

# CS 3568: Intelligent Systems

## Markov Decision Processes (part 2)



Instructor: Tara Salman

Texas Tech University

Computer Science Department

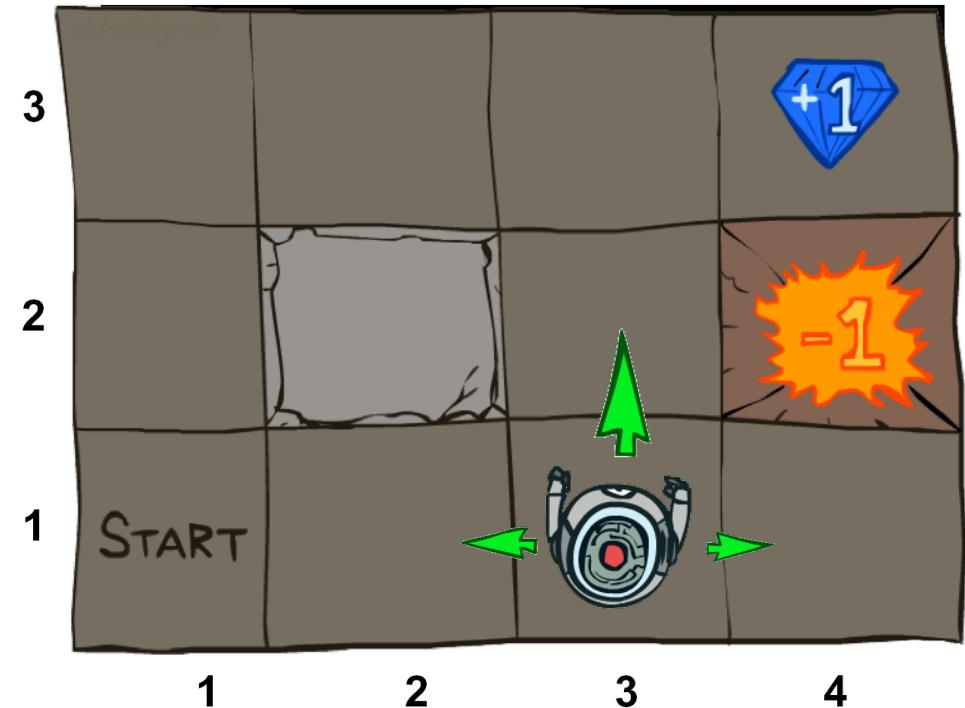
[These slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley ([ai.berkeley.edu](http://ai.berkeley.edu)).]

Texas Tech University

Tara Salman

# 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
  - 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 (discounted) rewards



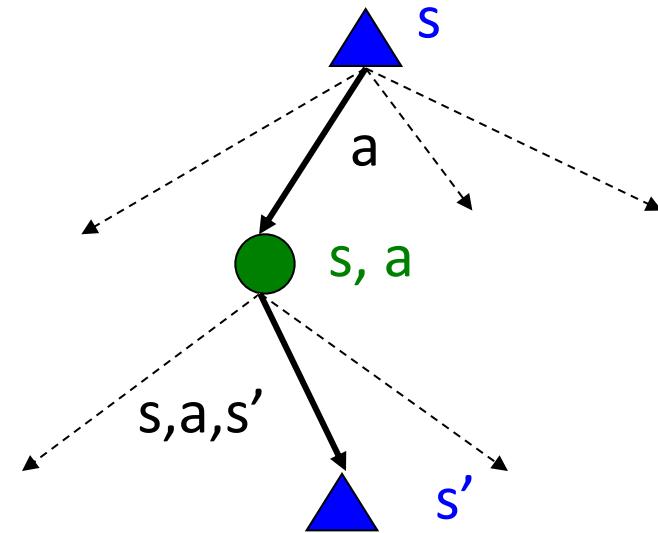
# Recap: MDPs

## ❑ Markov decision processes:

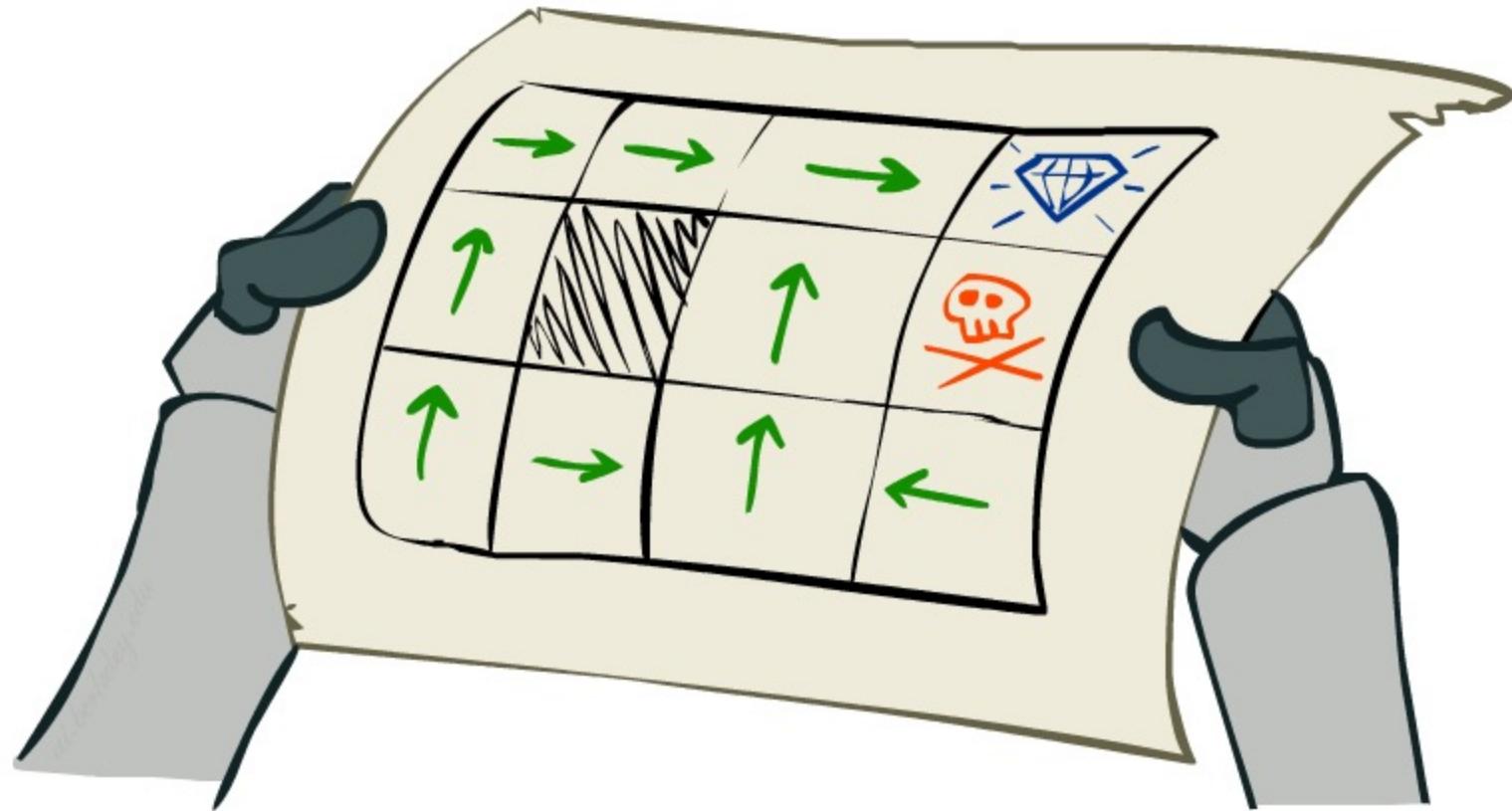
- States S
- Actions A
- Transitions  $P(s'|s,a)$  (or  $T(s,a,s')$ )
- Rewards  $R(s,a,s')$  (and discount  $\gamma$ )
- Start state  $s_0$

## ❑ Quantities:

- Policy = map of states to actions
- Utility = sum of discounted rewards
- Values = expected future utility from a state (max node)
- Q-Values = expected future utility from a q-state (chance node)

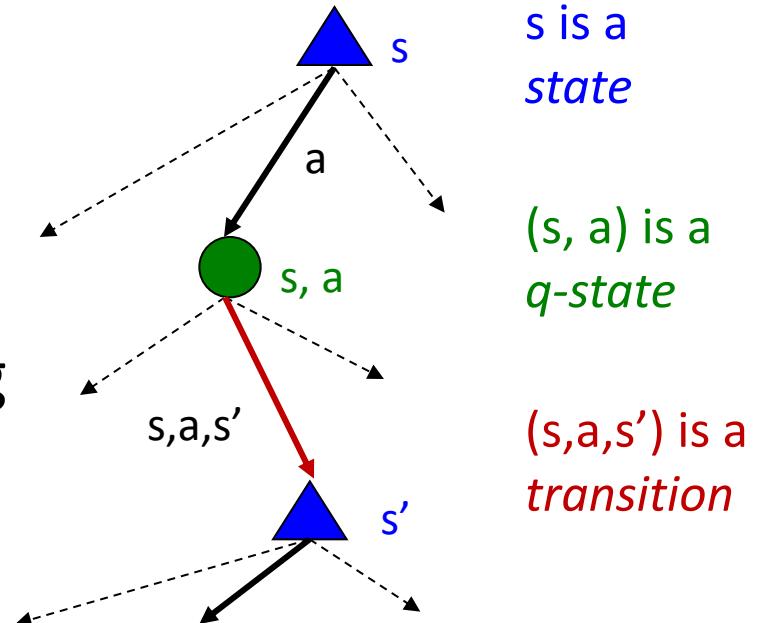


# Solving MDPs



# Optimal Quantities

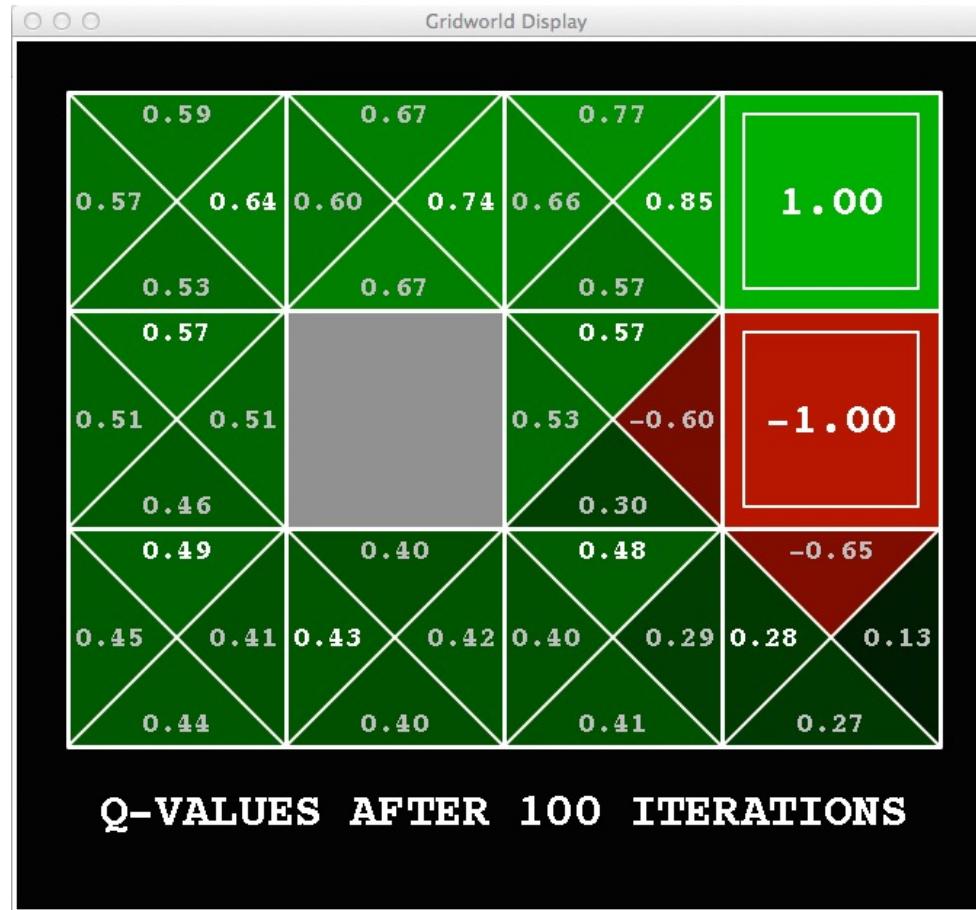
- The value (utility) of a state  $s$ :  
 $V^*(s)$  = expected utility starting in  $s$  and acting optimally
- The value (utility) of a q-state  $(s,a)$ :  
 $Q^*(s,a)$  = expected utility starting out having taken action  $a$  from state  $s$  and (thereafter) acting optimally
- The optimal policy:  
 $\pi^*(s)$  = optimal action from state  $s$



# Snapshot of Demo – Gridworld V Values



# Snapshot of Demo – Gridworld Q Values



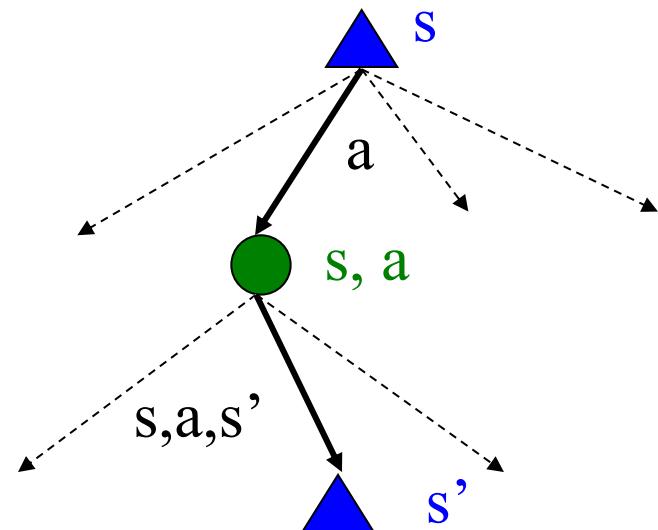
# Values of States

- 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:

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

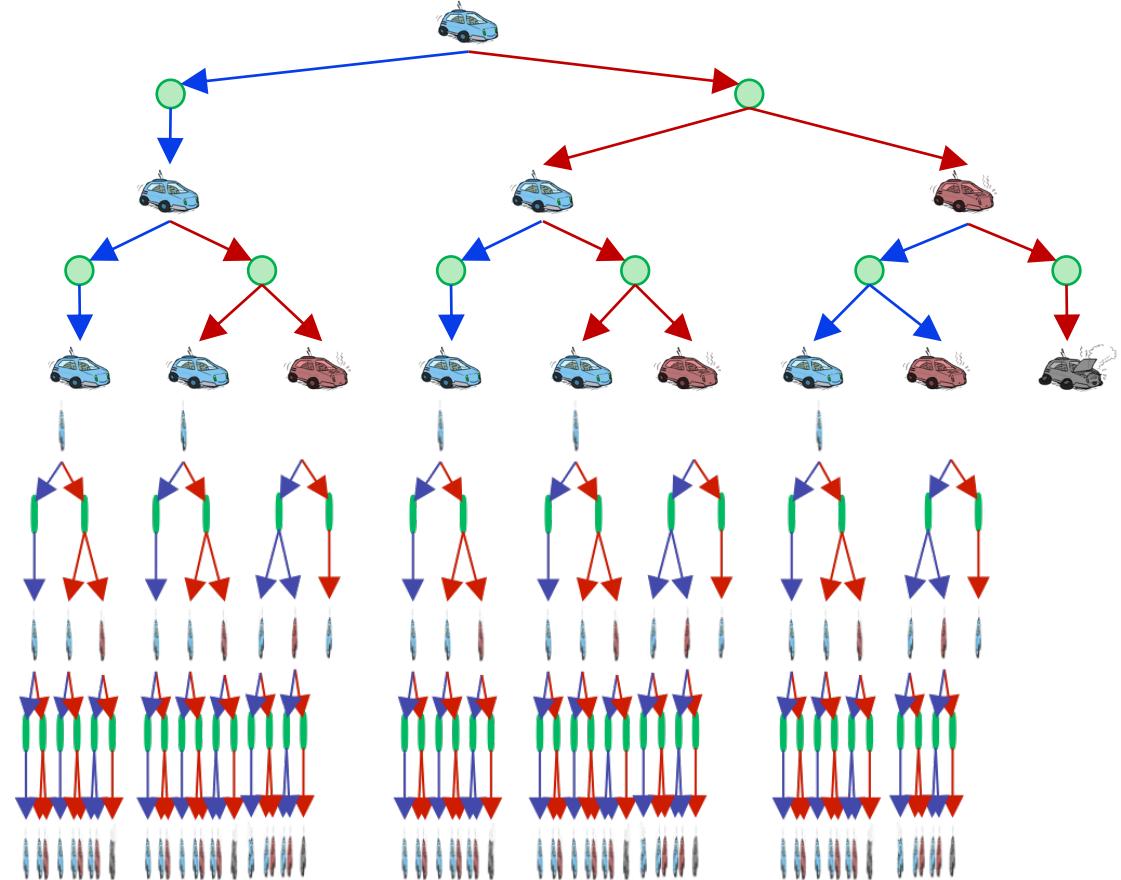
$$Q^*(s, a) = \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')]$$

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



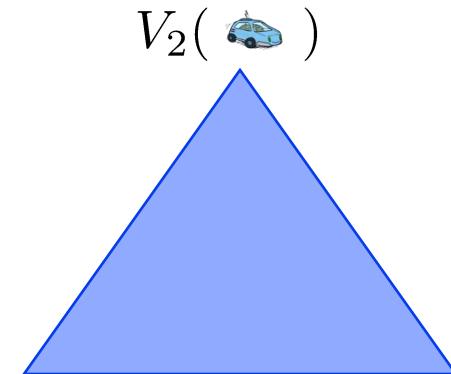
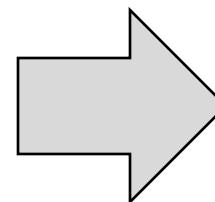
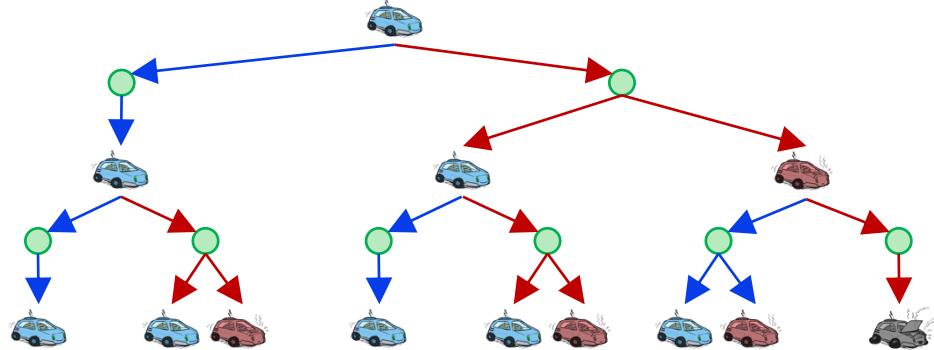
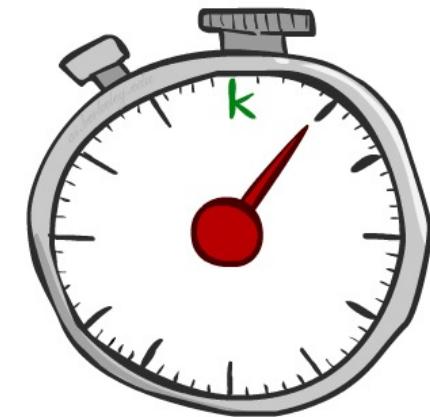
# Racing Search Tree

- ❑ 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  $\gamma < 1$

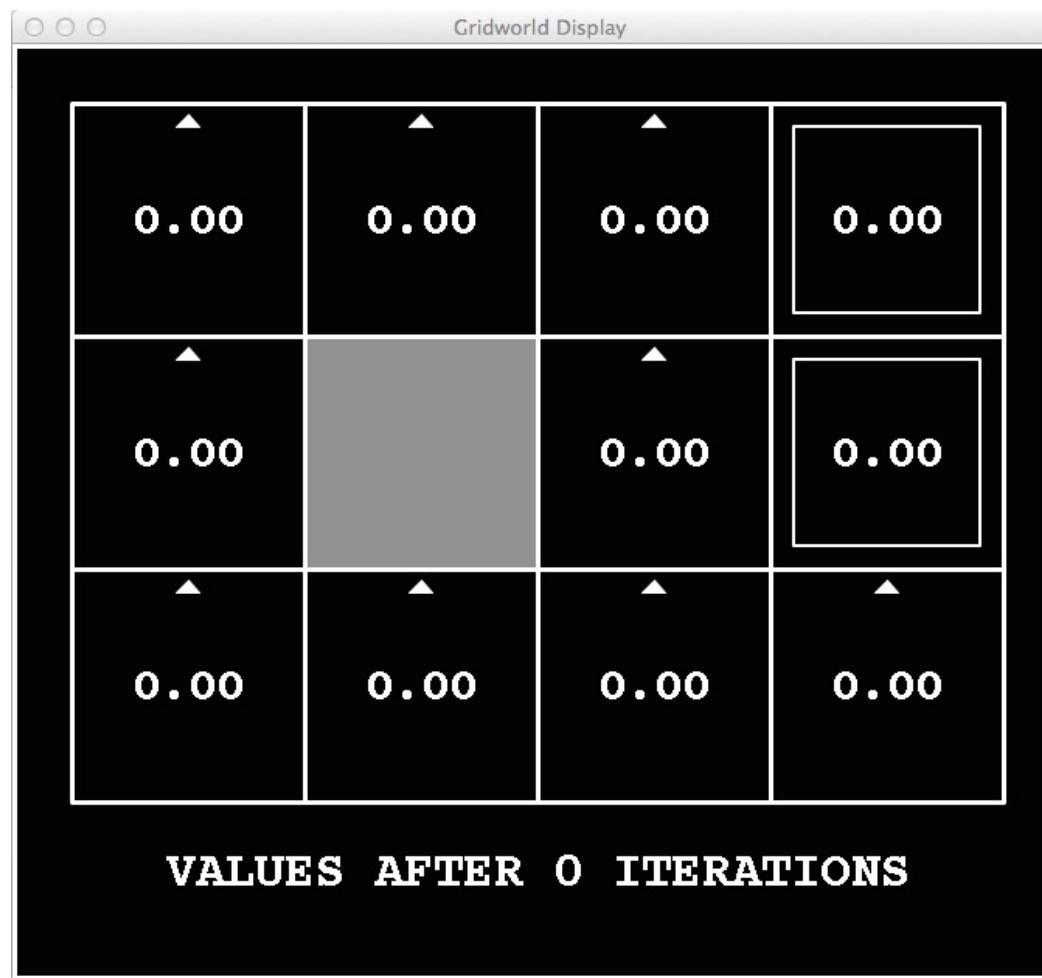


# Time-Limited Values

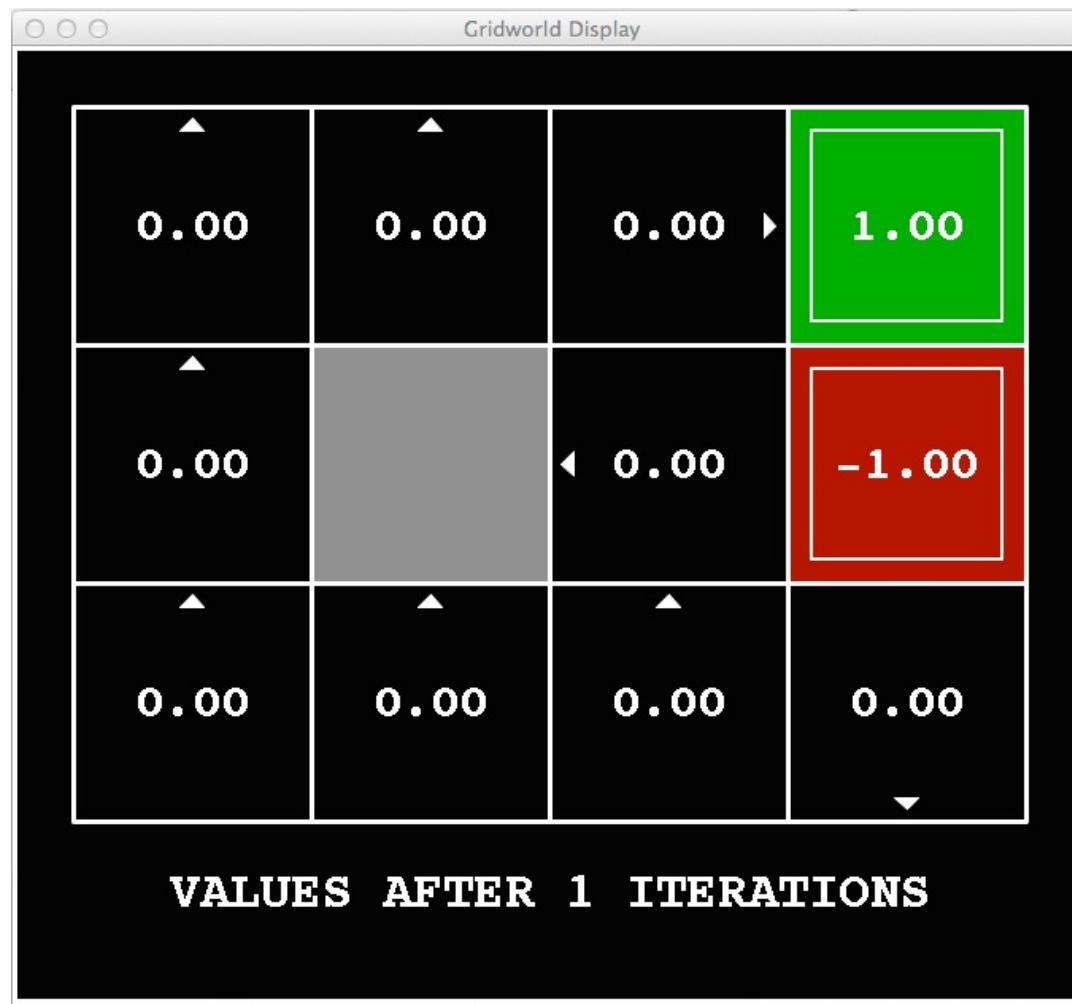
- ❑ Key idea: time-limited values
- ❑ Define  $V_k(s)$  to be the optimal value of  $s$  if the game ends in  $k$  more time steps
  - Equivalently, it's what a depth- $k$  expectimax would give from  $s$



**k=0**



**k=1**



**k=2**



**k=3**



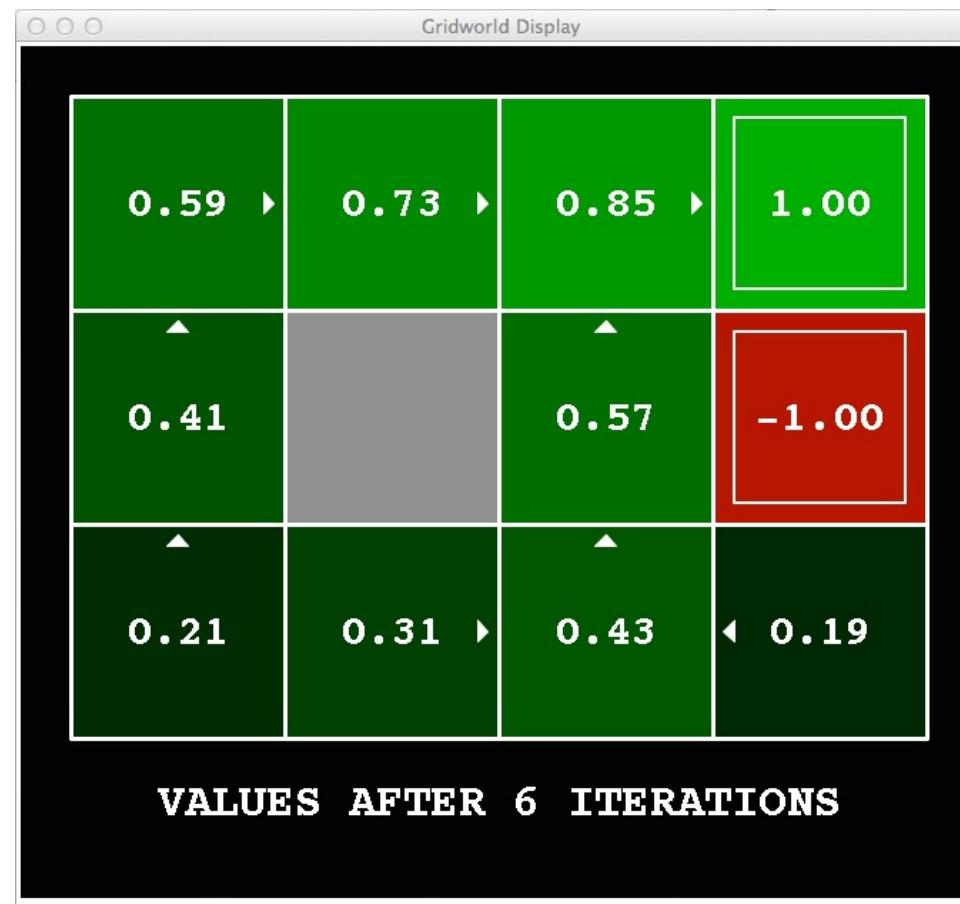
**k=4**



**k=5**



**k=6**



**k=7**



**k=8**



**k=9**



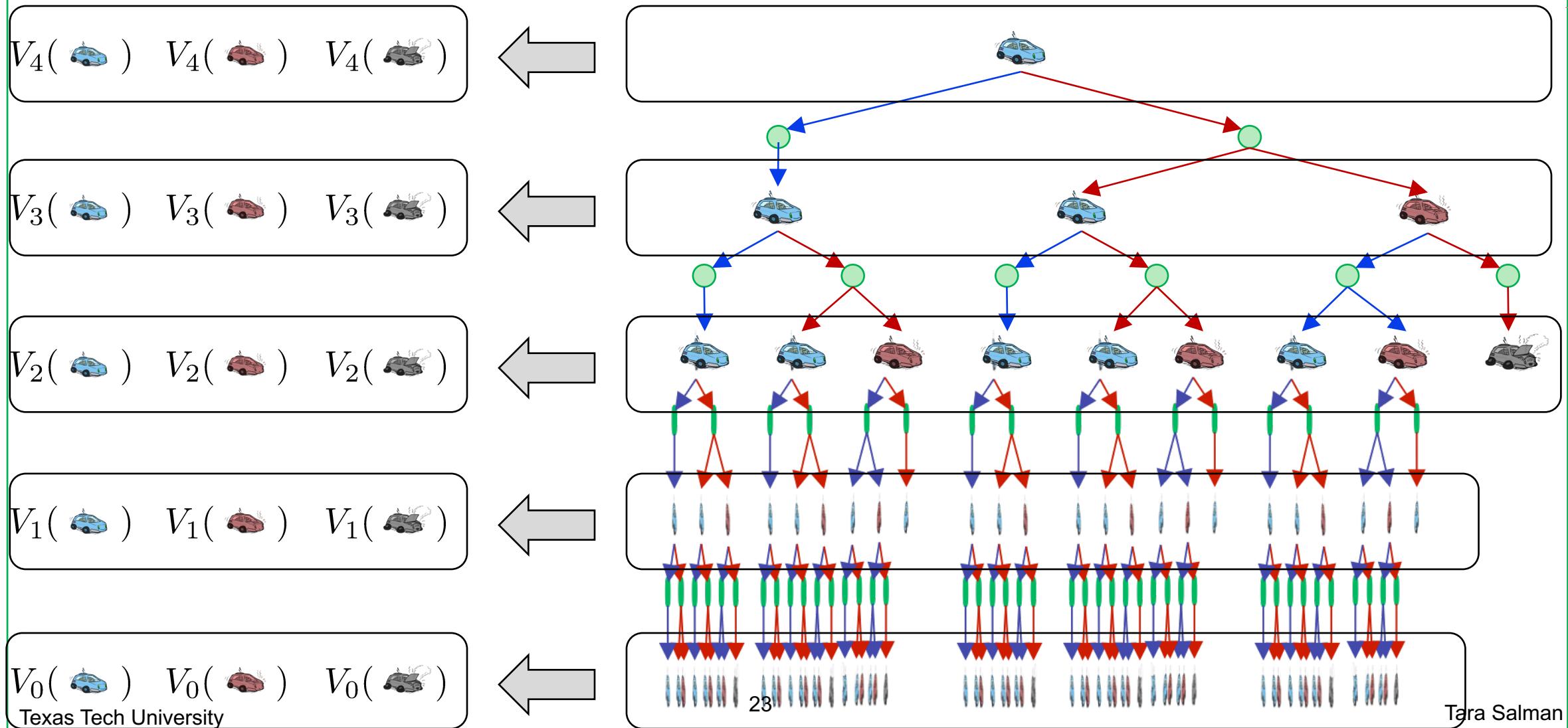
# k=10



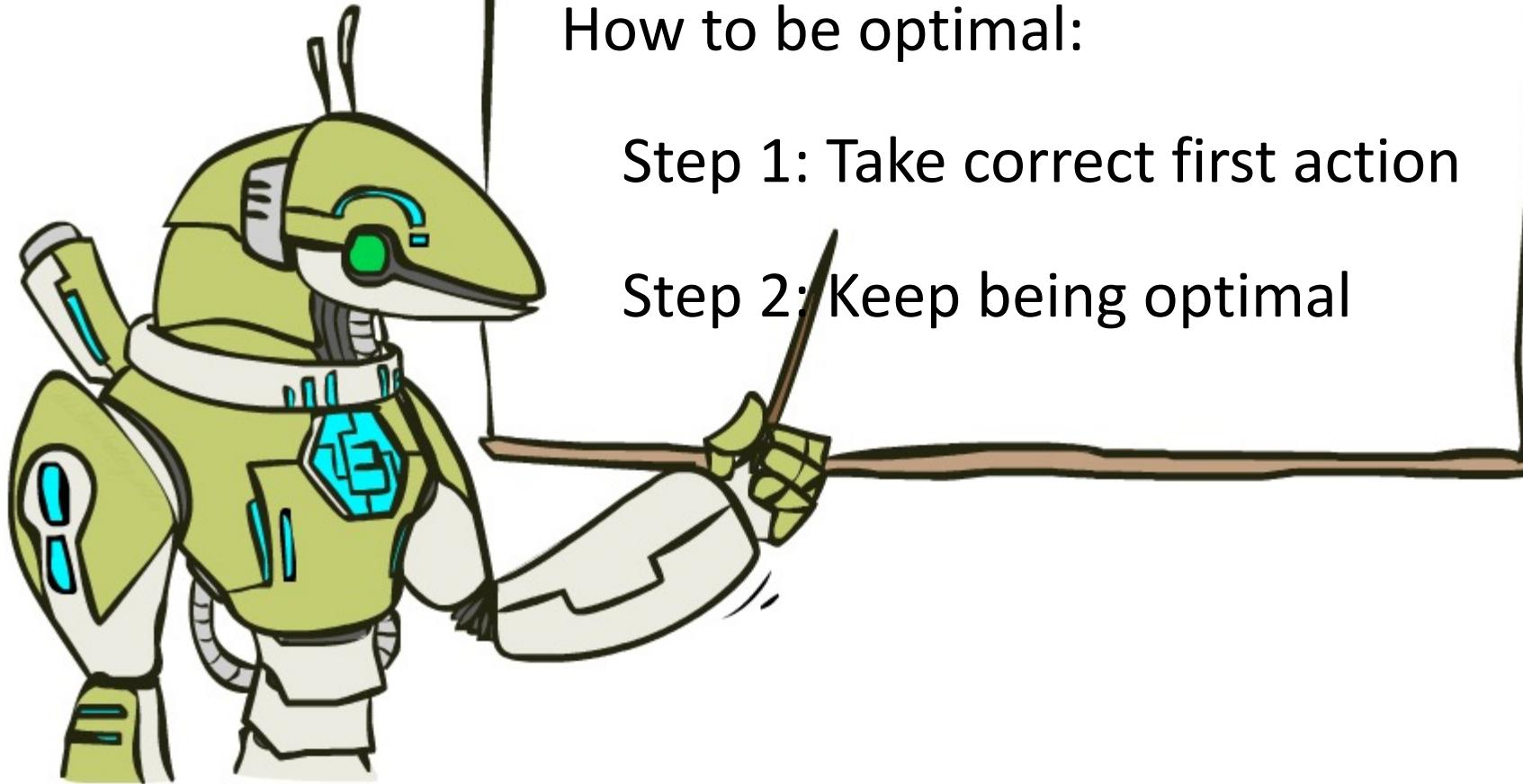
# k=100



# Computing Time-Limited Values



# The Bellman Equations



How to be optimal:

Step 1: Take correct first action

Step 2: Keep being optimal

# The Bellman Equations

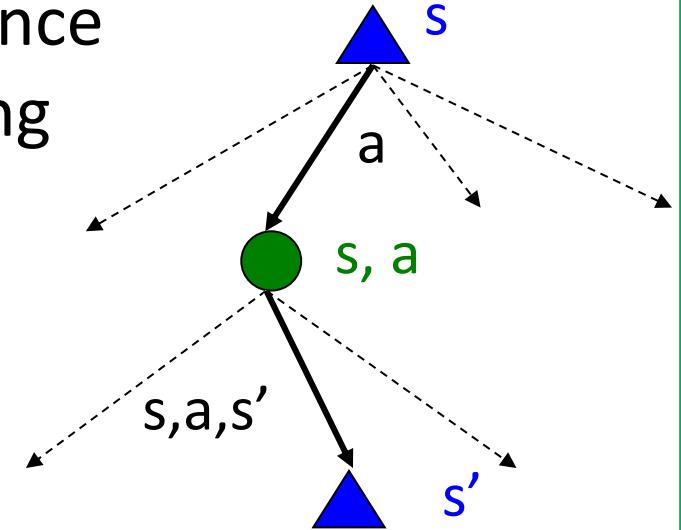
- Definition of “optimal utility” via expectimax recurrence gives a simple one-step lookahead relationship among optimal utility values

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

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

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

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



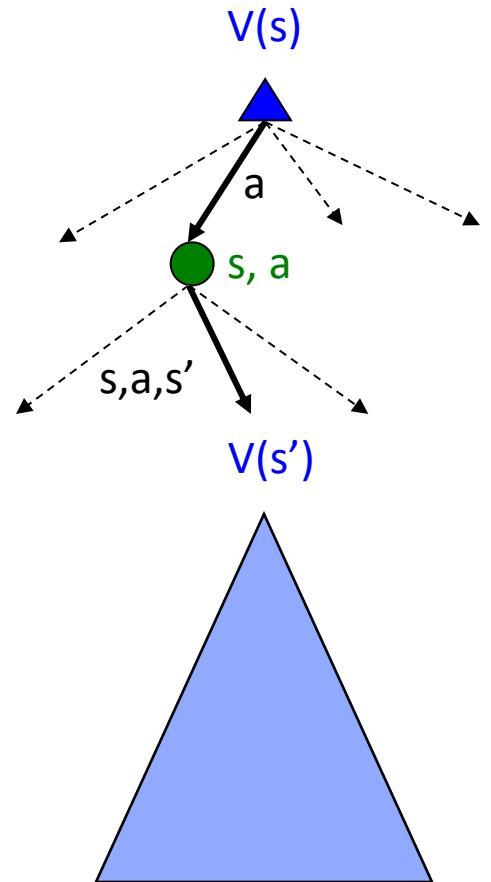
# Value Iteration

- Bellman equations **characterize** the optimal values:

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

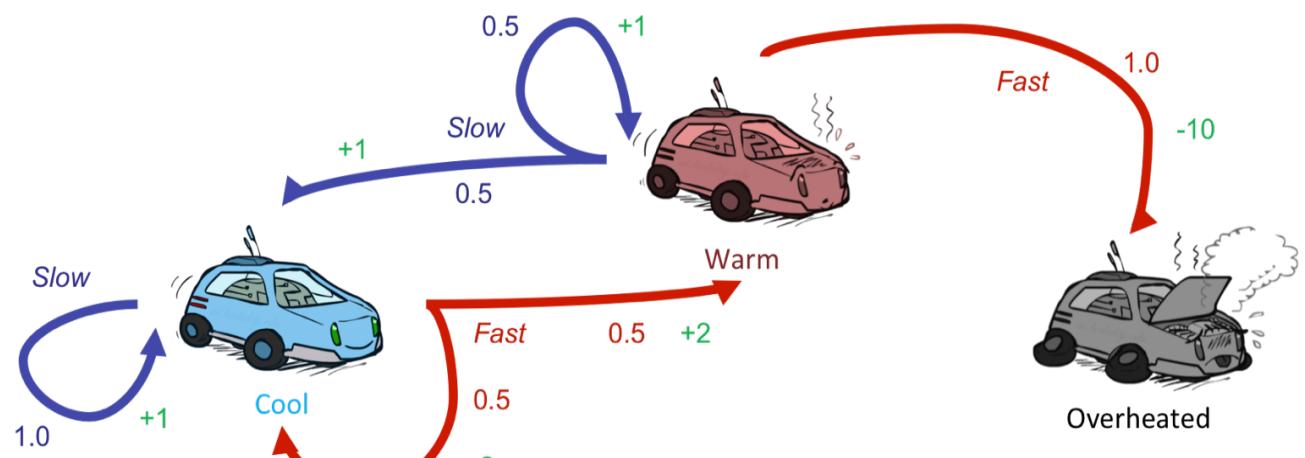
- Value iteration **computes** them:

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



# Example: Value Iteration

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



Assume no discount!

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

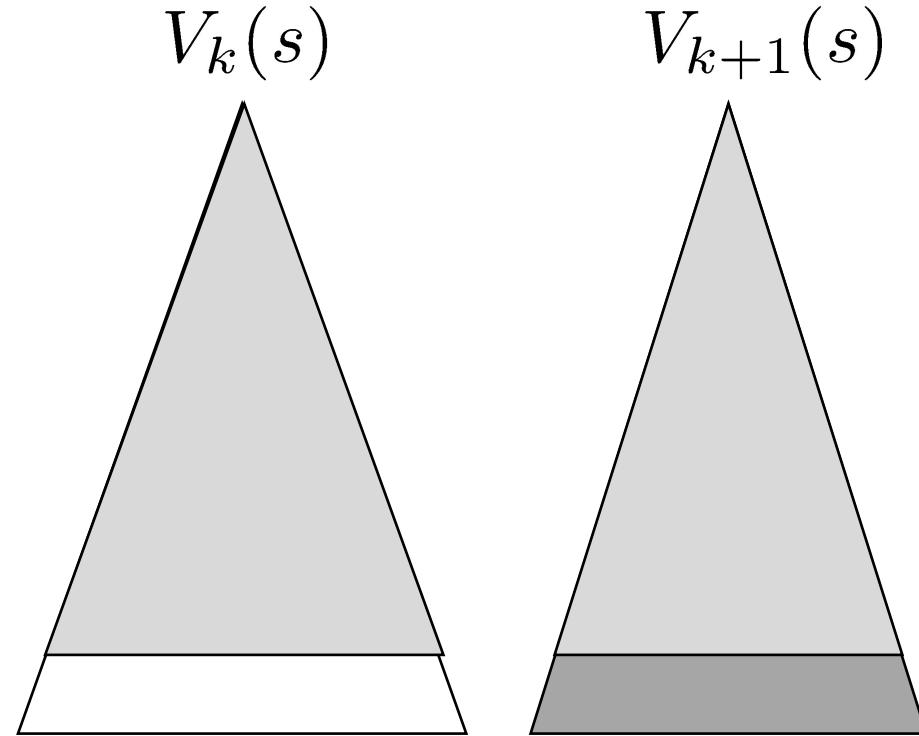
# Convergence\*

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

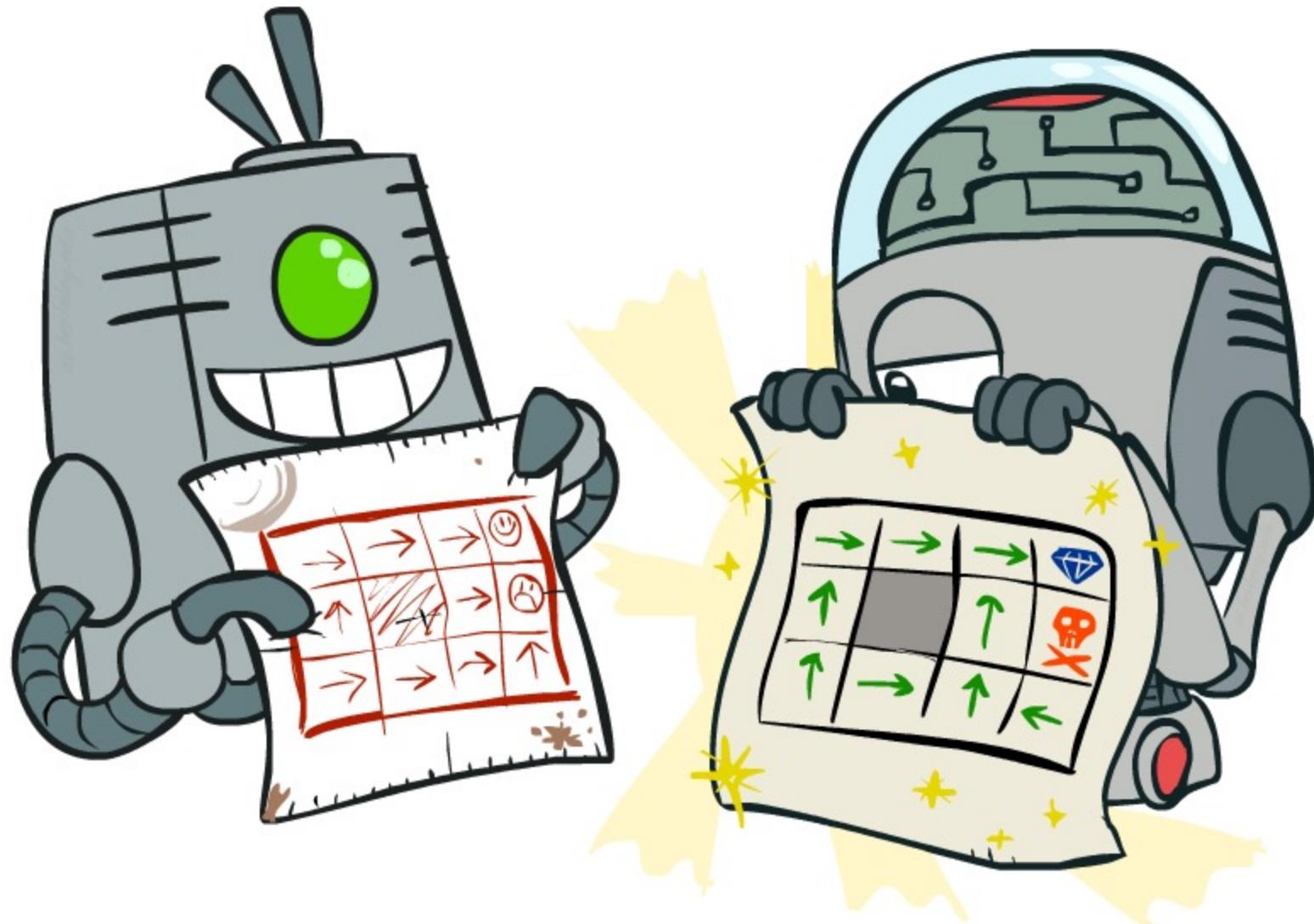
- Case 1: If the tree has maximum depth  $M$ , then  $V_M$  holds the actual untruncated values

- Case 2: If the discount is less than 1

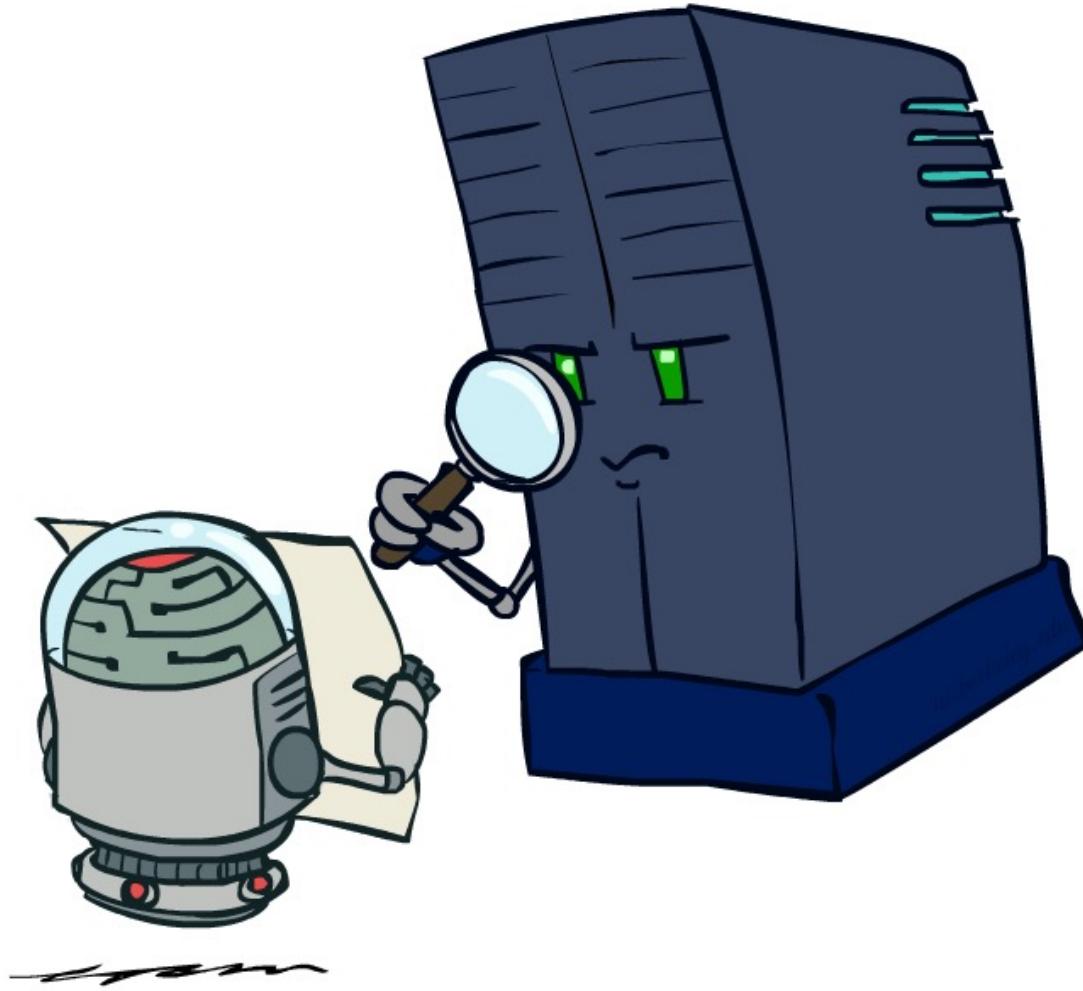
- Sketch: For any state  $V_k$  and  $V_{k+1}$  can be viewed as depth  $k+1$  expectimax results in nearly identical search trees
  - The difference is that on the bottom layer,  $V_{k+1}$  has actual rewards while  $V_k$  has zeros
  - That last layer is at best all  $R_{MAX}$
  - It is at worst  $R_{MIN}$
  - 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



# Policy Methods

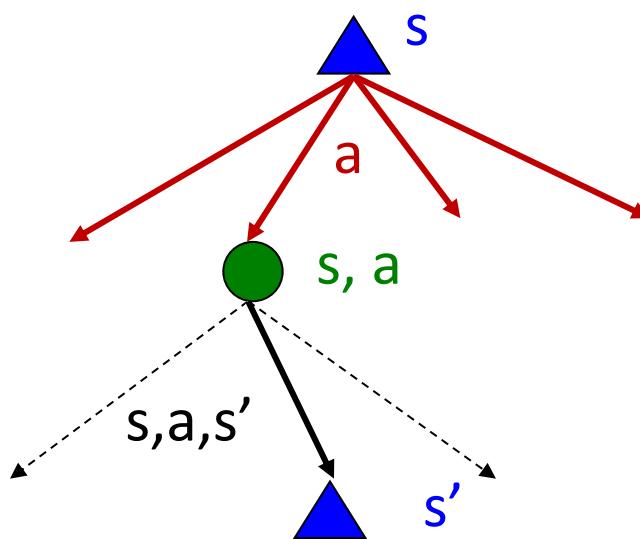


# Policy Evaluation

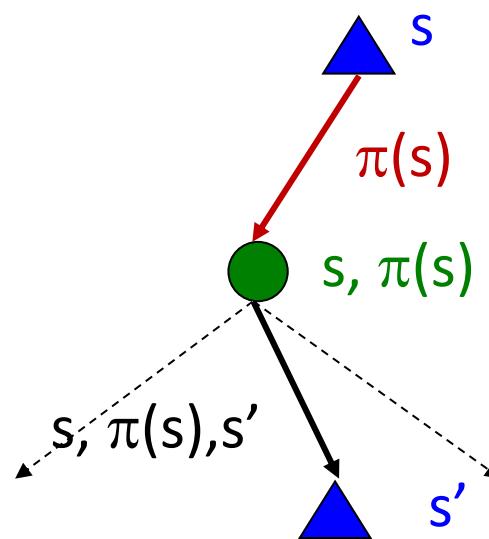


# Fixed Policies

Do the optimal action



Do what  $\pi$  says to do

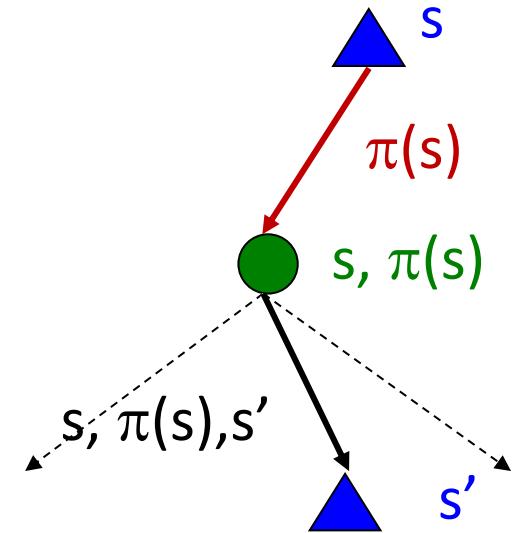


- ❑ Expectimax trees max over all actions to compute the optimal values
- ❑ If we fixed some policy  $\pi(s)$ , then the tree would be simpler – only one action per state
  - ... though the tree's value would depend on which policy we fixed

# Utilities for a Fixed Policy

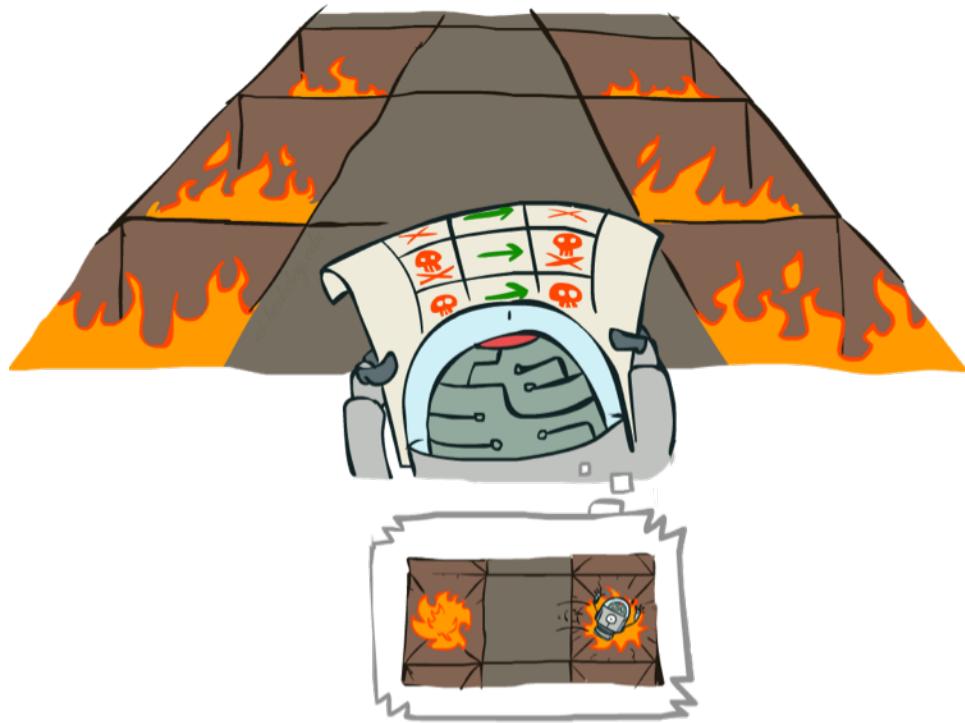
- ❑ Another basic operation: compute the utility of a state  $s$  under a fixed (generally non-optimal) policy
- ❑ Define the utility of a state  $s$ , under a fixed policy  $\pi$ :  
 $V^\pi(s)$  = expected total discounted rewards starting in  $s$  and following  $\pi$
- ❑ Recursive relation (one-step look-ahead / Bellman equation):

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

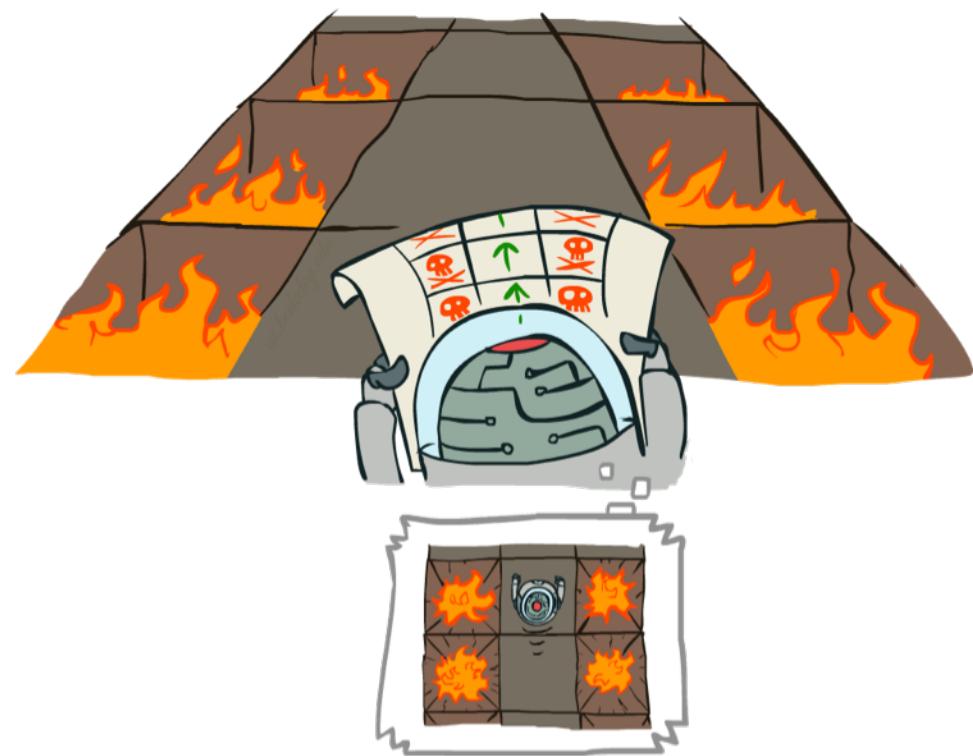


# Example: Policy Evaluation

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 maxes, the Bellman equations are just a linear system
  - Solve with Matlab (or your favorite linear system solver)

