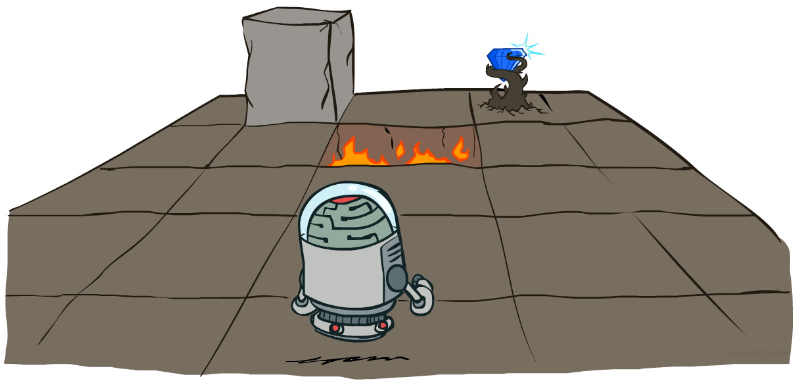
CS 573: Artificial Intelligence

Markov Decision Processes



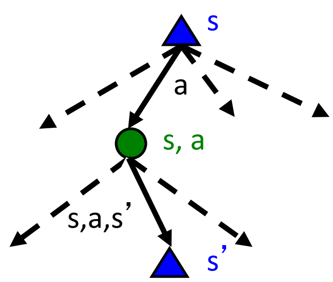
Dan Weld

University of Washington

Slides by Dan Klein & Pieter Abbeel / UC Berkeley. (http://ai.berkeley.edu) and by Mausam & Andrey Kolobov

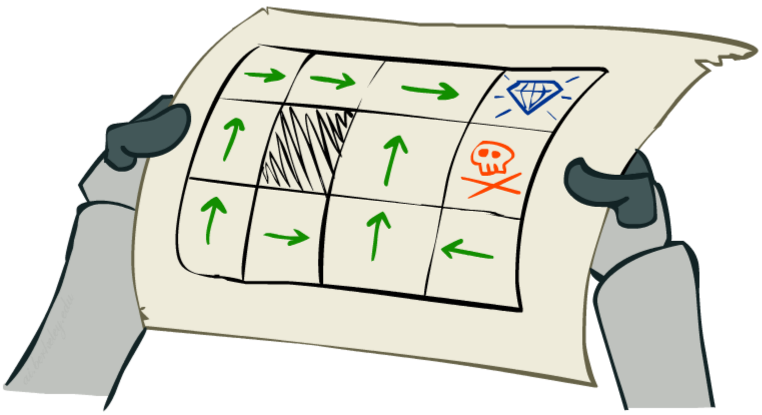
Recap: Defining MDPs

- 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 γ)



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

Solving MDPs



- Value Iteration
  - Asynchronous VI
- Policy Iteration
- Reinforcement Learning

V* = Optimal Value Function

The value (utility) of a state s:

$V^{*}(s)$

"expected utility starting in s & acting optimally forever"

Q*

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

$Q^*(s,a)$

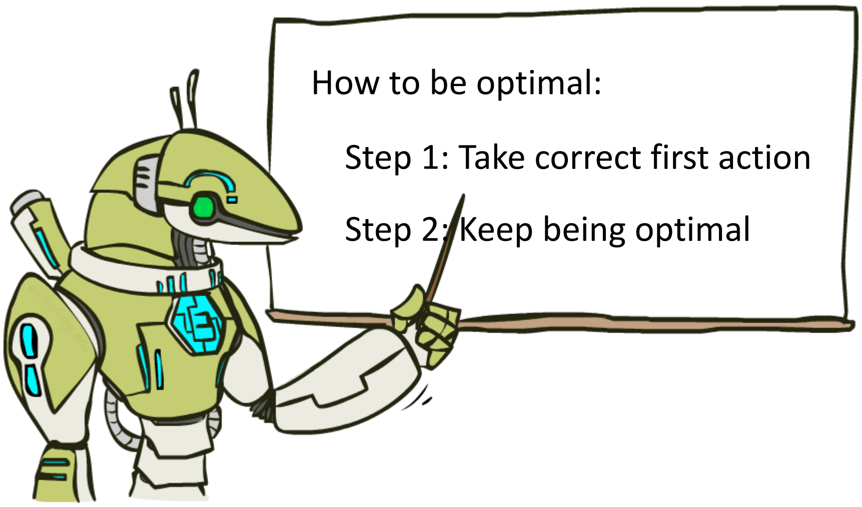
"expected utility of 1) starting in state ***s***
2) taking action ***a***
3) acting *optimally* forever after that"

Q*(s,a) = reward from executing a in s then ending in s'
plus… discounted value of V*(s')

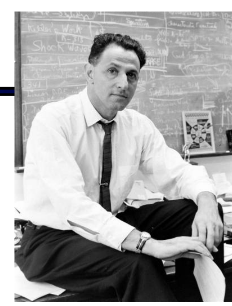
π* Specifies The Optimal Policy

$\pi^*(s)$ = optimal action from state s

The Bellman Equations



The Bellman Equations

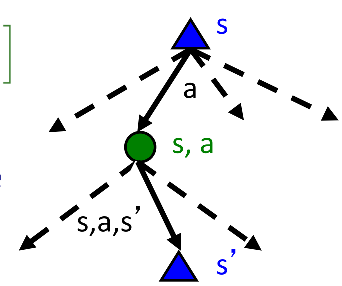


(1920-1984)

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

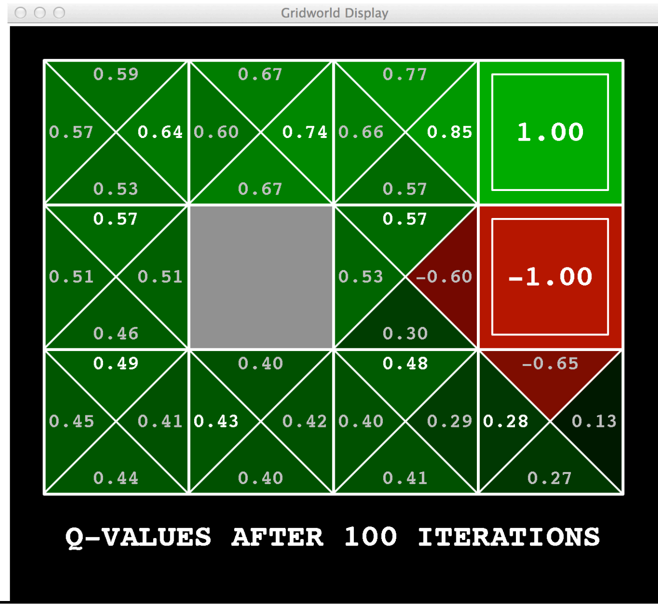
$$V^*(s) = \max_a Q^*(s,a)$$

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



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

Gridworld: Q*

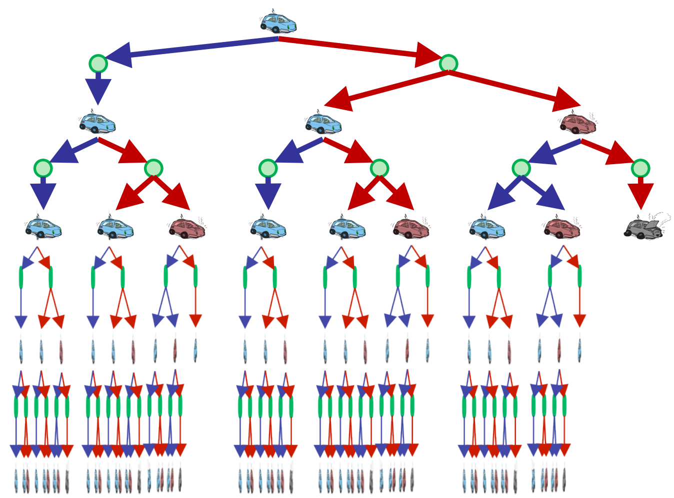


Gridworld Values V*

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

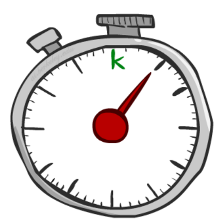
No End in Sight…

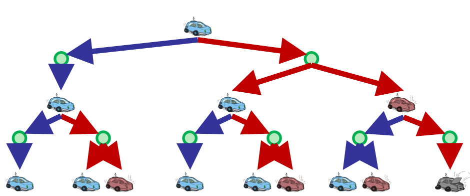
- We're doing way too much work with expectimax!
- Problem 1: States are repeated
  - Idea: Only compute needed quantities once
  - Like **graph search** (*vs.* tree search)
- Problem 2: Tree goes on forever
  - Rewards @ each step → **V changes**
  - Idea: Do a depth-limited computation, but with increasing depths until change is small
  - Note: deep parts of the tree *eventually* don't matter if $\gamma < 1$

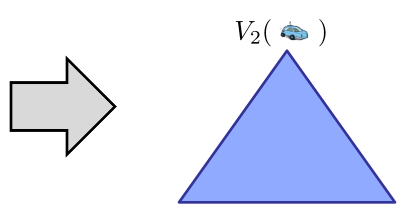


Time-Limited Values

- 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

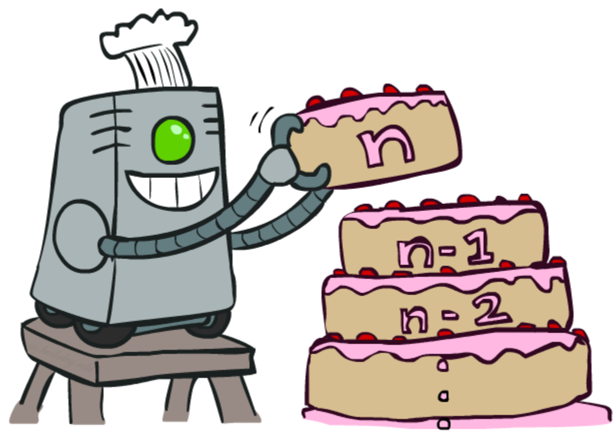






[Demo – time-limited values (L8D6)]

Value Iteration



Value Iteration

Called a "Bellman Backup"

- **Forall s, tialize $V_0(s) = 0$** *no time steps left means an expected reward of zero*
- **Repeat**

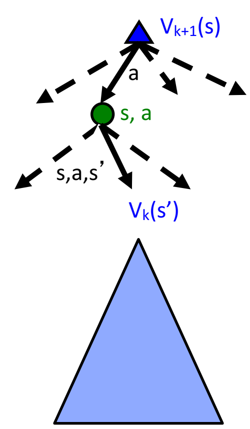
  K += 1

  $Q_{k+1}(s, a) = \Sigma_{s'} T(s, a, s') [R(s, a, s') + \gamma V_k(s')]$

  $V_{k+1}(s) = \text{Max}_a Q_{k+1}(s, a)$

  } do $\forall$ s, a

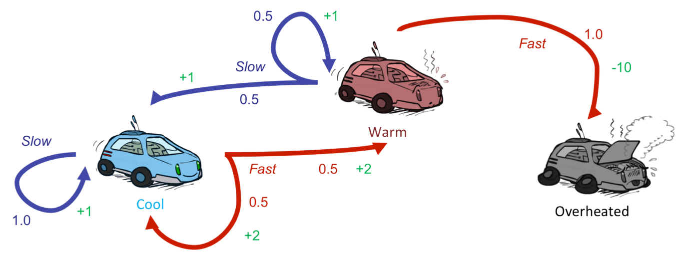
- **Repeat until $|V_{k+1}(s) - V_k(s)| < \varepsilon$, forall s** *"convergence"*



Successive approximation; dynamic programming

Example: Value Iteration

Assume no discount (gamma=1) to keep math simple!



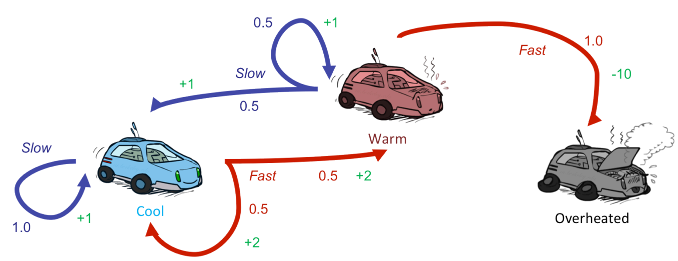
$Q_{k+1}(s, a) = \Sigma_{s'} T(s, a, s') [R(s, a, s') + \gamma V_k(s')]$

$V_{k+1}(s) = \text{Max}_a Q_{k+1} (s, a)$

Example: Value Iteration

Assume no discount (gamma=1) to keep math simple!

V_0	0	0	0
V_1			
V_2			



$Q_{k+1}(s, a) = \Sigma_{s'} T(s, a, s') [R(s, a, s') + \gamma V_k(s')]$

$V_{k+1}(s) = \text{Max}_a Q_{k+1} (s, a)$

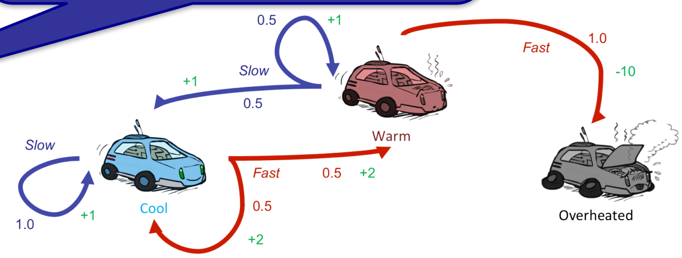
Example: Value Iteration

Q(,fast) =

Q(slow) =

math simple!

V_0	0	0	0
Q_1(s,a)=			0
V_1			0
V_2			



$Q_{k+1}(s, a) = \Sigma_{s'} T(s, a, s') [R(s, a, s') + \gamma V_k(s')]$

$V_{k+1}(s) = \text{Max}_a Q_{k+1}(s, a)$

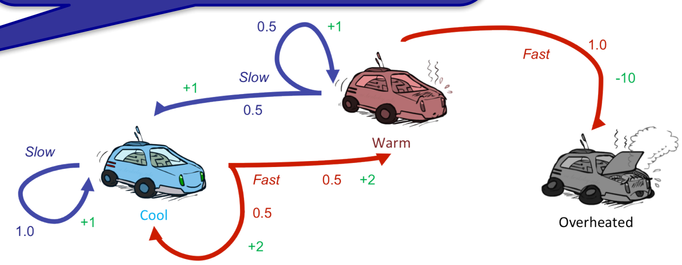
Example: Value Iteration

Q(,fast) = -10 + 0

Q(slow) = ½(1 + 0) + ½(1+0)

math simple!

V_0	0	0	0
Q_1(s,a)=		1, -10	0
V_1		1	0
V_2			



$Q_{k+1}(s, a) = \Sigma_{s'} T(s, a, s') [R(s, a, s') + \gamma V_k(s')]$

$V_{k+1}(s) = \text{Max}_a Q_{k+1}(s, a)$

Ex... n

Q(, fast) = ½(2 + 0) + ½(2 + 0)

Q(, slow) = 1*(1 + 0)

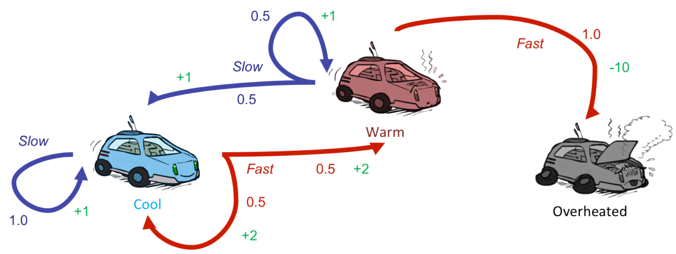
...ma=1) to keep math simple!

V_0: 0 0 0

$Q_1(s,a)$= 1, 2 1,-10 0

V_1: 2 1 0

V_2:



$$Q_{k+1}(s, a) = \Sigma_{s'} T(s, a, s') [R(s, a, s') + \gamma V_k(s')]$$

$$V_{k+1}(s) = \max_a Q_{k+1}(s, a)$$

Example: Value Iteration

Assume no discount (gamma=1) to keep math simple!

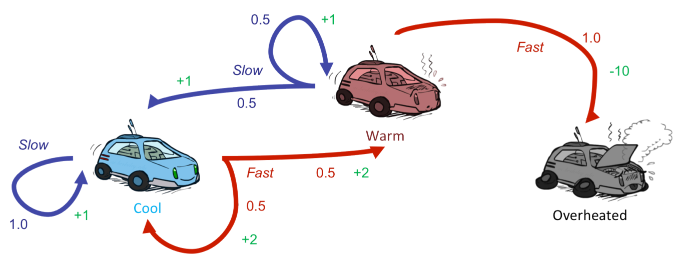
V_0: 0 0 0

$Q_1(s,a)$= 1, 2 1,-10 0

V_1: 2 1 0

$Q_2(s,a)$= 3,3.5 2.5,-10 0

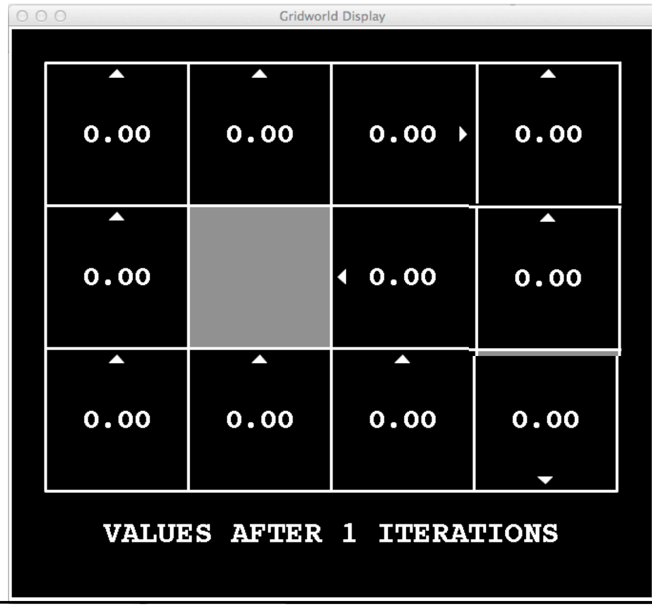
V_2: 3.5 2.5 0



$$Q_{k+1}(s, a) = \Sigma_{s'} T(s, a, s') [R(s, a, s') + \gamma V_k(s')]$$

$$V_{k+1}(s) = \max_a Q_{k+1}(s, a)$$

k=0



Noise = 0.2
Discount = 0.9
Living reward = 0

k=1

If agent is in 4,3, it only has one legal action: get jewel. It gets a reward and the game is over.
If agent is in the pit, it has only one legal action, die. It gets a penalty and the game is over.

Agent does NOT get a reward for moving INTO 4,3.

Noise = 0.2
Discount = 0.9
Living reward = 0

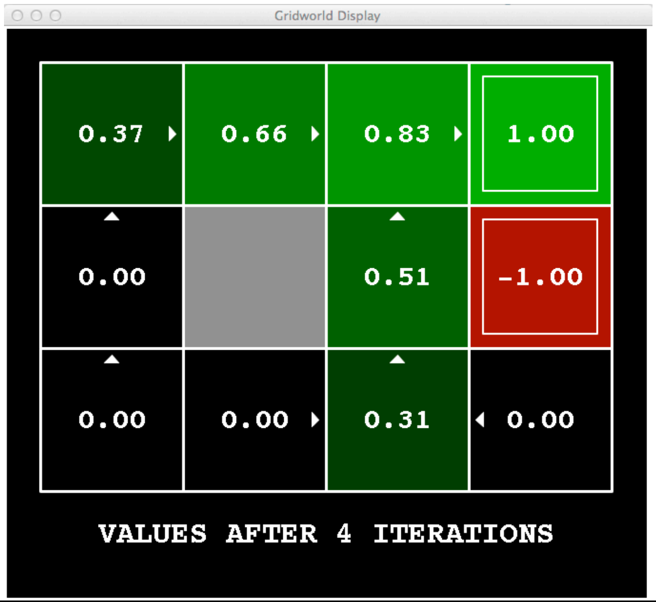
k=2

Noise = 0.2
Discount = 0.9
Living reward = 0

k=3

Noise = 0.2
Discount = 0.9
Living reward = 0

k=4

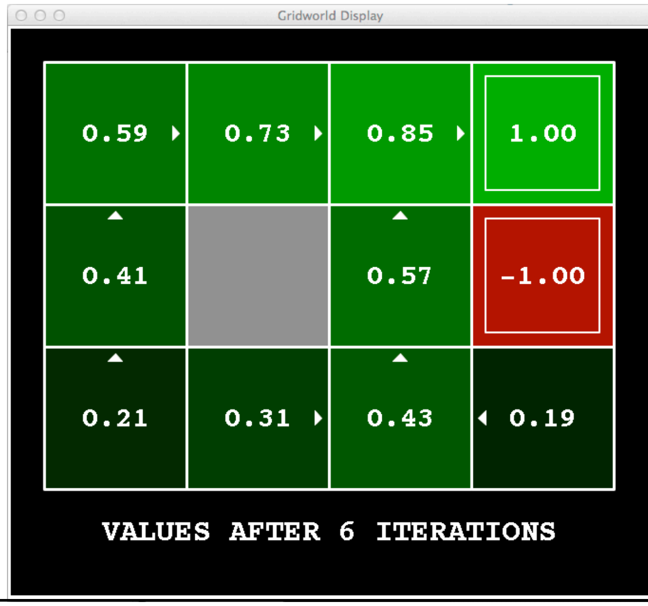


Noise = 0.2
Discount = 0.9
Living reward = 0

k=5

Noise = 0.2
Discount = 0.9
Living reward = 0

k=6

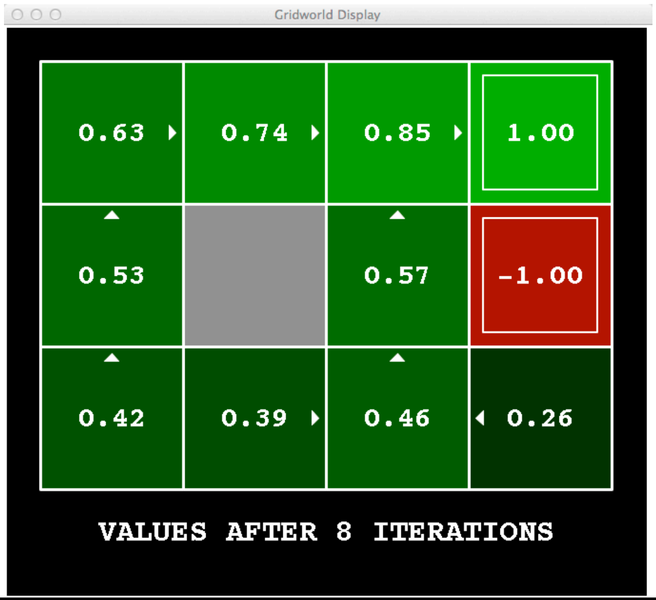


Noise = 0.2
Discount = 0.9
Living reward = 0

k=7

Noise = 0.2
Discount = 0.9
Living reward = 0

k=8

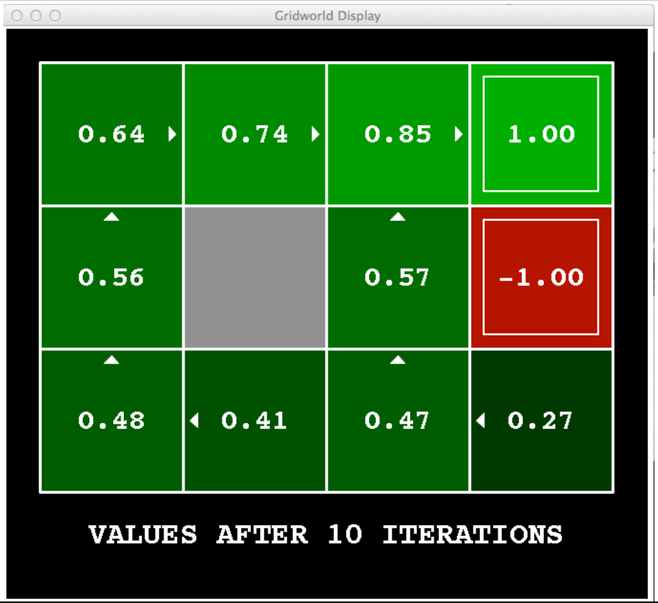


Noise = 0.2
Discount = 0.9
Living reward = 0

k=9

Noise = 0.2
Discount = 0.9
Living reward = 0

k=10

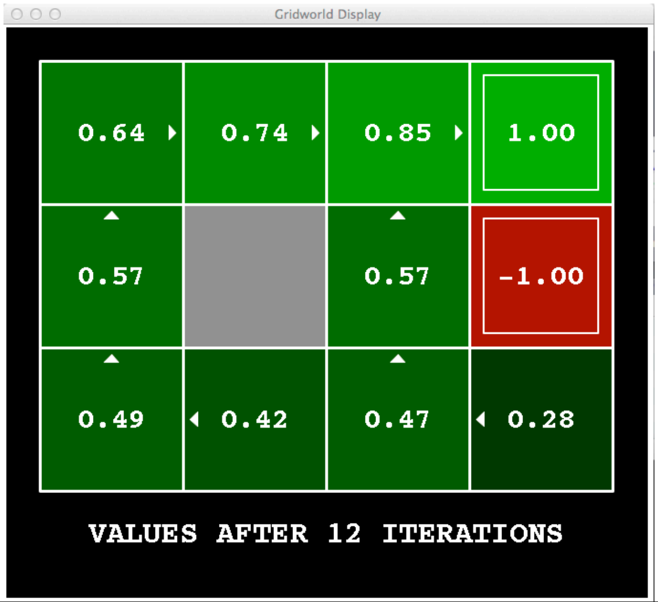


Noise = 0.2
Discount = 0.9
Living reward = 0

k=11

Noise = 0.2
Discount = 0.9
Living reward = 0

k=12

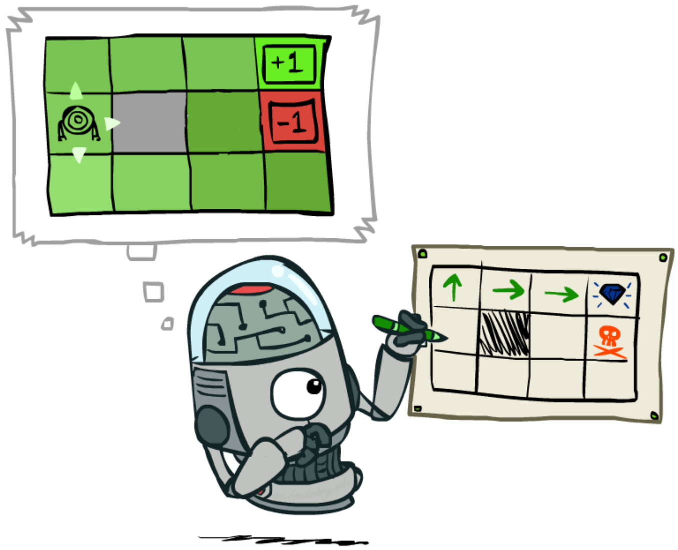


Noise = 0.2
Discount = 0.9
Living reward = 0

k=100

Noise = 0.2
Discount = 0.9
Living reward = 0

VI: Policy Extraction

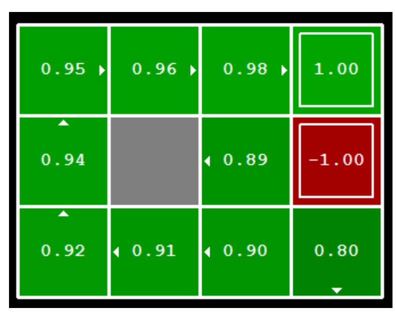


Computing Actions from Values

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

- How should we act?
  - In general, it's not obvious!

- We need to do a mini-expectimax (one step)

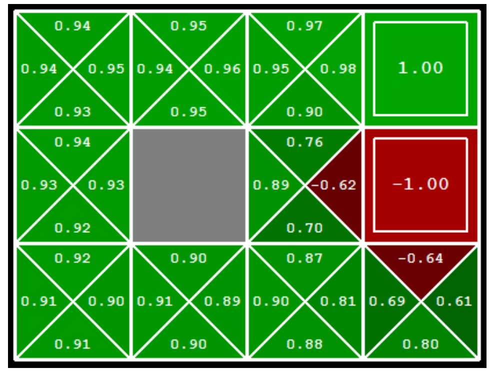


$$\pi^*(s) = \arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

- 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!

Value Iteration - Recap

- **Forall s, Initialize $V_0(s) = 0$** *no time steps left means an expected reward of zero*
- **Repeat** *do Bellman backups*

  K += 1

  Repeat for all states, s, and all actions, a:

  $Q_{k+1}(s, a) = \Sigma_{s'} T(s, a, s') [R(s, a, s') + \gamma V_k(s')]$

  $V_{k+1}(s) = \text{Max}_a Q_{k+1}(s, a)$

- **Until $|V_{k+1}(s) – V_k(s)| < \varepsilon$, forall s** *"convergence"*
- **Theorem: will converge to unique optimal values**