

## CSE 473: Artificial Intelligence

### Markov Decision Processes

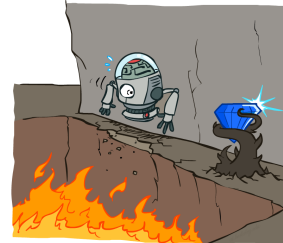


Dieter Fox

University of Washington

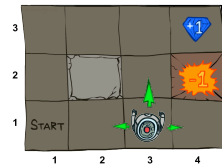
[Slides originally created by Dan Klein & Pieter Abbeel for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at <http://ai.berkeley.edu>.]

## 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

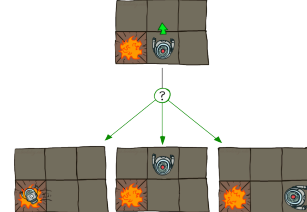


## Grid World Actions

### 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

$T(s_{11}, E, \dots)$   
 $T(s_{31}, N, s_{11}) = 0$   
 $T(s_{31}, N, s_{32}) = 0.8$   
 $T(s_{31}, N, s_{21}) = 0.1$   
 $T(s_{31}, N, s_{41}) = 0.1$   
 $\dots$

*T is a Big Table!*  
11 X 4 x 11 = 484 entries

For now, we give this as input to the agent



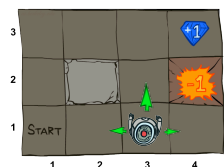
## Markov Decision Processes

- 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  $R(s, a, s')$

$R(s_{32}, N, s_{33}) = -0.01$  ← Cost of breathing  
 $R(s_{32}, N, s_{42}) = -1.01$   
 $R(s_{33}, E, s_{43}) = 0.99$

R is also a Big Table!

For now, we also give this to the agent



### Markov Decision Processes

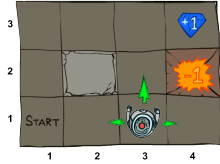
- An MDP is defined by:
  - A set of states  $s$  in  $S$
  - A set of actions  $a$  in  $A$
  - A transition function  $T(s, a, s')$ 
    - Probability that a from  $s$  leads to  $s'$ , i.e.,  $P(s' | s, a)$
    - Also called the model or the dynamics
  - A reward function  $R(s, a, s')$ 
    - Sometimes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

...

$R(s_{33}) = -0.01$

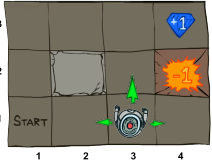
$R(s_{42}) = -1.01$

$R(s_{43}) = 0.99$




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



### What is Markov about MDPs?

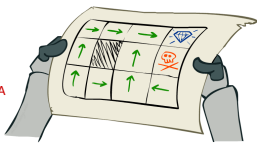
- "Markov" generally means that given the present state, the future and the past are independent
- For Markov decision processes, "Markov" means action outcomes depend only on the current state
 
$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1} = a_{t-1}, \dots, S_0 = s_0) = P(S_{t+1} = s' | S_t = s_t, A_t = a_t)$$
- This is just like search, where the successor function could only depend on the current state (not the history)



Andrey Markov  
(1856-1922)

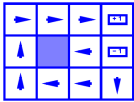
### Policies

- 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  $\pi$  gives an action for each state
  - An optimal policy is one that maximizes expected utility if followed
  - An explicit policy defines a reflex agent
- Expectimax didn't compute entire policies
  - It computed the action for a single state only

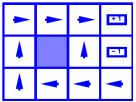


Optimal policy when  $R(s, a, s') = -0.03$  for all non-terminals  $s$

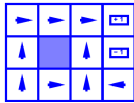
### Optimal Policies



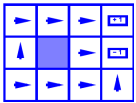
$R(s) = -0.01$



$R(s) = -0.03$



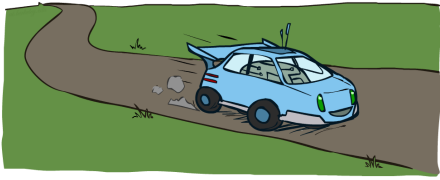
$R(s) = -0.4$



$R(s) = -2.0$

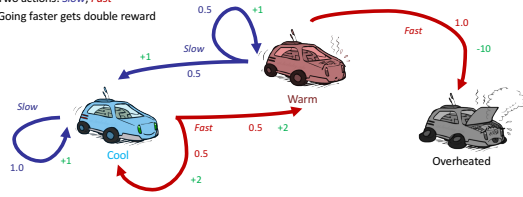
Cost of breathing

### Example: Racing

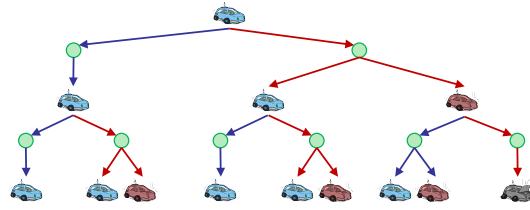


### Example: Racing

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

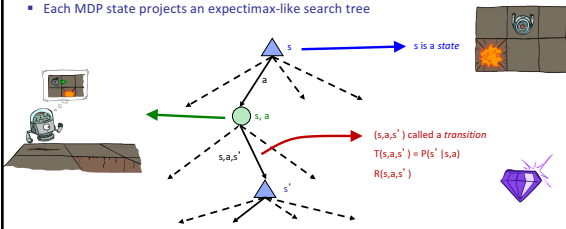


### Racing Search Tree

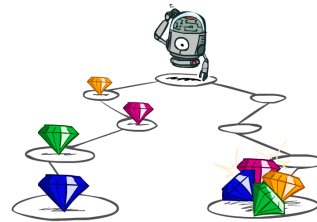


### MDP Search Trees

- Each MDP state projects an expectimax-like search tree

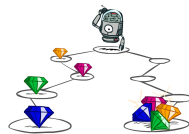


### Utilities of Sequences



### 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]



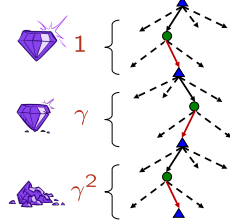
### Discounting

- 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



## Discounting

- How to discount?
  - Each time we descend a level, we multiply in the discount once
- Why discount?
  - Sooner rewards probably do have higher utility than later rewards
  - Also helps our algorithms converge
- Example: discount of 0.5
  - $U([1,2,3]) = 1*1 + 0.5*2 + 0.25*3$
  - $U([1,2,3]) < U([3,2,1])$



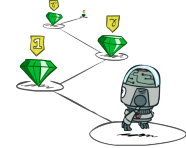
## Stationary Preferences

- Theorem: if we assume **stationary preferences**:

$$[a_1, a_2, \dots] \succ [b_1, b_2, \dots]$$

$$\Leftrightarrow$$

$$[r, a_1, a_2, \dots] \succ [r, b_1, b_2, \dots]$$



- Then: there are only two ways to define utilities

- Additive utility:  $U([r_0, r_1, r_2, \dots]) = r_0 + r_1 + r_2 + \dots$
- Discounted utility:  $U([r_0, r_1, r_2, \dots]) = r_0 + \gamma r_1 + \gamma^2 r_2 + \dots$

## Quiz: Discounting

- Given:
 

10				1
a	b	c	d	e

 $10 * \gamma^3 = 1 * \gamma$   
 $\gamma^2 = \frac{1}{10}$ 
  - 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  $\gamma$  are West and East equally good when in state d?

## Infinite Utilities?!

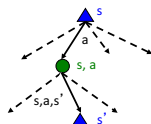
- Problem: What if the game lasts forever? Do we get infinite rewards?
- Solutions:
  - Finite horizon: (similar to depth-limited search)
    - Terminate episodes after a fixed T steps (e.g. life)
    - Gives nonstationary policies ( $\gamma$  depends on time left)
  - Discounting: use  $0 < \gamma 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

$$U([r_0, \dots, r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_t \leq R_{\max} / (1 - \gamma)$$
    - Smaller  $\gamma$  means smaller "horizon" – shorter term focus
  - Absorbing state: guarantee that for every policy, a terminal state will eventually be reached (like "overheated" for racing)



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



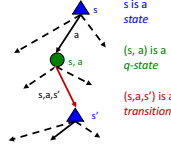
## Solving MDPs



- Value Iteration
- Policy Iteration
- Reinforcement Learning

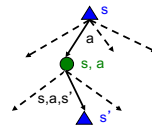
## Optimal Quantities

- 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  $s$



## Values of States

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



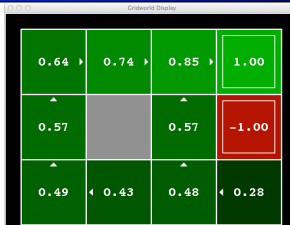
- Recursive definition of value:

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

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

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

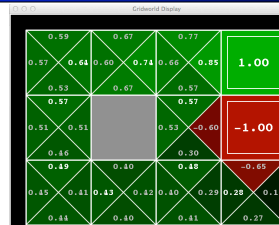
## Snapshot of Demo – Gridworld V Values



VALUES AFTER 100 ITERATIONS

Noise = 0.2  
Discount = 0.9  
Living reward = 0

## Snapshot of Demo – Gridworld Q Values

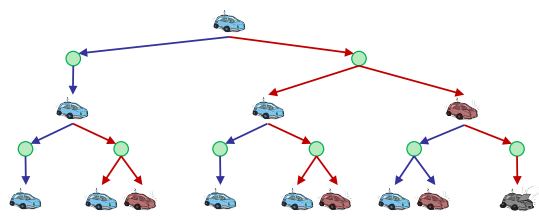


Q-VALUES AFTER 100 ITERATIONS

Noise = 0.2  
Discount = 0.9  
Living reward = 0

direction: q-function gets max

## Racing Search Tree

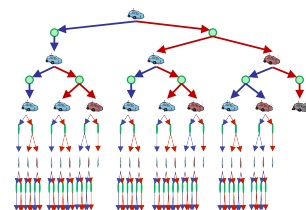


## Racing Search Tree

- 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  $\gamma < 1$



### 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  $s$

### Computing Time-Limited Values

### Value Iteration

### The Bellman Equations

### The Bellman Equations

- Definition of "optimal utility" via expectimax recurrence gives a simple one-step lookahead relationship amongst optimal utility values
 
$$V^*(s) = \max_a Q^*(s, a)$$

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

$$V^*(s) = \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')]$$
- 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:
 
$$V^*(s) = \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')]$$
- Value iteration computes them:
 
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_k(s')]$$
- Value iteration is just a **fixed point solution method**
  - ... though the  $V_k$  vectors are also interpretable as time-limited values

# Value Iteration Algorithm

- Start with  $V_0(s) = 0$ :
- Given vector of  $V_k(s)$  values, do one ply of expectimax from each state:
 
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_k(s')]$$
- Repeat until convergence
- Complexity of each iteration:  $O(S^2A)$
- Number of iterations:  $\text{poly}(|S|, |A|, 1/(1-\gamma))$
- Theorem: will converge to unique optimal values

- **Complexity of each iteration:**  $O(S^2A)$
- **Number of iterations:**  $\text{poly}(|S|, |A|, 1/(1-\gamma))$
- **Theorem:** will converge to unique optimal values

$k=0$

Gridworld Display

0.00	0.00	0.00	0.00
0.00		0.00	0.00
0.00	0.00	0.00	0.00

VALUES AFTER 0 ITERATIONS

Noise = 0.2  
Discount = 0.9  
Living reward = 0

[illegible]

$k=1$

Grid World			
0.00	0.00	0.00	1.00
0.00		0.00	-1.00
0.00	0.00	0.00	0.00

**VALUES AFTER 1 ITERATIONS**

Noise = 0.2  
 Discount = 0.9  
 Living reward = 0

0.00	0.00	0.00	1.00
0.00		0.00	-1.00
0.00	0.00	0.00	0.00

**k=2**

The grid world environment is a 3x4 grid. The start state is at (0,0) and the goal state is at (2,3). Obstacles are at (1,1) and (1,2). The Q-table below shows the values after 2 iterations.

Q-Table			
0.00	0.00	0.72	1.00
0.00		0.00	-1.00
0.00	0.00	0.00	0.00

0.00	0.00	0.72	1.00
0.00		0.00	-1.00
0.00	0.00	0.00	0.00

$k=3$

Environment: 4x4

0.00	0.52	0.78	1.00
0.00		0.43	-1.00
0.00	0.00	0.00	0.00

**VALUES AFTER 3 ITERATIONS**

Noise = 0.2  
 Discount = 0.9  
 Living reward = 0

Grid World Environment:

0.00	0.52	0.78	1.00
0.00		0.43	-1.00
0.00	0.00	0.00	0.00

$k=4$

Continued Q-value

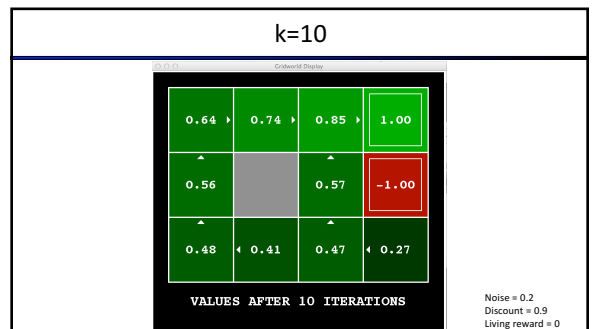
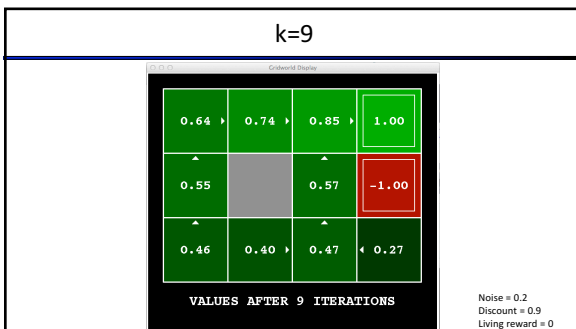
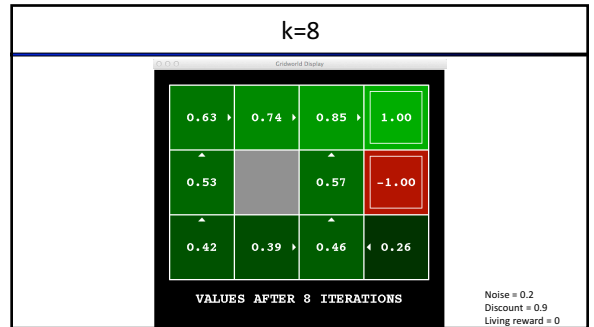
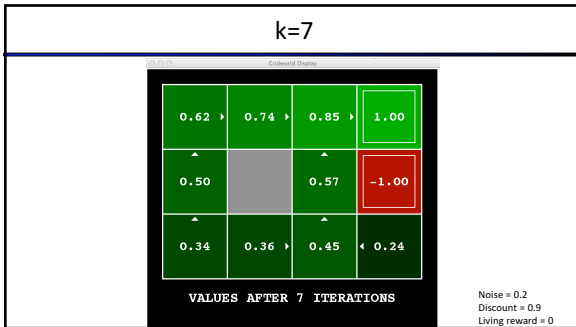
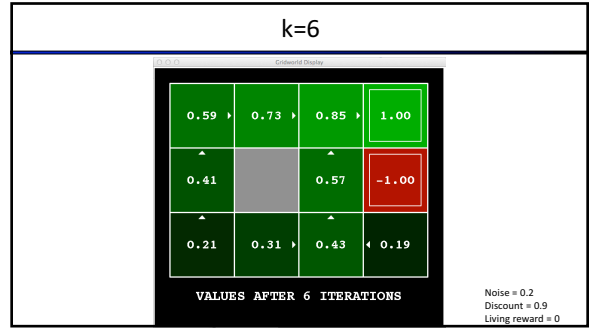
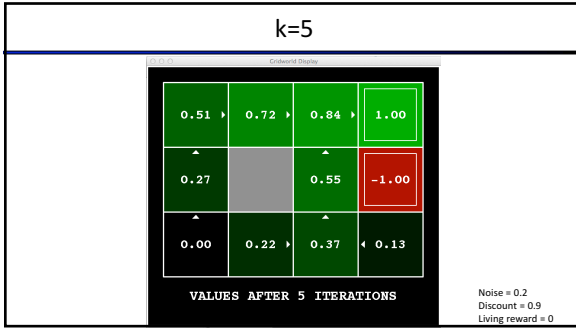
0.37	0.66	0.83	1.00
0.00		0.51	-1.00
0.00	0.00	0.31	0.00

VALUES AFTER 4 ITERATIONS

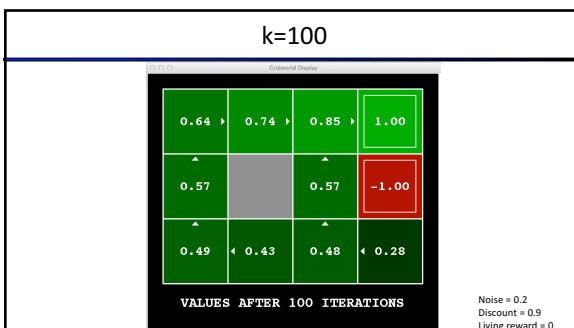
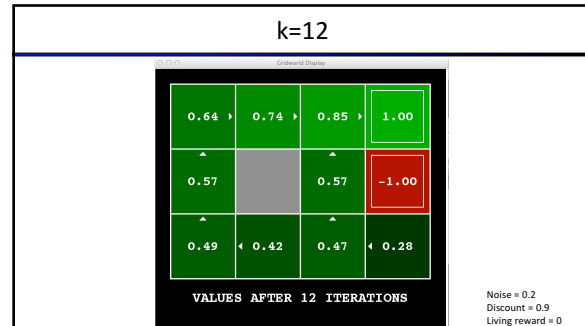
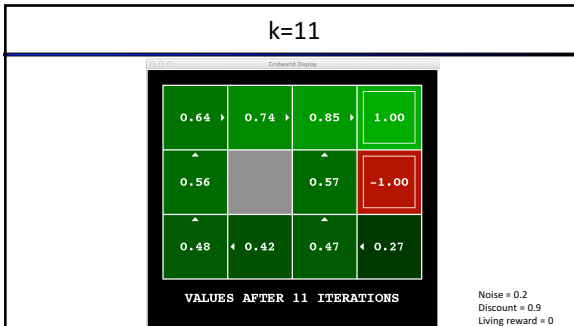
Noise = 0.2  
 Discount = 0.9  
 Learning Rate = 0

0.37	0.66	0.83	1.00
0.00		0.51	-1.00
0.00	0.00	0.31	0.00

VALUES AFTER 4 ITERATIONS







**Convergence\***

- How do we know the  $V_k$  vectors will converge?
- Case 1: If the tree has maximum depth  $M$ , then  $V_M$  holds the actual untruncated values  
**finite case**
- 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 max difference happens if big reward at  $k+1$  level
  - That last layer is at best all  $R_{\max}$
  - 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  
**discount factor!**

3 quantities:  
value  $v$   
action  $a$   
q-value  $(v,a)$

**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  
**find optimal action**

**Computing Actions from Q-Values**

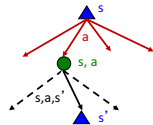
- Let's imagine we have the optimal q-values:
- How should we act?
  - Completely trivial to decide!
- $$\pi^*(s) = \arg \max_a Q^*(s, a)$$
- Important lesson: actions are easier to select from q-values than values!

## Problems with Value Iteration

- Value iteration repeats the Bellman updates:

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

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



## VI → Asynchronous VI

- Is it essential to back up **all** states in each iteration?
  - No!
- States may be backed up
  - many times or not at all
  - in any order
- As long as no state gets starved...
  - convergence properties still hold!!

56

k=1

0.00	0.00	0.00	1.00
0.00		0.00	-1.00
0.00	0.00	0.00	0.00

VALUES AFTER 1 ITERATIONS

Noise = 0.2  
Discount = 0.9  
Living reward = 0

k=2

0.00	0.00	0.72	1.00
0.00		0.00	-1.00
0.00	0.00	0.00	0.00

VALUES AFTER 2 ITERATIONS

Noise = 0.2  
Discount = 0.9  
Living reward = 0

k=3

0.00	0.52	0.78	1.00
0.00		0.43	-1.00
0.00	0.00	0.00	0.00

VALUES AFTER 3 ITERATIONS

Noise = 0.2  
Discount = 0.9  
Living reward = 0

## Asynch VI: Prioritized Sweeping

- Why backup a state if values of successors same? **update nearby position**
- Prefer backing a state
  - whose successors had most change
- Priority Queue of (state, expected change in value)
- Backup in the order of priority
- After backing a state update priority queue
  - for all predecessors