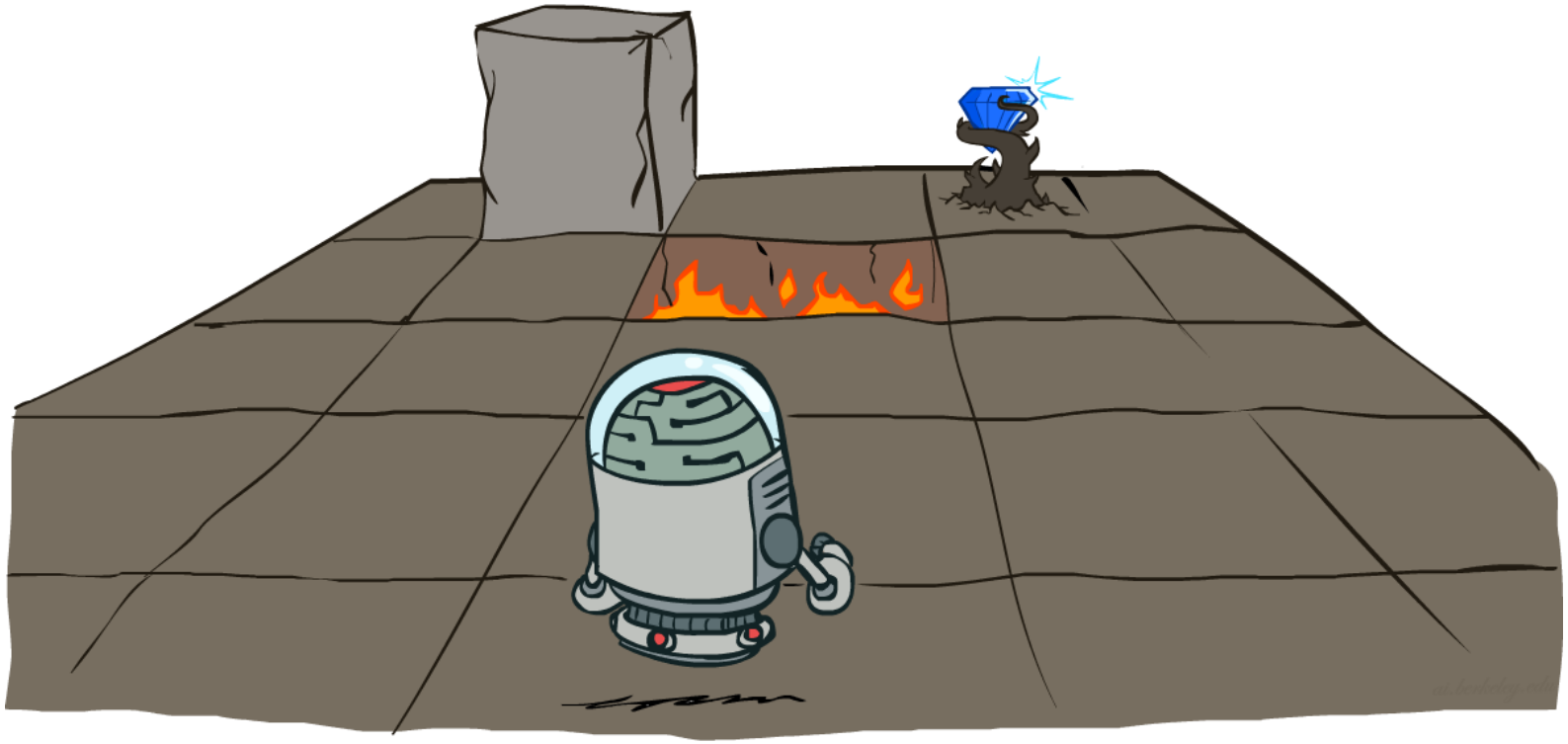
CS 594 Modern Reinforcement Learning

Lecture 1: Introduction

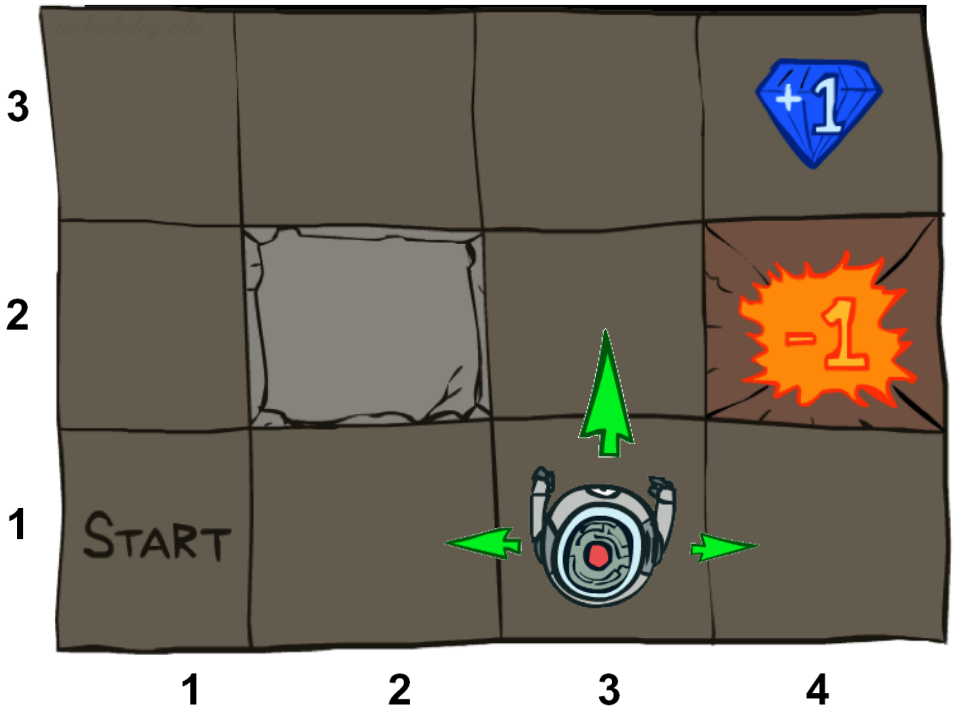
Review: Markov Decision Processes and Dynamic Programming (Chapters 3 and 4 of Sutton and Barto)



[Slides adapted from those created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley. All materials available at http://ai.berkeley.edu.]

Example: Grid World

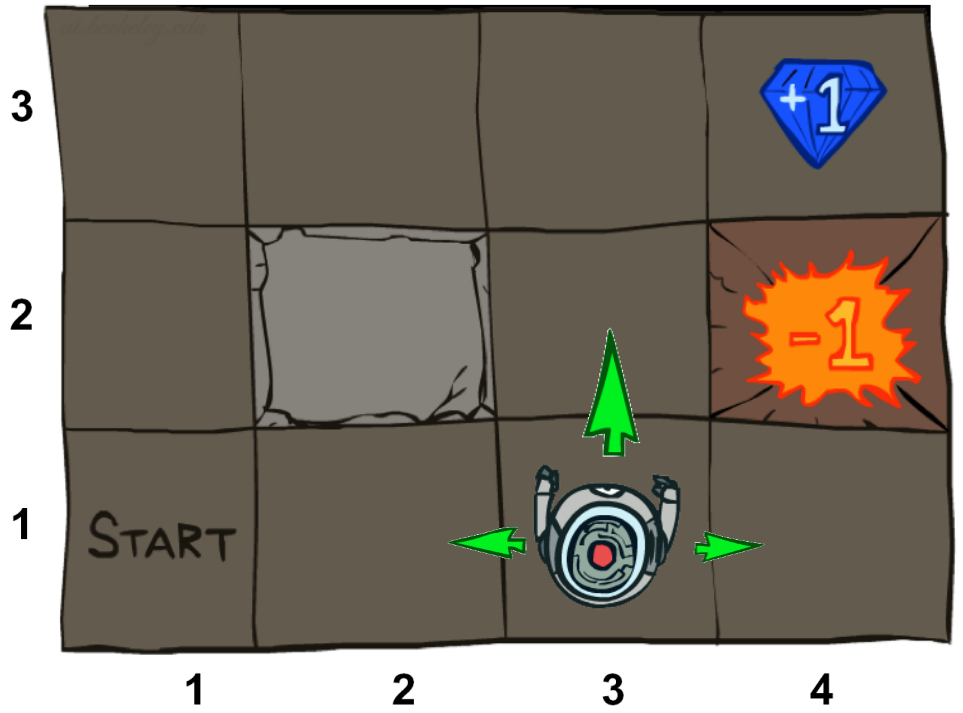
- A maze-like problem
  - The agent lives in a grid
  - Walls block the agent's path
- Noisy movement: actions do not always go as planned
  - 80% of the time, the action North takes the agent North (if there is no wall there)
  - 10% of the time, North takes the agent West; 10% East
  - If there is 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



Markov Decision Processes

- Trajectory $S_0, A_0, R_0, S_1, A_1, R_1, \ldots$

- A (finite) MDP is defined by:
  - A finite set of states $s \in S$
  - A finite set of actions $a \in A$
  - A finite set of rewards $r \in R$
  - Dynamics $p(s',r|s,a) = Pr(S_t=s',R_t=r|S_{t-1}=s,A_{t-1}=a)$
  - Discount Rate γ
  - Possibly start state (distribution), terminal state



- Derived Quantities:
  - State transition probabilities $p(s'|s,a) = Pr(S_t=s'|S_{t-1}=s,A_{t-1}=a) = \Sigma_{r \in R}\ p(s',r|s,a)$
  - Expected rewards $r(s,a) = E[R_t|S_{t-1}=s,A_{t-1}=a] = \Sigma_{r \in R} r\ \Sigma_{s' \in S}\ p(s',r|s,a)$
  - Or $r(s,a,s') = E[R_t|S_{t-1}=s,A_{t-1}=a, S_t=s'] = \Sigma_{r \in R} r\ p(s',r|s,a) / p(s'|s,a)$

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

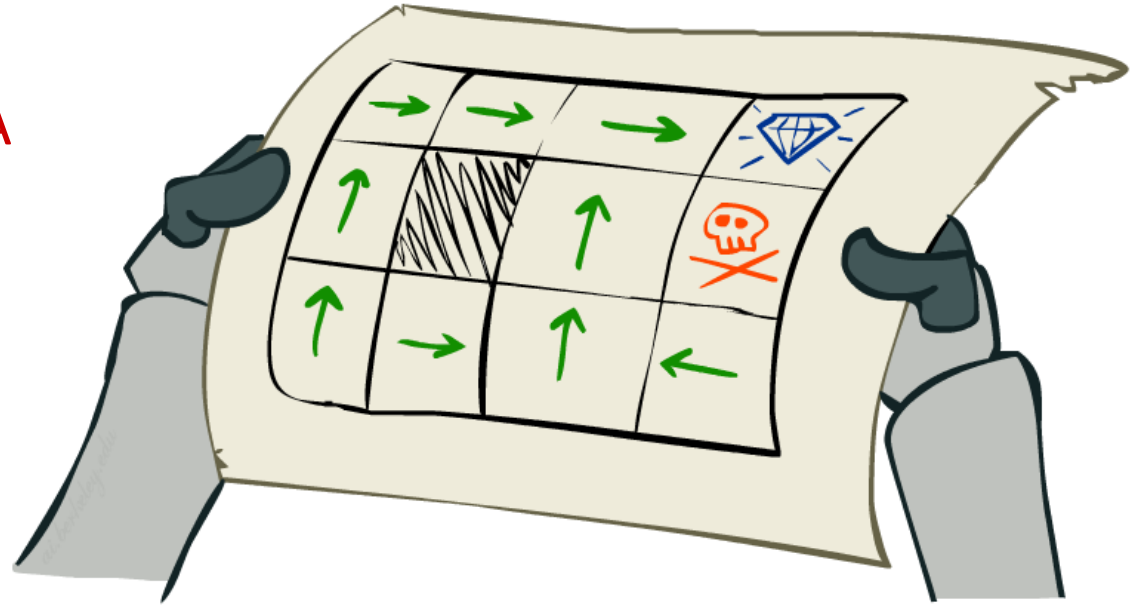
γ^2

Worth In Two Steps

- Return $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+2} + \ldots = \sum_{k=0}^{T} \gamma^k R_{t+k+1}$

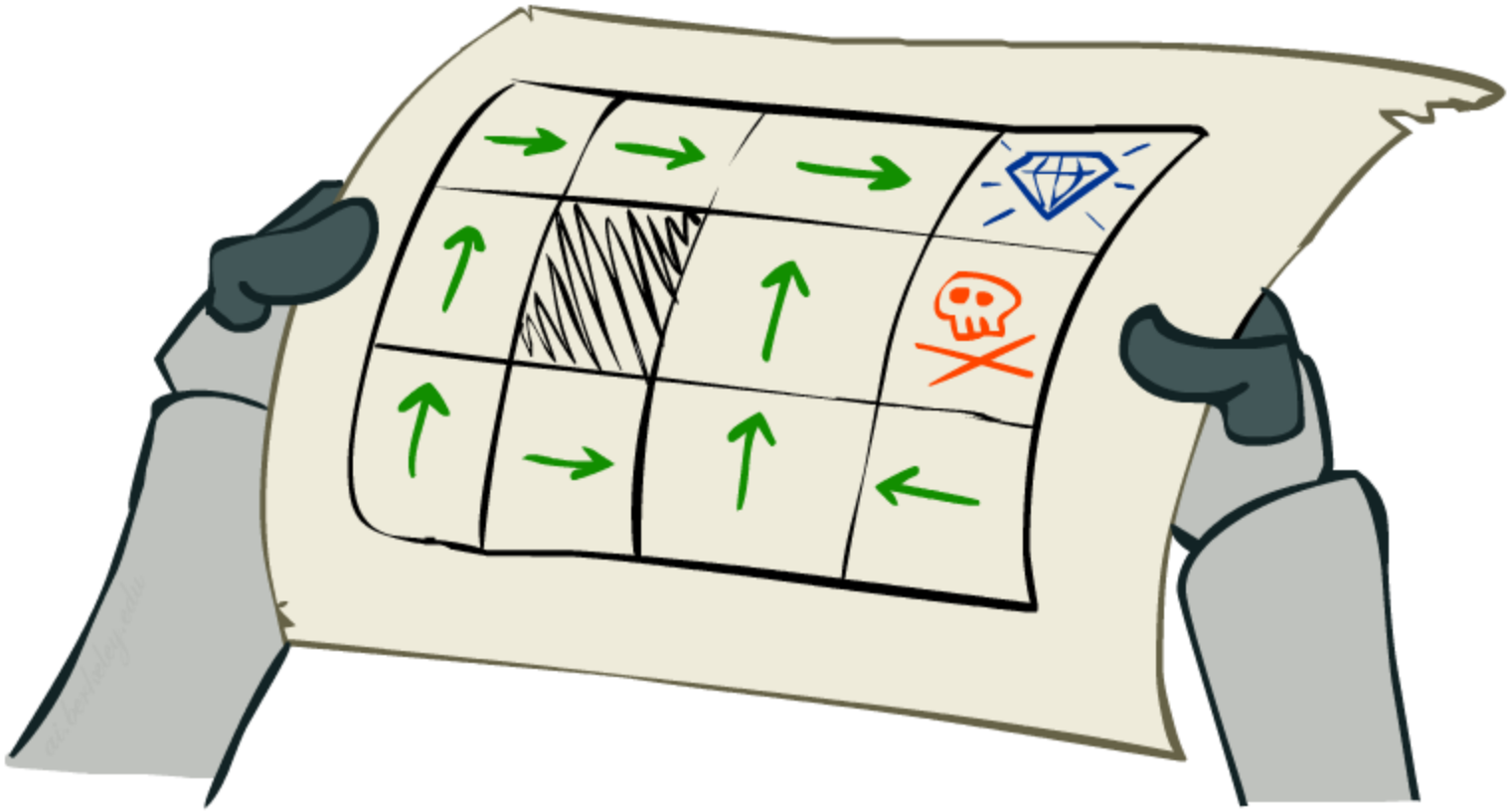
Policies

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



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

Solving MDPs



Optimal Quantities

- The optimal policy:

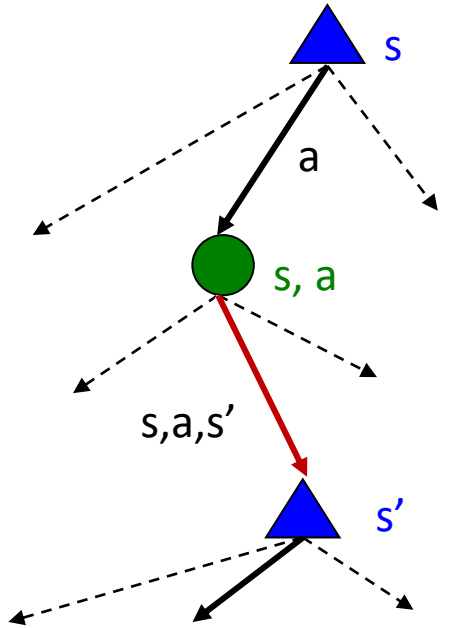
  $\pi_*(s)$ = optimal action from state s

- The value of a state s:

  $v_*(s)$ = expected return starting in s and acting optimally

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

  $q_*(s,a)$ = expected return starting out having taken action a from state s and (thereafter) acting optimally



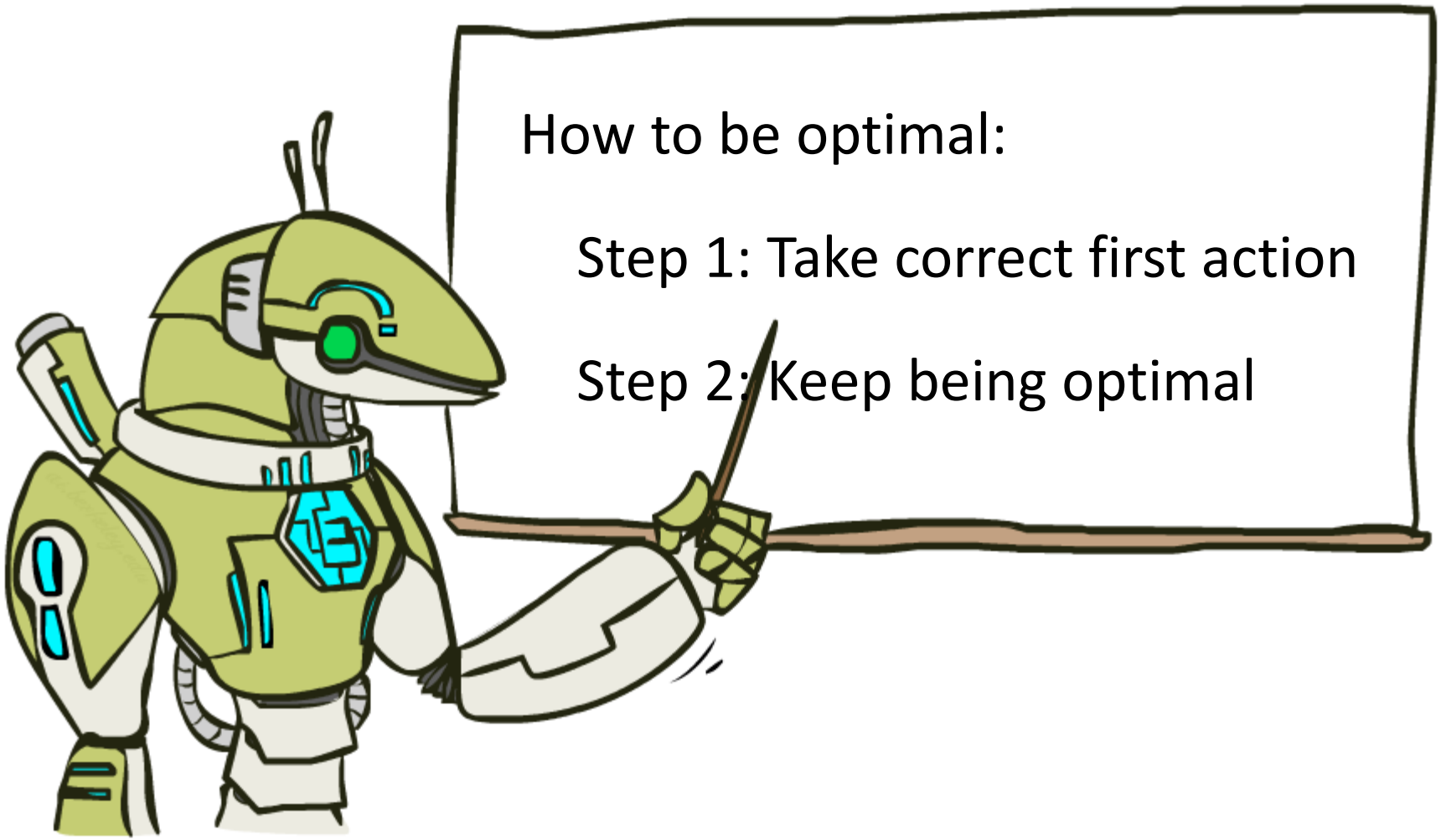
s is a *state*

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

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

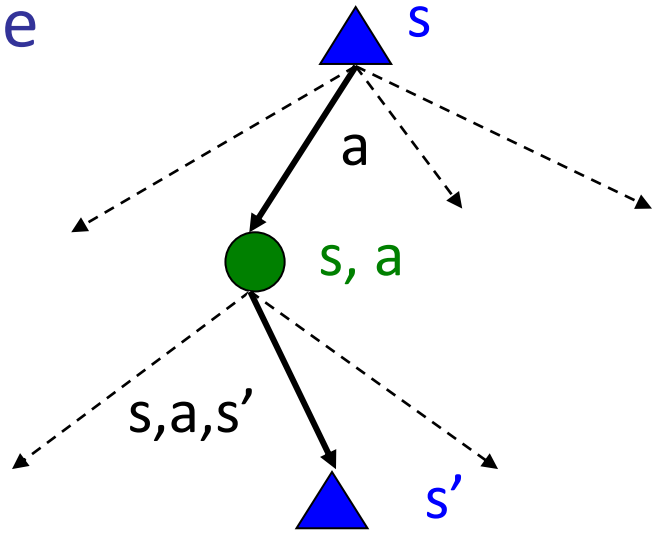
[Demo – gridworld values (L8D4)]

The Bellman Equations



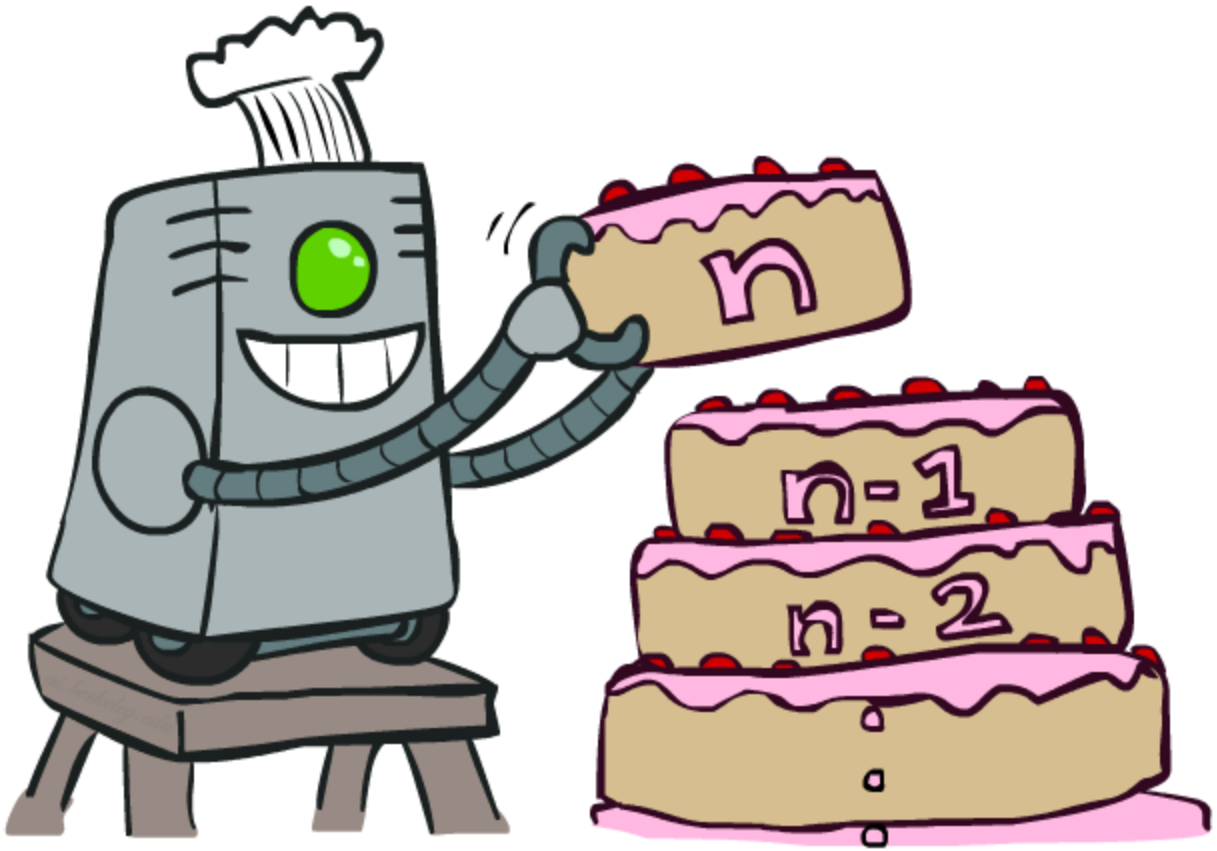
The Bellman Equations

- Definition of “optimal expected return” via recurrence gives a simple one-step lookahead relationship



- $v_*(s) = \max_a q_*(s,a)$
- $q_*(s,a) = E[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a]$
- $v_*(s) = \max_a \sum_{s',r} p(s',r|s,a)(r + \gamma v_*(s'))$
- $q_*(s,a) = \sum_{s',r} p(s',r|s,a)(r + \gamma \max_{a'} q_*(s',a'))$

Value Iteration



Value Iteration

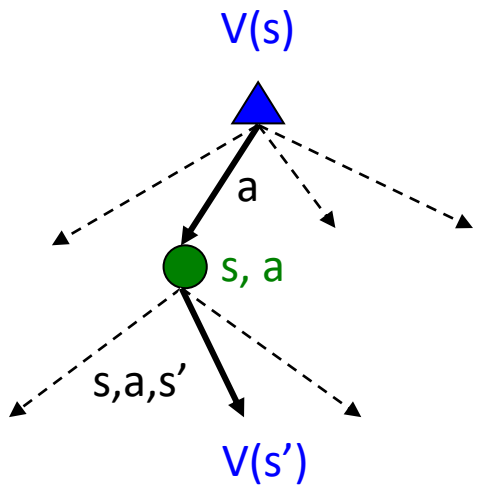
- Bellman equations characterize the optimal values:

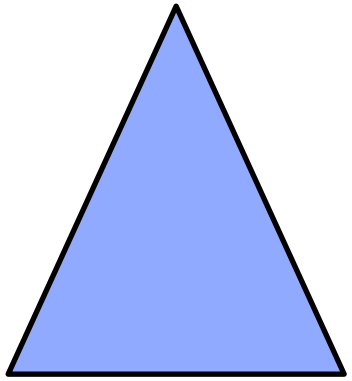
- $v_*(s) = \max_a \sum_{s',r} p(s', r|s, a)(r + \gamma v_*(s'))$

- Value iteration computes them:

- $v_{k+1}(s) = \max_a \sum_{s',r} p(s', r|s, a)(r + \gamma v_k(s'))$

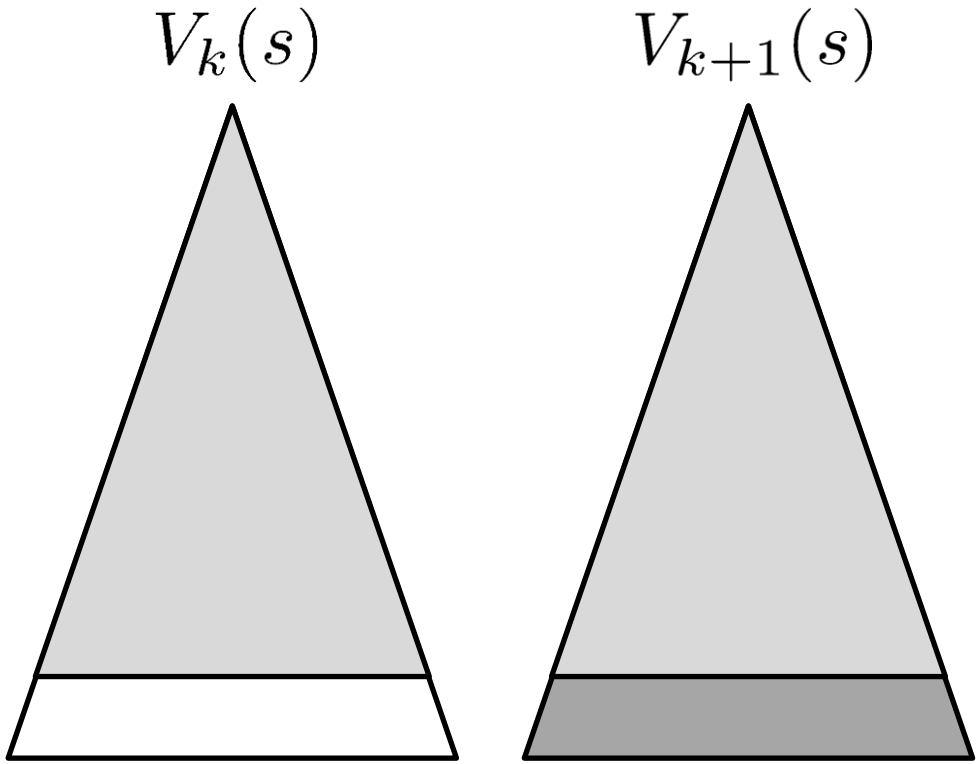
- Value iteration is just a fixed point solution method





Convergence Intuition

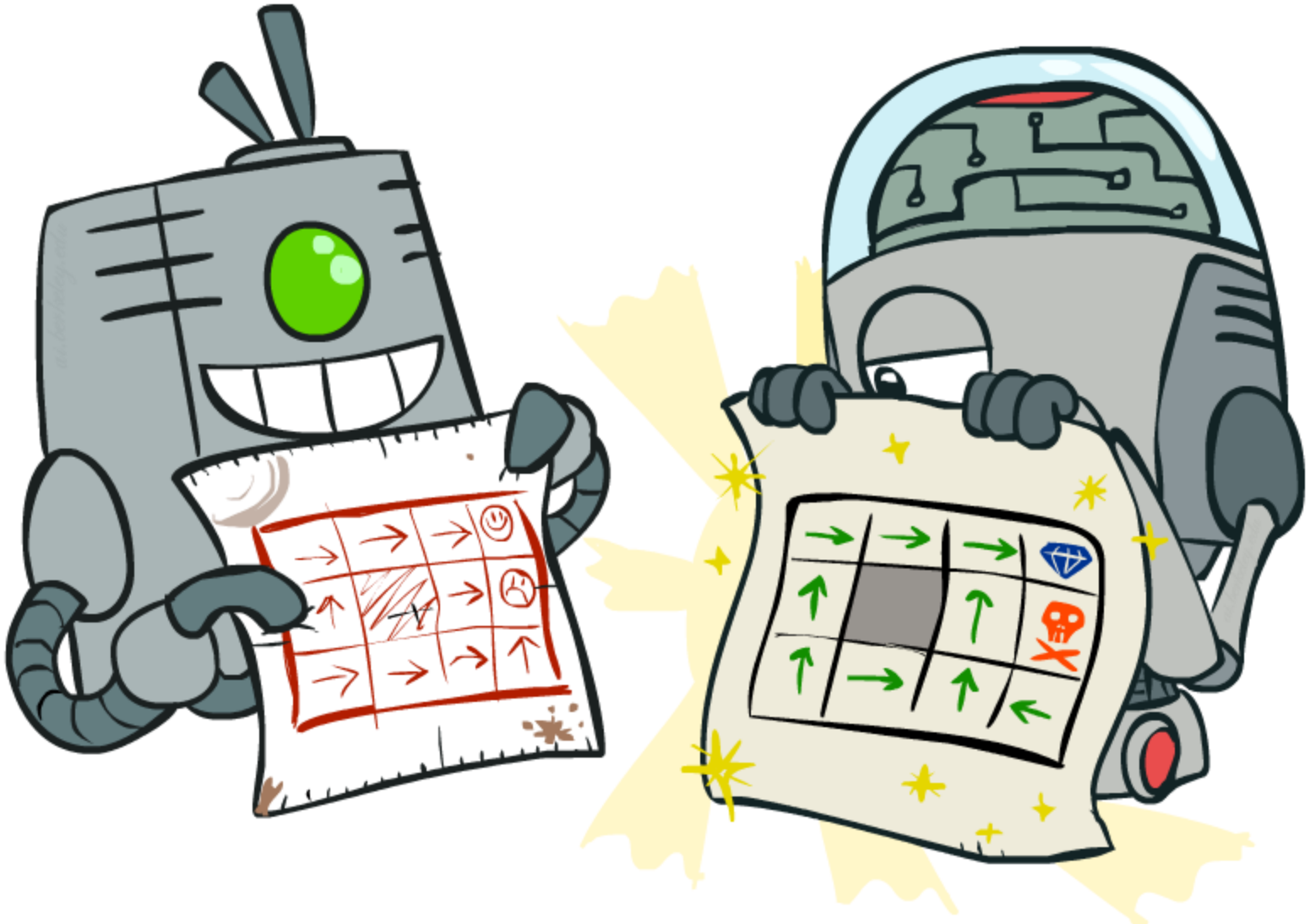
- How do we know the v_k vectors are going to converge?
- Case 1: If the tree has maximum depth T, then v_T holds the actual untruncated values
- Case 2: If the discount is less than 1
  - Sketch: For any state v_k and v_{k+1} can be viewed as depth k+1 expected returns in nearly identical settings
  - The difference is that on the bottom layer, v_{k+1} has actual rewards while v_k has zeros
  - That last layer is at best all R_{MAX}
  - It is at worst R_{MIN}
  - But everything is discounted by γ^k that far out
  - So V_k and V_{k+1} are at most γ^k max|R| different
  - So as k increases, the values converge



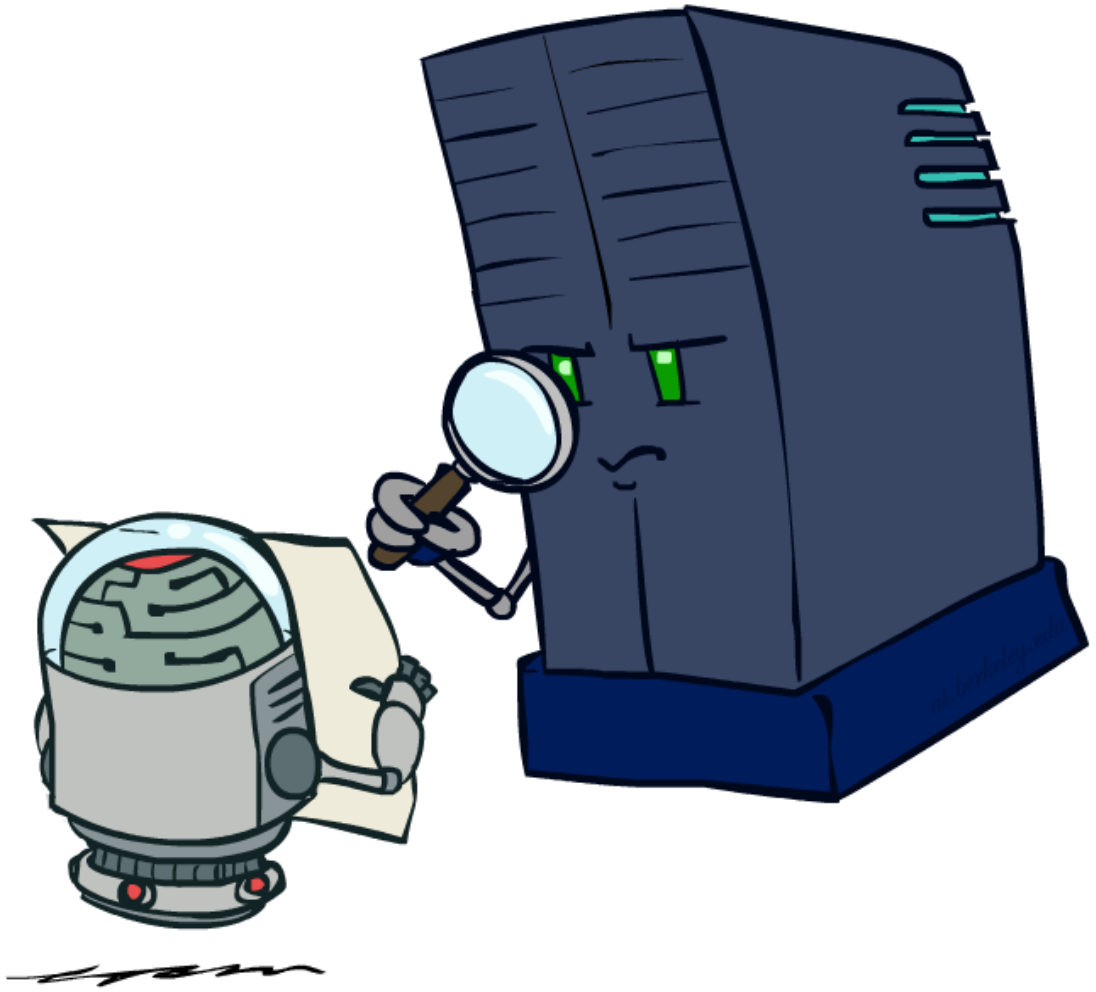
Convergence Proof for Case 2

- Let (X, d) be a complete metric space (e.g. R^k with some norm)
- $T: X \to X$ is a contraction map if $\mathrm{d}(\mathrm{T}(\mathrm{x}), \mathrm{T}(\mathrm{y})) \leq qd(x, y)$ for some $q \in [0,1)$ and all x,y
- Banach Fixed Point Theorem: a contraction map T has a unique fixed point x^*. Consider the sequence $x_{n+1} = T(x_n)$. Then $\lim_{n\to\infty} x_n = x^*$
- Apply the theorem with the value iteration operator as T, $q = \gamma$, and $d(v, v') = \left|\left|v - v'\right|\right|_\infty = \max_{s\in S} |v(s) - v'(s)|$

Policy Methods

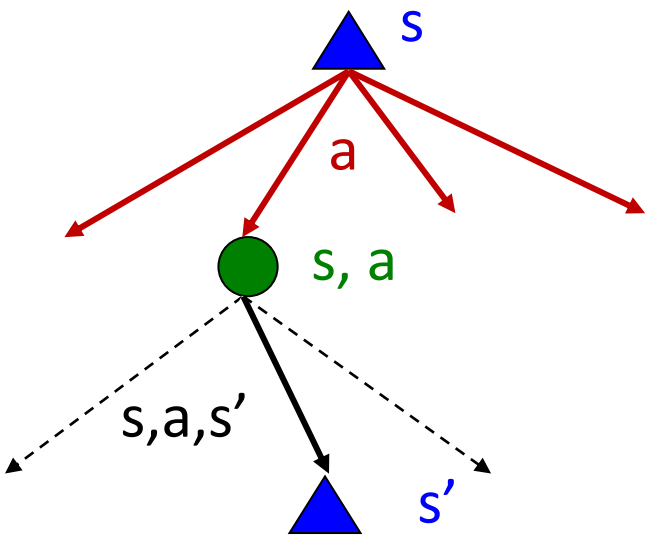


Policy Evaluation

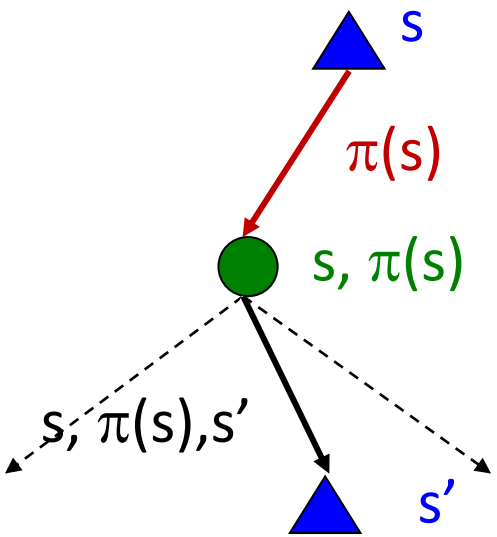


Fixed Policies

Do the optimal action

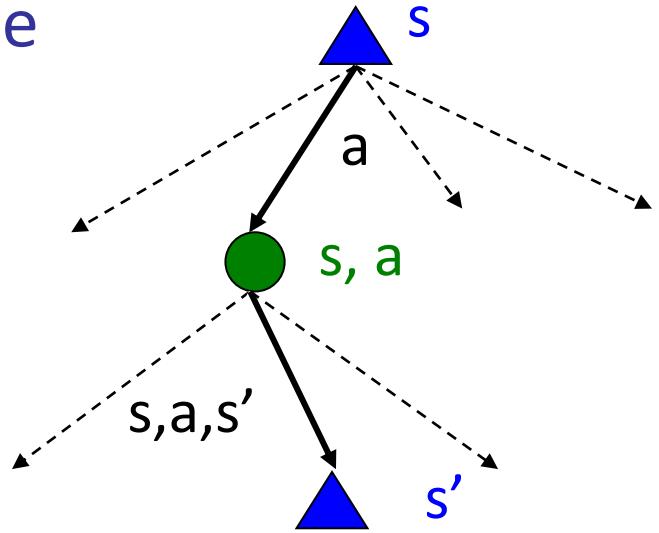


Do what π says to do



The Bellman Equations For Fixed Policies

- Definition of “optimal expected return” via recurrence gives a simple one-step lookahead relationship

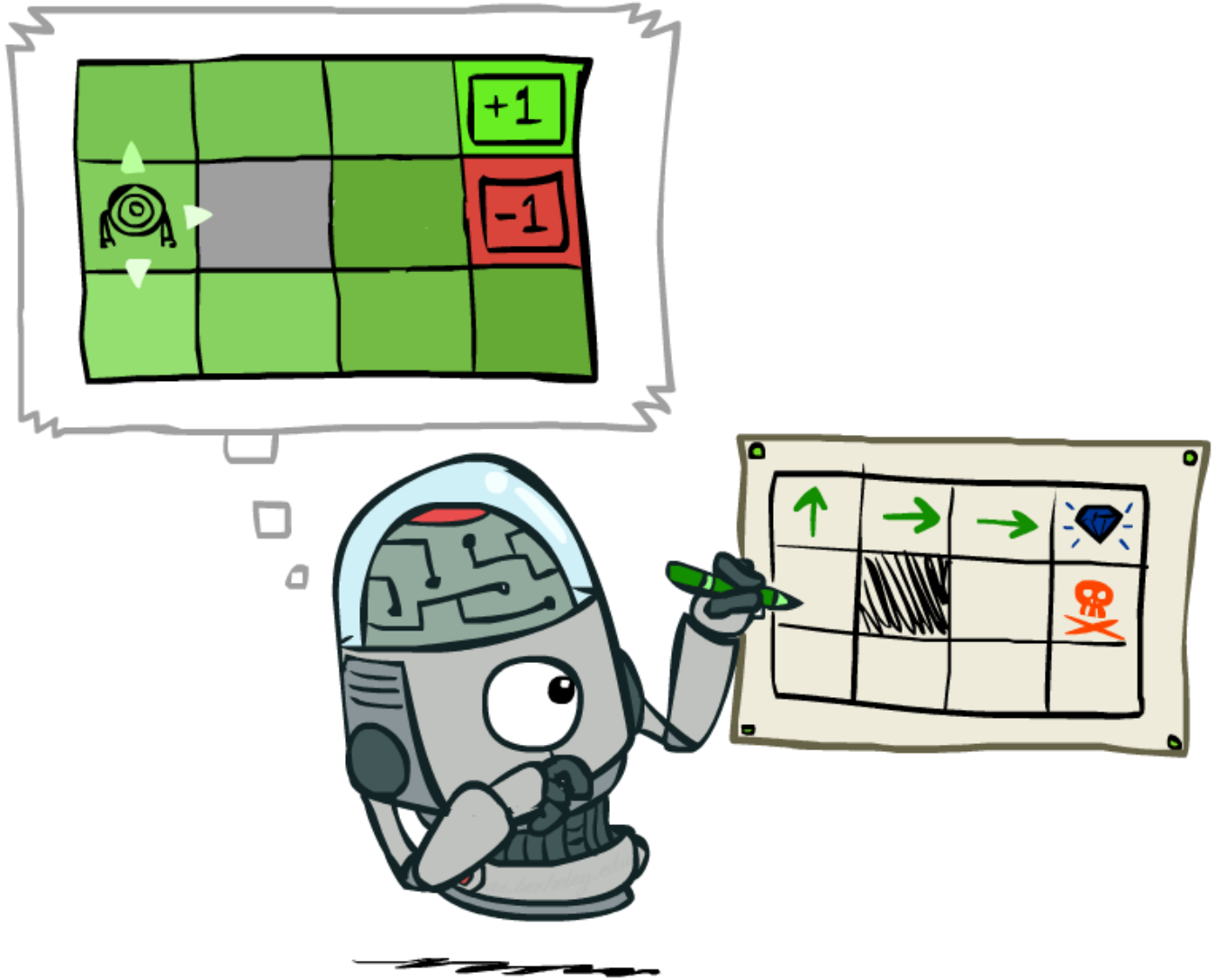


- $v_\pi(s) = E_\pi[G_t \mid S_t = s]$
- $v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)(r + \gamma v_\pi(s'))$
- $q_\pi(s, a) = E[G_t| S_t = s, A_t = a]$
- $q_\pi(s, a) = \sum_{s',r} p(s', r|s, a)(r + \gamma q_\pi(s', \pi(s')))$

Policy Evaluation

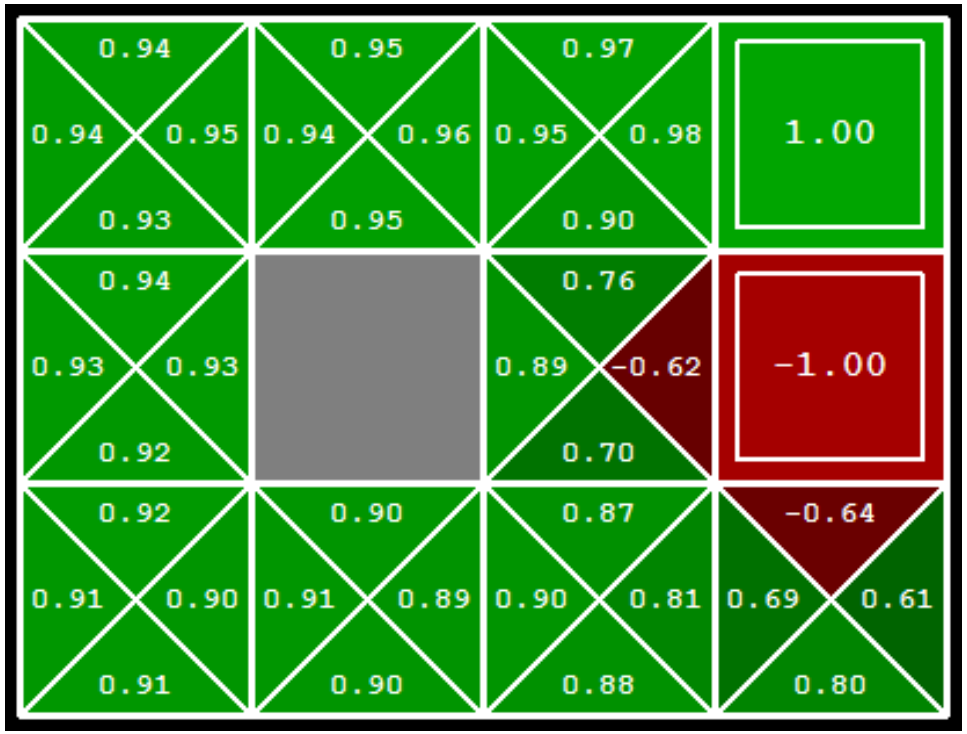
- How do we calculate the v's for a fixed policy π?
- Idea 1: Turn recursive Bellman equations into updates
  (like value iteration)
- $v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)(r + \gamma v_k(s'))$
- Idea 2: Without the maxes, the Bellman equations are just a linear system
- Solve with linear programming!

Policy Extraction



Computing Actions from Q-Values

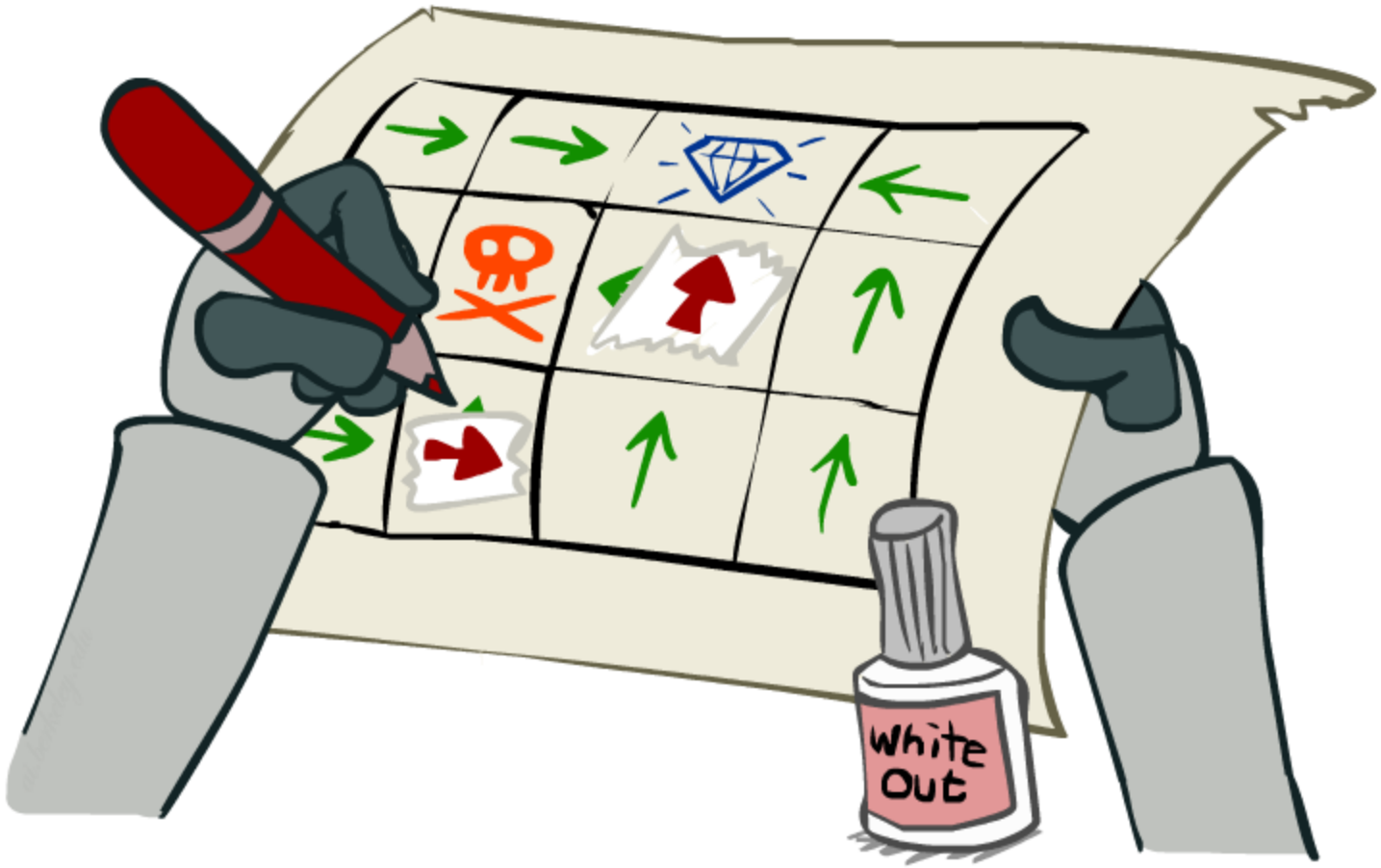
- Let's imagine we have the optimal q-values:



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

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

Policy Iteration



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 Improvement Theorem

If $q_\pi\big(s, \pi'(s)\big) \geq v_\pi(s)$ for all $s \in S$

Then

$v_{\pi'}(s) \geq v_\pi(s)$ for all $s \in S$.

Moreover, strictness in the former implies strictness in the latter.

Proof: See text.

Corollary: Policy iteration strictly improves the policy at each policy improvement step and therefore converges in a finite number of steps.

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
  - They differ only in whether we plug in a fixed policy or max over actions

- Next time: Chapters 5 and 6