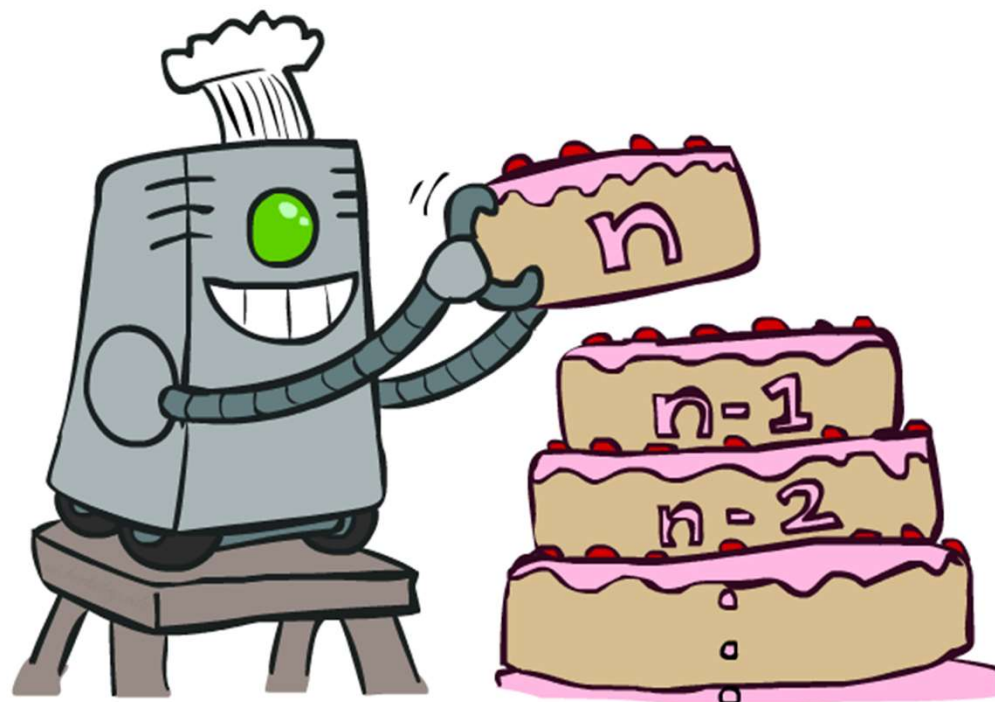# Value Iteration
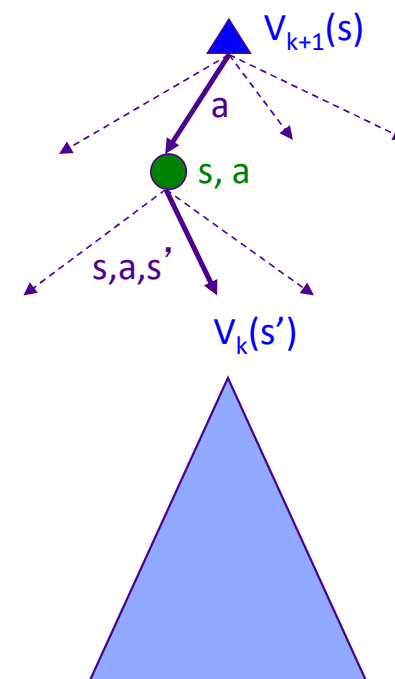
# Value Iteration

- 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:

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

V_{k+1}(s)

a

s, a

s,a,s'

V_k(s')

- Repeat until convergence

- Complexity of each iteration: $O(S^2 A)$

- 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



$$V_2 \quad \boxed{\quad 3.5 \qquad 2.5 \qquad 0 \quad}$$

$$V_1 \quad \boxed{\quad 2 \qquad 1 \qquad 0 \quad}$$

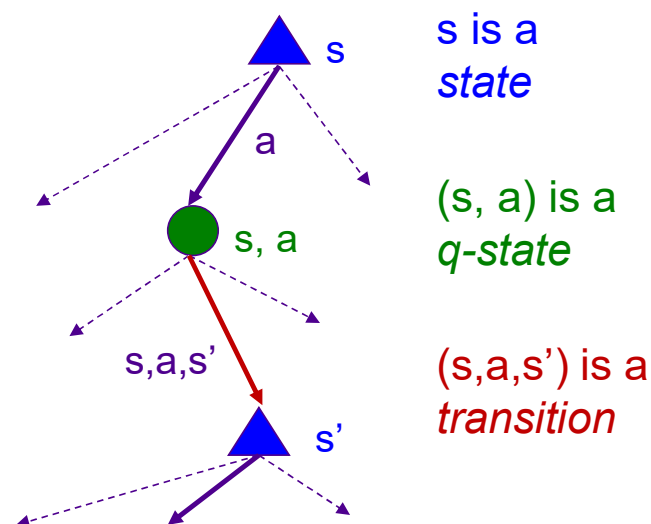$$V_0 \quad \boxed{\quad 0 \qquad 0 \qquad 0 \quad}$$
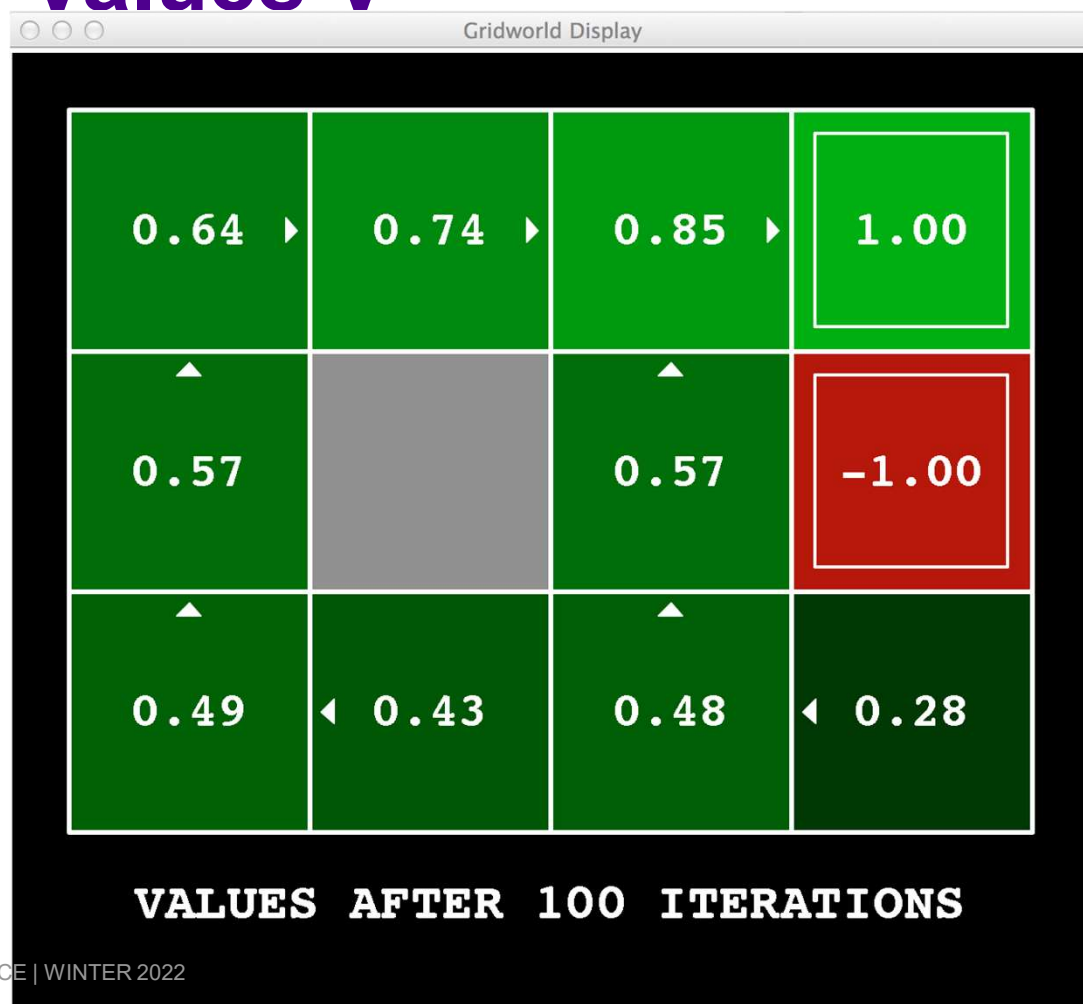
*Assume no discount!*

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

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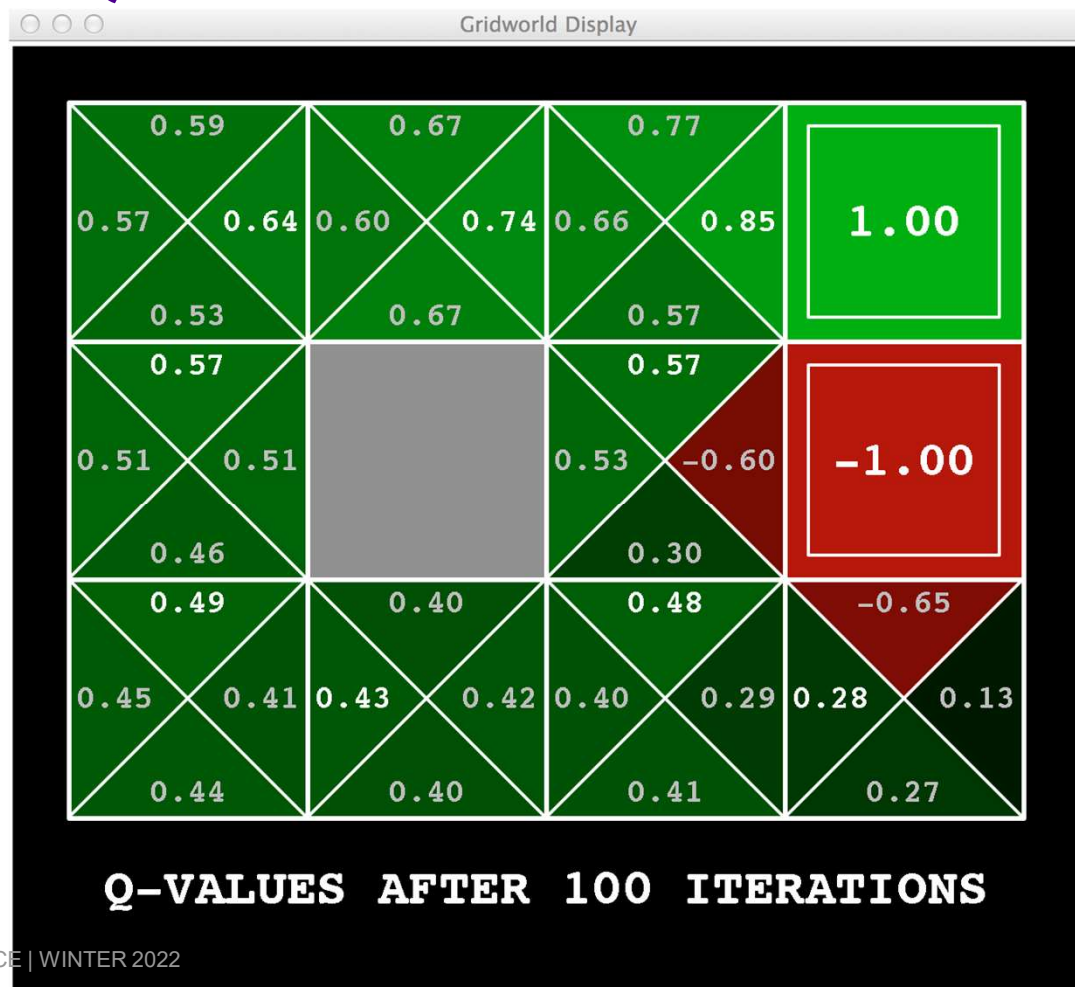
- The value (utility) of a state $s$:
    $V^*(s)$ = expected utility starting in $s$ and acting optimally

- 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*

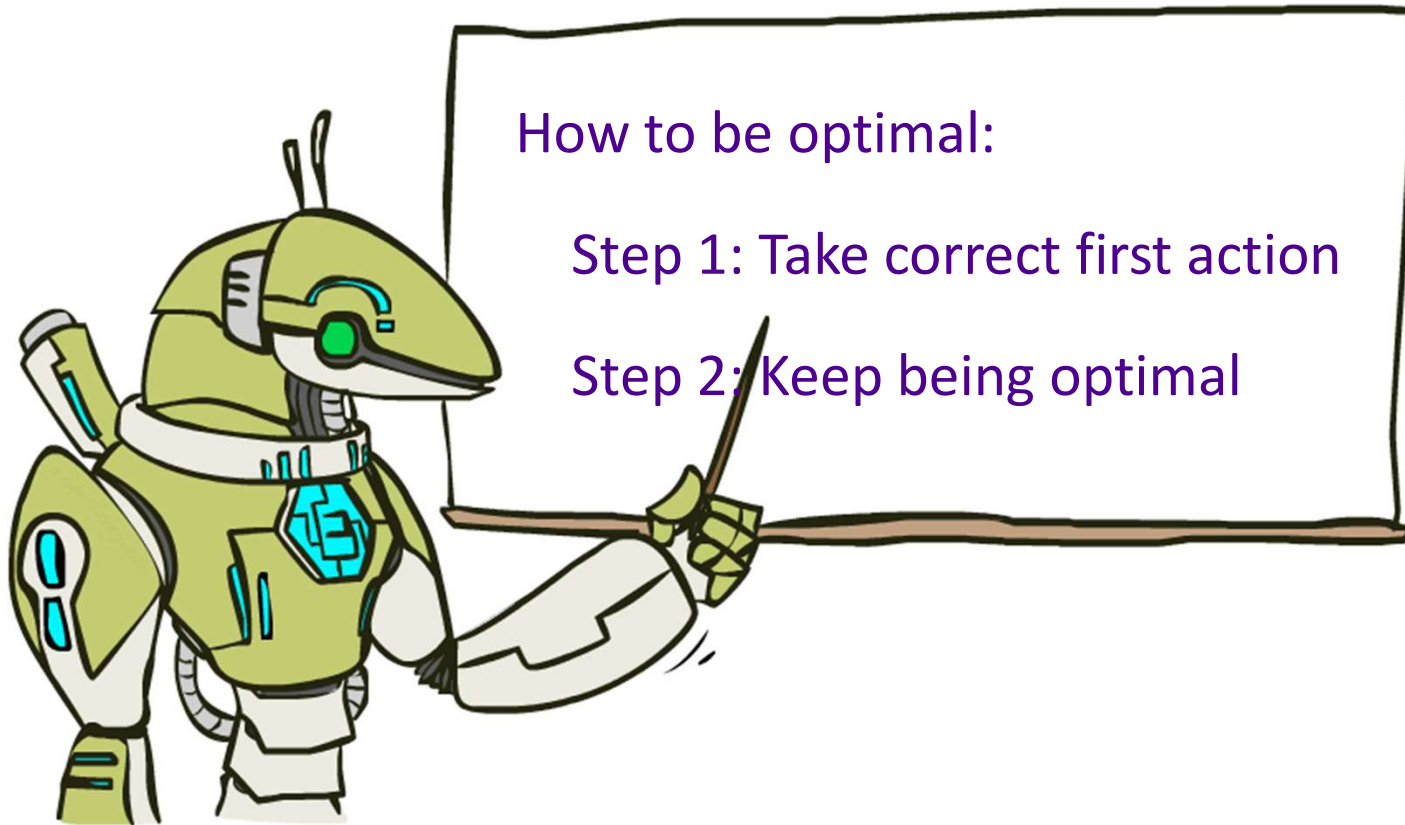(s, a) is a
*q-state*

s, a

s,a,s'

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

s'

[Demo:  gridworld values (L9D1)]

# Gridworld Values V*

# Gridworld: Q*

# The Bellman Equations

How to be optimal:

Step 1: Take correct first action

Step 2: Keep being optimal
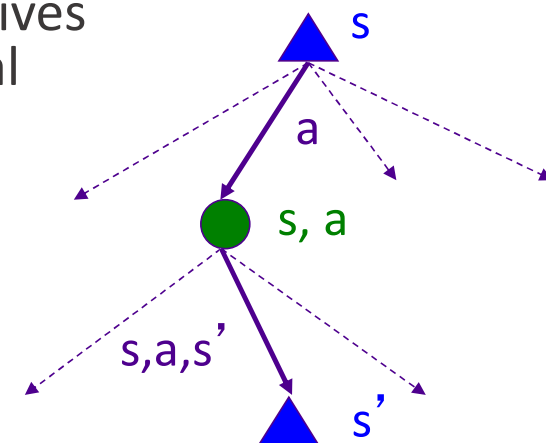
# The Bellman Equations

- Definition of "optimal utility" via expectimax recurrence gives a simple one-step lookahead relationship amongst optimal utility values

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

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

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

- These are the Bellman equations, and they characterize optimal values in a way we'll use over and over
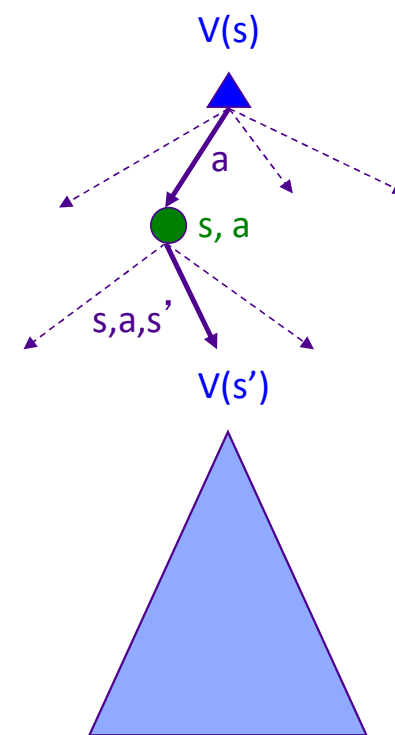
# Value Iteration

- 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

V(s)

a

s, a

s,a,s'

V(s')

# Convergence*

- How do we know the $V_k$ vectors are going to converge?

- Case 1: If the tree has maximum depth M, then $V_M$ holds the actual untruncated values

- Case 2: If the discount is less than 1
  - Sketch: For any state $V_k$ and $V_{k+1}$ can be viewed as depth k+1 expectimax results in nearly identical search trees
  - 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 $\gamma^k$ that far out
  - So $V_k$ and $V_{k+1}$ are at most $\gamma^k \max|R|$ different
  - So as k increases, the values converge

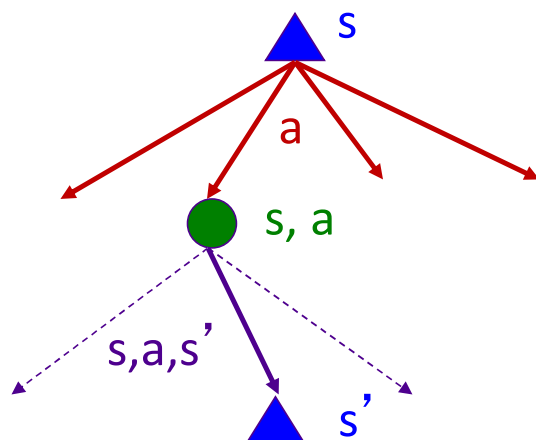$$V_k(s) \qquad V_{k+1}(s)$$
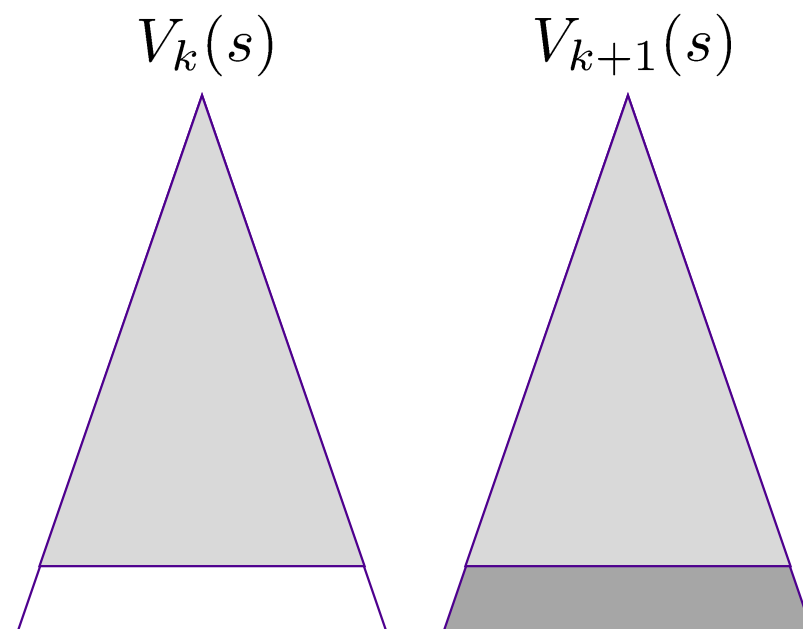
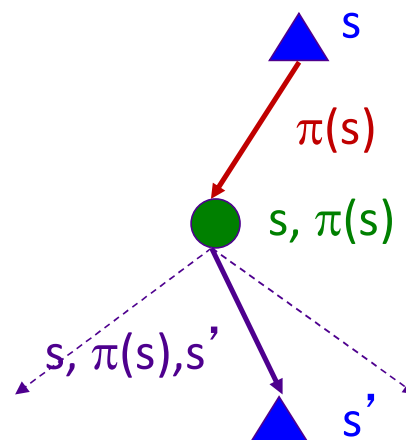# Policy Methods
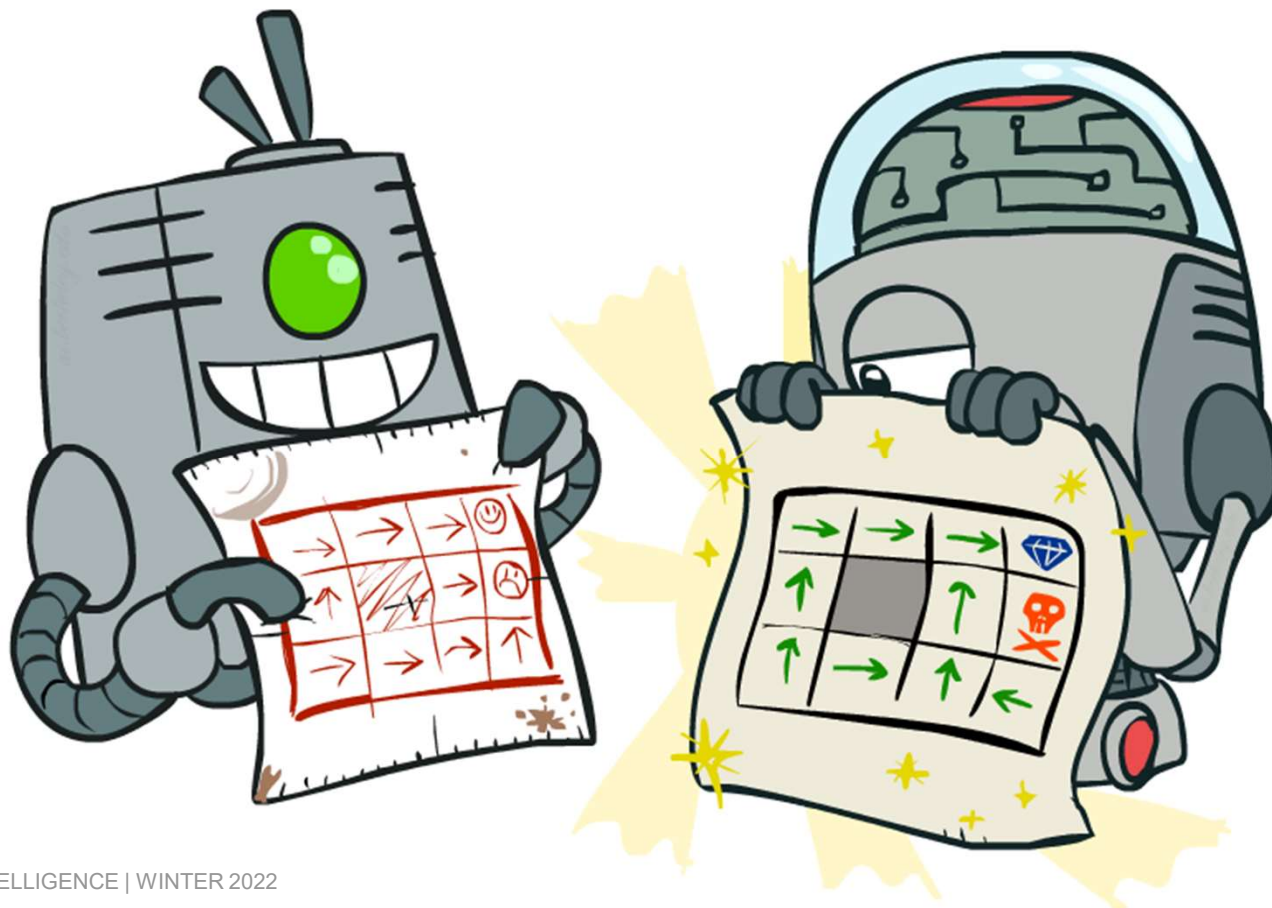
# Policy Evaluation
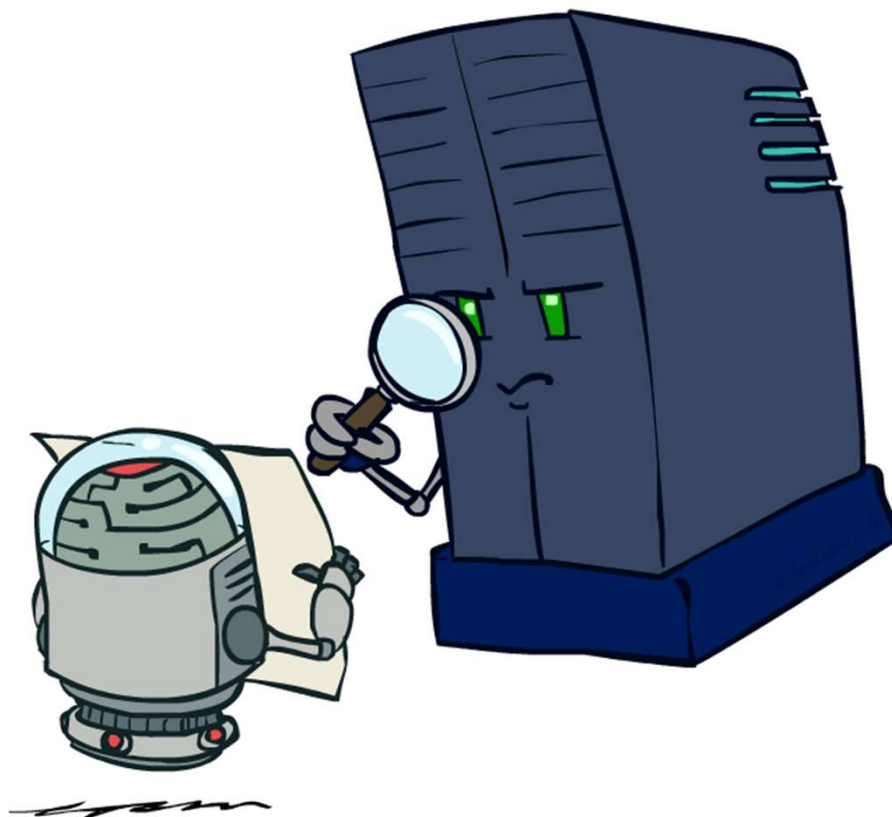
# Fixed Policies

Do the optimal action

Do what $\pi$ says to do



- Expectimax trees max over all actions to compute the optimal values

- If we fixed some policy $\pi(s)$, then the tree would be simpler – only one action per state
  - … though the tree's value would depend on which policy we fixed
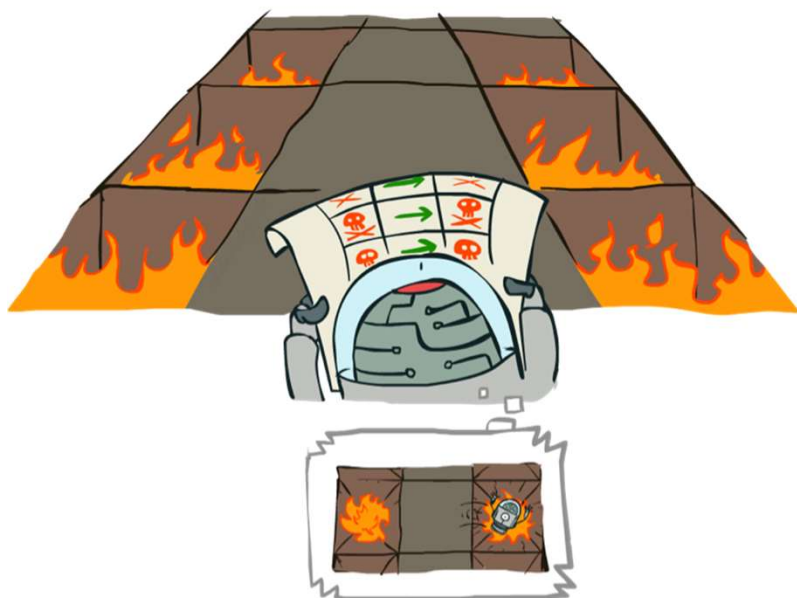
# Utilities for a Fixed Policy

- Another basic operation: compute the utility of a state $s$ under a fixed (generally non-optimal) policy

- Define the utility of a state $s$, under a fixed policy $\pi$:
  $V^\pi(s)$ = expected total discounted rewards starting in s and following $\pi$

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

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



s

$\pi(s)$

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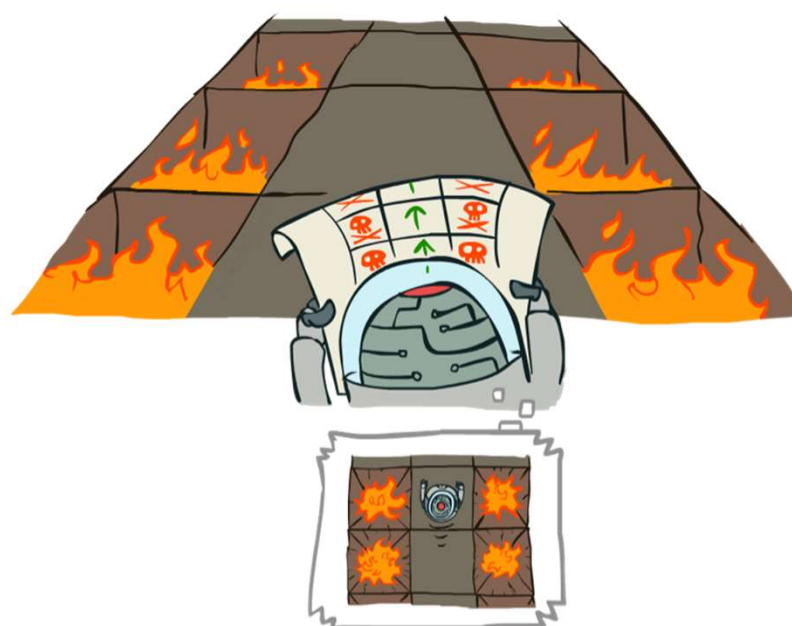
s, $\pi(s)$,s'

s'
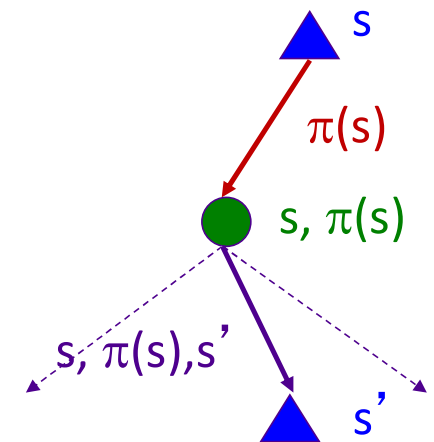
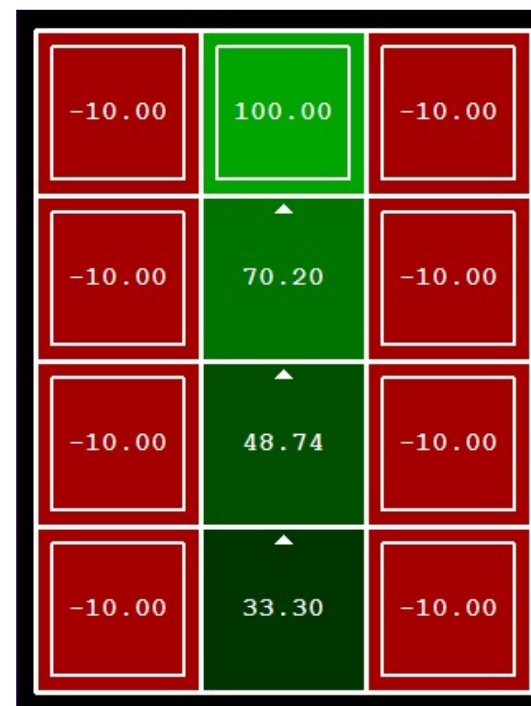# Example: Policy Evaluation

Always Go Right

Always Go Forward

# Example: Policy Evaluation

Always Go Right

Always Go Forward

# Policy Evaluation

- How do we calculate the $V$'s for a fixed policy $\pi$?

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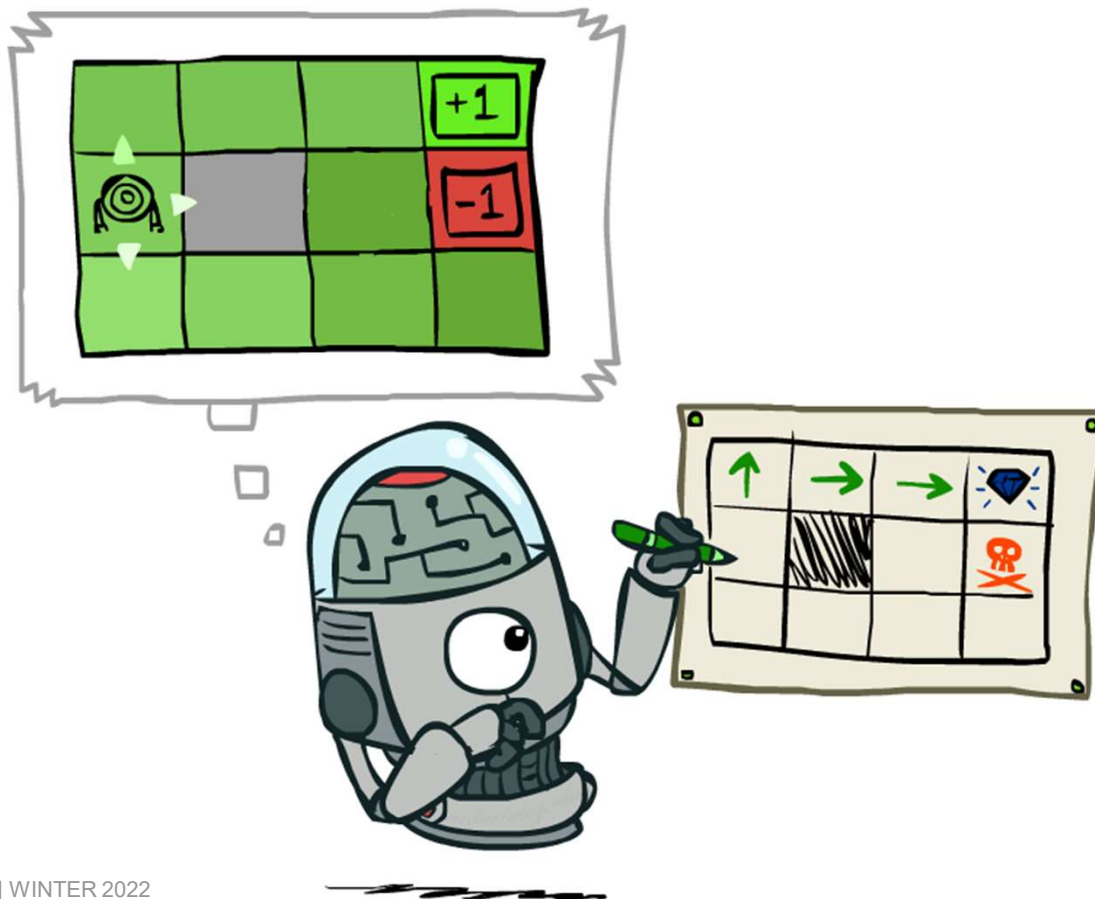  (like value iteration)

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

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

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

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

s

$\pi(s)$

s, $\pi(s)$

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

s'
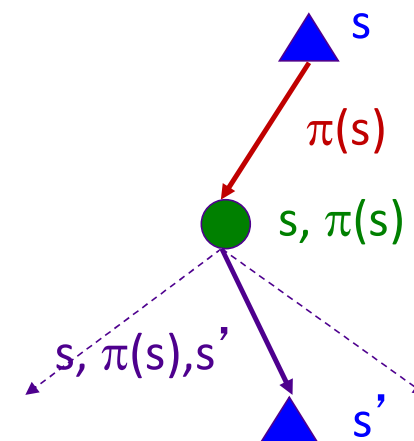
# Policy Extraction

# Computing Actions from Values

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

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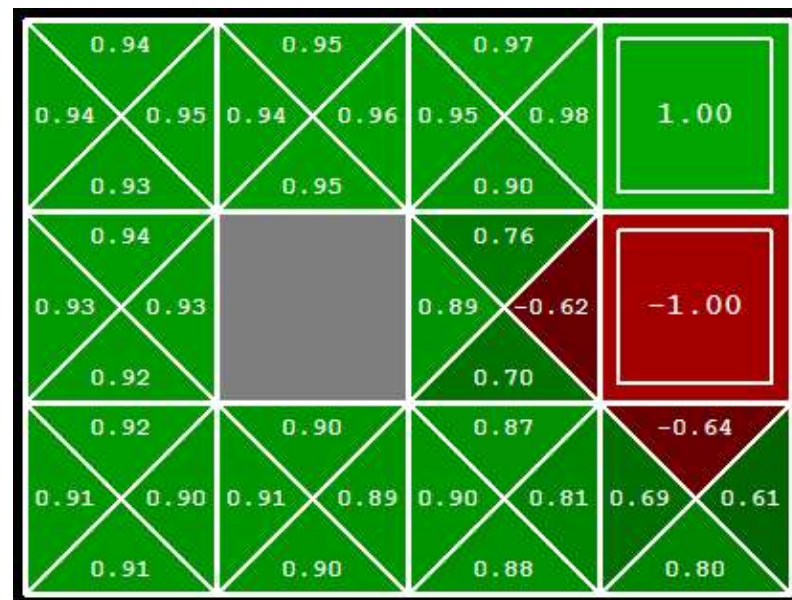
- We need to do a mini-expectimax (one step)

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

- This is called policy extraction, since it gets the policy implied by the values
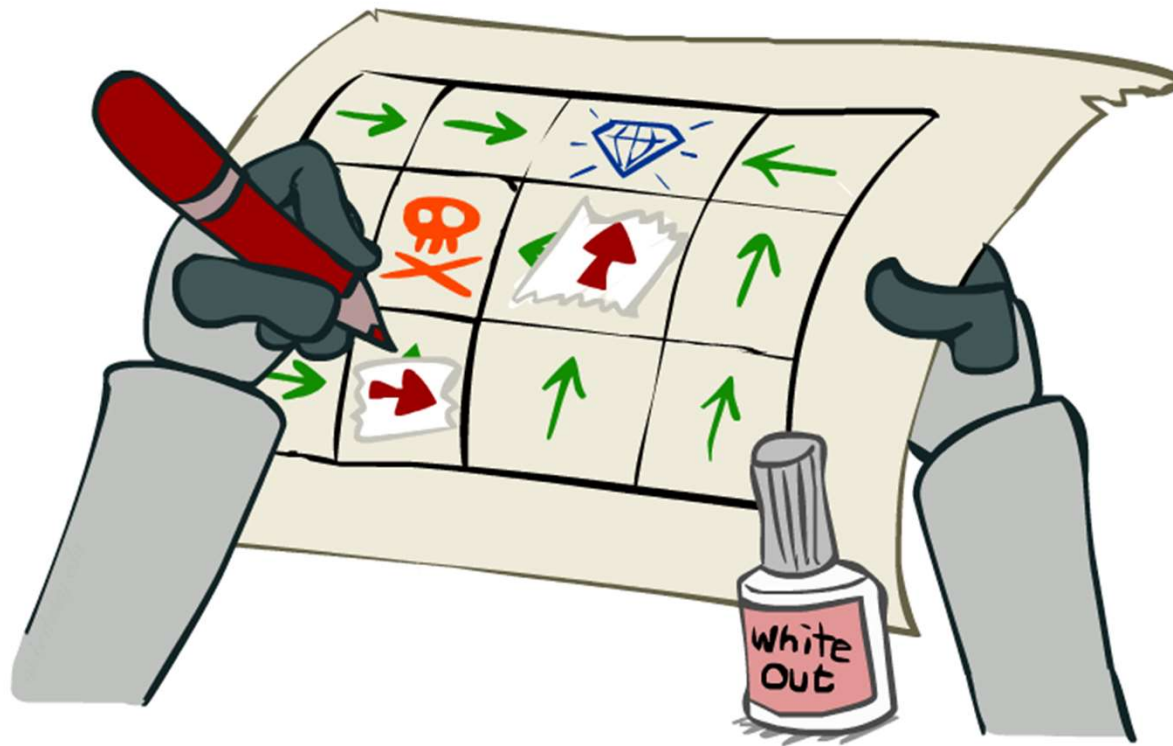
# Computing Actions from Q-Values

- 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 Iteration

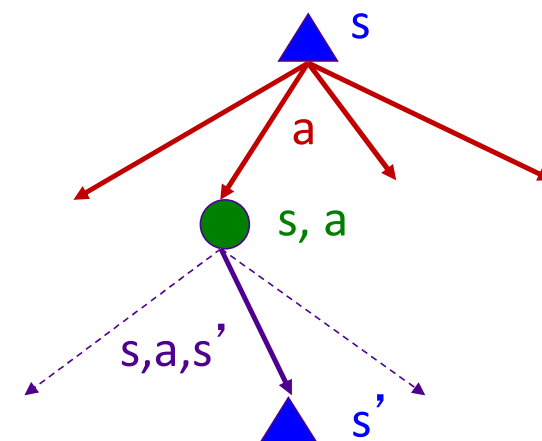# Problems with Value Iteration

- Value iteration repeats the Bellman updates:

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

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

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

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

**k=0**

Noise = 0.2
Discount = 0.9
Living reward = 0

**k=1**
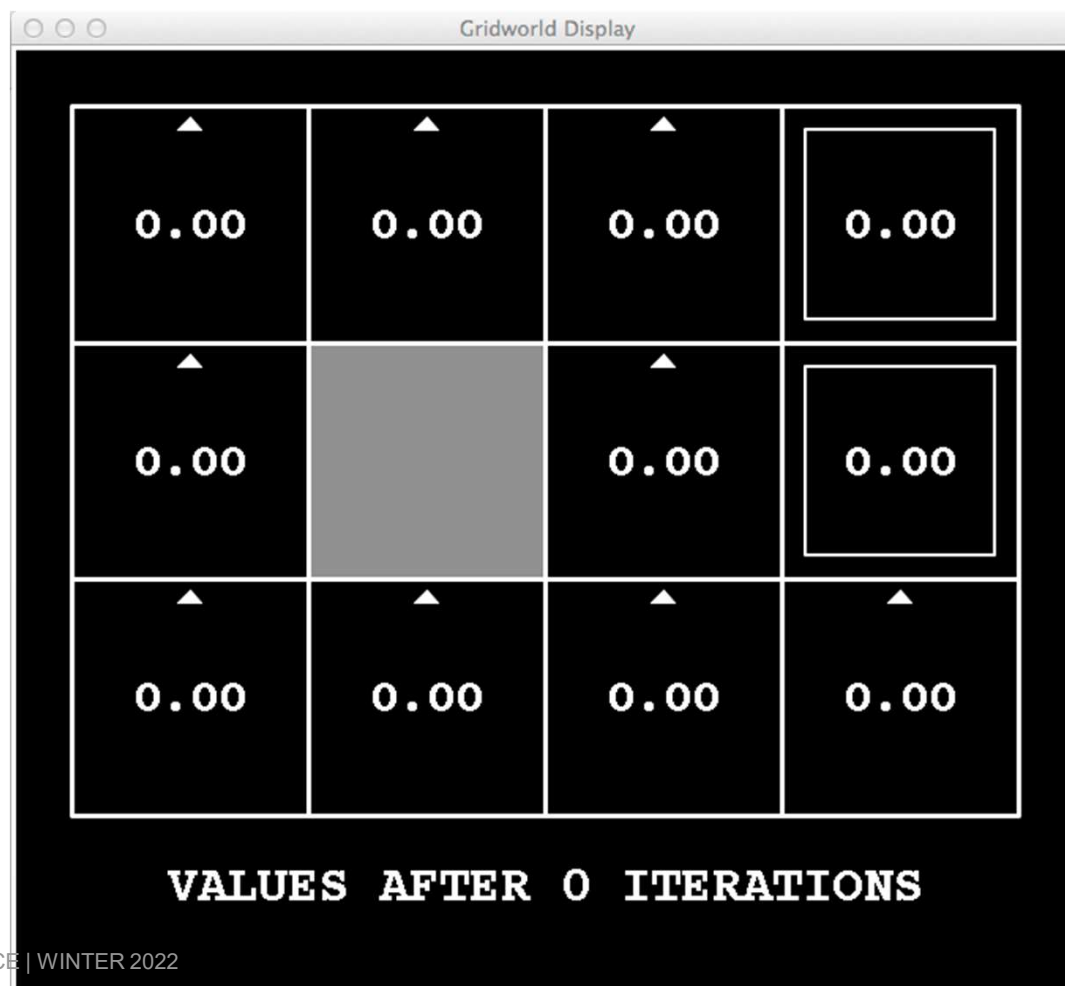
Noise = 0.2
Discount = 0.9
Living reward = 0    | 23

**k=2**



VALUES AFTER 2 ITERATIONS

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
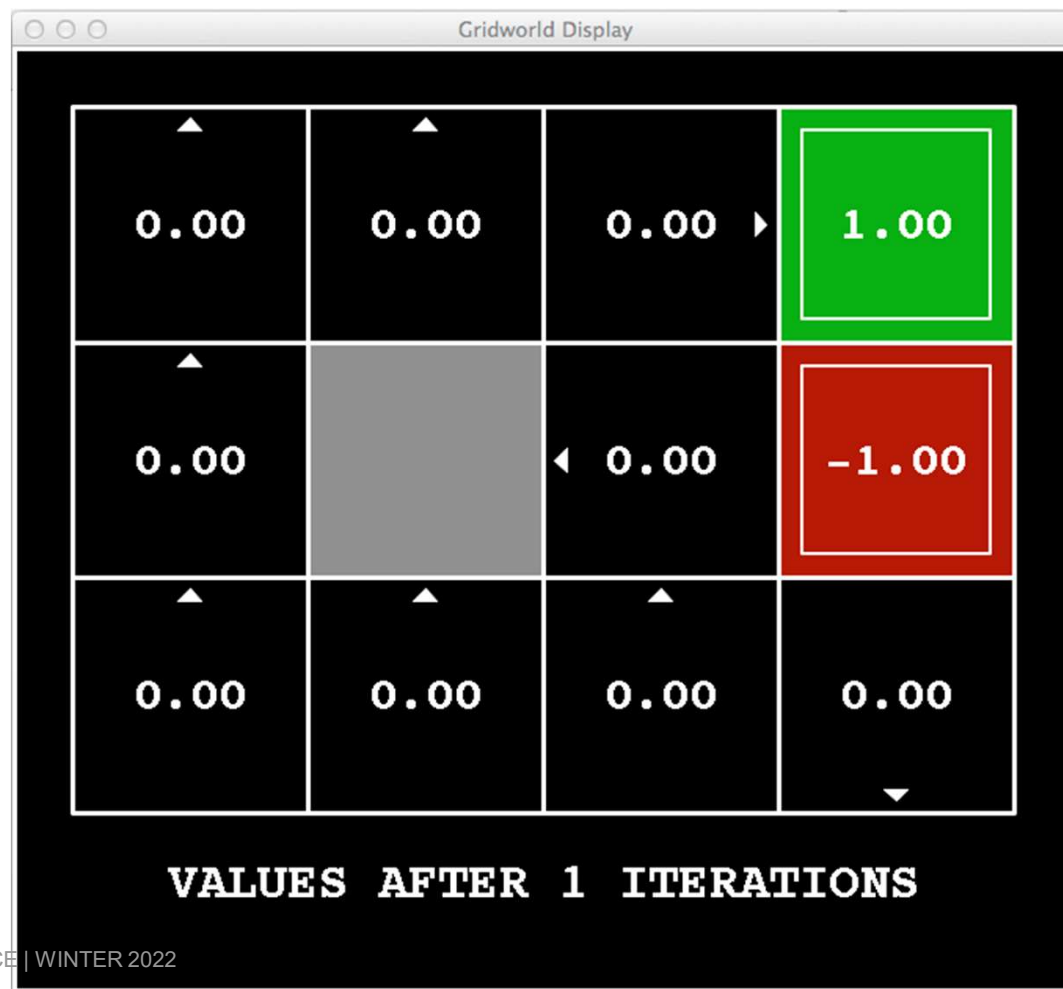Living reward = 0

# k=10



VALUES AFTER 10 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

# k=11



Noise = 0.2
Discount = 0.9
Living reward = 0

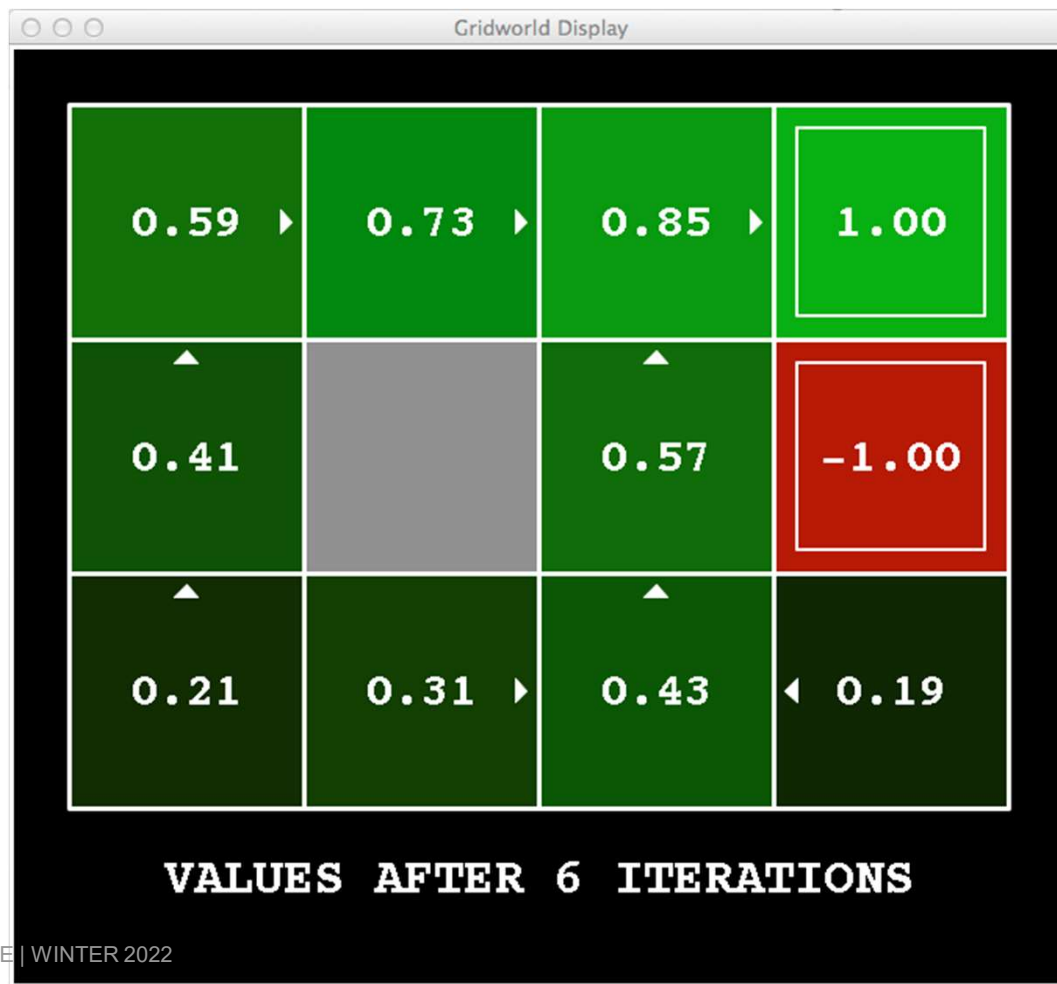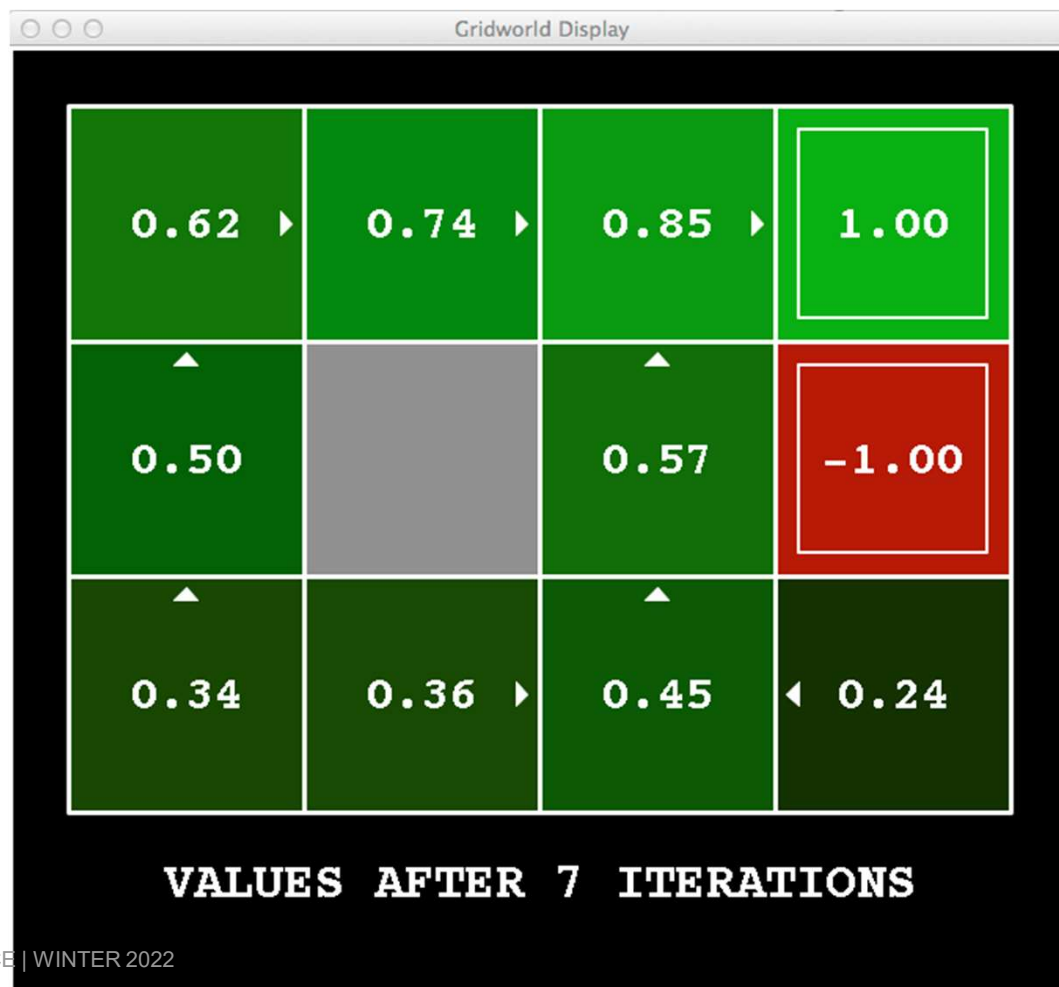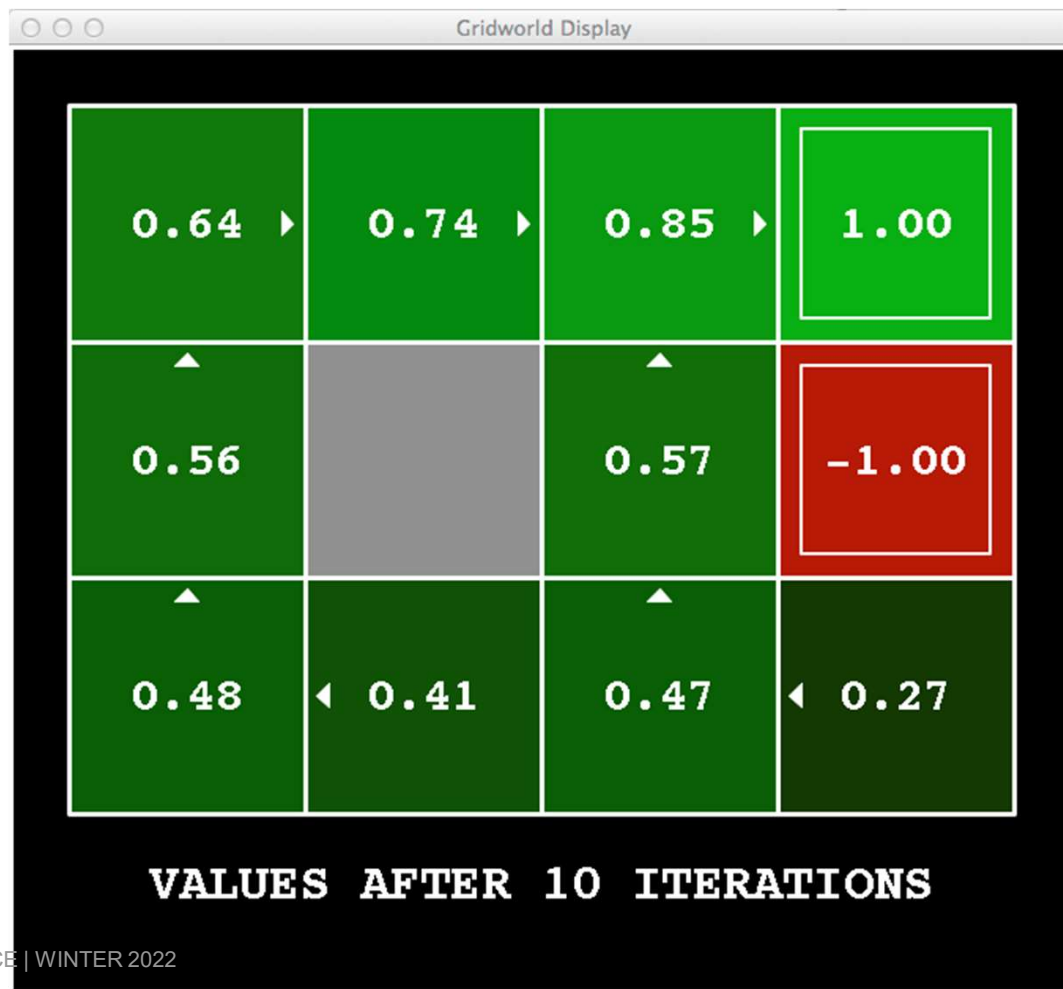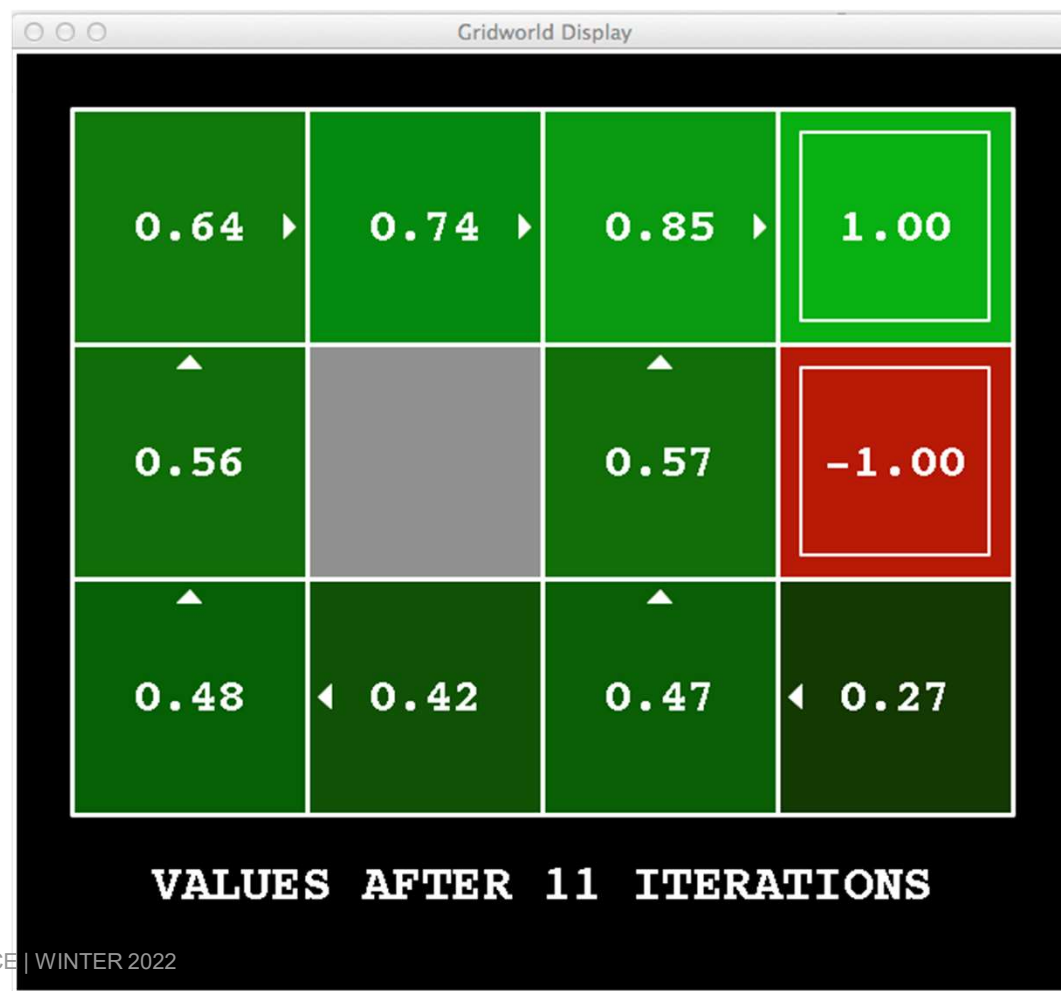**k=12**



Noise = 0.2
Discount = 0.9
Living reward = 0

# k=100



VALUES AFTER 100 ITERATIONS
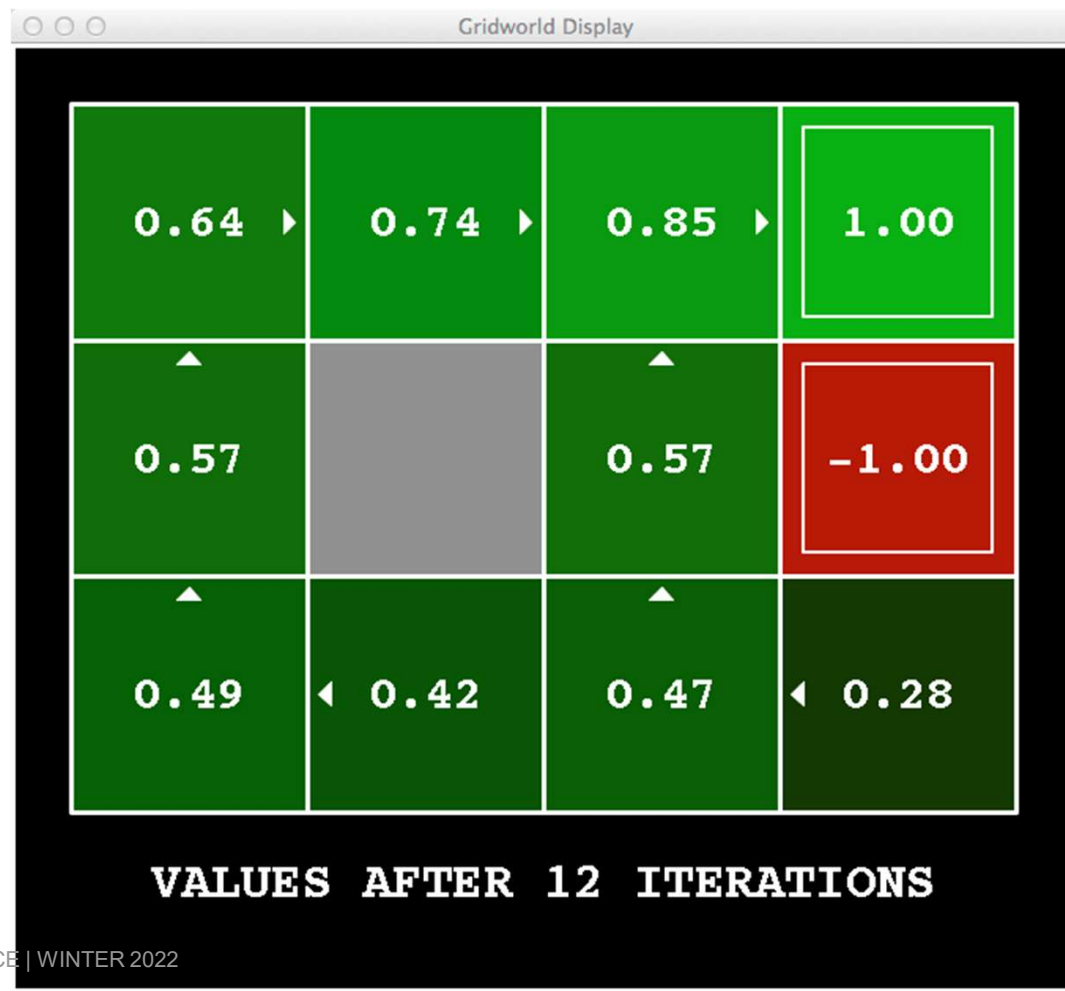
Noise = 0.2
Discount = 0.9
Living reward = 0

# 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 Iteration

- Evaluation: For fixed current policy $\pi$, 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:

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

# Comparison

- Both value iteration and policy iteration compute the same thing (all optimal values)

- In value iteration:
  - Every iteration updates both the values and (implicitly) the policy
  - We don't track the policy, but taking the max over actions implicitly recomputes it

- In policy iteration:
  - We do several passes that update utilities with fixed policy (each pass is fast because we consider only one action, not all of them)
  - After the policy is evaluated, a new policy is chosen (slow like a value iteration pass)
  - The new policy will be better (or we're done)

- Both are dynamic programs for solving MDPs

# Summary: MDP Algorithms

- So you want to….
  - 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