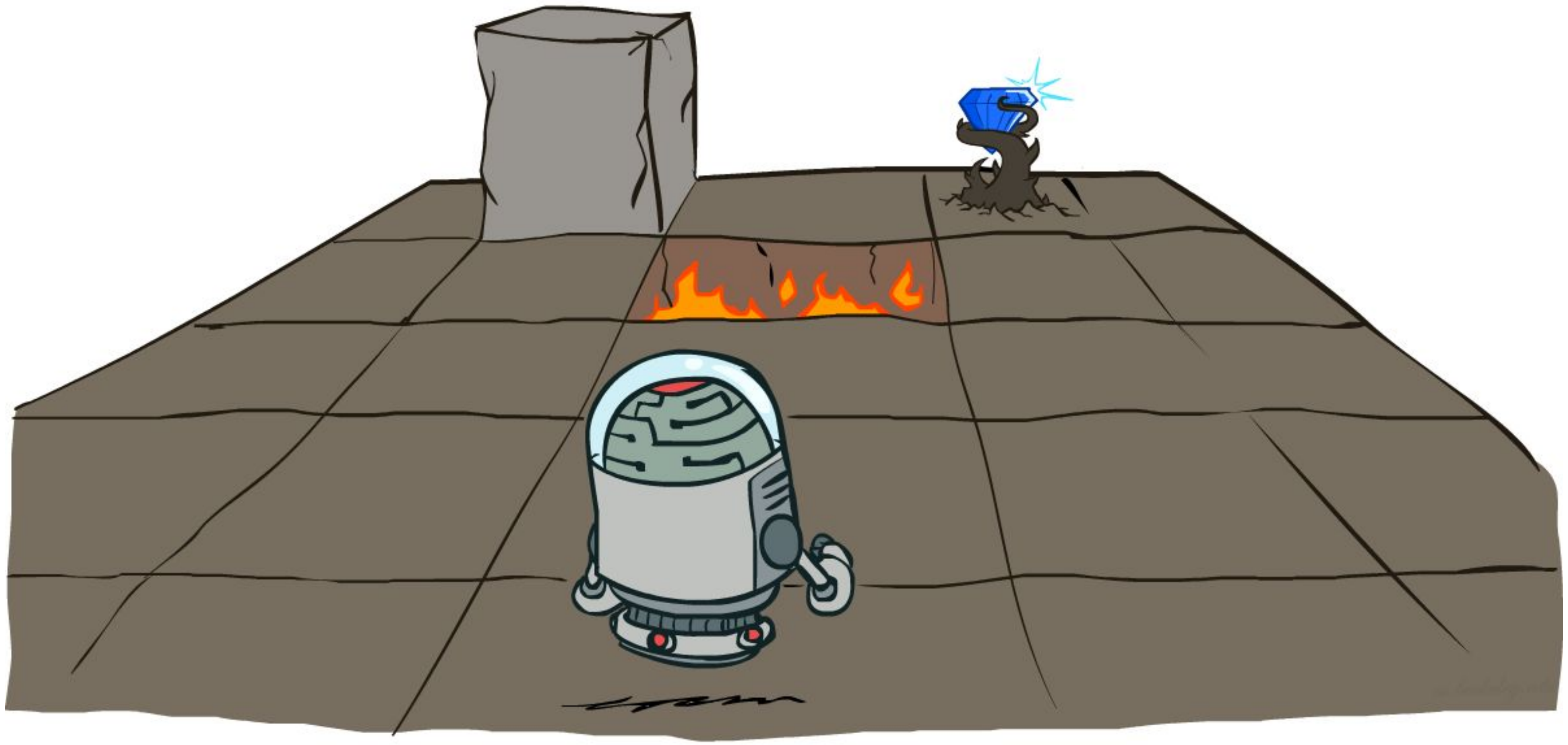
CS 188: Artificial Intelligence

MDP II: Value/Policy Iteration



Instructor: Stuart Russell and Dawn Song

University of California, Berkeley

Recap: Optimal Quantities

- The value (expected utility) of π in s_0 is written $U^{\pi}(s_0)$
  - It's the sum over all possible state sequences of (discounted sum of rewards) x (probability of state sequence)

$U^{\pi}(s_0)$ = $E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t, \pi(S_t), S_{t+1})\right]$

- The optimal policy:
  - $\pi^*(s)$ = optimal action from state s
  - Gives highest $U^{\pi}(s)$ for any π

- The value (utility) of a state s:
  - $U^*(s) = U^{\pi^*}(s)$ = expected utility starting in s and acting optimally

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

  $Q^*(s,a)$ = expected utility of taking action a in state s and (thereafter) acting optimally
  - $U^*(s) = \max_a Q^*(s,a)$

Recap: Bellman equations (Shapley, 1953)

- The value/utility of a state is
  - The expected reward for the next transition plus the discounted value/utility of the next state, assuming the agent chooses the optimal action

- Hence we have a recursive definition of value (Bellman equation):

$$U(s) = \max_{a \in A(s)} \sum_{s'} P(s' \mid s,a)[R(s,a,s') + \gamma U(s')]$$

- Similarly, Bellman equation for Q-functions

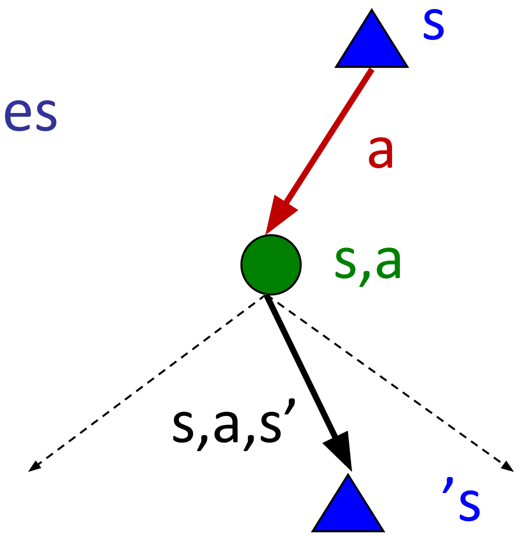
$$\begin{aligned} Q(s,a) &= \sum_{s'} P(s' \mid s,a)[R(s,a,s') + \gamma U(s')] \\ &= \sum_{s'} P(s' \mid s,a)[R(s,a,s') + \gamma \max_{a'} Q(s',a')] \end{aligned}$$

Recap: Value Iteration

- Start with (say) $U_0(s) = 0$ and some termination parameter ε
- Repeat until convergence (i.e., until all updates smaller than ε)
  - Do a ***Bellman update*** (essentially one ply of expectimax) from each state:

$$U_{k+1}(s) \leftarrow \max_a \sum_{s'} P(s' \mid a,s)\,[R(s,a,s') + \gamma U_k(s')\,]$$

$$\boldsymbol{U} \leftarrow \mathrm{B}\boldsymbol{U}$$



- Theorem: will converge to unique optimal values

How do we know it will converge?*

- New concept: ***contraction***
  - If some operator F is a contraction by a factor, it brings any pair of objects ***closer*** to each other (according to some metric d(,))
    - For any x, y $d(Fx,Fy) \leq c\, d(x,y)$ where $c < 1$

      E.g., $Fx = x/2$
  - If F is a contraction it has a unique fixed point z (i.e., $Fz=z$)
- Reminder: Value iteration is just $\boldsymbol{U}_{k+1} \leftarrow BU_k$
- **The Bellman update B is a contraction by γ**
  - Metric is the ***max norm***: $\|\boldsymbol{V} - \boldsymbol{W}\| = \max_s |V(s) - W(s)|$
  - Proof: follows from definition of B, i.e., Bellman equation
- What's the fixed point for B?
  - $BU^* = U^*$

How fast does VI converge?

- Look at what happens to the distance between $\boldsymbol{U}_k$ and $\boldsymbol{U}^*$

$$|| \quad \boldsymbol{U}_{k+1} - \boldsymbol{U}^* || \quad ? \quad || \quad \boldsymbol{U}_k - \boldsymbol{U}^* ||$$

How fast does VI converge?

- Look at what happens to the distance between U_k and U^*

$$
\begin{aligned}
& \| U_{k+1} - U^* \| \\
&= \| BU_k - U^* \| && \text{(definition of } U_{k+1} \text{ from VI update)} \\
&= \| BU_k - BU^* \| && (U^* \text{ is the fixed point of } B) \\
&\le \gamma \| U_k - U^* \| && (B \text{ is a contraction by } \gamma)
\end{aligned}
$$

- I.e., the **error is reduced by at least a factor γ on every iteration**
  - Exponentially fast convergence!
  - E.g., if $\gamma=0.9$, 22 iterations reduces error by 10
    - 44 iterations reduces error by 100
    - 220 iterations reduces error by 10^{10}

How do we know the answer is (nearly) right?

- VI doesn't usually converge exactly; stops when change < $\epsilon(1-\gamma)/\gamma$
- I.e., $\| U_{k+1} - U_k \| < \epsilon(1-\gamma)/\gamma$
- What about $\| U_{k+1} - U^* \|$ when $\| U_{k+1} - U_k \| < \epsilon(1-\gamma)/\gamma$?
- We need some connection between $\| U_{k+1} - U_k \|$ and $\| U_{k+1} - U^* \|$
- Useful properties:
  - $\| U_{k+1} - U^* \| \leq \gamma \| U_k - U^* \|$
  - Triangle inequality!

$$\| U_k - U^* \| \leq \| U_{k+1} - U_k \| + \| U_{k+1} - U^* \|$$

How do we know the answer is (nearly) right?

- VI doesn't usually converge exactly; stops when change < $\varepsilon(1-\gamma)/\gamma$
- I.e., $\| U_{k+1} - U_k \| < \varepsilon(1-\gamma)/\gamma$
- What about $\| U_{k+1} - U^* \|$ when $\| U_{k+1} - U_k \| < \varepsilon(1-\gamma)/\gamma$?
- We need some connection between $\| U_{k+1} - U_k \|$ and $\| U_{k+1} - U^* \|$
  - Triangle inequality!

B is a contraction by γ

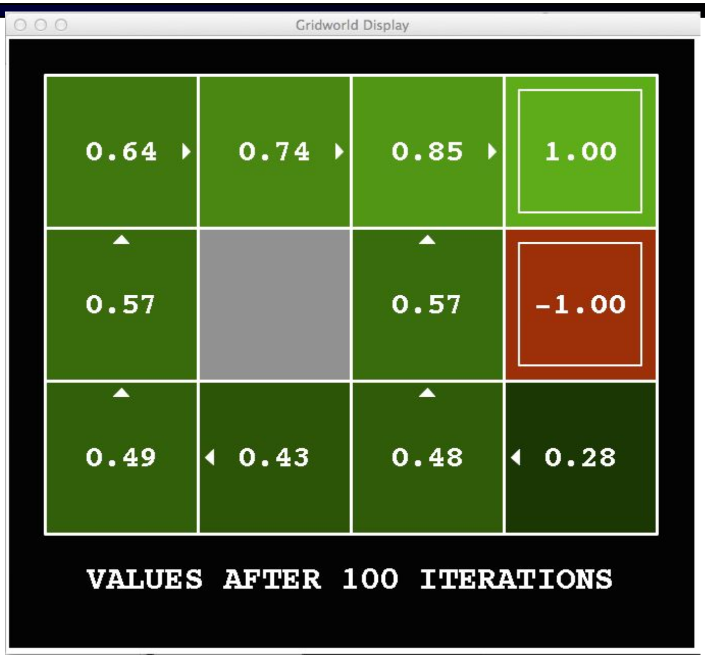
$$\| U_k - U^* \| \leq \| U_{k+1} - U_k \| + \| U_{k+1} - U^* \|$$

  - $1/\gamma \, \| U_{k+1} - U^* \| \leq \| U_{k+1} - U_k \| + \| U \ldots U^* \|$

Assume we have stopped

  - $(1/\gamma - 1) \, \| U_{k+1} - U^* \| \leq \| U_{k+1} - U_k \|$
  - $(1/\gamma - 1) \, \| U_{k+1} - U^* \| < \varepsilon(1-\gamma)/\gamma$
  - $\| U_{k+1} - U^* \| < \varepsilon$
- I.e., when we stop, the max-norm error in U_{k+1} is less than ε

Wait! The agent needs a policy, not a value function!



- How should the agent act given ***U*** (s)?
- Maximize expected utility! (as if ***U*** is correct)

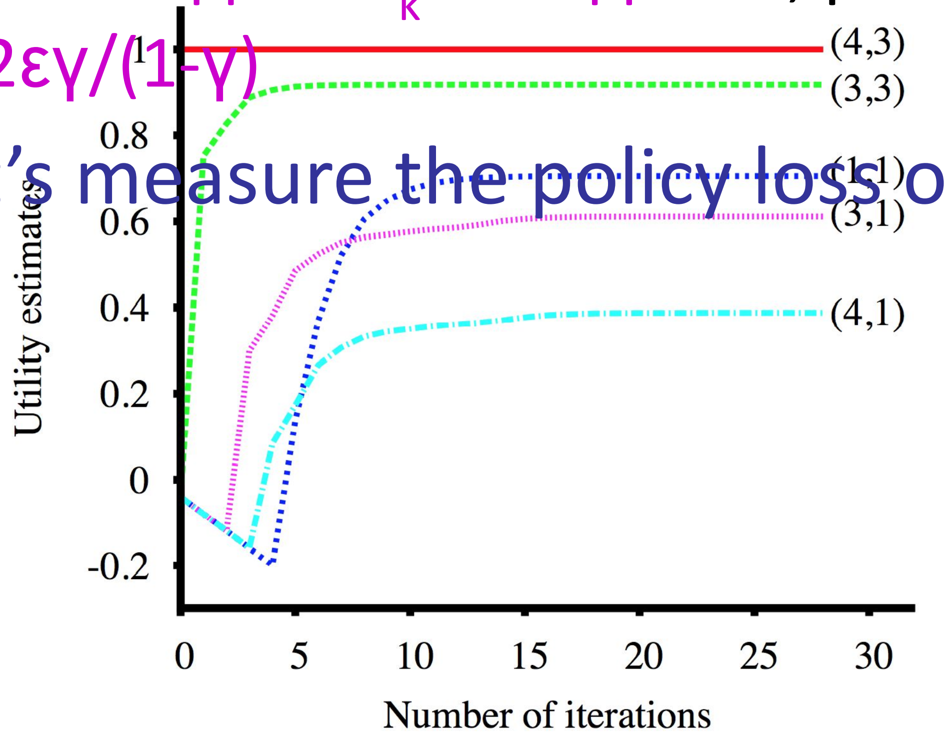
- I.e., do a mini-expectimax (greedy one-step):

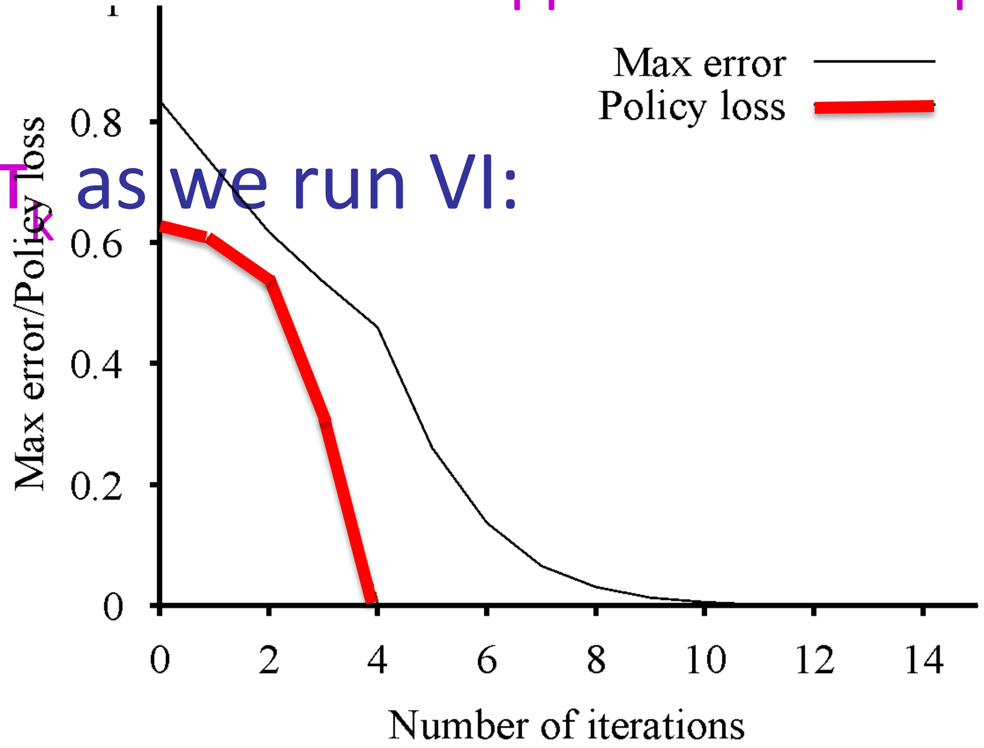
$$\pi_U(s) = \text{argmax}_a \sum_{s'} P(s' \mid a,s)\,[R(s,a,s') + \gamma U(s')]$$

- This is called ***policy extraction***, since it finds the policy π_U implied by the values ***U***

How good is the policy extracted from VI?

- The quality of a policy π is measured by the ***policy loss*** $|| U^{\pi} - U^* ||$
- Let $\pi_k = \pi_{U_k}$ i.e. the implied policy at step k; in case you were worried:
  - When $|| U_k - U^* || \le \varepsilon$, policy loss is bounded: $|| U^{\pi_k} - U^* || \le 2\varepsilon\gamma/(1-\gamma)$
- Let's measure the policy loss of π_k as we run VI:

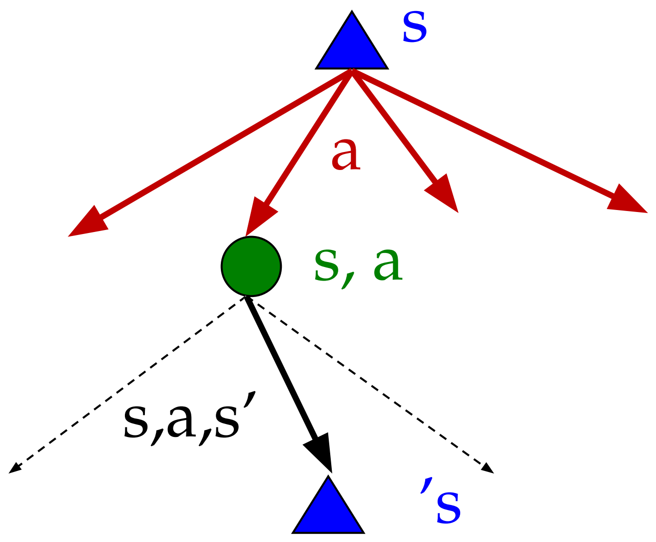




Problems with Value Iteration

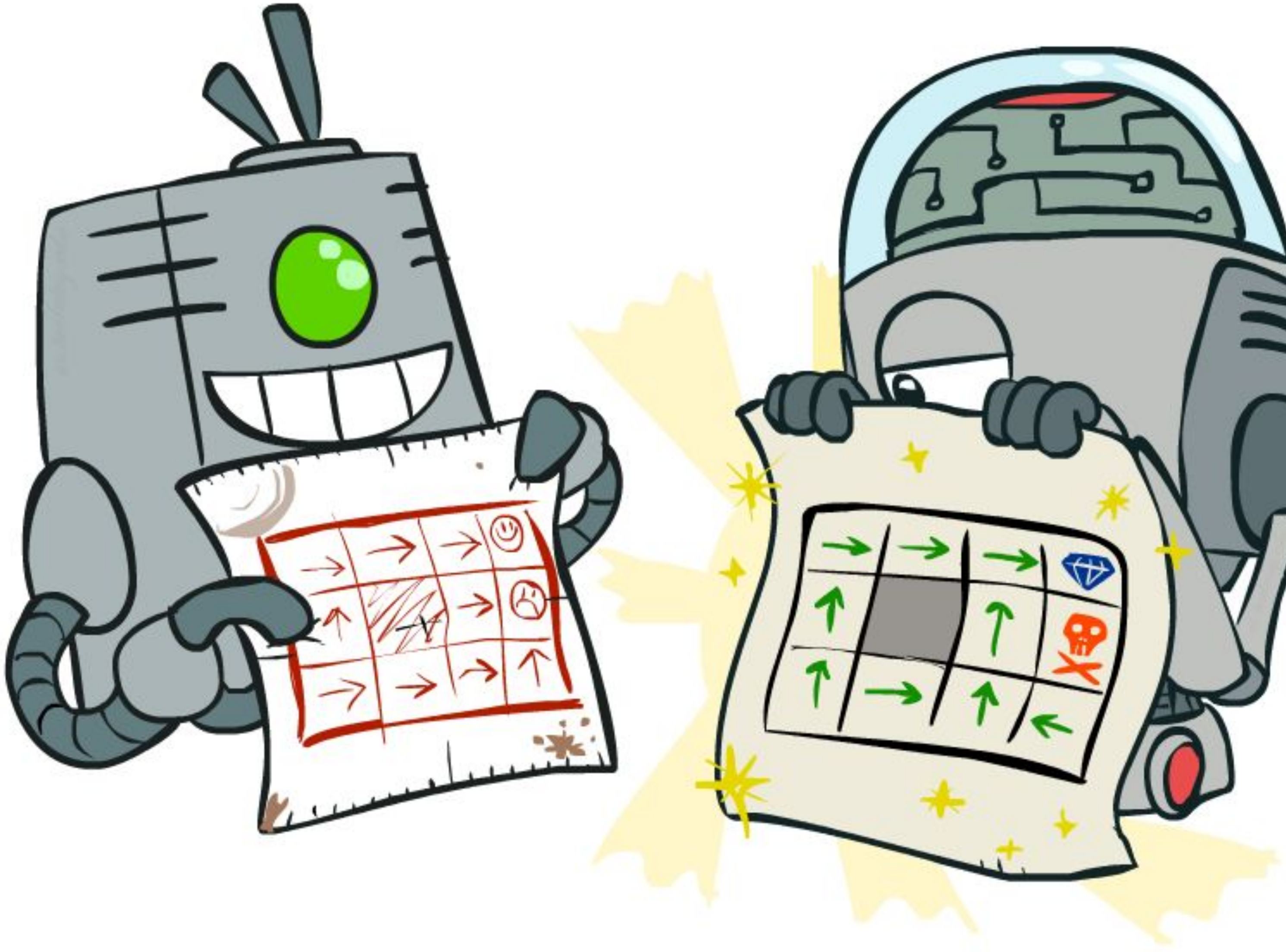
- Value iteration repeats the Bellman updates:

$$U_{k+1}(s) \leftarrow \max_a \sum_{s'} P(s' \mid a,s) \left[R(s,a,s') + \gamma U_k(s')\right]$$

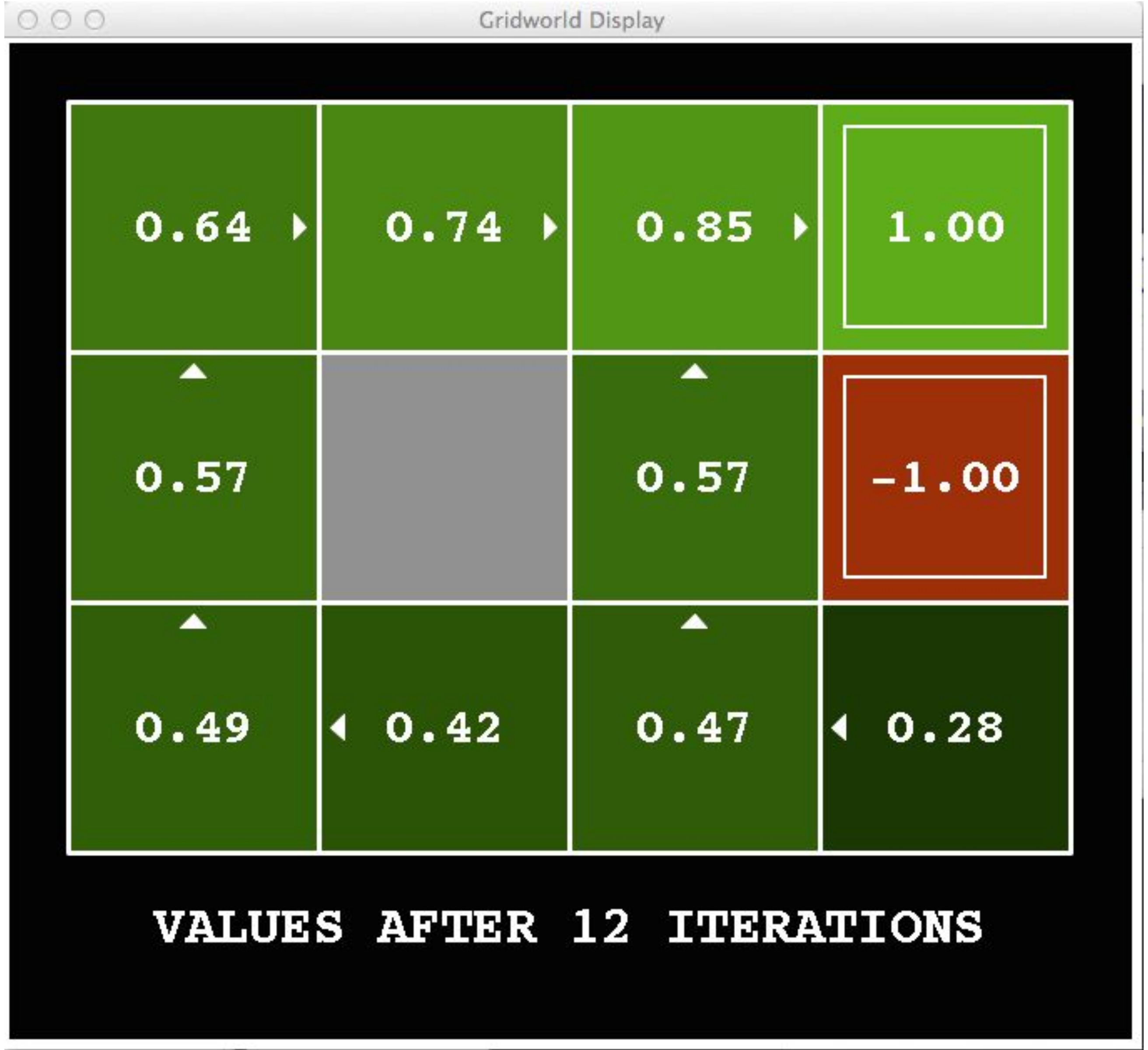


- Problem 1: It's slow – $O(S^2A)$ per iteration
- Problem 2: The "max" at each state rarely changes
- Problem 3: The policy often converges long before the values

Policy Iteration

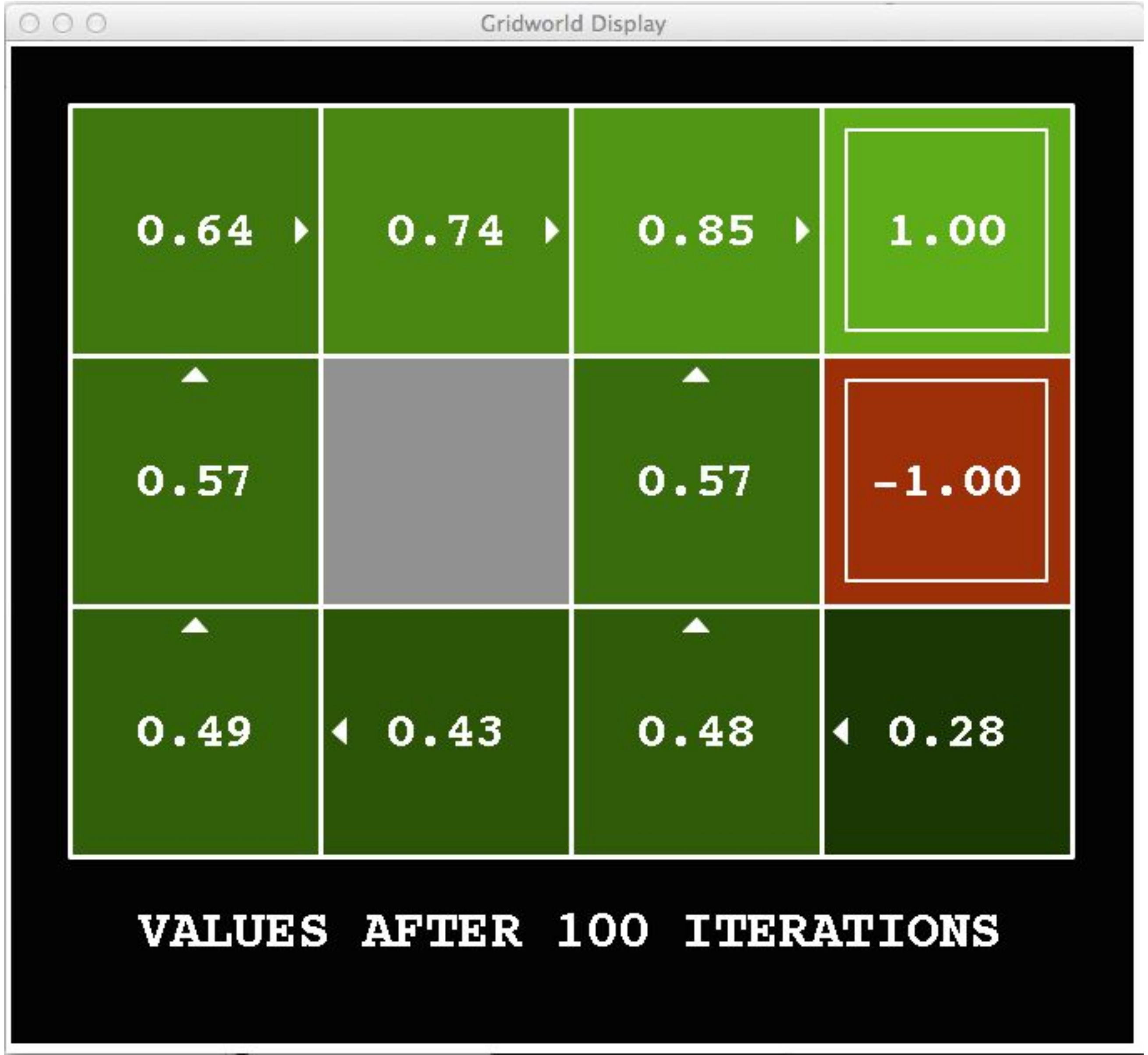


k=12



Noise = 0.2
Discount = 0.9
Living reward = 0

k=100

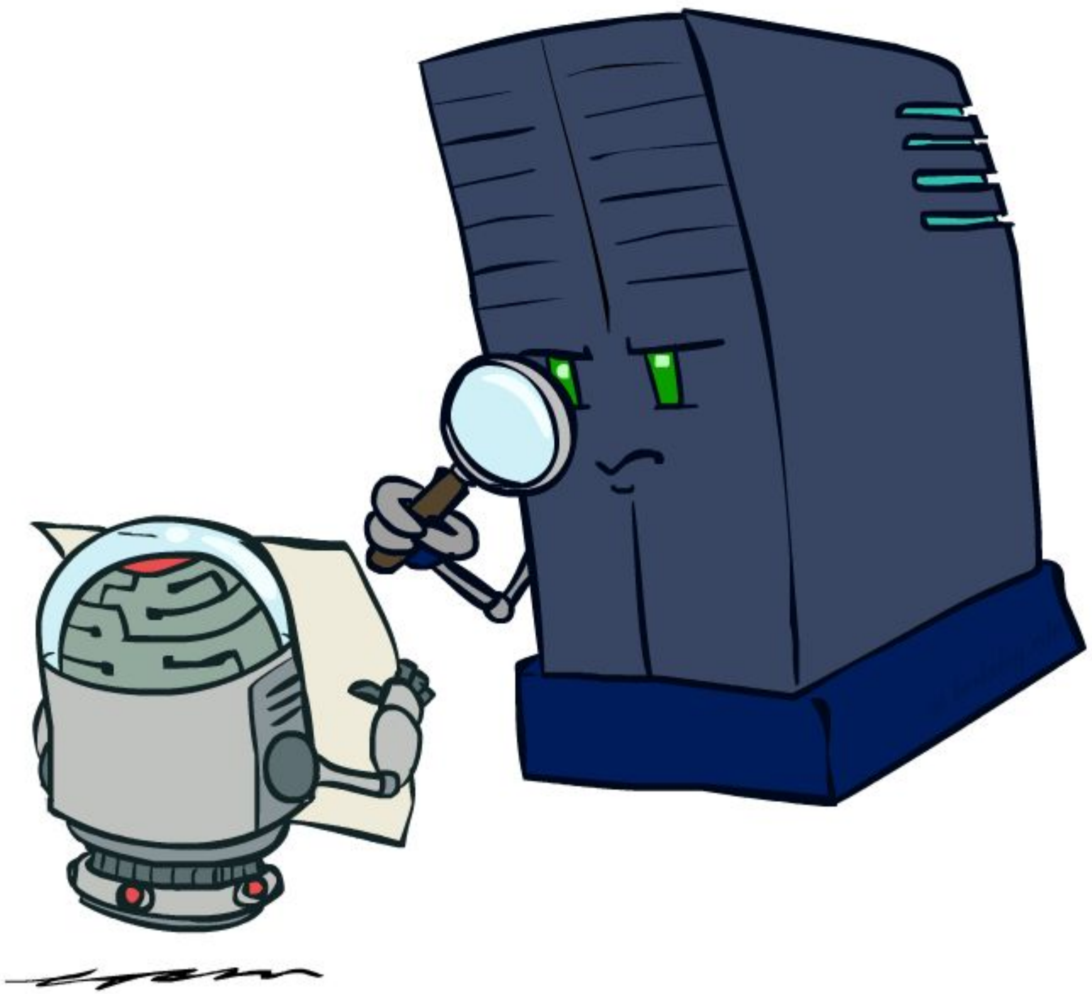


Noise = 0.2
Discount = 0.9
Living reward = 0

Policy Iteration

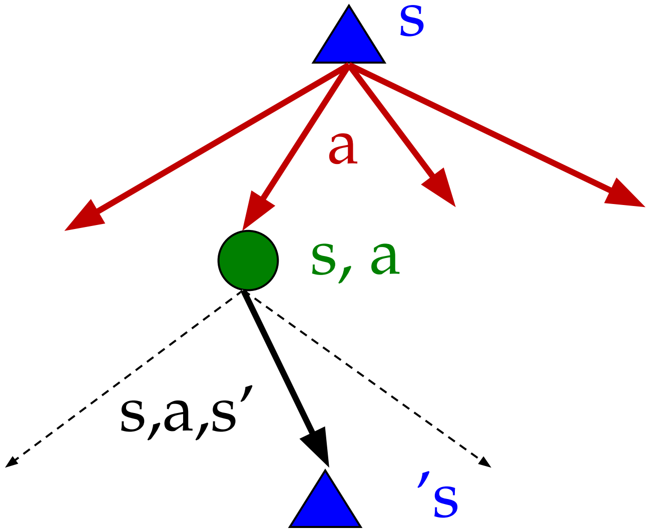
- Basic idea: make the implied policy in U explicit, compute its ***long-term*** implications for value
- Repeat until no change in policy:
  - Step 1: Policy evaluation: calculate value U^{π_k} for current policy π_k
  - Step 2: Policy improvement: extract the new implied policy π_{k+1} from U^{π_k}
- It's still optimal!
- Can converge (much) faster under some conditions

Policy Evaluation

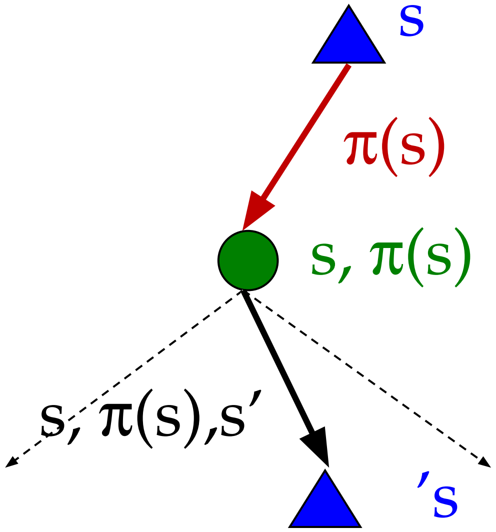


Fixed Policies

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

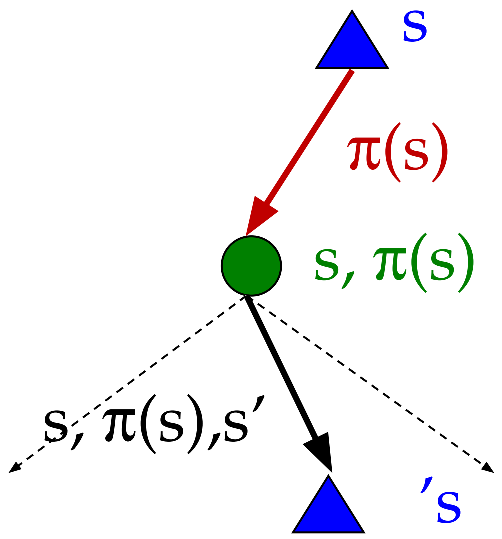
Utilities for a Fixed Policy

- 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:

  $U^{\pi}(s)$ = expected total discounted rewards starting in s and following π

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



$$U_i(s) = \sum_{s'} P(s' \mid s, \pi_i(s))[R(s, \pi_i(s), s') + \gamma U_i(s')]$$

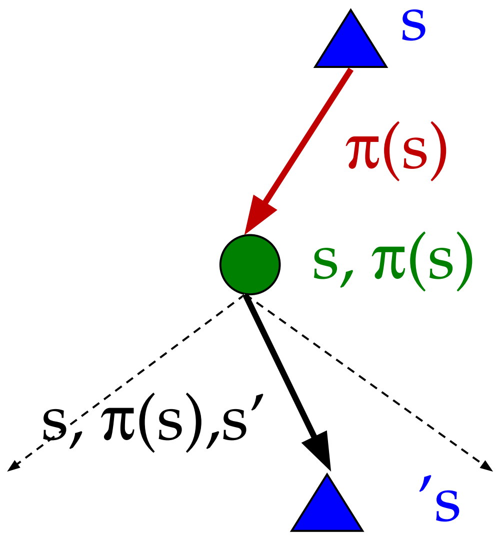
Policy Evaluation

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

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

  $U_0^{\pi}(s) = 0$

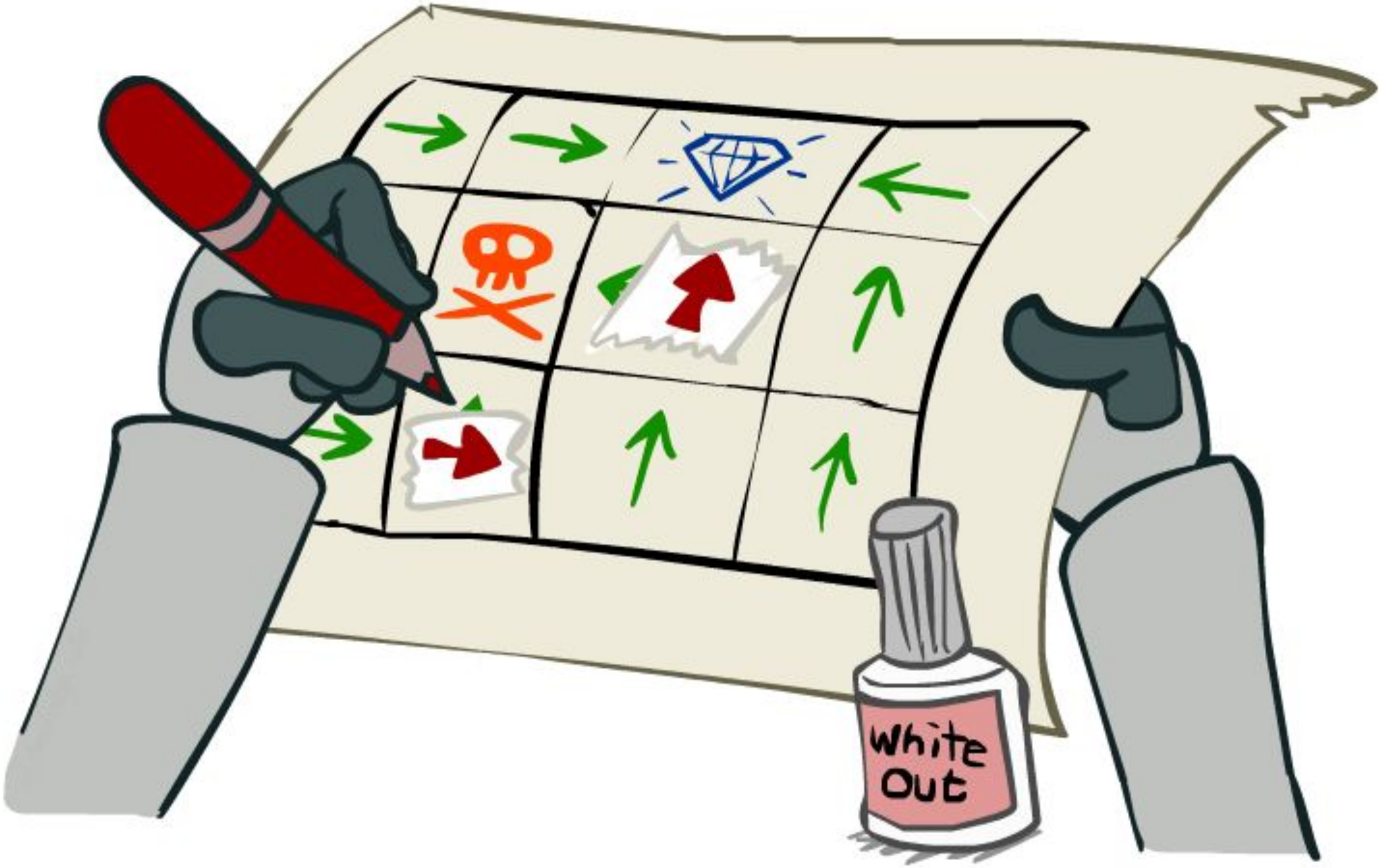
$$U_{i+1}(s) \leftarrow \sum_{s'} P(s' \mid s, \pi_i(s))[R(s, \pi_i(s), s') + \gamma U_i(s')]$$



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

- Idea 2: Without the maxes, the Bellman equations are just a linear system
  - Solve with Matlab (or your favorite linear system solver)

Policy Iteration



Policy Iteration

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

$$U_{i+1}(s) \leftarrow \sum_{s'} P(s' \mid s, \pi_i(s))[R(s, \pi_i(s), s') + \gamma U_i(s')]$$

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

$$\pi_{i+1}(s) = \arg\max_a \sum_{s'} P(s' \mid s,a) \left[R(s,a,s') + \gamma\, U^{\pi_i}(s')\right]$$

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)
    - Policy evaluation reveals long-term effects of policy, unlike local value updates
  - After the policy is evaluated (looking at those long-term effects), 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
- In fact, any fair sequence of value and/or policy updates on any states will converge to an optimal solution!

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

The Bellman Equations