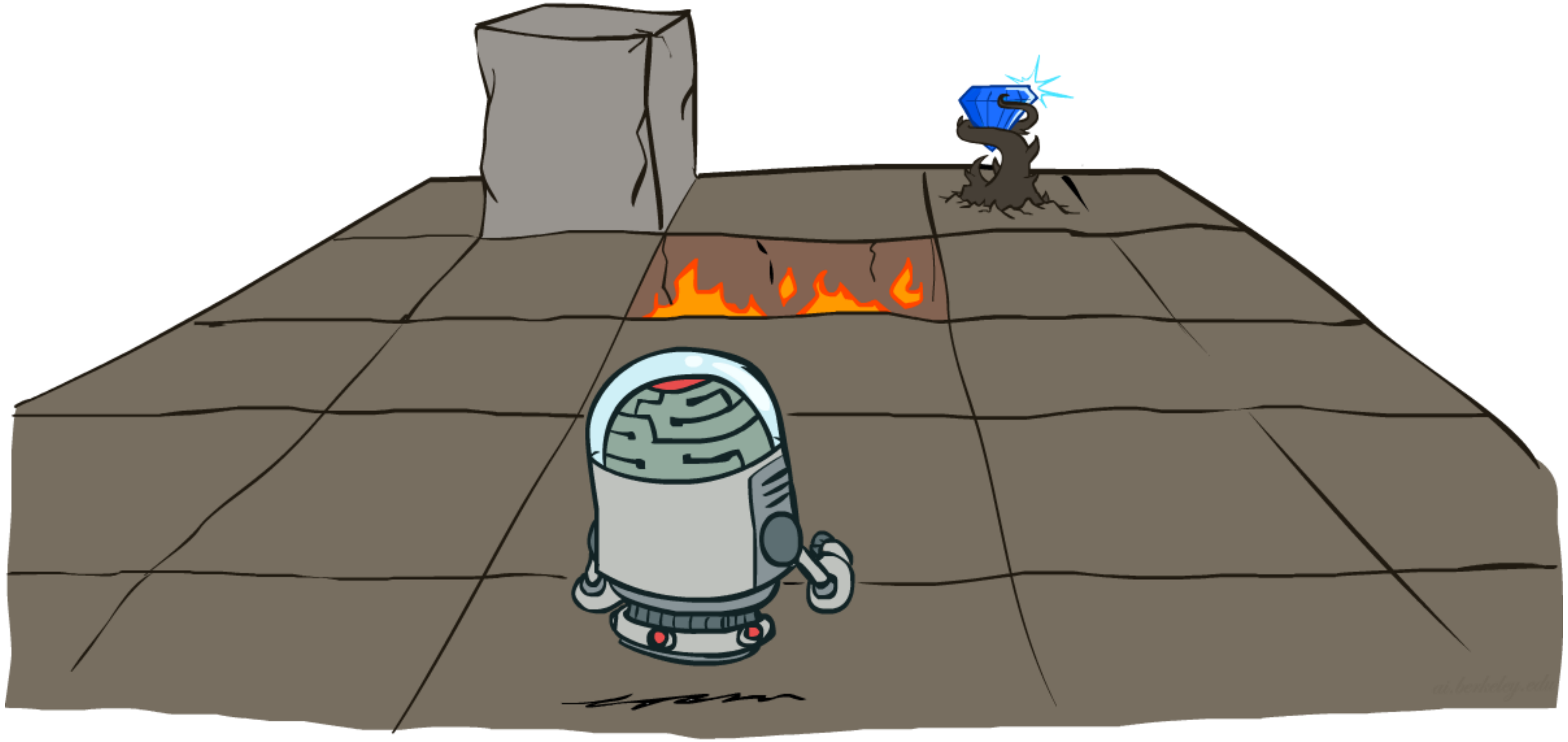
Intro/Refresher on MDPs and Reinforcement Learning



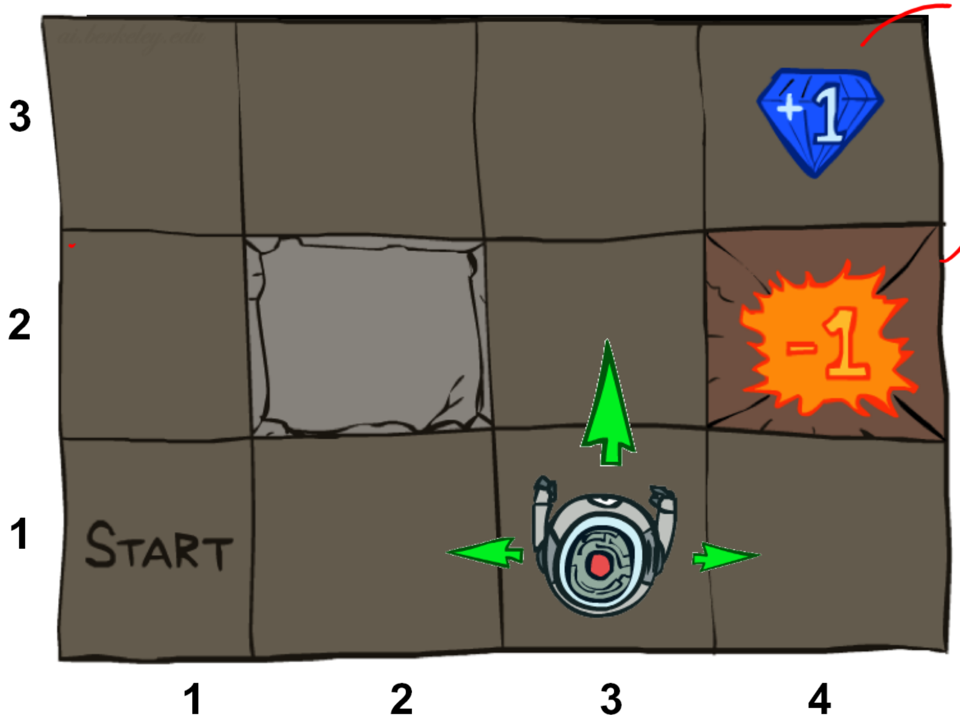
Instructor: Daniel Brown

University of Utah

[Based on slides created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley. http://ai.berkeley.edu.]

Markov Decision Processes (MDP)

- An MDP is defined by: $s \xrightarrow{a} s'$
  - A set of states s ∈ S
  - A set of actions a ∈ A
  - A transition function T(s, a, s')
    - Probability that a from s leads to s', i.e., P(s' | s, a)
    - Also called the model or the dynamics
  - A reward function R(s, a, s')
    - Sometimes just R(s), R(s,a), or R(s')
  - A start state
  - Maybe a terminal state

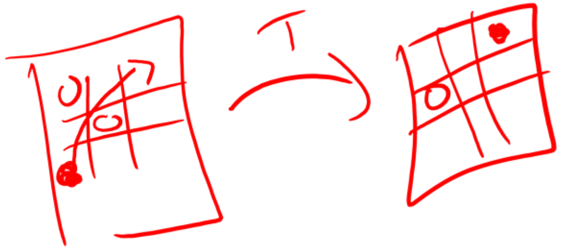


in general
- MDPs are non-deterministic search problems
  - One way to solve them is with expectimax search
  - We'll have a new tool soon

[Demo – gridworld manual intro (L8D1)]

Other examples of MDPs

- Checkers Boardgame



$S = \{\text{all board states}\}$

$A = \{\text{all legal actions}\}$

$T = P(s' \mid a, s)$

include other players move

option A

$R = \begin{cases} +1 & \text{capture} \\ -1 & \text{lose piece} \end{cases}$

$= \begin{cases} +1 & \text{win} \\ -1 & \text{loss} \\ 0 & \text{draw} \end{cases}$

- Medication treatment

Other examples of MDPs

- Self-driving car

- Language Generation (ChatGPT)

What is Markov about MDPs?

- "Markov" generally means that given the present state, the future and the past are independent. Conditional Independence!

- For Markov decision processes, "Markov" means action outcomes depend only on the current state

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

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

Andrey Markov
(1856-1922)

Types of Markov Models

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \rightarrow s_3$$

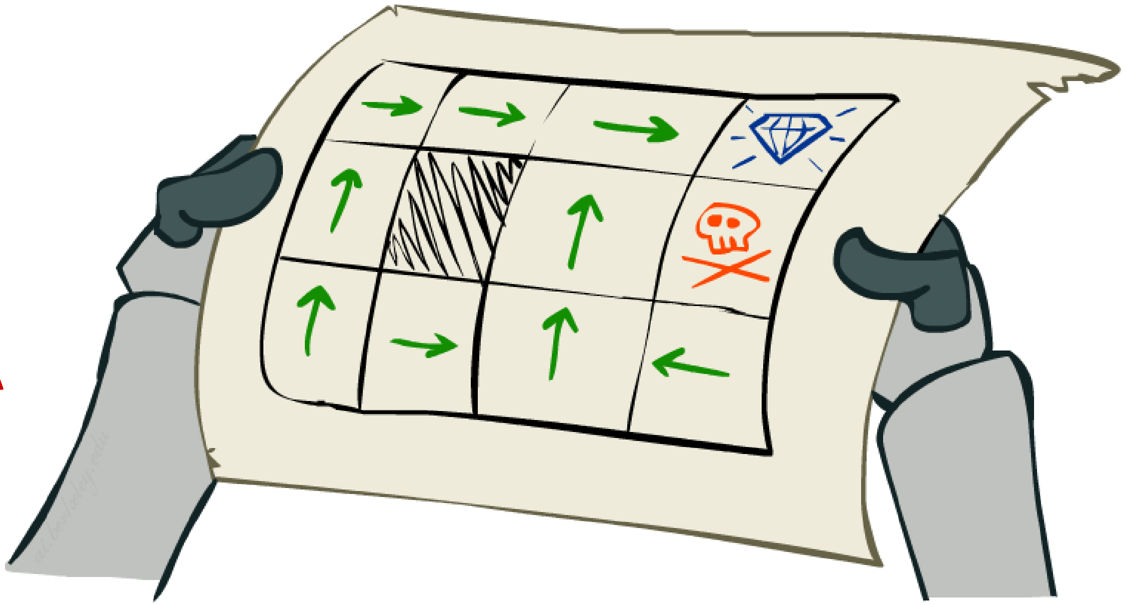
$$\begin{array}{ccccc} o_1 & & o_2 & & o_3 \\ \uparrow & & \uparrow & & \uparrow \\ s_1 & \rightarrow & s_2 & \rightarrow & s_3 \end{array}$$

	System state is fully observable	System state is partially observable
System is autonomous	Markov chain Markov model	Hidden Markov model (HMM)
System is controlled	Markov decision process (MDP)	Partially observable Markov decision process (POMDP)

Policies

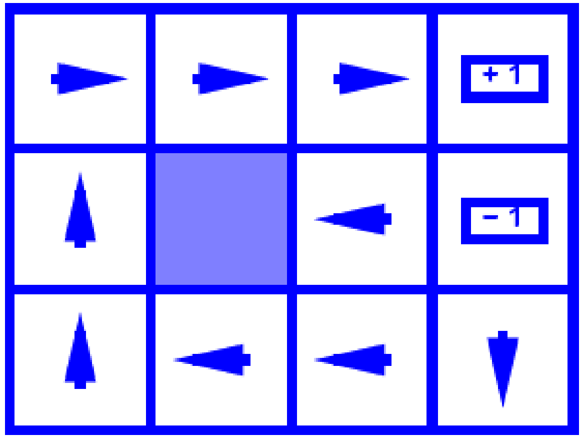
- In deterministic single-agent search problems, we wanted an optimal plan, or sequence of actions, from start to a goal

- For MDPs, we want an optimal policy $\pi^*: S \rightarrow A$
  - A policy π gives an action for each state
  - An optimal policy is one that maximizes expected utility if followed
  - An explicit policy defines a reflex agent

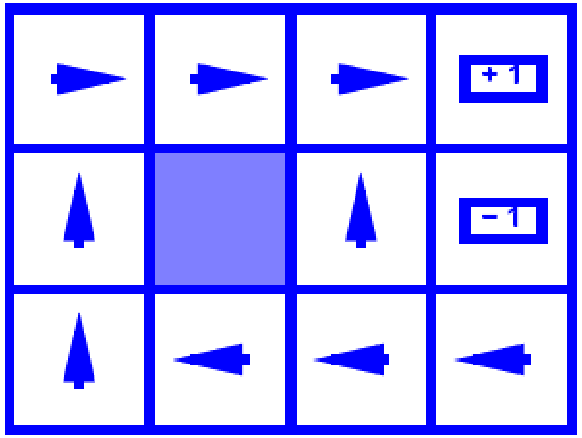


Optimal policy when R(s, a, s') = -0.03 for all non-terminals s

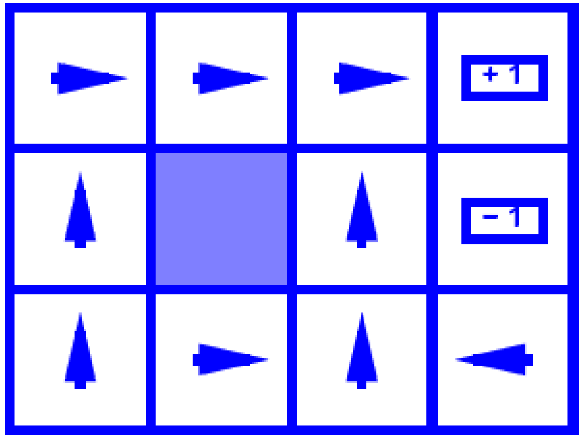
Optimal Policies



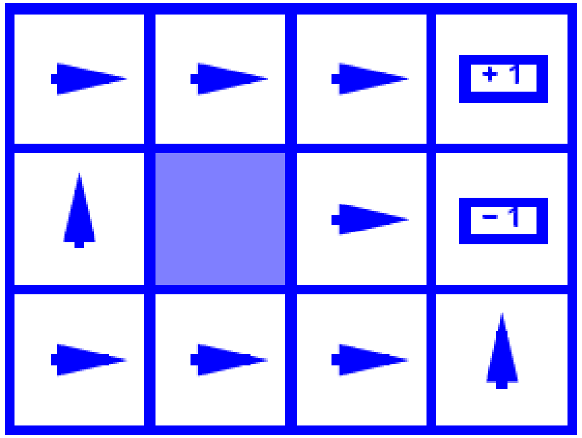
R(s) = -0.01



R(s) = -0.03



R(s) = -0.4



R(s) = -2.0

Discounting

$\gamma \in (0,1)$

- It's reasonable to maximize the sum of rewards
- It's also reasonable to prefer rewards now to rewards later
- One solution: values of rewards decay exponentially

1

Worth Now

γ

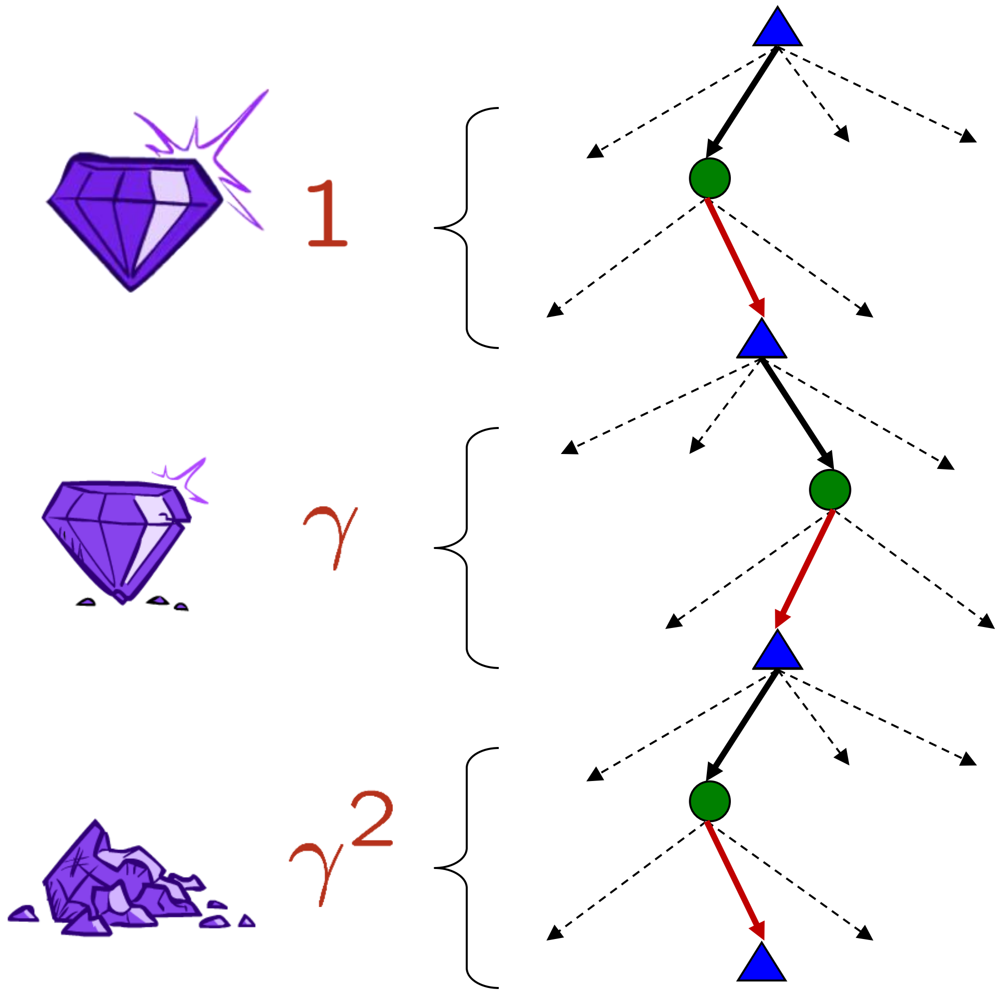
Worth Next Step

γ^2

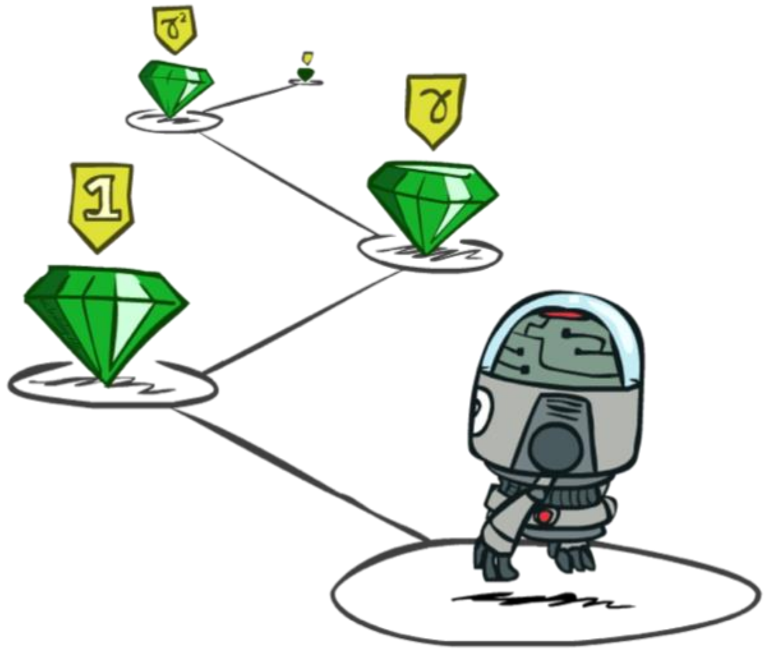
Worth In Two Steps

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



Stationary Preferences



- Theorem: if we assume stationary preferences:

$$[a_1, a_2, \ldots] \succ [b_1, b_2, \ldots]$$
$$\Updownarrow$$
$$[r, a_1, a_2, \ldots] \succ [r, b_1, b_2, \ldots]$$

- Then: there are only two ways to define utilities
  - Additive utility: $U([r_0, r_1, r_2, \ldots]) = r_0 + r_1 + r_2 + \cdots$
  - Discounted utility: $U([r_0, r_1, r_2, \ldots]) = r_0 + \gamma r_1 + \gamma^2 r_2 \cdots$

Quiz: Discounting

done

- Given: reward

a	b	c	d	e
10				1

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

$\rightarrow$ $1 \cdot \gamma$ $1/10$

$\leftarrow$ $0 + \gamma 0 + \gamma^2 0 + \gamma^3 10$ $0.001 \cdot 10$ $1/100$

- Quiz 1: For γ = 1, what is the optimal policy?

10	←	←	←	1

- Quiz 2: For γ = 0.1, what is the optimal policy?

γ γ γ

10	←	←	→	1

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

$\gamma = 10\gamma^3$ $1 = 10\gamma^2$ $\gamma = \sqrt{\frac{1}{10}} \approx .316$

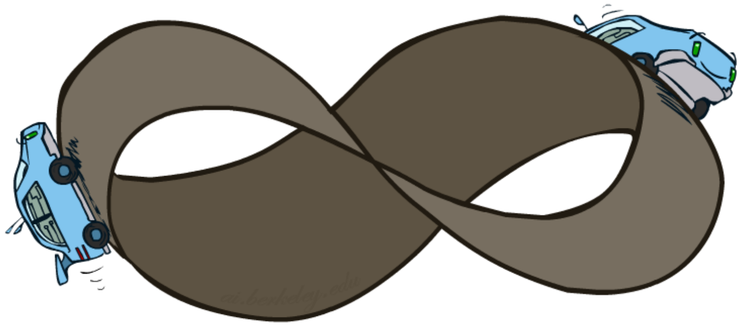
Infinite Utilities?!

- Problem: What if the game lasts forever? Do we get infinite rewards?
- Solutions:
  - Finite horizon: (similar to depth-limited search)
    - Terminate episodes after a fixed T steps (e.g. life)
    - Gives nonstationary policies (π depends on time left)
  - Discounting: use $0 < \gamma < 1$

Why?

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

    - Smaller γ means smaller "horizon" – shorter term focus
  - Absorbing state: guarantee that for every policy, a terminal state will eventually be reached (like "overheated" for racing)



$$A = \sum_{t=0}^{\infty} R_{max} \gamma^t = R_{max} + \cancel{\gamma R_{max}} + \cancel{\gamma^2 R_{max}} + \cdots$$

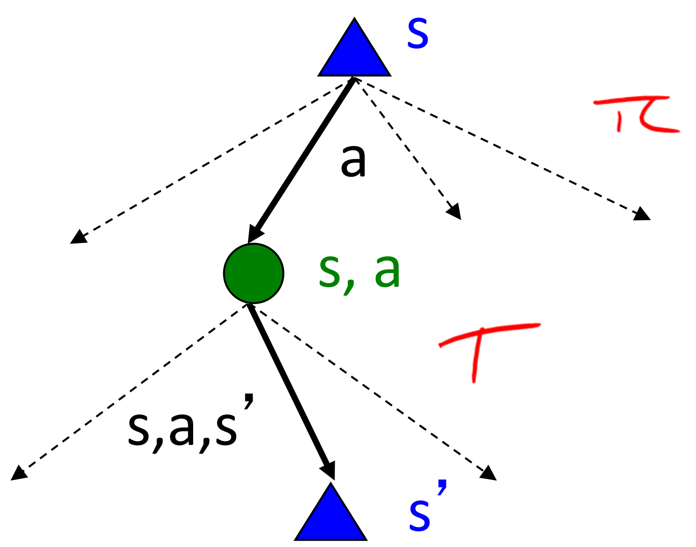
$$B = \gamma \sum_{t=0}^{\infty} R_{max} \gamma^t = \cancel{\gamma R_{max}} + \cancel{\gamma^2 R_{max}} + \cancel{\gamma^3 R_{max}} \cdots$$

$$A - B = R_{max}$$

$$\underbrace{\left(\sum_{t=0}^{\infty} R_{max} \gamma^t \right)}_{= \frac{R_{max}}{1-\gamma}} (1-\gamma) = R_{max}$$

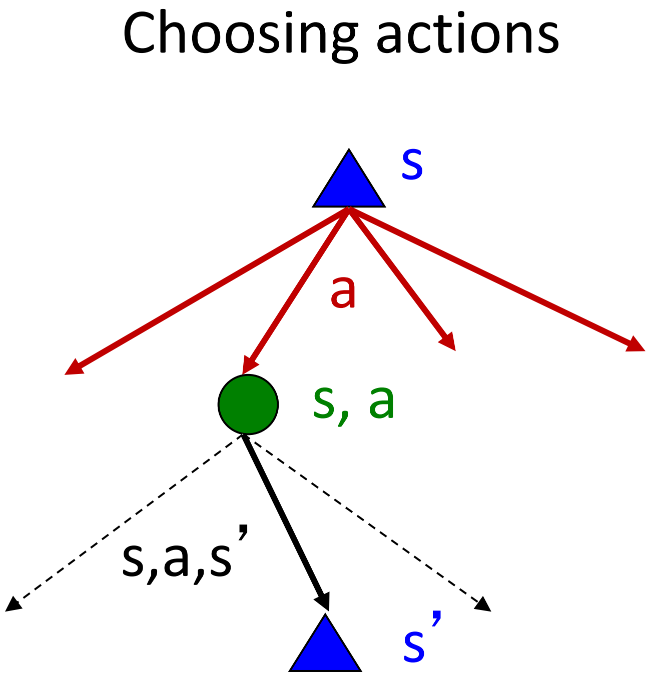
MDP Notation

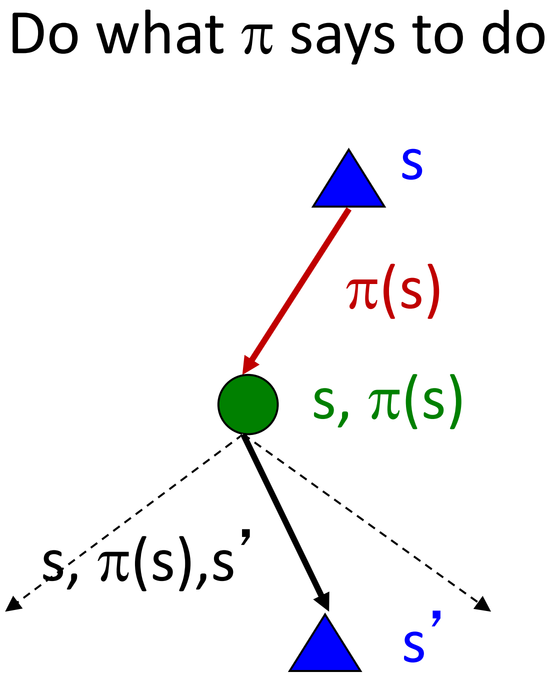
- Markov decision processes:
  - Set of states S
  - Start state s_0
  - Set of actions A
  - Transitions P(s'|s,a) (or T(s,a,s'))
  - Rewards R(s,a,s') (and discount γ)



- Important MDP quantities:
  - Policy = Choice of action for each state
  - Utility = expected sum of (discounted) rewards = "expected return"

Fixed Policies





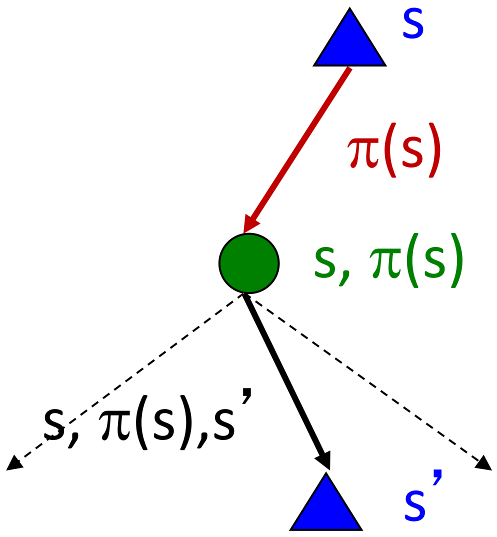
- If we fixed some policy π(s), then the computation is simpler – only one action per state
  - … though the performance now depend on which policy we fixed

Performance of a Fixed Policy

- Goal: 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^π(s) = expected total discounted rewards starting in s and following π



- Recursive relation (one-step look-ahead):

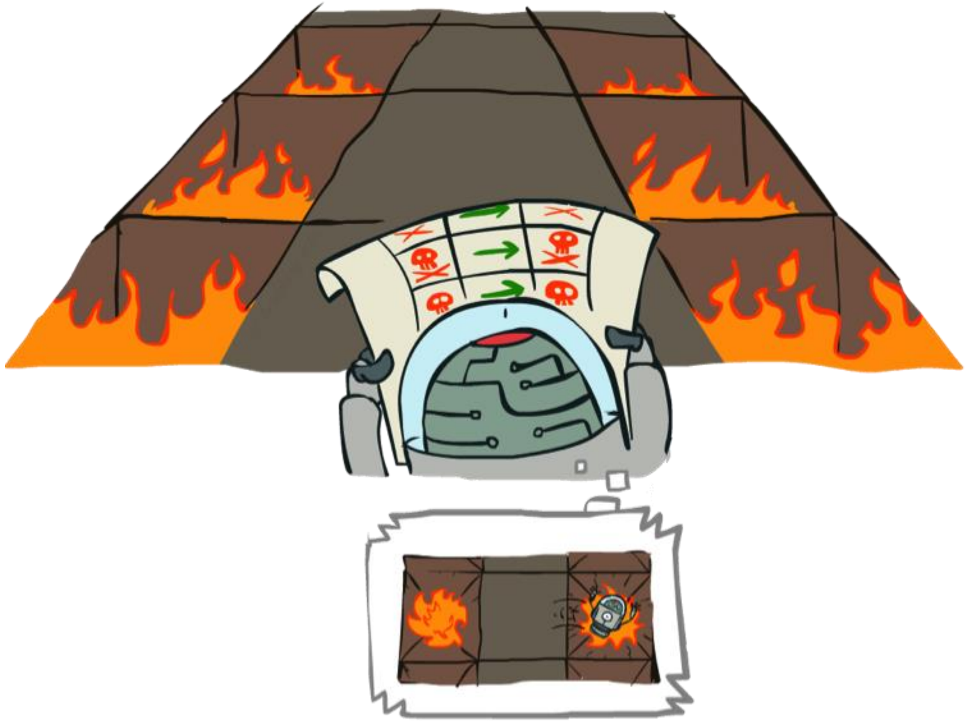
$$V^\pi(s) = \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^\pi(s')]$$

$P(s' | s, \pi(s))$

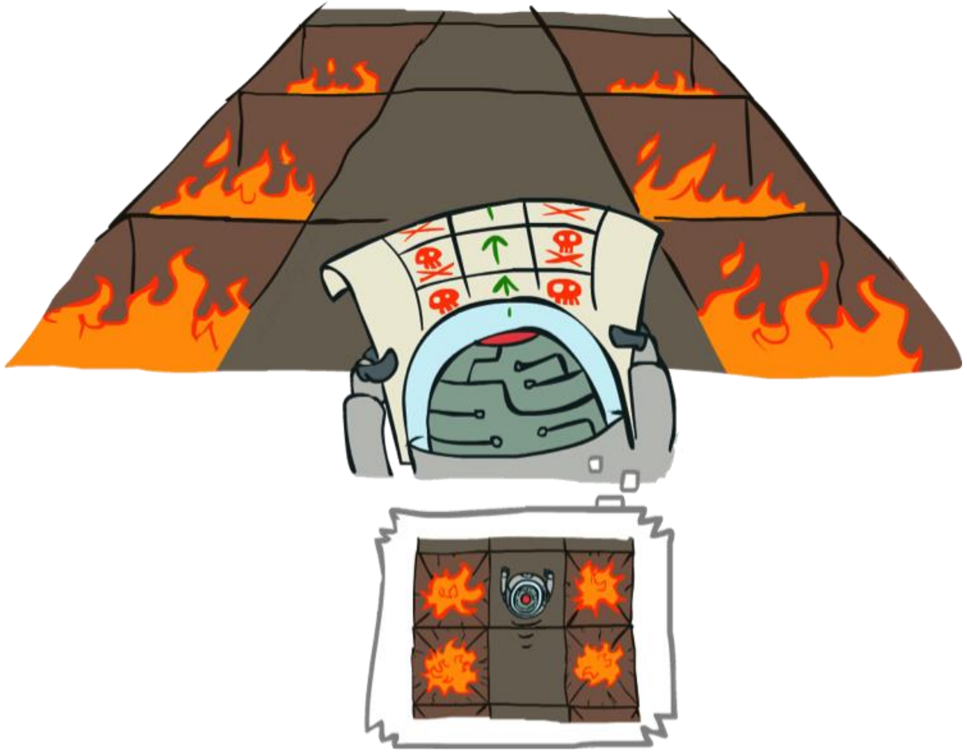
$$= \mathbb{E}_{s'}\left[R(s, \pi(s), s') + \gamma V^\pi(s') \right]$$

Example: Policy Evaluation

Always Go Right



Always Go Forward



Example: Policy Evaluation

$\gamma = 0.9$

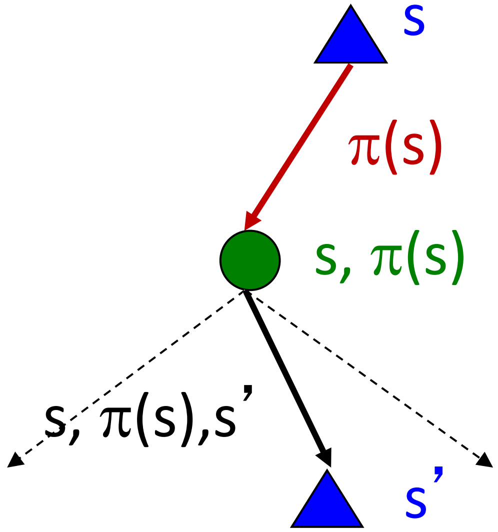
Always Go Right

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



0

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

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

$V_1^\pi(s)$

- Efficiency: O(S²) per iteration

- Idea 2: Just a linear system
  - Solve with Numpy or Matlab (or your favorite linear system solver)

Policy Evaluation

G $\xrightarrow{+10}$ 0 Terminal

- Idea 2: The Policy Evaluatoin Bellman equations are just a linear system
  - Solve with Numpy or Matlab (or your favorite linear system solver)

$$V = \begin{bmatrix} V(s_1) \\ \vdots \\ V(s_n) \end{bmatrix}$$

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

distribute

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

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

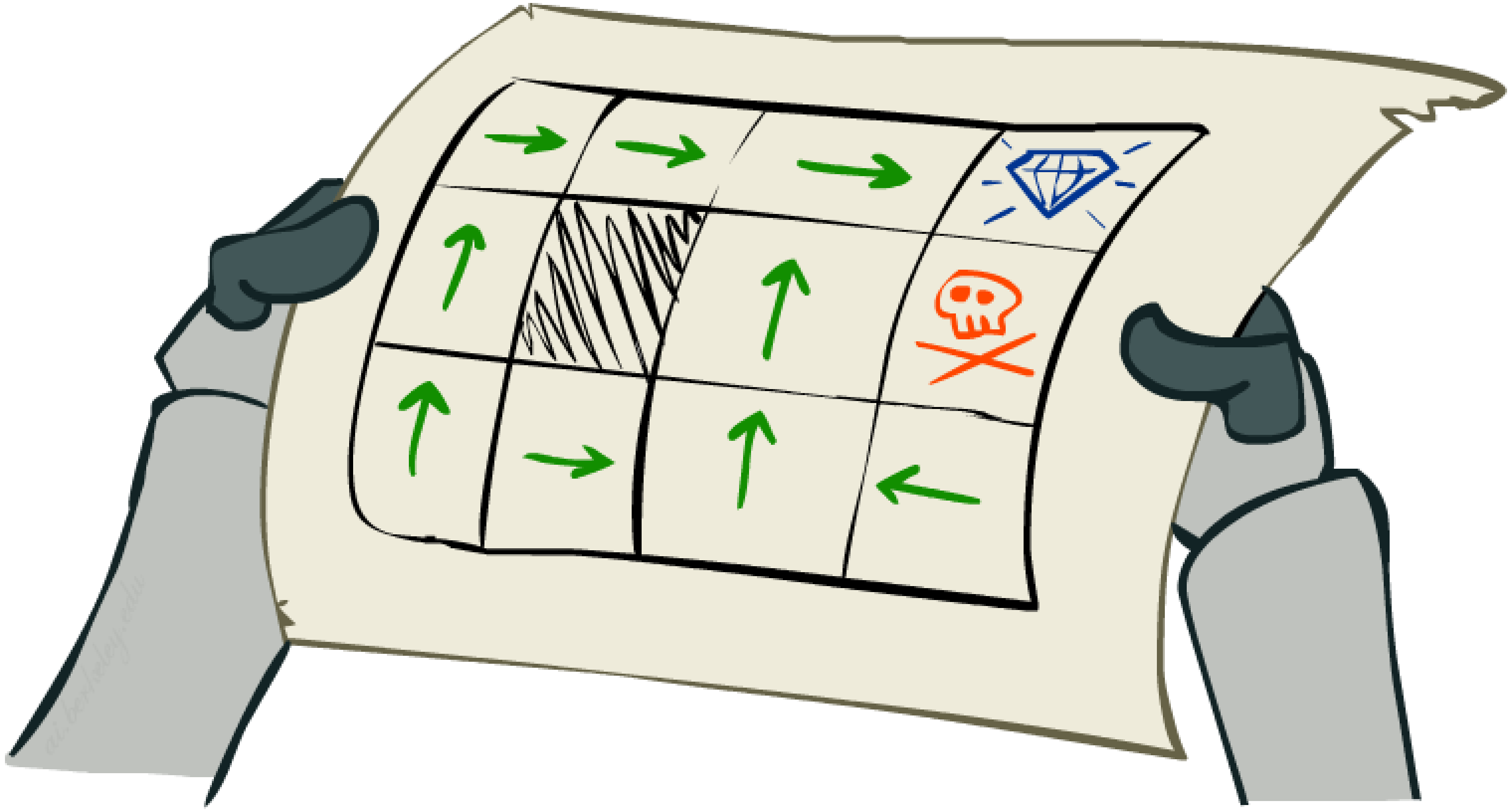
$$T^\pi(i, j) = P(j \mid i, \pi)$$

$$\underbrace{V^\pi}_{|S|\times 1} = \underbrace{\bar{R}}_{|S|\times 1} + \gamma \underbrace{T^\pi}_{|S|\times|S|} V^\pi \Rightarrow V^\pi - \gamma T^\pi V^\pi = R \Rightarrow (I - \gamma T^\pi) V^\pi = R$$

$$(I - \gamma T^\pi) V^\pi = \bar{R} \quad \Rightarrow V^\pi = (I - \gamma T^\pi)^{-1} \bar{R}$$

$Ax = b$

Solving MDPs



Optimal Quantities

- The value (utility) of a state s:

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

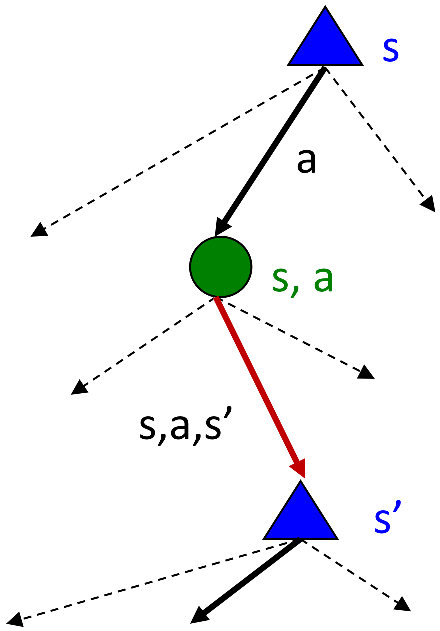
- The value (utility) of a q-state (s,a):

  $Q^*(s,a)$ = 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

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



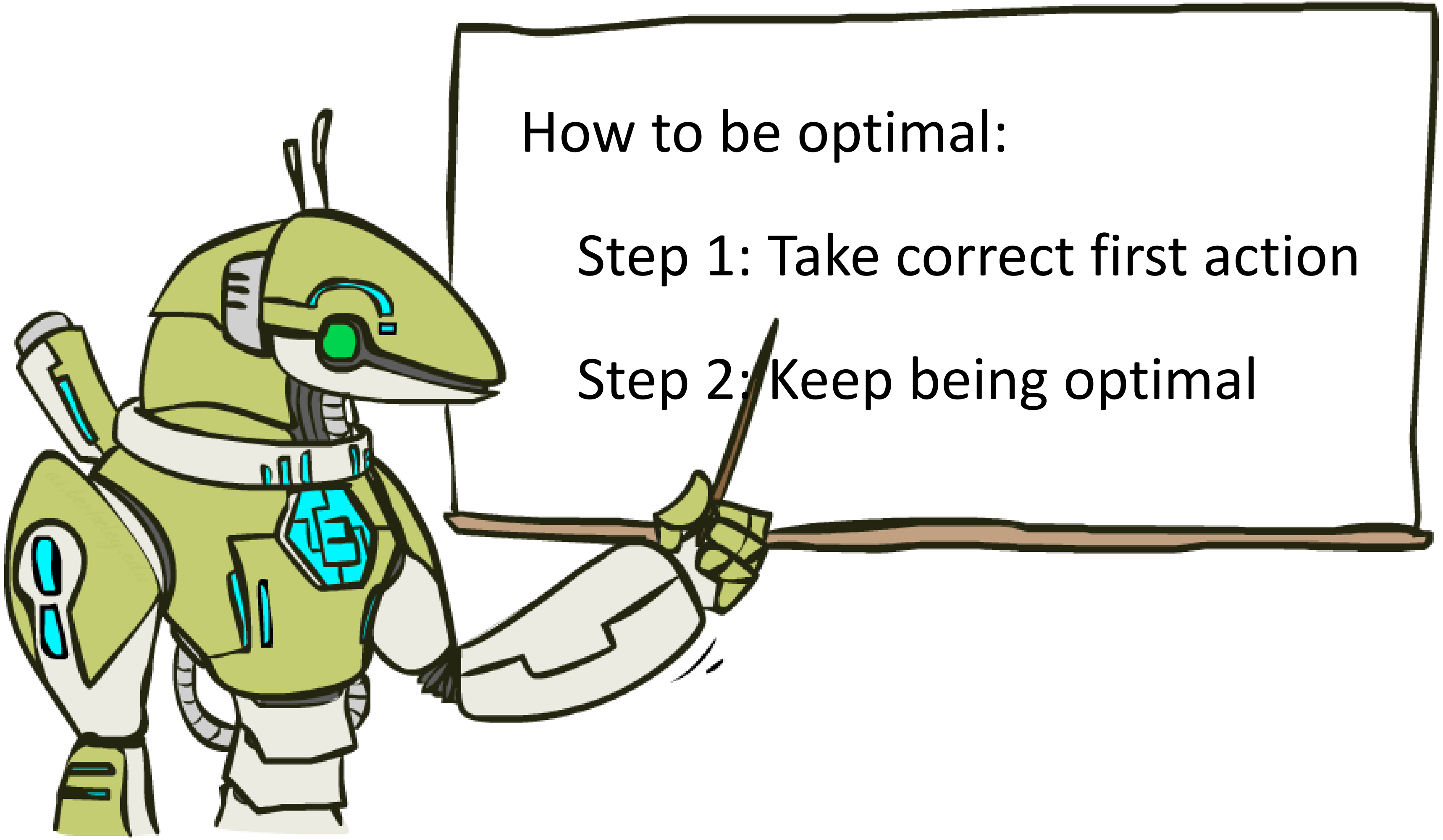
s is a *state*

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

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

Can we write the optimal policy in terms of Q*?

The Bellman Equations

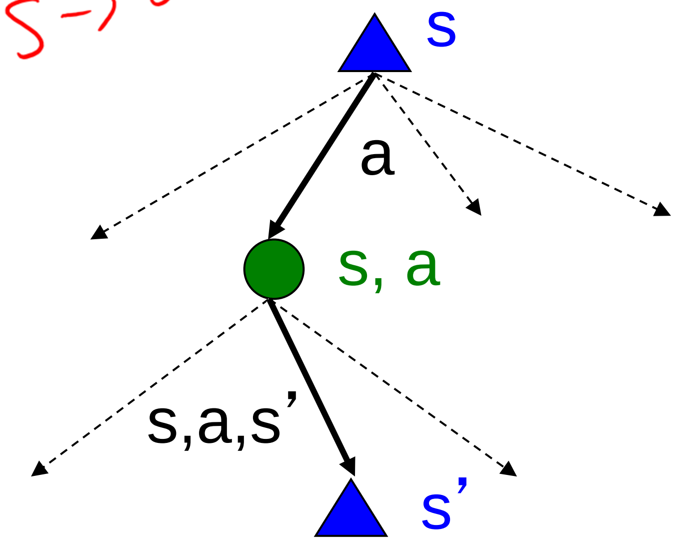


Bellman Equations

- Fundamental operation: compute the (expectimax) value of a state
  - Expected utility under optimal action
  - Average sum of (discounted) rewards
  - This is just what expectimax computed!

stoch $\pi : S \to d(A)$

det $\pi : S \to A$



- Recursive definition of value:

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

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

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

Aside: Different ways to write Bellman Eqns

- What if R only depends on state and action? e.g. R(s,a,s') = R(s,a)

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

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

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

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

Aside: Different ways to write Bellman Eqns

- What if R only depends on state? e.g. R(s,a,s') = R(s) *HW3

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

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

Value Iteration

γ^T $T \to \infty$

- 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:

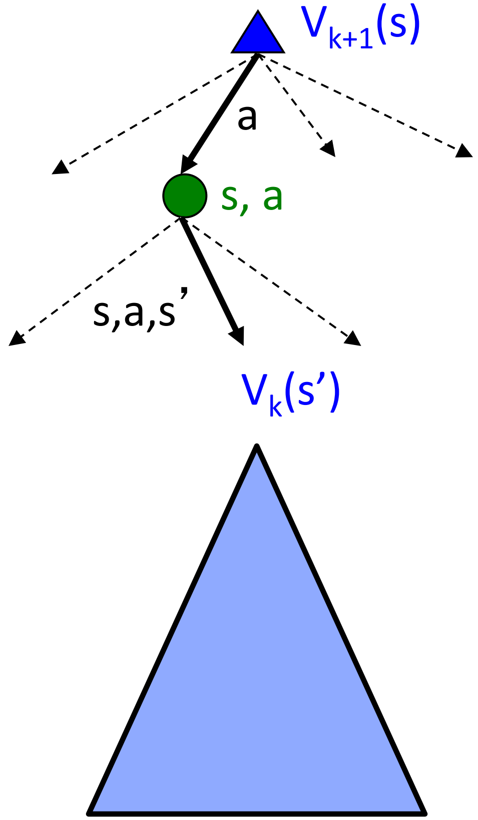
$$V_{k+1}(s) \leftarrow \max_a \overbrace{\sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s')\right]}^{Q_k(s,a)}$$

Bellman Update Equation

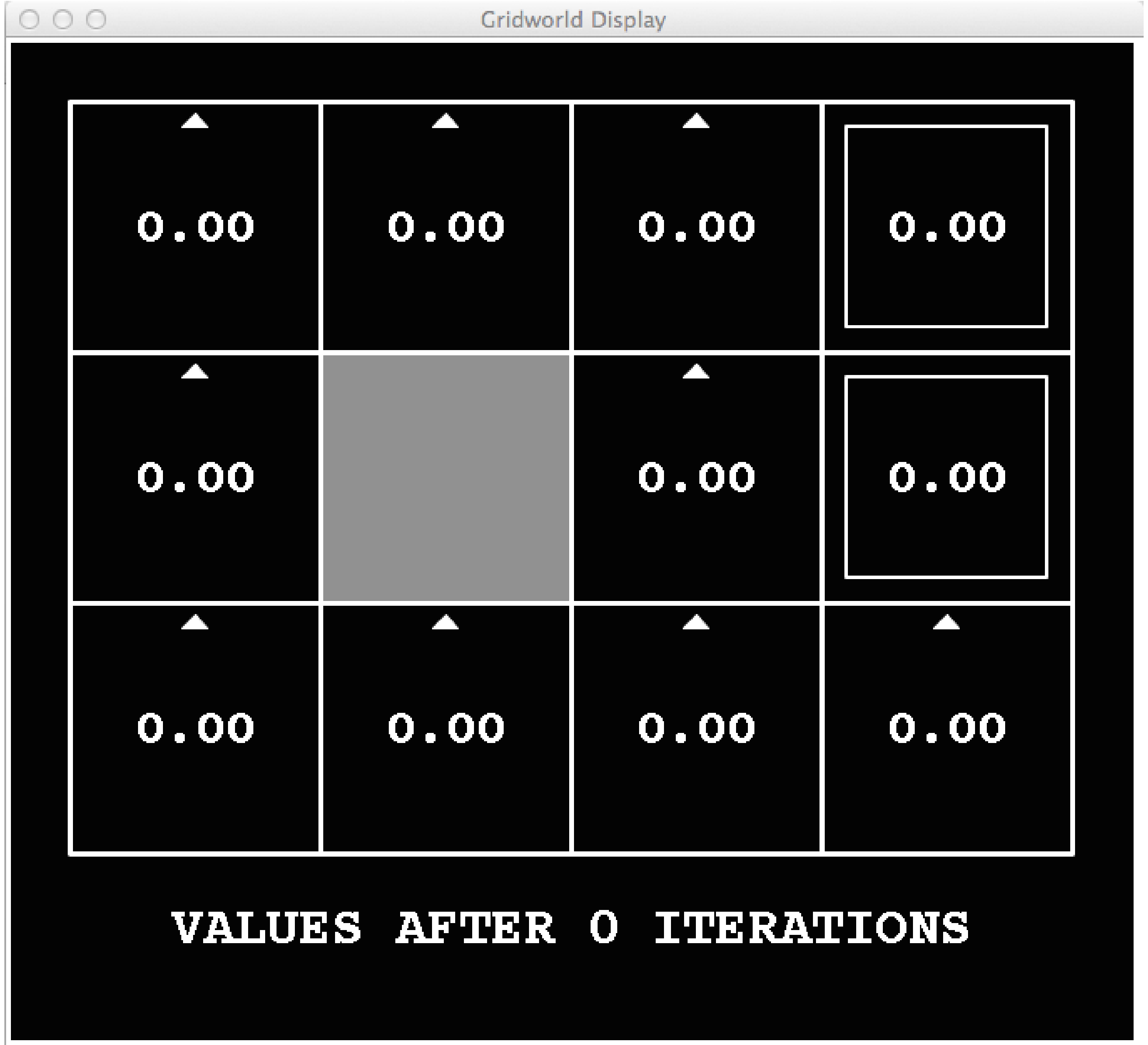
- Repeat until convergence

$$\max_s \left| V_{k+1}(s) - V_k(s) \right| < \varepsilon$$

- 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

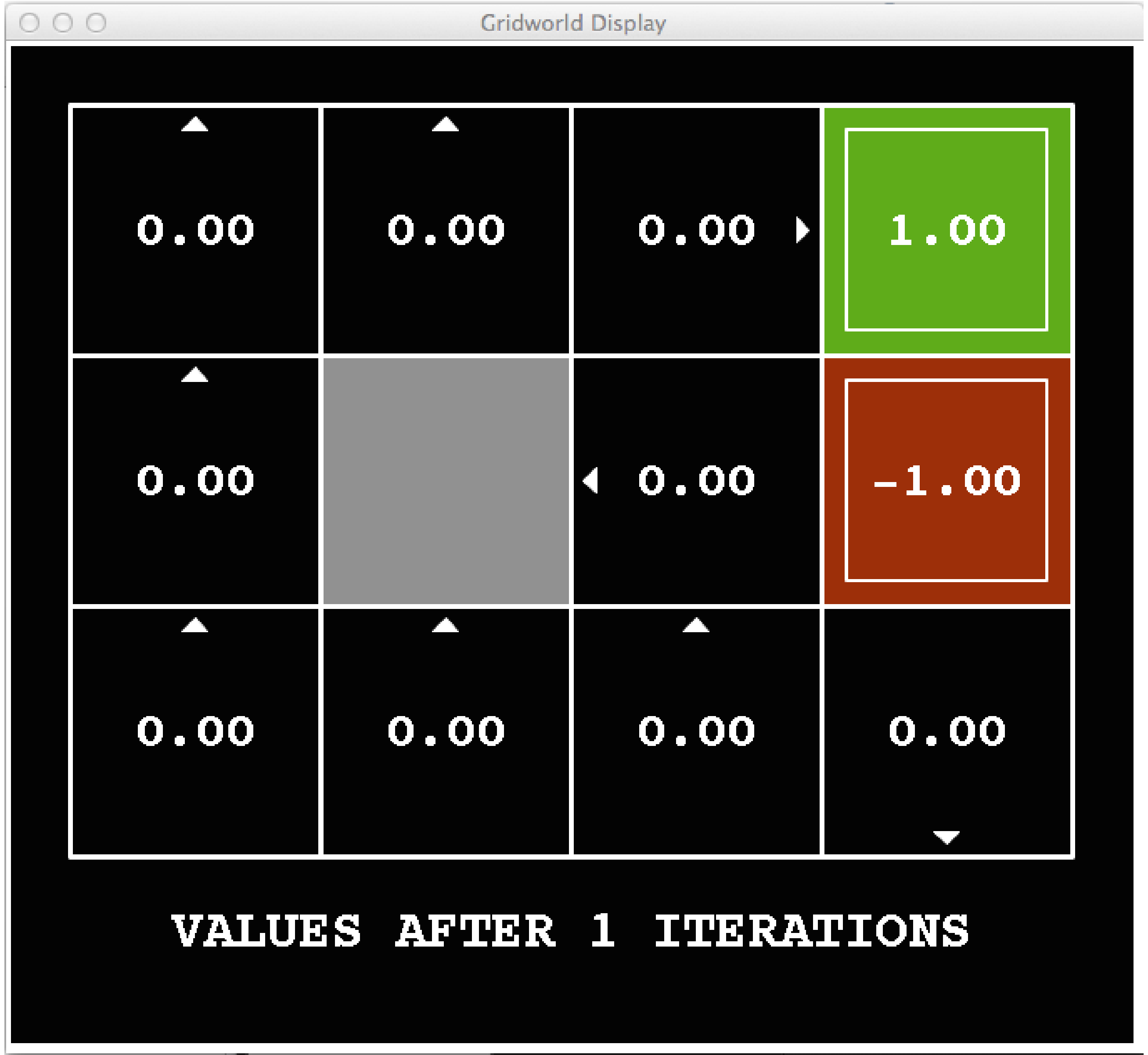


k=0



Noise = 0.2
Discount = 0.9
Living reward = 0

k=1



Noise = 0.2
Discount = 0.9
Living reward = 0

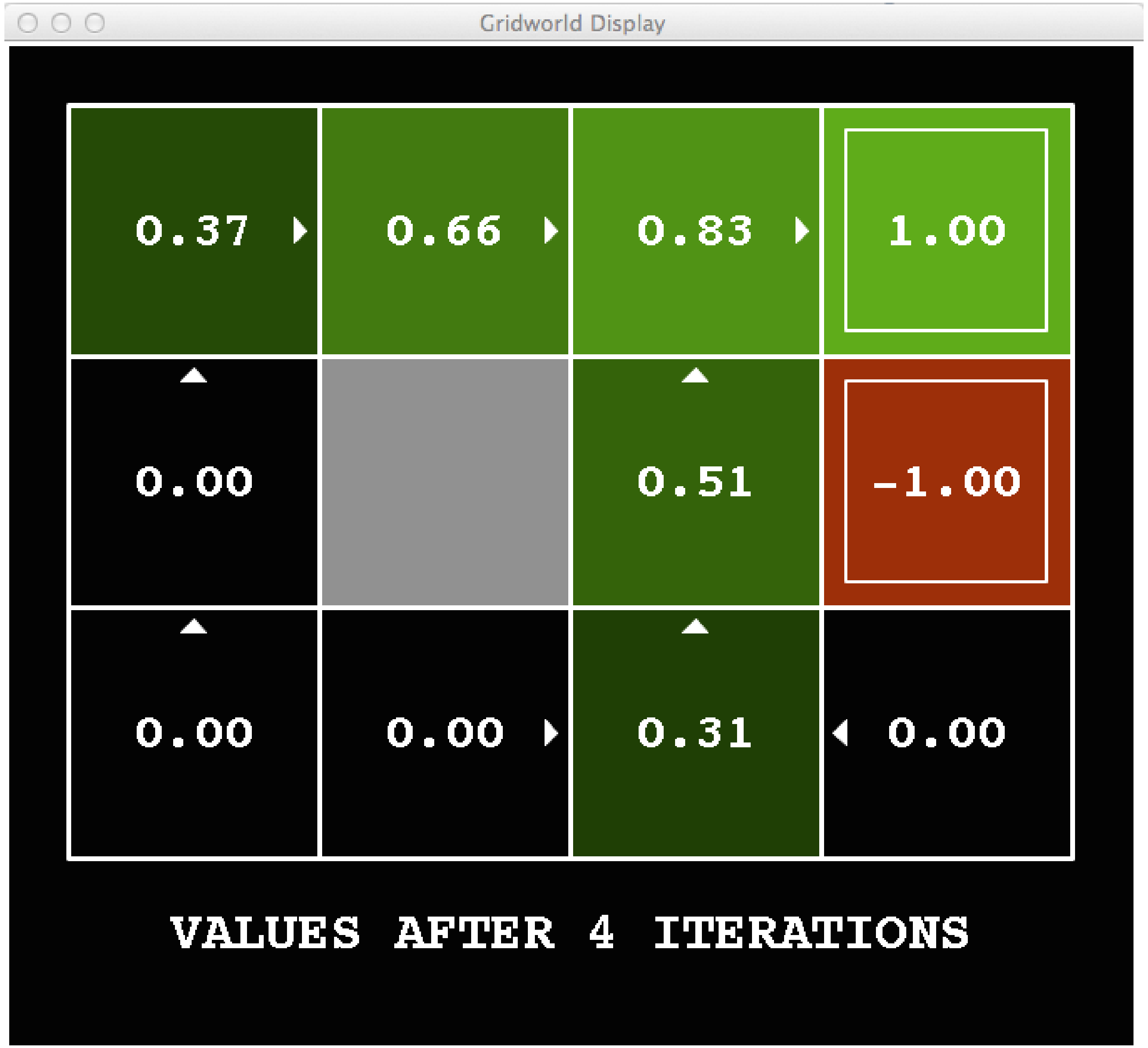
k=2

Noise = 0.2
Discount = 0.9
Living reward = 0

k=3

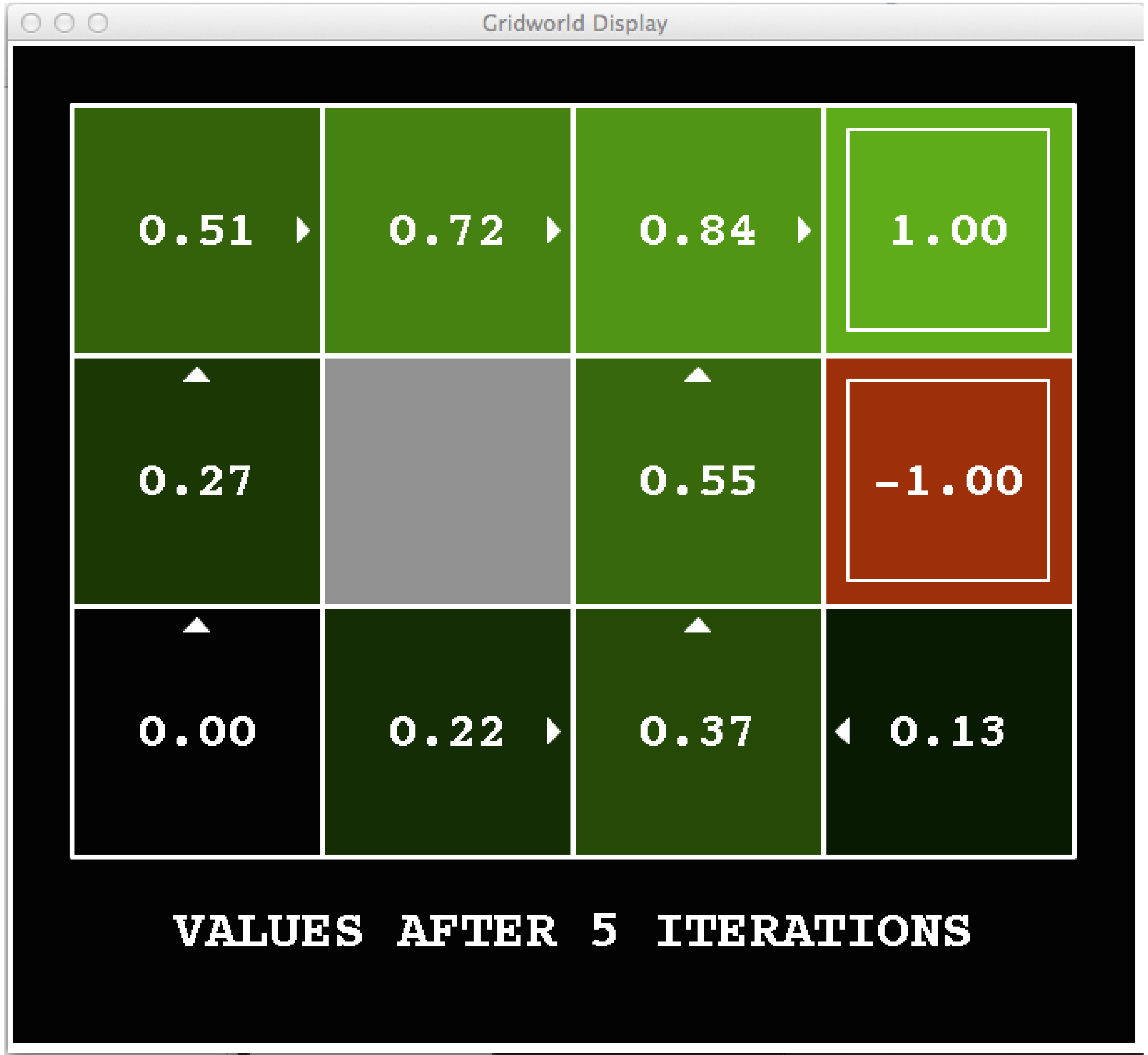
Noise = 0.2
Discount = 0.9
Living reward = 0

k=4



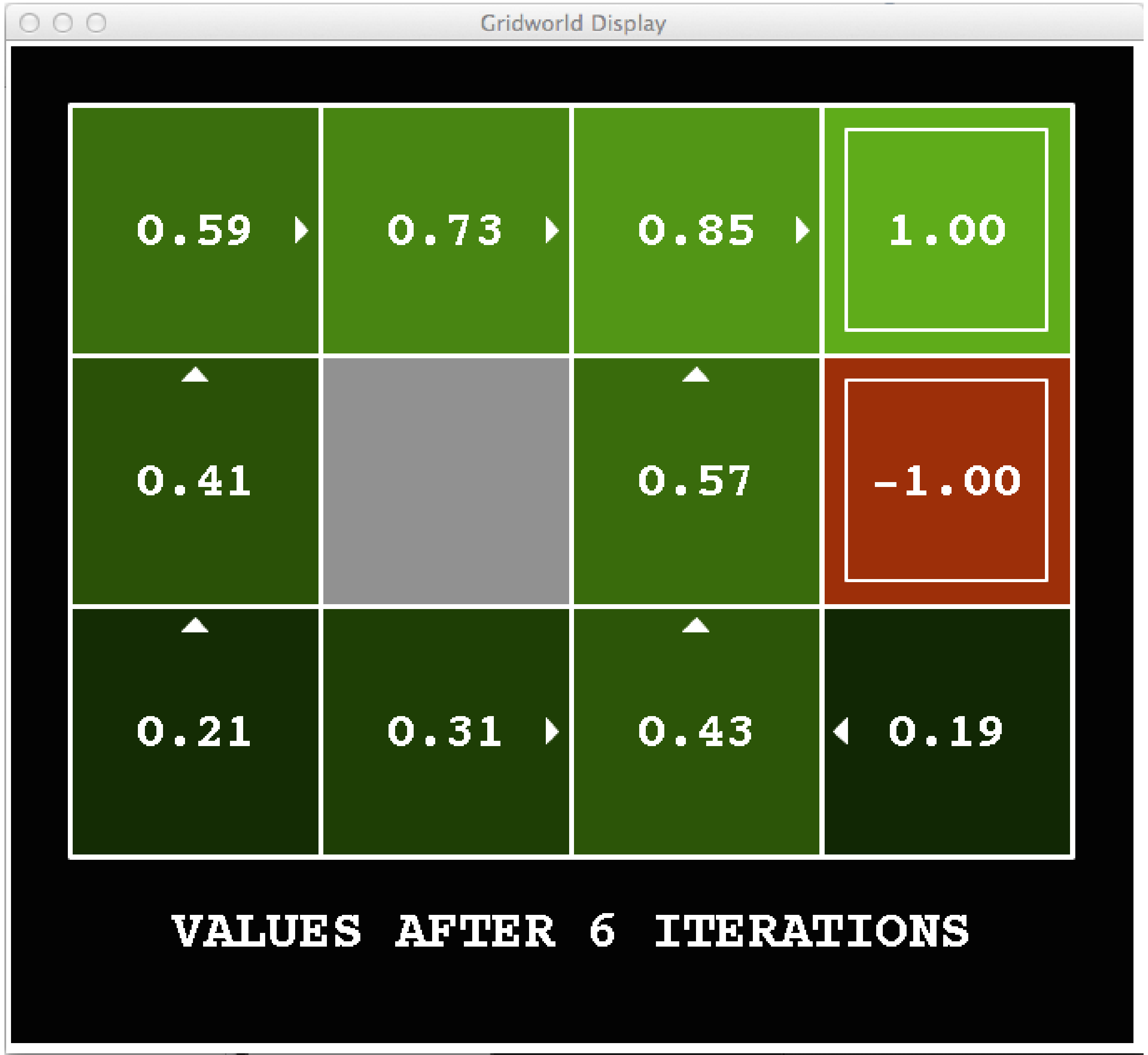
Noise = 0.2
Discount = 0.9
Living reward = 0

k=5



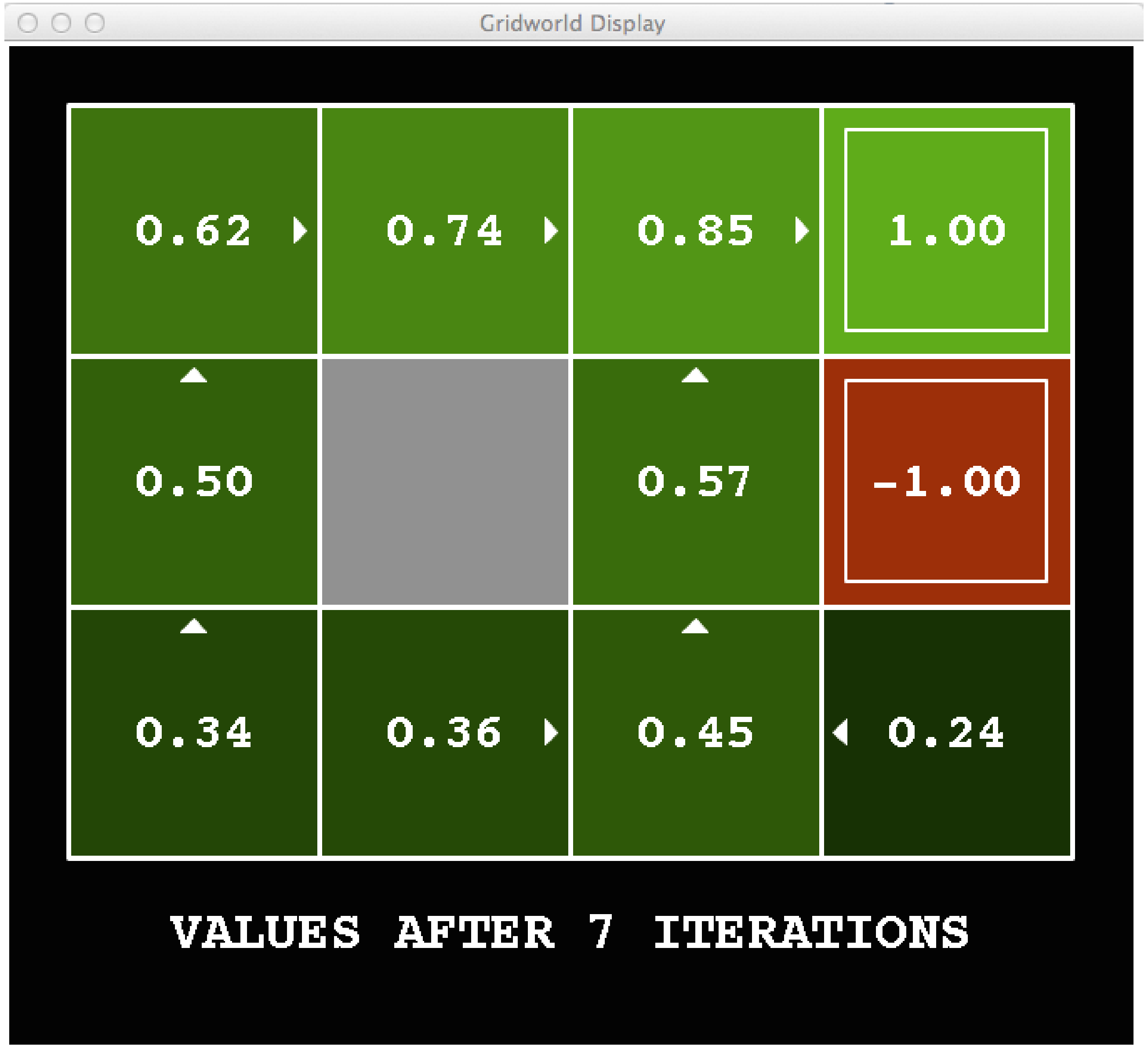
Noise = 0.2
Discount = 0.9
Living reward = 0

k=6



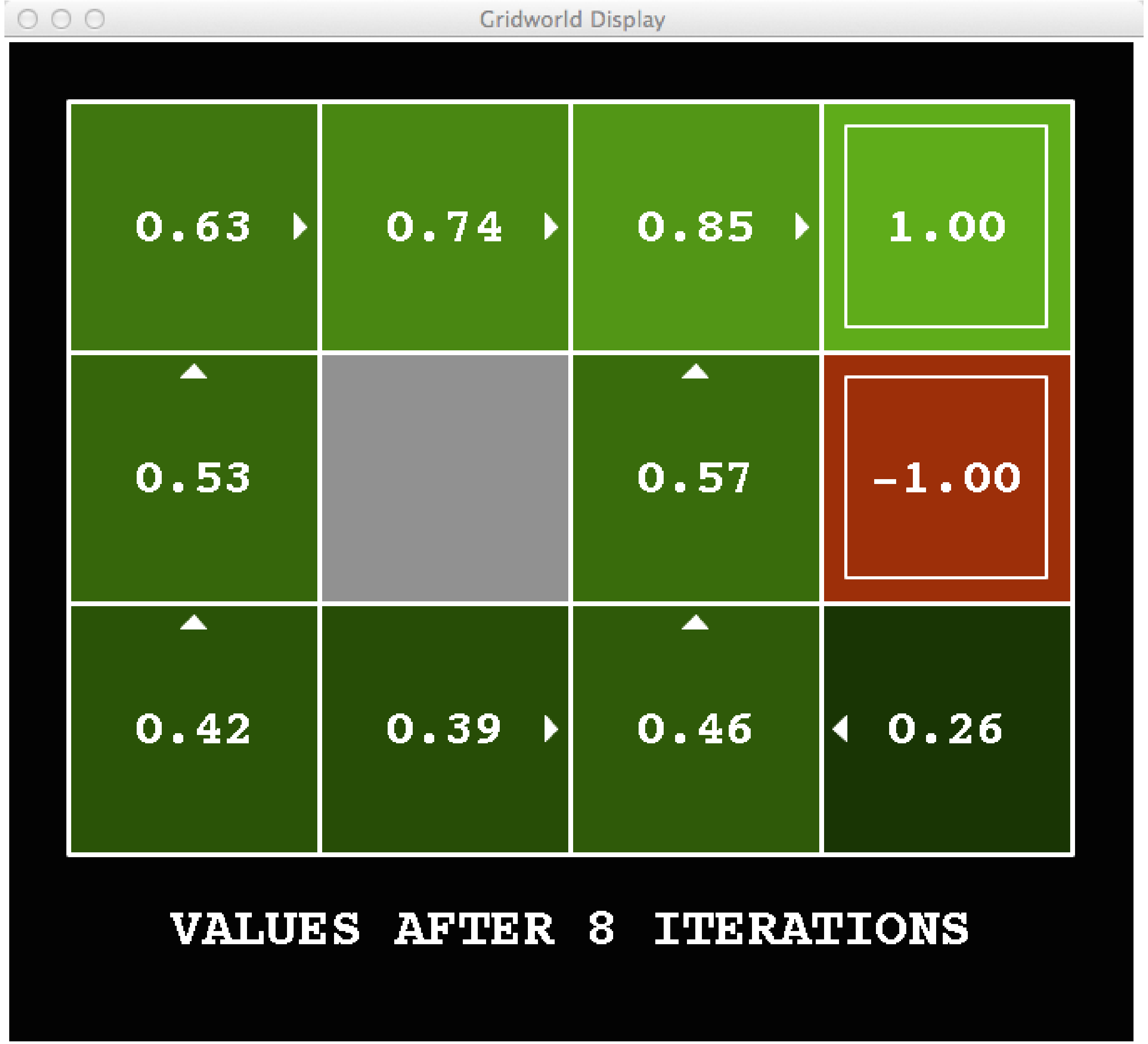
Noise = 0.2
Discount = 0.9
Living reward = 0

k=7



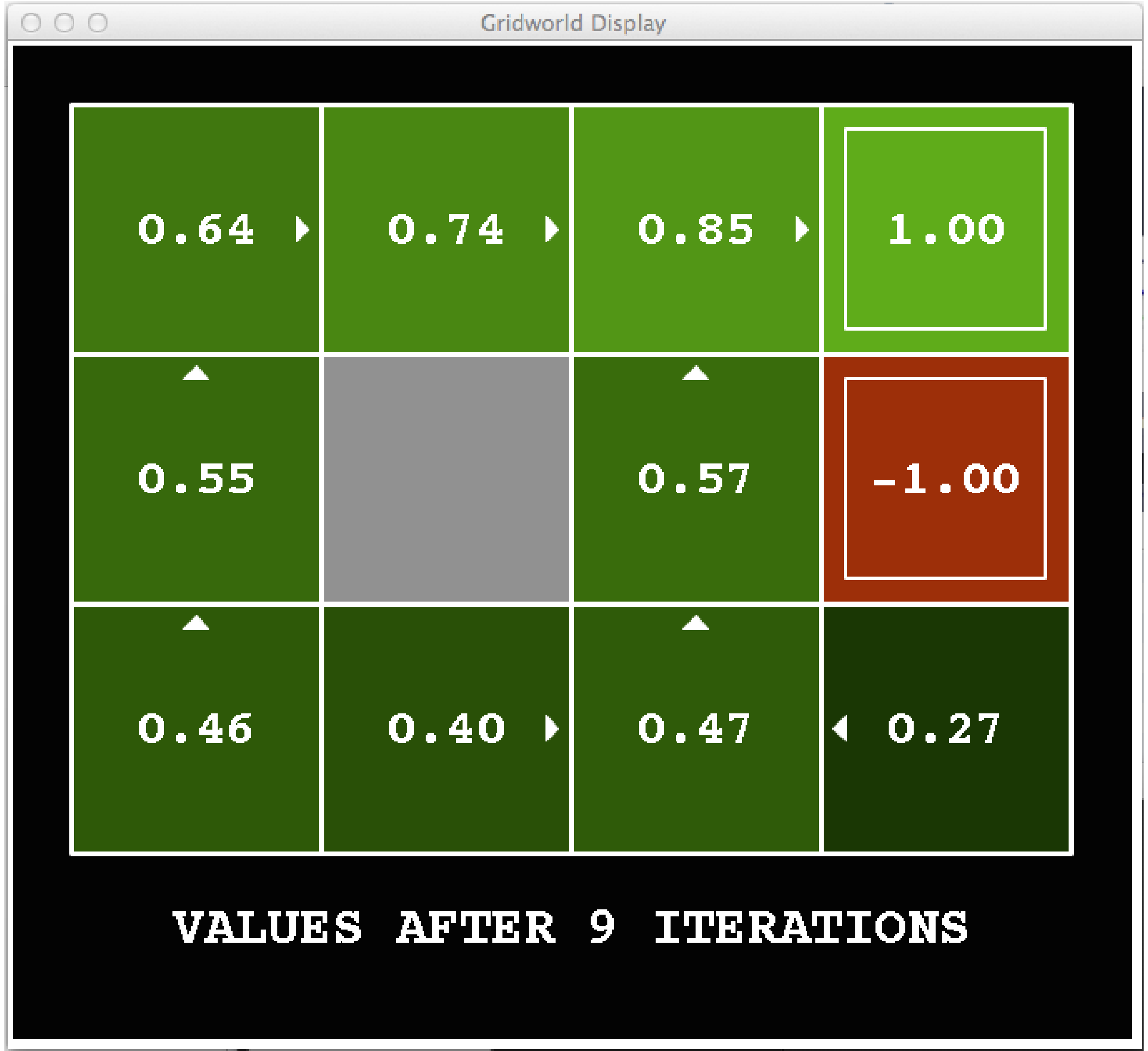
Noise = 0.2
Discount = 0.9
Living reward = 0

k=8



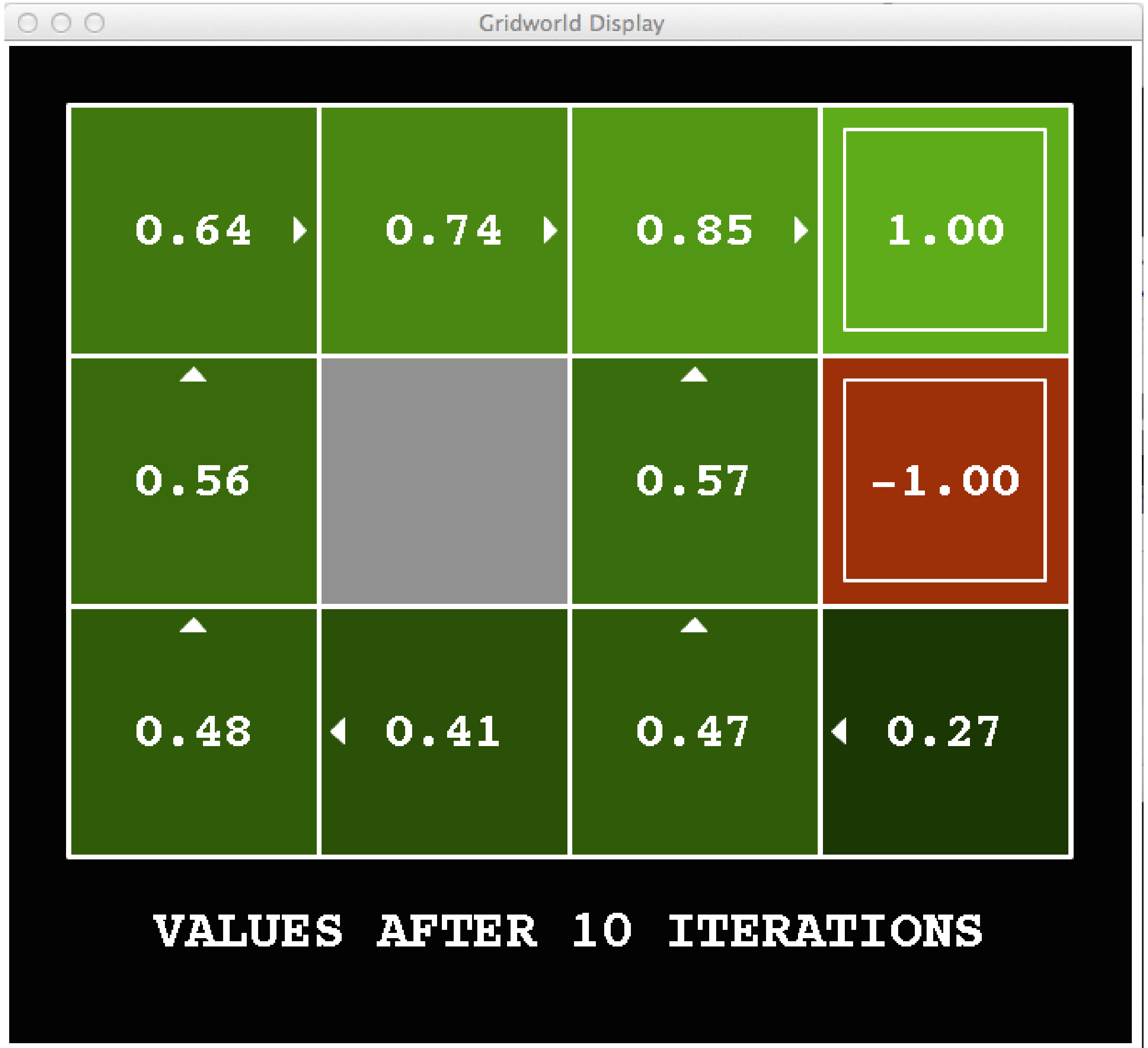
Noise = 0.2
Discount = 0.9
Living reward = 0

k=9



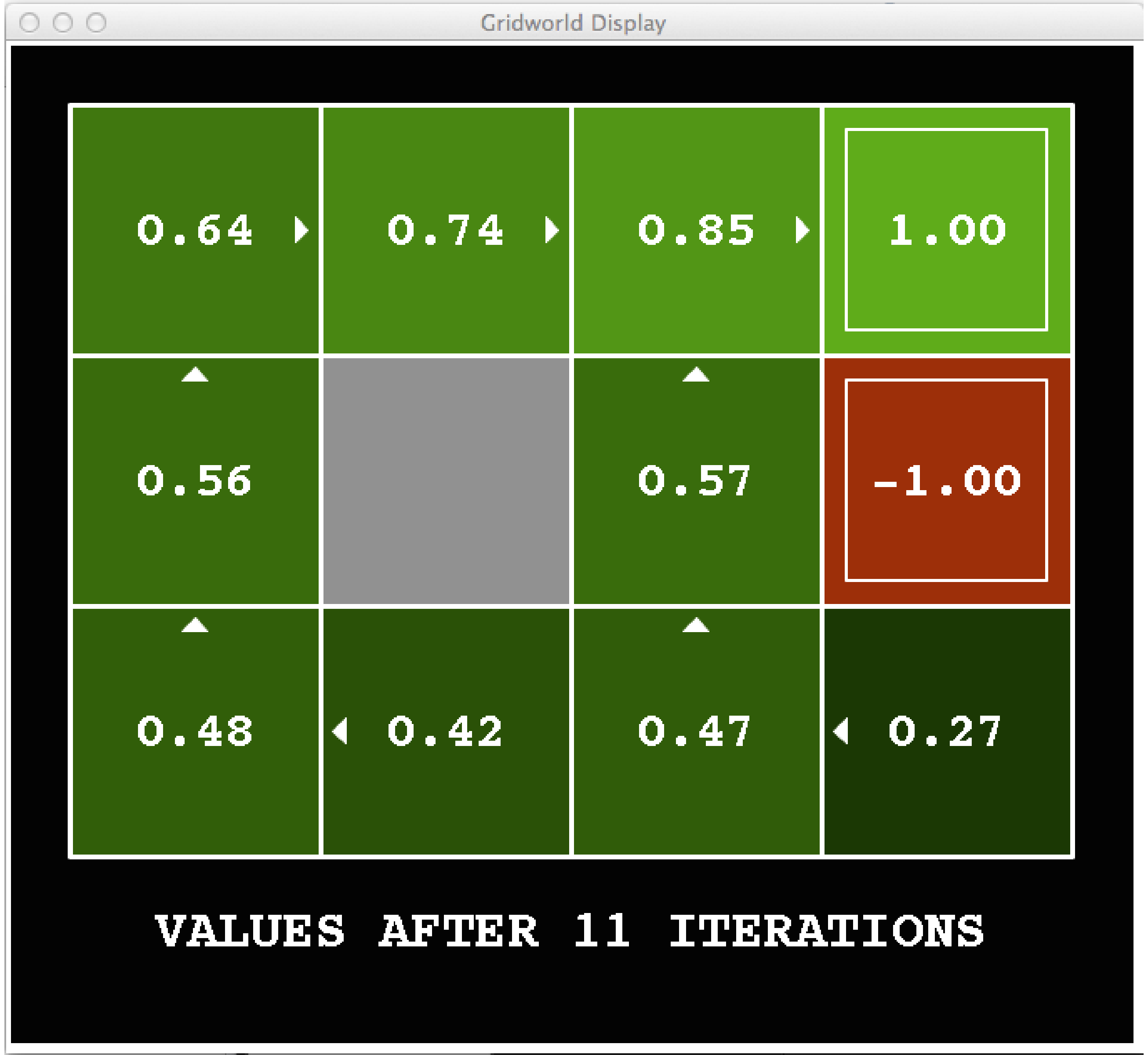
Noise = 0.2
Discount = 0.9
Living reward = 0

k=10



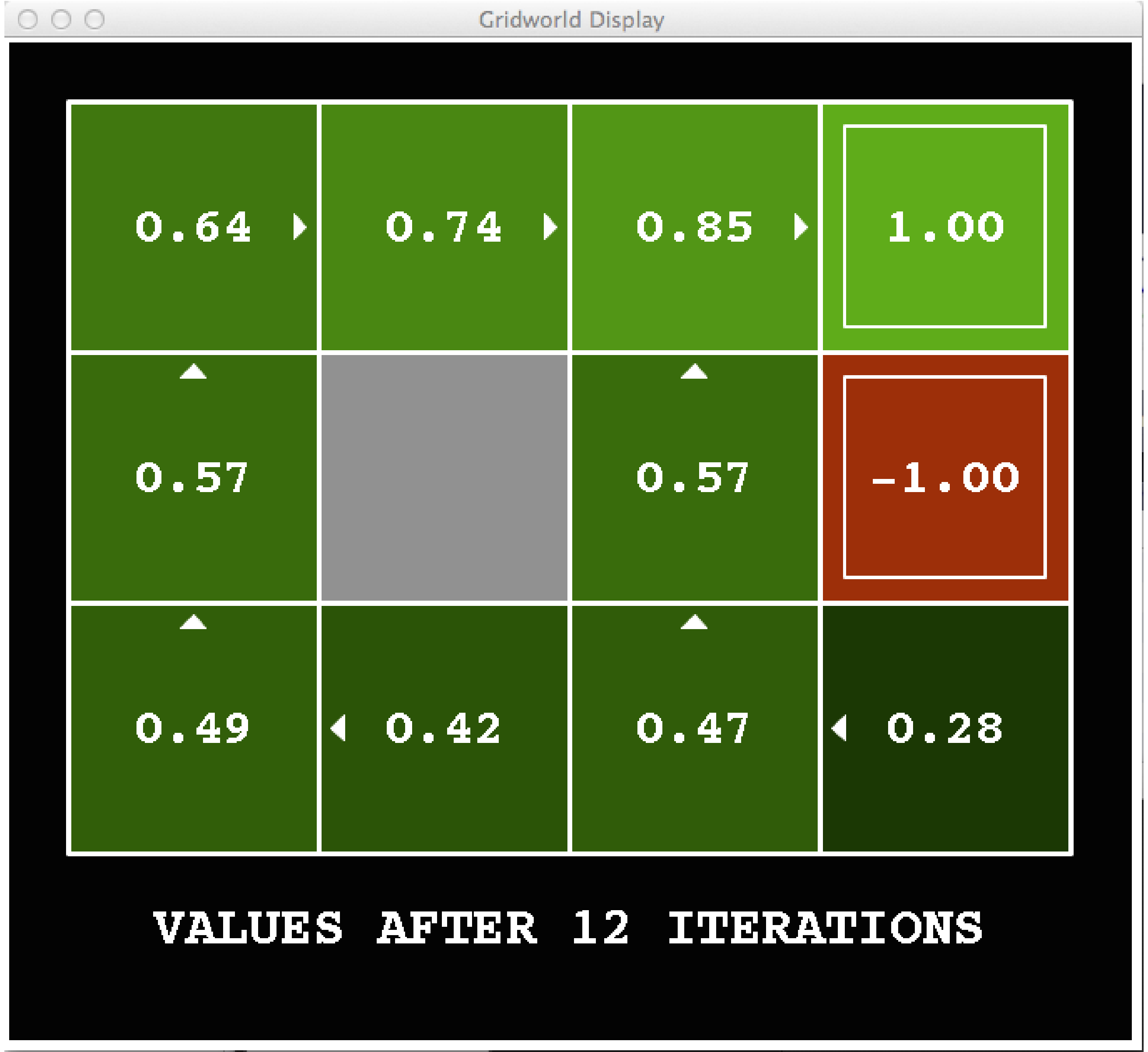
Noise = 0.2
Discount = 0.9
Living reward = 0

k=11



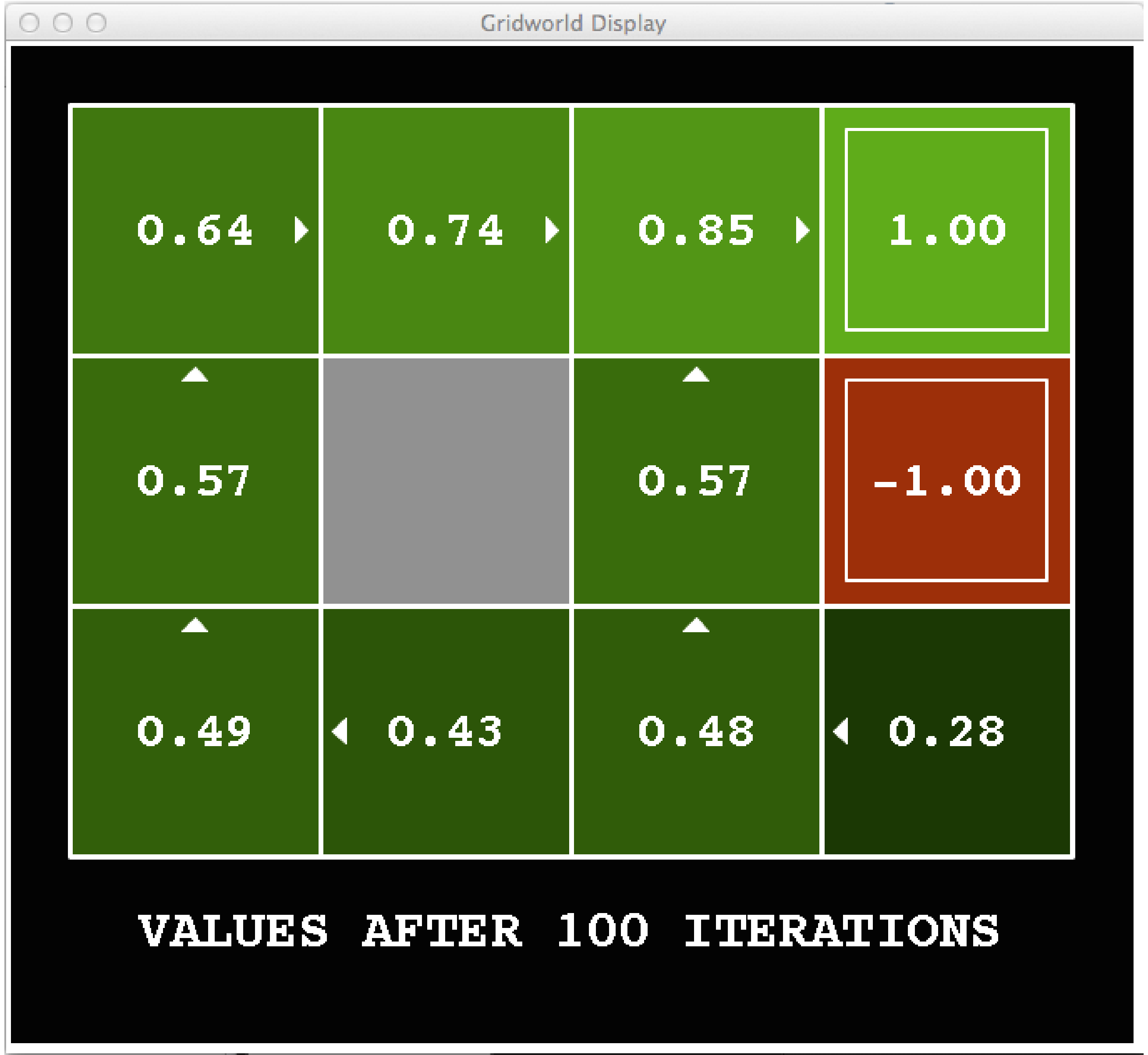
Noise = 0.2
Discount = 0.9
Living reward = 0

k=12



Noise = 0.2
Discount = 0.9
Living reward = 0

k=100



Noise = 0.2
Discount = 0.9
Living reward = 0

Value Iteration

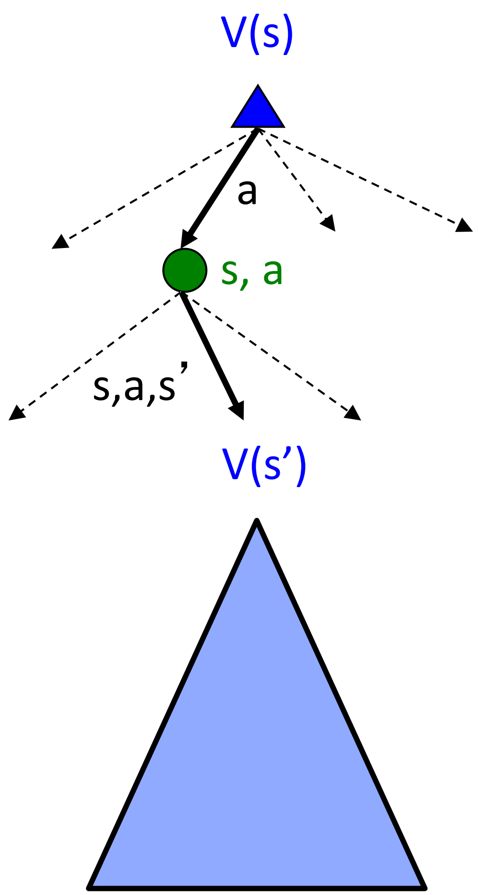
- Bellman equations characterize the optimal values:

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

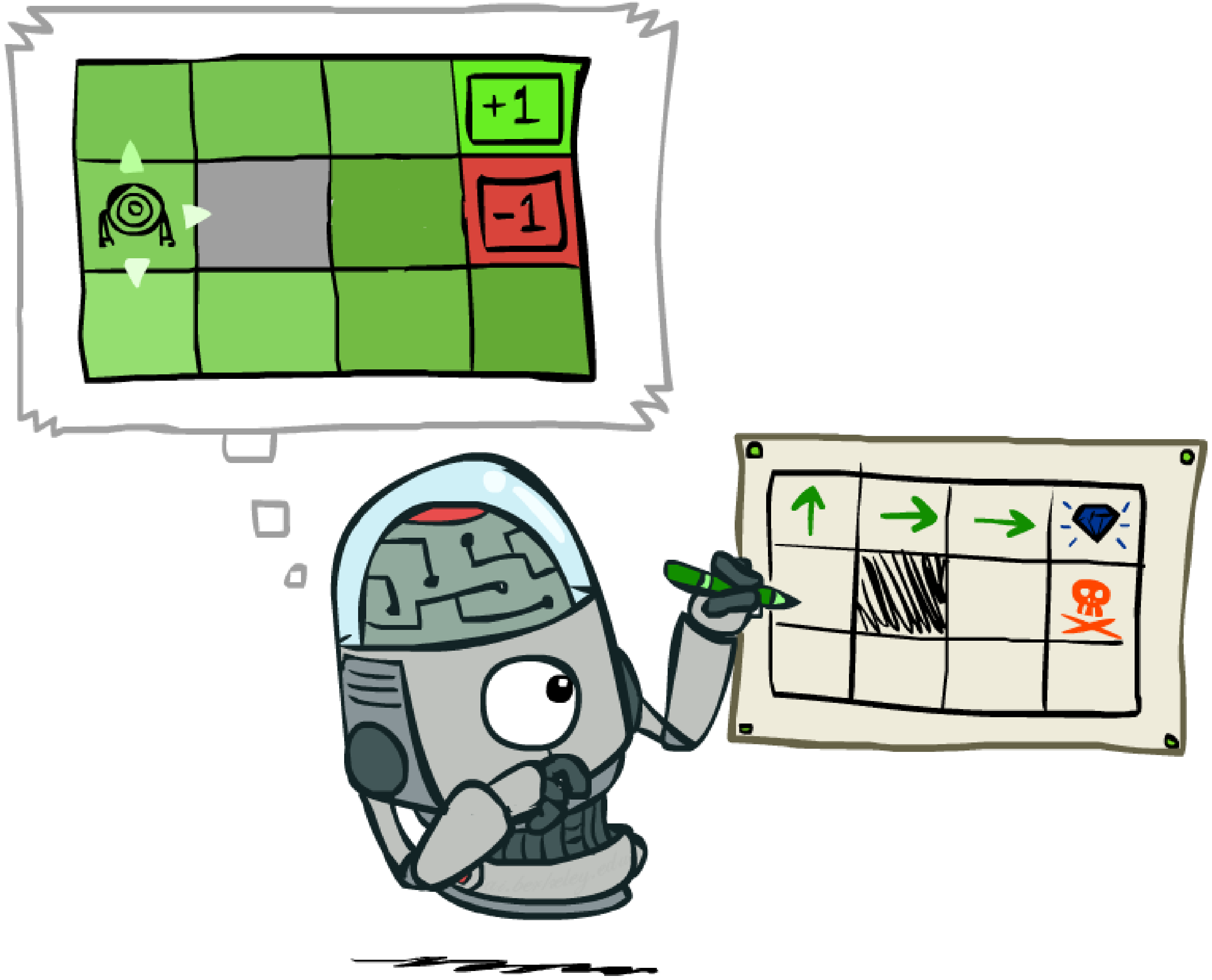
- Value iteration computes them:

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

- Value iteration is just a fixed point solution method
  - … though the V_k vectors are also interpretable as time-limited values

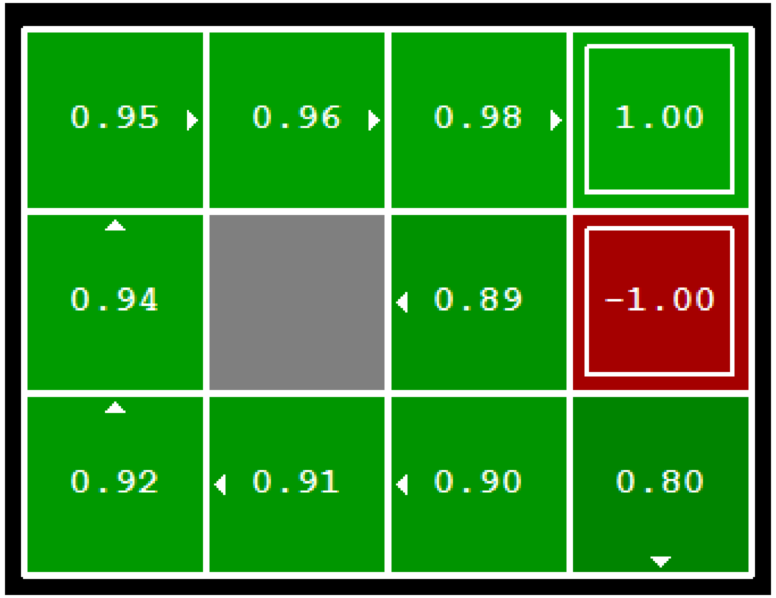


Policy Extraction



Computing Actions from Values

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



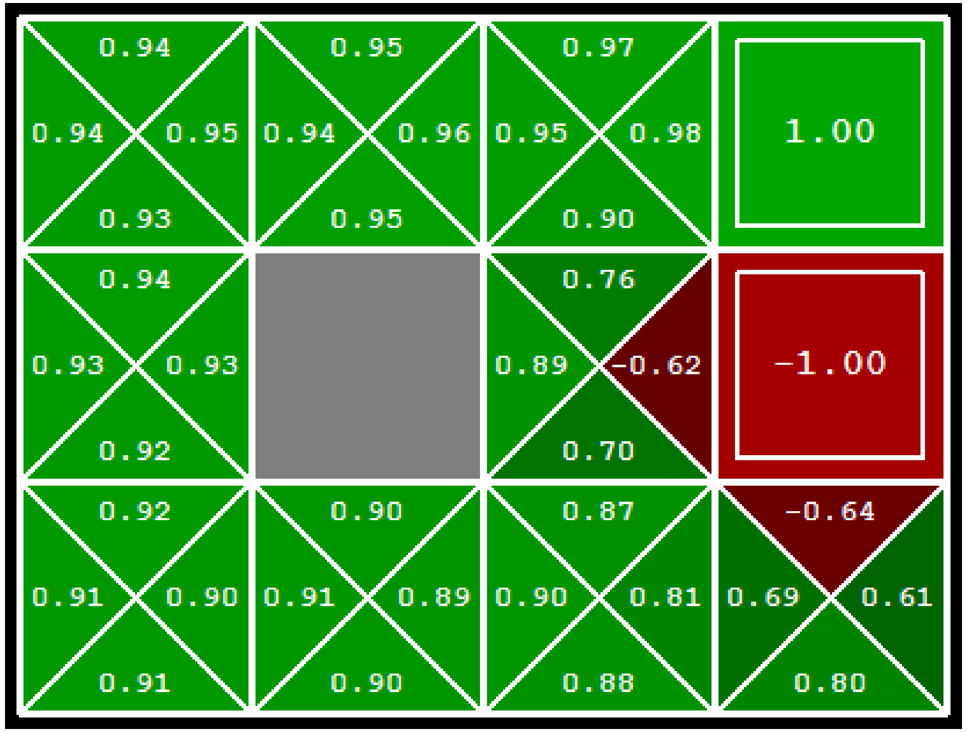
- How should we act?
  - It's not obvious!
- We need to do a mini-expectimax (one step)

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

- 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:

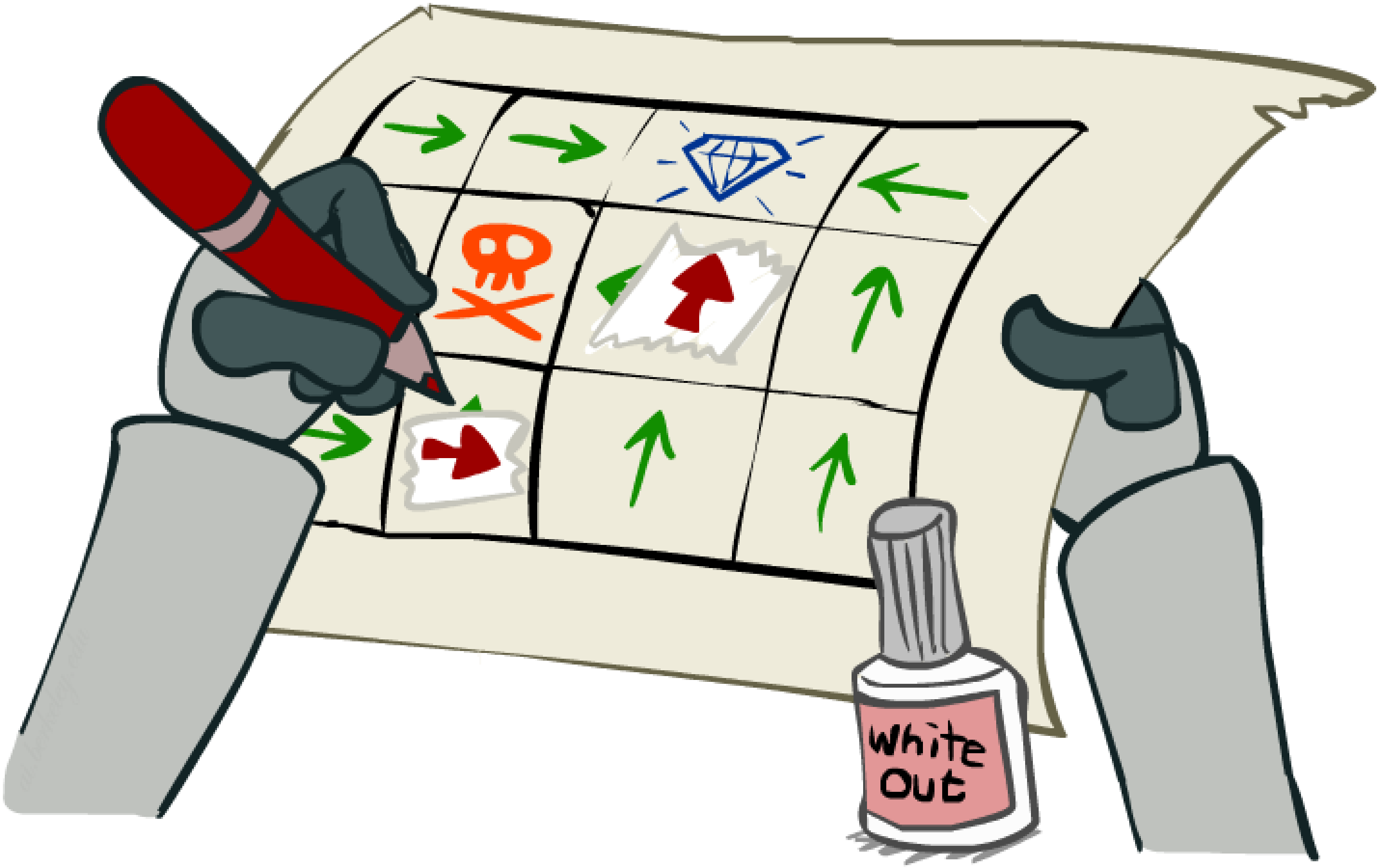


- How should we act?
  - Completely trivial to decide!

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

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

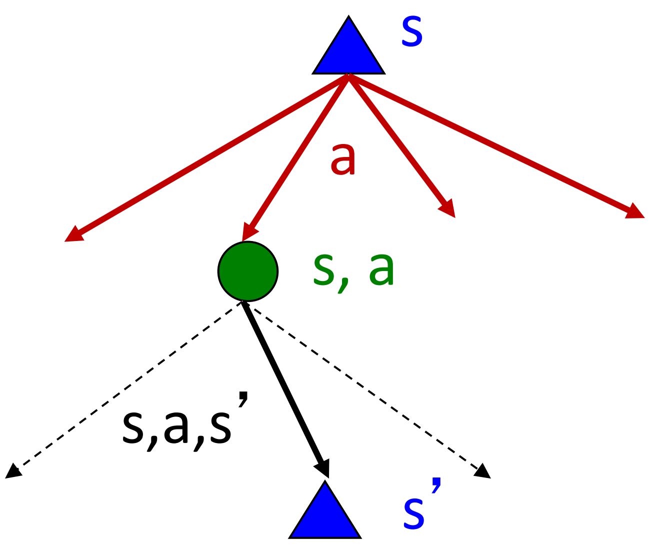
Policy Iteration



Problems with Value Iteration

- Value iteration repeats the Bellman updates:

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



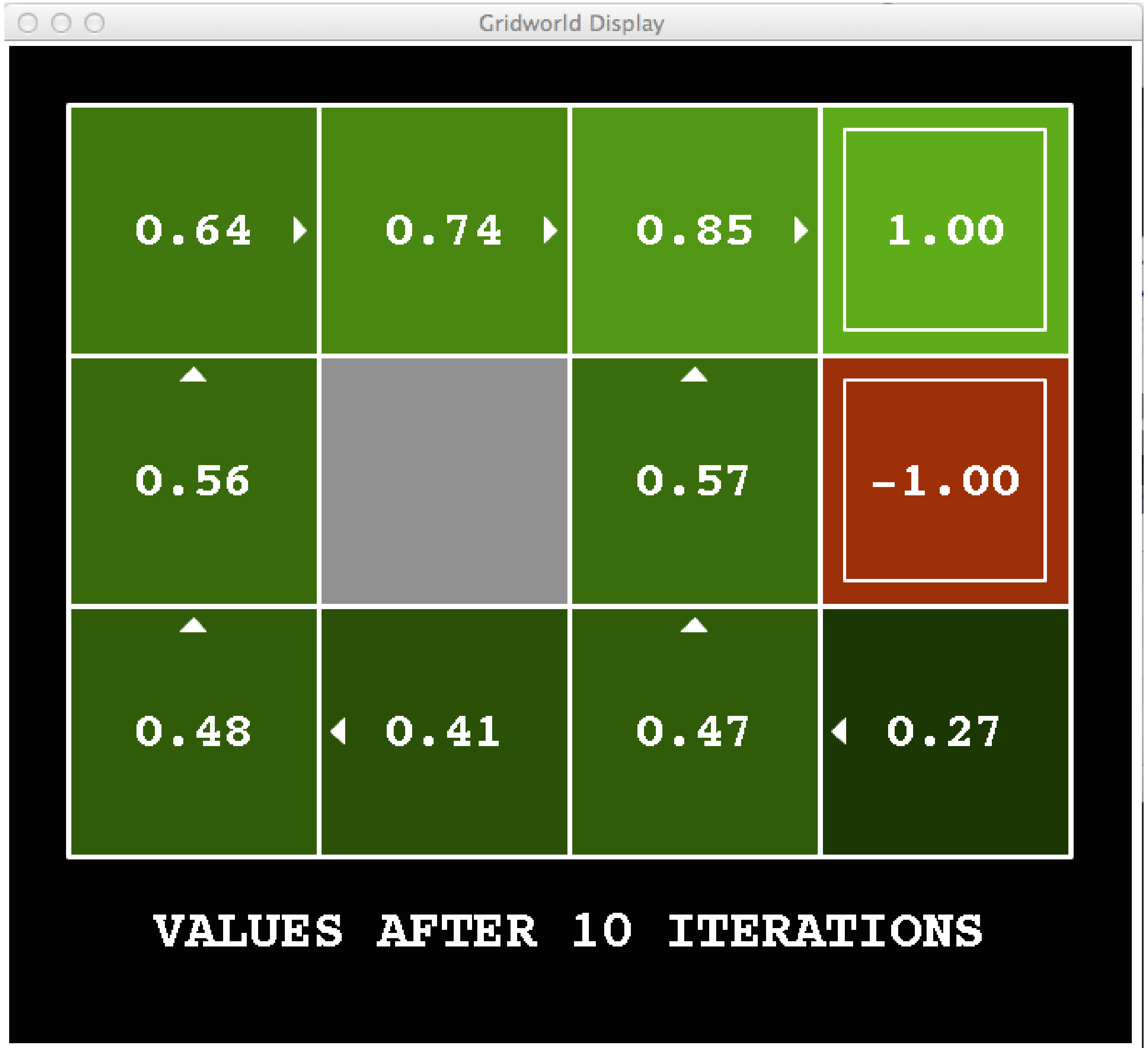
- Problem 1: It's slow – $O(S^2A)$ per iteration

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

- Problem 3: The policy often converges long before the values

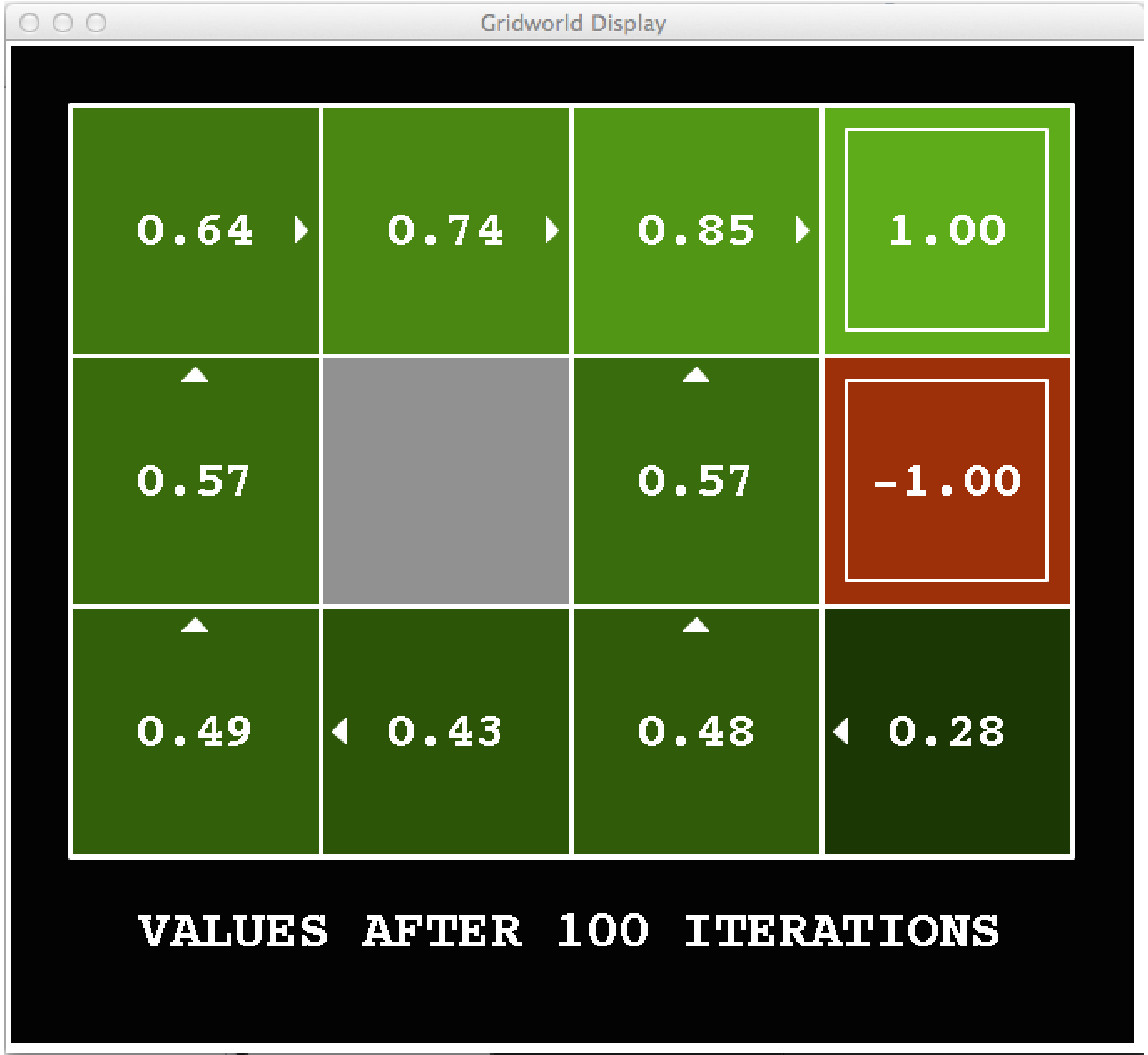
[Demo: value iteration (L9D2)]

k=10



Noise = 0.2
Discount = 0.9
Living reward = 0

k=100



Noise = 0.2
Discount = 0.9
Living reward = 0

Policy Iteration

Step 0: start w/ random π

- Alternative approach for optimal values:
  - Step 1: Policy evaluation: calculate utilities for some fixed policy (not optimal utilities!) 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

- Evaluation: For fixed current policy π_i, find values with policy evaluation:
  - Iterate until values converge:

alternative: run for K iterations

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

- Improvement: For fixed values, get a better policy using policy extraction
  - One-step look-ahead:

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

stop when $\pi_{i+1} = \pi_i$

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

Aside: Linear Programming

x_1, x_2 : Decision variables

$$
\begin{aligned}
\max \quad & 350x_1 + 300x_2 \\
\text{subject to} \quad & \\
& x_1 + x_2 \leq 200 \\
& 9x_1 + 6x_2 \leq 1566 \\
& 12x_1 + 16x_2 \leq 2880 \\
& x_1, x_2 \geq 0
\end{aligned}
$$

Objective function

Constraints

General form

$$
\begin{aligned}
\max \quad & c^T x \\
\text{s.t.} \quad & Ax \geq b \\
& x \geq 0
\end{aligned}
$$

Primal Linear Programming Solutions

Basic idea: we can capture the constraint

$$V(s) \geq R(s) + \gamma \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s, |s, a) V(s')$$

via the set of $|\mathcal{A}|$ linear constraints

$Ax \geq b$

$$V(s) \geq R(s) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V(s'), \quad \forall a \in \mathcal{A}$$

Primal Linear Programming Solutions

Now consider the linear program

We want $V^*(s)$ $\forall s$

$$\underset{V}{\text{minimize}} \quad \sum_{s} V(s)$$

$$\text{subject to} \quad V(s) \geq R(s) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V(s'), \quad \forall a \in \mathcal{A}, s \in \mathcal{S}$$

Dual Linear Programming for MDPs

- One-to-one correspondence between stochastic policies and state-action occupancy frequencies:

$$\pi : S \times A \mapsto [0, 1] \quad \Longleftrightarrow \quad u_{sa} = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t \mathbf{1}_{(s_t=s, a_t=a)}\right]$$

Expected discounted # times we take a in s

Linear Programming for MDPs

- One-to-one correspondence between stochastic policies and state-action occupancy frequencies:

$$\pi(a|s) = \frac{u_{sa}}{\sum_a u_{sa}} \quad \Longleftrightarrow \quad u_{sa} = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \mathbf{1}_{(s_t=s, a_t=a)}\right]$$

Linear Programming for MDPs

- One-to-one correspondence between stochastic policies and state-action occupancy frequencies:

$$\max_u \sum_{s,a} r_{sa} u_{sa}$$

Reward for taking action a in state s

$R(s,a)$

such that

$$\sum_a u_{sa} = p_0(s) + \gamma \sum_{s',a} u_{s'a} P(s', a, s), \forall s$$

$$u_{sa} \geq 0, \forall s, a$$

Linear Programming for MDPs

- One-to-one correspondence between stochastic policies and state-action occupancy frequencies:

$$\max_{u} \sum_{s,a} r_{sa} \boxed{u_{sa}}$$

$$\boxed{u_{sa} = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \mathbf{1}_{(s_t=s, a_t=a)}\right]}$$

State Occupancies

such that

Bellman Flow constraint ->

$$\sum_{a} \boxed{u_{sa}} = p_0(s) + \gamma \sum_{s',a} \boxed{u_{s'a}} P(s', a, s), \forall s$$

$$\boxed{u_{sa}} \geq 0, \forall s, a$$

Linear Programming for MDPs

- One-to-one correspondence between stochastic policies and state-action occupancy frequencies:

$$\max_{u} \quad \sum_{s,a} r_{sa} u_{sa}$$

such that

Initial state distribution

$$\sum_{a} u_{sa} = \boxed{p_0(s)} + \gamma \sum_{s',a} u_{s'a} P(s', a, s), \forall s$$

$$u_{sa} \geq 0, \forall s, a$$

Linear Programming for MDPs

- One-to-one correspondence between stochastic policies and state-action occupancy frequencies:

$$\max_{u} \quad \sum_{s,a} r_{sa} u_{sa}$$

such that

Transition Probability

$$\sum_{a} u_{sa} = p_0(s) + \gamma \sum_{s',a} u_{s'a} \boxed{P(s', a, s)}, \forall s$$

$$u_{sa} \geq 0, \forall s, a$$

Linear Programming for MDPs

- One-to-one correspondence between stochastic policies and state-action occupancy frequencies:

$$\max_u \sum_{s,a} r_{sa} u_{sa}$$

Reward for taking action a in state s

$$\text{such that}$$

Initial state distribution

Transition Probability

$$\sum_a u_{sa} = p_0(s) + \gamma \sum_{s',a} u_{s'a} P(s', a, s), \forall s$$

State Occupancies $u_{sa} \geq 0, \forall s, a$

Discount factor

Linear Programming for MDPs

- One-to-one correspondence between stochastic policies and state-action occupancy frequencies:

$$\max_{u} \quad \sum_{s,a} r_{sa} u_{sa}$$

How often do I start in s?

How often do I visit other states s' and then transition to state s?

such that

How often do I visit state s?

$$\boxed{\sum_{a} u_{sa}} = \boxed{p_0(s)} + \boxed{\gamma \sum_{s',a} u_{s'a} P(s', a, s)}, \forall s$$

$$u_{sa} \geq 0, \forall s, a$$

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 computations