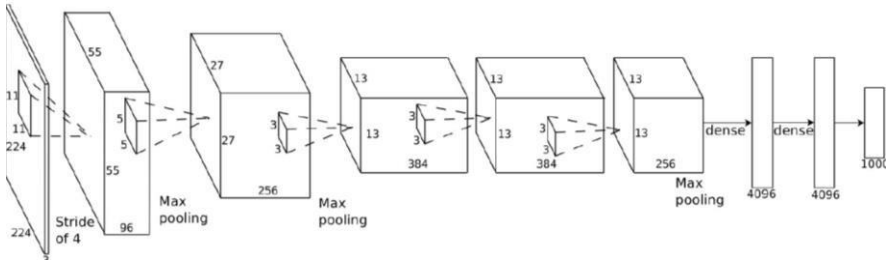
CS 8803
Deep Reinforcement Learning

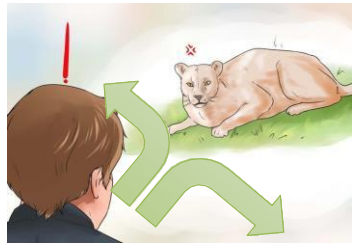
Lec 2: Intro to RL
Fall 2024

Animesh Garg

Definitions

Terminology & notation

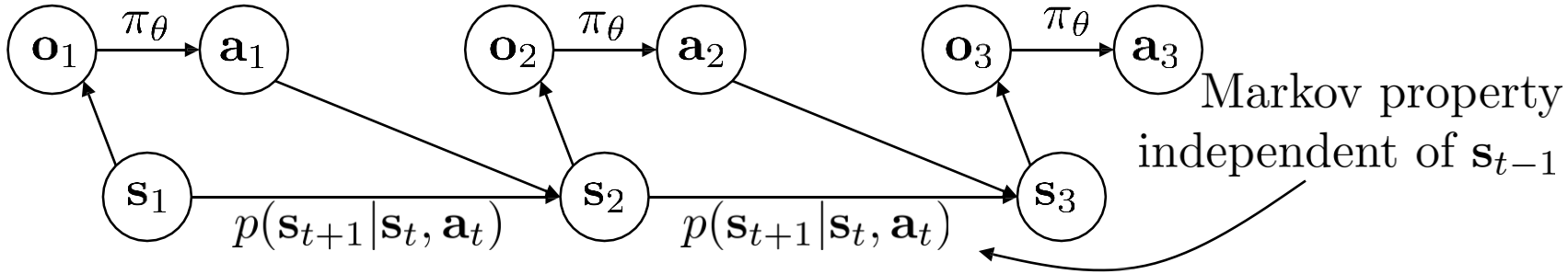




$\mathbf{o}_t$ $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ $\mathbf{a}_t$

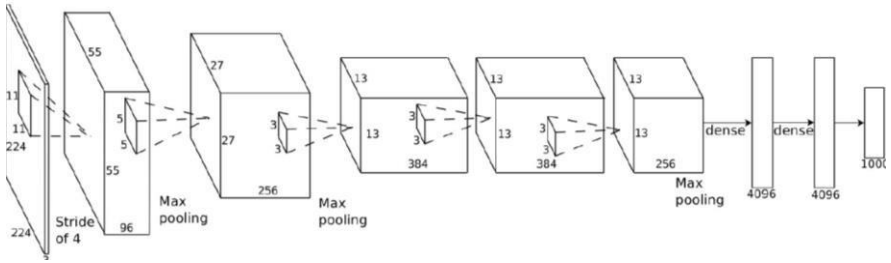
$\mathbf{s}_t$ – state
$\mathbf{o}_t$ – observation
$\mathbf{a}_t$ – action

$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ – policy
$\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ – policy (fully observed)

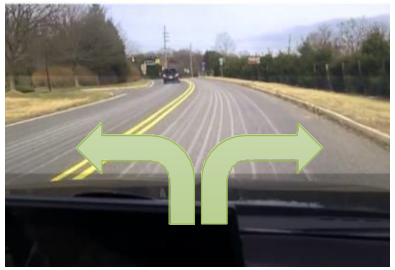


Imitation Learning

$\mathbf{o}_t$



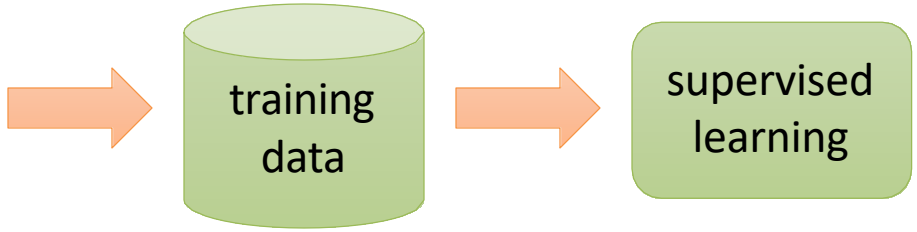
$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$



$\mathbf{a}_t$

$\mathbf{o}_t$

$\mathbf{a}_t$

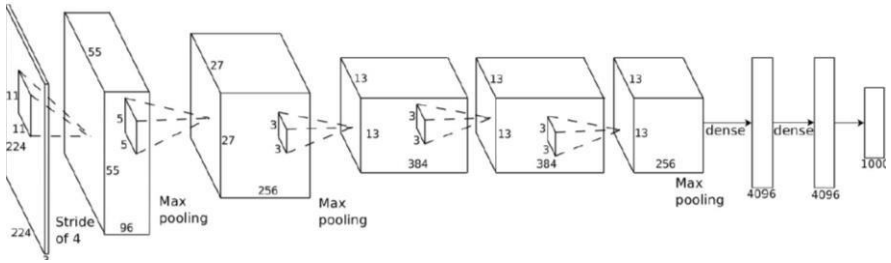


$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$

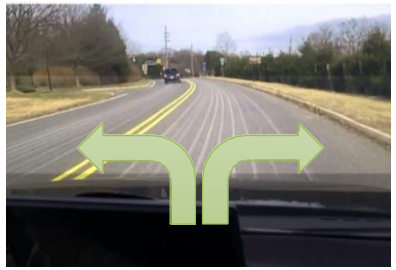
Images: Bojarski et al. ‘16, NVIDIA

Reward functions

$\mathbf{o}_t$



$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$



$\mathbf{a}_t$

which action is better or worse?

$r(\mathbf{s}, \mathbf{a})$: reward function
tells us which states and actions are better

$\mathbf{s}$, $\mathbf{a}$, $r(\mathbf{s}, \mathbf{a})$, and $p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ define Markov decision process

high reward

low reward

Definitions

Andrey Markov

Markov chain

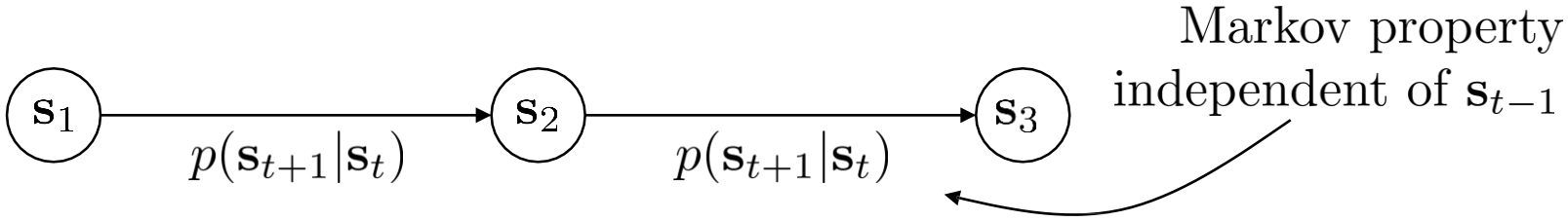
$\mathcal{M} = \{\mathcal{S}, \mathcal{T}\}$

$\mathcal{S}$ – state space — states $s \in \mathcal{S}$ (discrete or continuous)

$\mathcal{T}$ – transition operator — $p(s_{t+1}|s_t)$

why "operator"? — let $\mu_{t,i} = p(s_t = i)$ — $\vec{\mu}_t$ is a vector of probabilities

let $\mathcal{T}_{i,j} = p(s_{t+1} = i | s_t = j)$ — then $\vec{\mu}_{t+1} = \mathcal{T}\vec{\mu}_t$



Definitions

Markov decision process $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{T}, r\}$

$\mathcal{S}$ – state space states $s \in \mathcal{S}$ (discrete or continuous)

$\mathcal{A}$ – action space actions $a \in \mathcal{A}$ (discrete or continuous)

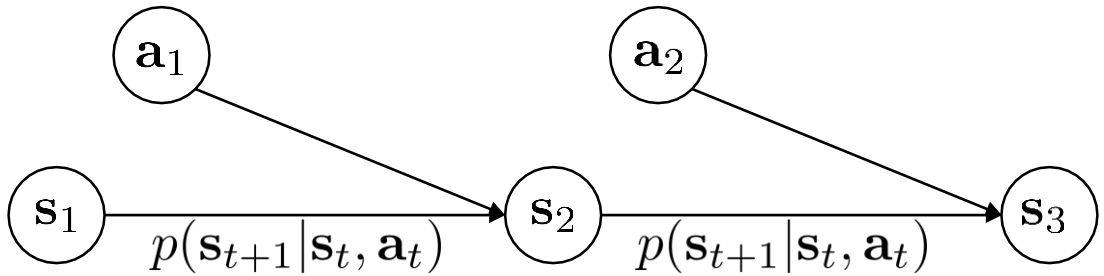
$\mathcal{T}$ – transition operator (now a tensor!)

let $\mu_{t,j} = p(s_t = j)$

let $\xi_{t,k} = p(a_t = k)$

let $\mathcal{T}_{i,j,k} = p(s_{t+1} = i | s_t = j, a_t = k)$

$$\mu_{t+1,i} = \sum_{j,k} \mathcal{T}_{i,j,k} \mu_{t,j} \xi_{t,k}$$



Richard Bellman

Definitions

Markov decision process $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{T}, r\}$

$\mathcal{S}$ – state space states $s \in \mathcal{S}$ (discrete or continuous)

$\mathcal{A}$ – action space actions $a \in \mathcal{A}$ (discrete or continuous)

$\mathcal{T}$ – transition operator (now a tensor!)

r – reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$

$r(s_t, a_t)$ – reward

Richard Bellman

Definitions

partially observed Markov decision process $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{E}, r\}$

$\mathcal{S}$ – state space states $s \in \mathcal{S}$ (discrete or continuous)

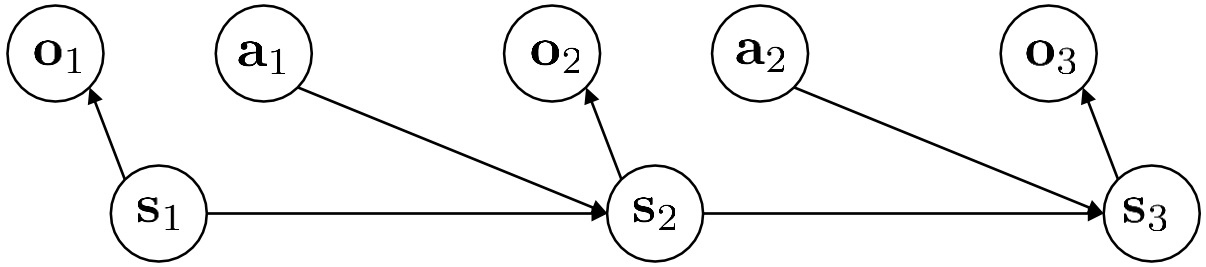
$\mathcal{A}$ – action space actions $a \in \mathcal{A}$ (discrete or continuous)

$\mathcal{O}$ – observation space observations $o \in \mathcal{O}$ (discrete or continuous)

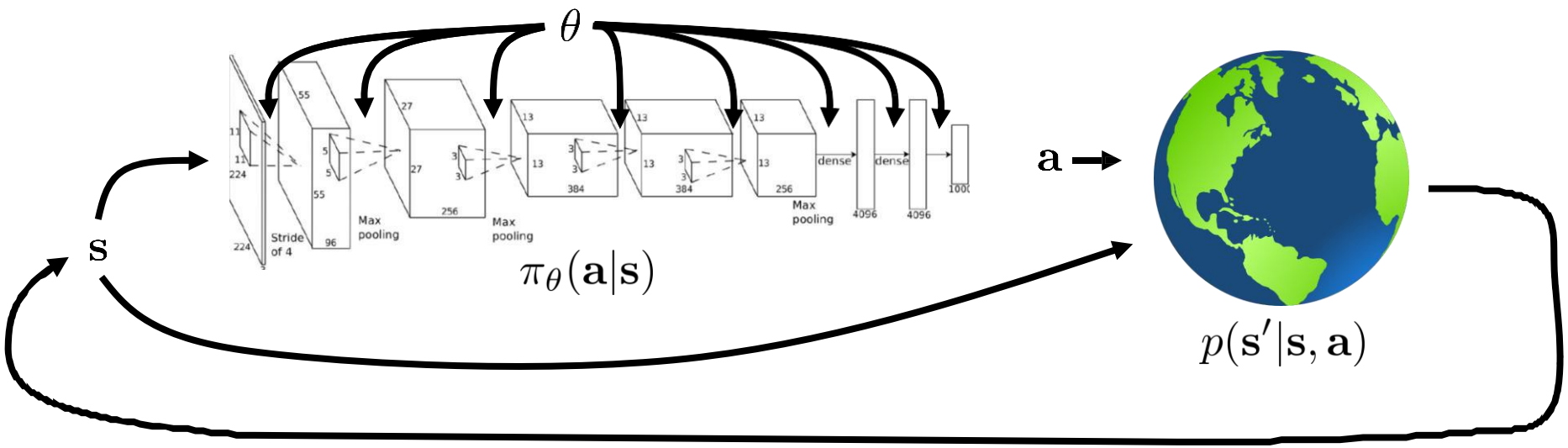
$\mathcal{T}$ – transition operator (like before)

$\mathcal{E}$ – emission probability $p(o_t|s_t)$

r – reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$



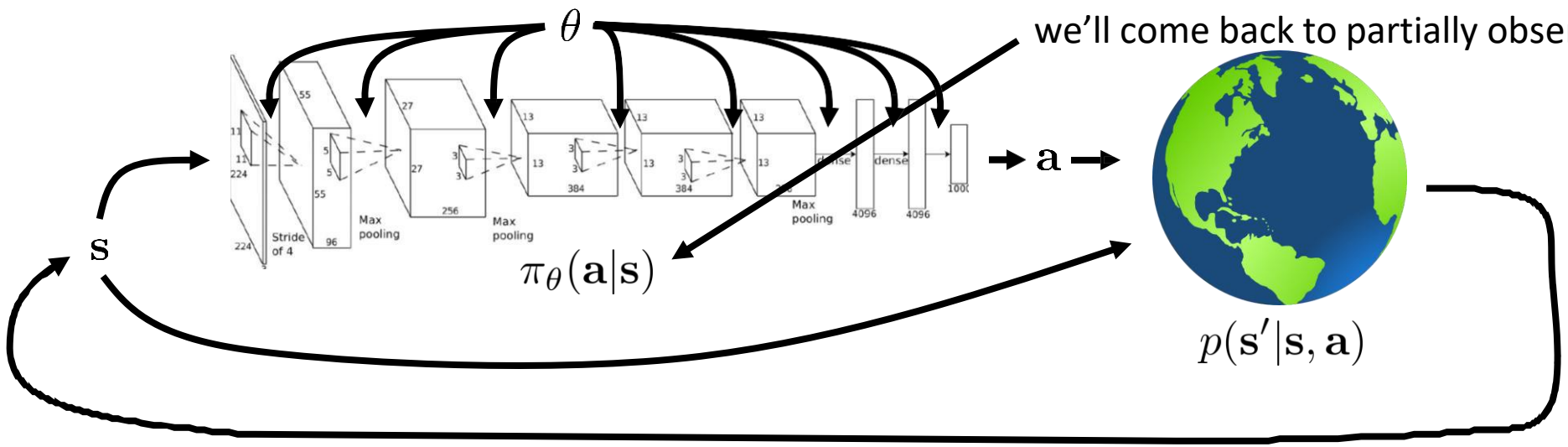
The goal of reinforcement learning



$$\underbrace{p_\theta(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T)}_{p_\theta(\tau)} = p(\mathbf{s}_1) \prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

$$\theta^\star = \arg\max_\theta E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

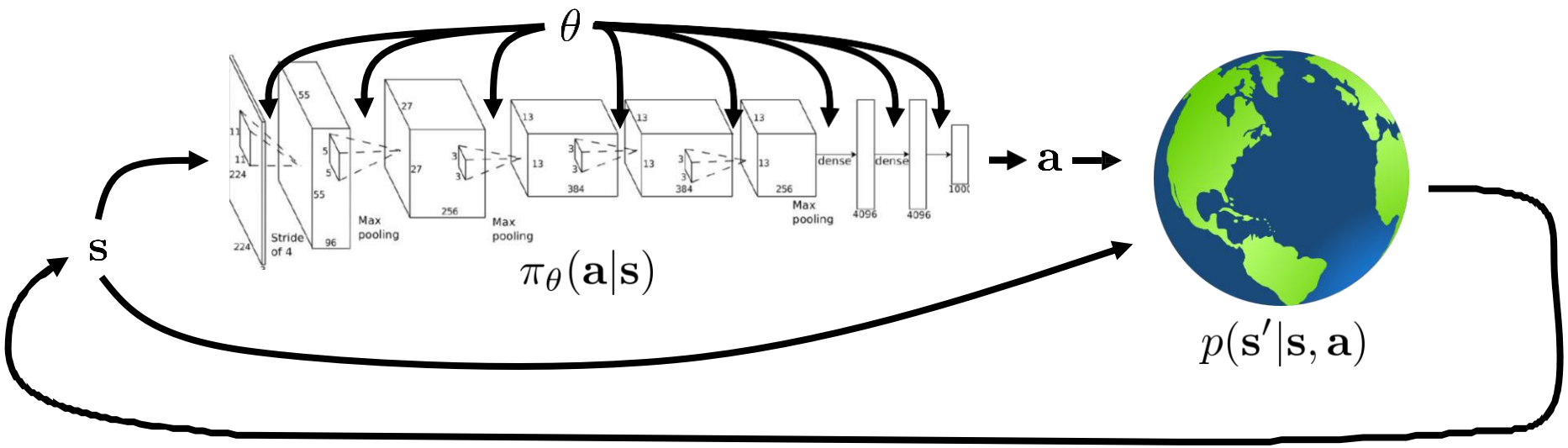
The goal of reinforcement learning



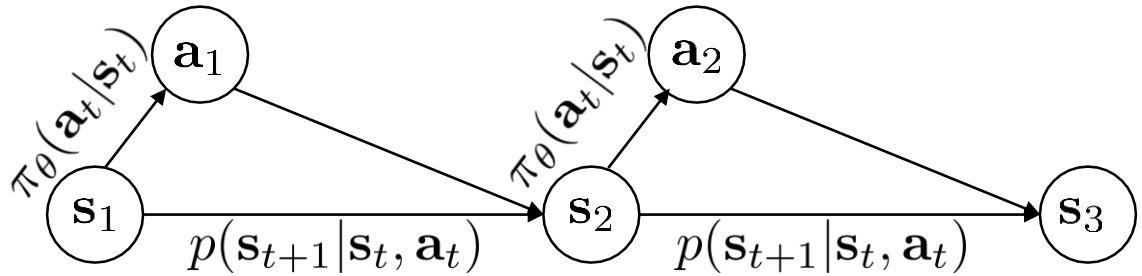
$$\underbrace{p_\theta(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T)}_{p_\theta(\tau)} = p(\mathbf{s}_1) \prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

$$\theta^\star = \arg\max_\theta E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

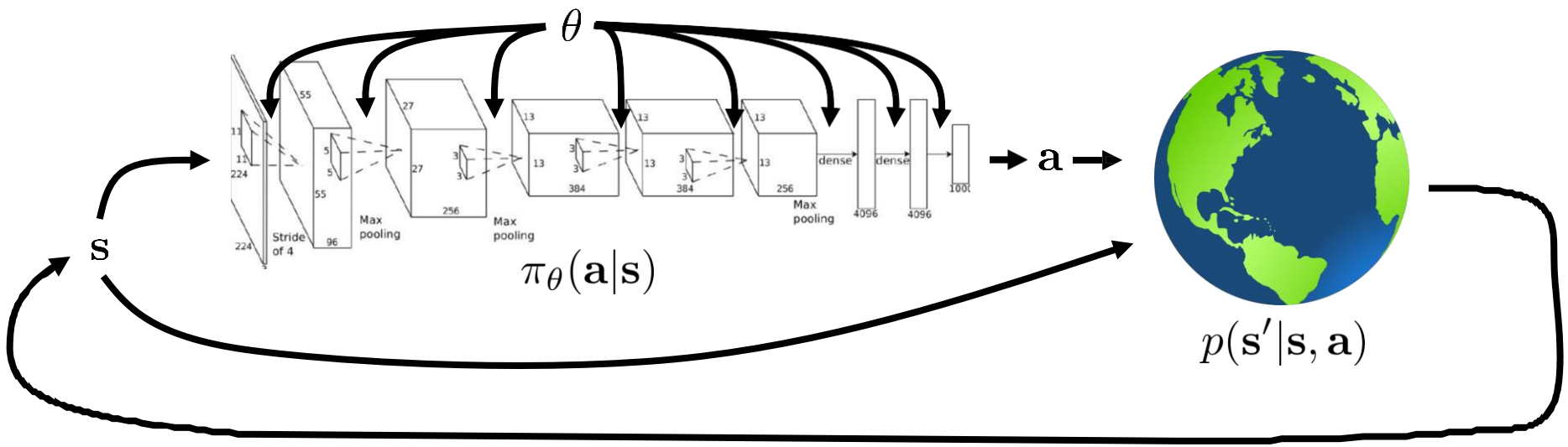
The goal of reinforcement learning



$$\underbrace{p_\theta(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T)}_{p_\theta(\tau)} = p(\mathbf{s}_1) \prod_{t=1}^{T} \underbrace{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t) p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)}_{\text{Markov chain on } (\mathbf{s}, \mathbf{a})}$$

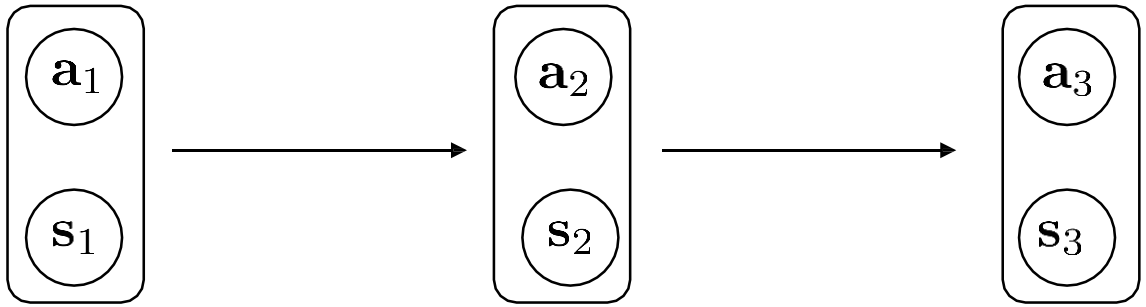


The goal of reinforcement learning



$$\underbrace{p_\theta(\mathbf{s}_1,\mathbf{a}_1,\ldots,\mathbf{s}_T,\mathbf{a}_T)}_{p_\theta(\tau)} = p(\mathbf{s}_1)\prod_{t=1}^{T}\underbrace{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)}_{\text{Markov chain on } (\mathbf{s},\mathbf{a})}$$

$$p((\mathbf{s}_{t+1},\mathbf{a}_{t+1})|(\mathbf{s}_t,\mathbf{a}_t)) = p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)\pi_\theta(\mathbf{a}_{t+1}|\mathbf{s}_{t+1})$$



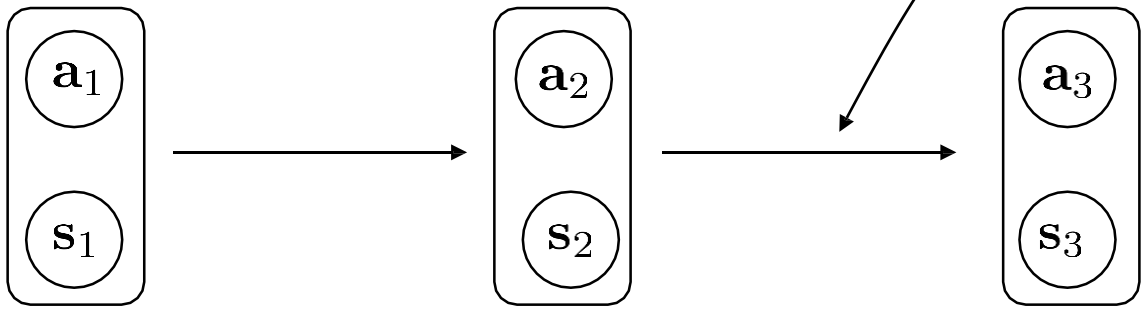
Finite horizon case: state-action marginal

$$\theta^\star = \arg\max_\theta E_{\tau\sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)\right]$$

$$= \arg\max_\theta \sum_{t=1}^{T} E_{(\mathbf{s}_t,\mathbf{a}_t)\sim p_\theta(\mathbf{s}_t,\mathbf{a}_t)}[r(\mathbf{s}_t, \mathbf{a}_t)]$$

$p_\theta(\mathbf{s}_t, \mathbf{a}_t)$ state-action marginal

$$p((\mathbf{s}_{t+1}, \mathbf{a}_{t+1})|(\mathbf{s}_t, \mathbf{a}_t)) =$$
$$p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)\pi_\theta(\mathbf{a}_{t+1}|\mathbf{s}_{t+1})$$



Infinite horizon case: stationary distribution

$$\theta^\star = \arg\max_\theta \sum_{t=1}^{T} E_{(\mathbf{s}_t,\mathbf{a}_t)\sim p_\theta(\mathbf{s}_t,\mathbf{a}_t)}[r(\mathbf{s}_t,\mathbf{a}_t)]$$

what if $T = \infty$?

does $p(\mathbf{s}_t, \mathbf{a}_t)$ converge to a *stationary* distribution?

$\mu = \mathcal{T}\mu$

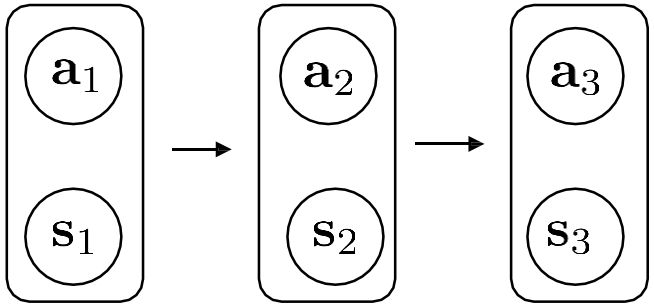
stationary = the same before and after transition

$(\mathcal{T} - \mathbf{I})\mu = 0$

μ is eigenvector of $\mathcal{T}$ with eigenvalue 1!

(always exists under some regularity conditions)

$\mu = p_\theta(\mathbf{s}, \mathbf{a})$ stationary distribution



state-action transition operator

$$\begin{pmatrix} \mathbf{s}_{t+1} \\ \mathbf{a}_{t+1} \end{pmatrix} = \mathcal{T} \begin{pmatrix} \mathbf{s}_t \\ \mathbf{a}_t \end{pmatrix} \quad \begin{pmatrix} \mathbf{s}_{t+k} \\ \mathbf{a}_{t+k} \end{pmatrix} = \mathcal{T}^k \begin{pmatrix} \mathbf{s}_t \\ \mathbf{a}_t \end{pmatrix}$$

Infinite horizon case: stationary distribution

$$\theta^\star = \arg\max_\theta \frac{1}{T} \sum_{t=1}^{T} E_{(\mathbf{s}_t, \mathbf{a}_t) \sim p_\theta(\mathbf{s}_t, \mathbf{a}_t)}[r(\mathbf{s}_t, \mathbf{a}_t)] \rightarrow E_{(\mathbf{s}, \mathbf{a}) \sim p_\theta(\mathbf{s}, \mathbf{a})}[r(\mathbf{s}, \mathbf{a})]$$

(in the limit as $T \rightarrow \infty$)

what if $T = \infty$?

does $p(\mathbf{s}_t, \mathbf{a}_t)$ converge to a *stationary* distribution?

$\mu = \mathcal{T}\mu$

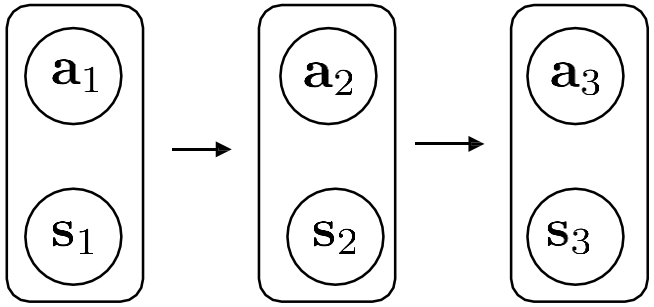
stationary = the same before and after transition

$(\mathcal{T} - \mathbf{I})\mu = 0$

μ is eigenvector of $\mathcal{T}$ with eigenvalue 1!

(always exists under some regularity conditions)

$\mu = p_\theta(\mathbf{s}, \mathbf{a})$ stationary distribution



state-action transition operator

$$\begin{pmatrix} \mathbf{s}_{t+1} \\ \mathbf{a}_{t+1} \end{pmatrix} = \mathcal{T} \begin{pmatrix} \mathbf{s}_t \\ \mathbf{a}_t \end{pmatrix} \quad \begin{pmatrix} \mathbf{s}_{t+k} \\ \mathbf{a}_{t+k} \end{pmatrix} = \mathcal{T}^k \begin{pmatrix} \mathbf{s}_t \\ \mathbf{a}_t \end{pmatrix}$$

Expectations and stochastic systems

$$\theta^\star = \arg\max_\theta E_{(\mathbf{s},\mathbf{a})\sim p_\theta(\mathbf{s},\mathbf{a})}[r(\mathbf{s},\mathbf{a})]$$

infinite horizon case

$$\theta^\star = \arg\max_\theta \sum_{t=1}^{T} E_{(\mathbf{s}_t,\mathbf{a}_t)\sim p_\theta(\mathbf{s}_t,\mathbf{a}_t)}[r(\mathbf{s}_t,\mathbf{a}_t)]$$

finite horizon case

In RL, we almost always care about *expectations*

$r(\mathbf{x})$ – *not* smooth

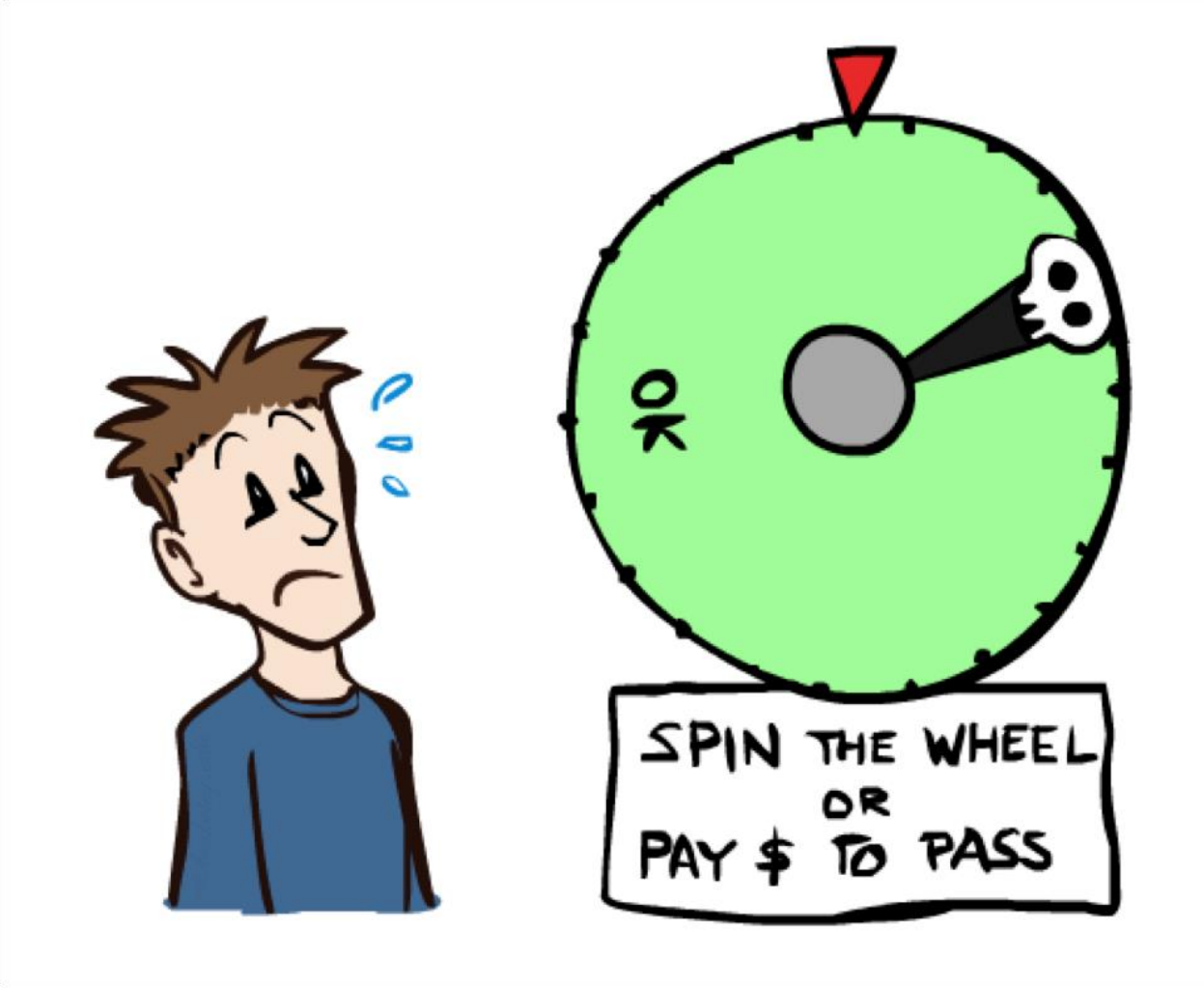
$\pi_\theta(\mathbf{a} = \text{fall}) = \theta$

$E_{\pi_\theta}[r(\mathbf{x})]$ – *smooth* in θ!

Outline

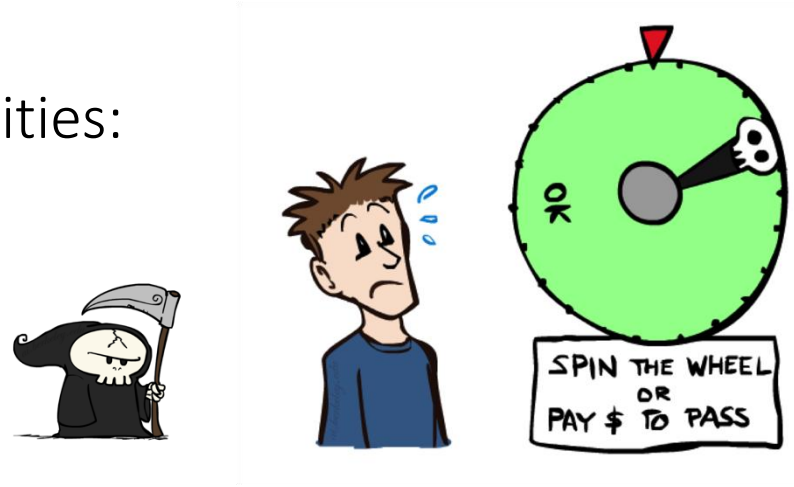
- Utilities to MDPs
- Value Iteration
- Policy Extraction
- Policy Iteration
- Convergence Analysis

Human Utilities

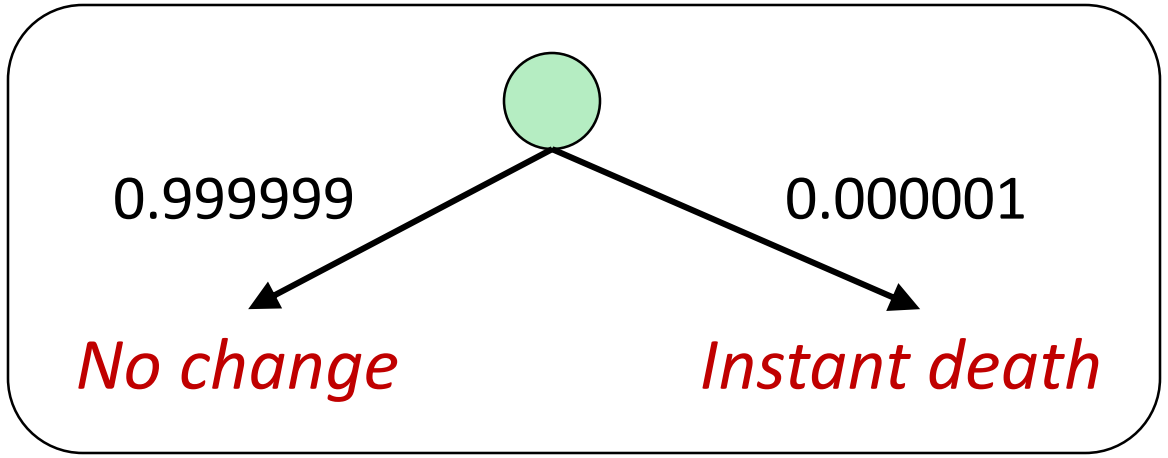


Human Utilities

- Utilities map states to real numbers. Which numbers?
- Standard approach to assessment (elicitation) of human utilities:
  - Compare a prize A to a standard lottery L_p between
    - "best possible prize" u_+ with probability p
    - "worst possible catastrophe" u_- with probability 1-p
  - Adjust lottery probability p until indifference: A ~ L_p
  - Resulting p is a utility in [0,1]



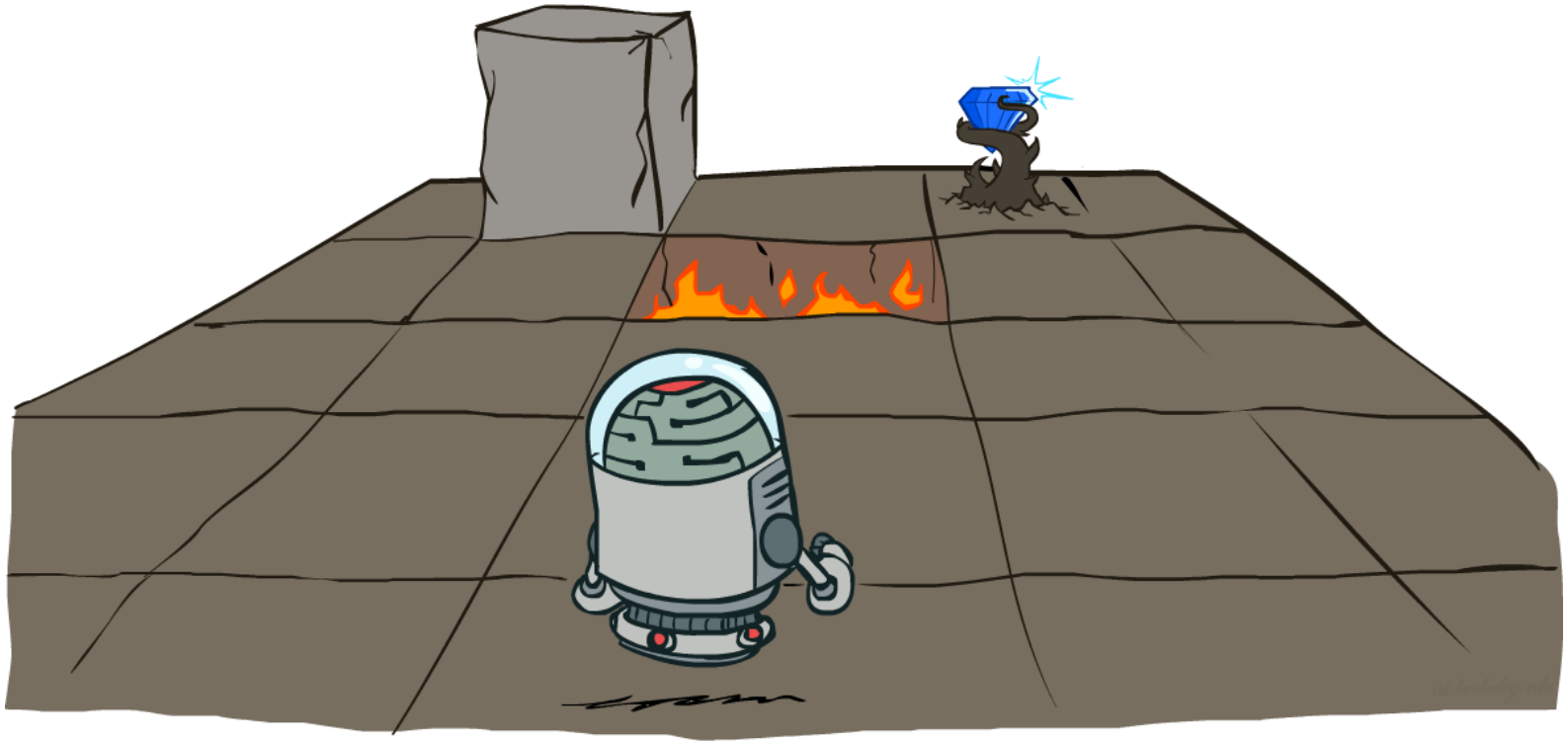
Pay $30 ~



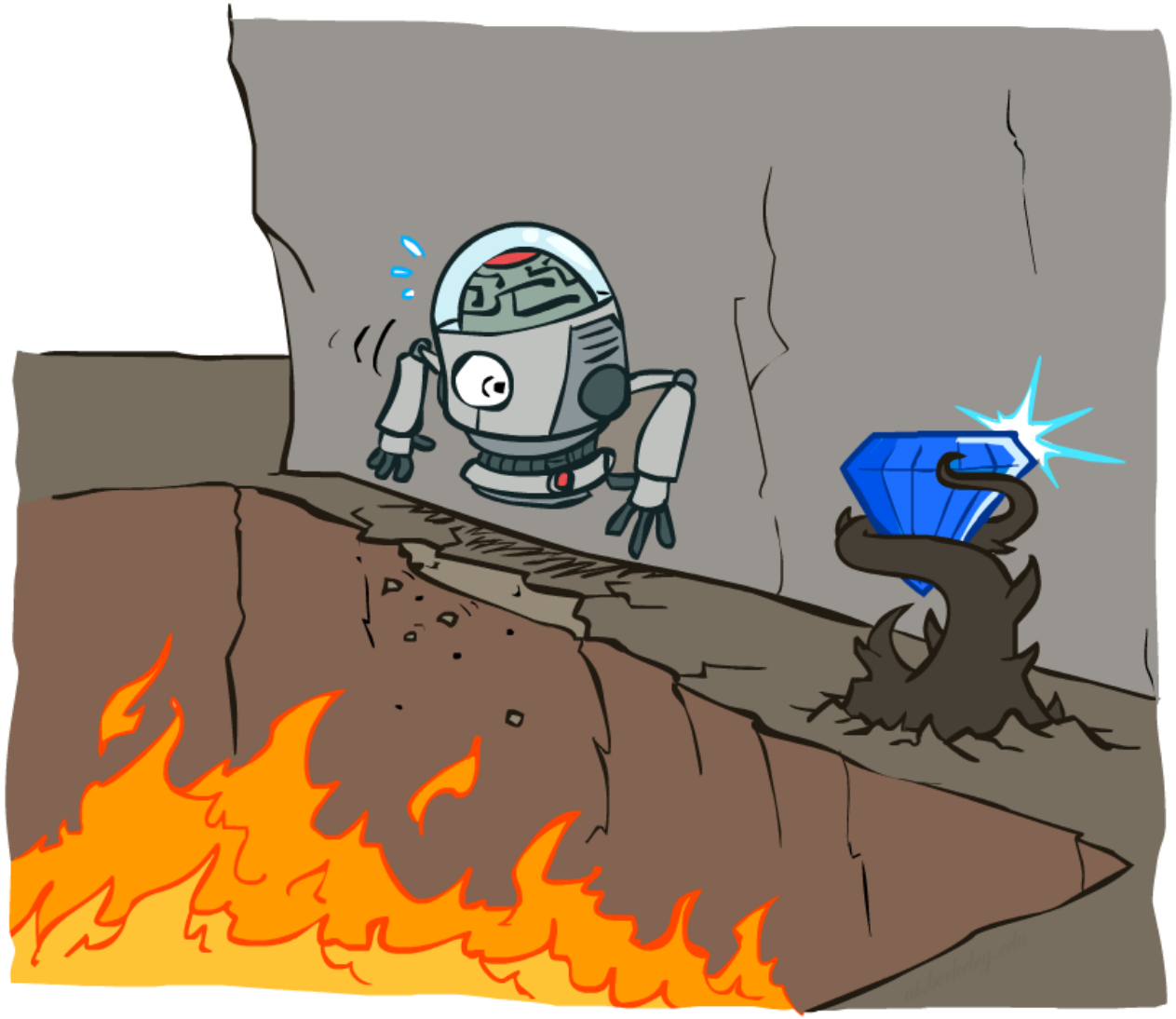
Example: Human Rationality?

- Famous example of Allais (1953)
  - A: [0.8, $4k; 0.2, $0]
  - B: [1.0, $3k; 0.0, $0]
  - C: [0.2, $4k; 0.8, $0]
  - D: [0.25, $3k; 0.75, $0]
- Most people prefer B > A, C > D
- But if U($0) = 0, then
  - B > A $\Rightarrow$ U($3k) > 0.8 U($4k) = $3200

Markov Decision Process

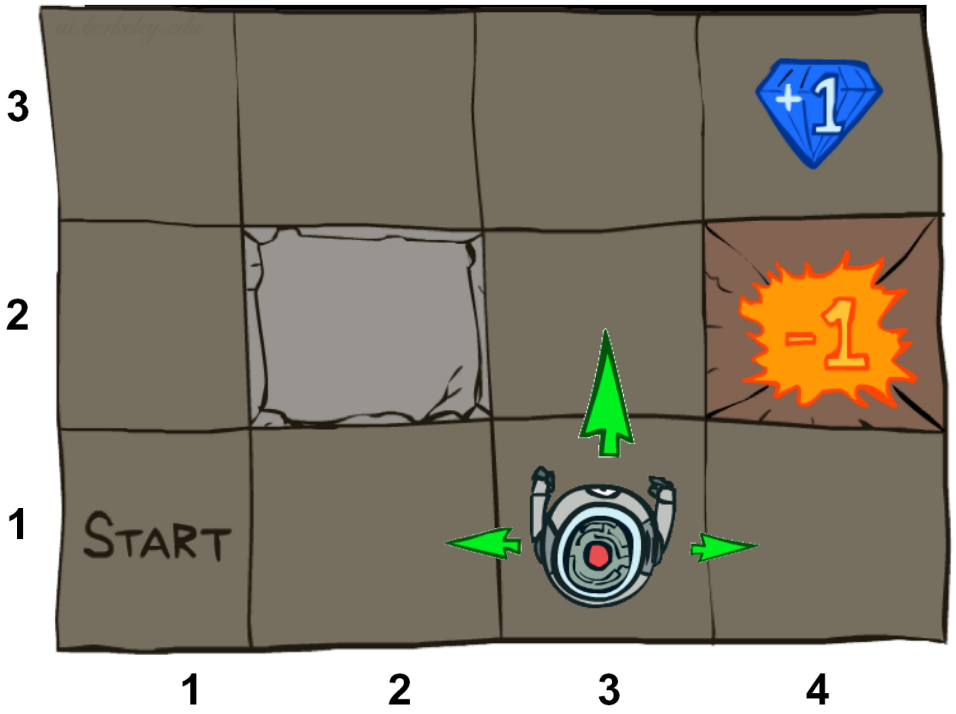


Non-Deterministic Search



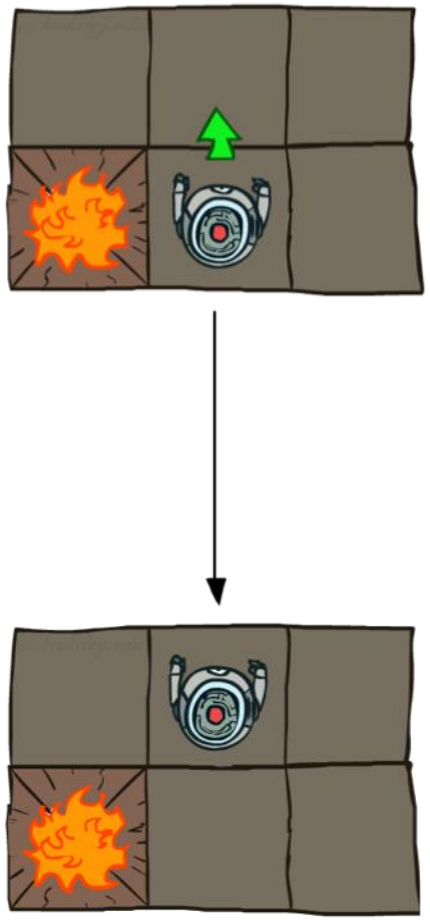
Example: Grid World

- 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

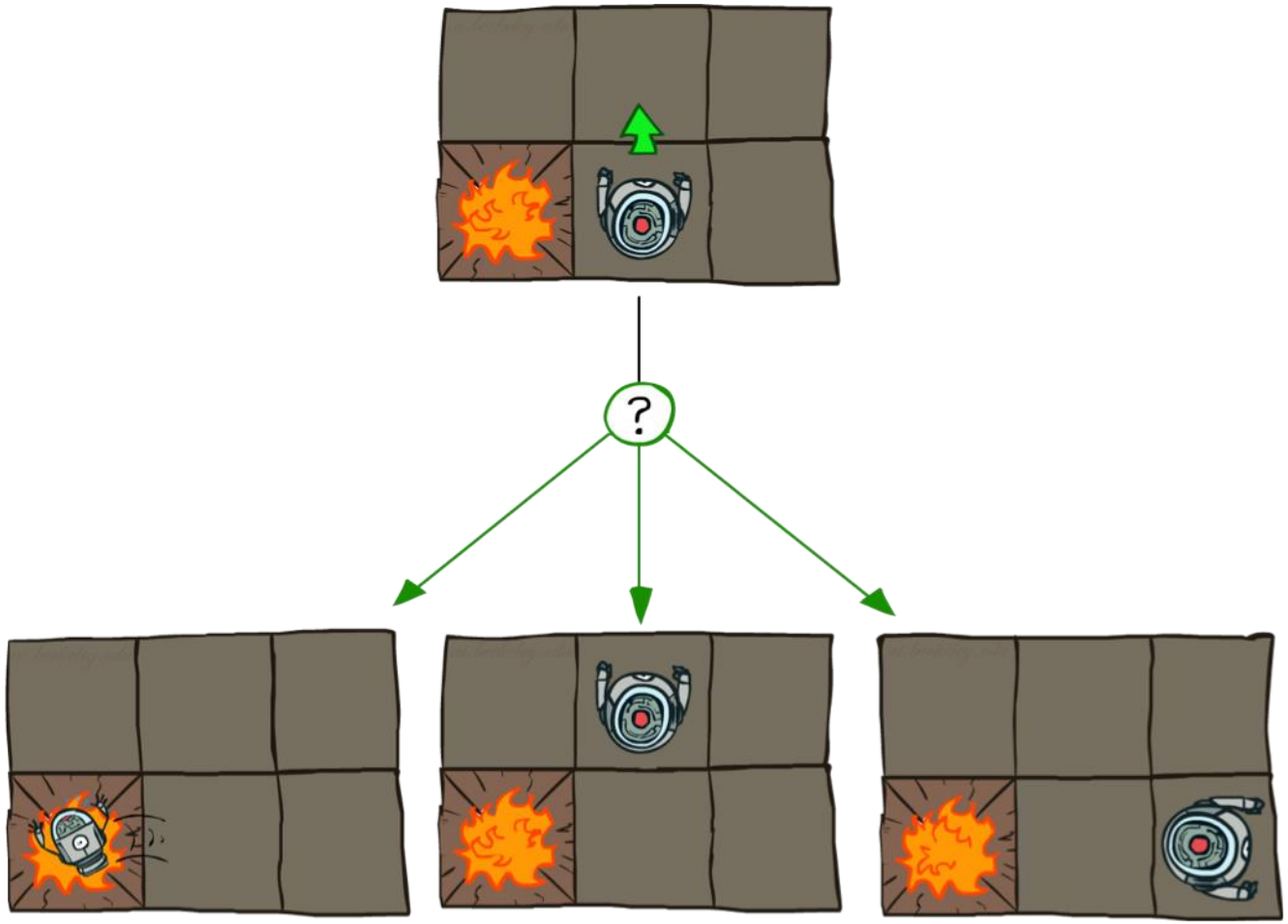


Grid World Actions

Deterministic Grid World

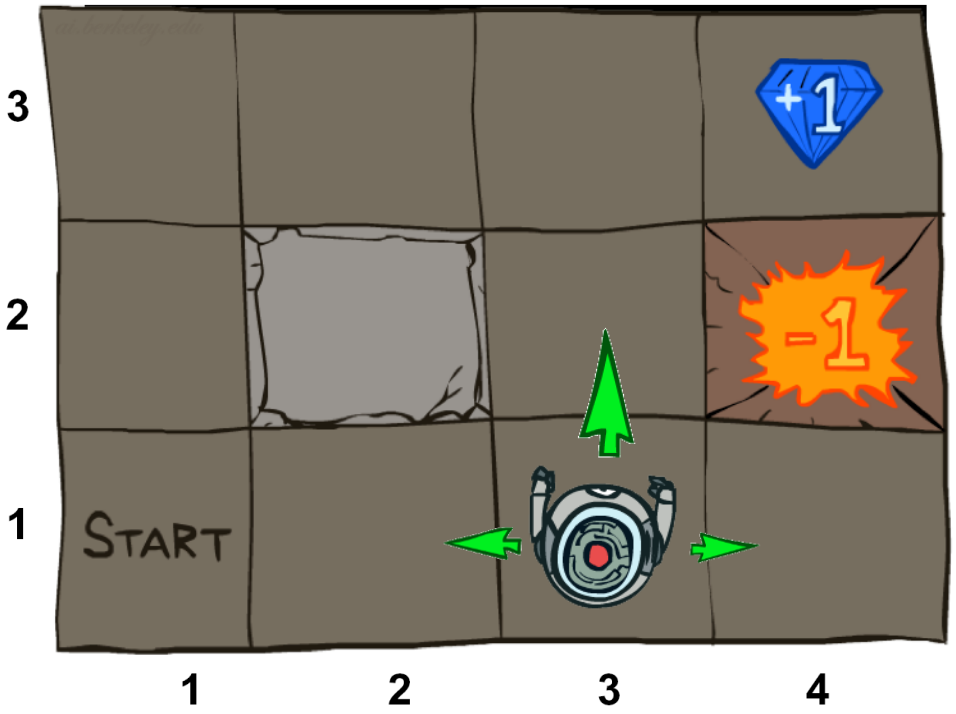


Stochastic Grid World



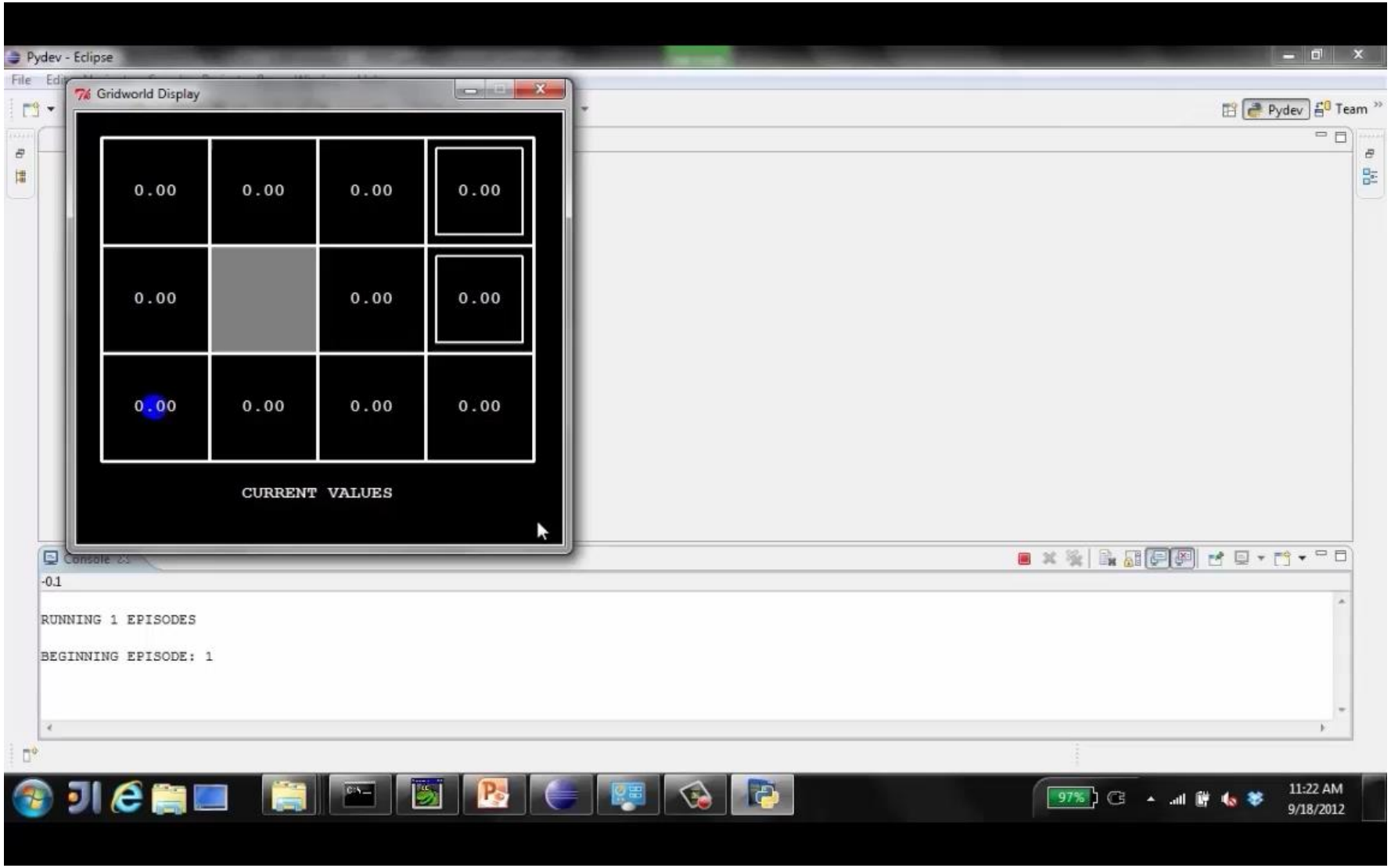
Markov Decision Processes

- An MDP is defined by:
  - A set of states $s \in S$
  - A set of actions $a \in A$
  - A transition function $T(s, a, s')$
    - 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) or R(s')
  - A start state
  - Maybe a terminal state



[Demo – gridworld manual intro (L8D1)]

Video of Demo Gridworld Manual Intro



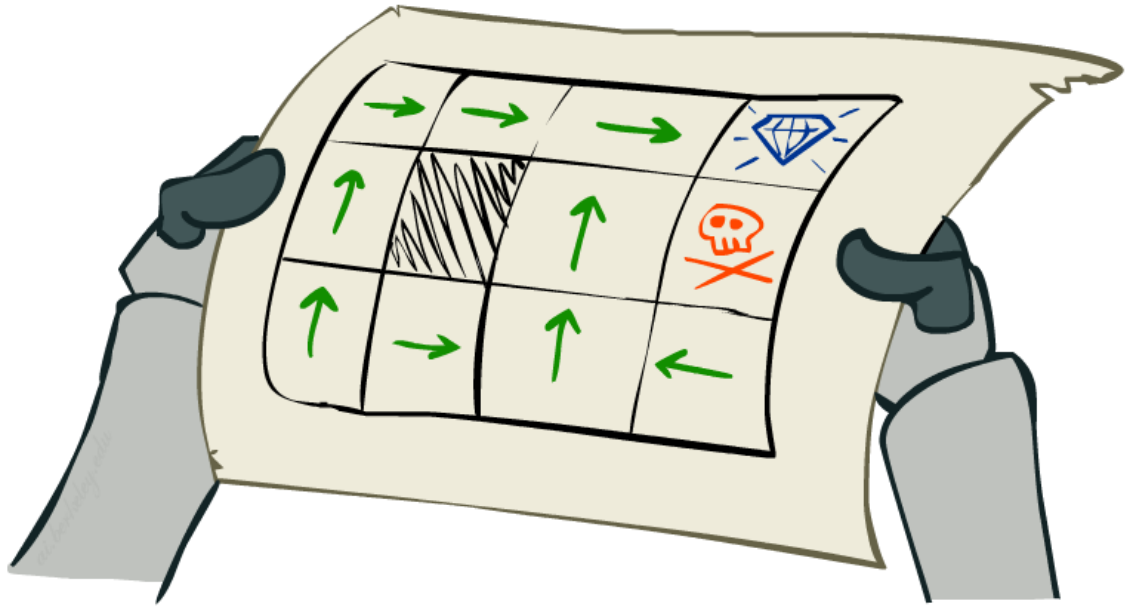
Policies

- 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

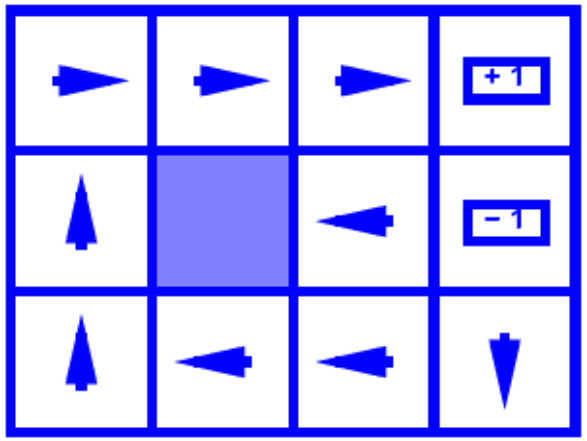
  policy $\pi^*: S \rightarrow A$

  - 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

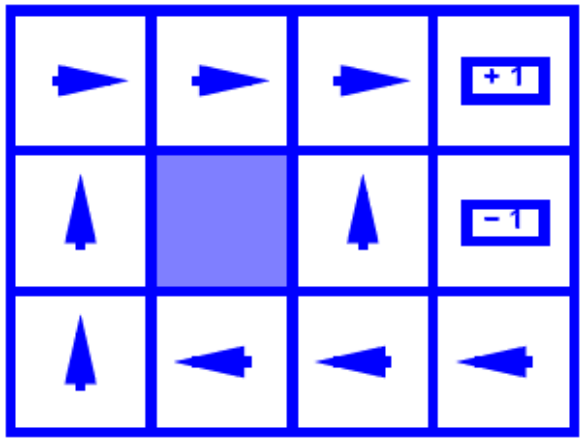


Optimal policy when R(s, a, s') = -0.03
for all non-terminals s

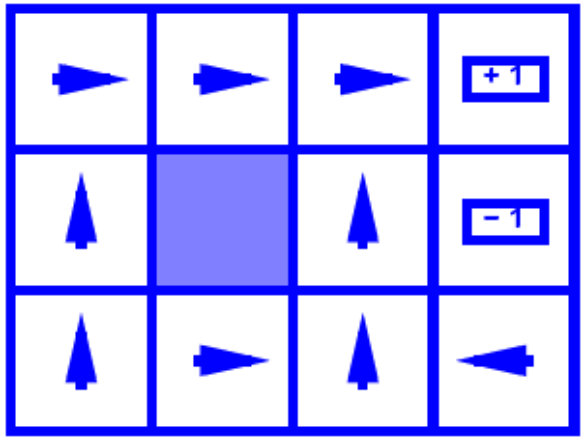
Optimal Policies



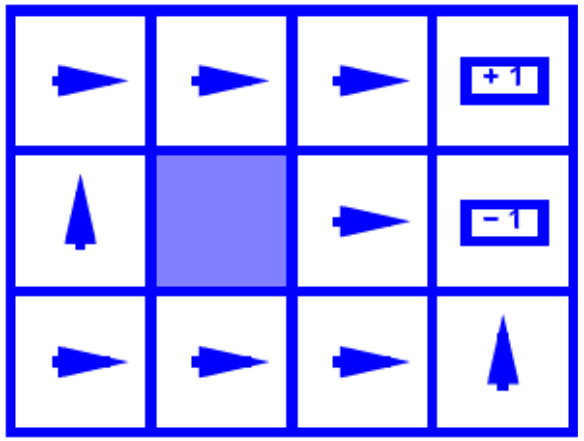
R(s) = -0.01



R(s) = -0.03

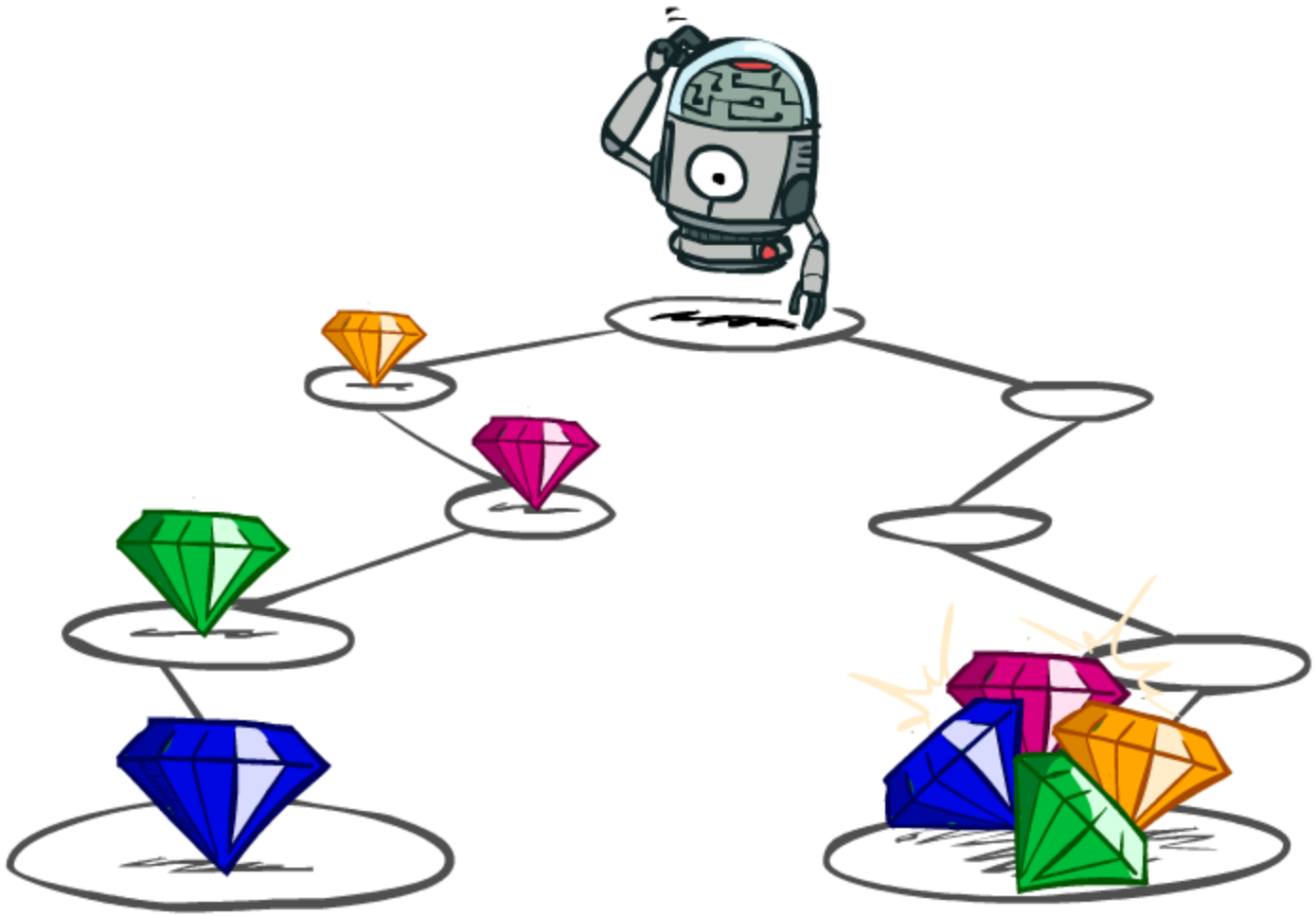


R(s) = -0.4



R(s) = -2.0

Utilities of Sequences



Utilities of Sequences

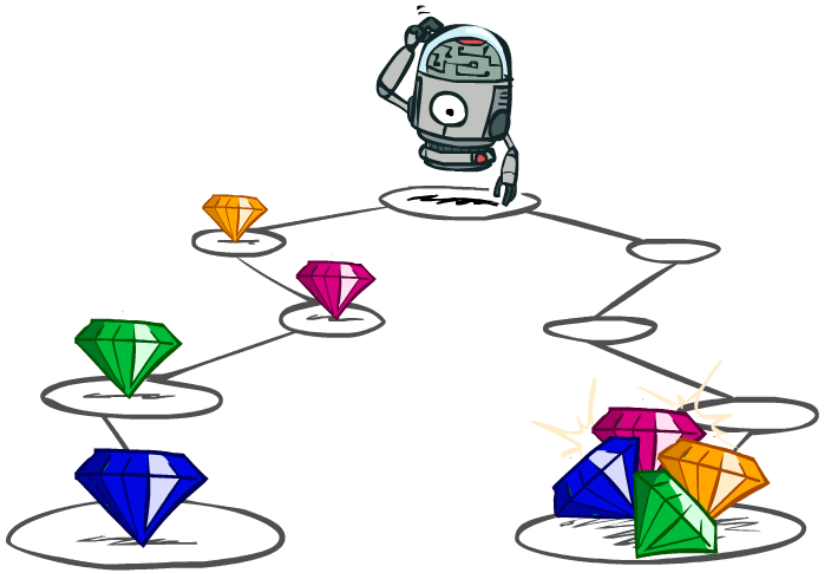
- What preferences should an agent have over reward sequences?

- More or less?

  [1, 2, 2] or [2, 3, 4]

- Now or later?

  [0, 0, 1] or [1, 0, 0]



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

1

Worth Now

γ

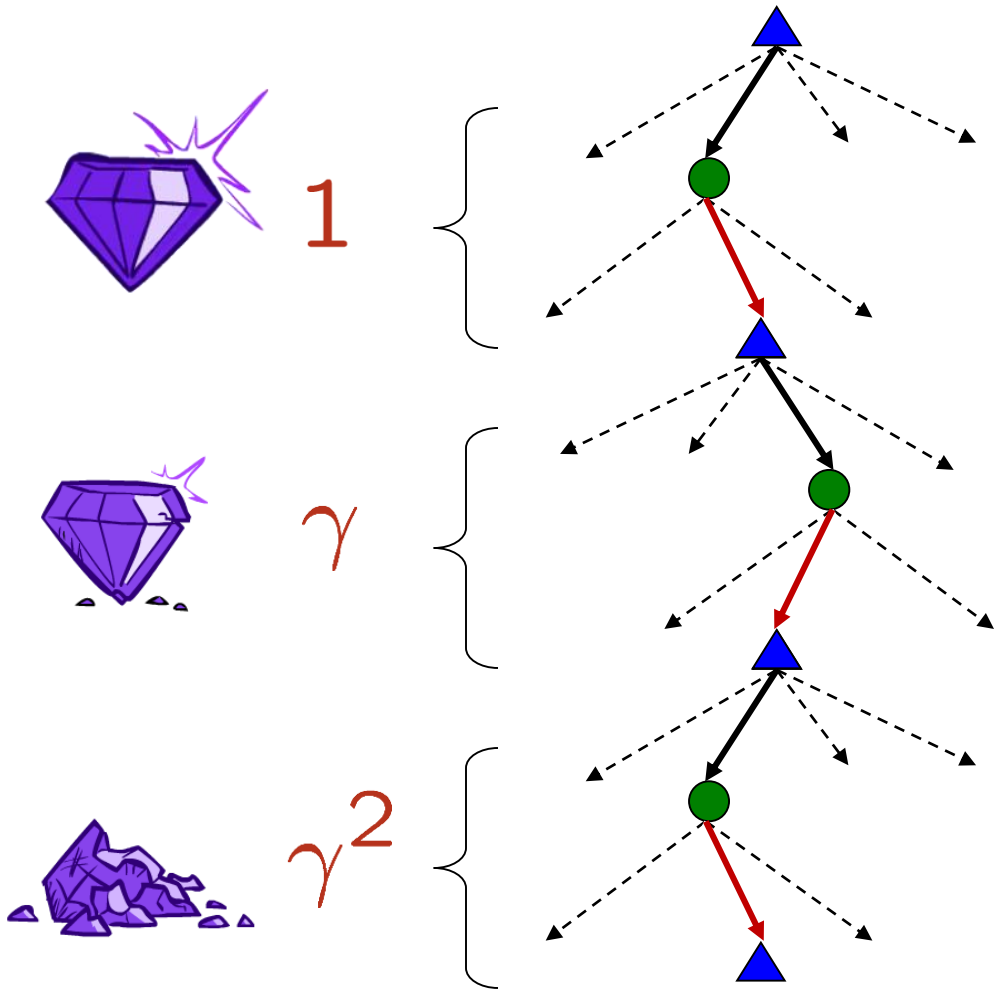
Worth Next Step

γ^2

Worth In Two Steps

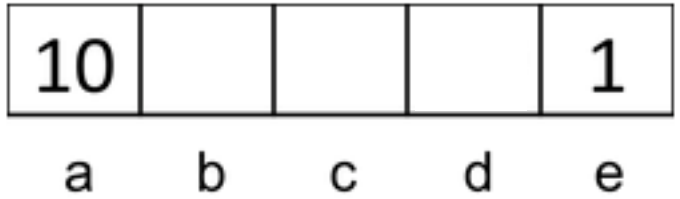
Discounting

- How to discount?
  - Each time we descend a level, we multiply in the discount once
- Why discount?
  - Think of it as a gamma chance of ending the process at every step
  - 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])



Quiz: Discounting

- Given:



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

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

10	<-	<-	<-	1

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

10	<-	<-	->	1

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

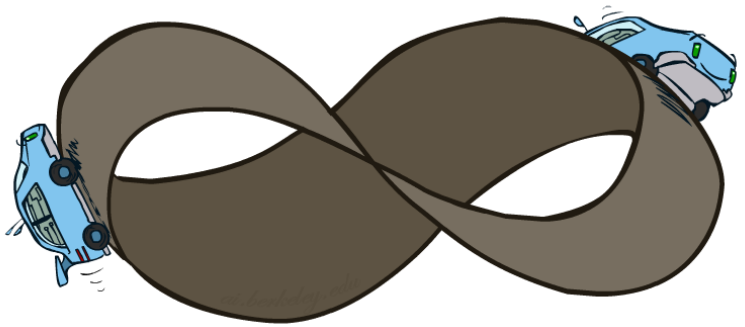
$1\gamma = 10\,\gamma^3$

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 $0 < \gamma < 1$

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

    - 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)



Optimal Quantities

- The value (utility) of a state s:

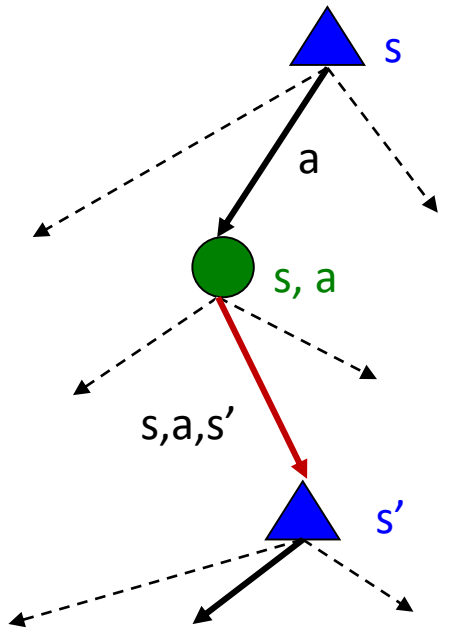
  $V^*(s)$ = expected utility starting in s and acting optimally

- 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

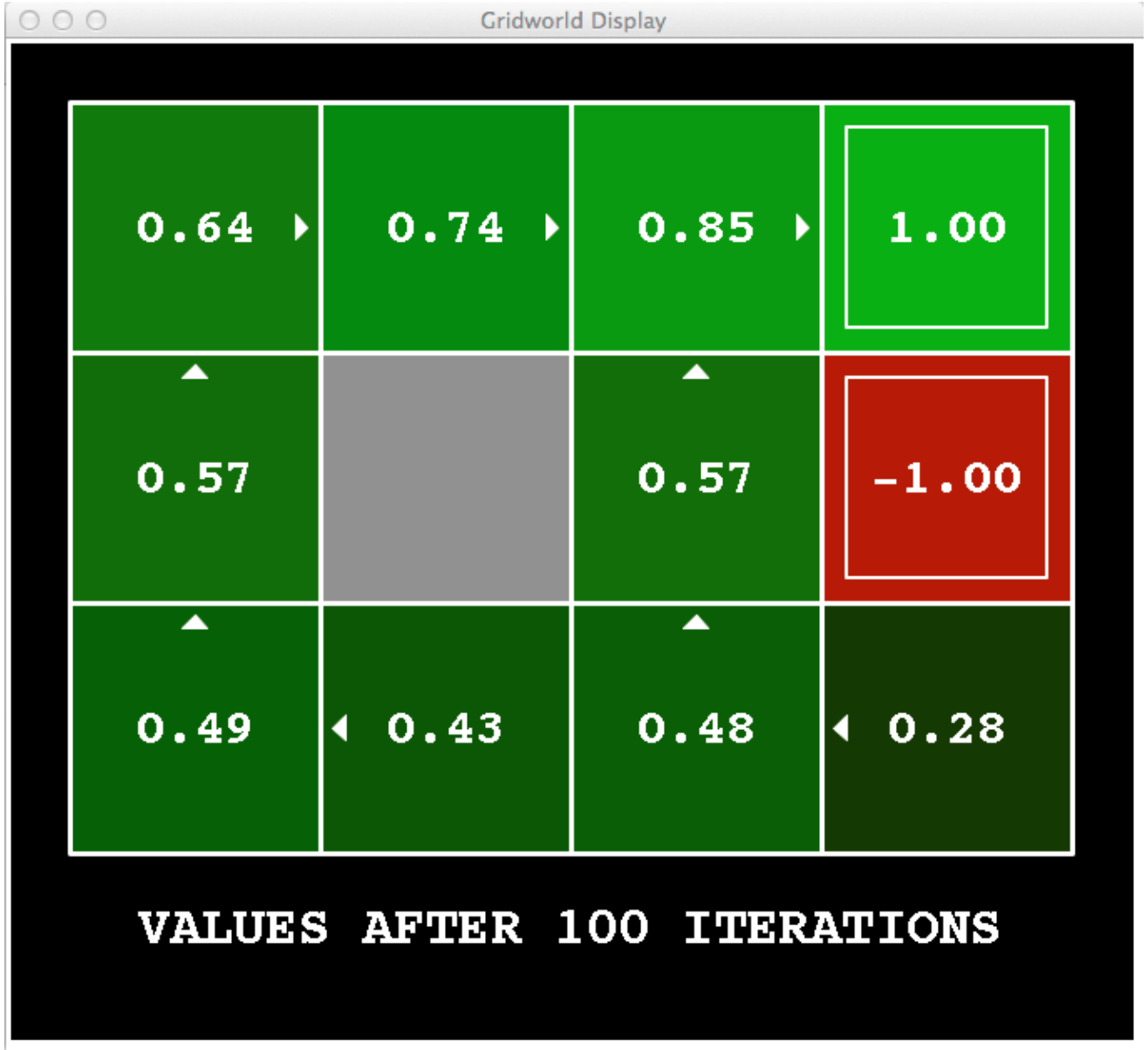


s is a *state*

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

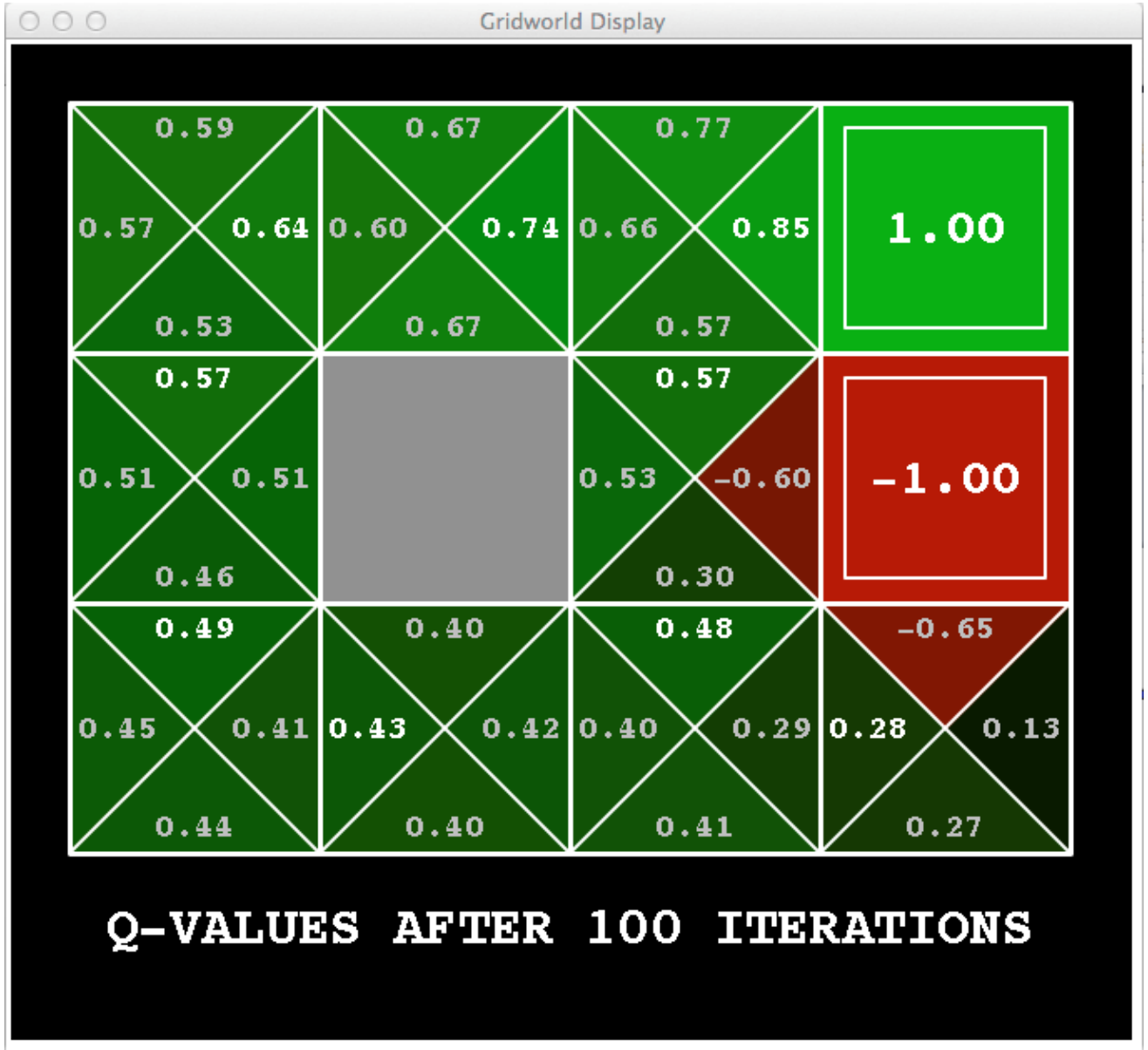
(s,a,s') is a *transition*

Snapshot of Demo – Gridworld V Values



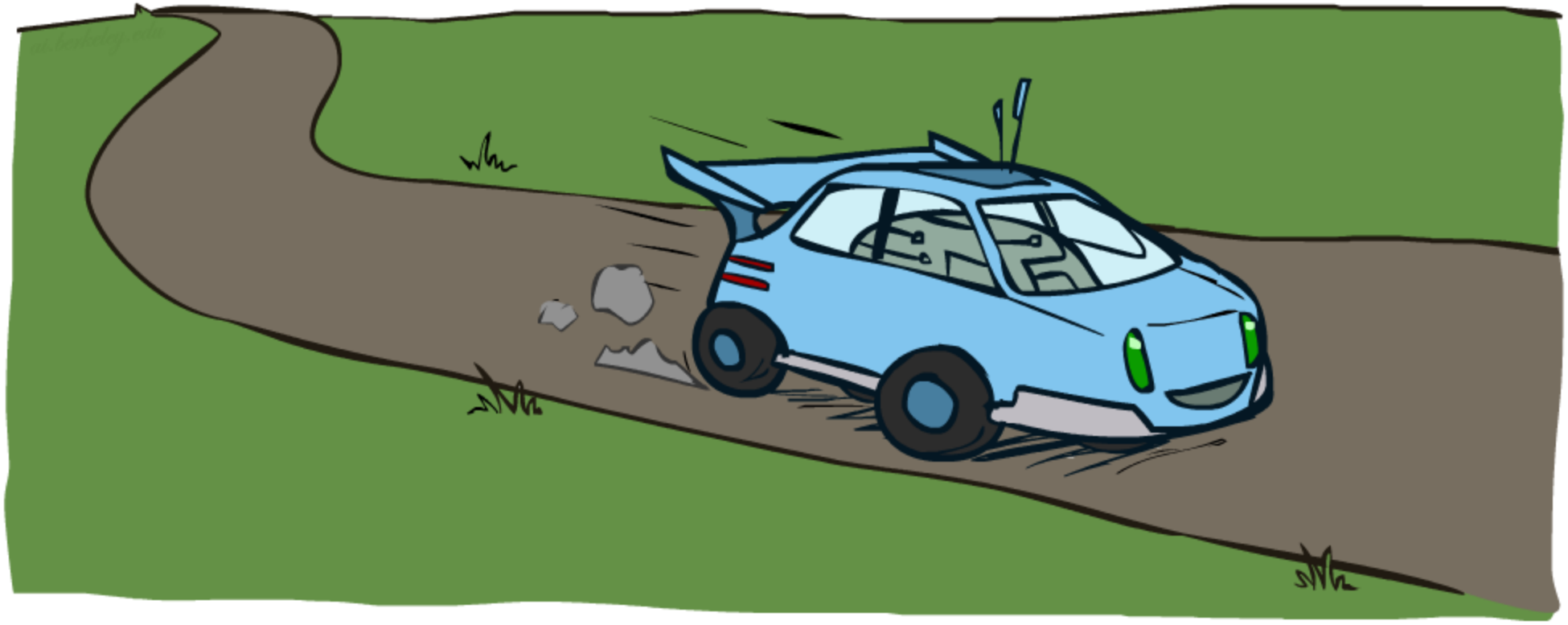
Noise = 0.2
Discount = 0.9
Living reward = 0

Snapshot of Demo – Gridworld Q Values



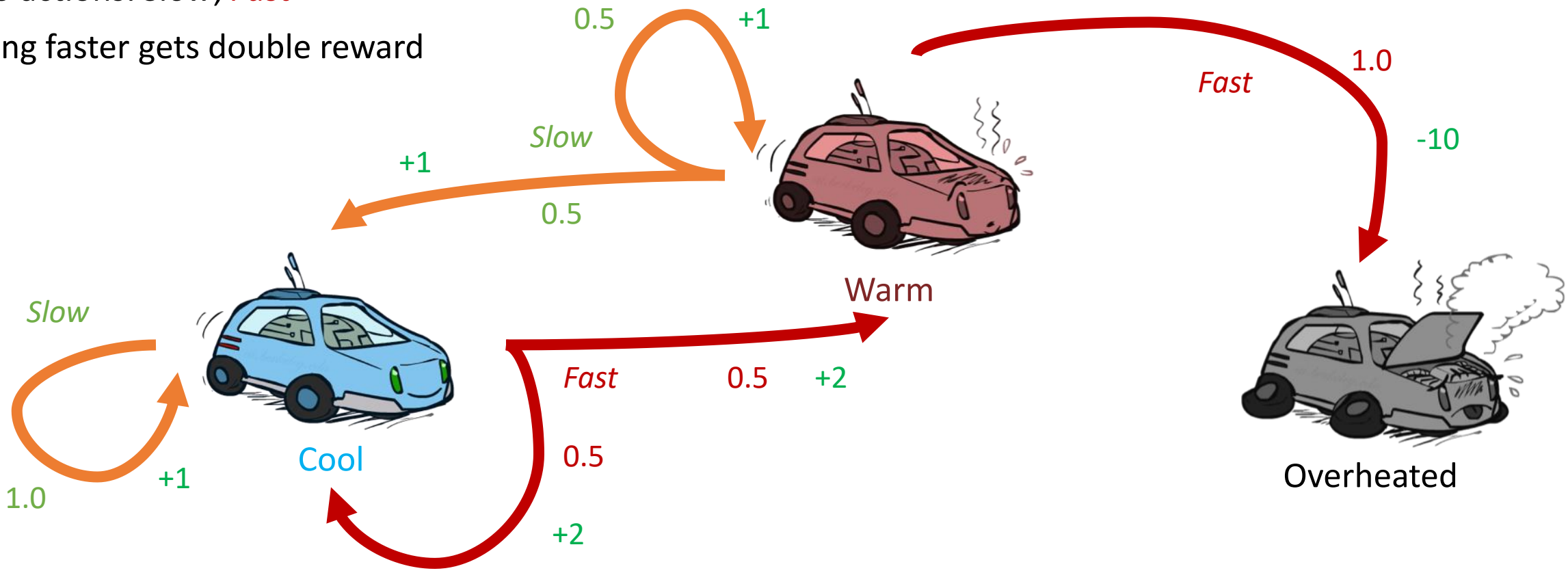
Noise = 0.2
Discount = 0.9
Living reward = 0

Example: Racing

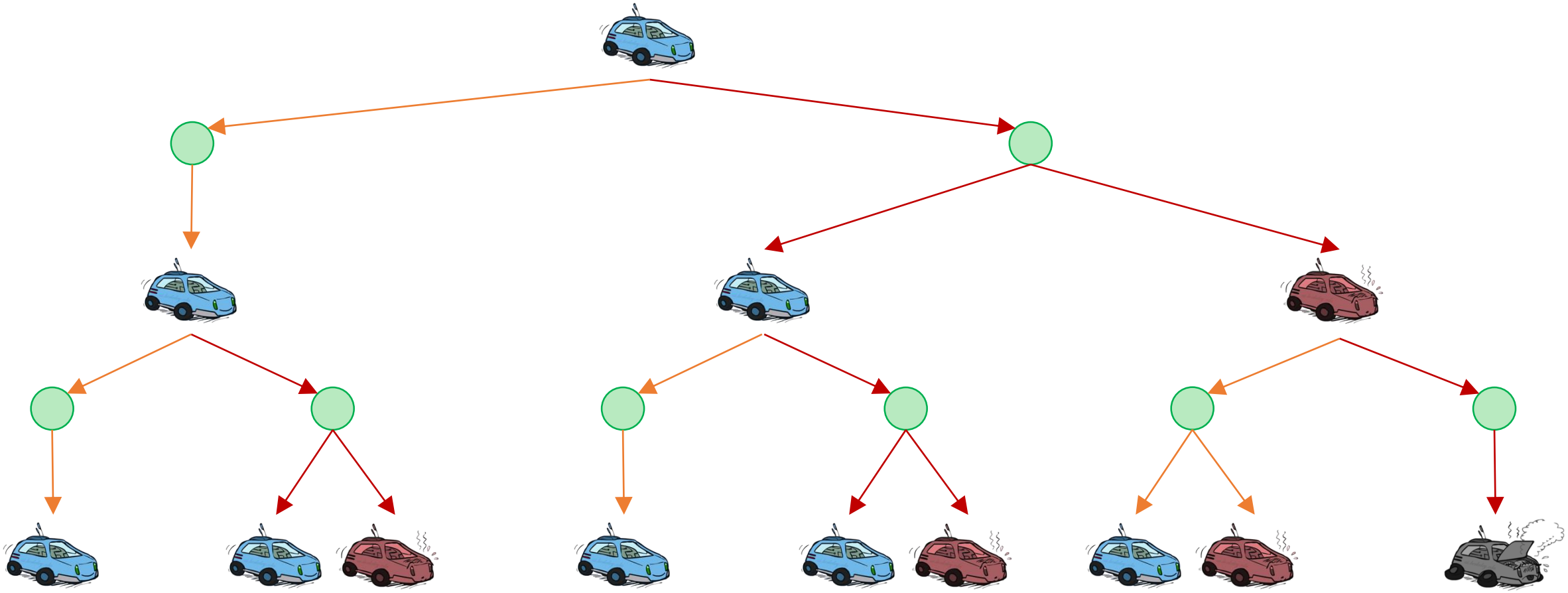


Example: Racing

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

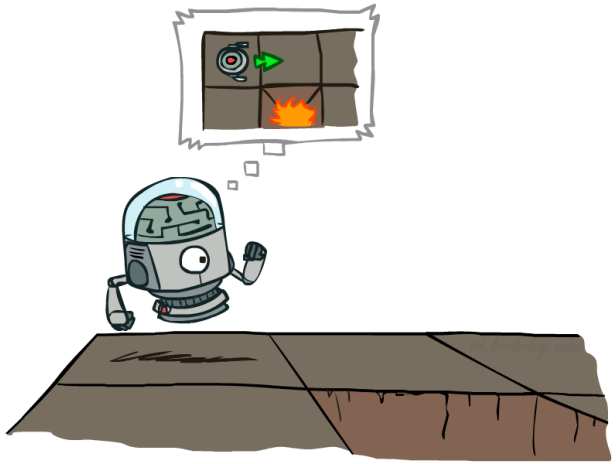


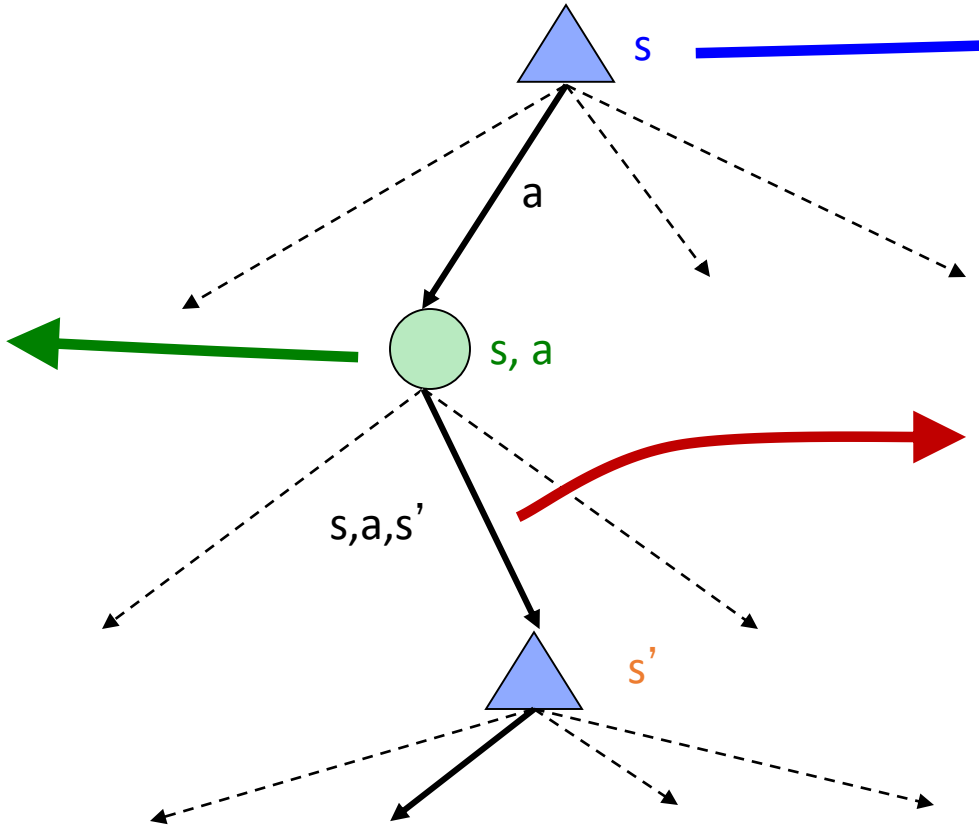
Racing Search Tree



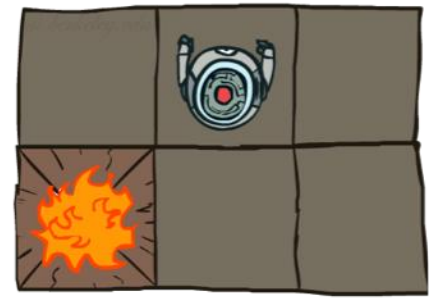
MDP Search Trees

- Each MDP state projects an expectimax-like search tree





s is a *state*



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

$T(s,a,s') = P(s'|s,a)$

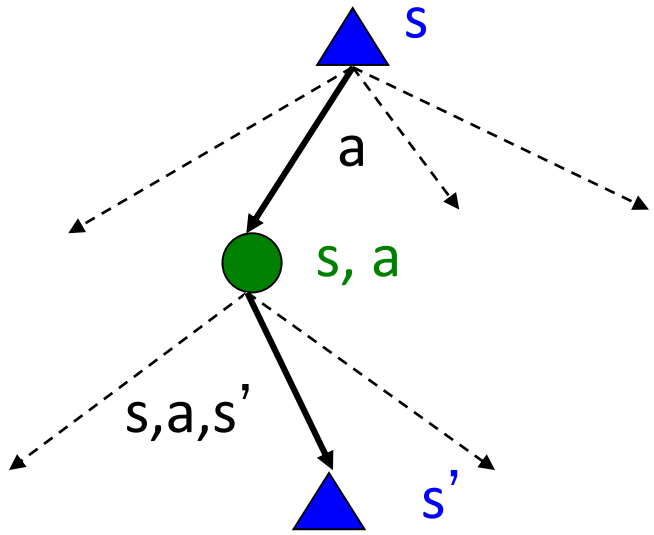
$R(s,a,s')$

[Expectimax Search Reference]

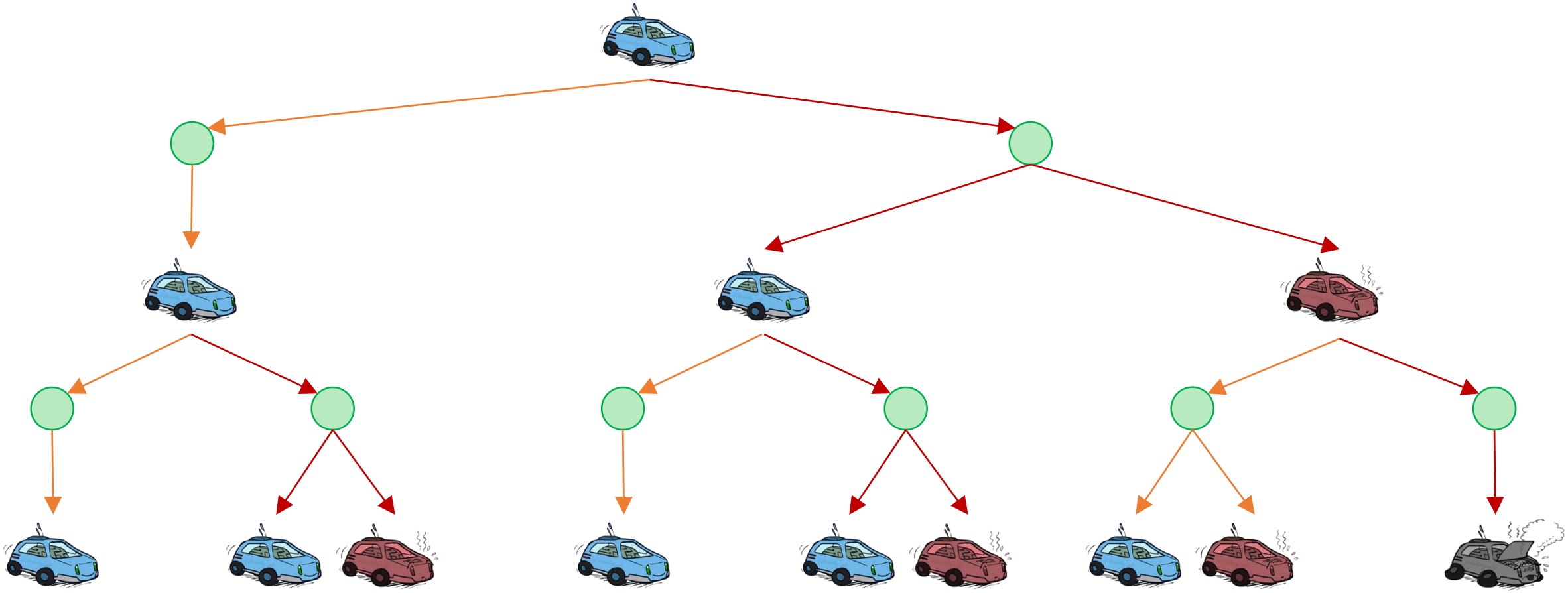
Recap: Defining MDPs

- 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)

- MDP quantities so far:
  - Policy = Choice of action for each state
  - Utility = sum of (discounted) rewards



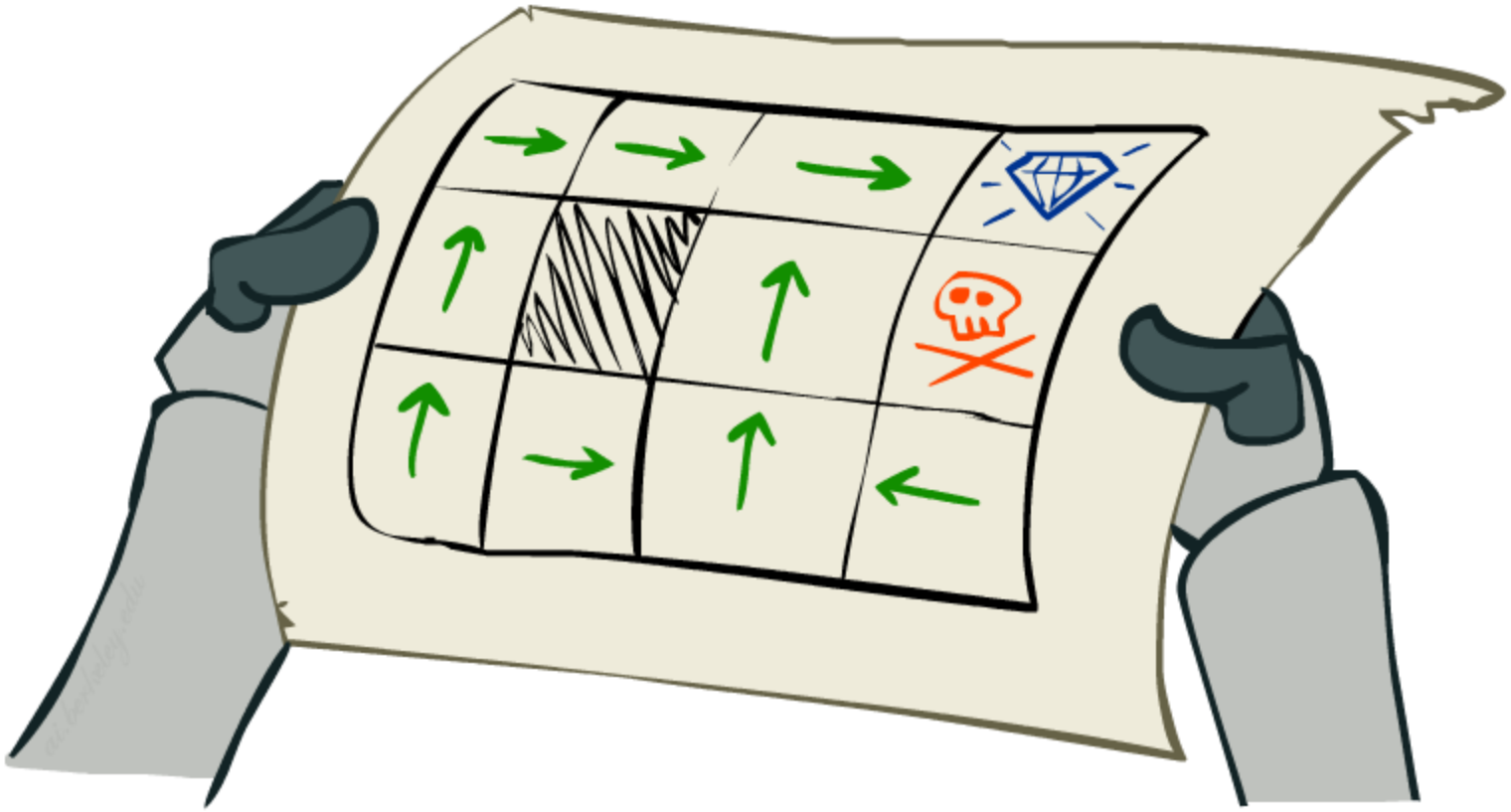
Racing Search Tree



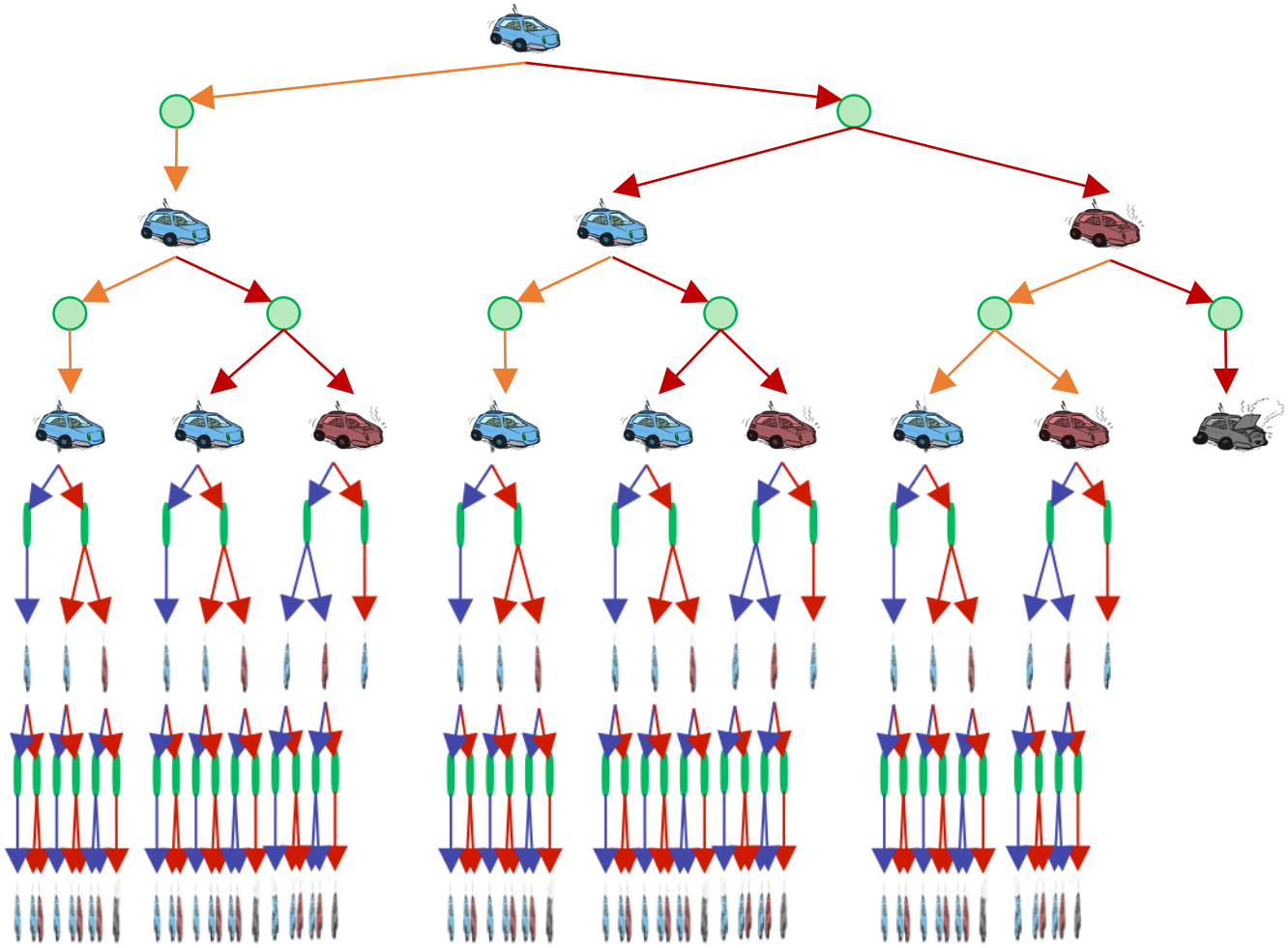
Outline

- Utilities to MDPs
- Value Iteration
- Policy Extraction
- Policy Iteration
- Convergence Analysis

Solving MDPs

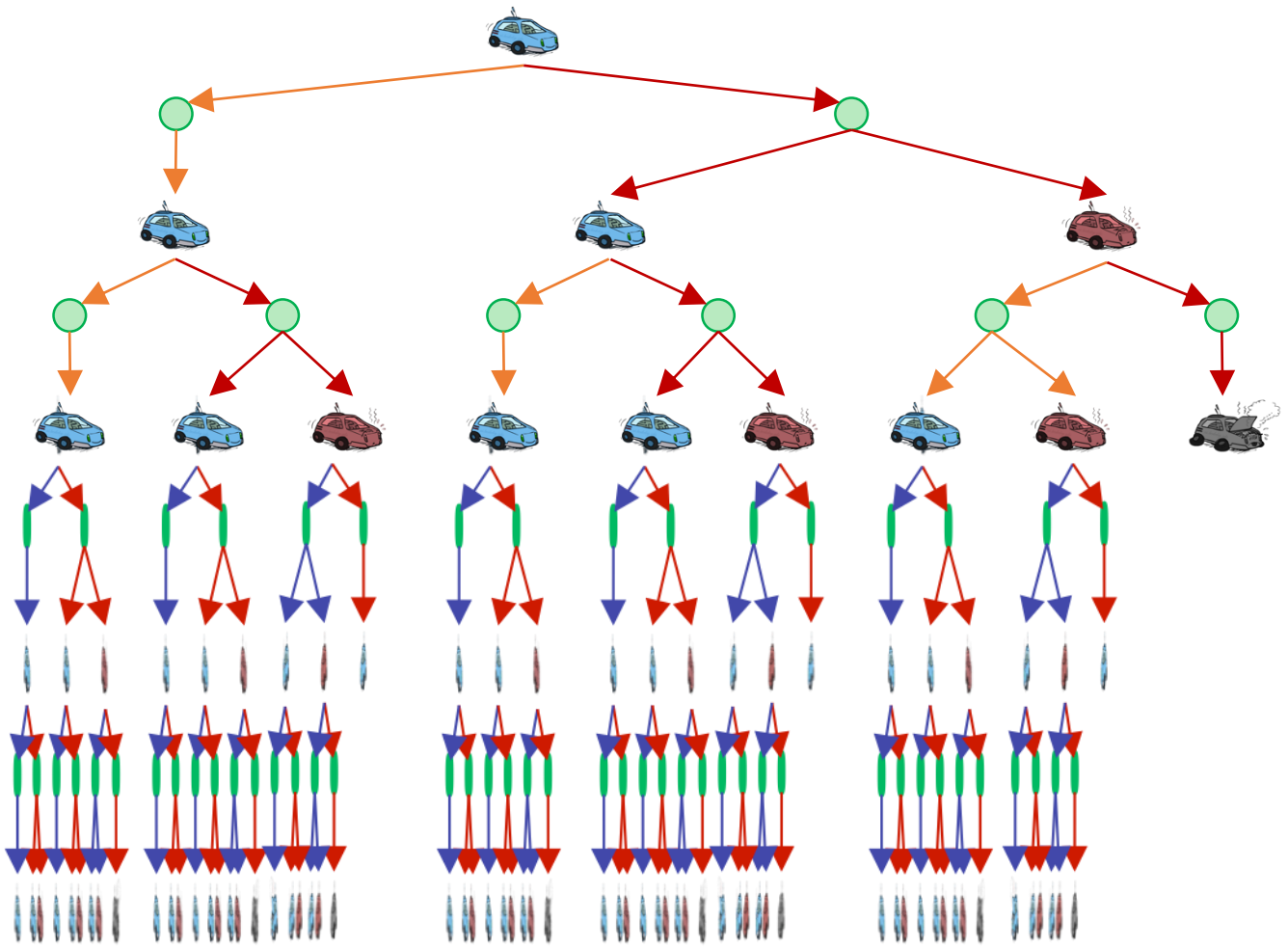


Racing Search Tree



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 $\gamma < 1$

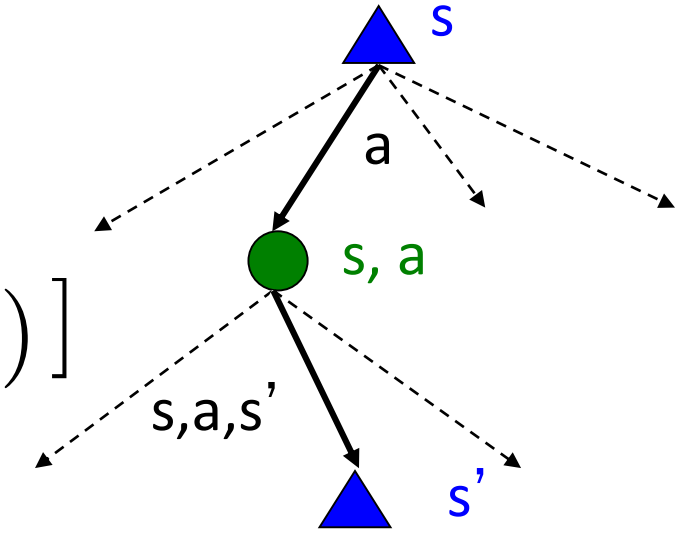


Values of States

Recursive definition of value:

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

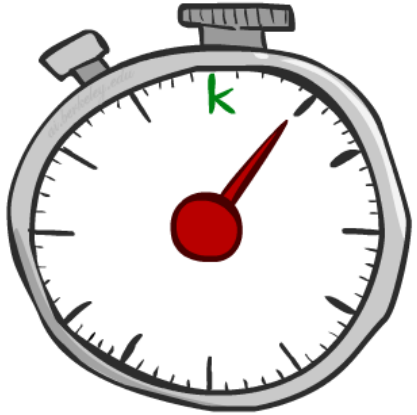
$$Q^*(s,a) = \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V^*(s')\right]$$

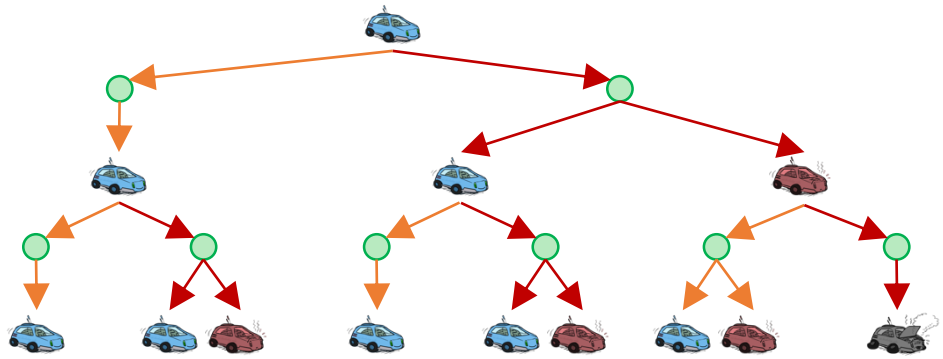


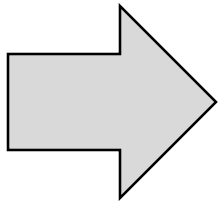
$$V^*(s) = \max_a \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V^*(s')\right]$$

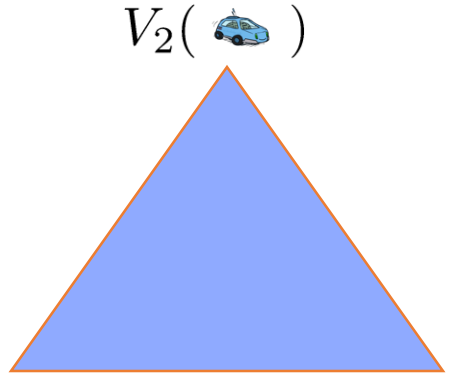
Time-Limited Values

- Key idea: time-limited values
- Define $V_k(s)$ to be the optimal value of s if the game ends in k more time steps
  - Equivalently, it's what a depth-k expectimax would give from s



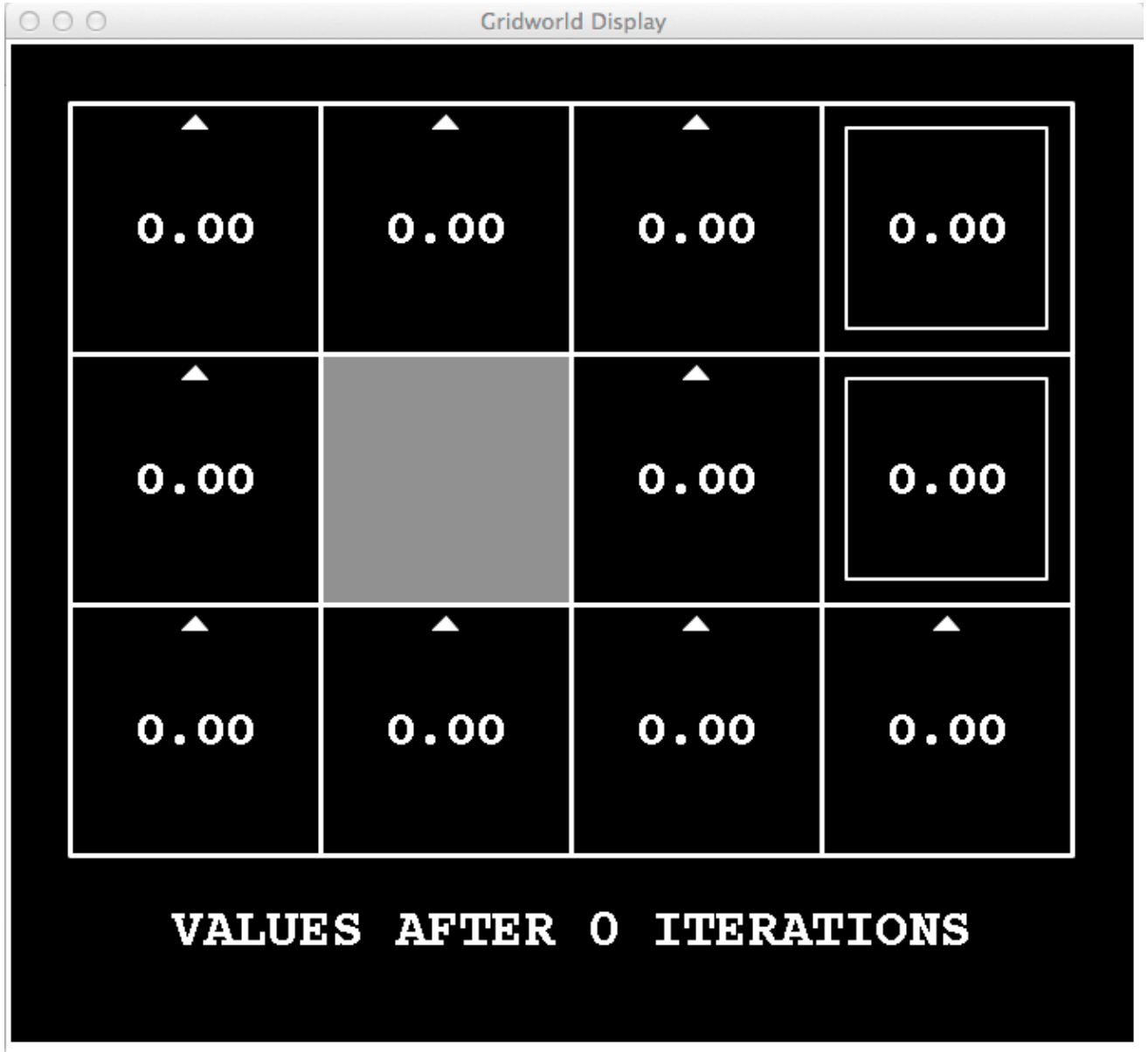






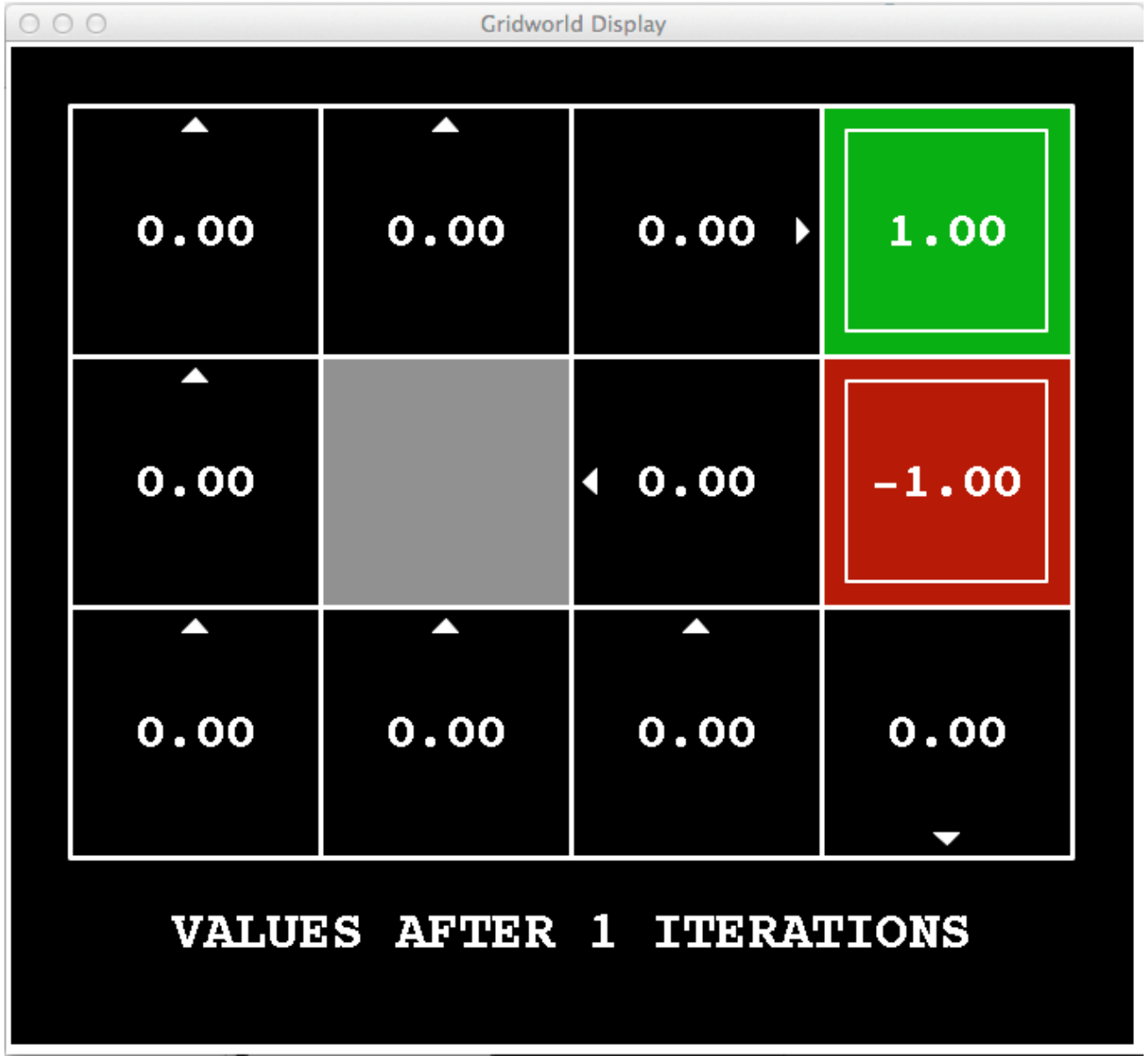
[Demo – time-limited values (L8D6)]

k=0



Noise = 0.2
Discount = 0.9
Living reward = 0

k=1



Noise = 0.2
Discount = 0.9
Living reward = 0

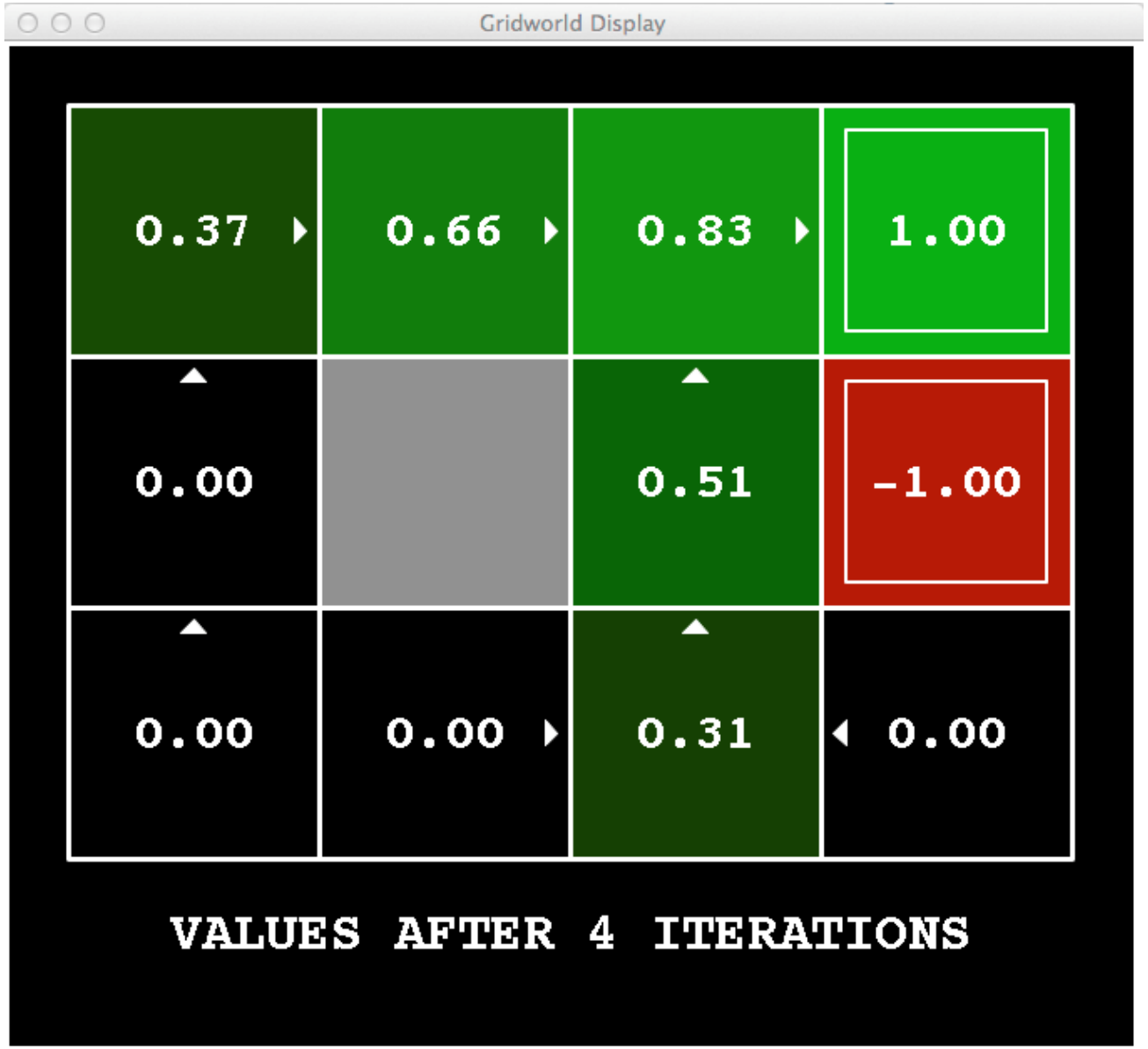
k=2

Noise = 0.2
Discount = 0.9
Living reward = 0

k=3

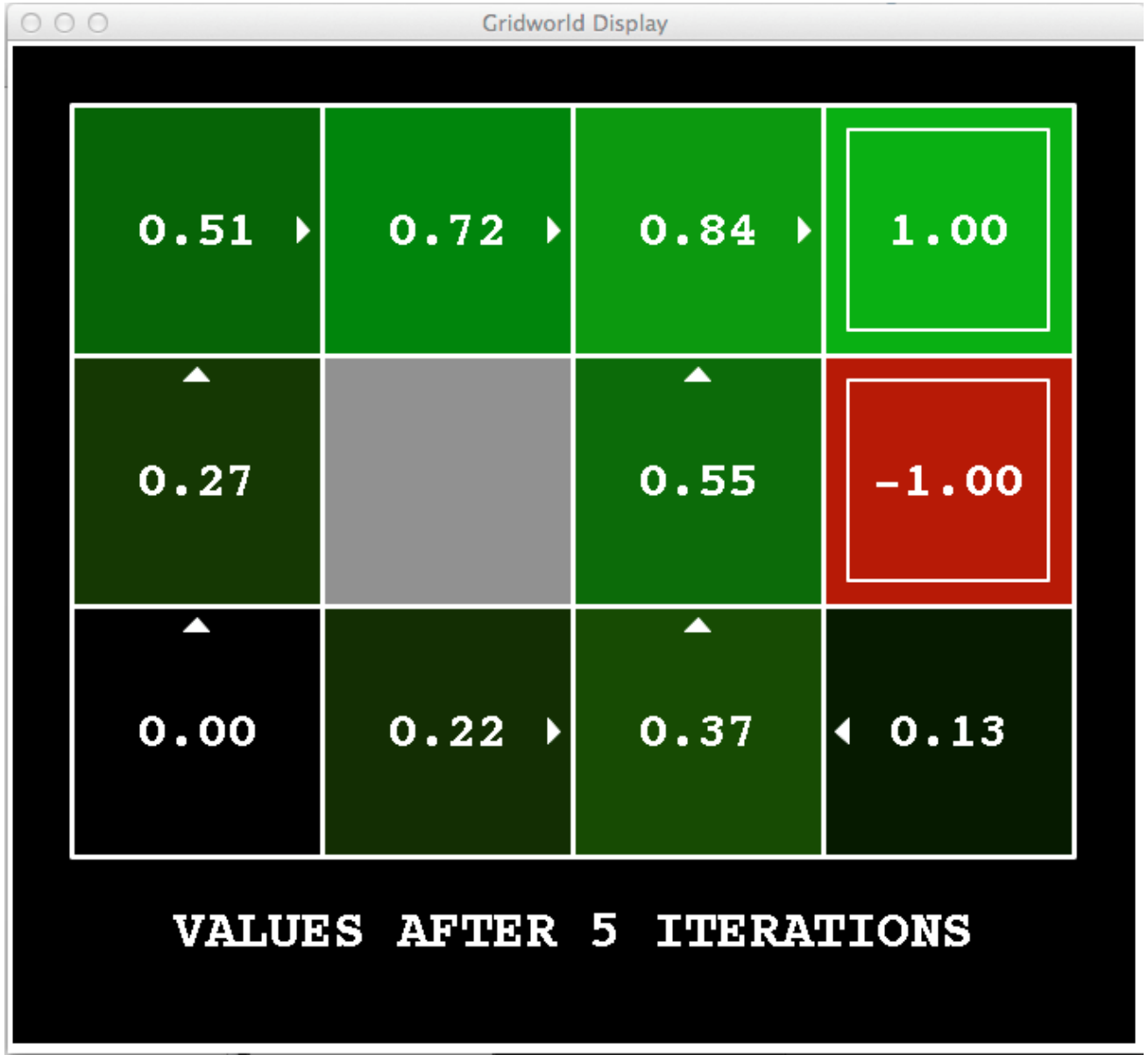
Noise = 0.2
Discount = 0.9
Living reward = 0

k=4



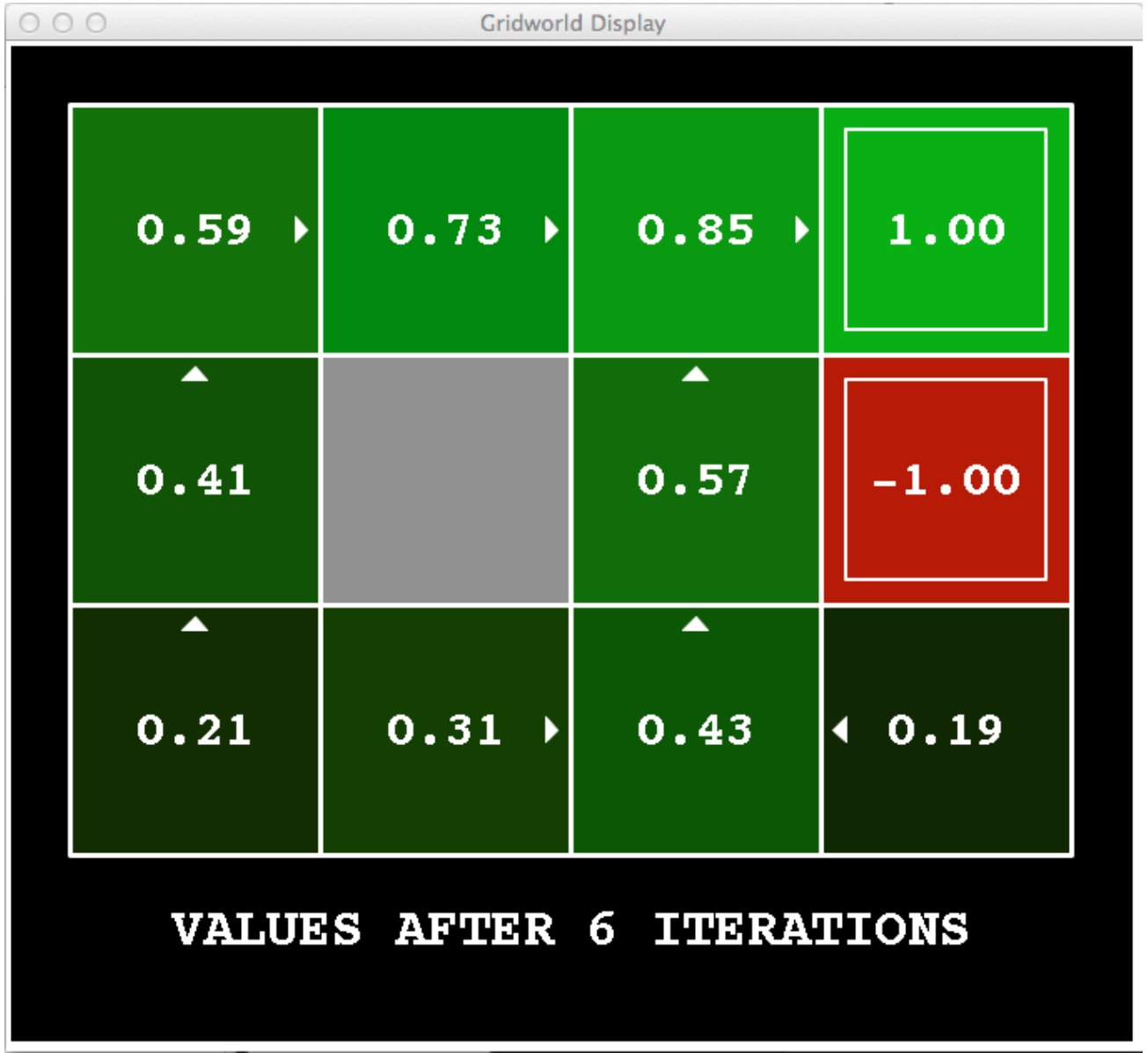
Noise = 0.2
Discount = 0.9
Living reward = 0

k=5



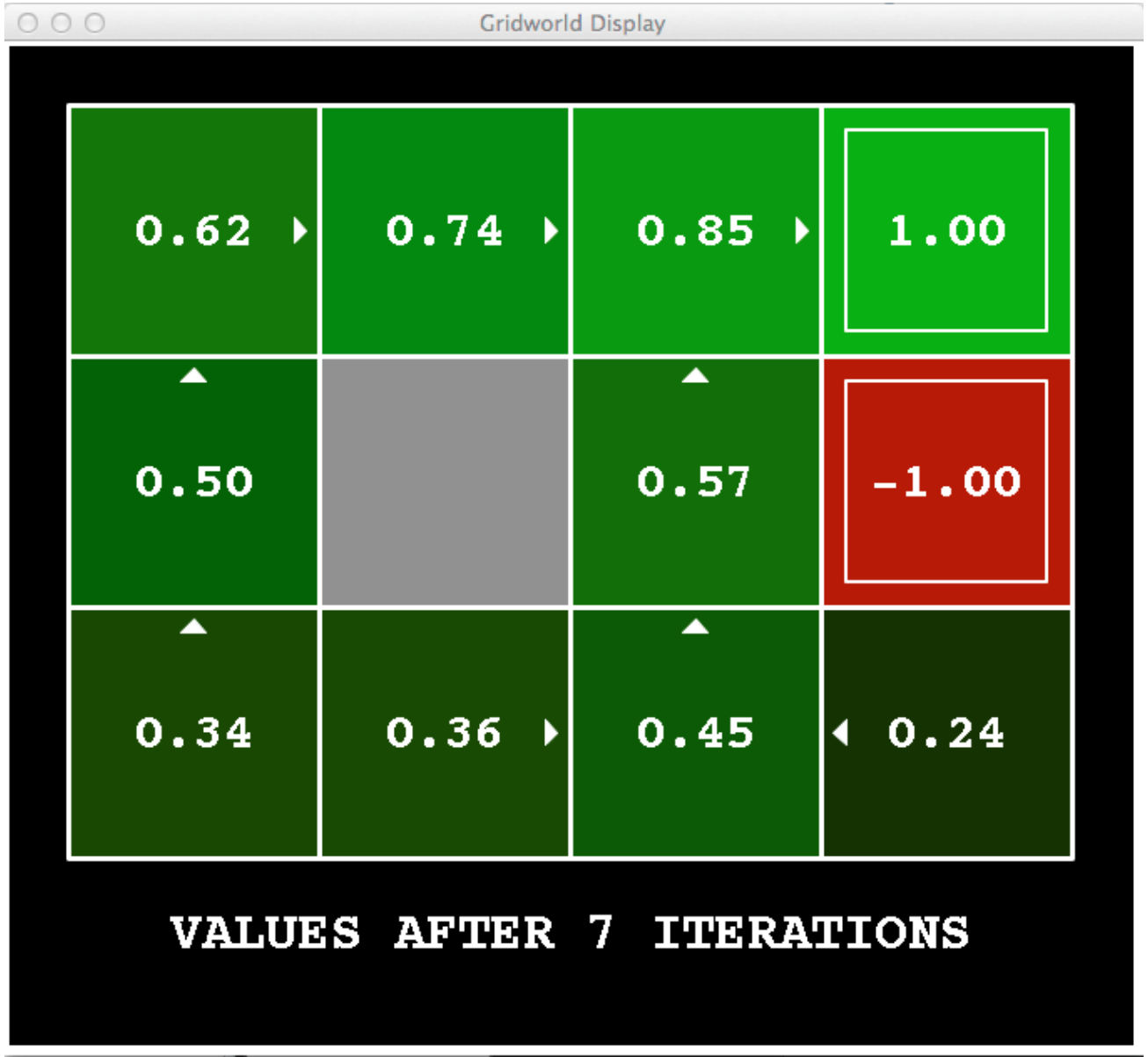
Noise = 0.2
Discount = 0.9
Living reward = 0

k=6



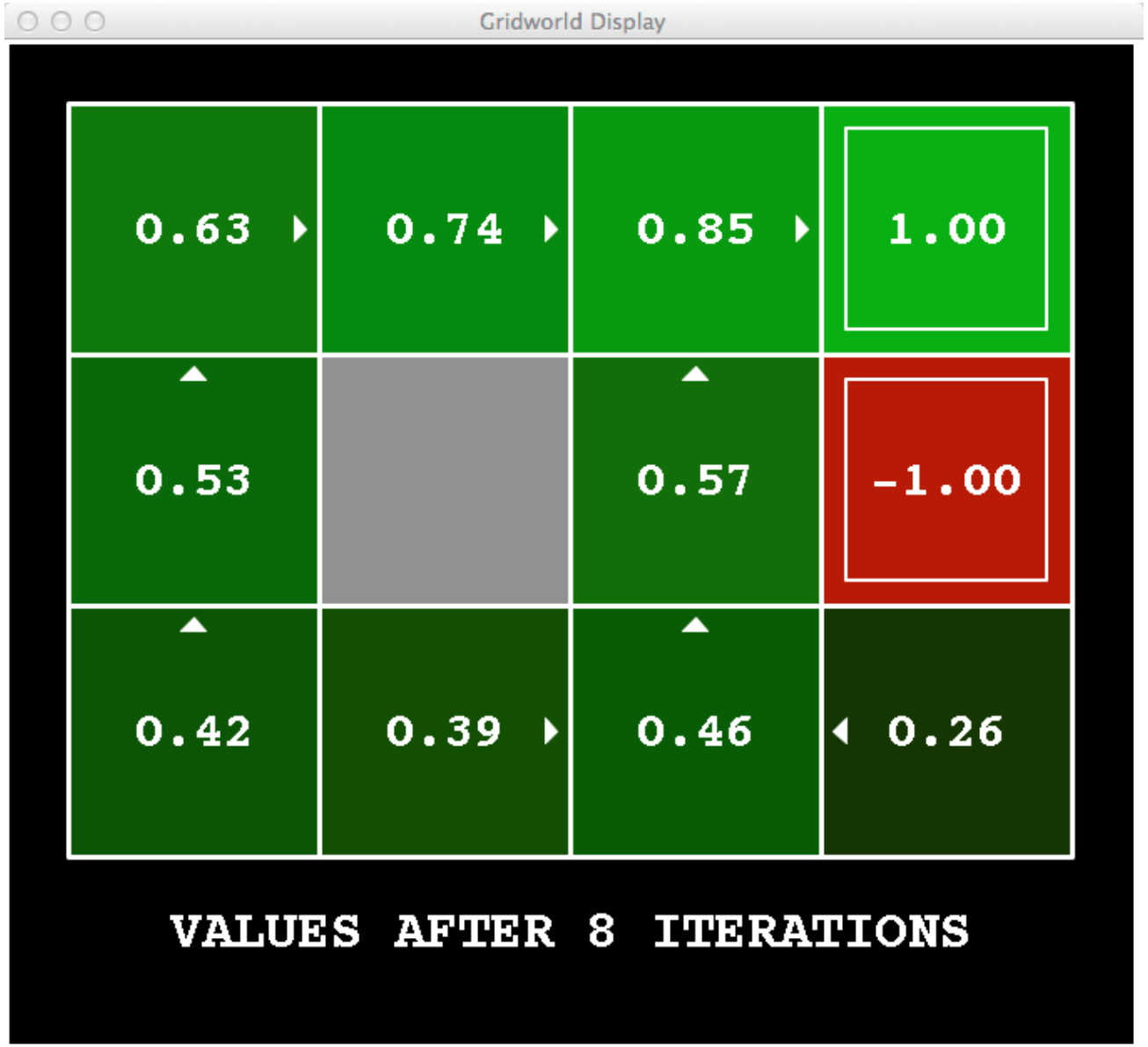
Noise = 0.2
Discount = 0.9
Living reward = 0

k=7



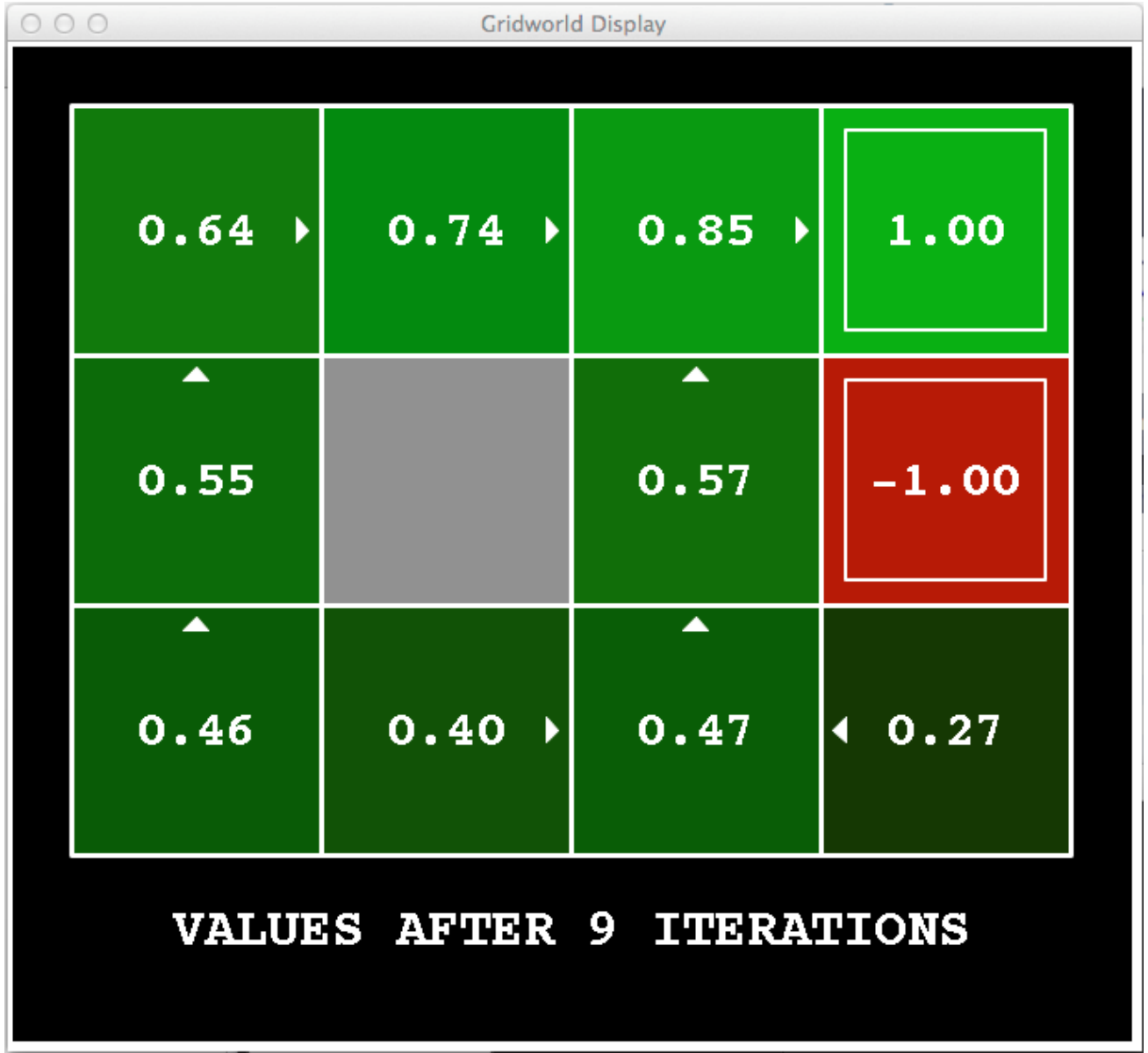
Noise = 0.2
Discount = 0.9
Living reward = 0

k=8



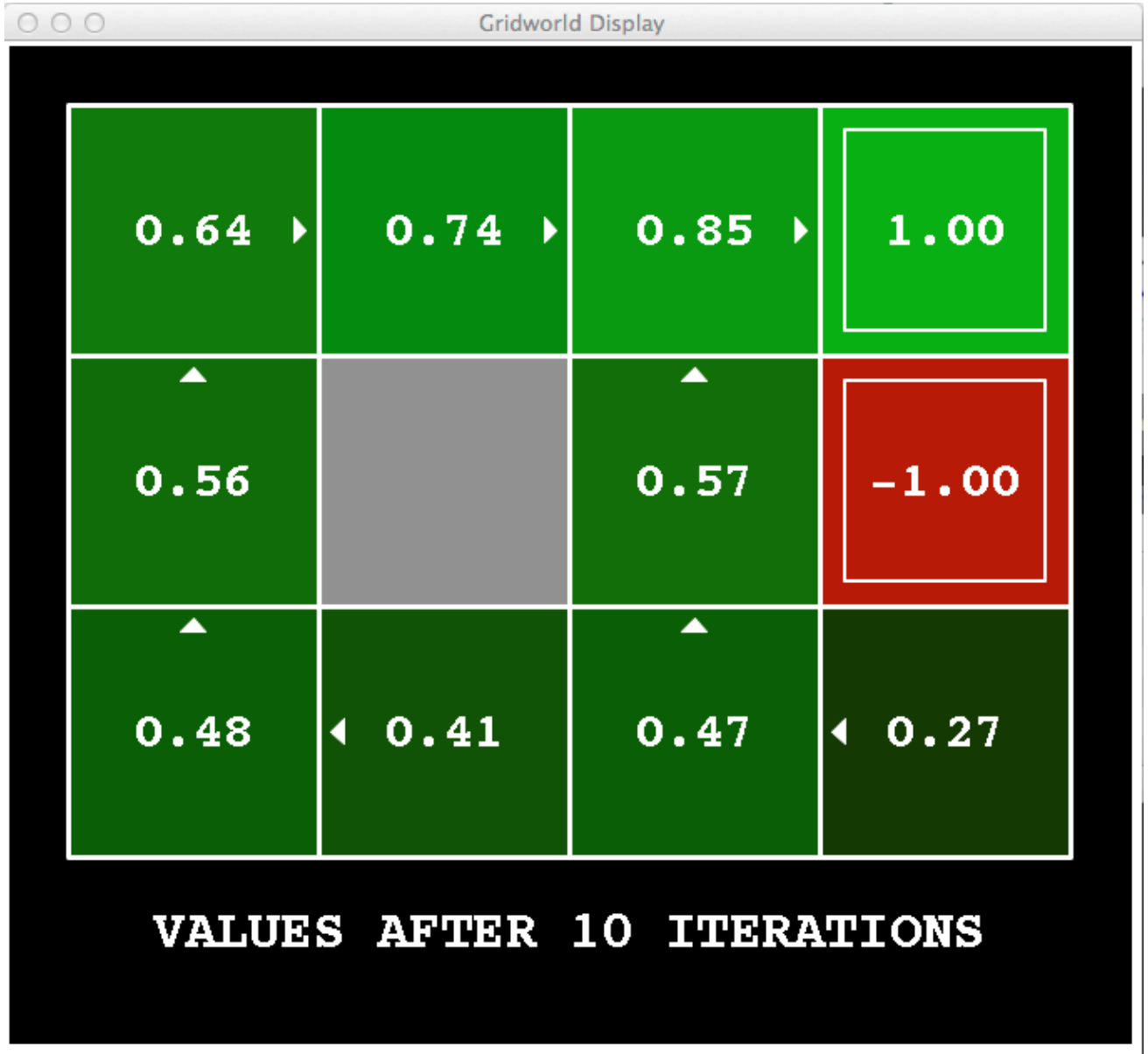
Noise = 0.2
Discount = 0.9
Living reward = 0

k=9



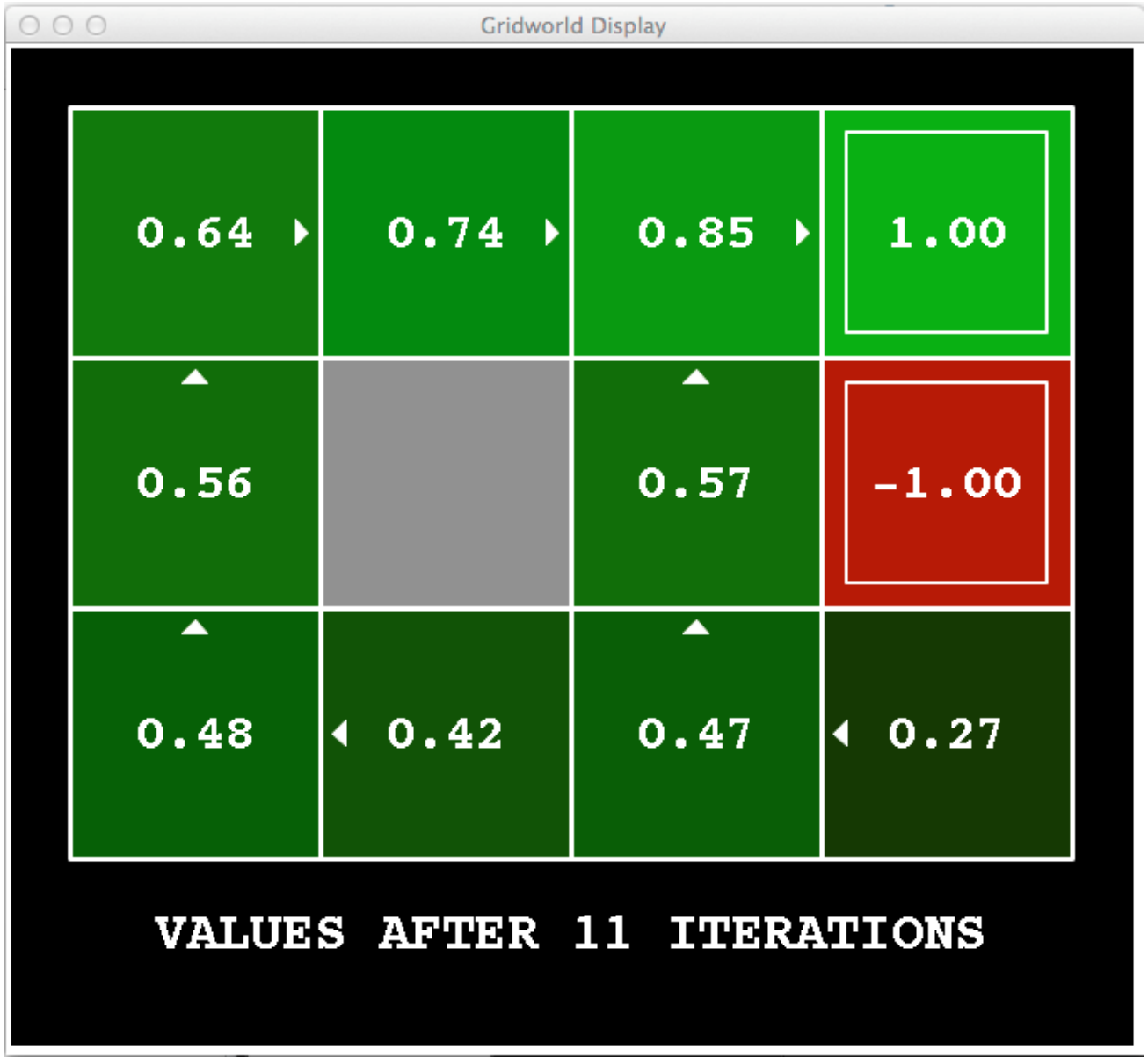
Noise = 0.2
Discount = 0.9
Living reward = 0

k=10



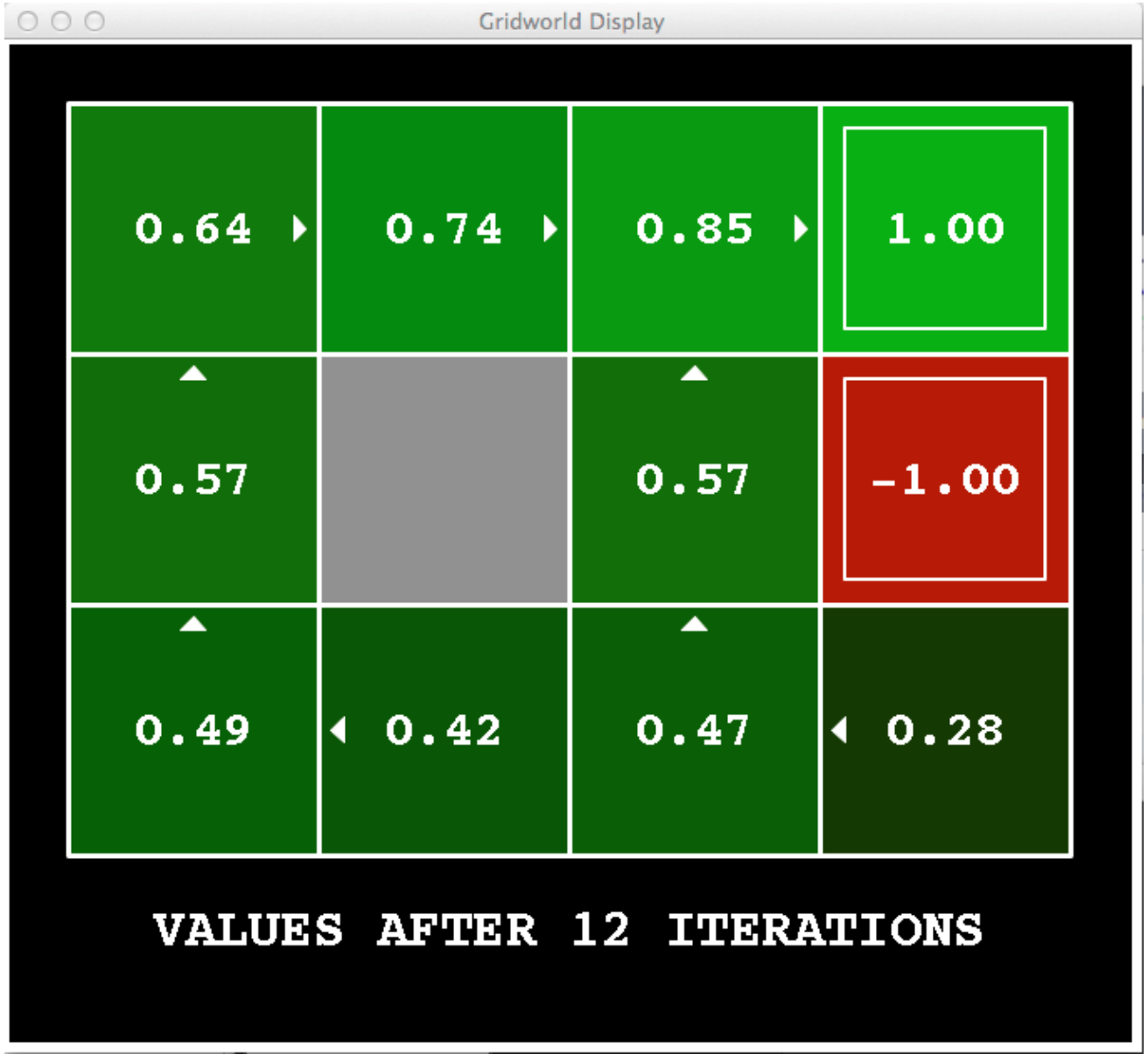
Noise = 0.2
Discount = 0.9
Living reward = 0

k=11



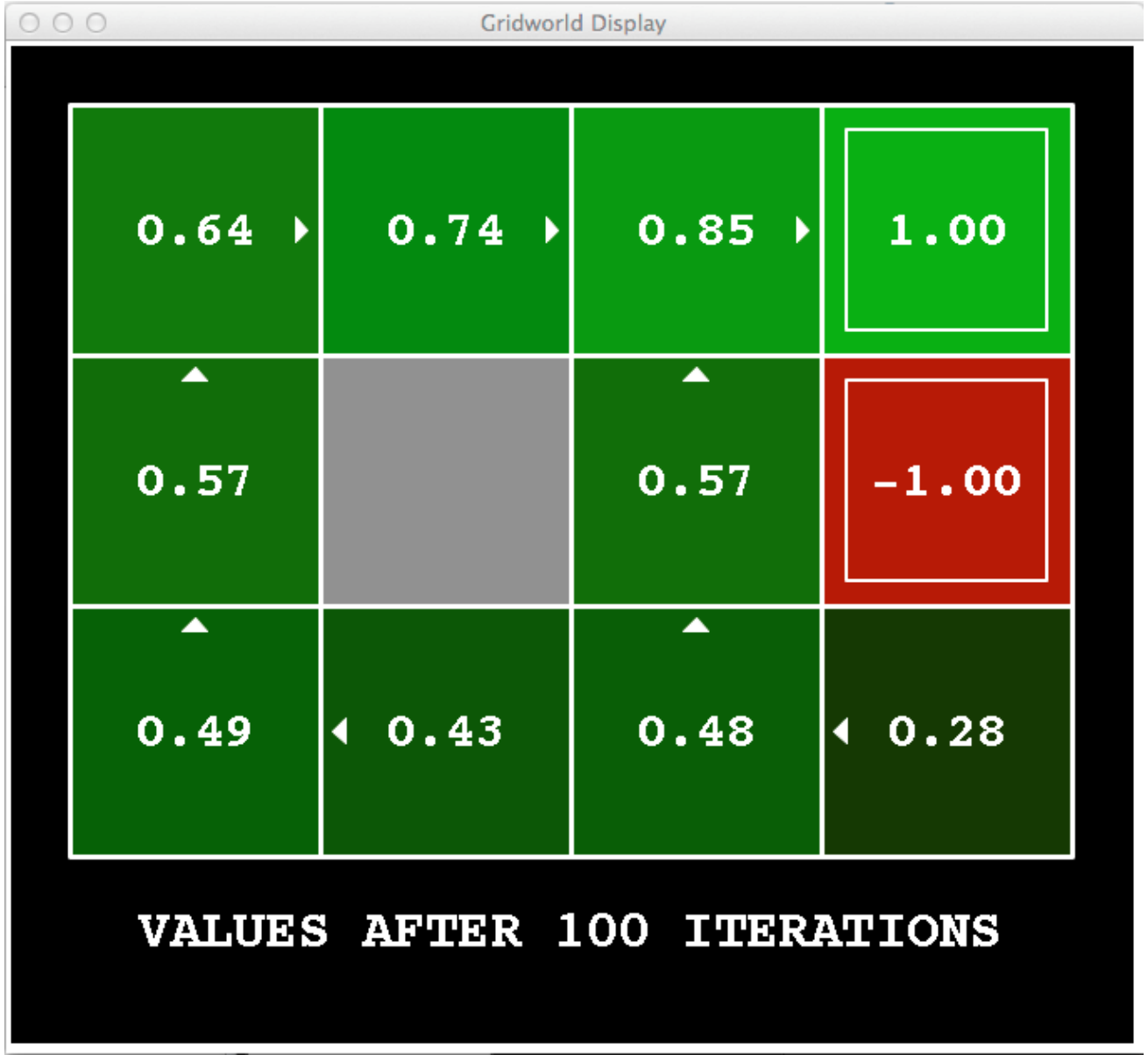
Noise = 0.2
Discount = 0.9
Living reward = 0

k=12



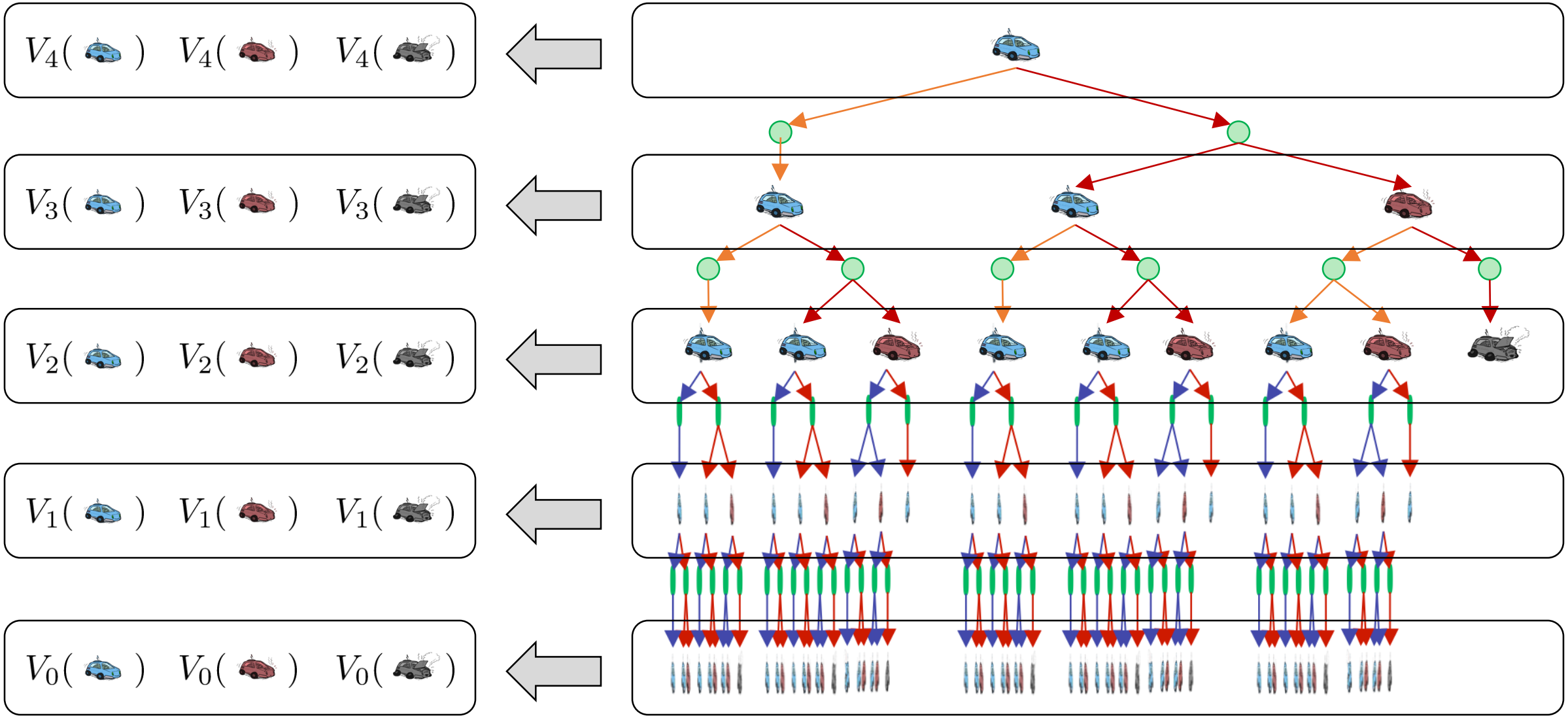
Noise = 0.2
Discount = 0.9
Living reward = 0

k=100

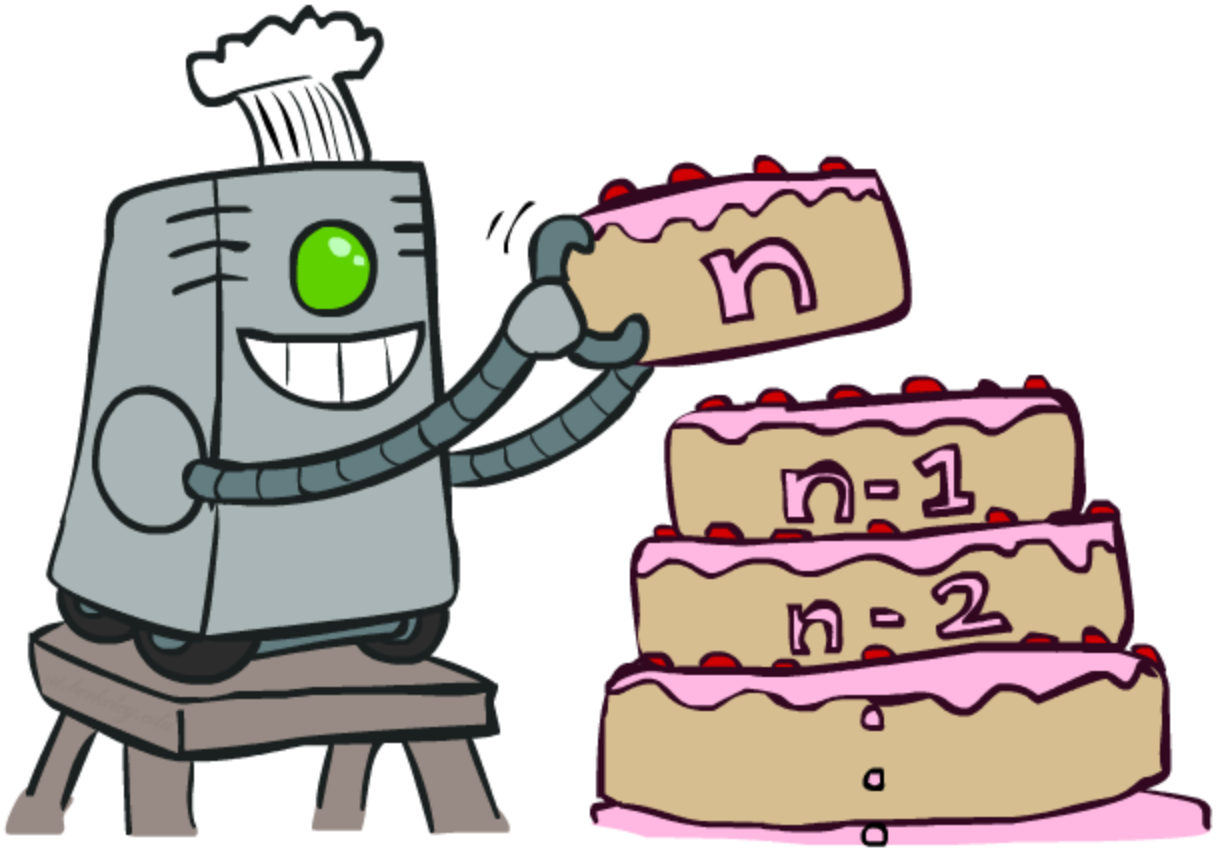


Noise = 0.2
Discount = 0.9
Living reward = 0

Computing Time-Limited Values



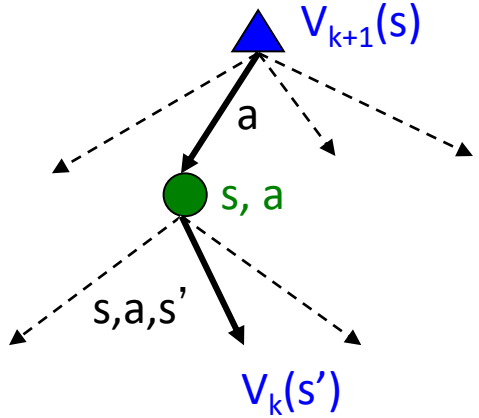
Value Iteration



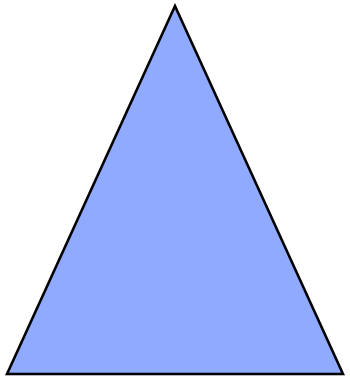
Value Iteration

- 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:

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



- 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



Example: Value Iteration

V_2

V_1

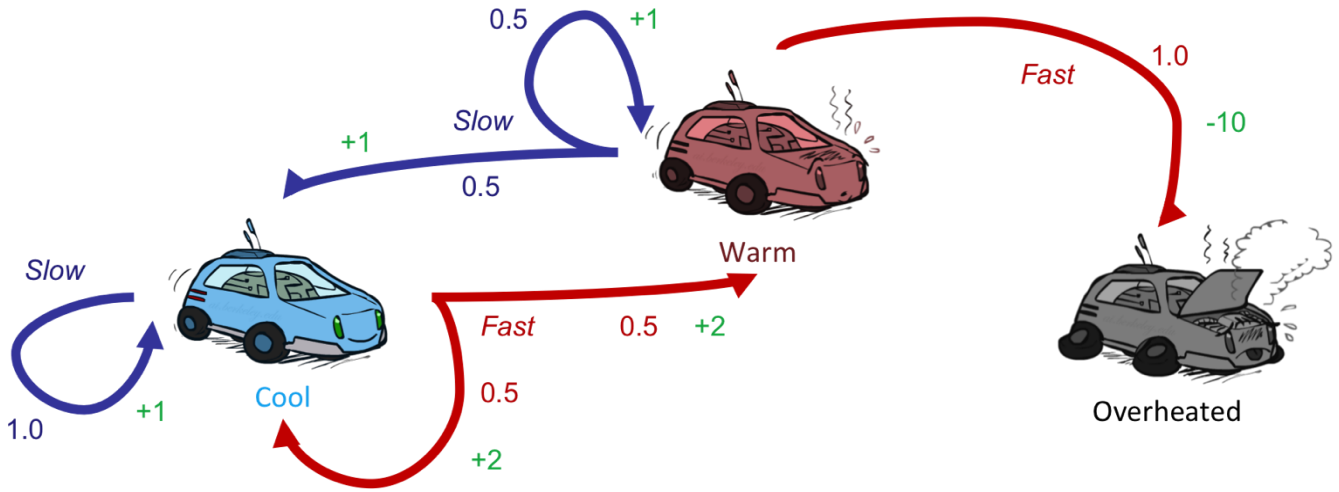
S: 1

F: .5*2+.5*2=2

V_0

0 0

0



Assume no discount!

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

Example: Value Iteration

V_2

V_1

2

S: .5*1+.5*1=1

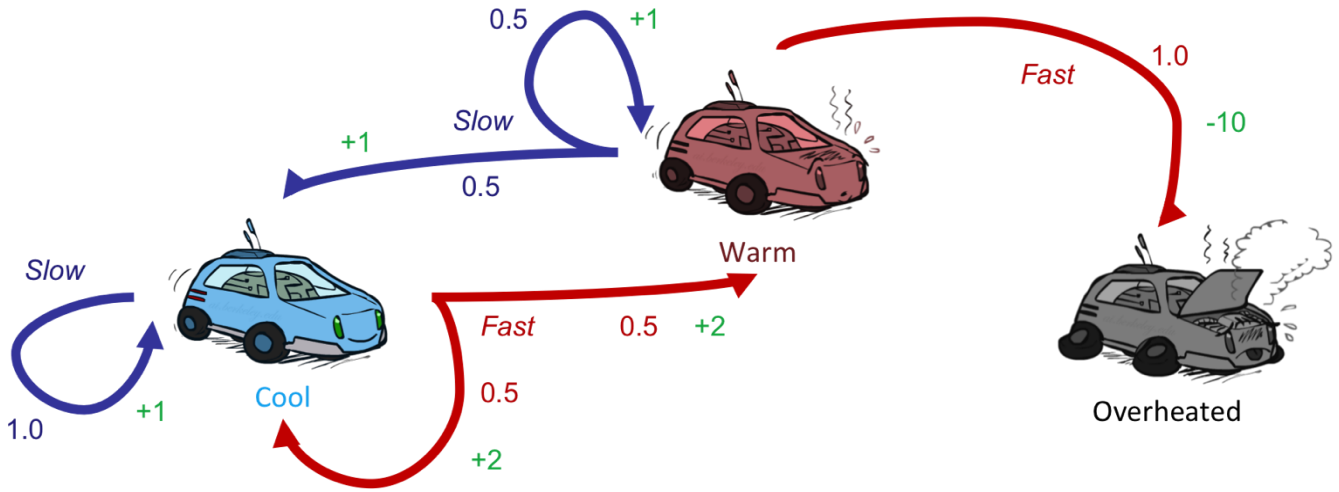
F: -10

V_0

0

0

0



Assume no discount!

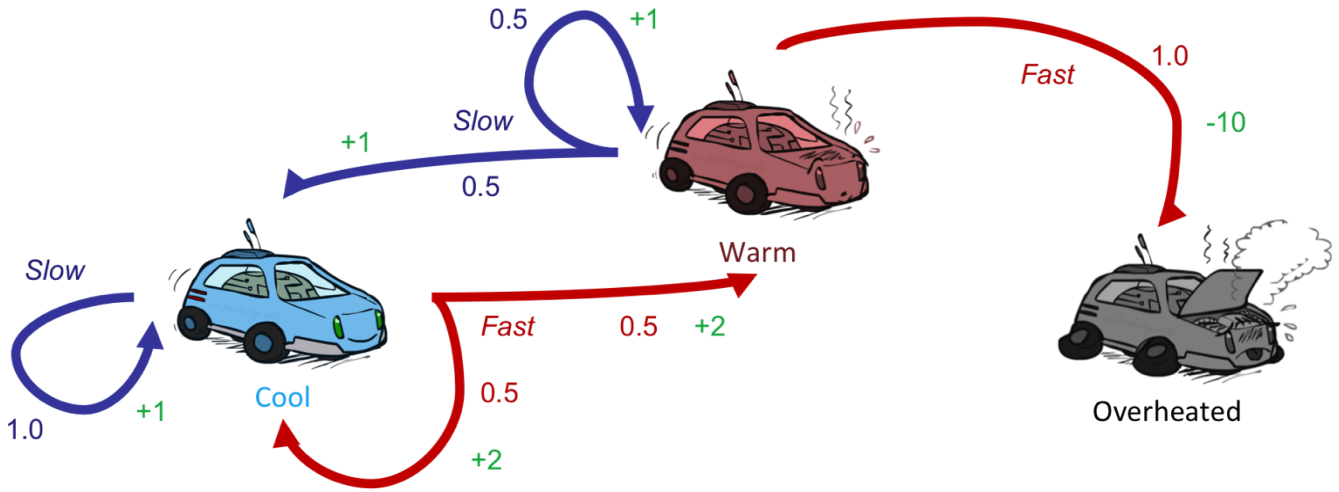
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right]$$

Example: Value Iteration

V_2

V_1 2 1 0

V_0 0 0 0



Assume no discount!

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

Example: Value Iteration

V_2

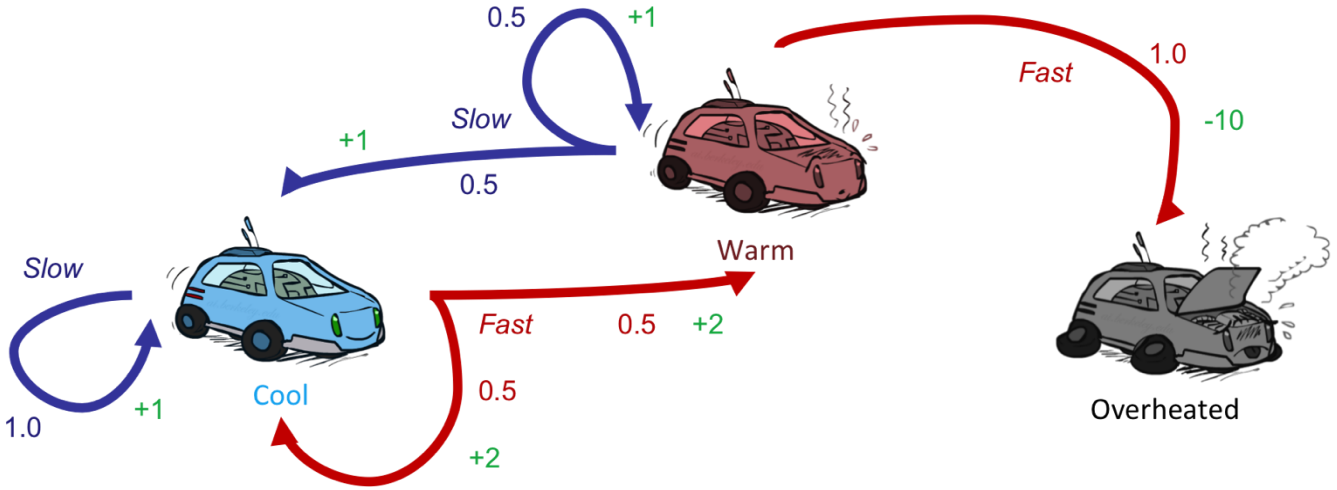
S: 1+2=3

F:

0.5*(2+2)+0.5*(2+1)=3.5

V_1 2 1 0

V_0 0 0 0

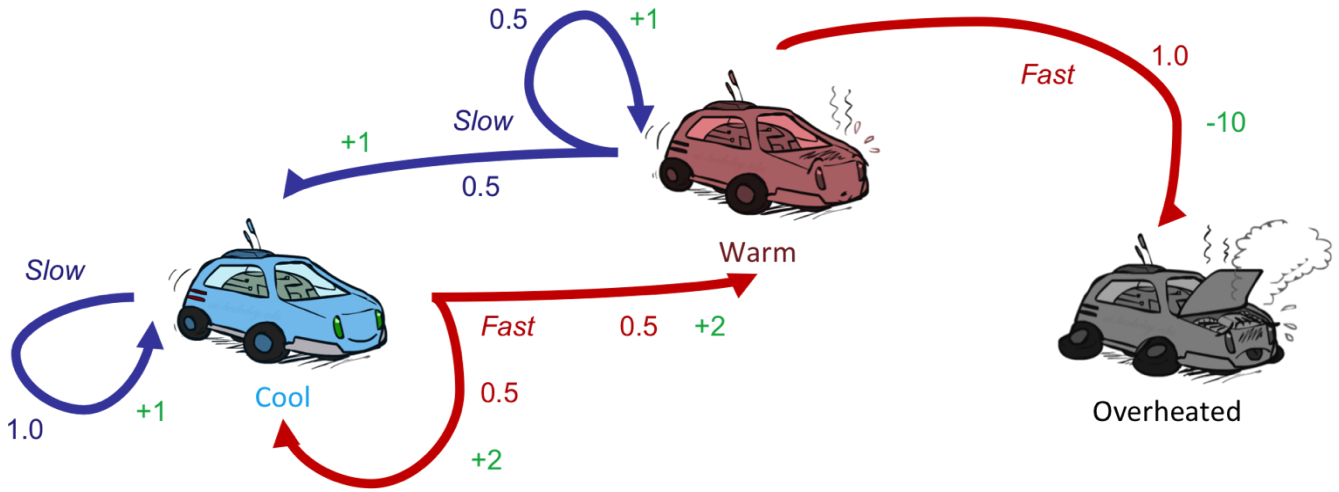


Assume no discount!

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

Example: Value Iteration

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



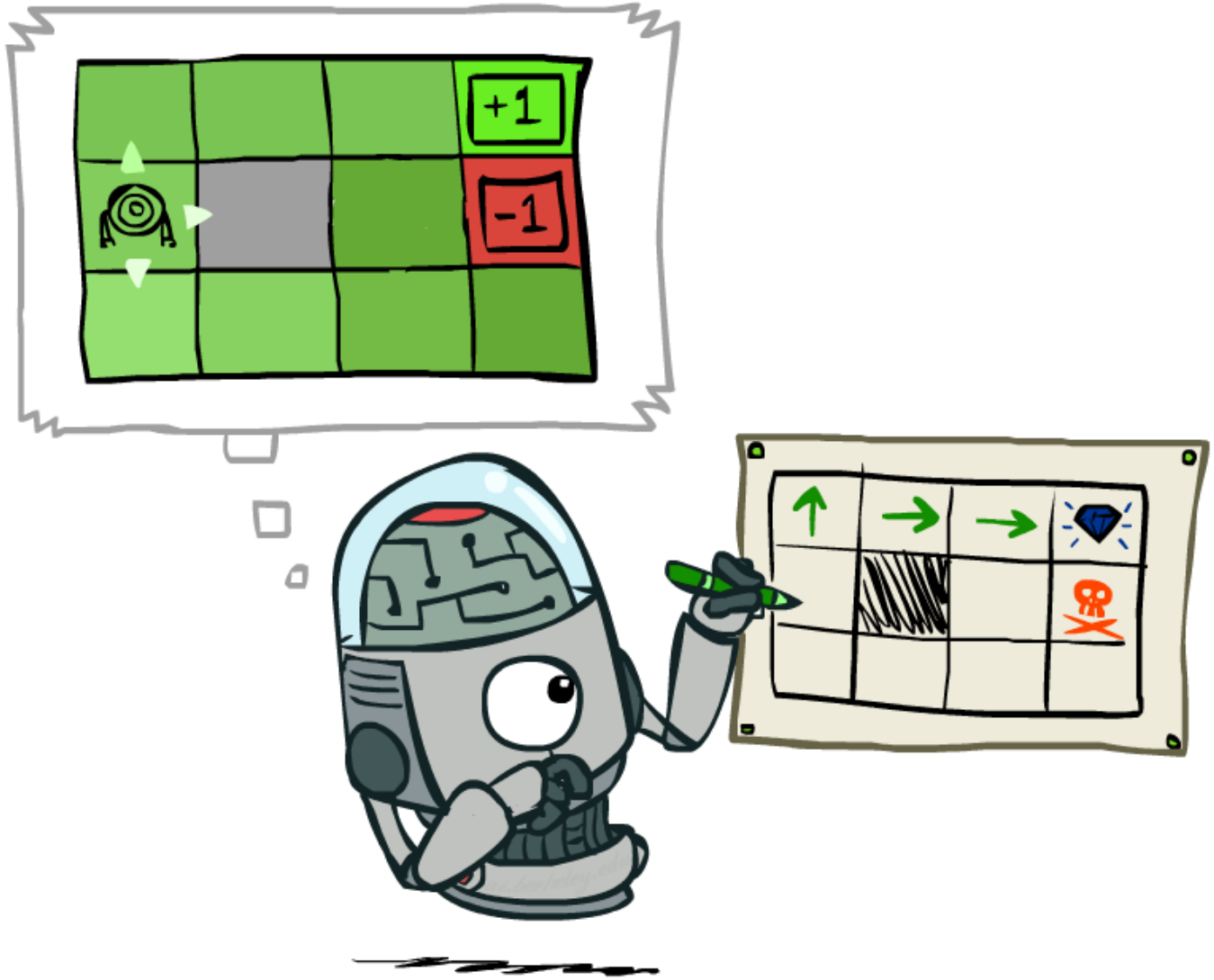
Assume no discount!

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

Outline

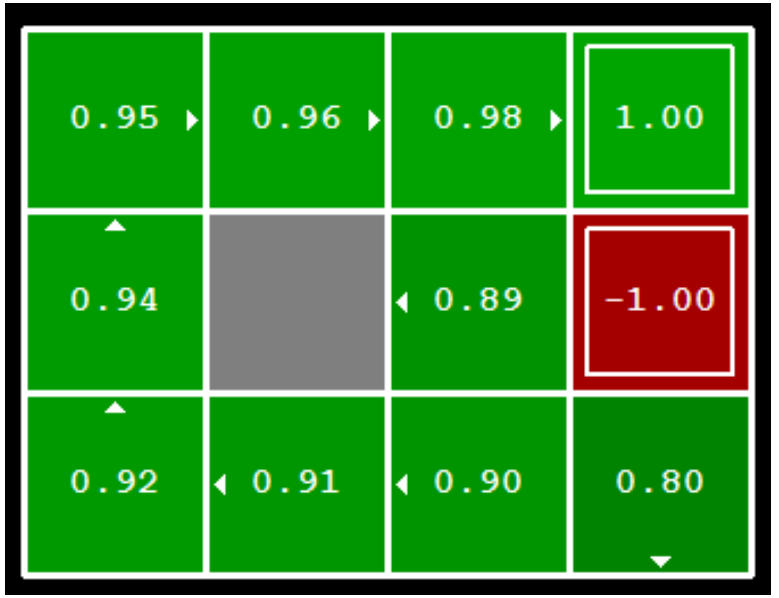
- Utilities to MDPs
- Value Iteration
- Policy Extraction
- Policy Iteration
- Convergence Analysis

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)

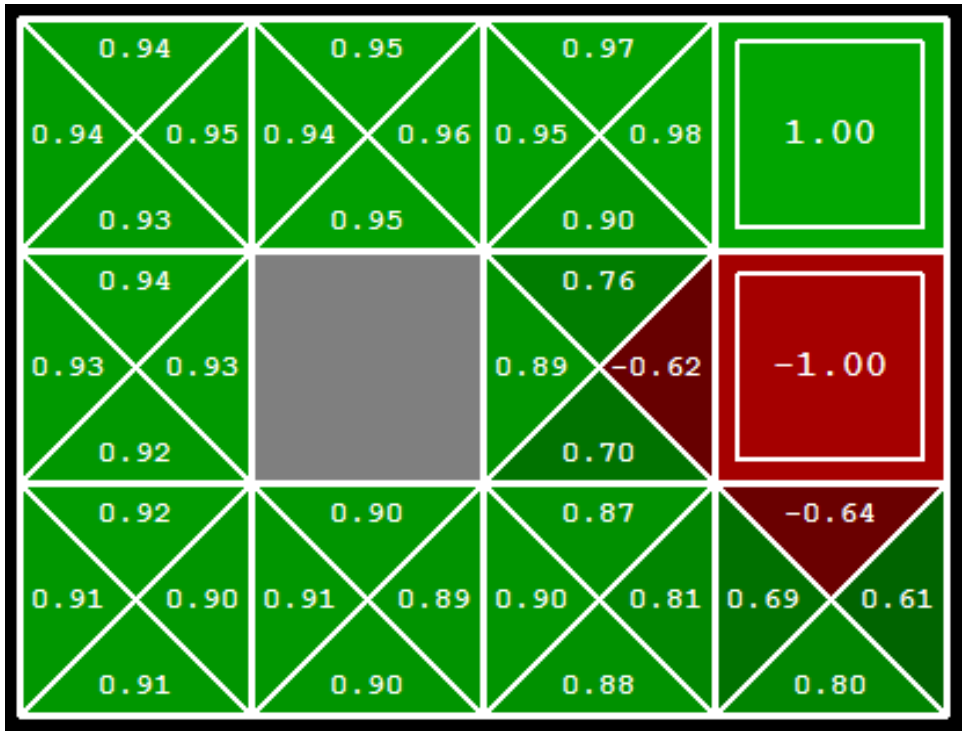


$$\pi^*(s) = \arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

- 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:

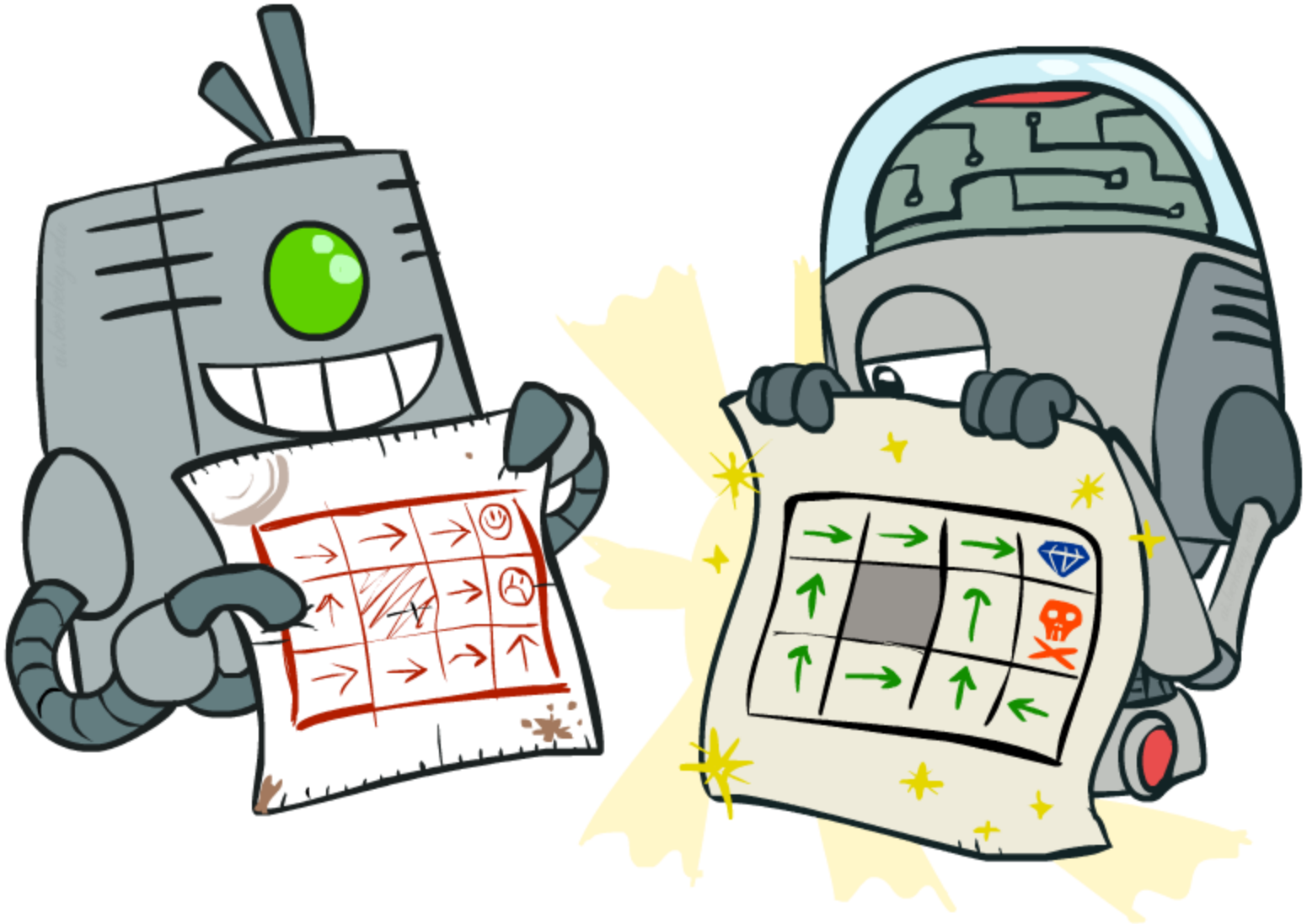


- How should we act?
  - Completely trivial to decide!

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

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

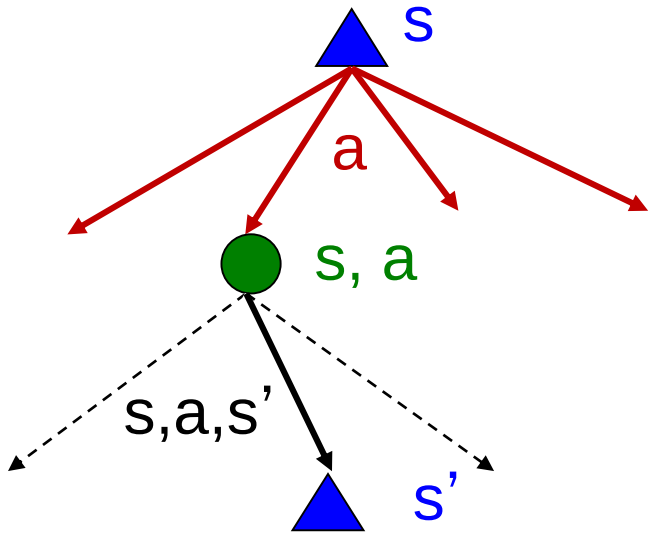
Policy Methods



Problems with Value Iteration

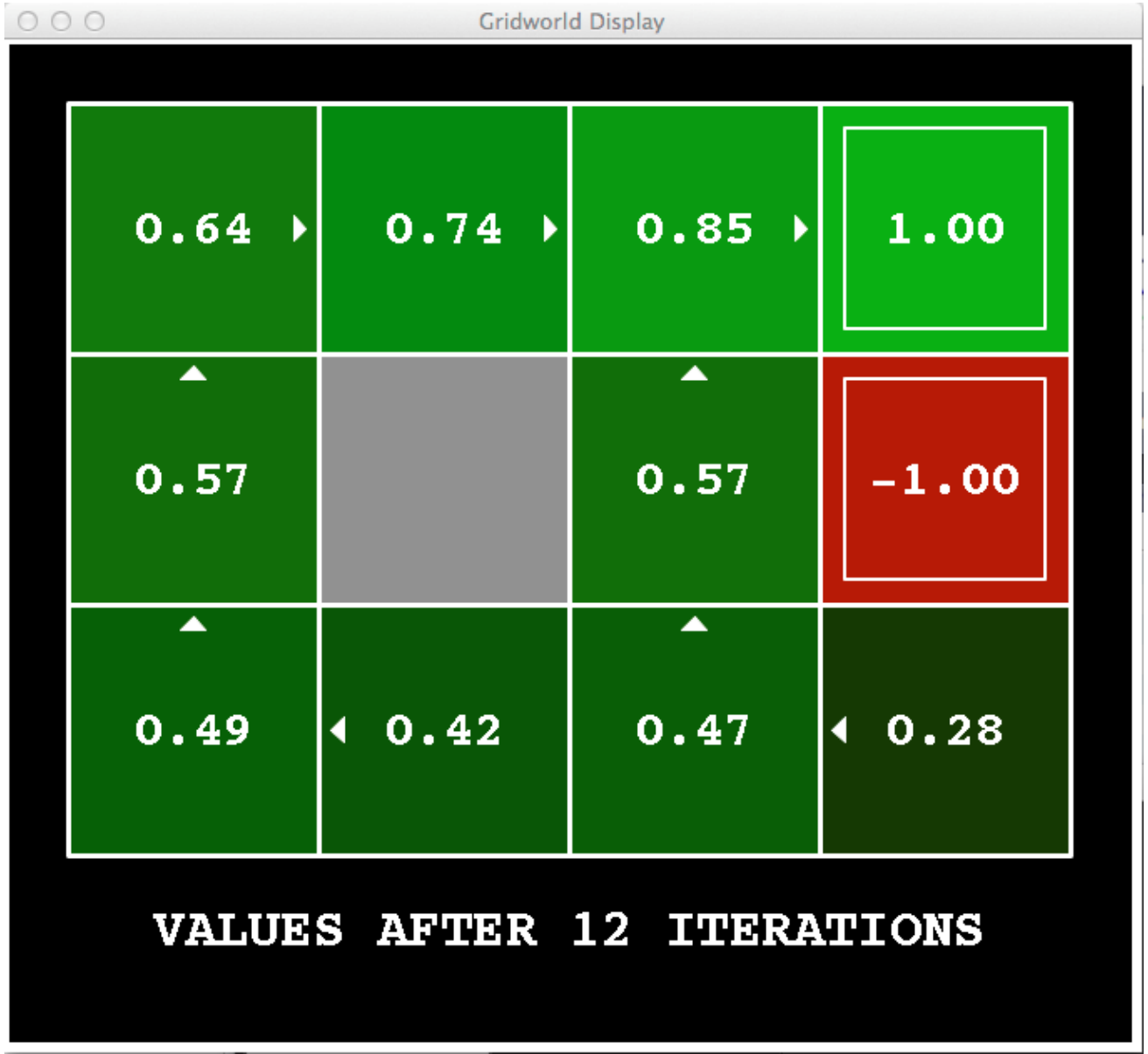
- Value iteration repeats the Bellman updates:

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



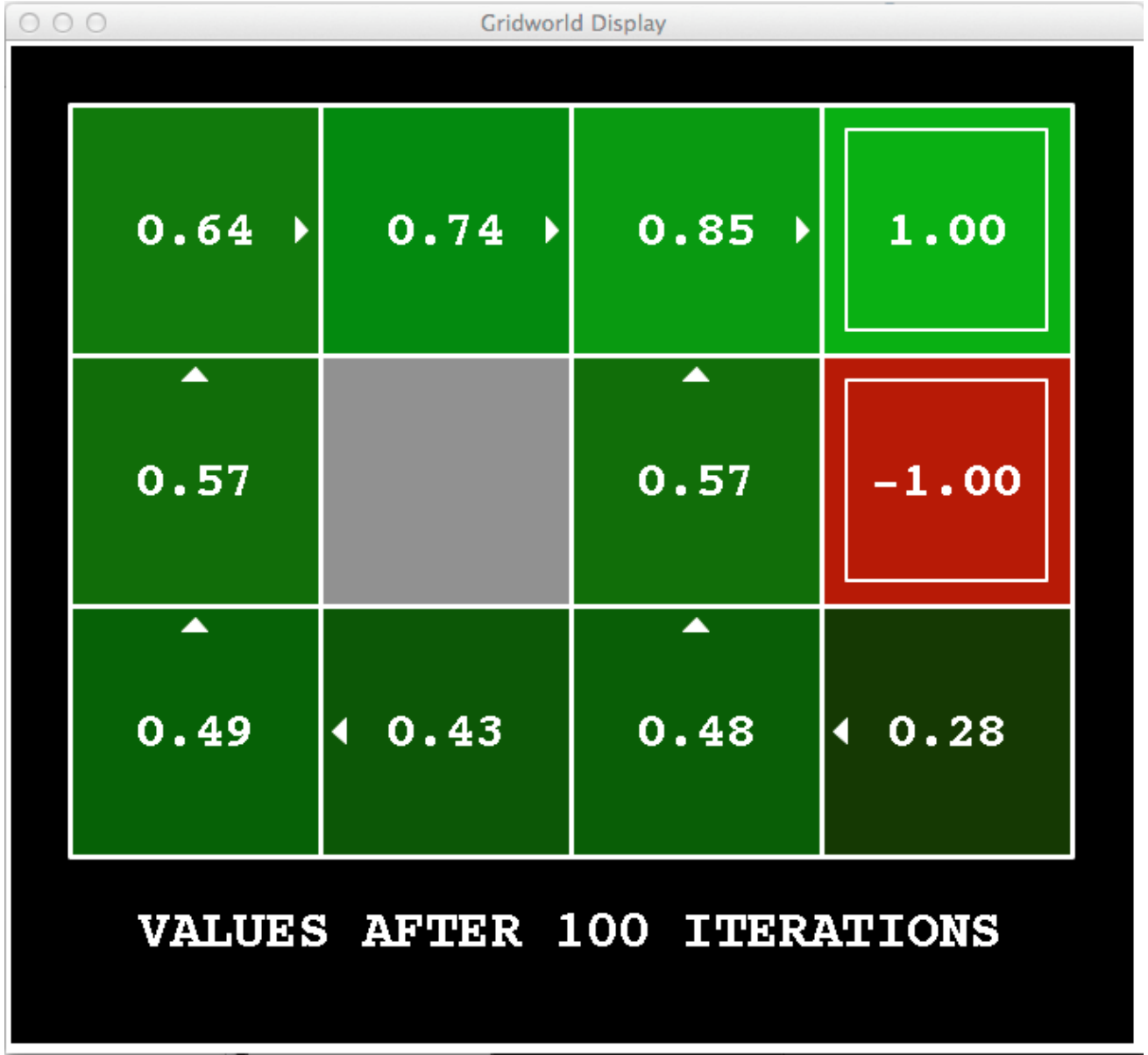
- Problem 1: It's slow – $O(S^2A)$ per iteration
- Problem 2: The "max" at each state rarely changes
- Problem 3: The policy often converges long before the values

k=12



Noise = 0.2
Discount = 0.9
Living reward = 0

k=100



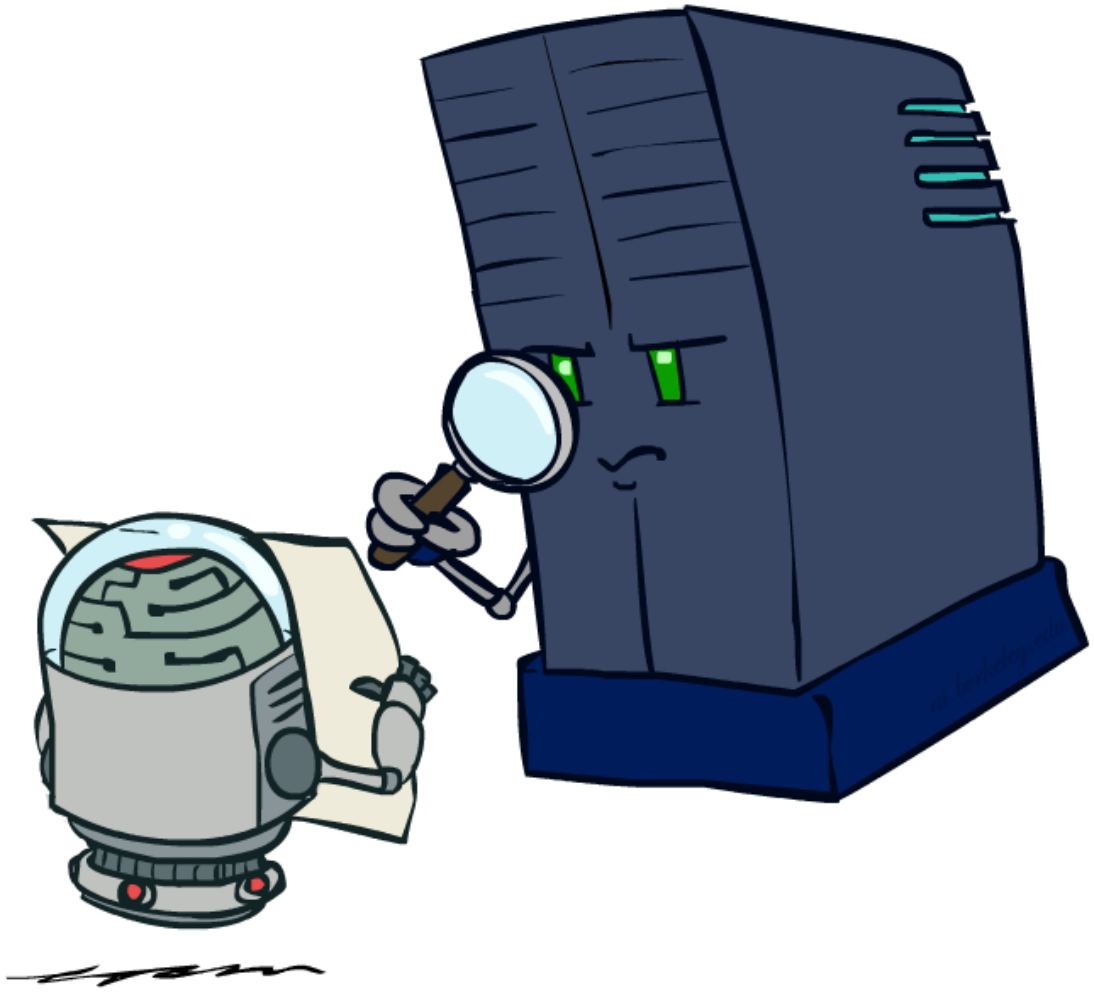
Noise = 0.2
Discount = 0.9
Living reward = 0

Policy Iteration

- Alternative approach for optimal values:
  - Step 1: Policy evaluation: calculate utilities for some fixed policy (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

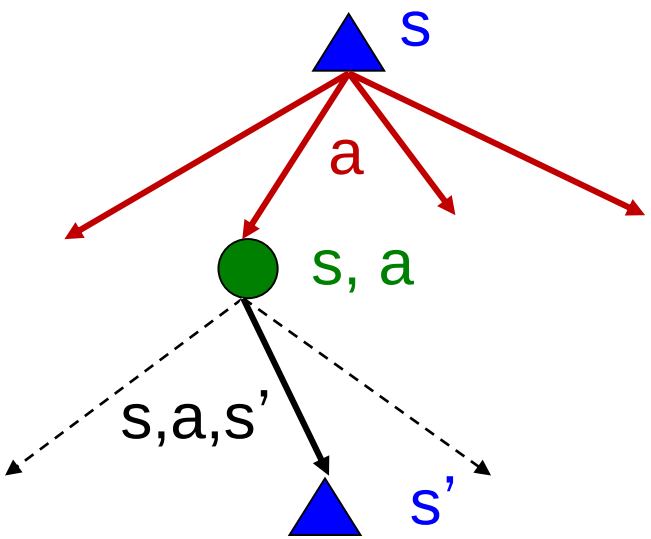
- This is policy iteration
  - It's still optimal!
  - Can converge (much) faster under some conditions

Policy Evaluation

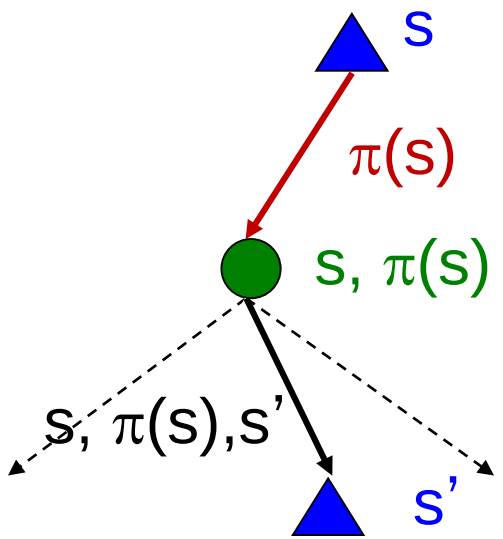


Fixed Policies

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 π(s), then the tree would be simpler – only one action per state
  - … though the tree's value would depend on which policy we fixed

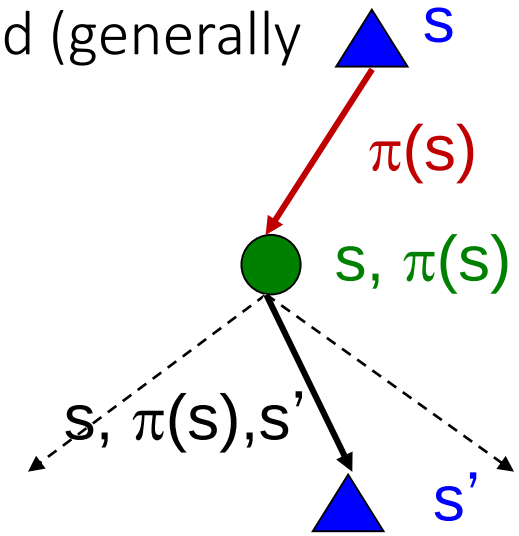
Utilities for a Fixed Policy

- 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):



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

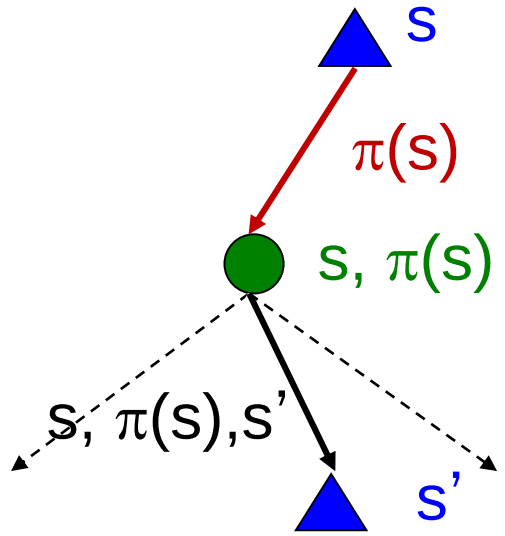
Policy Evaluation

- How do we calculate the V's for a fixed policy π?
- Idea 1: Turn recursive Bellman equations into updates (like value iteration)

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

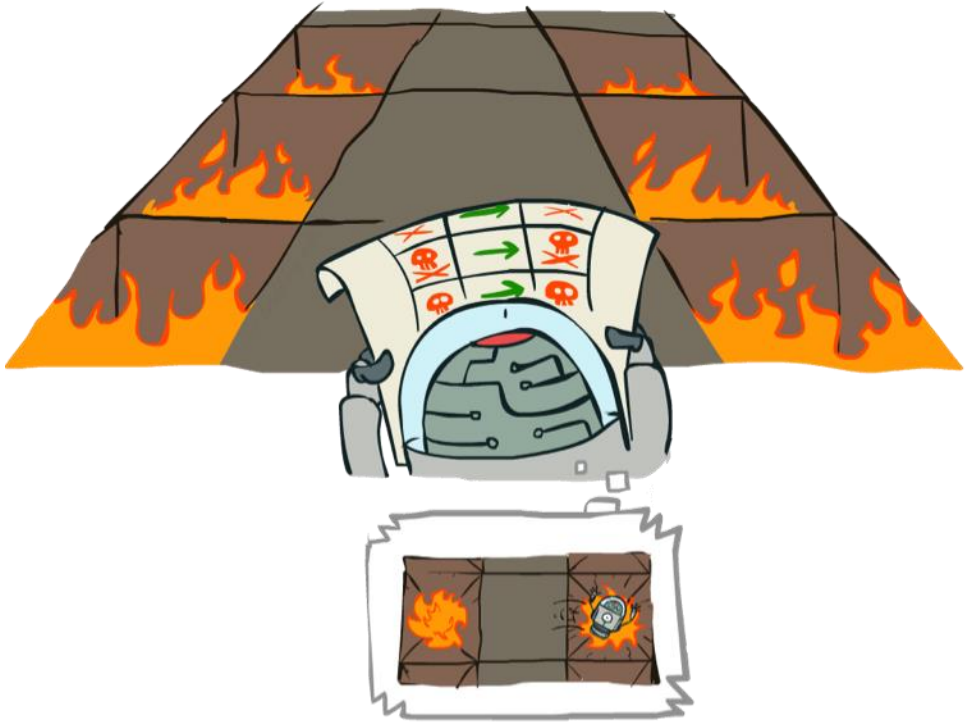
$$V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V_k^\pi(s')]$$

- 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

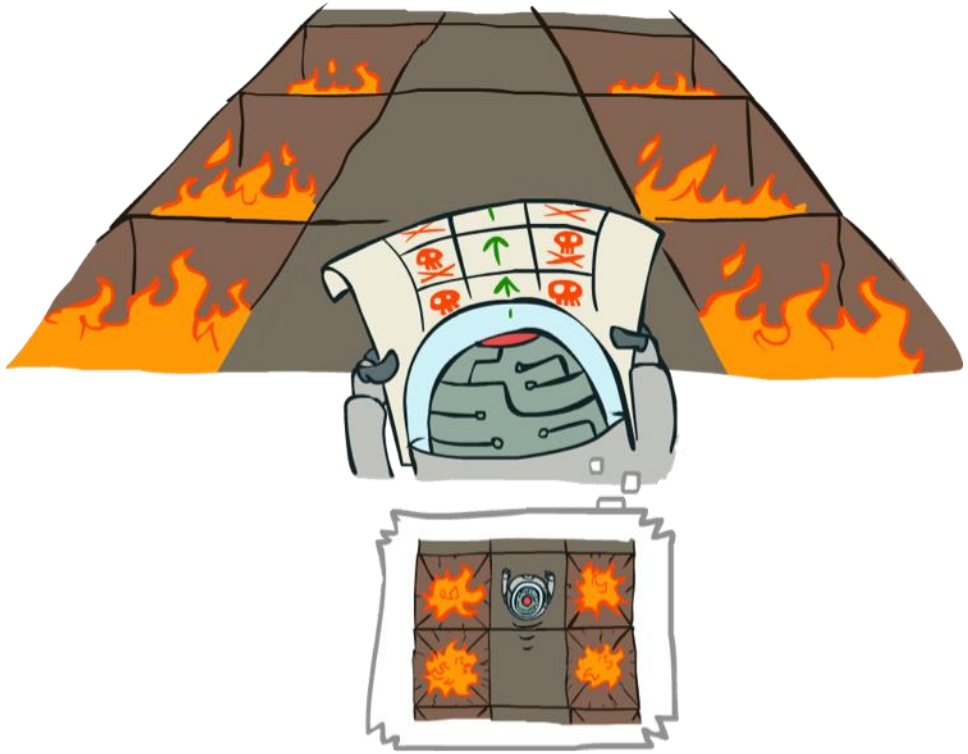


Example: Policy Evaluation

Always Go Right



Always Go Forward



Example: Policy Evaluation

Always Go Right

Always Go Forward

Quiz: Policy Evaluation – Mars Rover

s_1	s_2	s_3	s_4	s_5	s_6	s_7
						

$$P(s'|s,a_1) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \quad P(s'|s,a_2) = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

2 Deterministic Actions

Quiz: Policy Evaluation – Mars Rover

s_1	s_2	s_3	s_4	s_5	s_6	s_7
						

We will shortly be interested in not just evaluating the value of a single policy, but finding an optimal policy. Given this it is informative to think about properties of the potential policy space. First for the Mars rover example
[7 discrete states (location of rover); 2 actions: Left or Right]

How many deterministic policies are there?

- Pick one 2 / 14 / 7^2 / 2^7

Is the optimal policy (one with highest value) for a MDP unique?

- Yes / No / Not Sure

Quiz: Policy Evaluation – Mars Rover

s_1	s_2	s_3	s_4	s_5	s_6	s_7
						

We will shortly be interested in not just evaluating the value of a single policy, but finding an optimal policy. Given this it is informative to think about properties of the potential policy space. First for the Mars rover example
[7 discrete states (location of rover); 2 actions: Left or Right]

How many deterministic policies are there?

- Pick one 2 / 14 / 7^2 / 2^7 (max total policies $|A|^{|S|}$)

Is the optimal policy (one with highest value) for a MDP unique?

- Yes / No / Not Sure
  there may be two actions that have the same optimal value function

Quiz: Policy Evaluation – Mars Rover

- Dynamics: $p\ (s_6|s_6, a_1) = 0.5, p\ (s_7|s_6, a_1) = 0.5$…..
- Reward: for all actions, +1 in state s_1, +10 in state s_7, 0 otherwise
- Let $\pi(s) = a_1\ \forall s$, assume V_k=[1 0 0 0 0 0 0 10] for $k = 1, \gamma = 0.5$
- Calculate $V_{k+1}\ (s_6)$?

Quiz: Policy Evaluation – Mars Rover

- Dynamics: $p\ (s_6|s_6, a_1) = 0.5, p\ (s_7|s_6, a_1) = 0.5$.....
- Reward: for all actions, +1 in state s_1, +10 in state s_7, 0 otherwise
- Let $\pi(s) = a_1\ \forall s$, assume V_k=[1 0 0 0 0 0 0 10] for $k = 1, \gamma = 0.5$
- Calculate $V_{k+1}\ (s_6)$?

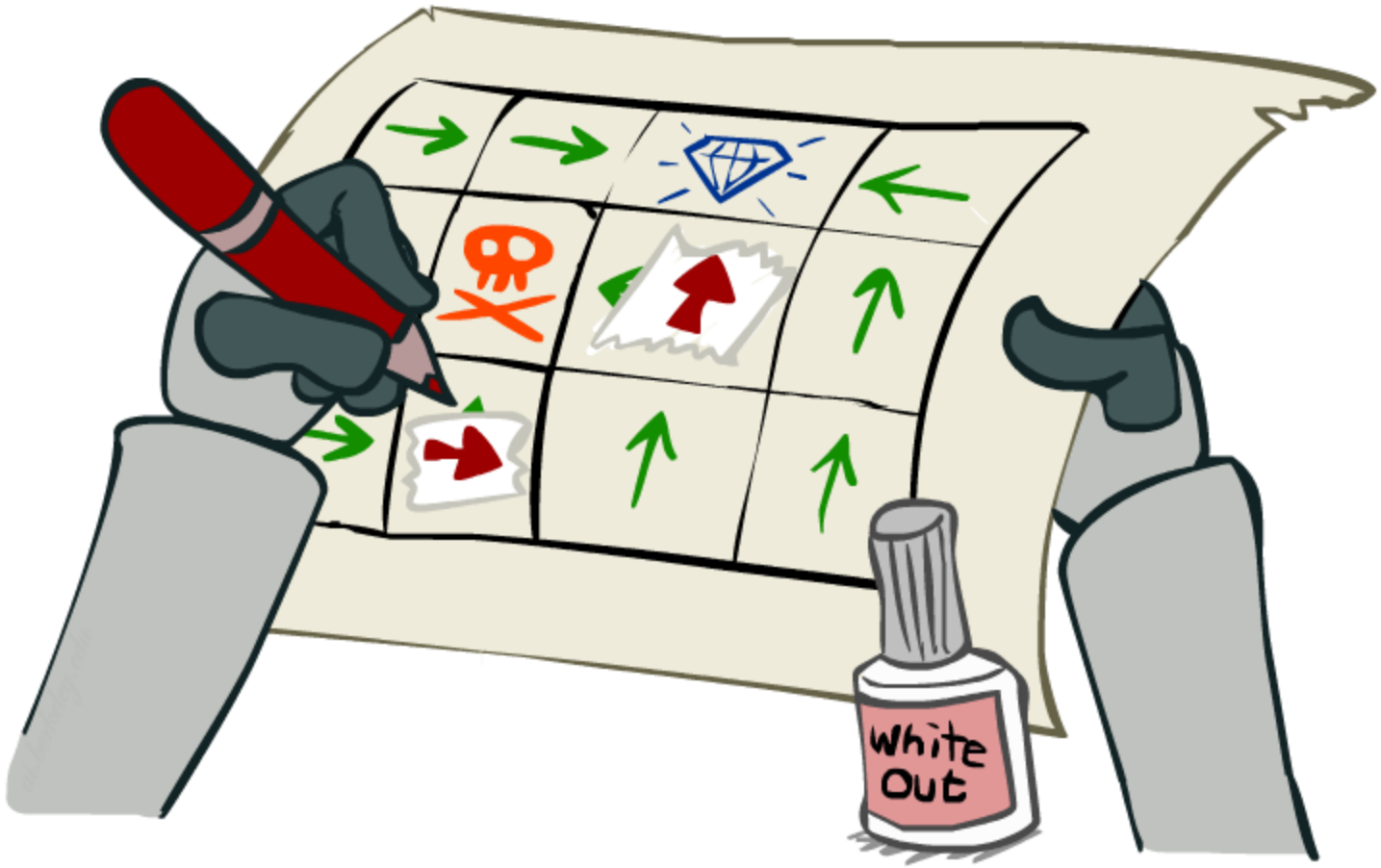
$$V_{k+1}^{\pi}(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V_k^{\pi}(s')]$$

$$V_{k+1}(s_6) = r(s_6, a_1) + \gamma * 0.5 * V_k(s_6) + \gamma * 0.5 * V_k|(s_7)$$

$$V_{k+1}(s_6) = 0 + 0.5 * 0.5 * 0 + .5 * 0.5 * 10$$

$$V_{k+1}(s_6) = 2.5$$

Policy Iteration



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

- There exists a unique optimal value function
- Optimal policy for a MDP in an infinite horizon problem (agent acts forever) is
  - Deterministic
  - Stationary (does not depend on time step)
  - Unique?
    Not necessarily, may have state-actions with identical optimal values

Policy Iteration

$$Q^{\pi_i}(s,a) =| R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^{\pi_i}(s')$$

$$\max_a Q^{\pi_i}(s,a) \geq R(s,\pi_i(s)) + \gamma \sum_{s' \in S} P(s'|s,\pi_i(s)) V^{\pi_i}(s') = V^{\pi_i}(s)$$

$$\pi_{i+1}(s) = \arg\max_a Q^{\pi_i}(s,a)$$

Suppose we take π_{i+1} for one step and follow π_i thereafter:

- Expected sum of rewards is at least as good as if we had always followed π_i

Hence π_{i+1} improves over π_i

Monotonic Improvement in Policy Iteration

- Definition:

$$V^{\pi_1} \geq V^{\pi_2} : V^{\pi_1}(s) \geq V^{\pi_2}(s), \forall s \in S$$

- Proposition $V^{\pi_{i+1}} \geq V^{\pi_i}$ with strict inequality if π_i is suboptimal, since we can find a π_{i+1} we get from policy improvement on π_i

Monotonic Improvement in Policy Iteration

$$
\begin{aligned}
V^{\pi_i}(s) &\leq \max_a Q^{\pi_i}(s,a) \\
&= \max_a R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^{\pi_i}(s') \\
&= R(s, \pi_{i+1}(s)) + \gamma \sum_{s' \in S} P(s'|s, \pi_{i+1}(s)) V^{\pi_i}(s') \ \text{//by the definition of } \pi_{i+1} \\
&\leq R(s, \pi_{i+1}(s)) + \gamma \sum_{s' \in S} P(s'|s, \pi_{i+1}(s)) \left(\max_{a'} Q^{\pi_i}(s', a') \right) \\
&= R(s, \pi_{i+1}(s)) + \gamma \sum_{s' \in S} P(s'|s, \pi_{i+1}(s)) \\
&\quad \left(R(s', \pi_{i+1}(s')) + \gamma \sum_{s'' \in S} P(s''|s', \pi_{i+1}(s')) V^{\pi_i}(s'') \right) \\
&\quad \vdots \\
&= V^{\pi_{i+1}}(s)
\end{aligned}
$$

Policy Iteration as Bellman Recursion

Bellman backup operator B^π for a particular policy is defined as

$$B^\pi V(s) = R^\pi(s) + \gamma \sum_{s' \in S} P^\pi(s'|s) V(s)$$

Policy evaluation amounts to computing the fixed point of B^π

To do policy evaluation, repeatedly apply operator until V stops changing

$$V^\pi = B^\pi B^\pi \cdots B^\pi V$$

To do policy improvement

$$\pi_{k+1}(s) = \arg\max_a R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^{\pi_k}(s')$$

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 Equations

- Value iteration equation:

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

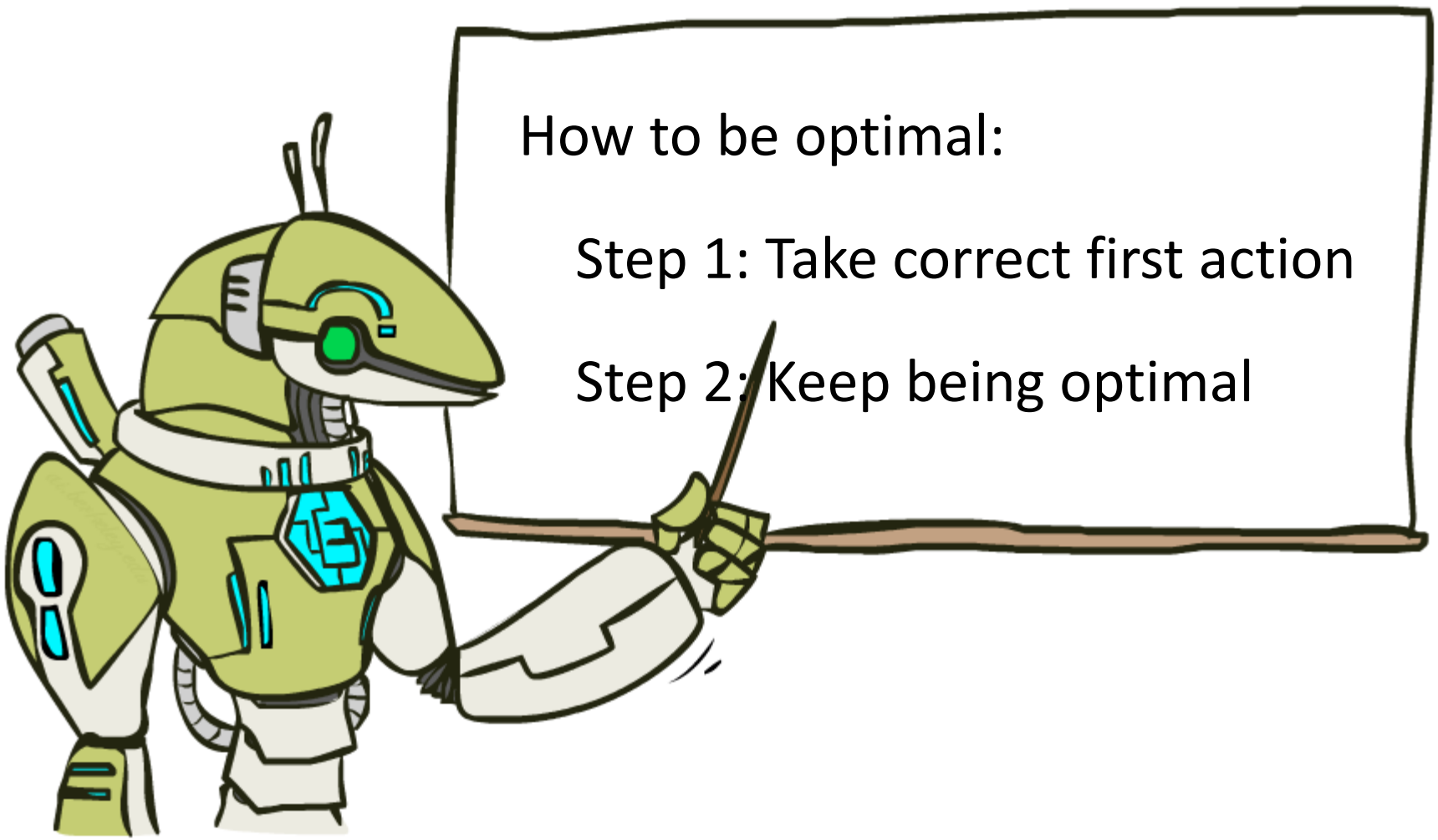
- Policy evaluation equation:

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

- Policy iteration equation:

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

The Bellman Equations



Convergence when Solving MDPs

- Redefine value update as general Bellman Utility update
  - Recursive update or utility (sum of discounted reward)

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

$$\downarrow$$

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum P(s'|s,a) U_i(s')$$

- How does this converge?
  - Assume fixed policy $\pi_i(s)$.
  - *R(s)* is the short term reward of being in *s*

Convergence when Solving MDPs

- How does this update rule converge?

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum P(s'|s,a) U_i(s')$$

- Re-write update: $U_{i+1} \leftarrow BU_i \qquad v' = Av$
  - *B* is a linear operator (like a matrix)
  - *U* is a vector

Bellman Recursion

- Interested in delta between Utilities: $\|U_{i+1} - U_i\|$

$$\|BU_{i+1} - BU_i\| \leq \gamma \|U_{i+1} - U_i\|$$

Detour: Convergence of Bellman backup

Let $\|V - V'\| = \max_s |V(s) - V'(s)|$ be the infinity norm

$$
\begin{aligned}
\|BV_k - BV_j\| &= \left\| \max_a \left(R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V_k(s') \right) - \max_{a'} \left(R(s,a') + \gamma \sum_{s' \in S} P(s'|s,a') V_j(s') \right) \right\| \\
&\leq \max_a \left\| \left(R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V_k(s') - R(s,a) - \gamma \sum_{s' \in S} P(s'|s,a) V_j(s') \right) \right\| \\
&= \max_a \left\| \gamma \sum_{s' \in S} P(s'|s,a)(V_k(s') - V_j(s')) \right\| \\
&\leq \max_a \left\| \gamma \sum_{s' \in S} P(s'|s,a) \|V_k - V_j\|) \right\| \\
&= \max_a \left\| \gamma \|V_k - V_j\| \sum_{s' \in S} P(s'|s,a)) \right\| \\
&= \gamma \|V_k - V_j\|
\end{aligned}
$$

Note: Even if all inequalities are equalities, this is still a contraction if $\gamma < 1$

Convergence when Solving MDPs

- How does this delta converge?

$$\|BU_{i+1} - BU_i\| \leq \gamma \|U_{i+1} - U_i\|$$

  - Utility error estimate reduced by γ each iteration:
- Total Utilities are bounded,

$$\sum_{i=0}^{\infty} R_{max}\gamma^i \qquad \pm \frac{R_{max}}{(1-\gamma)}$$

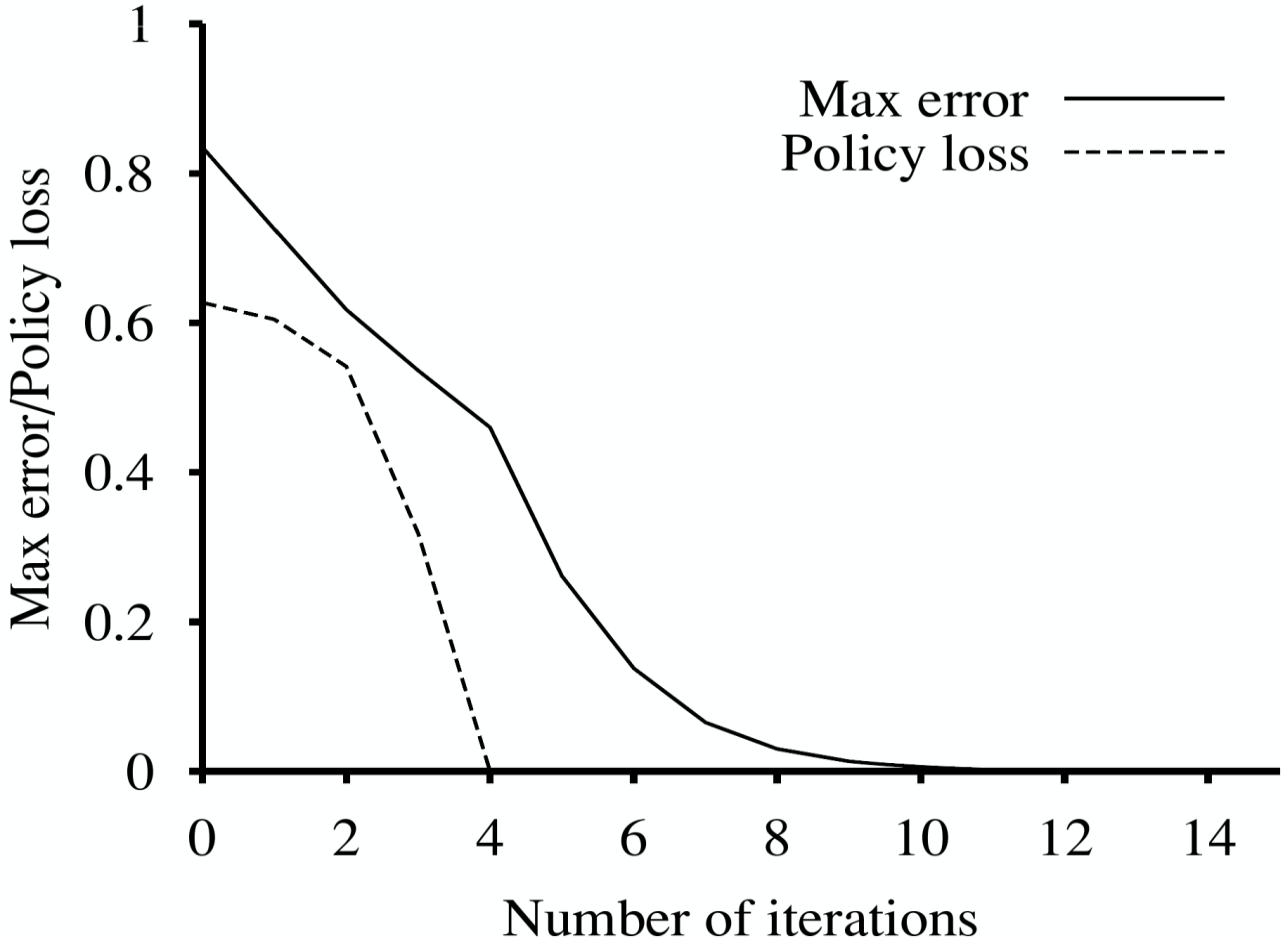
- Consider minimum initial error:
  - (Max norm)
- Max error: reduce by discount each step. $\|U_0 - U\| \leq \frac{2R_{max}}{(1-\gamma)}$

Utility Error Bound

- Error at step 0: $$\|U_0 - U\| \leq \frac{2R_{max}}{(1-\gamma)}$$
- Error at step N: $$\|U_N - U\| = \gamma^N \cdot \frac{2R_{max}}{(1-\gamma)} < \epsilon$$
- Steps for error below ϵ: $$N = \frac{log\left(\frac{2R_{max}}{\epsilon(1-\gamma)}\right)}{log(\frac{1}{\gamma})}$$

MDP Convergence Visualized

- Value iteration converges exponentially (with discount factor)
- Policy iteration will converge linearly to 0.



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

Acknowledgements

Slides adapted from

CS 188 UC Berkeley
Pieter Abbeel, Dan Klein et al.

CS 285 UC Berkeley
Sergey Levine

CSC 498 Univ of Toronto
Animesh Garg