Intro to Reinforcement Learning

Yuchu Luo

Computer Animation and Multimedia Analysis LAB
Summer Machine Learning Class

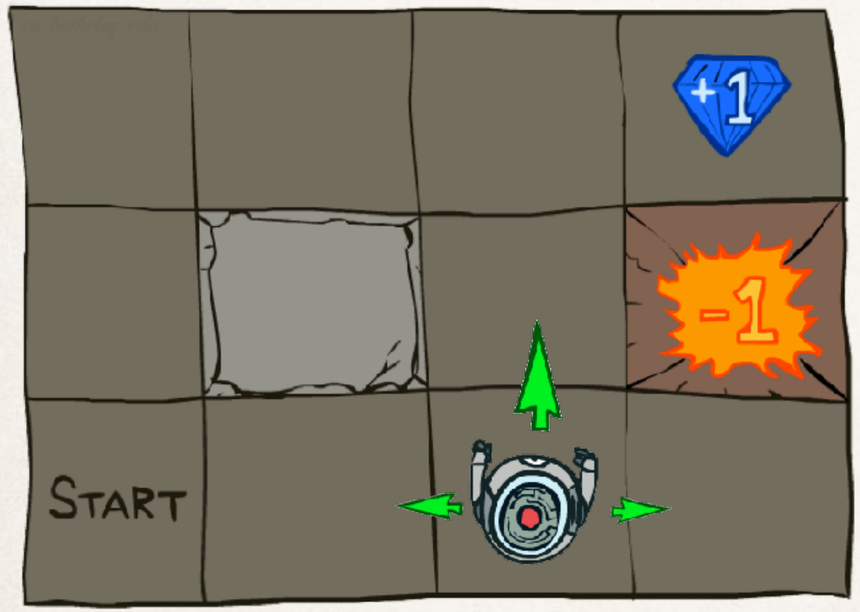
Thursday, 23 August 2017

Supervised Learning: $y = f(x)$ Find f (function approximation)

Unsupervised Learning: $f(x)$ Find f (clusters description)

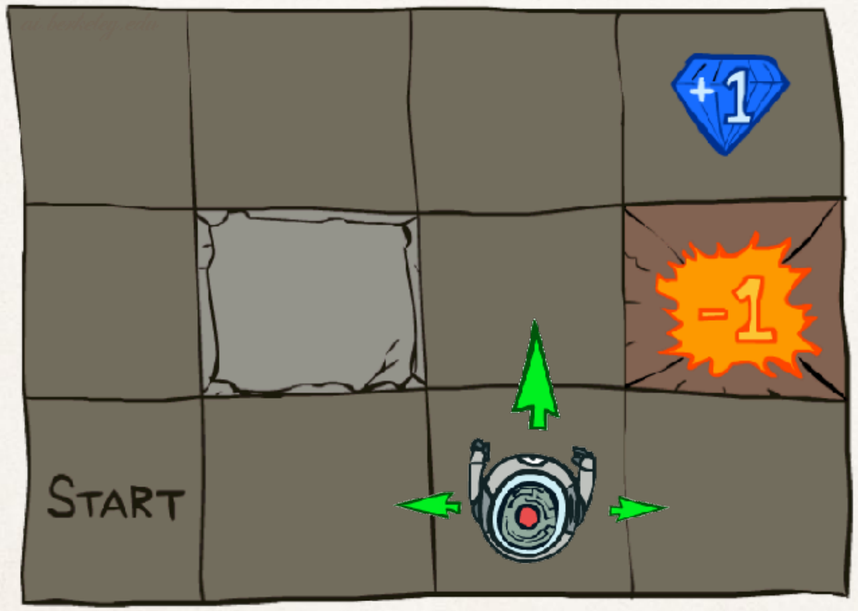
Reinforcement Learning: $y = f(x), z$ Find f (that generate y)

Example: The Grid World



- **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**

Markov Decision Processes



$$< S, A, R, P >$$

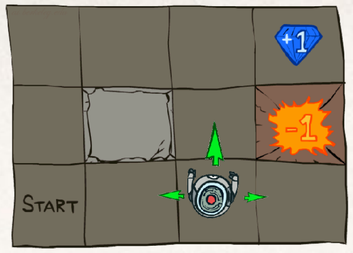
States: $s \in S$

Model: $T(s,a,s') \sim \Pr(s' \mid s,a)$

Actions: $a \in A$

Reward: $R(s,a) = E[R_{t+1} \mid s,a]$

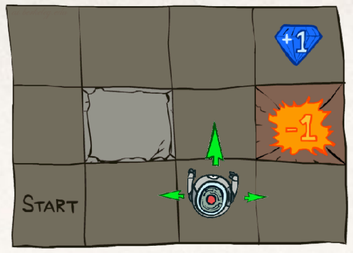
Policy: $\pi(s) \rightarrow a,\ \pi^*$



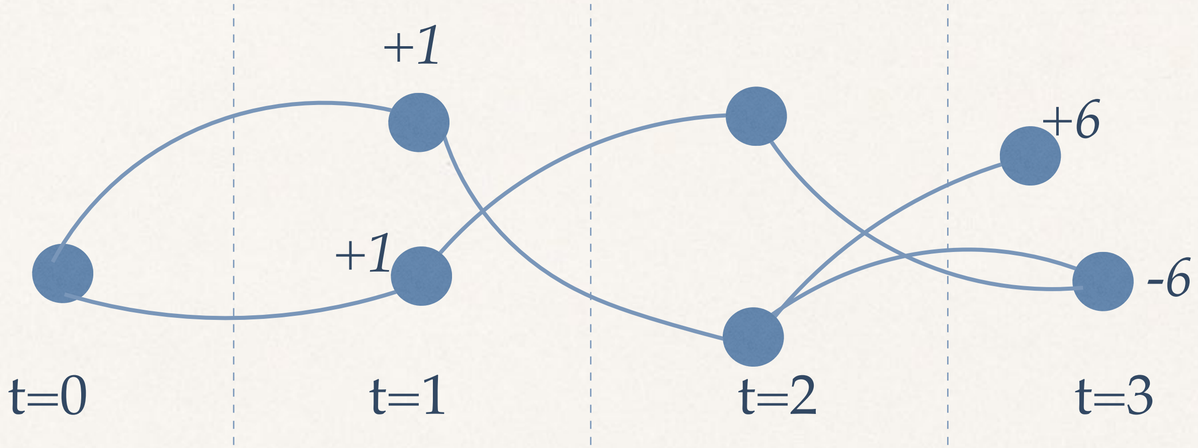
Markov Decision Processes

"Markov" means **Only Present Matter** and **Stationary Distribution**

$$P(S_{t+1} = s' \mid S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1}, S_0 = s_0)$$
$$=$$
$$P(S_{t+1} = s' \mid S_t = s_t, A_t = a_t)$$



Markov **Reward** Processes



- delay reward
- minor changes matter

How to find the optimal plan (policy π^*)?

How to find the optimal plan (policy π^*)?

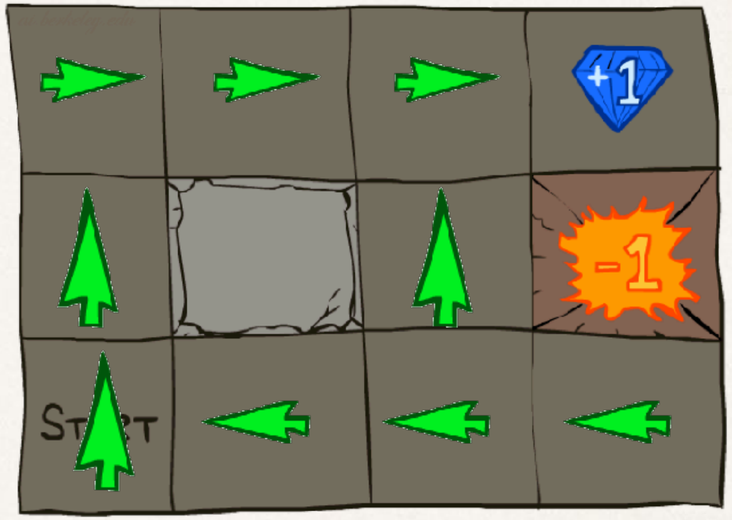
How to calculate the lone-term total reward?

$$\pi(s) \rightarrow a$$

Sequences of Rewards

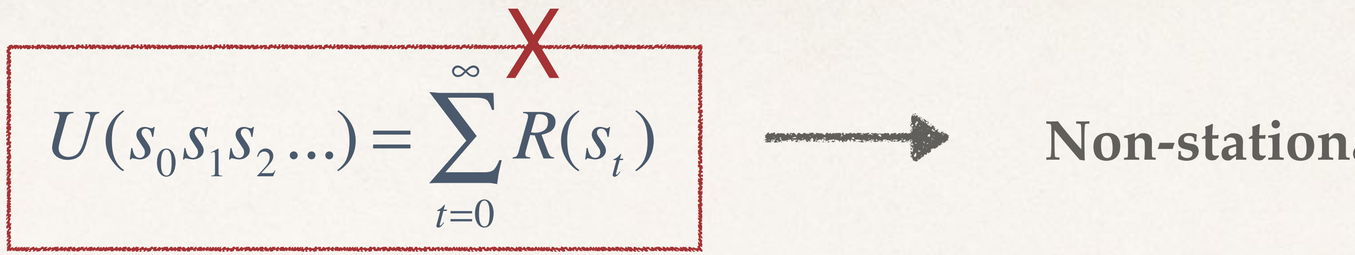
- Infinite Horizons
- Utility of Sequences

$$U(s_0 s_1 s_2 \ldots) = \sum_{t=0}^{\infty} R(s_t)$$

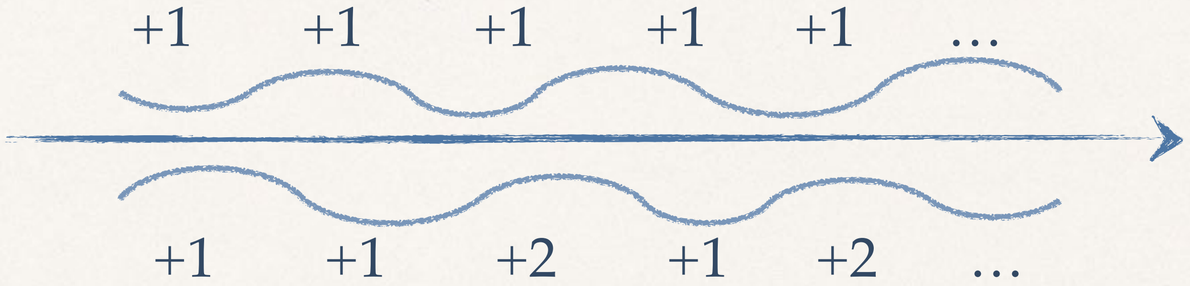


R(s) = -0.3

$$U(s_0 s_1 s_2 \ldots) = \sum_{t=0}^{\infty} R(s_t)$$



Non-stationarity



Discount $0 \le \gamma < 1$

$$U(s_0 s_1 s_2 \ldots) = \sum_{t=0}^{\infty} \gamma^t R(s_t) \le \sum_{t=0}^{\infty} \gamma^t R_{\max} = \frac{R_{\max}}{1-\gamma}$$

Recap: Defining MDPs

States: $s \in S$

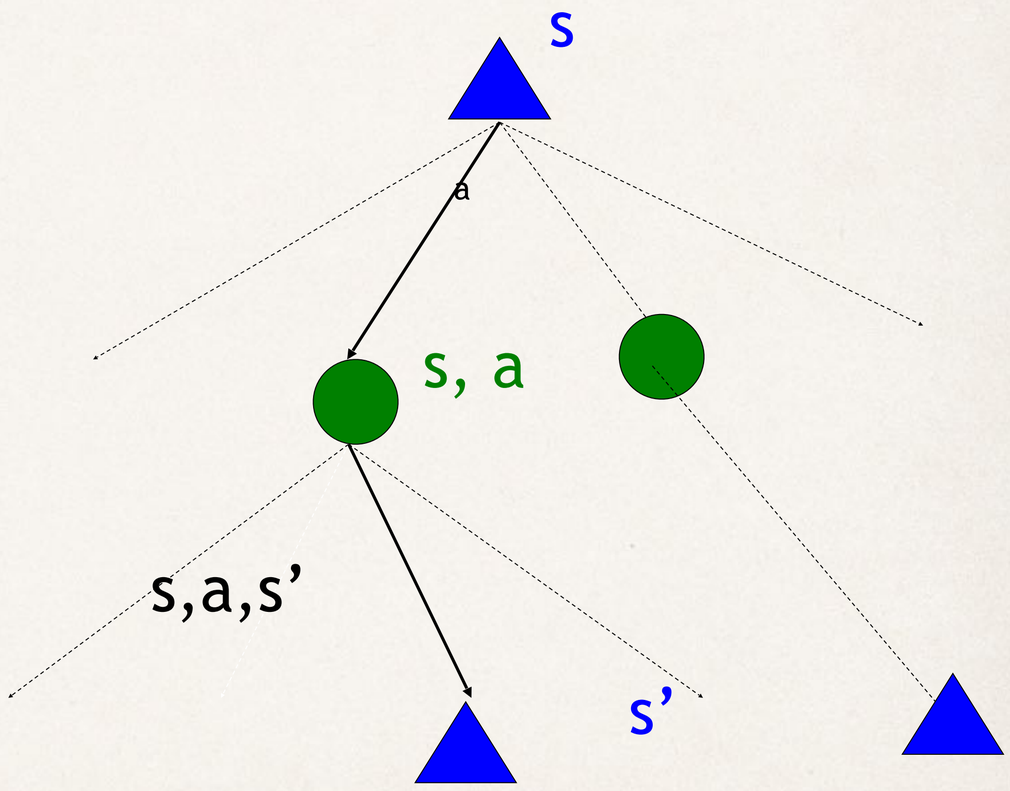
Model: $T(s,a,s') \sim \Pr(s' \mid s,a)$

Actions: $a \in A$

Reward: $R(s,a) = E[R_{t+1} \mid s,a]$

Policy: $\pi(s) \rightarrow a,\ \pi^*$

Utility: $U(s_0 s_1 s_2 \ldots) = \sum_{t=0}^{\infty} \gamma^t R(s_t)$



Solving MDPs

- The value (utility) of a state s:

  $\mathbf{V^*(s)}$ = expected utility starting in s and acting optimally

  $$V(s) = E[U_t \mid S_t = s]$$

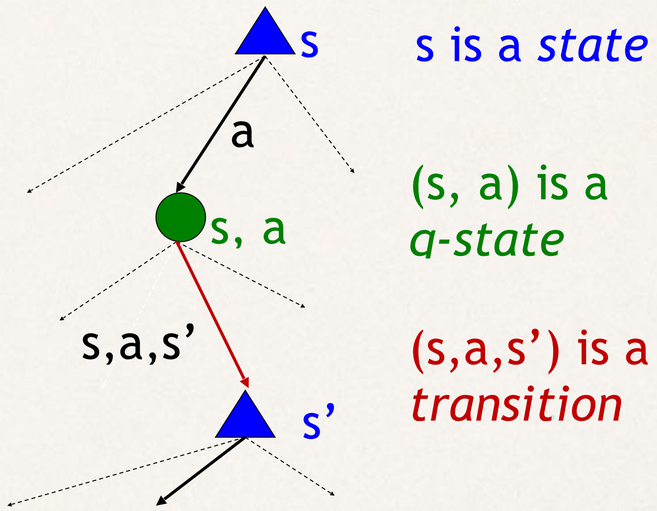
- The value (utility) of a q-state (s,a):

  $\mathbf{Q^*(s,a)}$ = expected utility starting out having taken action a from state s and (thereafter) acting optimally

  $$Q(s,a) = E[U_t \mid S_t = s, A_t = a]$$

- The optimal policy:

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

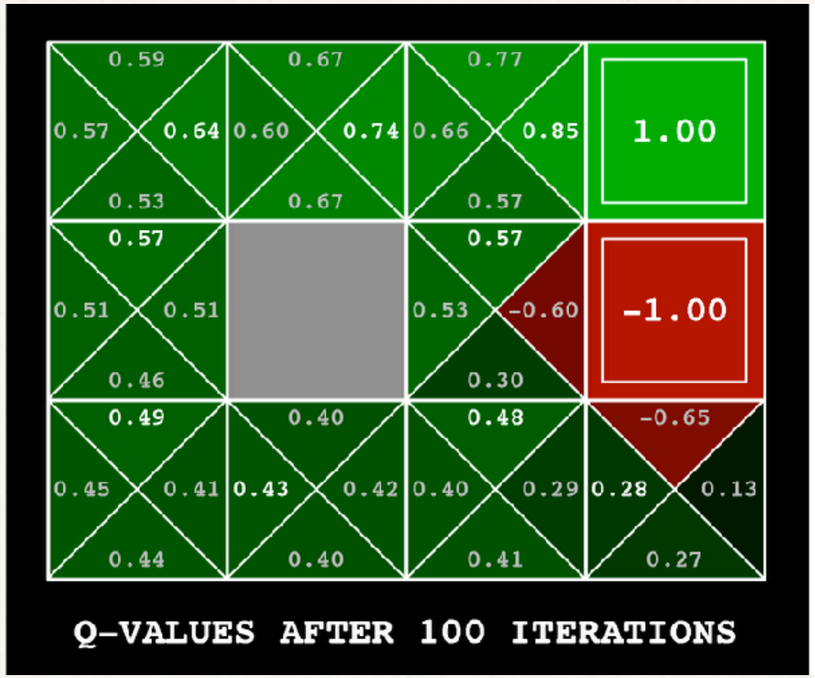


s is a *state*

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

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

State-Value Function



Q-Value Function

The Bellman Equations

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

$$Q^*(s,a) = E[U_t \mid S_t = s, A_t = a] = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \max_{a'} Q^*(s',a')$$

Value Iteration

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

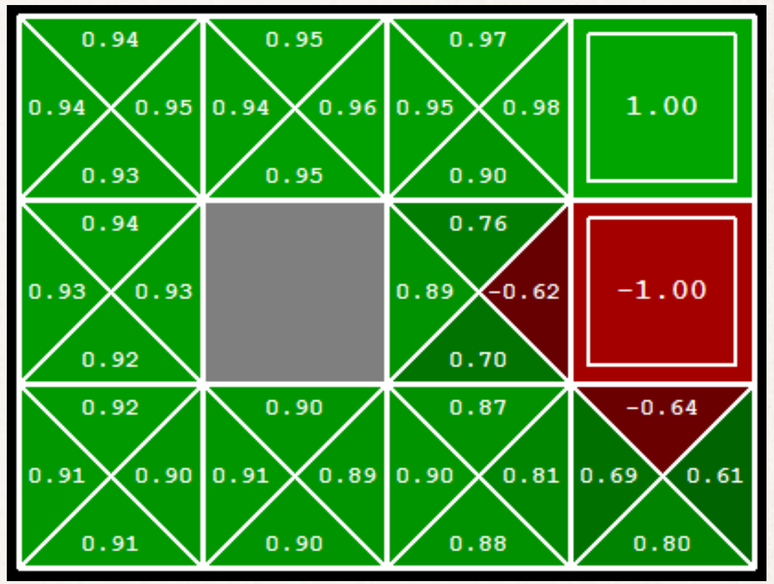
(Convergent)

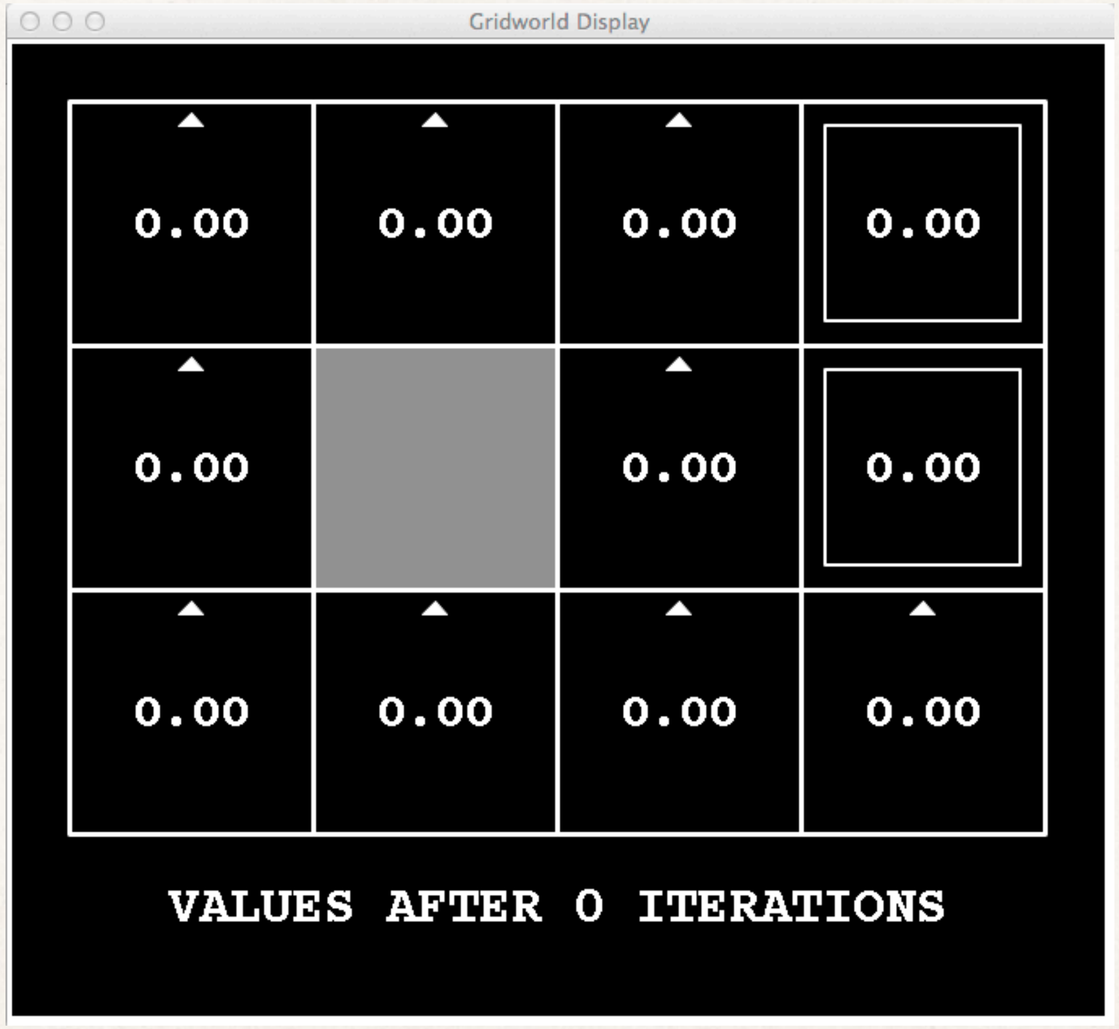
$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')]$$

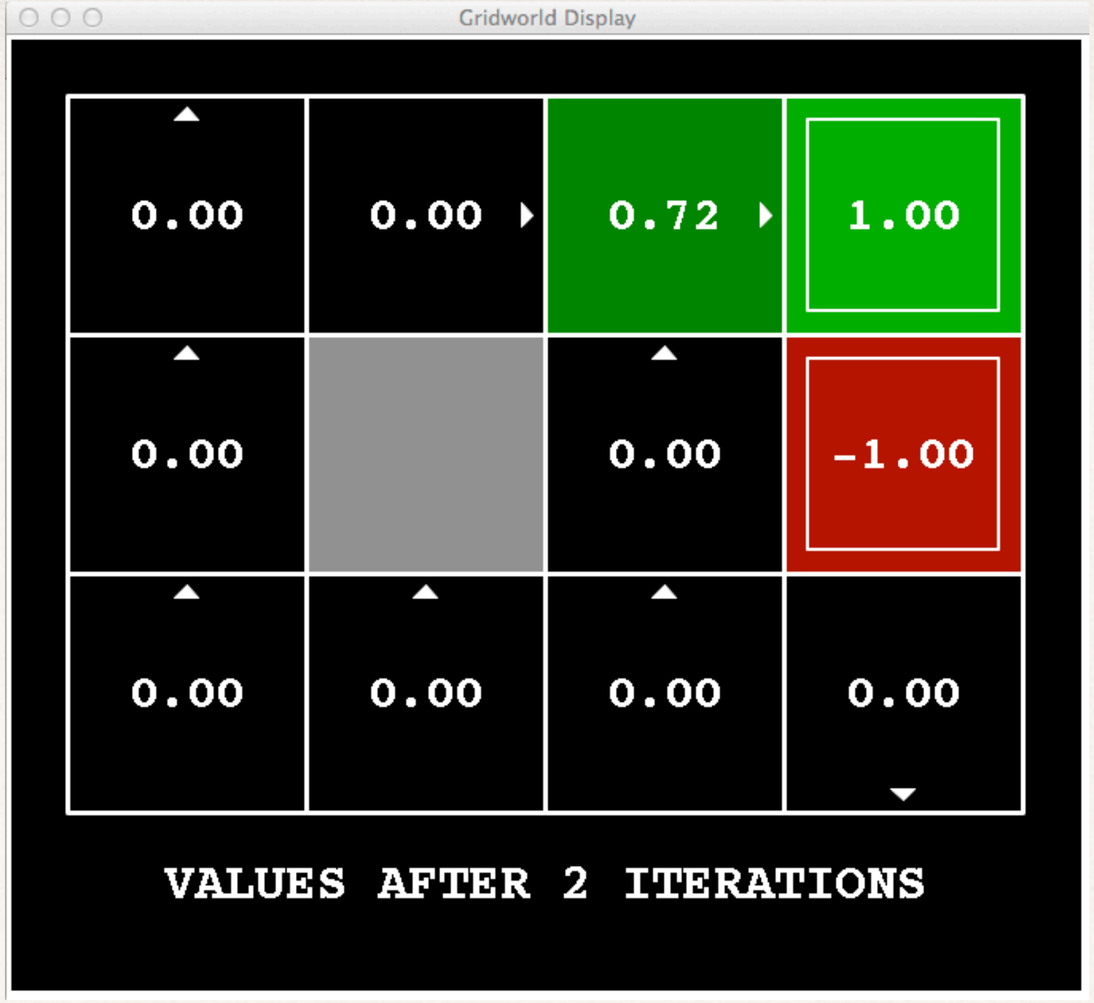
Finding Policies (via Value Iteration)

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

- Slow
- The "max" at each state rarely changes
- The policy often converges long before the values




Gridworld Display
0.00
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



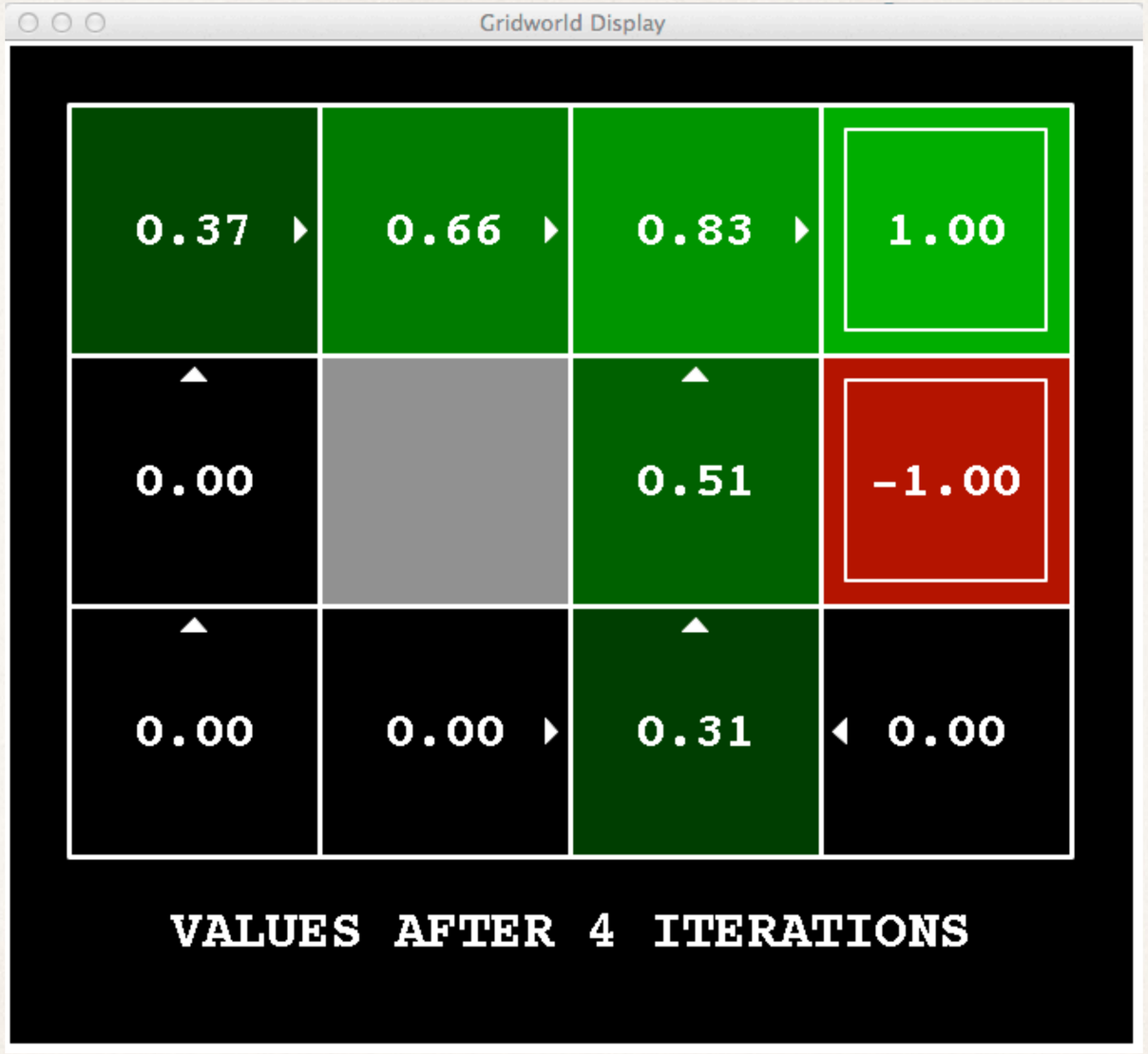
$$V_{k+1}(s) \leftarrow \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_k(s')]$$

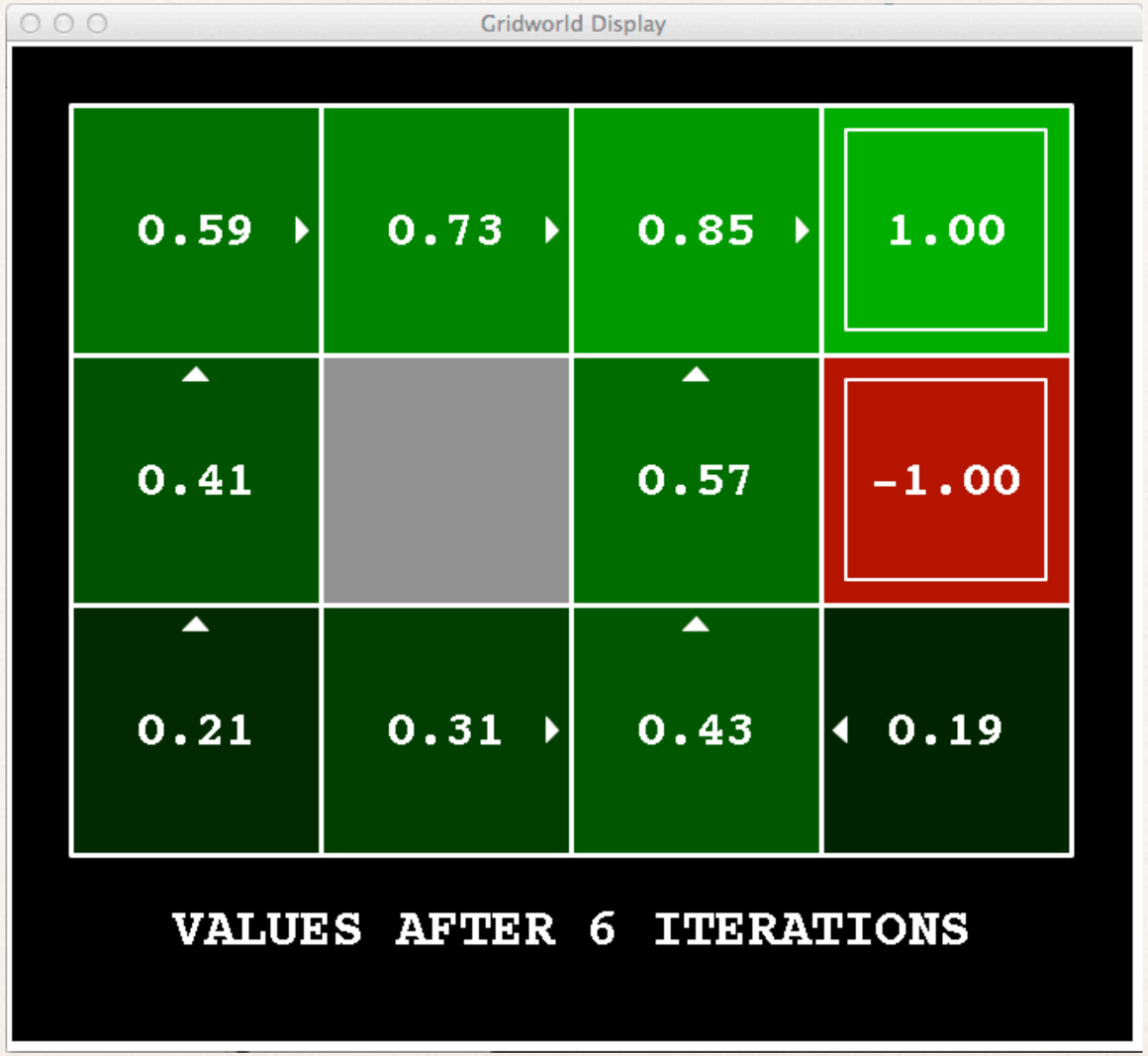
$$0.72 = 0.8*0.9*1.00 + 0.2*0.9*0.00$$

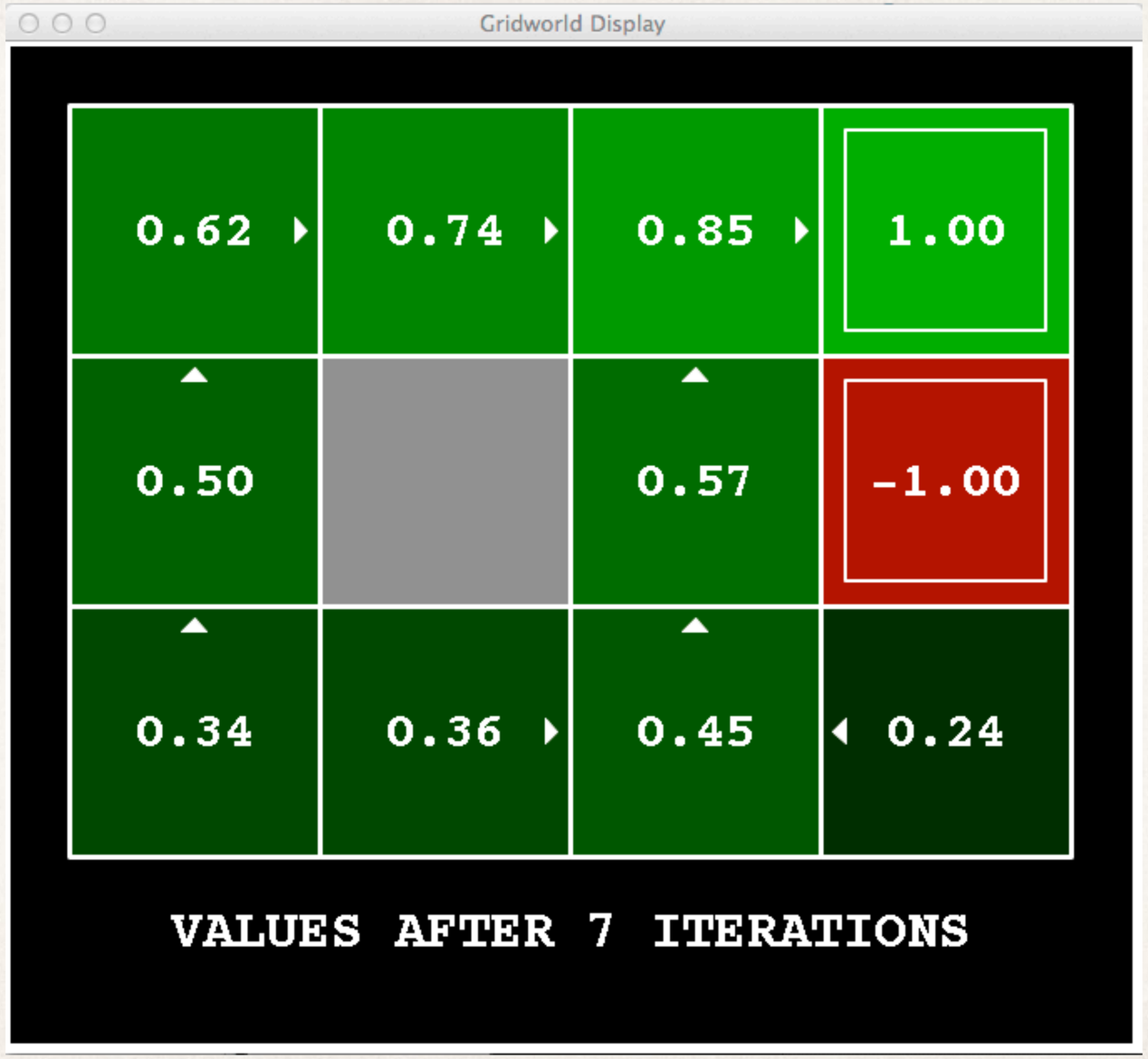
0.8, 0.2 from T(s,a,s'), 0.9 is discount

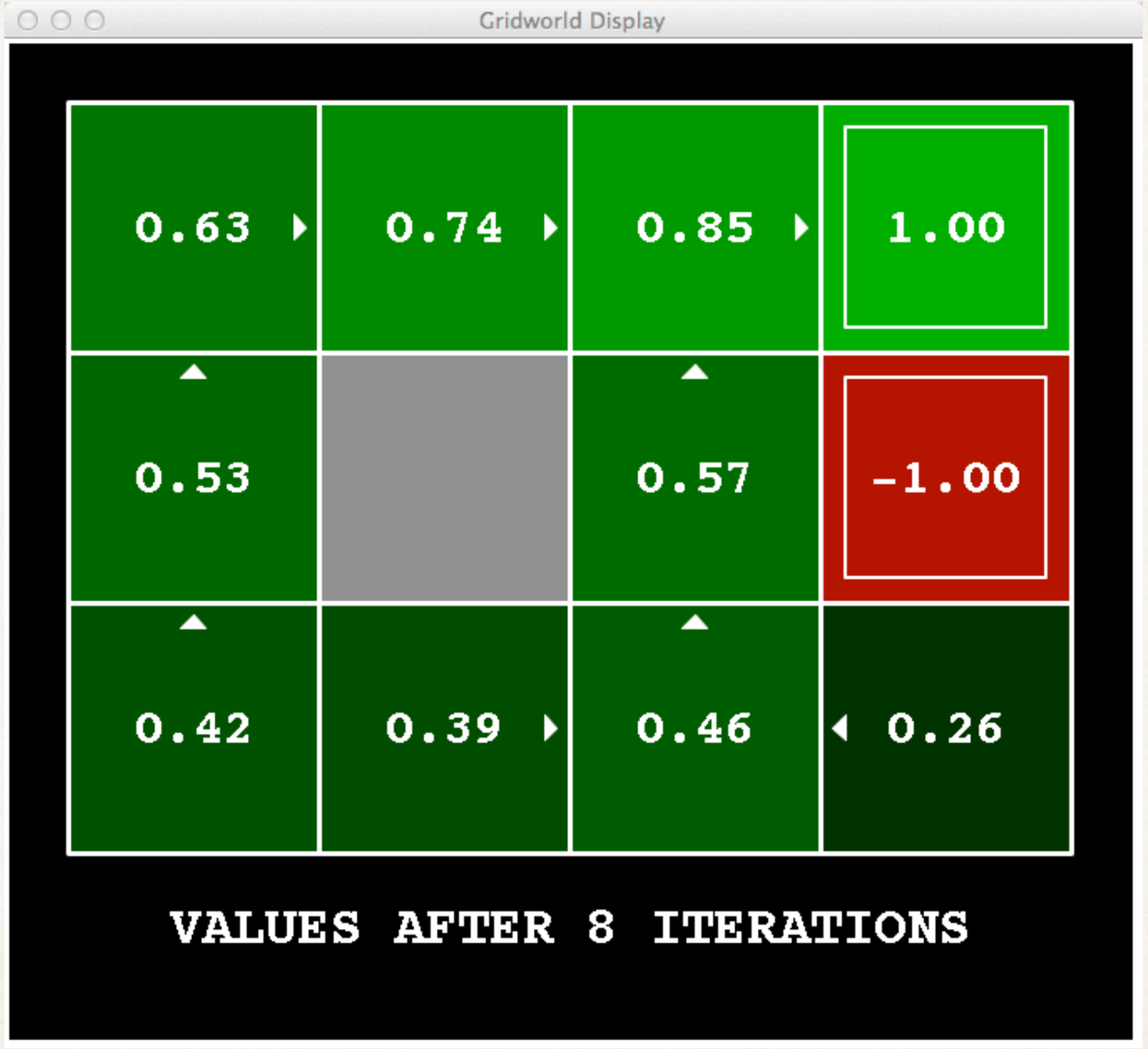
$$V_{k+1}(s) \leftarrow \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_k(s')]$$

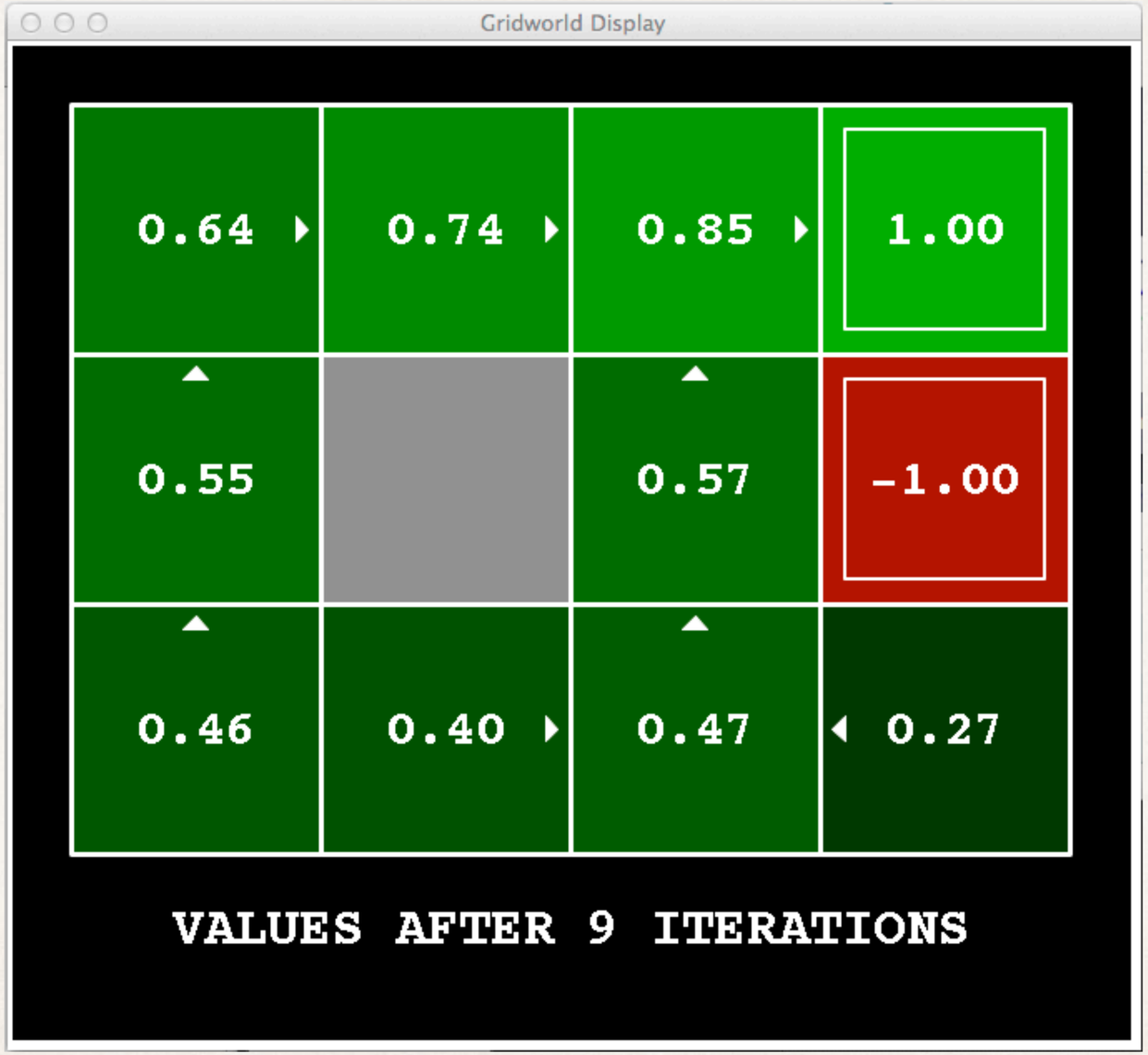
$$0.52 = 0.8*0.9*0.72 + 0.2*0.9*0.00$$

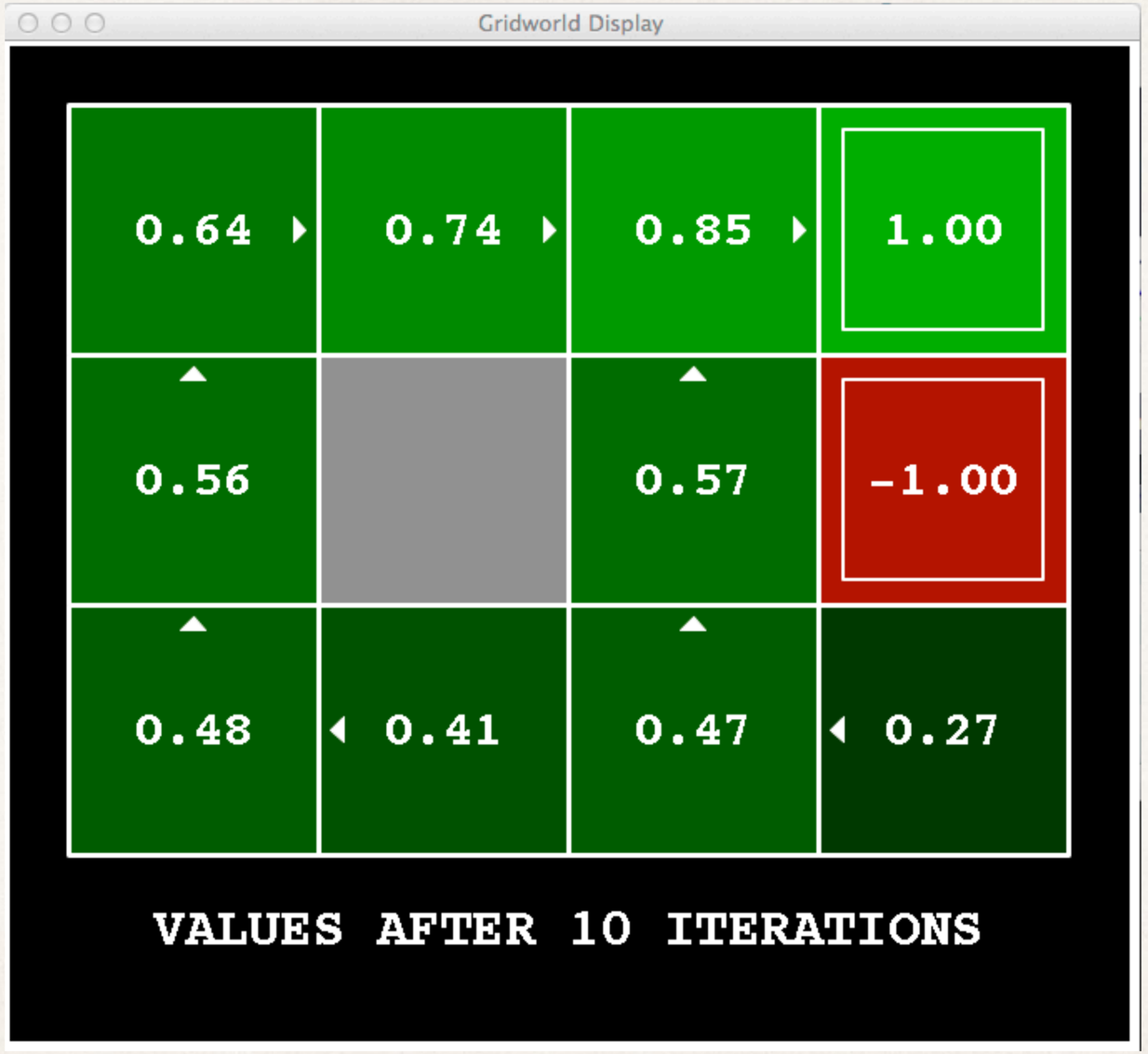

Gridworld Display
0.37
0.66
0.83
1.00
0.00
0.51
-1.00
0.00
0.00
0.31
0.00
VALUES AFTER 4 ITERATIONS

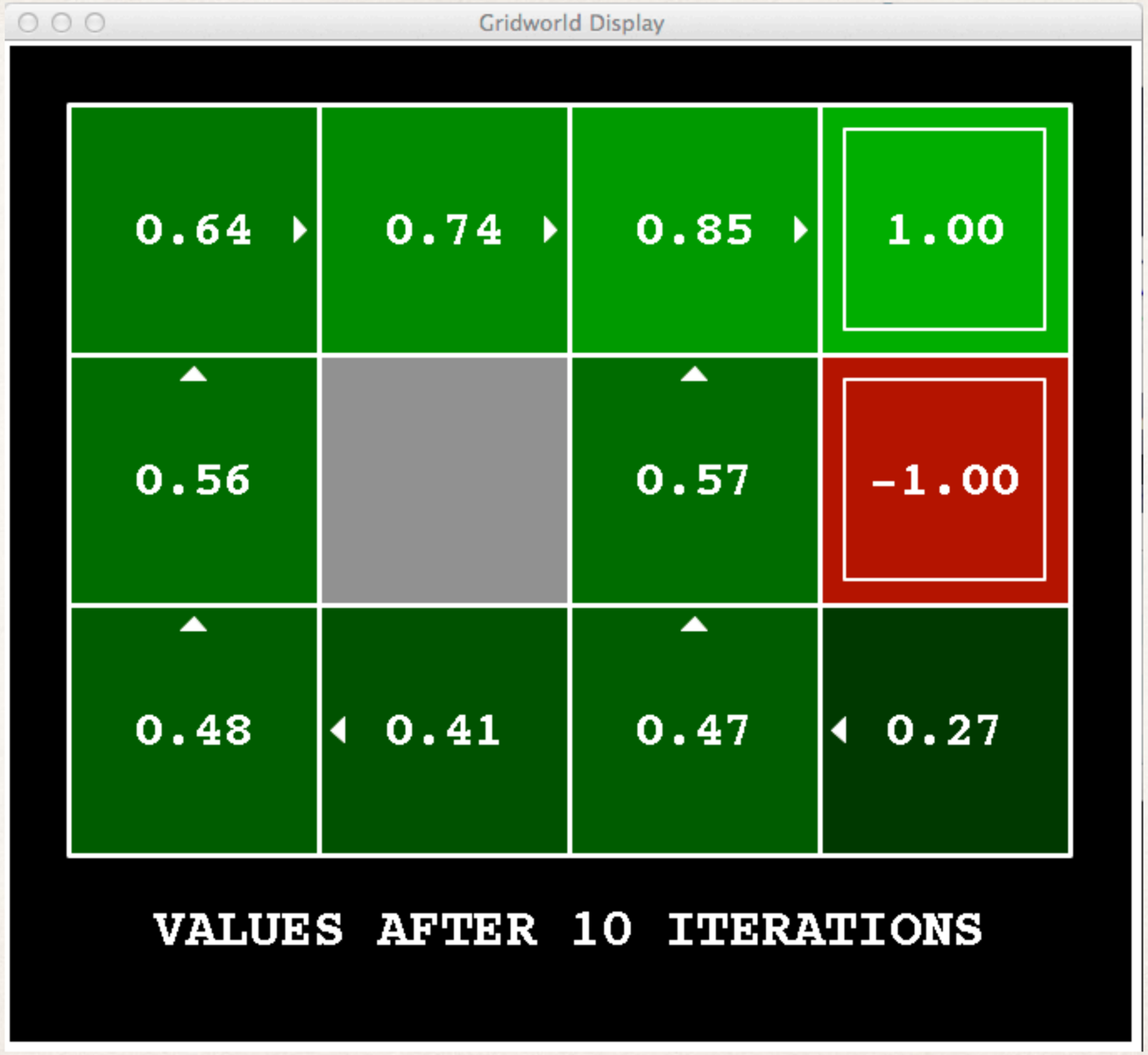

Gridworld Display
0.59
0.73
0.85
1.00
0.41
0.57
-1.00
0.21
0.31
0.43
0.19
VALUES AFTER 6 ITERATIONS

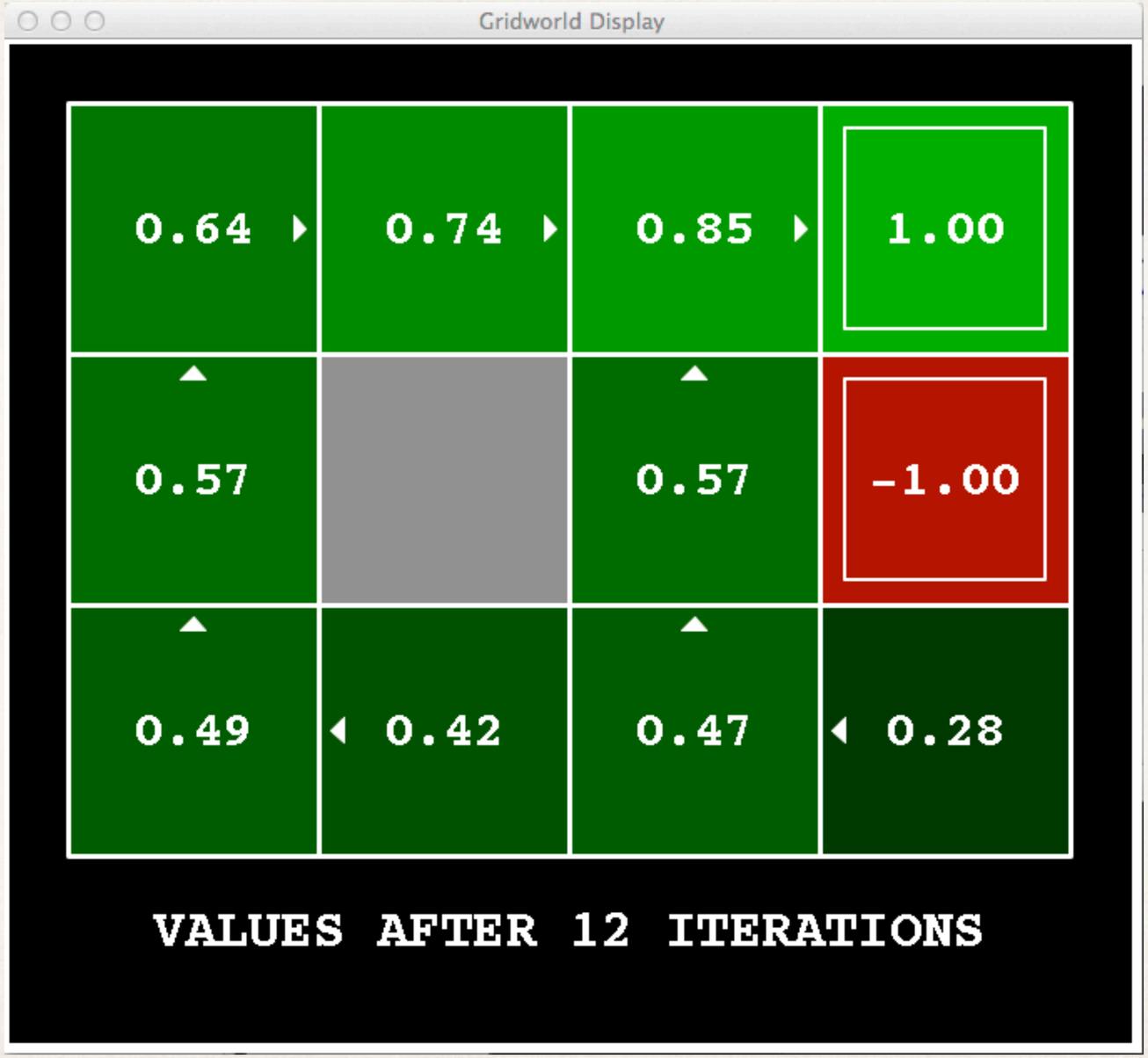

Gridworld Display
0.62
0.74
0.85
1.00
0.50
0.57
-1.00
0.34
0.36
0.45
0.24
VALUES AFTER 7 ITERATIONS

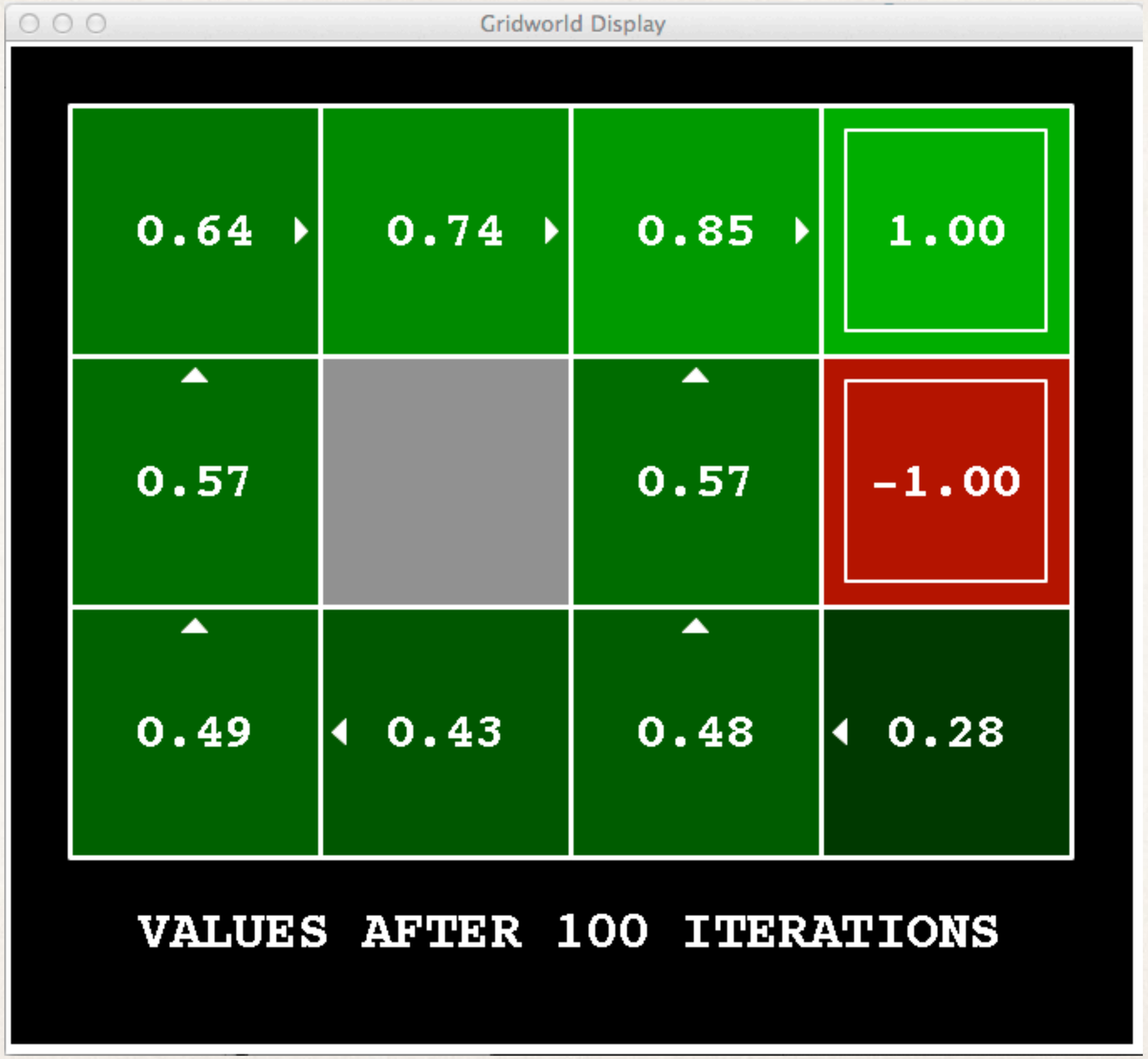

Gridworld Display
0.63
0.74
0.85
1.00
0.53
0.57
-1.00
0.42
0.39
0.46
0.26
VALUES AFTER 8 ITERATIONS


Gridworld Display
0.64
0.74
0.85
1.00
0.55
0.57
-1.00
0.46
0.40
0.47
0.27
VALUES AFTER 9 ITERATIONS


Gridworld Display
0.64
0.74
0.85
1.00
0.56
0.57
-1.00
0.48
0.41
0.47
0.27
VALUES AFTER 10 ITERATIONS


Gridworld Display
0.64
0.74
0.85
1.00
0.56
0.57
-1.00
0.48
0.41
0.47
0.27
VALUES AFTER 10 ITERATIONS


Gridworld Display
0.64
0.74
0.85
1.00
0.57
0.57
-1.00
0.49
0.42
0.47
0.28
VALUES AFTER 12 ITERATIONS


Gridworld Display
0.64
0.74
0.85
1.00
0.57
0.57
-1.00
0.49
0.43
0.48
0.28
VALUES AFTER 100 ITERATIONS

Policy Iteration

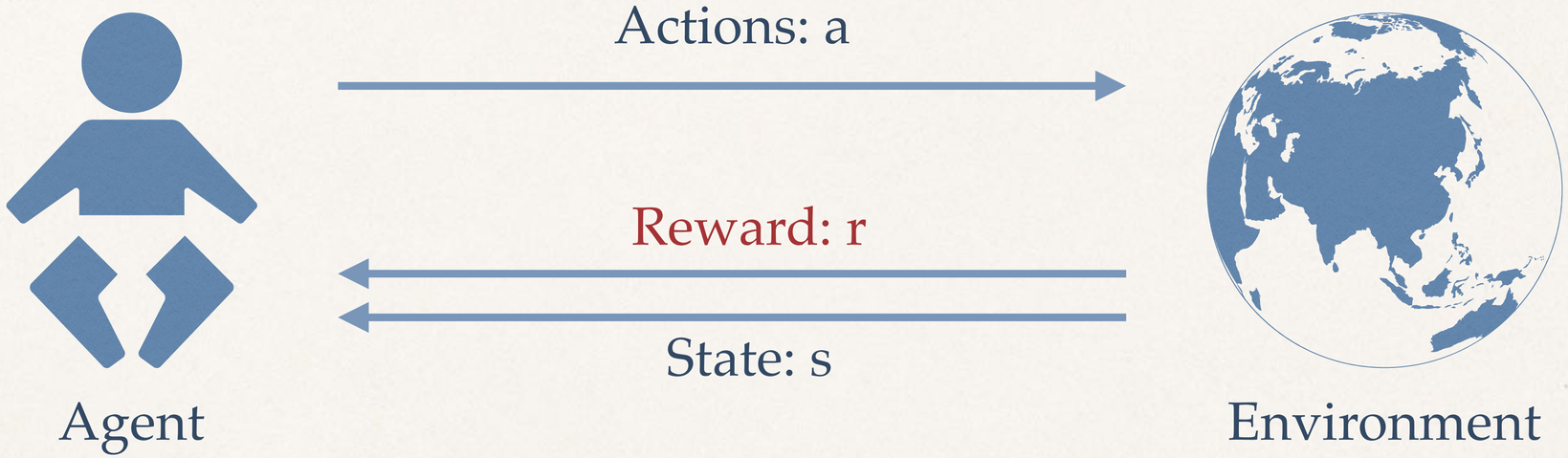
Step 1
Evaluation

Calculate utilities for
fixed policy

Step 2
Improvement

Select action when
Q-Value > State-Value

Reinforcement Learning



Example: Learning to Walk

Initial

[Kohl and Stone, ICRA 2004]

Example: Learning to Walk

Training

[Kohl and Stone, ICRA 2004]

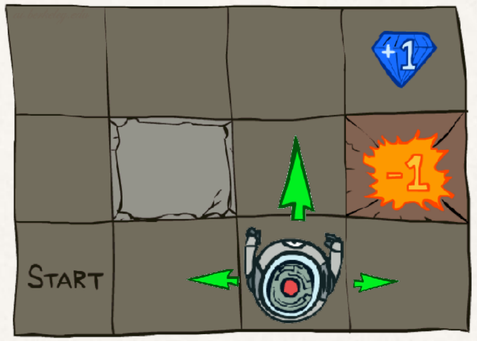
Example: Learning to Walk

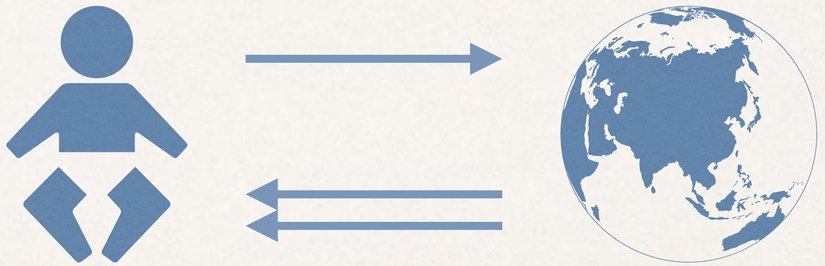
Finished

[Kohl and Stone, ICRA 2004]

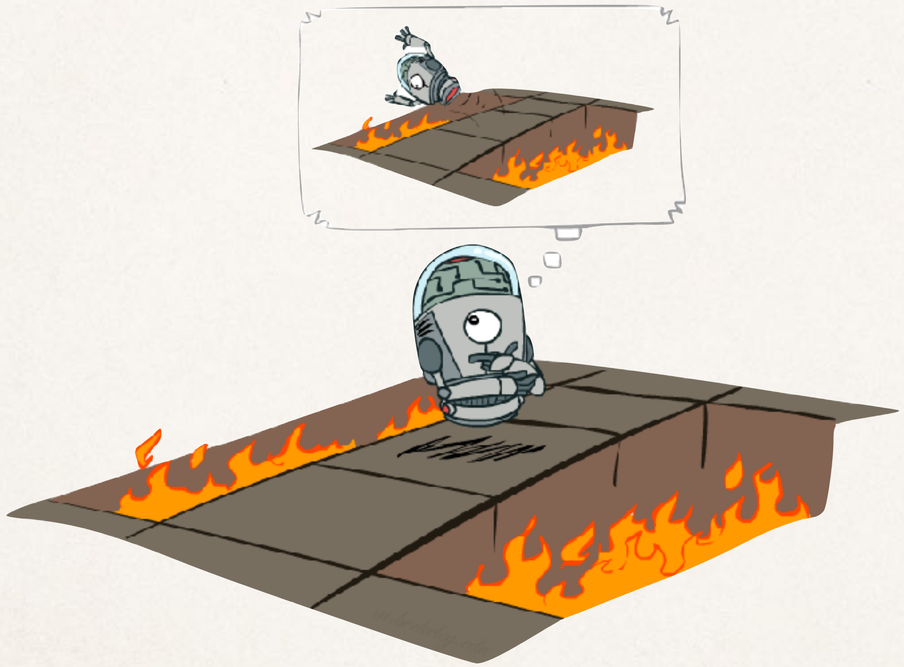
MDPs and RL

- Common ideas:
  - A set of states $s \in S$
  - A set of actions (per state) A
  - A model T(s,a,s')
  - A reward function R(s,a,s')
- Still looking for a policy $\pi(s)$
- New twist: don't know T or R
  - I.e. we don't know which states are good or what the actions do
  - Must actually try actions and states out to learn

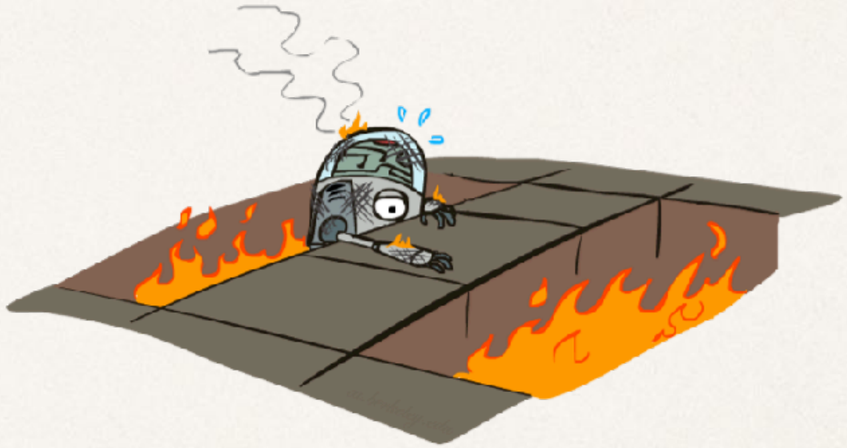




MDPs and RL



MDPs (Offline)



RL (Online)

Model-Based Learning

Step 1: learn a model of how the environment works from its observations

Learn $\hat{T}(s,a,s')$ and $\hat{R}(s,a,s')$ through (s, a, s′) pairs.

Step 2: plan a solution using that model

For example, use value iteration solve the learned MDP as before.

Example: Expected Age

Known P(A)

$$E[A] = \sum_a P(a) \cdot a = 0.3 \times 18 + \ldots$$

Without P(A), instead collect samples [a1, a2, … aN]

Unknown P(A): "Model Based"

$$\hat{P}(a) = \frac{num(a)}{N}$$

$$E[A] \approx \sum_a \hat{P}(a) \cdot a$$

Unknown P(A): "Model Free"

$$E[A] \approx \frac{1}{N} \sum_i a_i$$

Model-Free Learning

Q-Learning

In MDP:

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')]$$

In RL (Model-Free):

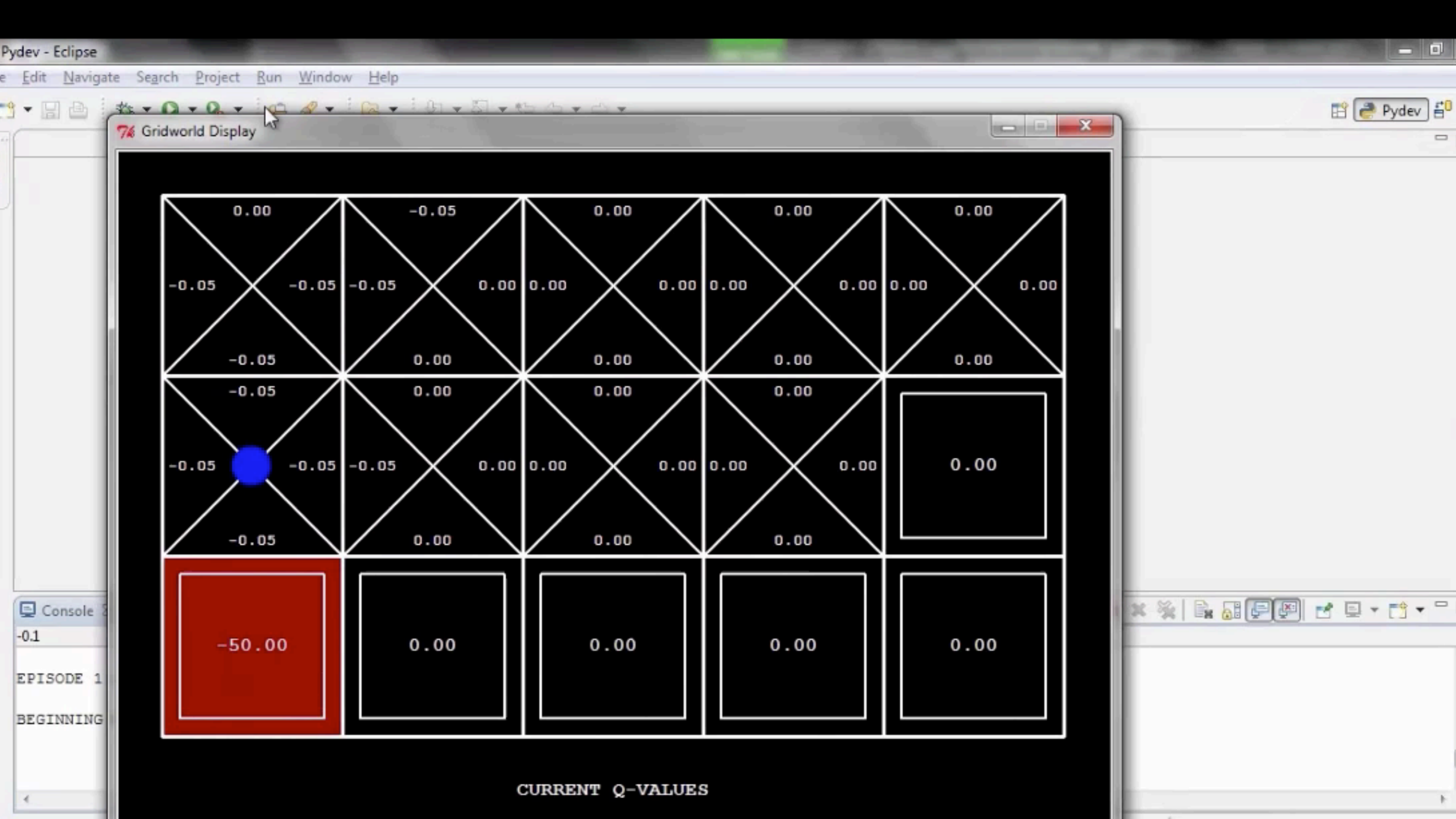
- Receive a sample (s,a,s',r)
- Consider your old estimate: Q(s,a)
- Consider your new sample estimate (sample suggest Q-value):

$$sample = Q_{suggest}(s,a) = R(s,a,s') + \gamma \max_{a'} Q(s',a')$$

- Incorporate the new estimate into a running average:

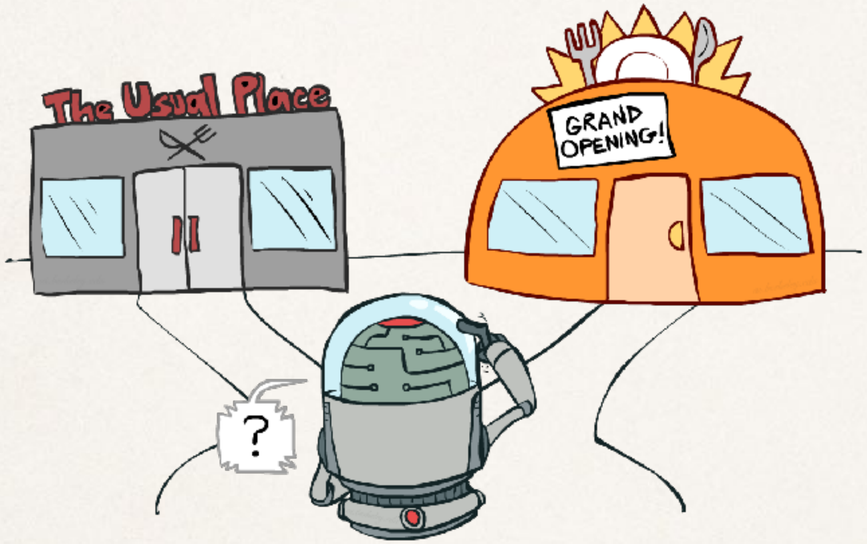
$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha \cdot sample$$

From CS188x


Pydev - Eclipse
Edit Navigate Search Project Run Window Help
Pydev
Gridworld Display
0.00 -0.05 0.00 0.00 0.00
-0.05 -0.05 -0.05 0.00 0.00 0.00 0.00 0.00 0.00 0.00
-0.05 0.00 0.00 0.00 0.00
-0.05 0.00 0.00 0.00
-0.05 -0.05 -0.05 0.00 0.00 0.00 0.00 0.00
-0.05 0.00 0.00 0.00
0.00
-50.00
0.00
0.00
0.00
0.00
CURRENT Q-VALUES
Console
-0.1
EPISODE 1
BEGINNING

python gridworld.py -m -v -g BridgeGrid -p -a q -k 100 -n 0

Exploration and Exploitation

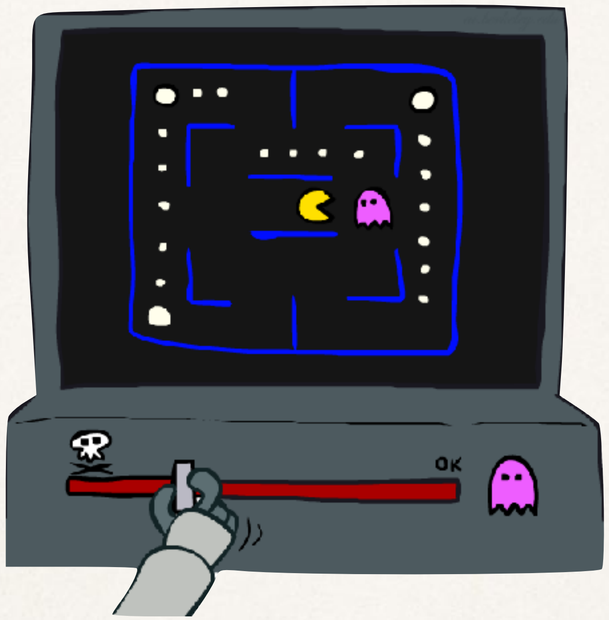


How to Explore?

Random Actions (ε-greedy)

- Every time step, flip a coin
- With (small) probability ε, act randomly
- With (large) probability 1-ε, act on current policy
- Lower ε over time

Approximate Q-Learning

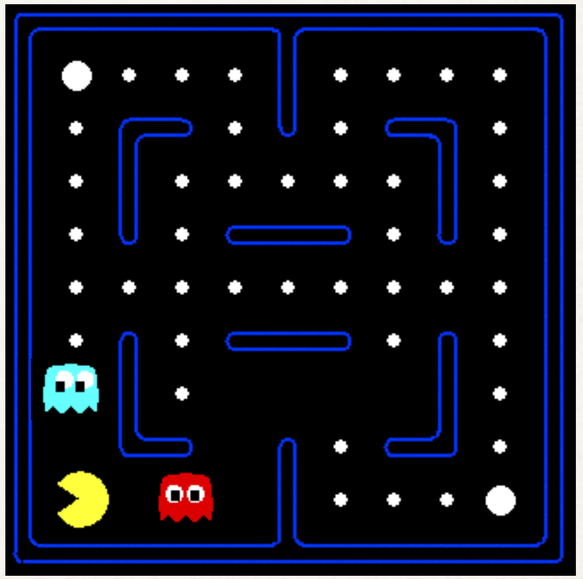


The Problem:

- Too many states to visit
- Too many states to hold the q-tables in memory

Feature-Based Representations

- Solution: describe a state using a vector of features (properties)
  - Features are functions from states to real numbers (often 0/1) that capture important properties of the state
  - Example features:
    - Distance to closest ghost
    - Distance to closest dot
    - Number of ghosts
    - 1 / (dist to dot)^2
    - Is Pacman in a tunnel? (0/1)
    - …… etc.
    - Is it the exact state on this slide?
  - Can also describe a q-state (s, a) with features (e.g. action moves closer to food)



Approximate Q-Learning

Linear Value Functions

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

Update

$$difference = [r + \gamma \max_{a'} Q(s',a')] - Q(s,a)$$

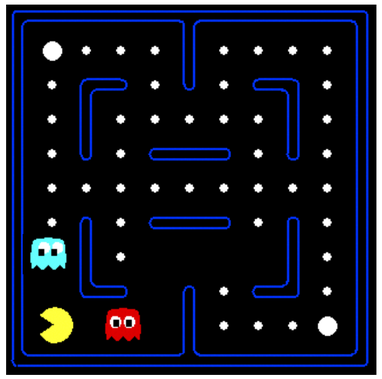
$$Q(s,a) \leftarrow Q(s,a) + \alpha \cdot difference$$

$$w_i \leftarrow w_i + \alpha \cdot difference \cdot f_i(s,a)$$

Example: Q-Pacman

$$Q(s,a) = 4.0 f_{DOT}(s,a) - 1.0 f_{GST}(s,a)$$

s

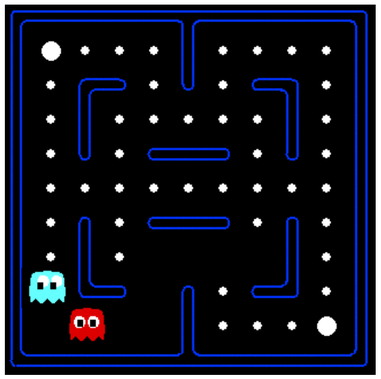


$f_{DOT}(s, \text{NORTH}) = 0.5$

$f_{GST}(s, \text{NORTH}) = 1.0$

$a = \text{NORTH}$
$r = -500$

s'



$Q(s, \text{NORTH}) = +1$

$Q(s', \cdot) = 0$

$$r + \gamma \max_{a'} Q(s', a') = -500 + 0$$

difference $= -501$

$$w_{DOT} \leftarrow 4.0 + \alpha\,[-501]\,0.5$$
$$w_{GST} \leftarrow -1.0 + \alpha\,[-501]\,1.0$$

$$Q(s,a) = 3.0 f_{DOT}(s,a) - 3.0 f_{GST}(s,a)$$

Pydev - Eclipse

File Edit Navigate Search Project Run Window Help

CS188 Pacman

Console

1.5

Beginning 300 episodes of Training

Next time: Deep Reinforcement Learning