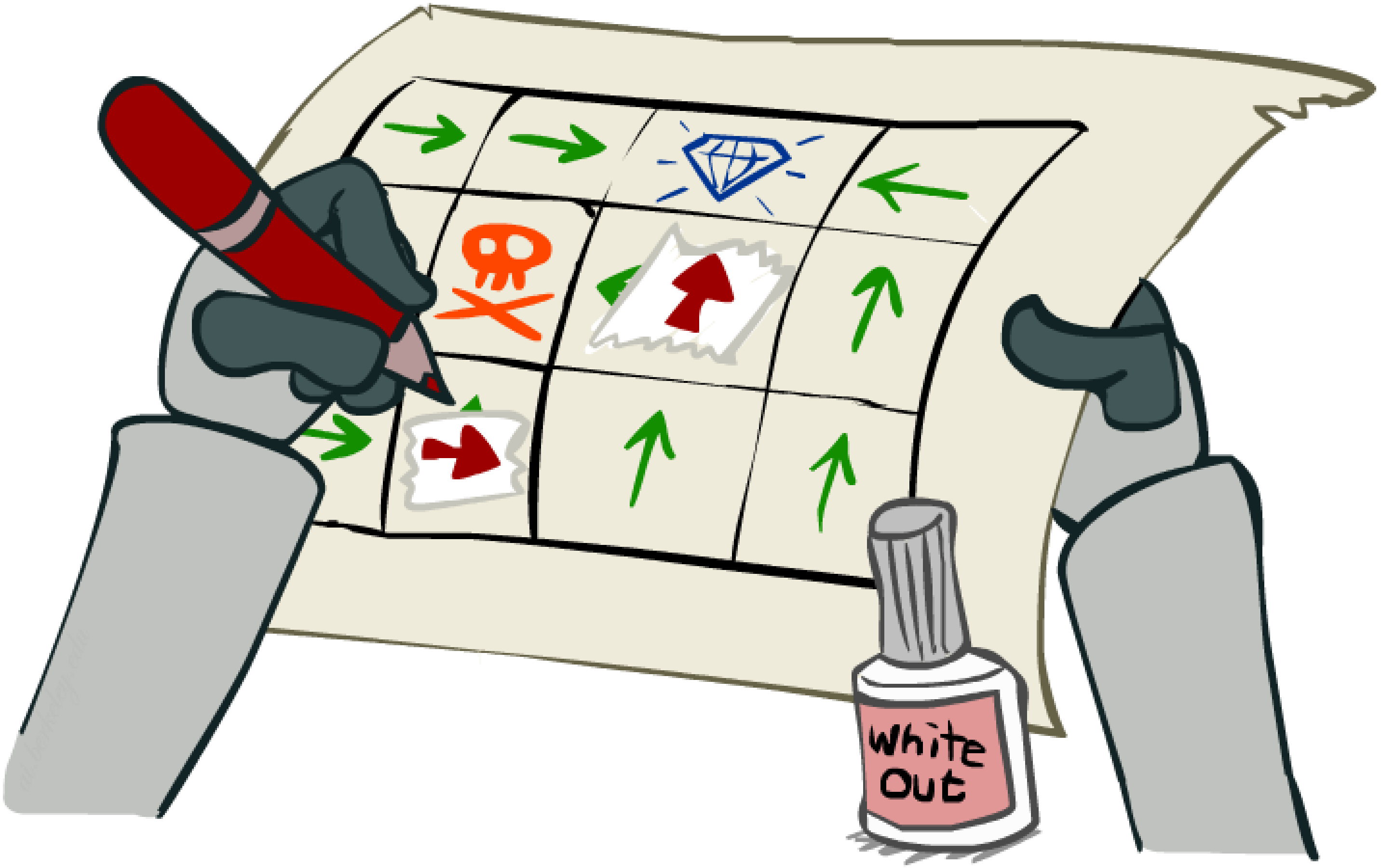
$$V^\pi(s) = \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^\pi(s')]$$

Treat $V^\pi(s)$ as variables

How many variables?

How many constraints?

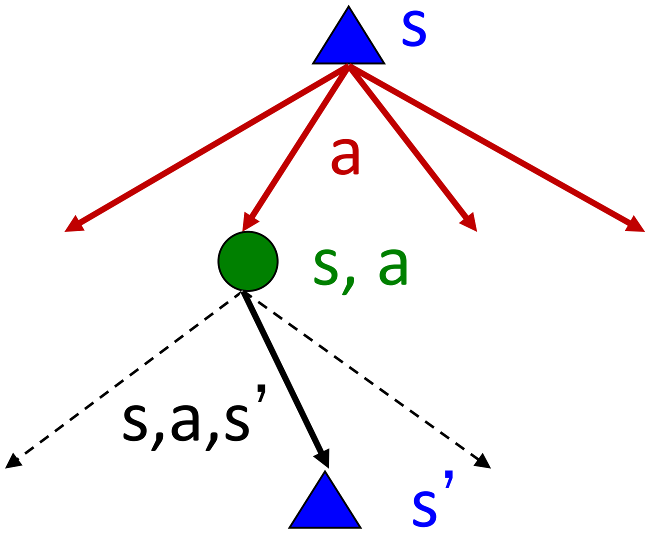
Policy Iteration



Problems with Value Iteration

Value iteration repeats the Bellman updates:

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



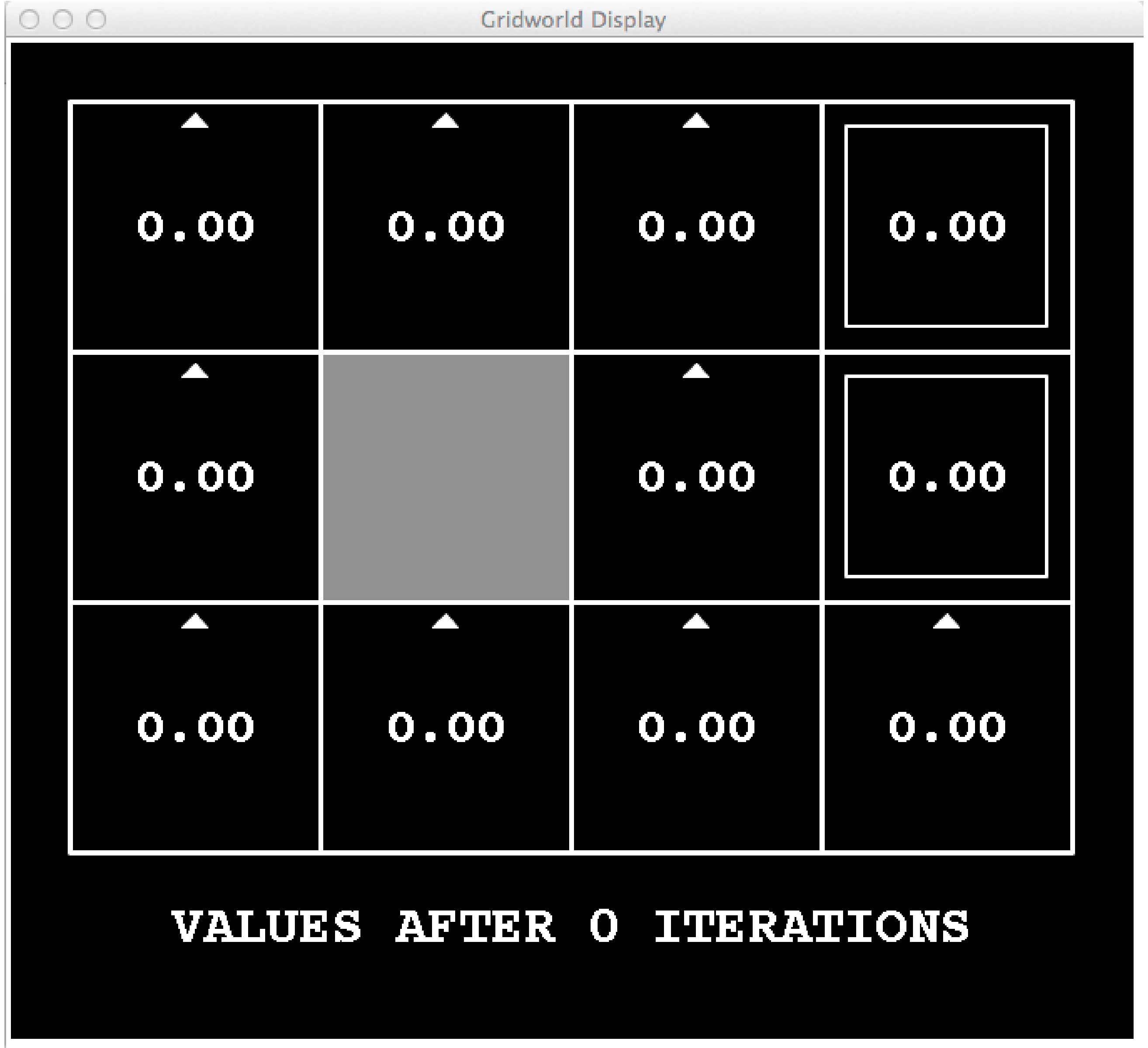
Problem 1: It's slow – O($|S|^2|A|$) per iteration

Problem 2: The "max" at each state rarely changes

Problem 3: The policy often converges long before the values

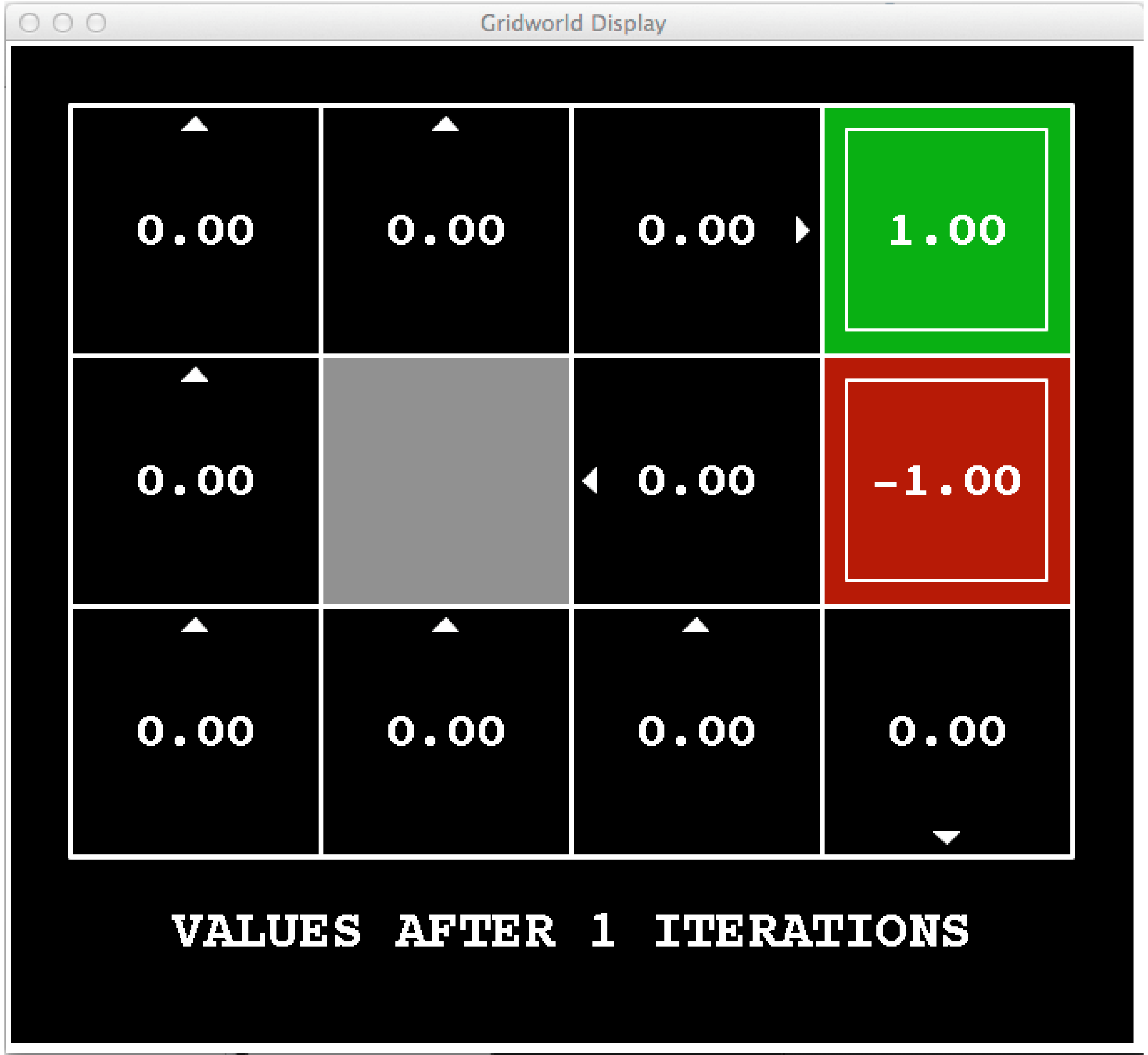
[Demo: value iteration (L9D2)]

k=0



Noise = 0.2
Discount = 0.9
Living reward = 0

k=1



Noise = 0.2
Discount = 0.9
Living reward = 0

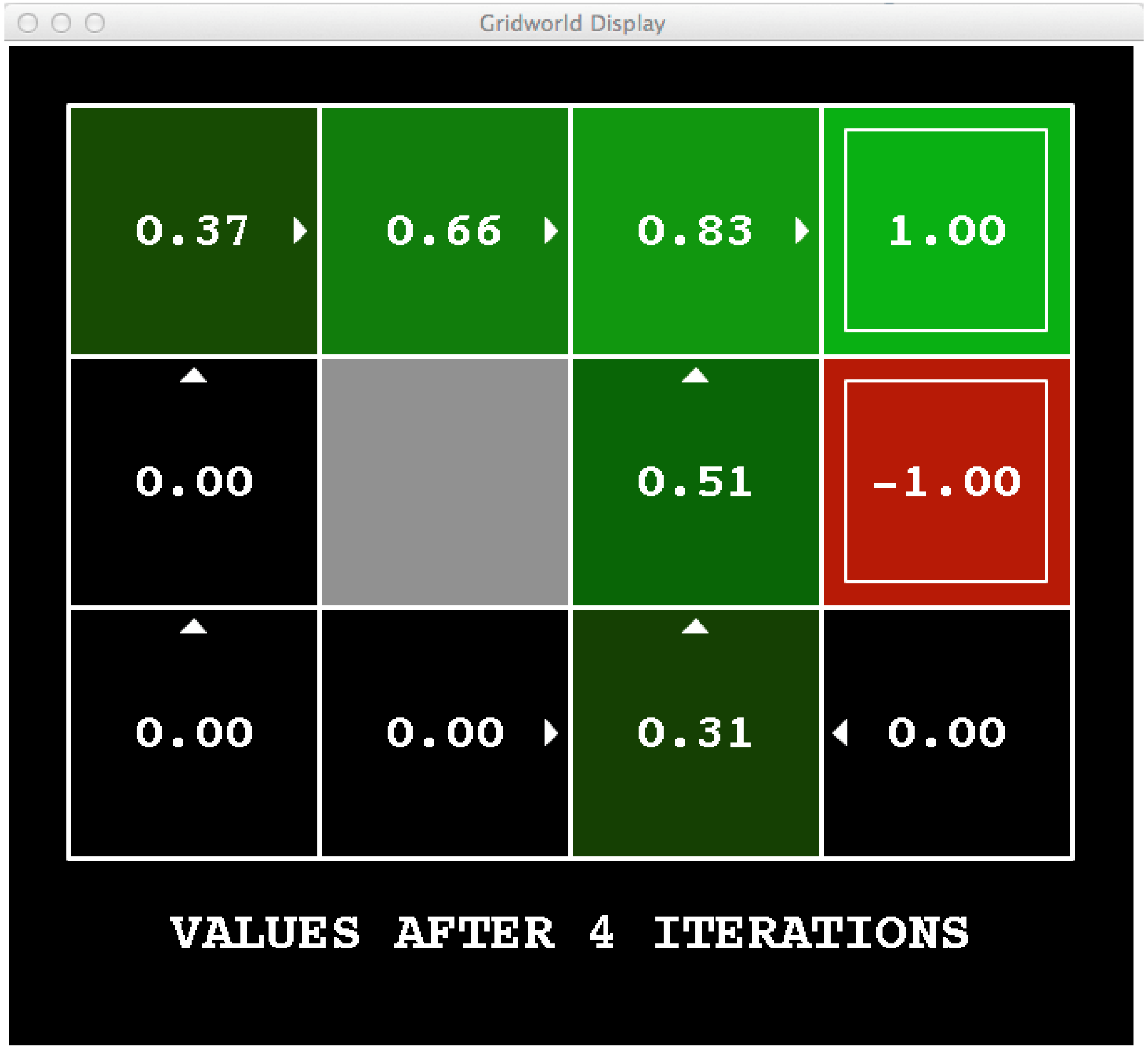
k=2

Noise = 0.2
Discount = 0.9
Living reward = 0

k=3

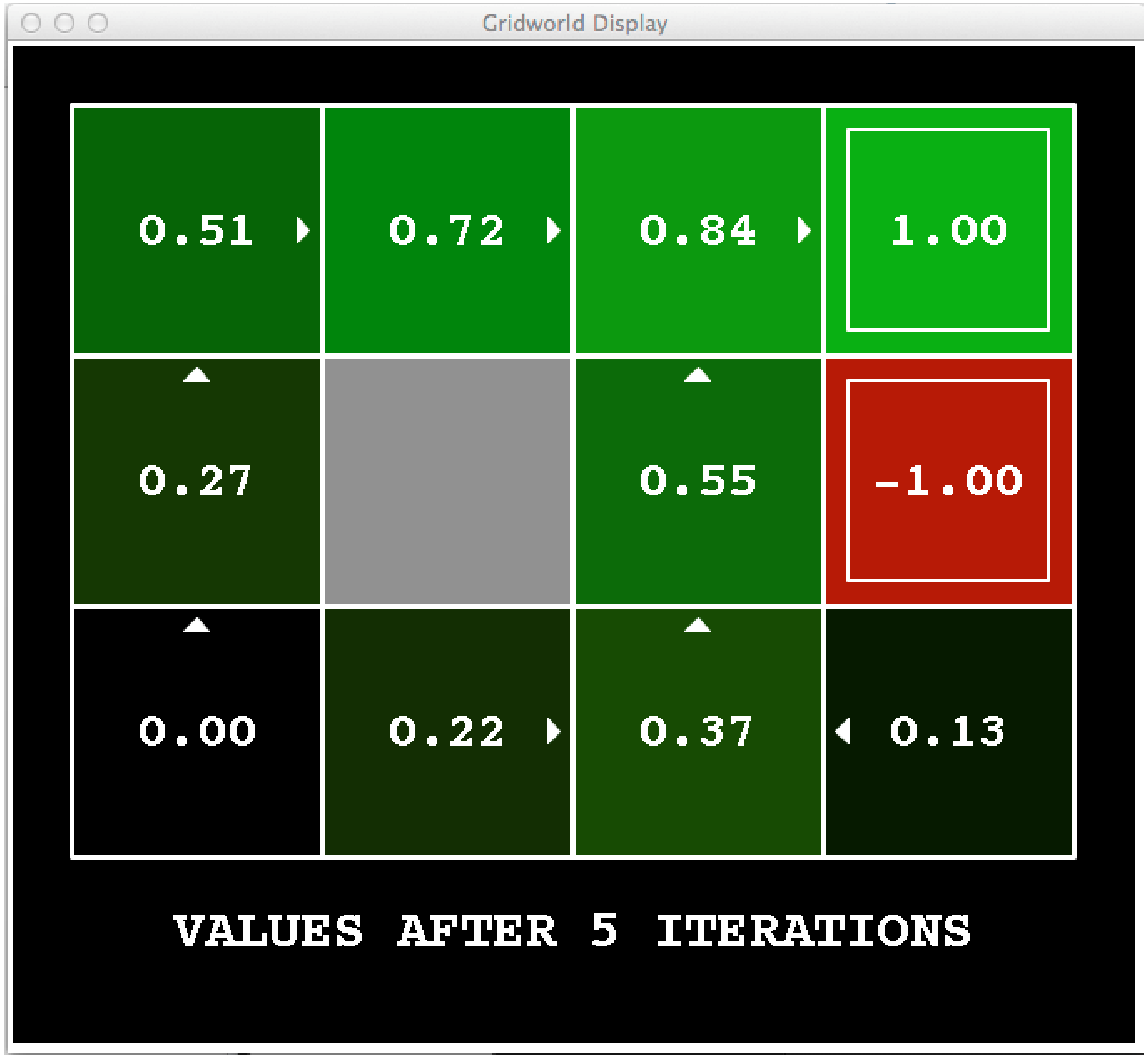
Noise = 0.2
Discount = 0.9
Living reward = 0

k=4



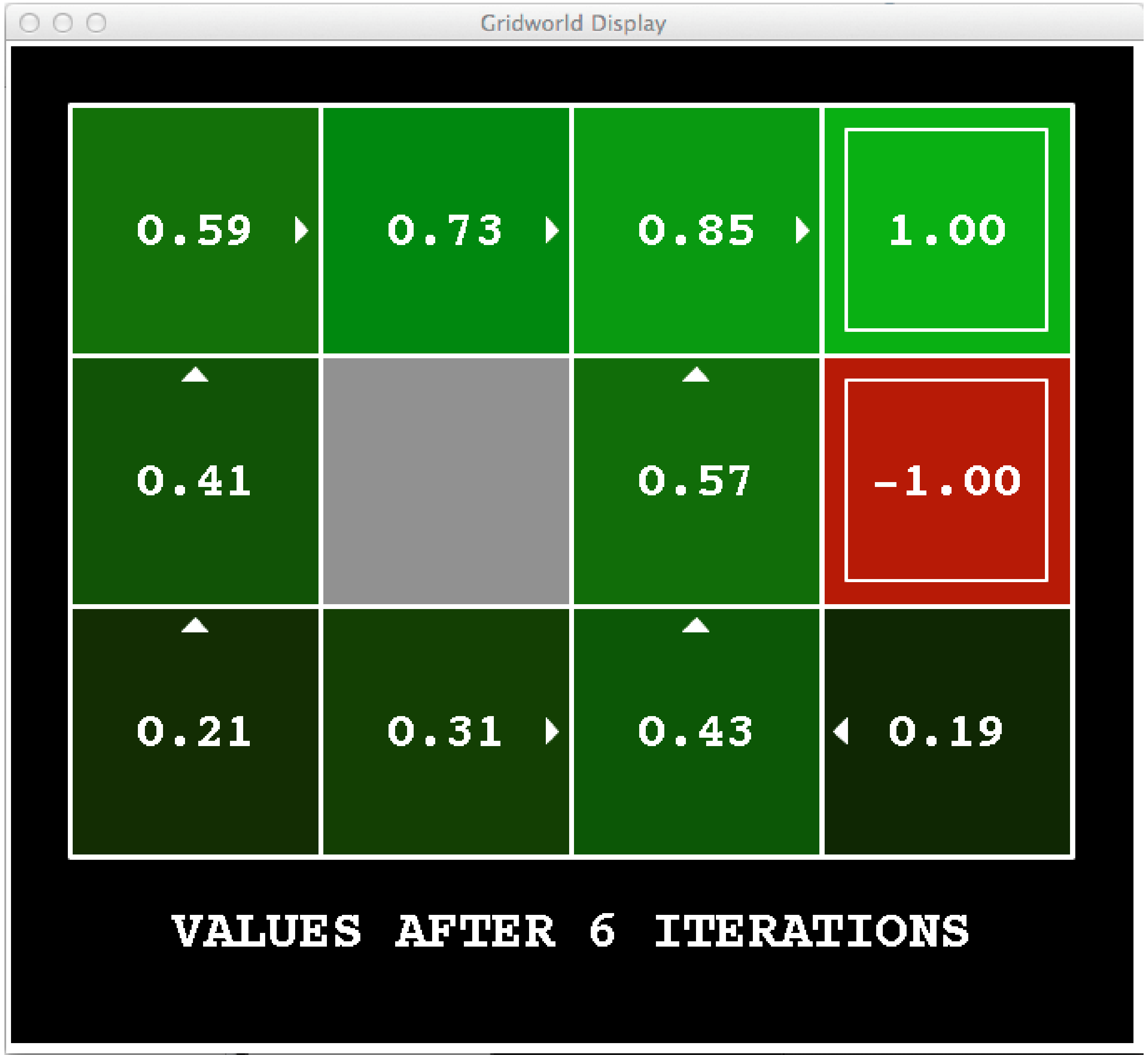
Noise = 0.2
Discount = 0.9
Living reward = 0

k=5



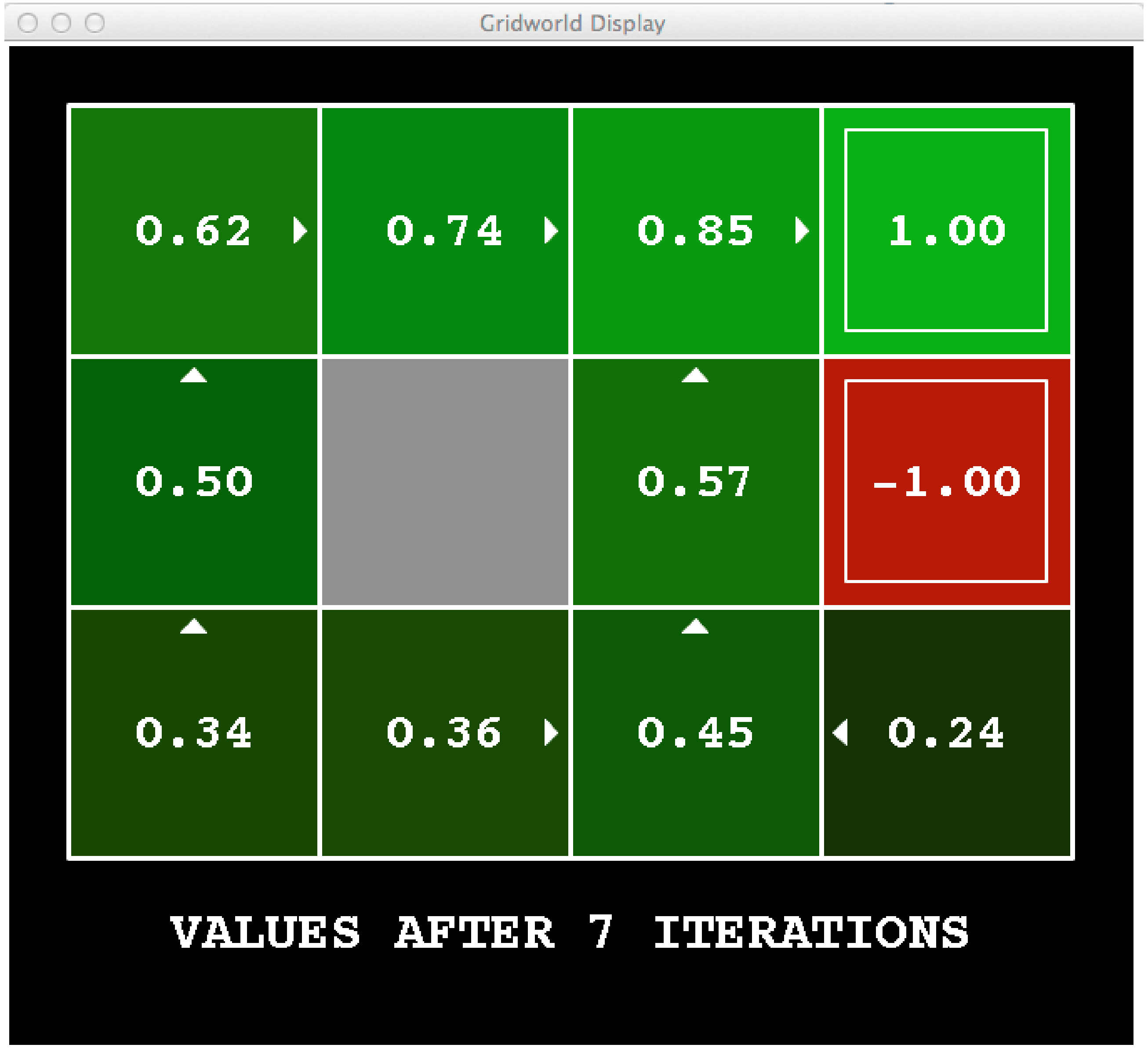
Noise = 0.2
Discount = 0.9
Living reward = 0

k=6



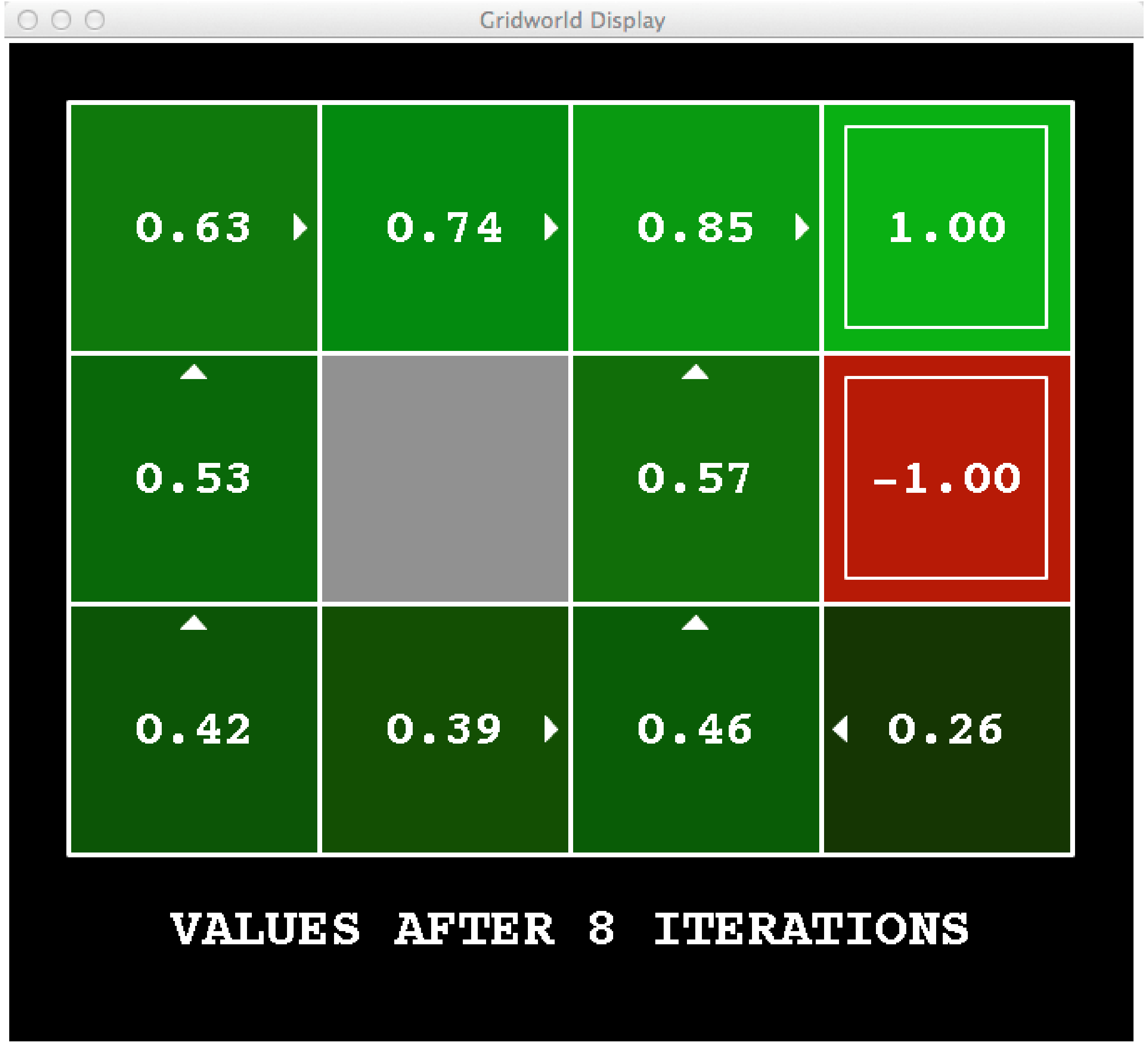
Noise = 0.2
Discount = 0.9
Living reward = 0

k=7



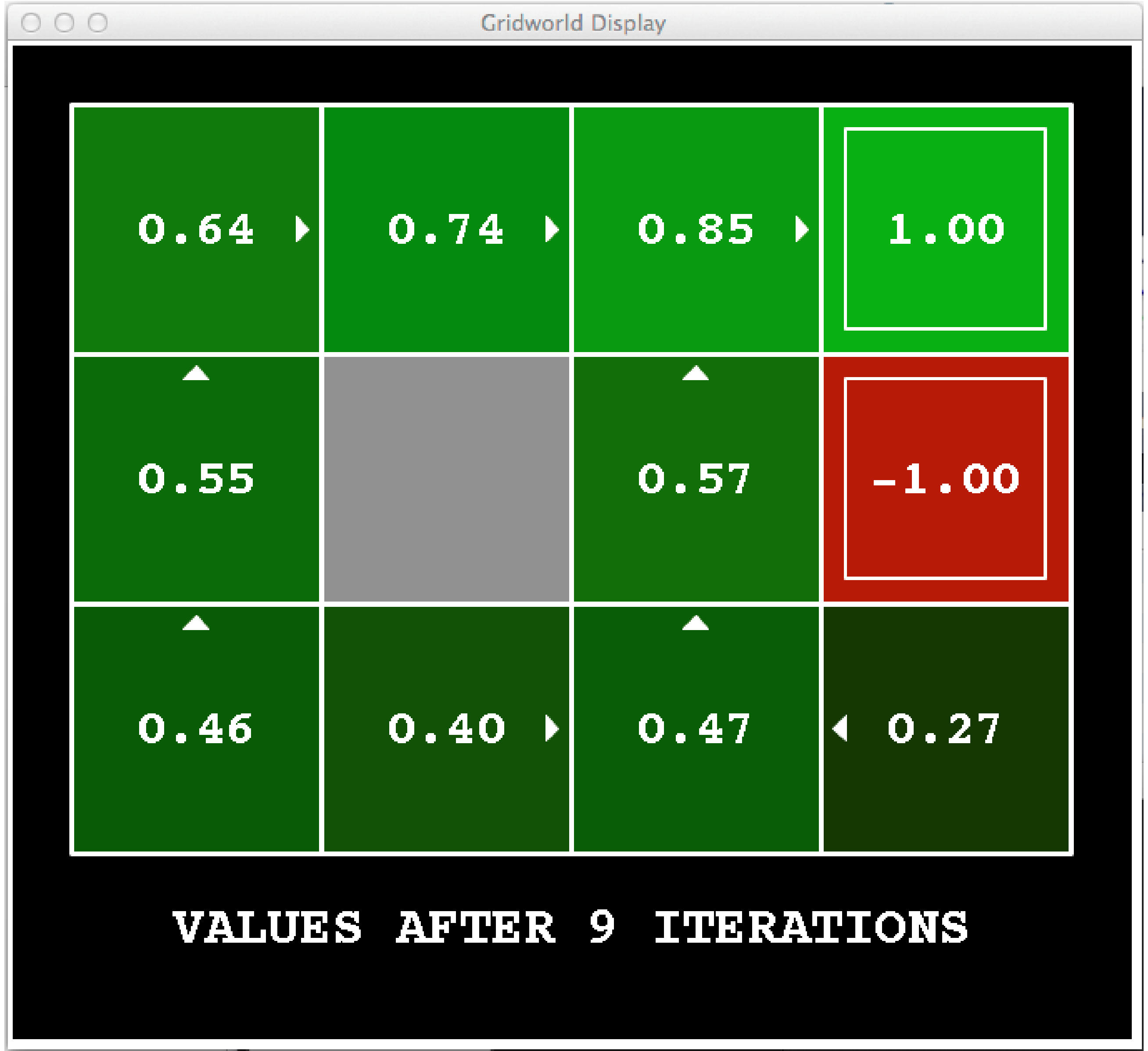
Noise = 0.2
Discount = 0.9
Living reward = 0

k=8



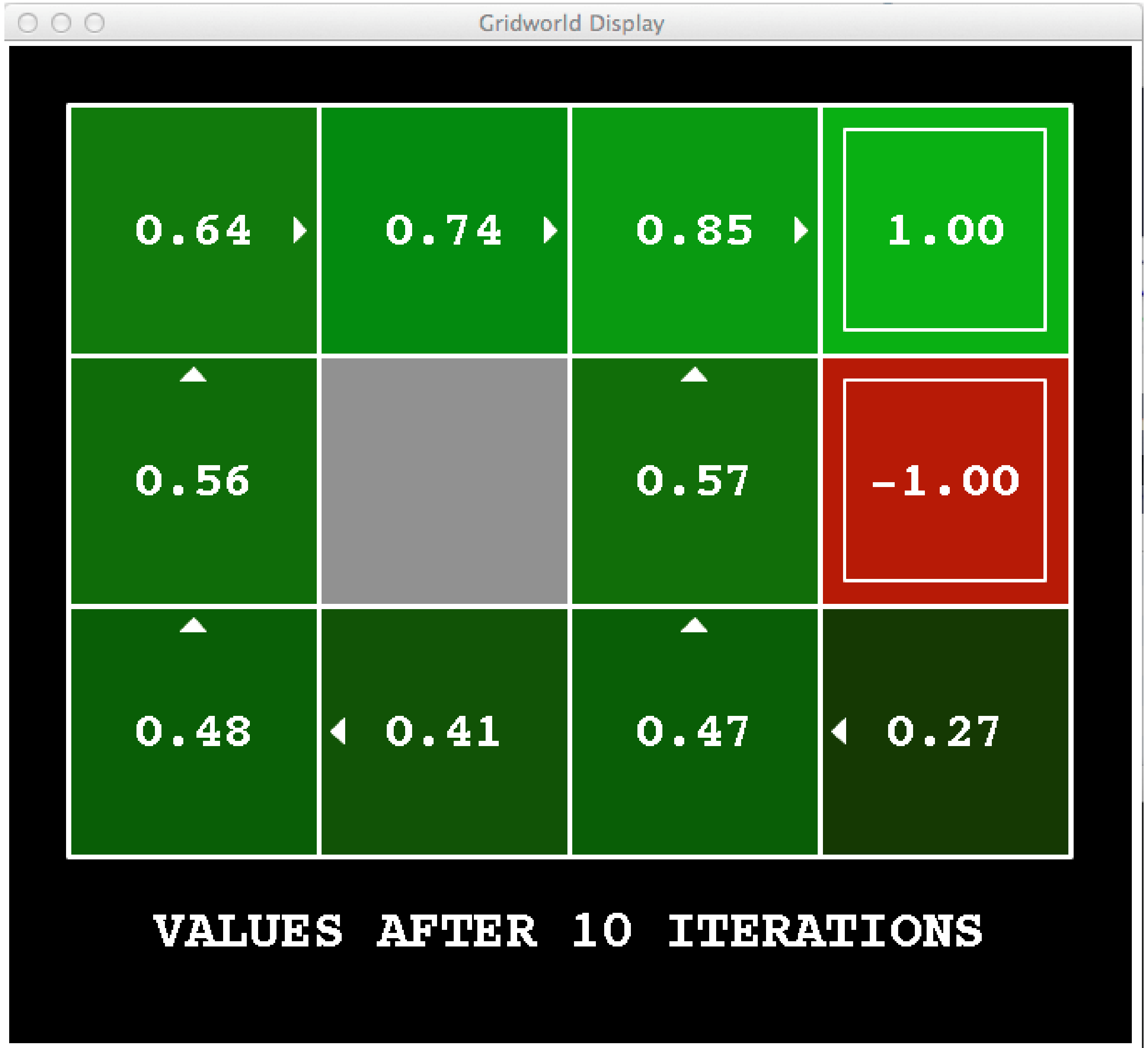
Noise = 0.2
Discount = 0.9
Living reward = 0

k=9



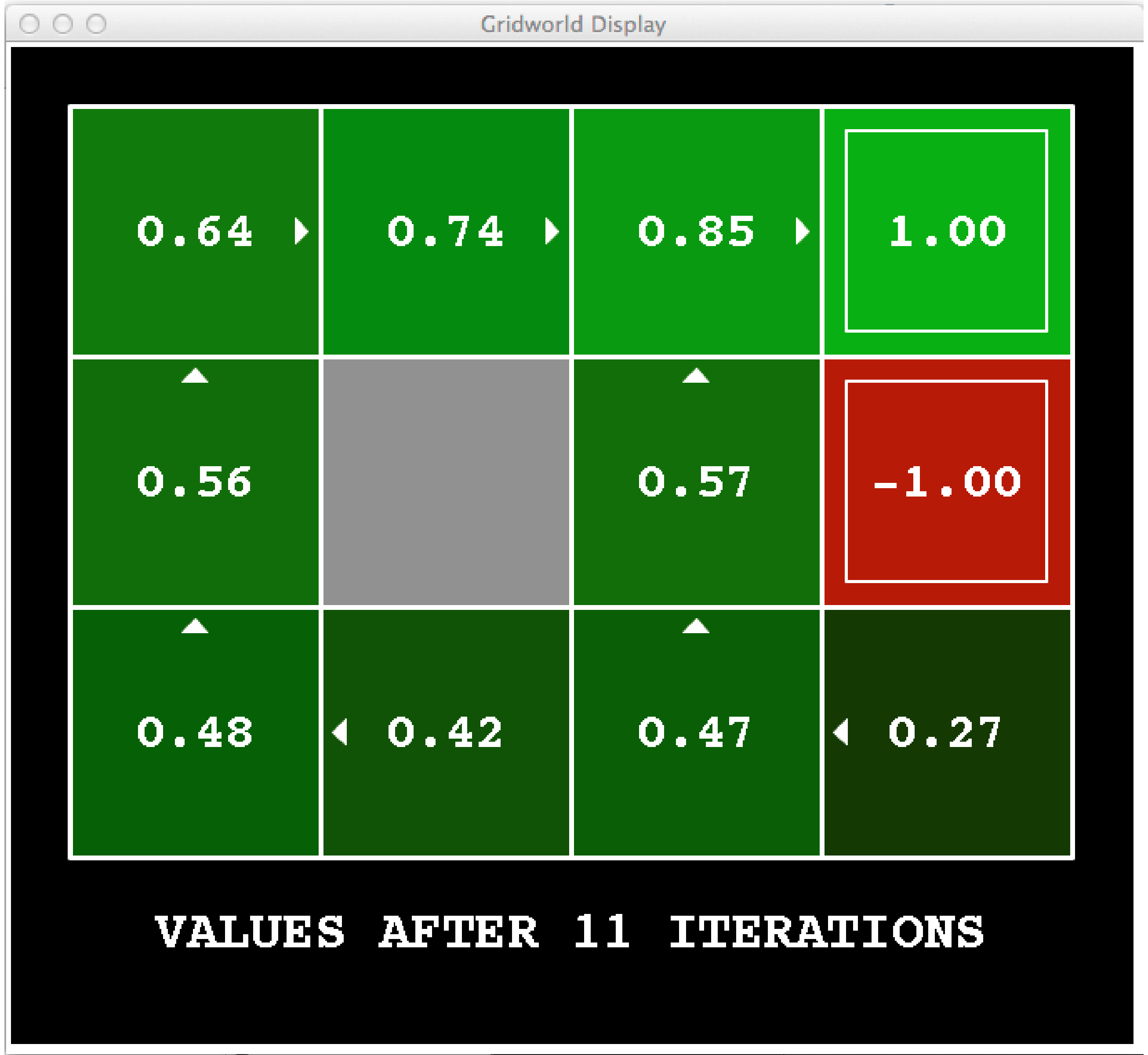
Noise = 0.2
Discount = 0.9
Living reward = 0

k=10



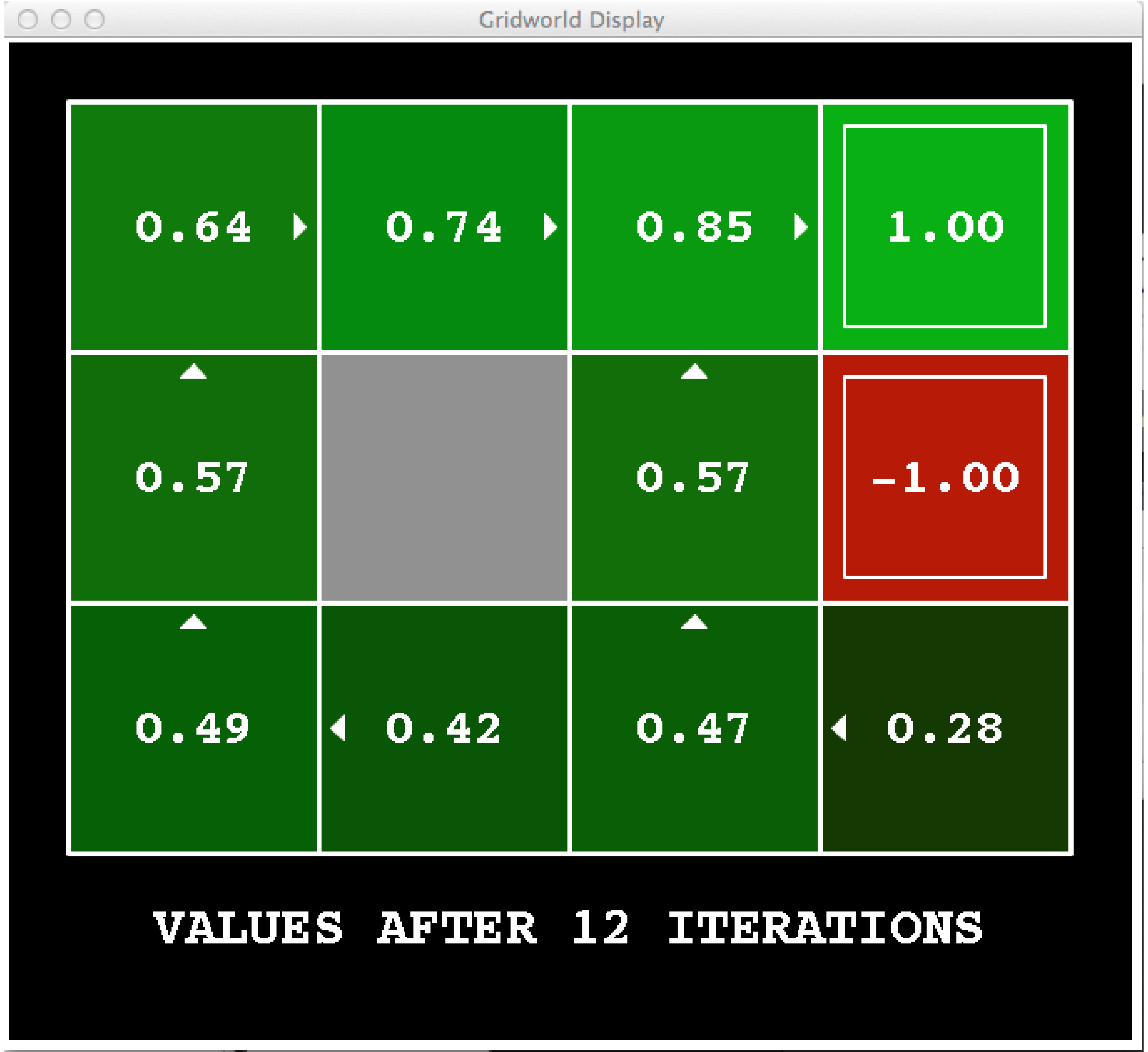
Noise = 0.2
Discount = 0.9
Living reward = 0

k=11



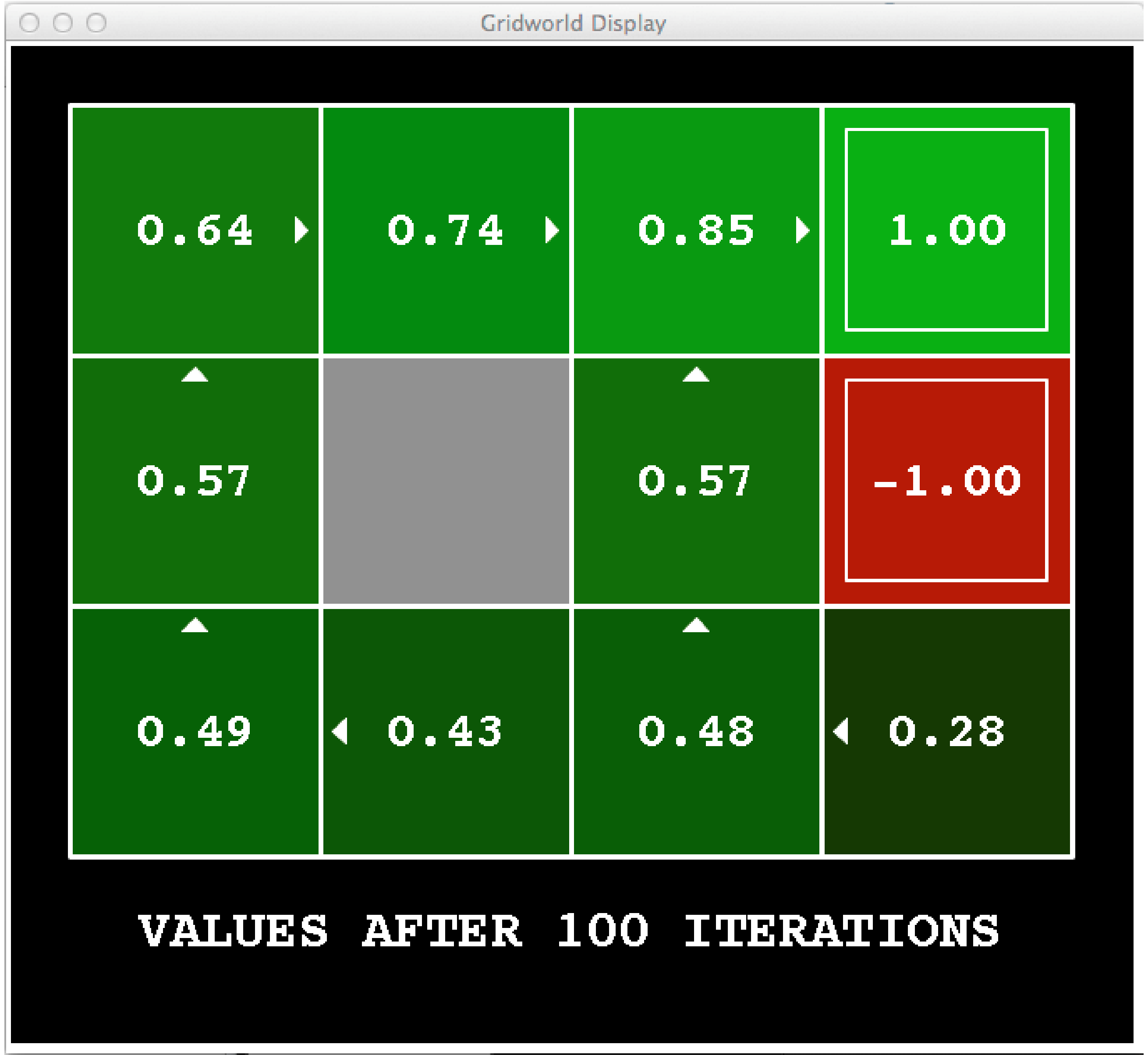
Noise = 0.2
Discount = 0.9
Living reward = 0

k=12



Noise = 0.2
Discount = 0.9
Living reward = 0

k=100



Noise = 0.2
Discount = 0.9
Living reward = 0

Policy Iteration

Alternative approach for optimal values:

- Step 1: Policy evaluation: calculate utilities for some fixed policy (may not be optimal!) until convergence
- Step 2: Policy improvement: update policy using one-step look-ahead with resulting converged (but not optimal!) utilities as future values
- Repeat steps until policy converges

This is policy iteration

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

Policy Iteration

Policy Evaluation: For fixed current policy π, find values w.r.t. the policy

- Iterate until values converge:

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

Policy Improvement: For fixed values, get a better policy with one-step look-ahead:

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

Similar to how you derive optimal policy π^* given optimal value V^*

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 utilities with fixed policy (each pass is fast because we consider only one action, not all of them)
- After the policy is evaluated, a new policy is chosen (slow like a value iteration pass)
- The new policy will be better (or we’re done)

(Both are dynamic programs for solving MDPs)

Summary: MDP Algorithms

So you want to….

- Turn values into a policy: use one-step lookahead
- Compute optimal values: use value iteration or policy iteration
- Compute values for a particular policy: use policy evaluation

These all look the same!

- They basically are – they are all variations of Bellman updates
- They all use one-step lookahead expectimax fragments
- They differ only in whether we plug in a fixed policy or max over actions

MDP Notation

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

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

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

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

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

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

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

MDP Notation

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

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

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

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

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

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

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

MDP Notation

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

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

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

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

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

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

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

MDP Notation

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

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

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

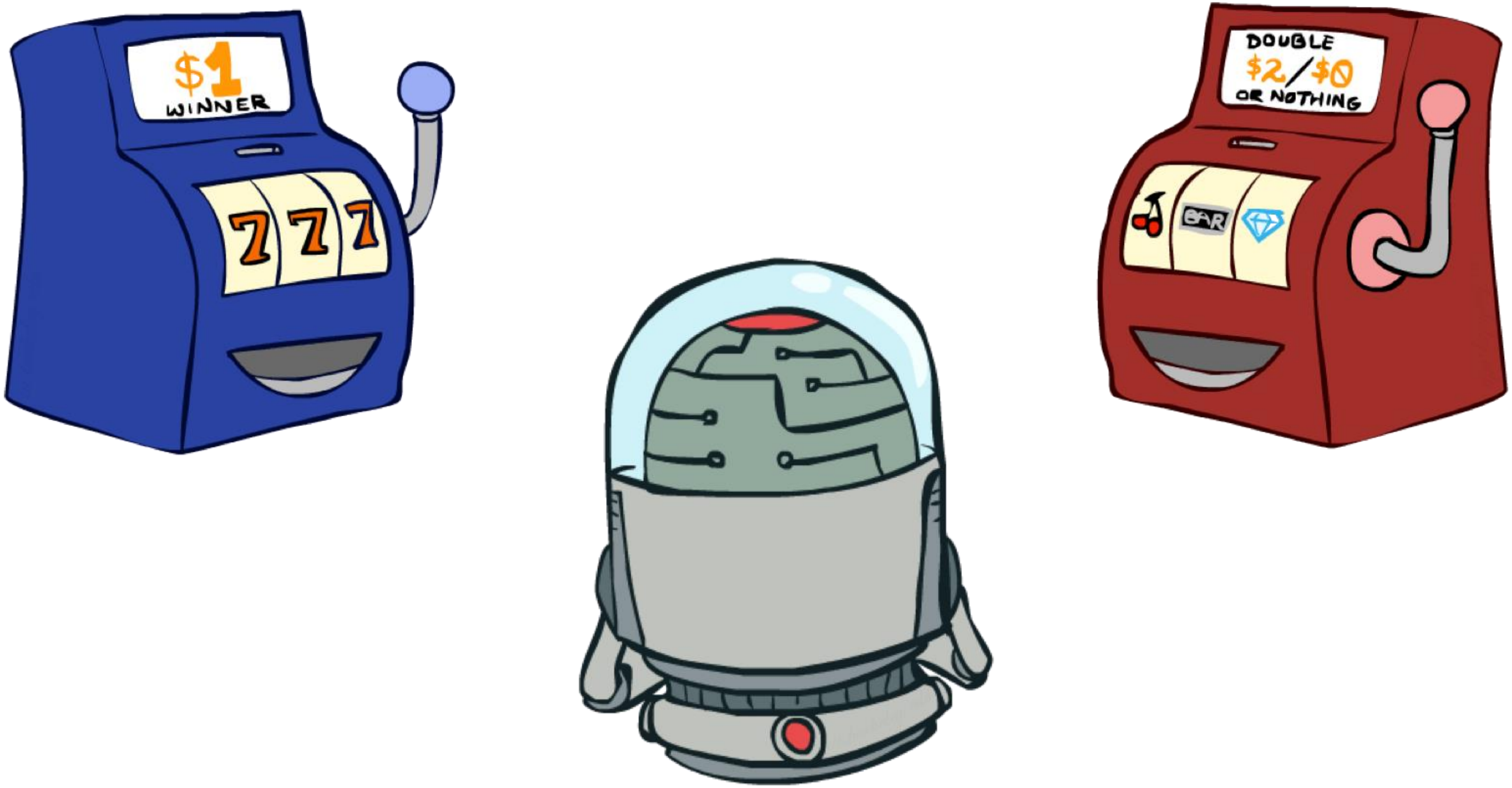
Q-iteration: $Q_{k+1}(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')], \quad \forall\, s,a$

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

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

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

Double Bandits



Double-Bandit MDP

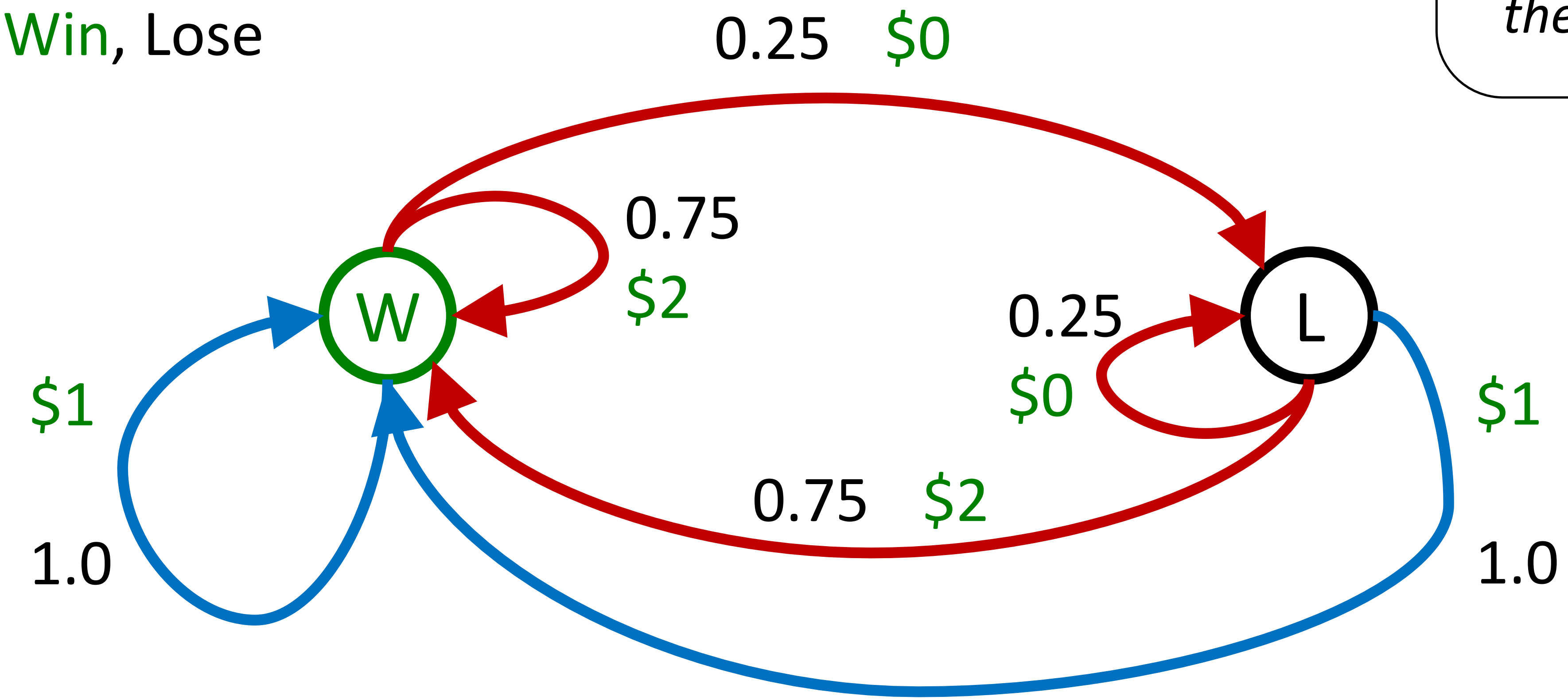
No discount

100 time steps

Both states have the same value

Actions: *Blue*, *Red*

States: Win, Lose



Actually a simple MDP where the current state does not impact transition or reward:
$P(s'|s,a) = P(s'|a)$ and $R(s,a,s') = R(a,s')$

Offline Planning

Solving MDPs is offline planning

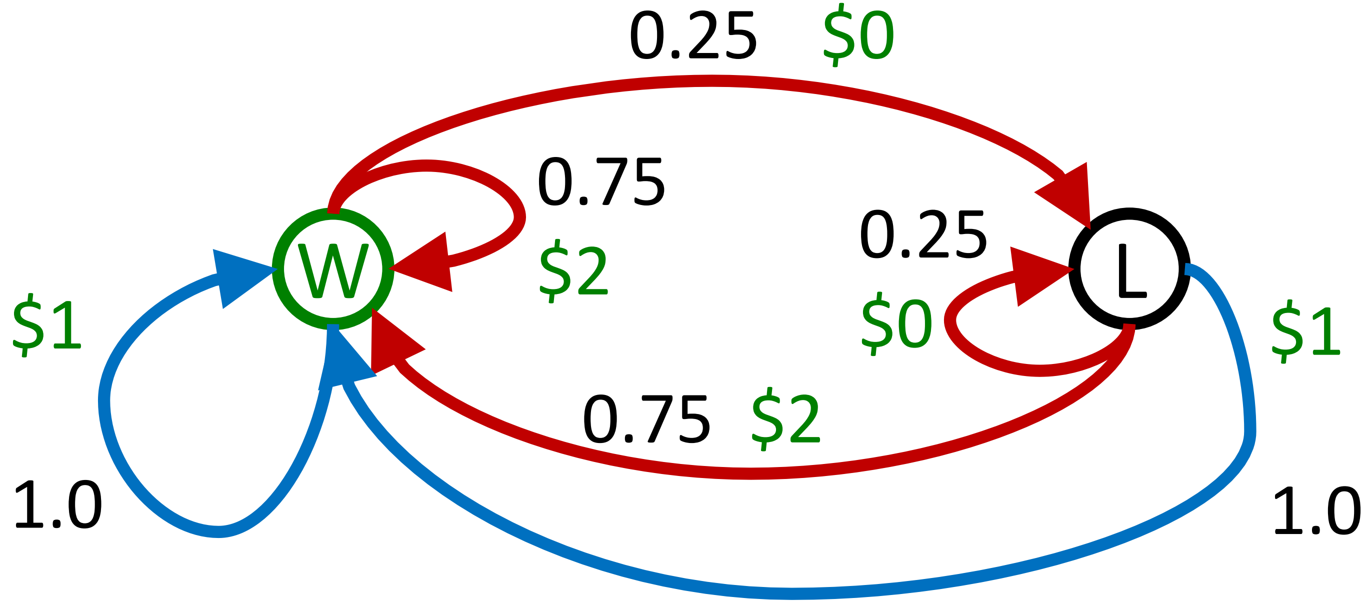
- You determine all quantities through computation
- You need to know the details of the MDP
- You do not actually play the game!

No discount

100 time steps

Both states have the same value

	Value
Play Red	150
Play Blue	100



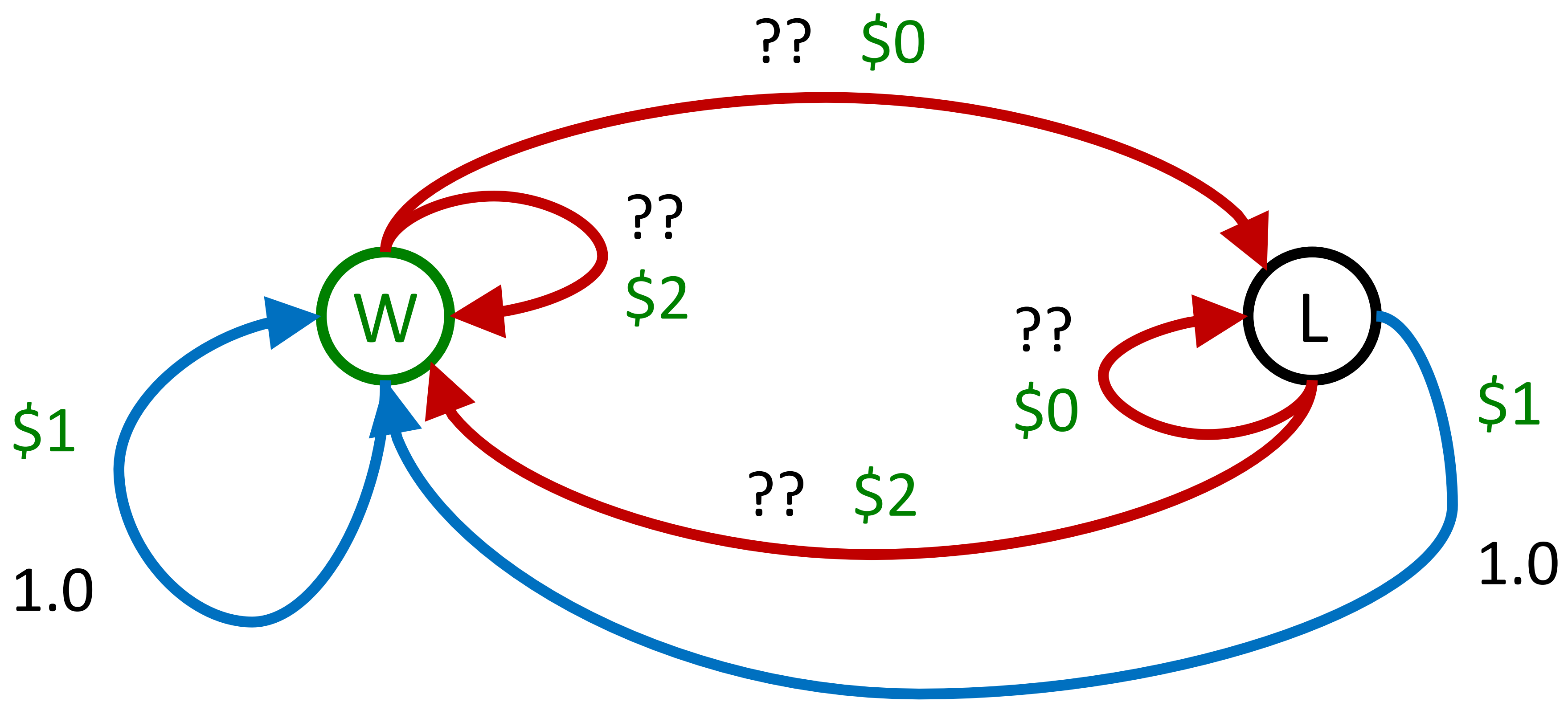
Let’s Play!

$2 $2 $0 $2 $2

$2 $2 $0 $0 $0

Online Planning

Rules changed! Red's win chance is different.



Let’s Play!

$0 $0 $0 $2 $0

$2 $0 $0 $0 $0

What Just Happened?

That wasn't planning, it was learning!

- Specifically, reinforcement learning
- There was an MDP, but you couldn't solve it with just computation
- You needed to actually act to figure it out

Important ideas in reinforcement learning that came up

- Exploration: you have to try unknown actions to get information
- Exploitation: eventually, you have to use what you know
- Regret: even if you learn intelligently, you make mistakes
- Sampling: because of chance, you have to try things repeatedly
- Difficulty: learning can be much harder than solving a known MDP

Next Time: Reinforcement Learning!