Dynamic Programming (DP)

Chris Amato
Northeastern University

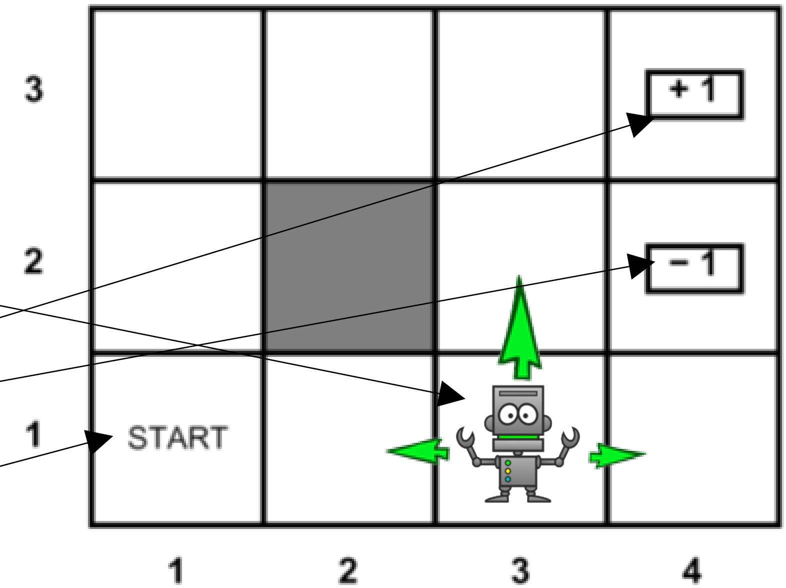
with some slides from Rob Platt, Lawson Wong and UAlberta

Announcements

- Exercise 2 (MDPs) due Monday 9/30
- Exercise 3 (DP) out soon
  - Due Wednesday Oct 9

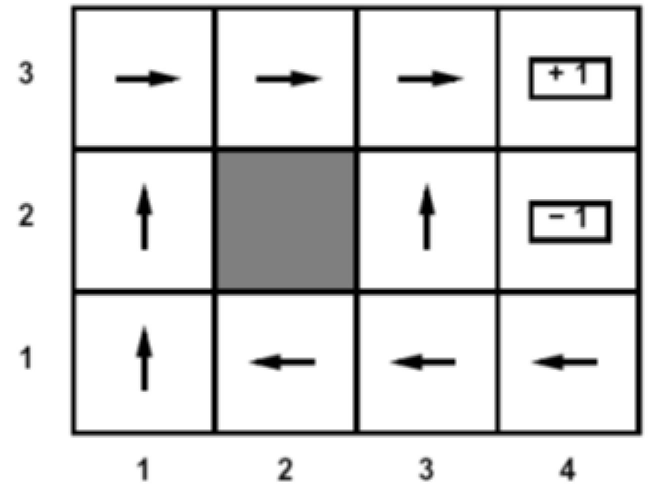
Markov Decision Processes

- An MDP is defined by:
  - A set of states s ∈ S
  - A set of actions a ∈ A
  - A transition function T(s, a, s')
    - Returns probability that action a from state s leads to s', i.e., P(s' | s, a)
  - A reward function R(s, a, s')
    - Sometimes just R(s) or R(s')
  - A start state
  - Maybe terminal state(s)
- Objective: calculate a strategy for acting so as to maximize the (discounted) sum of future rewards.
  - We will calculate a policy that will tell us how to act



Policies

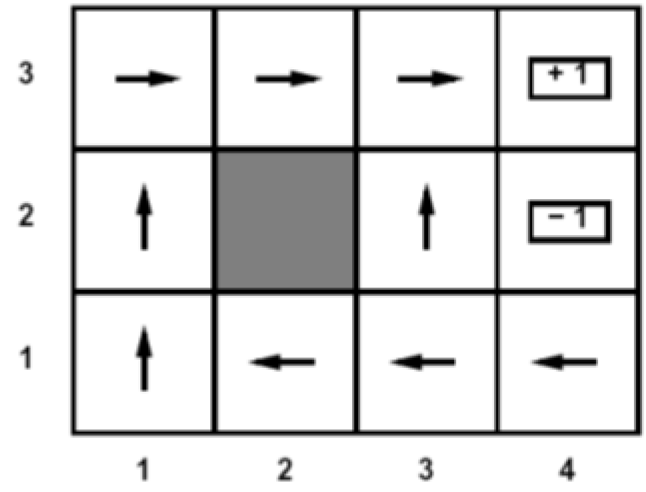
- For MDPs, we want an optimal policy $\pi^*: S \rightarrow A$
  - A policy π gives an action for each state
  - An optimal policy is one that maximizes expected returns if followed



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

Policies

- For MDPs, we want an optimal policy $\pi^*: S \rightarrow A$
  - A policy π gives an action for each state
  - An optimal policy is one that maximizes expected returns if followed



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

A *policy* is a rule for selecting actions: $\pi(s) = a$

If agent is in *this* state, then take *this* action

A policy can be stochastic: $\pi(a|s) = P(a_t = a | s_t = s)$

The goal of this lecture is to develop new ways of calculating an optimal policy
(assuming we know the full MDP model–no learning yet)

Calculating an optimal policy

- The goal of this lecture is to develop new ways of calculating an optimal policy
  - first, develop methods of calculating value function for an arbitrary policy (policy evaluation)
  - then, develop methods of calculating an optimal value function (and policy) by iteratively calculating value function and then improving policy

Many of the RL method will work to approximate the DP methods we talk about

Recall: Value Function

Value of state s when acting according to policy π :

$$V^{\pi}(s) = \mathbb{E}_{\pi}[G_t | s_t = s]$$

Value of a state == expected return from that state
if agent follows policy π

Recall: Value Function

Value of state s when acting according to policy π :

$$V^{\pi}(s) = \mathbb{E}_{\pi}[G_t | s_t = s]$$

Value of a state == expected return from that state
if agent follows policy π

But, how do we compute the value function?
(for a particular policy)

Recall: Value Function

Value of state s when acting according to policy π :

$$V^{\pi}(s) = \mathbb{E}_{\pi}[G_t | s_t = s]$$

Value of a state == expected return from that state if agent follows policy π

But, how do we compute the value function? (for a particular policy)

<u>Possible methods:</u>

1. Monte Carlo methods

Recall: Value Function

Value of state s when acting according to policy π :

$$V^{\pi}(s) = \mathbb{E}_{\pi}[G_t | s_t = s]$$

Value of a state == expected return from that state
if agent follows policy π

But, how do we compute the value function?
(for a particular policy)

Dynamic programming

New method that will
be introduced today

Bellman on "dynamic programming”

An interesting question is,
“Where did the name, dynamic programming, come from?”
The 1950s were not good years for mathematical research.

We had a very interesting gentleman in Washington named Wilson. He was Secretary of Defense, and he actually had a pathological fear and hatred of the word, research.

Bellman on "dynamic programming”

An interesting question is,
“Where did the name, dynamic programming, come from?”
The 1950s were not good years for mathematical research.

We had a very interesting gentleman in Washington named Wilson. He was Secretary of Defense, and he actually had a pathological fear and hatred of the word, research.

I’m not using the term lightly; I’m using it precisely. His face would suffuse, he would turn red, and he would get violent if people used the term, research, in his presence. You can imagine how he felt, then, about the term, mathematical.

The RAND Corporation was employed by the Air Force, and the Air Force had Wilson as its boss, essentially.

Hence, I felt I had to do something to shield Wilson and the Air Force from the fact that I was really doing mathematics inside the RAND Corporation.

Bellman on "dynamic programming”

What title, what name, could I choose? In the first place I was interested in planning, in decision making, in thinking.
But planning, is not a good word for various reasons.
I decided therefore to use the word, ‘programming.’

I wanted to get across the idea that this was dynamic, this was multistage, this was time-varying – I thought, let’s kill two birds with one stone.

Bellman on "dynamic programming”

What title, what name, could I choose? In the first place I was interested in planning, in decision making, in thinking.
But planning, is not a good word for various reasons.
I decided therefore to use the word, ‘programming.’

I wanted to get across the idea that this was dynamic, this was multistage, this was time-varying – I thought, let’s kill two birds with one stone.

Let’s take a word that has an absolutely precise meaning, namely dynamic, in the classical physical sense.
It also has a very interesting property as an adjective, and that is it’s impossible to use the word, dynamic, in a pejorative sense.
Try thinking of some combination that will possibly give it a pejorative meaning. It’s impossible.

Bellman on "dynamic programming”

What title, what name, could I choose? In the first place I was interested in planning, in decision making, in thinking.
But planning, is not a good word for various reasons.
I decided therefore to use the word, ‘programming.’

I wanted to get across the idea that this was dynamic, this was multistage, this was time-varying – I thought, let’s kill two birds with one stone.

Let’s take a word that has an absolutely precise meaning, namely dynamic, in the classical physical sense.
It also has a very interesting property as an adjective, and that is it’s impossible to use the word, dynamic, in a pejorative sense.
Try thinking of some combination that will possibly give it a pejorative meaning. It’s impossible.

Thus, I thought dynamic programming was a good name. It was something not even a Congressman could object to. So, I used it as an umbrella for my activities.

Bellman on "dynamic programming"

What title, what name, could I choose? In the first place I was interested in planning, in decision making, in thinking.
But ***planning***, is not a good word for various reasons.
I decided therefore to use the word, ‘programming.’

I wanted to get across the idea that this was dynamic, this was ***multistage***, this was time-varying – I thought, let’s kill two birds with one stone.

Let’s take a … ning,
namely dyna…
It also has a … and that is
it’s impossib… ve sense.
Try thinking … ve it a
pejorative meaning. It’s impossible.

Dynamic programming
=
Multistage planning

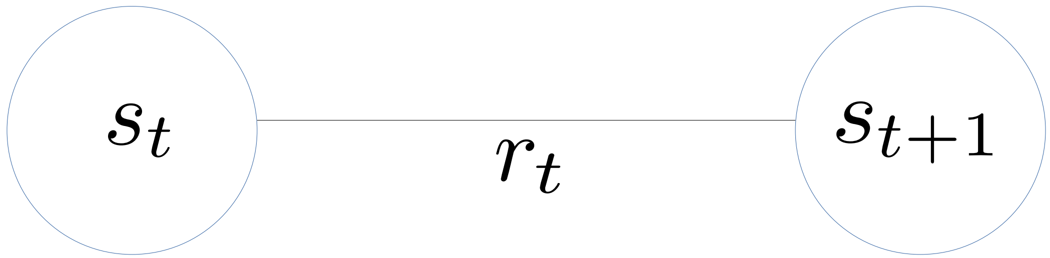
Thus, I thought dynamic programming was a good name. It was something not even a Congressman could object to. So I used it as an umbrella for my activities.

Step 1: Policy Evaluation

How do we calculate the value function for a given policy?

Policy Evaluation

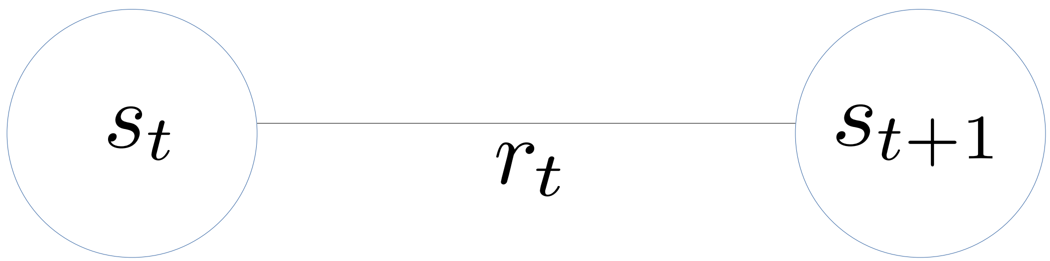
How do we calculate the value function for a given policy?



$$V^{\pi}(s_t = s) = r_{t+1} + \text{expected value of being at } s_{t+1}$$

$$= \mathbb{E}_{\pi}[r_{t+1} + \gamma V^{\pi}(s_{t+1} = s') | s_t = s]$$

How do we calculate the value function for a given policy?

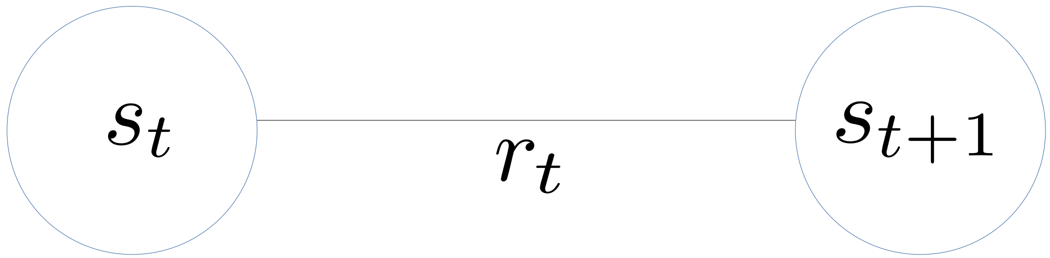


$$
\begin{aligned}
V^{\pi}(s_t = s) &= r_{t+1} + \text{expected value of being at } s_{t+1} \\
&= \mathbb{E}_{\pi}[r_{t+1} + \gamma V^{\pi}(s_{t+1} = s') | s_t = s] \\
&= \sum_{s',r} p(s', r | s, \pi(s)) [r + \gamma V^{\pi}(s_{t+1} = s')]
\end{aligned}
$$

Deterministic action

Policy Evaluation

How do we calculate the value function for a given policy?

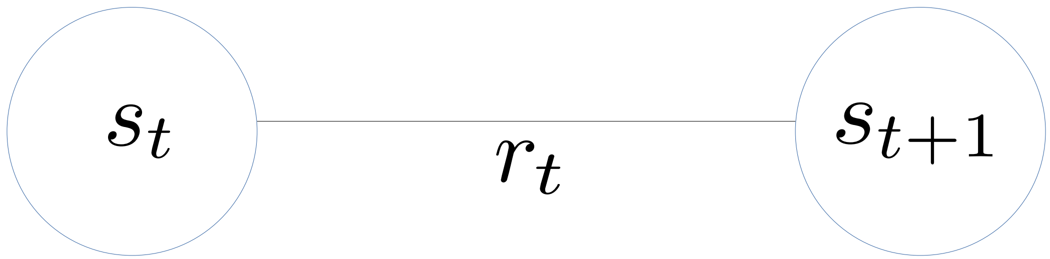


$$
\begin{aligned}
V^\pi(s_t = s) &= r_{t+1} + \text{expected value of being at } s_{t+1} \\
&= \mathbb{E}_\pi[r_{t+1} + \gamma V^\pi(s_{t+1} = s') | s_t = s] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma V^\pi(s_{t+1} = s')]
\end{aligned}
$$

Stochastic action

Policy Evaluation

How do we calculate the value function for a given policy?



$$V^\pi(s_t = s) = r_{t+1} + \text{expected value of being at } s_{t+1}$$

$$= \mathbb{E}_\pi[r_{t+1} + \gamma V^\pi(s_{t+1} = s') | s_t = s]$$

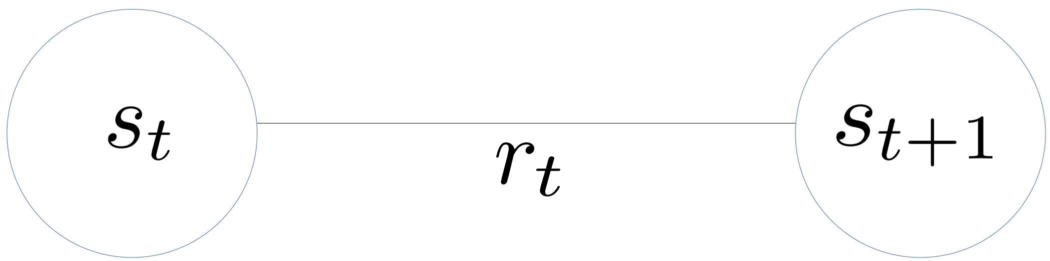
$$= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma V^\pi(s_{t+1} = s')]$$

Or, more simply: $$V^\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma V^\pi(s')]$$

(SB, eqn 4.4)

Policy Evaluation

How do we calculate the value function for a given policy?



$$
\begin{aligned}
V^\pi(s_t = s) &= r_{t+1} + \text{expected value of being at } s_{t+1} \\
&= \mathbb{E}_\pi[r_{t+1} + \gamma V^\pi(s_{t+1} = s') | s_t = s] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r | s, a)[r + \gamma V^\pi(s_{t+1} = s')]
\end{aligned}
$$

The *Bellman Equation*

Or, more simply: $V^\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma V^\pi(s')]$

(SB, eqn 4.4)

Policy Evaluation Algorithm

Iterative policy evaluation, SB pp 61

Policy Evaluation Algorithm

Iterative policy evaluation, SB pp 61

Input π, the policy to be evaluated

Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$

Repeat

$\quad \Delta \leftarrow 0$

$\quad$ For each $s \in \mathcal{S}$:

$\quad\quad v \leftarrow V(s)$

$\quad\quad V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a)\big[r + \gamma V(s')\big]$

$\quad\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \theta$ (a small positive number)

Output $V \approx v_\pi$

Policy Evaluation Algorithm

Iterative policy evaluation, SB pp 61

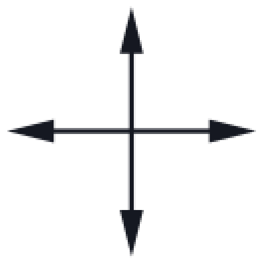
Input π, the policy to be evaluated
Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
    $\Delta \leftarrow 0$
    For each $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx v_\pi$

Bellman Equation

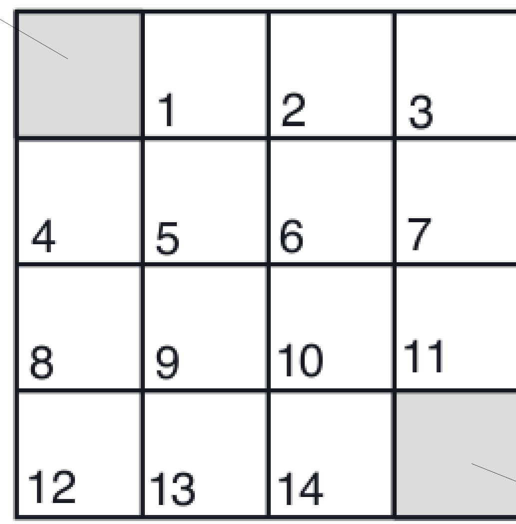
Is this the 'in place' (one array) version or the two array version?

Policy Evaluation Algorithm: SB example 4.1

Terminal state



actions



$$R_t = -1$$

on all transitions

Terminal state

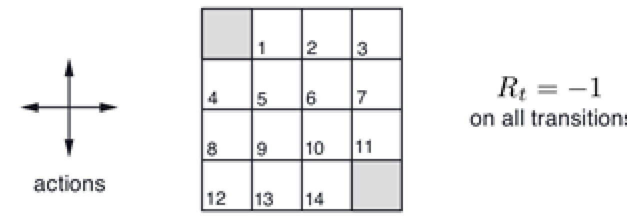
$$\mathcal{S} = \{1, \ldots, 14\}$$
$$\mathcal{A} = \{left, right, up, down\}$$

State transitions: deterministic
Undiscounted

Policy Evaluation Algorithm: SB example 4.1

Initialize value function at zero

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

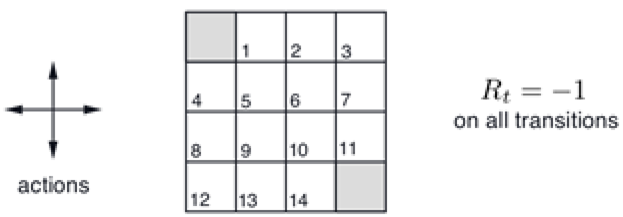


Evaluate *V* for a policy that selects actions uniformly randomly

SB example 4.1

What does this value become on first iteration?

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



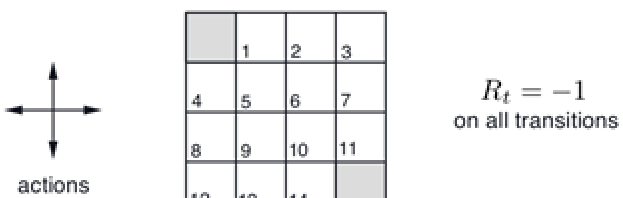
Input π, the policy to be evaluated
Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx v_\pi$

SB example 4.1

What does this value become on first iteration?

$$V^{\pi}(s) = \sum_{a} \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma V^{\pi}(s')]$$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



Input π, the policy to be evaluated
Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx v_{\pi}$

SB example 4.1

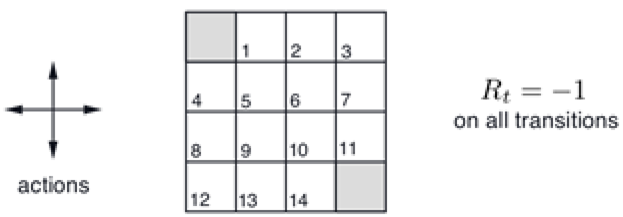
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

<u>What does this value become on first iteration?</u>

$$V^\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma V^\pi(s')]$$

$$= \sum_a \pi(a|s)[r + \gamma V^\pi(s')] \quad \text{(b/c deterministic)}$$

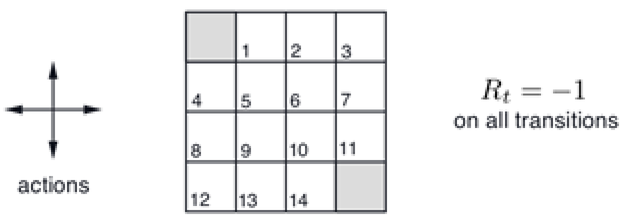
$$= \sum_a 0.25[-1 + \gamma V^\pi(s')]$$



Input π, the policy to be evaluated
Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx v_\pi$

SB example 4.1

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



What does this value become on first iteration?

$$V^\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma V^\pi(s')]$$

$$= \sum_a \pi(a|s)[r + \gamma V^\pi(s')] \quad \text{(b/c deterministic)}$$

$$= \sum_a 0.25[-1 + \gamma V^\pi(s')]$$

$$= \sum_a 0.25[-1 + \gamma 0]$$

$$= -1$$

Input π, the policy to be evaluated
Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx v_\pi$

SB example 4.1

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

<u>What does this value become on first iteration?</u>

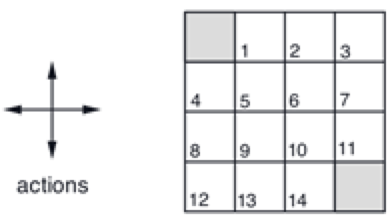
$$V^\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma V^\pi(s')]$$

$$= \sum_a \pi(a|s)[r + \gamma V^\pi(s')] \quad \text{(b/c deterministic)}$$

$$= \sum_a 0.25[-1 + \gamma V^\pi(s')]$$

$$= \sum_a 0.25[-1 + \gamma 0]$$

$$= -1$$



$R_t = -1$
on all transitions

Input π, the policy to be evaluated
Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
    $\Delta \leftarrow 0$
    For each $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx v_\pi$

SB example 4.1

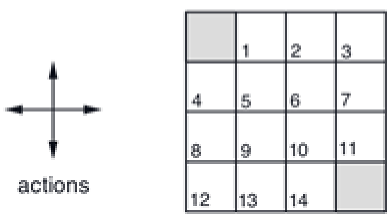
0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

<u>What does this value become on second iteration?</u>

$$V^\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma V^\pi(s')]$$

$$= \sum_a \pi(a|s)[r + \gamma V^\pi(s')] \quad \text{(b/c deterministic)}$$

$$= \sum_a 0.25[-1 + \gamma V^\pi(s')]$$

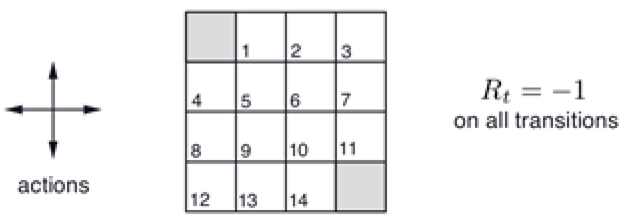


$R_t = -1$
on all transitions

Input π, the policy to be evaluated
Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
    $\Delta \leftarrow 0$
    For each $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx v_\pi$

SB example 4.1

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0



<u>What does this value become on second iteration?</u>

$$V^\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma V^\pi(s')]$$

$$= \sum_a \pi(a|s)[r + \gamma V^\pi(s')]$$ (b/c deterministic)

$$= \sum_a 0.25[-1 + \gamma V^\pi(s')]$$

$$= -\frac{1}{4} - \frac{2}{4} - \frac{2}{4} - \frac{2}{4}$$

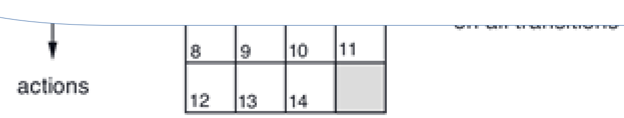
$$= -1.75$$

Input π, the policy to be evaluated
Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx v_\pi$

SB example 4.1

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2	1.7	0.0

Why is this value NOT -1.75?



<u>What does this value become on second iteration?</u>

$$
\begin{aligned}
V^\pi(s) &= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma V^\pi(s')] \\
&= \sum_a \pi(a|s)[r + \gamma V^\pi(s')] \quad \text{(b/c deterministic)} \\
&= \sum_a 0.25[-1 + \gamma V^\pi(s')] \\
&= -\frac{1}{4} - \frac{2}{4} - \frac{2}{4} - \frac{2}{4} \\
&= -1.75
\end{aligned}
$$

Input π, the policy to be evaluated
Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx v_\pi$

Think-pair-share

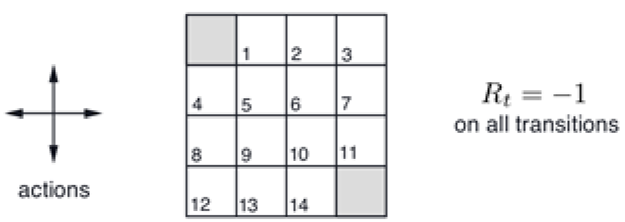
What does this value become on third iteration?

$$V^{\pi}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma V^{\pi}(s')]$$

$$= \sum_a \pi(a|s)[r + \gamma V^{\pi}(s')] \quad \text{(b/c deterministic)}$$

$$=$$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0



Input π, the policy to be evaluated
Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
    $\Delta \leftarrow 0$
    For each $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx v_\pi$

SB example 4.1

<u>What does this value become on third iteration?</u>

$$V^\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma V^\pi(s')]$$

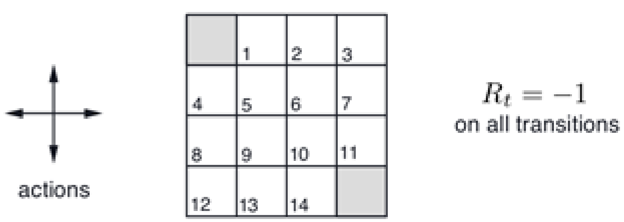
$$= \sum_a \pi(a|s)[r + \gamma V^\pi(s')] \quad \text{(b/c deterministic)}$$

$$= \sum_a 0.25[-1 + \gamma V^\pi(s')]$$

$$= -\frac{2.75}{4} - \frac{3}{4} - \frac{3}{4} - \frac{1}{4}$$

$$= -2.43$$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0



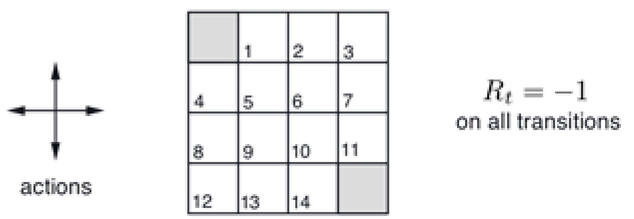
Input π, the policy to be evaluated
Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx v_\pi$

Question

Policy evaluation converges to these values

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

Can you think of a simple interpretation of the values of these states when policy evaluation converges?



Input π, the policy to be evaluated
Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r | s, a)\big[r + \gamma V(s')\big]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx v_\pi$

Policy evaluation

- Why does this converge?

Policy evaluation

- Why does this converge?
- Three cases
  - Finite-horizon undiscounted: After k steps of policy evaluation, have horizon k value
  - Episodic (indefinite horizon) undiscounted: If you will always reach a terminal state in a finite number of steps, see above, if only with prob 1 in the limit, see below
  - Infinite-horizon (discounted): After k steps of policy evaluation, have *discounted* horizon k value and discount causes reward value to shrink as k increases (so V^k will approach V^π arbitrarily closely as k increases)

Policy evaluation

- Why does this converge?
- Three cases
  - Finite-horizon undiscounted: After k steps of policy evaluation, have horizon k value
  - Episodic (indefinite horizon) undiscounted: If you will always reach a terminal state in a finite number of steps, see above, if only with prob 1 in the limit, see below
  - Infinite-horizon (discounted): After k steps of policy evaluation, have *discounted* horizon k value and discount causes reward value to shrink as k increases (so V^k will approach V^π arbitrarily closely as k increases)

- Can also evaluate with a system of $|S|$ linear equations

$$V^\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma V^\pi(s')]$$

Policy evaluation

Bellman equation gives

|S| linear equations (one per state)

with |S| unknowns (V(s) for each state s)

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r | s, a)\Big[r + \gamma v_\pi(s')\Big]$$

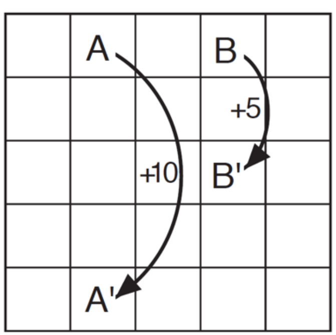
Write it out

$$V_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\Big[r + \gamma V_\pi(s')\Big] \leftarrow$$

$\underline{V_\pi((2,2))}$

$$= \sum_{a,s',r} \Big\{\pi(a|s)\, p(s',r|s,a)\Big] \Big[r + \gamma V_\pi(s')\Big]$$

$$= \begin{array}{ll} \left[\begin{array}{l} [0.25 \times 1] \times [0 + \gamma V_\pi((2,1))] + \\ [0.25 \times 1] \times [0 + \gamma V_\pi((2,3))] + \\ [0.25 \times 1] \times [0 + \gamma V_\pi((1,2))] + \\ [0.25 \times 1] \times [0 + \gamma V_\pi((3,2))] \end{array}\right] & \begin{array}{l} \text{left} \\ \text{right} \\ \text{up} \\ \text{down} \end{array} \end{array}$$



Policy improvement

Policy evaluation is great, but how do we use it to find better policies?

Policy improvement

Policy evaluation is great, but how do we use it to find better policies?

Given: value function, $V^{\pi}(s)$, for a given policy π

Calculate: a new policy, π', that is at least as good as π

Policy improvement

Policy evaluation is great, but how do we use it to find better policies?

<u>Given:</u> value function, $V^\pi(s)$, for a given policy π

<u>Calculate:</u> a new policy, π', that is at least as good as π

Policy improvement procedure:

1. calculate the value function, $V^\pi(s)$, for the latest policy, π

2. use Q to calculate a better policy: $\pi'(s) = \arg\max_a Q^\pi(s,a)$

Policy improvement

Policy improvement procedure:

1. calculate the value function, $V^{\pi}(s)$, for the latest policy, π

2. use Q to calculate a better policy: $\pi'(s) = \arg\max_a Q^{\pi}(s,a)$

But, how do we calculate $Q^{\pi}(s,a)$ **from** $V^{\pi}(s)$ **?**

Policy improvement

Value of being in state s, taking action a, and following policy π after that.

Value of being in state s, and following policy π after that.

But, how do we calculate $Q^{\pi}(s,a)$ **from** $V^{\pi}(s)$ **?**

Think-pair-share

Policy improvement procedure:

1. calculate the action-value function, $V^{\pi}(s)$, for the latest policy, π

2. use Q to calculate a better policy: $\pi'(s) = \arg\max_a Q^{\pi}(s,a)$

But, how do we calculate $Q^{\pi}(s,a)$ **from** $V^{\pi}(s)$ **?**

$$Q^{\pi}(s_t = s, a_t = a) = \boxed{}$$

Think-pair-share

Policy improvement procedure:

1. calculate the action-value function, $V^\pi(s)$, for the latest policy, π

2. use Q to calculate a better policy: $\pi'(s) = \arg\max_a Q^\pi(s,a)$

But, how do we calculate $Q^\pi(s,a)$ **from** $V^\pi(s)$ **?**

$$Q^\pi(s_t = s, a_t = a) = \boxed{}$$

<u>Hint: remember the Bellman eqn:</u>

$$V^\pi(s_t = s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma V^\pi(s_{t+1} = s')]$$

Use 4-tuple version

Think-pair-share

Policy improvement procedure:

1. calculate the action-value function, $V^\pi(s)$, for the latest policy, π

2. use Q to calculate a better policy: $\pi'(s) = \arg\max_a Q^\pi(s,a)$

But, how do we calculate $Q^\pi(s,a)$ **from** $V^\pi(s)$ **?**

$$Q^\pi(s_t = s, a_t = a) = \sum_{s',r} p(s',r|s,a)[r + \gamma V^\pi(s_{t+1} = s')]$$

(SB, eqn 4.6)

<u>Hint: remember the Bellman eqn:</u>

$$V^\pi(s_t = s) = \xcancel{\sum_a \pi(a|s)} \sum_{s',r} p(s',r|s,a)[r + \gamma V^\pi(s_{t+1} = s')]$$

Policy Improvement

Policy improvement procedure:

1. calculate the value function, $V^{\pi}(s)$, for the latest policy, π

2. use Q to calculate a better policy: $\pi'(s) = \arg\max_a Q^{\pi}(s, a)$

where $$Q^{\pi}(s, a) = \sum_{s', r} p(s', r | s, a)[r + \gamma V^{\pi}(s')]$$

Policy Improvement

Policy improvement procedure:

1. calculate the value function, $V^\pi(s)$, for the latest policy, π

2. use Q to calculate a better policy: $\pi'(s) = \arg\max_a Q^\pi(s,a)$

where $$Q^\pi(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma V^\pi(s')]$$

Policy improvement theorem:

Let π and π' be arbitrary deterministic policies such that

$$Q^\pi(s,\pi'(s)) \geq V^\pi(s), \forall s \in \mathcal{S}$$ (SB, eqn 4.7)

Then $V^{\pi'}(s) \geq V^\pi(s), \forall s \in \mathcal{S}$ (SB, eqn 4.8)

Policy Improvement

Policy improvement procedure:

1. calculate the value function, $V^{\pi}(s)$, for the latest policy, π

2. use Q to calculate

where Q

Policy π' is at least as good as π

Policy improvement theorem:

Let π and π' be arbitrary deterministic policies such that

$$Q^{\pi}(s, \pi'(s)) \geq V^{\pi}(s), \forall s \in \mathcal{S}$$ (SB, eqn 4.7)

Then $V^{\pi'}(s) \geq V^{\pi}(s), \forall s \in \mathcal{S}$ (SB, eqn 4.8)

Policy Improvement

- If it is better to select a in s once, then it is always better to select a in s
- Why does the policy improvement theorem make sense?

Policy Improvement

- If it is better to select a in s once, then it is always better to select a in s
- Policy improvement theorem proof:

$$
\begin{aligned}
v_\pi(s) &\leq q_\pi(s, \pi'(s)) \quad \forall s \in \mathrm{S} \\
&= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = \pi'(s)] && \text{(by (4.6))} \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s] && \text{(by (4.7))} \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}_{\pi'}[R_{t+2} + \gamma v_\pi(S_{t+2}) | S_{t+1}] \mid S_t = s] \\
&= \mathbb{E}_{\pi'}\big[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) \bigm| S_t = s\big] \\
&\leq \mathbb{E}_{\pi'}\big[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 v_\pi(S_{t+3}) \bigm| S_t = s\big] \\
&\;\;\vdots \\
&\leq \mathbb{E}_{\pi'}\big[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \bigm| S_t = s\big] \\
&= v_{\pi'}(s).
\end{aligned}
$$

Question

What if $v_{\pi'} = v_\pi$?

i.e., for all $s \in \mathcal{S}$, $v_\pi(s) = \max_a \sum_{s'} p(s' \mid s,a)\left[r(s,a,s') + \gamma\, v_\pi(s')\right]$?

Policy Improvement

What if $v_{\pi'} = v_{\pi}$?

i.e., for all $s \in \mathcal{S}$, $v_{\pi}(s) = \max_a \sum_{s'} p(s' \mid s,a)\left[r(s,a,s') + \gamma\, v_{\pi}(s')\right]$?

But this is the Bellman Optimality Equation.

So $v_{\pi} = v_*$ and both π and π' are optimal policies.

Policy Improvement

What if $v_{\pi'} = v_\pi$?

i.e., for all $s \in \mathcal{S}$, $v_\pi(s) = \max_a \sum_{s'} p(s' \mid s,a)\left[r(s,a,s') + \gamma\, v_\pi(s')\right]$?

But this is the Bellman Optimality Equation.

So $v_\pi = v_*$ and both π and π' are optimal policies.

So can converge to an optimal policy by using policy improvement

Policy Iteration

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

Policy Iteration

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
   $\quad \Delta \leftarrow 0$
   $\quad$ Loop for each $s \in \mathcal{S}$:
   $\qquad v \leftarrow V(s)$
   $\qquad V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s)) \big[r + \gamma V(s')\big]$
   $\qquad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

Policy Iteration

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
   $\Delta \leftarrow 0$
   Loop for each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s)) \big[r + \gamma V(s')\big]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   old-action $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s', r \mid s, a) \big[r + \gamma V(s')\big]$
   If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

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

Policy Iteration

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
   $\quad \Delta \leftarrow 0$
   $\quad$ Loop for each $s \in \mathcal{S}$:
   $\quad\quad v \leftarrow V(s)$
   $\quad\quad V(s) \leftarrow \sum_{s',r} p(s',r \,|\, s,\pi(s))\big[r + \gamma V(s')\big]$
   $\quad\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $\quad$ *old-action* $\leftarrow \pi(s)$
   $\quad \pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s',r \,|\, s,a)\big[r + \gamma V(s')\big]$
   $\quad$ If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

$\boxed{\text{Guaranteed to converge to } \pi^*}$

Policy Iteration

Policy iteration: combines policy evaluation and policy improvement

$$\pi_0 \xrightarrow{\mathrm{E}} v_{\pi_0} \xrightarrow{\mathrm{I}} \pi_1 \xrightarrow{\mathrm{E}} v_{\pi_1} \xrightarrow{\mathrm{I}} \pi_2 \xrightarrow{\mathrm{E}} \cdots \xrightarrow{\mathrm{I}} \pi_* \xrightarrow{\mathrm{E}} v_*$$

Policy Iteration

Policy iteration: combines policy evaluation and policy improvement

$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_*$$

Arbitrary initial policy

Policy Iteration

Policy iteration: combines policy evaluation and policy improvement

$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_*$$

Policy evaluation

Policy Iteration

Policy iteration: combines policy evaluation and policy improvement

$$\pi_0 \xrightarrow{\mathrm{E}} v_{\pi_0} \xrightarrow{\mathrm{I}} \pi_1 \xrightarrow{\mathrm{E}} v_{\pi_1} \xrightarrow{\mathrm{I}} \pi_2 \xrightarrow{\mathrm{E}} \cdots \xrightarrow{\mathrm{I}} \pi_* \xrightarrow{\mathrm{E}} v_*$$

Policy improvement

Policy Iteration

Policy iteration: combines policy evaluation and policy improvement

$$\pi_0 \xrightarrow{\mathrm{E}} v_{\pi_0} \xrightarrow{\mathrm{I}} \pi_1 \xrightarrow{\mathrm{E}} v_{\pi_1} \xrightarrow{\mathrm{I}} \pi_2 \xrightarrow{\mathrm{E}} \cdots \xrightarrow{\mathrm{I}} \pi_* \xrightarrow{\mathrm{E}} v_*$$

New policy

Policy Iteration

Policy iteration: combines policy evaluation and policy improvement

$$\pi_0 \xrightarrow{\mathrm{E}} v_{\pi_0} \xrightarrow{\mathrm{I}} \pi_1 \xrightarrow{\mathrm{E}} v_{\pi_1} \xrightarrow{\mathrm{I}} \pi_2 \xrightarrow{\mathrm{E}} \cdots \xrightarrow{\mathrm{I}} \pi_* \xrightarrow{\mathrm{E}} v_*$$

Policy evaluation

Policy Iteration

Policy iteration: combines policy evaluation and policy improvement

$$\pi_0 \xrightarrow{\mathrm{E}} v_{\pi_0} \xrightarrow{\mathrm{I}} \pi_1 \xrightarrow{\mathrm{E}} v_{\pi_1} \xrightarrow{\mathrm{I}} \pi_2 \xrightarrow{\mathrm{E}} \cdots \xrightarrow{\mathrm{I}} \pi_* \xrightarrow{\mathrm{E}} v_*$$

Policy improvement

Policy Iteration

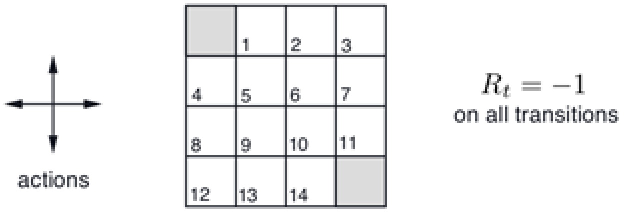
Policy iteration: combines policy evaluation and policy improvement

$$\pi_0 \xrightarrow{\mathrm{E}} v_{\pi_0} \xrightarrow{\mathrm{I}} \pi_1 \xrightarrow{\mathrm{E}} v_{\pi_1} \xrightarrow{\mathrm{I}} \pi_2 \xrightarrow{\mathrm{E}} \cdots \xrightarrow{\mathrm{I}} \pi_* \xrightarrow{\mathrm{E}} v_*$$

New policy

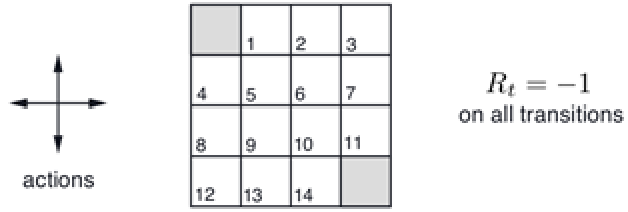
Policy Iteration Example

Recall grid world problem from earlier:

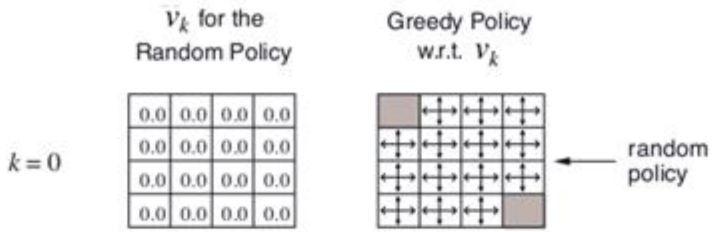


Policy Iteration Example

Recall grid world problem from earlier:

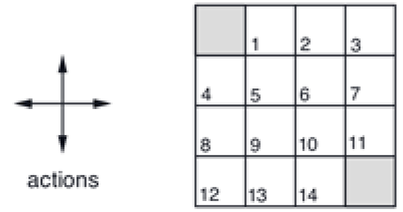


First iteration of PI:



Policy Iteration Example

Recall grid world problem from earlier:



$R_t = -1$
on all transitions

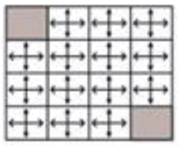
First iteration of PI:

v_k for the Random Policy

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

Greedy Policy w.r.t. v_k



random policy

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

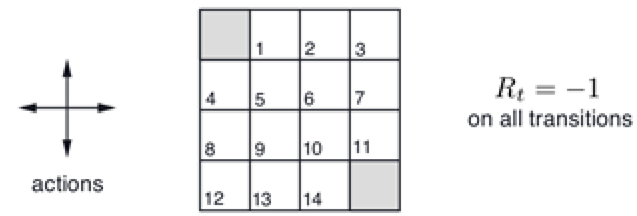
0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

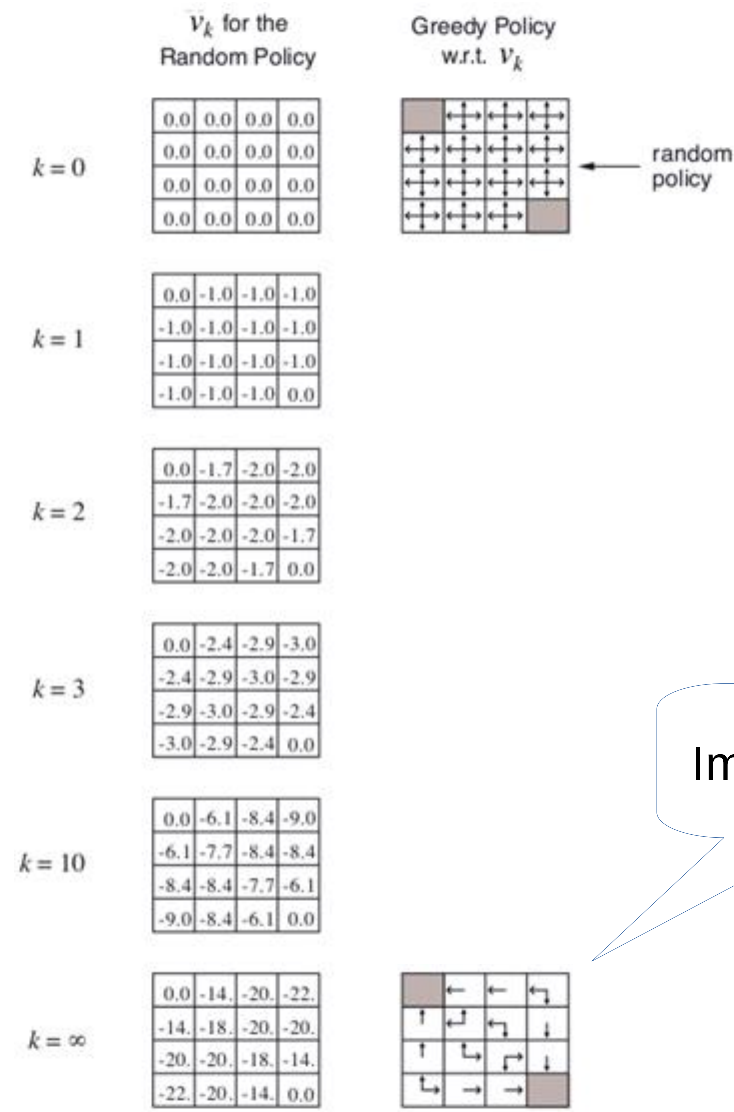
0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

Policy Iteration Example

Recall grid world problem from earlier:



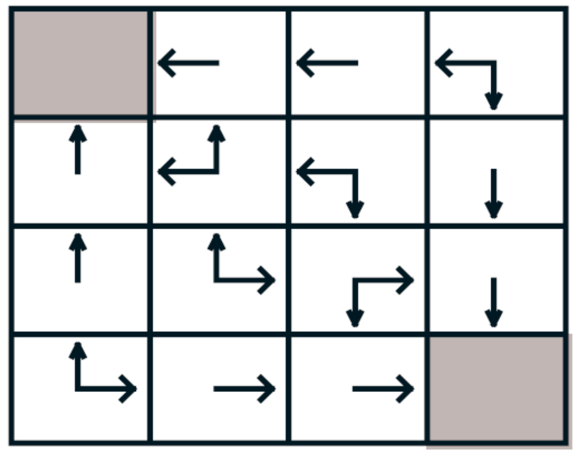
First iteration of PI:



Improved policy

Policy Iteration Example

New policy:



2. Iterative policy evaluation

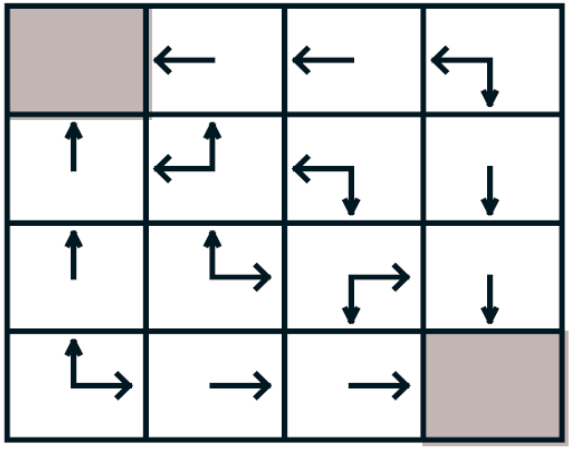
3. Policy improvement

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

Start from the latest value function

Policy Iteration Example

New policy:



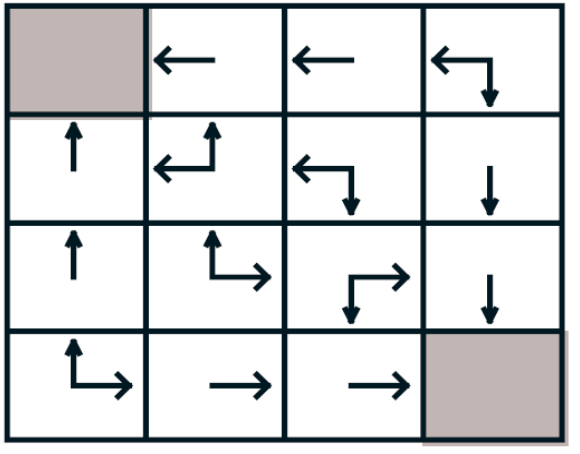
2. Iterative policy evaluation

3. Policy improvement

0	0	-8.35236	-8.96732
-6.13797	-7.7374	-8.42783	-8.35236
-8.35236	-8.42783	-7.7374	-6.13797
-8.96732	-8.35236	-6.13797	0

Policy Iteration Example

New policy:



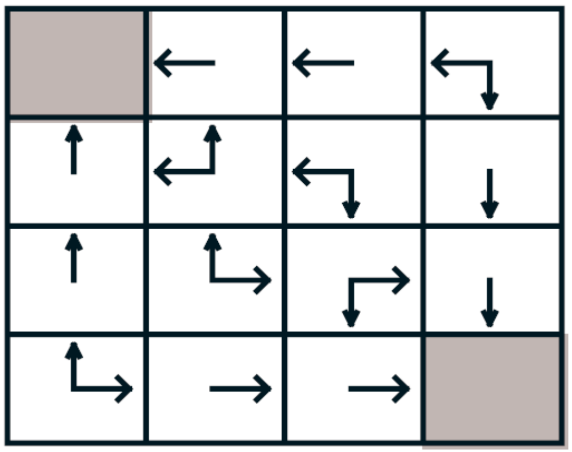
2. Iterative policy evaluation

3. Policy improvement

0	-1	-7.13797	-9.35236
-1	-7.13797	-8.7374	-7.13797
-7.13797	-8.7374	-7.13797	-1
-9.35236	-7.13797	-1	0

Policy Iteration Example

New policy:



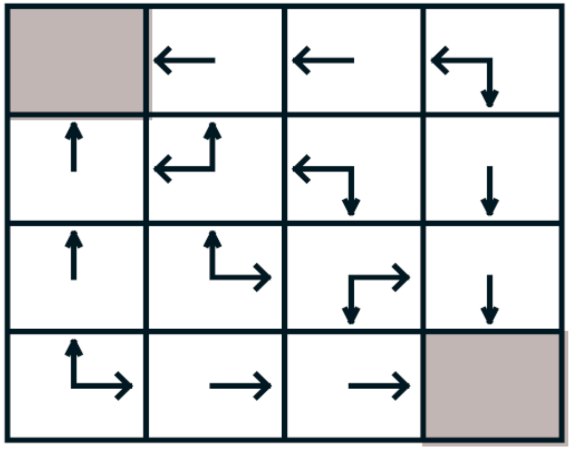
2. Iterative policy evaluation

0	-1	-2	-8.13797
-1	-2	-8.13797	-2
-2	-8.13797	-2	-1
-8.13797	-2	-1	0

3. Policy improvement

Policy Iteration Example

New policy:



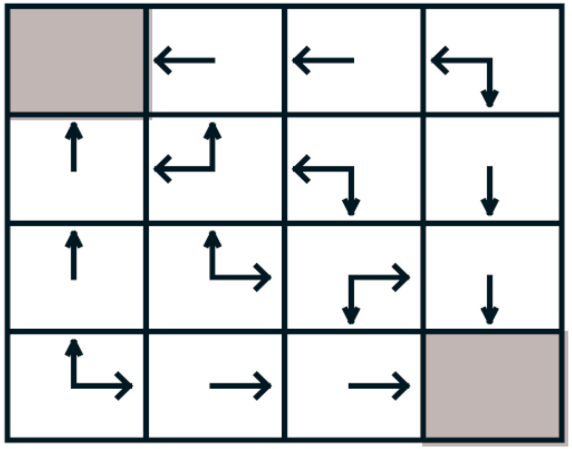
2. Iterative policy evaluation

0	-1	-2	-3
-1	-2	-3	-2
-2	-3	-2	-1
-3	-2	-1	0

3. Policy improvement

Policy Iteration Example

New policy:



2. Iterative policy evaluation

0	-1	-2	-3
-1	-2	-3	-2
-2	-3	-2	-1
-3	-2	-1	0

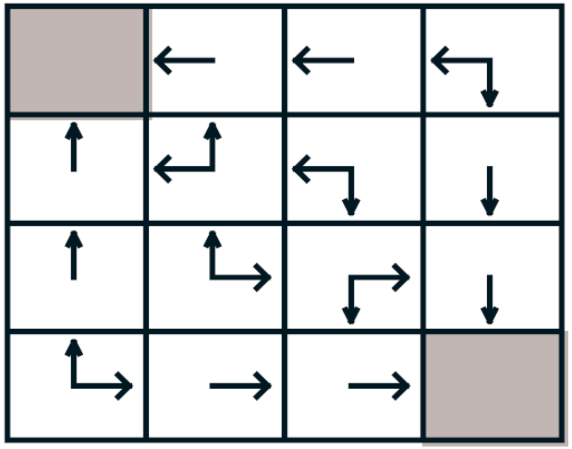
Values have converged!

3. Policy improvement

Proceed with step 3.

Policy Iteration Example

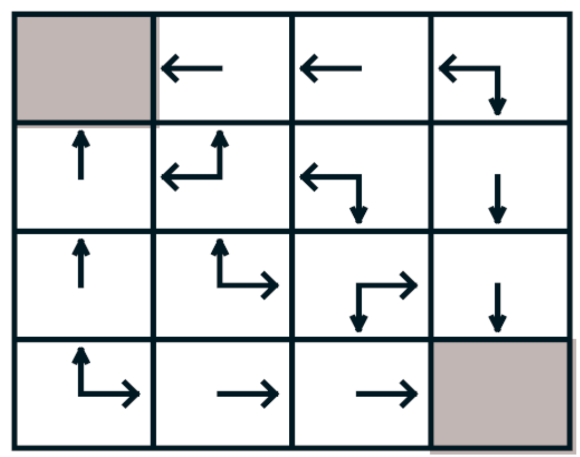
New policy:



2. Iterative policy evaluation

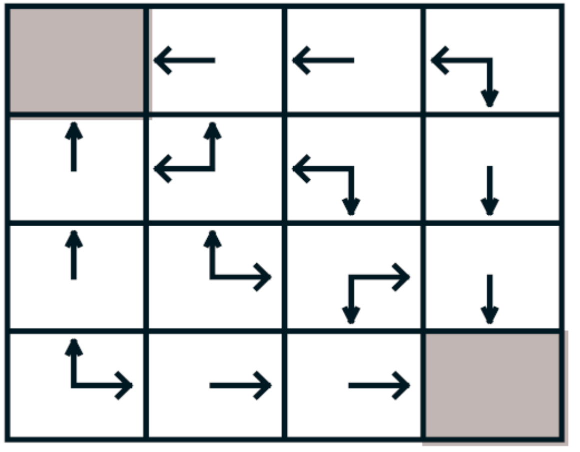
0	-1	-2	-3
-1	-2	-3	-2
-2	-3	-2	-1
-3	-2	-1	0

3. Policy improvement



Policy Iteration Example

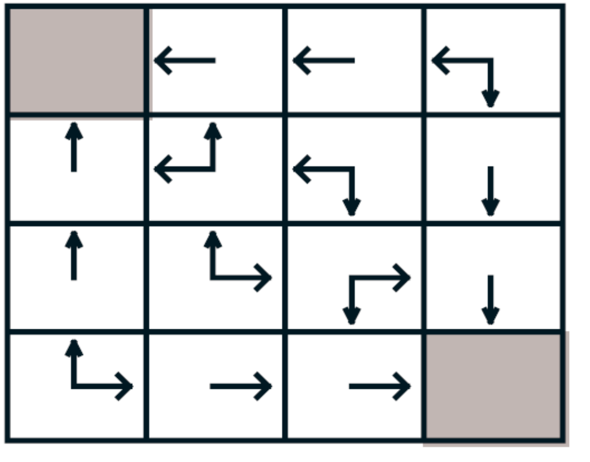
New policy:



2. Iterative policy evaluation

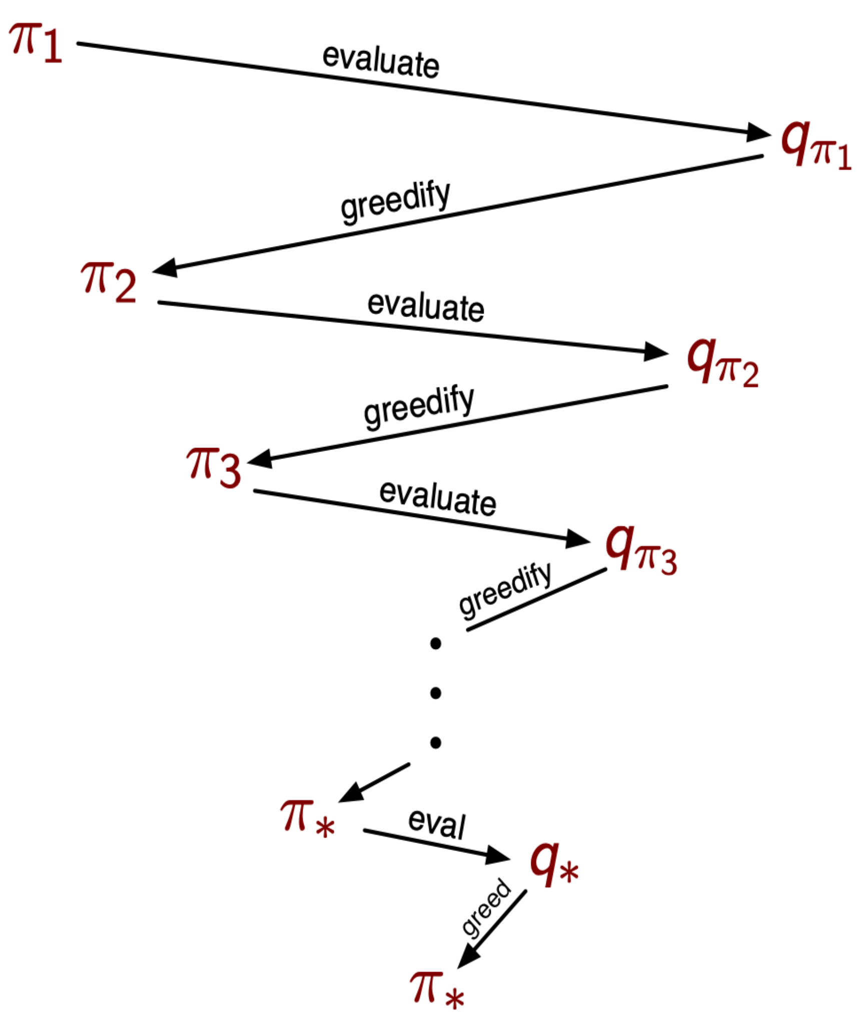
0	-1	-2	-3
-1	-2	-3	-2
-2	-3	-2	-1
-3	-2	-1	0

3. Policy improvement



Policy has converged! Done.

The dance of policy and value (Policy Iteration)



Any policy evaluates to a unique value function (soon we will see how to learn it)

which can be greedified to produce a better policy

That in turn evaluates to a value function

which can in turn be greedified…

Each policy is *strictly better* than the previous, until *eventually both are optimal*

There are *no local optima*

The dance converges in a finite number of steps, usually very few

Think-pair-share

Exercise 4.5 How would policy iteration be defined for action values? Give a complete algorithm for computing q_*, analogous to that on page 80 for computing v_*. Please pay special attention to this exercise, because the ideas involved will be used throughout the rest of the book. □

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
   $\quad\Delta \leftarrow 0$
   $\quad$For each $s \in \mathcal{S}$:
   $\qquad v \leftarrow V(s)$
   $\qquad V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s))\big[r + \gamma V(s')\big]$
   $\qquad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $\quad$*old-action* $\leftarrow \pi(s)$
   $\quad \pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$
   $\quad$If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Stopping Policy Evaluation Early

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
   - $\Delta \leftarrow 0$
   - For each $s \in \mathcal{S}$:
     - $v \leftarrow V(s)$
     - $V(s) \leftarrow \sum_{s',r} p(s', r \,|\, s, \pi(s))\big[r + \gamma V(s')\big]$
     - $\Delta \leftarrow \max(\Delta, |v - V(s)|)$

   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   - *old-action* $\leftarrow \pi(s)$
   - $\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s', r \,|\, s, a)\big[r + \gamma V(s')\big]$
   - If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*

   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Do we have to run policy evaluation until convergence, or can we stop early?

Stopping Policy Evaluation Early

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
   $\quad \Delta \leftarrow 0$
   $\quad$ For each $s \in \mathcal{S}$:
   $\quad\quad v \leftarrow V(s)$
   $\quad\quad V(s) \leftarrow \sum_{s',r} p(s', r | s, \pi(s)) \big[r + \gamma V(s') \big]$
   $\quad\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $\quad$ *old-action* $\leftarrow \pi(s)$
   $\quad \pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s', r$
   $\quad$ If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Do we have to run policy evaluation until convergence, or can we stop early?

<u>We can stop early.</u>
– how many iterations are needed?

Stopping Policy Evaluation Early

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
   $\Delta \leftarrow 0$
   For each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r | s, \pi(s)) \big[r + \gamma V(s')\big]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   old-action $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s', r$
   If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Do we have to run policy evaluation until convergence, or can we stop early?

<u>We can stop early.</u>
– how many iterations are needed?
– it turns out that even just one works.

Stopping Policy Evaluation Early

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
   $\quad \Delta \leftarrow 0$
   $\quad$ For each $s \in \mathcal{S}$:
   $\quad\quad v \leftarrow V(s)$
   $\quad\quad V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))\big[r + \gamma V(s')\big]$
   $\quad\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $\quad$ *old-action* $\leftarrow \pi(s)$
   $\quad \pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r$
   $\quad$ If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Do we have to run policy evaluation until convergence, or can we stop early?

<u>We can stop early.</u>
– how many iterations are needed?
– it turns out that even just one works.
– but any number is okay...

Stopping Policy Evaluation Early

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
   - $\Delta \leftarrow 0$
   - For each $s \in \mathcal{S}$:
     - $v \leftarrow V(s)$
     - $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))\big[r + \gamma V(s')\big]$
     - $\Delta \leftarrow \max(\Delta, |v - V(s)|)$

   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   - *old-action* $\leftarrow \pi(s)$
   - $\pi(s) \leftarrow \text{argmax}_a \sum_{s',r} p(s', r$
   - If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*

   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to

Do we have to run policy evaluation until convergence, or can we stop early?

We can stop early.
– how many iterations are needed?
– it turns out that even just one works.
– but any number is okay...

This is called value iteration

Value Iteration

Initialize array V arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)

Repeat
    $\Delta \leftarrow 0$
    For each $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \max_a \sum_{s',r} p(s', r | s, a)\big[r + \gamma V(s')\big]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, $\pi \approx \pi_*$, such that
    $\pi(s) = \arg\max_a \sum_{s',r} p(s', r | s, a)\big[r + \gamma V(s')\big]$

Value Iteration

Initialize array V arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)

Repeat

$\quad \Delta \leftarrow 0$

$\quad$ For each $s \in \mathcal{S}$:

$\quad\quad v \leftarrow V(s)$

$\quad\quad V(s) \leftarrow \overbrace{\max_a}^{\text{Policy improvement}} \overbrace{\sum_{s',r} p(s', r \,|\, s, a)\big[r + \gamma V(s')\big]}^{\text{One step of policy evaluation}}$

$\quad\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, $\pi \approx \pi_*$, such that

$\quad \pi(s) = \arg\max_a \sum_{s',r} p(s', r \,|\, s, a)\big[r + \gamma V(s')\big]$

Value Iteration

Initialize array V arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)

Repeat

$\quad \Delta \leftarrow 0$

$\quad$ For each $s \in \mathcal{S}$:

$\quad\quad v \leftarrow V(s)$

$$V(s) \leftarrow \overbrace{\max_a}^{\text{Policy improvement}} \overbrace{\sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]}^{\text{One step of policy evaluation}}$$

$\quad\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, $\pi \approx \pi_*$, such that

$$\pi(s) = \arg\max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$$

<u>In-place</u> update of V(s)

Typically converges faster than two-array (old+new) version

- Uses the latest value estimate

Value Iteration

Initialize array V arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)

Update rule based on Bellman Eqn

Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $\boxed{V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]}$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, $\pi \approx \pi_*$, such that
  $\pi(s) = \arg\max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$

Another way to understand value iteration is to see that we are iteratively re-applying the Bellman Equation, used as an update rule.

Value Iteration

Another way to understand value iteration is to see that we are iteratively re-applying the Bellman Equation, used as an update rule.

1. $V_0(s) = \text{arbitrary}$
2. $V_1(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V_0(s')]$
3. $V_2(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V_1(s')]$
4. $V_3(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V_2(s')]$
5. $\vdots$

Value Iteration

Another way to understand value iteration is to see that we are iteratively re-applying the Bellman Equation, used as an update rule.

Initial value

1. $V_0(s) = \text{arbitrary}$
2. $V_1(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V_0(s')]$
3. $V_2(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V_1(s')]$
4. $V_3(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V_2(s')]$
5. $\vdots$

Value Iteration

Another way to understand value iteration is to see that we are iteratively re-applying the Bellman Equation, used as an update rule.

Value over time horizon == 1

1. $V_0(s) = \text{arbitrary}$
2. $V_1(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V_0(s')]$
3. $V_2(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V_1(s')]$
4. $V_3(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V_2(s')]$
5. $\vdots$

Value Iteration

Another way to understand value iteration is to see that we are iteratively re-applying the Bellman Equation, used as an update rule.

1. $V_0(s) = \text{arbitrary}$
2. $V_1(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V_0(s')]$
3. $V_2(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V_1(s')]$
4. $V_3(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V_2(s')]$
5. $\vdots$

Value over time horizon == 2

Value Iteration

Another way to understand value iteration is to see that we are iteratively re-applying the Bellman Equation, used as an update rule.

1. $V_0(s) = \text{arbitrary}$
2. $V_1(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V_0(s')]$
3. $V_2(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V_1(s')]$
4. $V_3(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V_2(s')]$
5. $\vdots$

Value over time horizon == 3

Value Iteration

Another way to understand value iteration is to see that we are iteratively re-applying the Bellman Equation, used as an update rule.

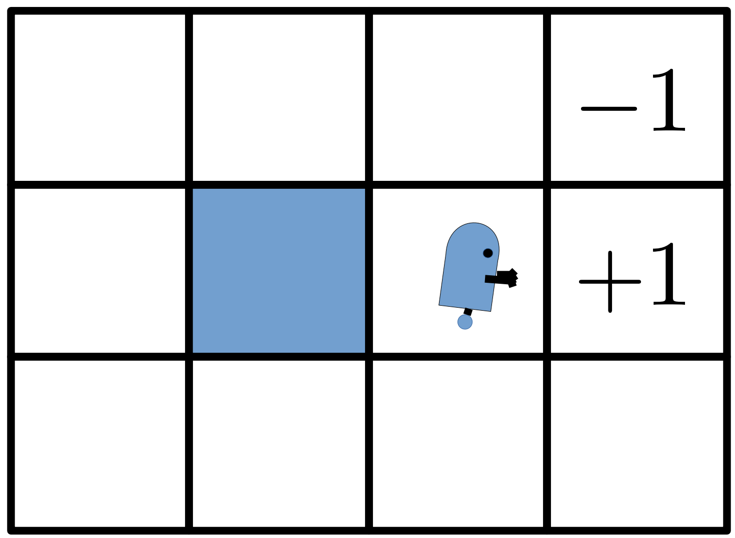
Value over time horizon == 3

1. $V_0(s) = \text{arbitrary}$
2. $V_1(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V_0(s')]$
3. $V_2(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V_1(s')]$
4. $V_3(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V_2(s')]$
5. $\vdots$

Converges to optimal value function over infinite time horizon

Value Iteration Example

Noise = 0.2
Discount = 0.9
Living reward = 0

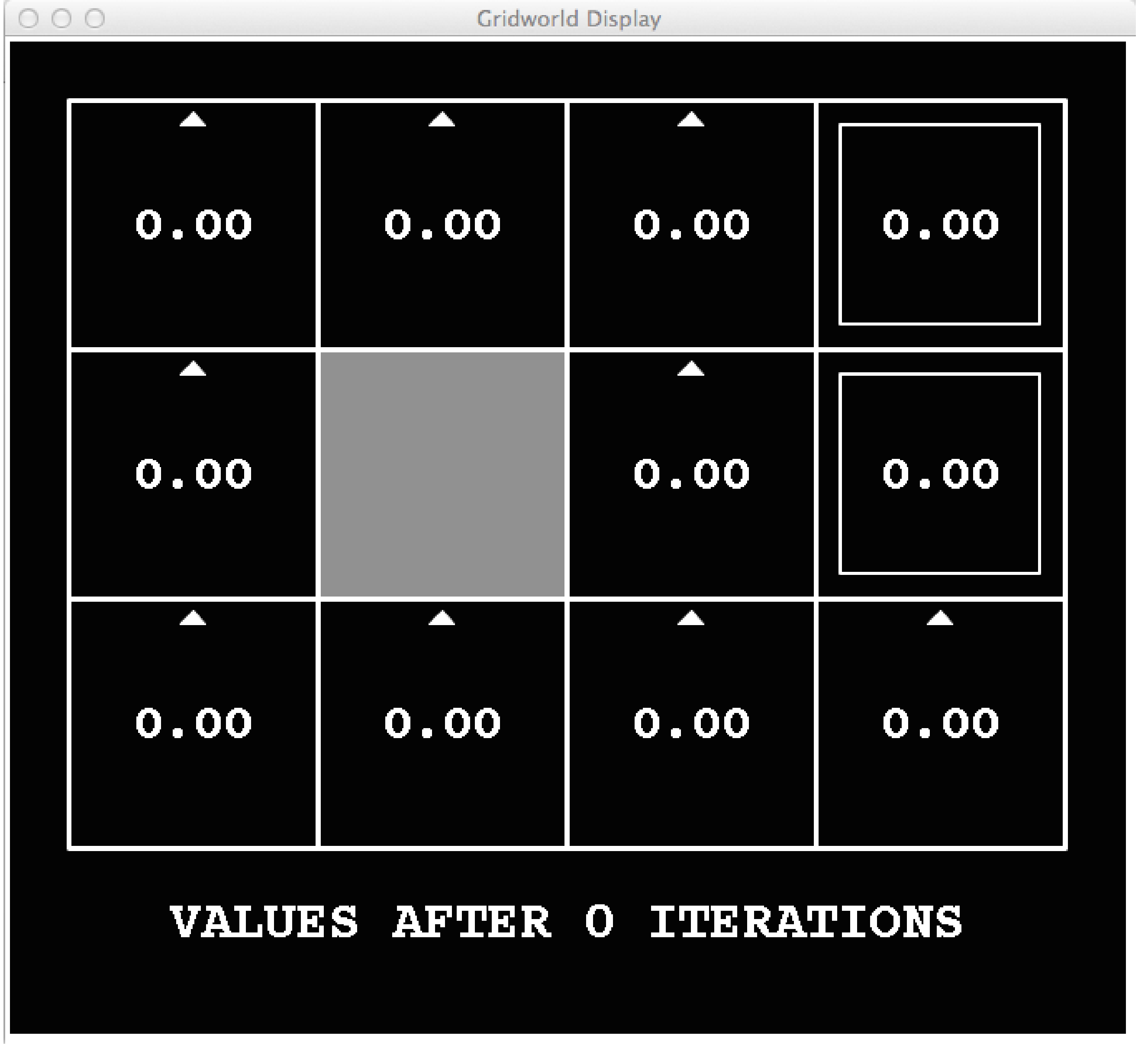


Actions: left, right, up, down
– take one action per time step
– actions are stochastic: only go in intended direction 80% of the time; 10% of the time go 90deg to the left, 10% of the time go 90deg to the right.

Grid world:
– agent lives on grid
– always occupies a single cell
– can move left, right, up, down
– but movements are noisy/stochastic
– gets zero reward unless in “+1” or “-1” cells

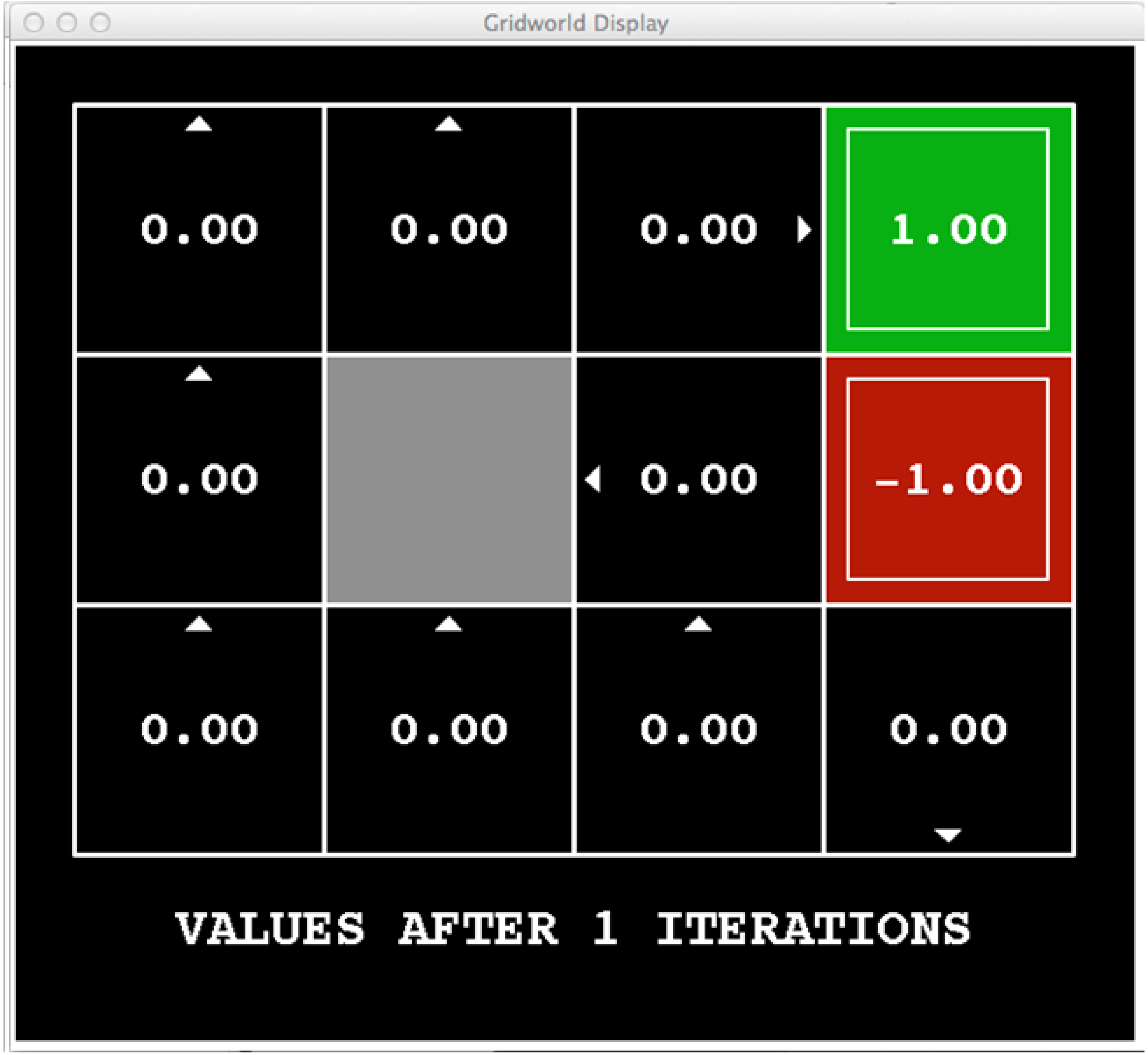
Value Iteration Example

Noise = 0.2
Discount = 0.9
Living reward = 0



Value Iteration Example

Noise = 0.2
Discount = 0.9
Living reward = 0



Value Iteration Example

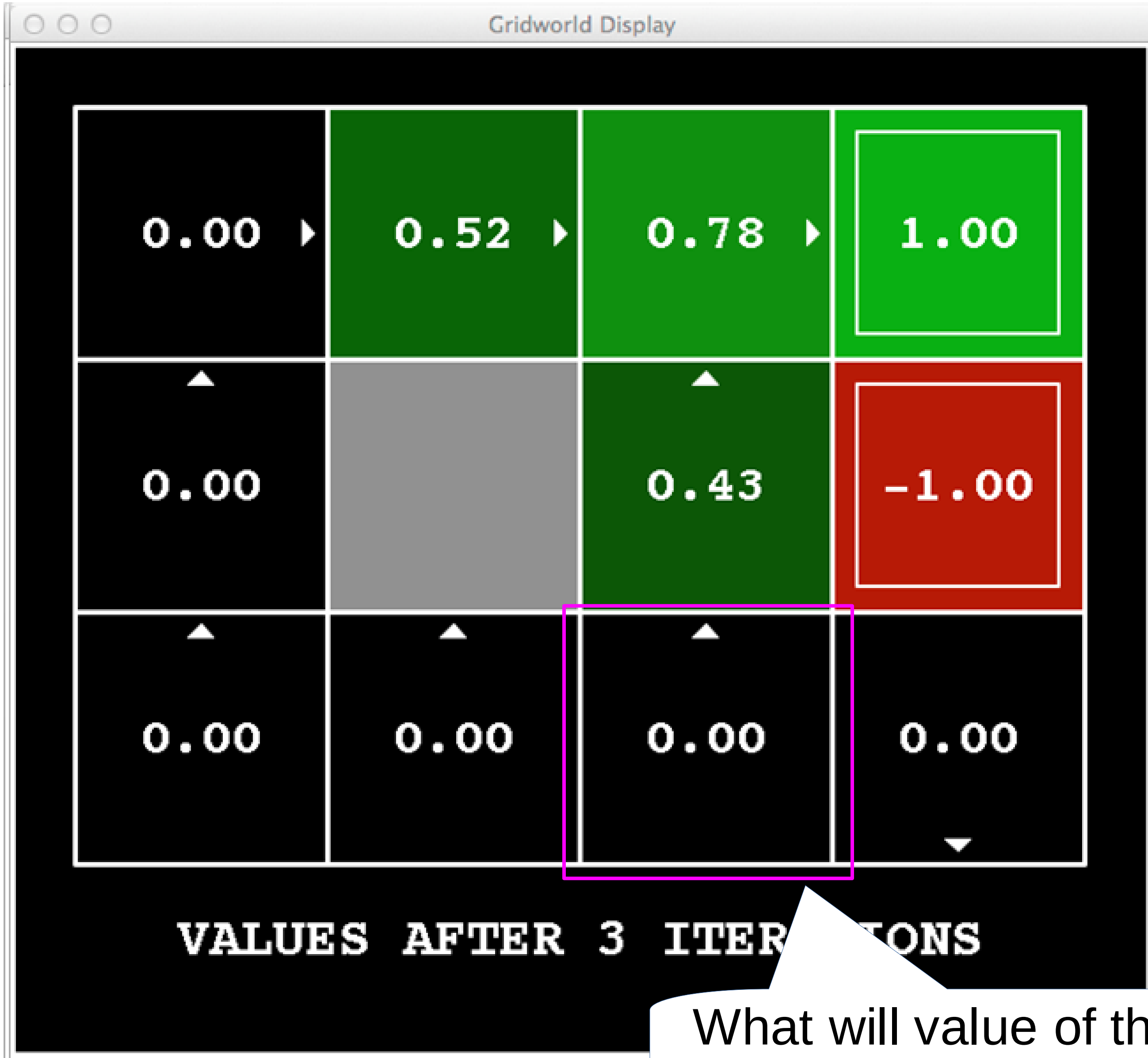
Noise = 0.2
Discount = 0.9
Living reward = 0

Value Iteration Example

Noise = 0.2
Discount = 0.9
Living reward = 0

Think-pair-share

Noise = 0.2
Discount = 0.9
Living reward = 0



What will value of this state be on the next time step?

Value Iteration Example

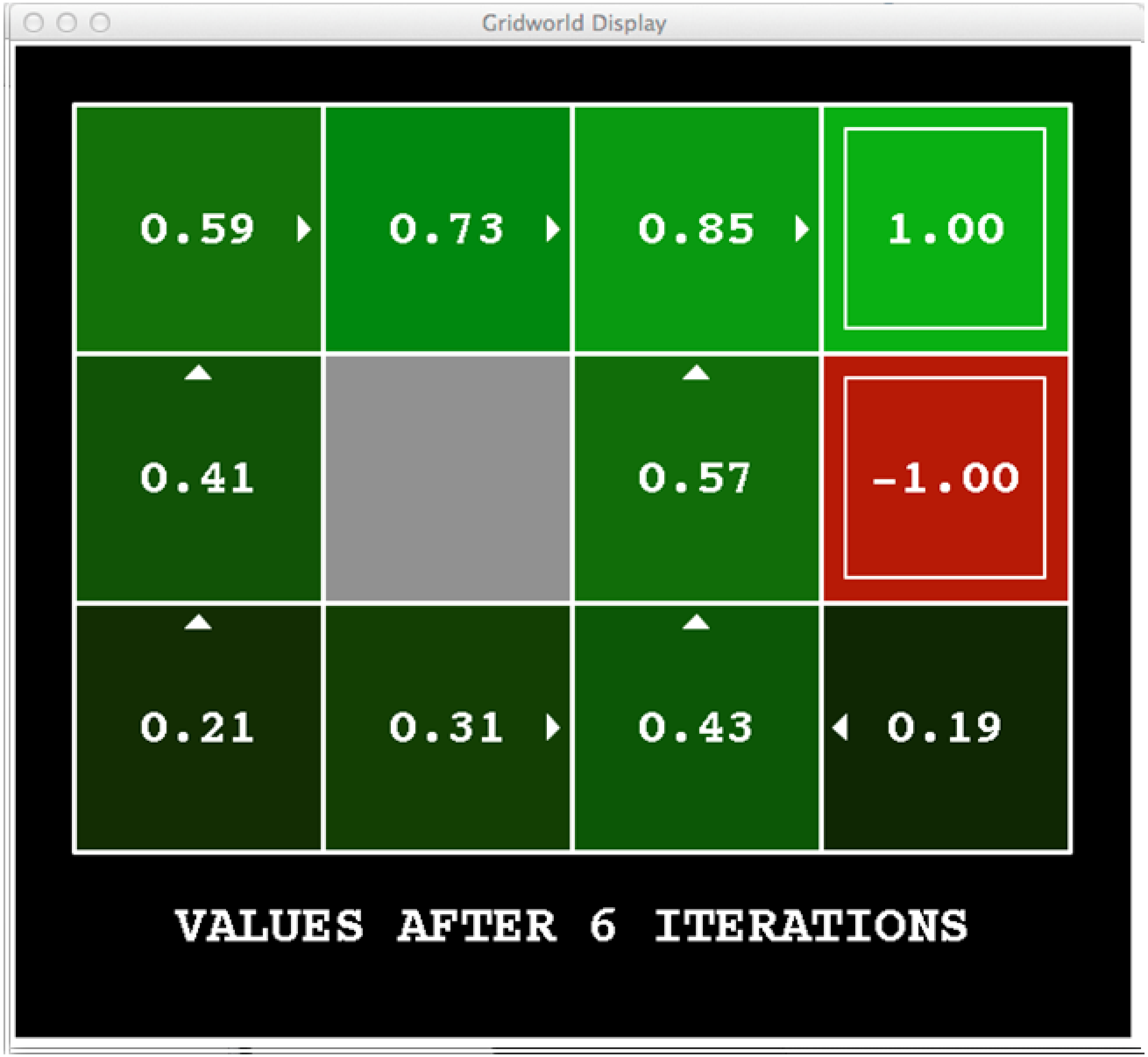
Noise = 0.2
Discount = 0.9
Living reward = 0

Value Iteration Example

Noise = 0.2
Discount = 0.9
Living reward = 0

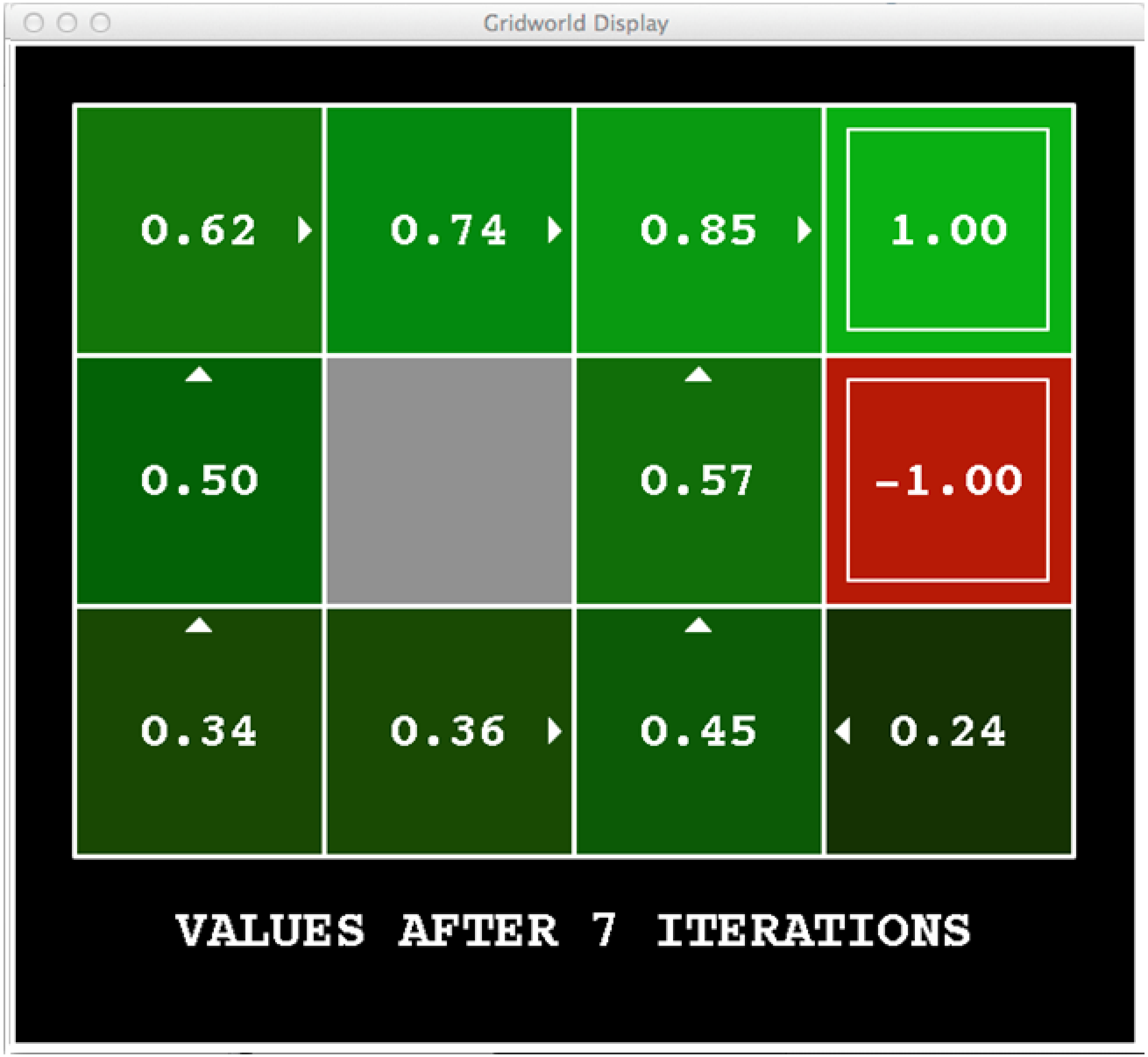
Value Iteration Example

Noise = 0.2
Discount = 0.9
Living reward = 0



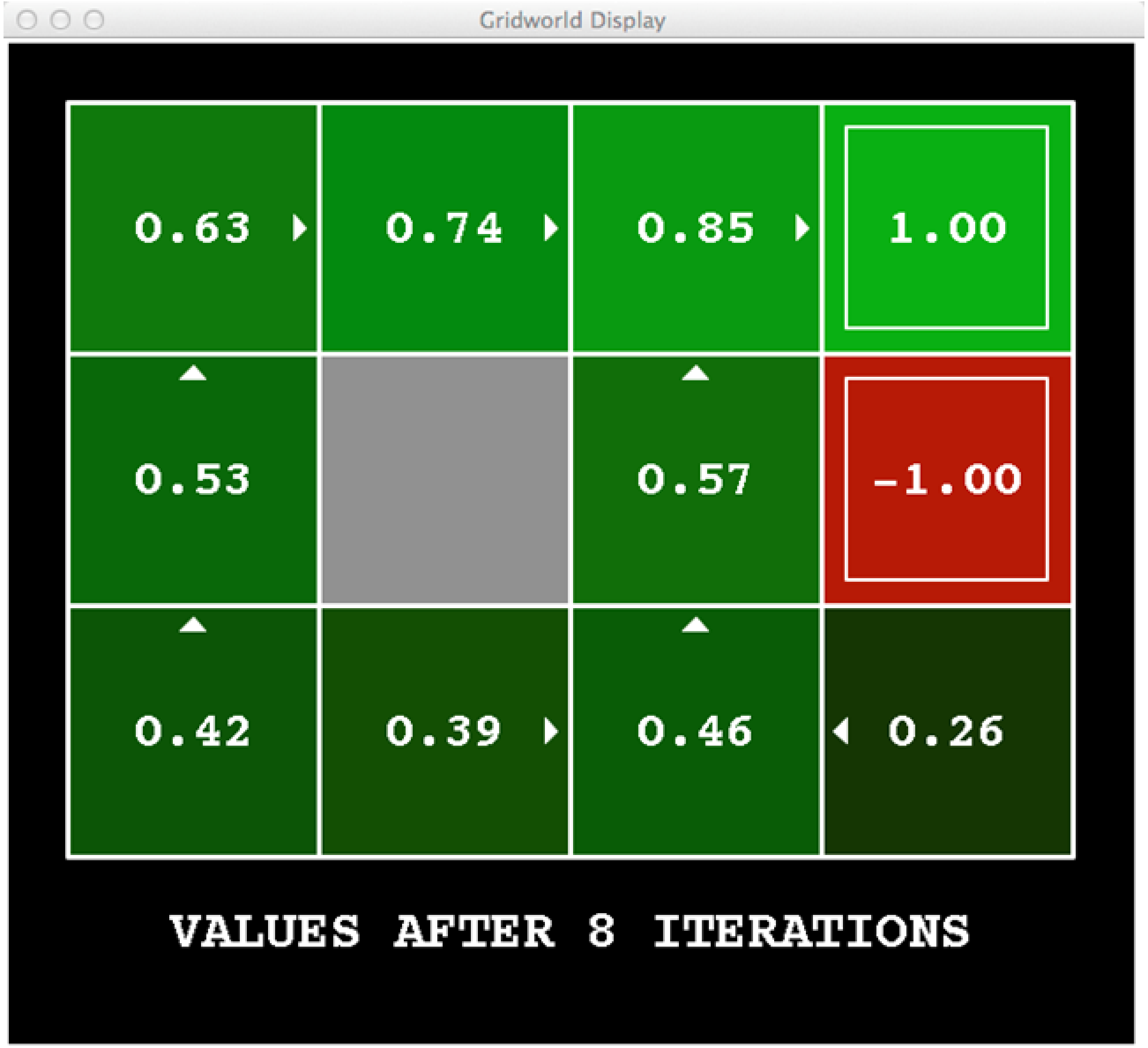
Value Iteration Example

Noise = 0.2
Discount = 0.9
Living reward = 0



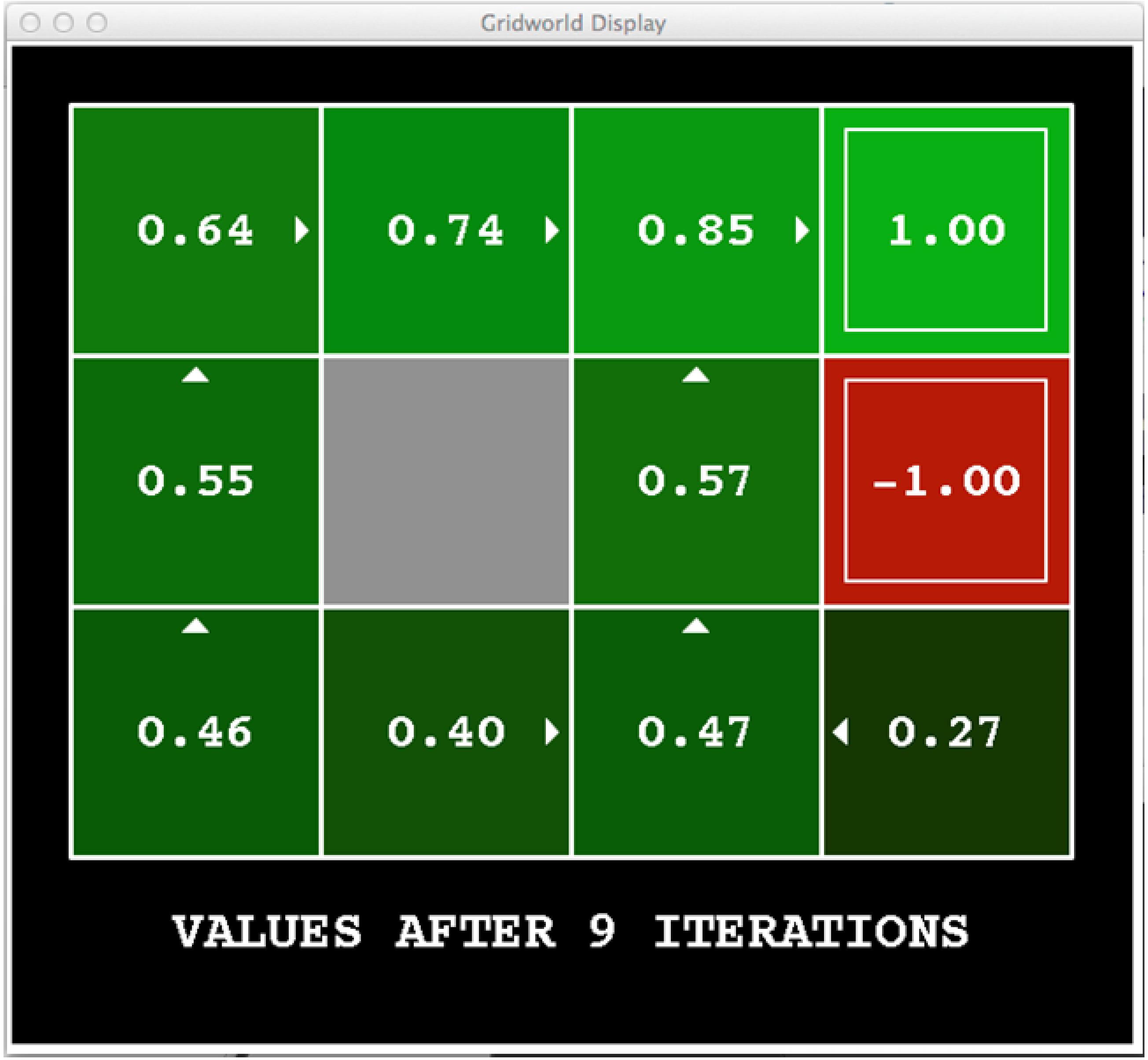
Value Iteration Example

Noise = 0.2
Discount = 0.9
Living reward = 0



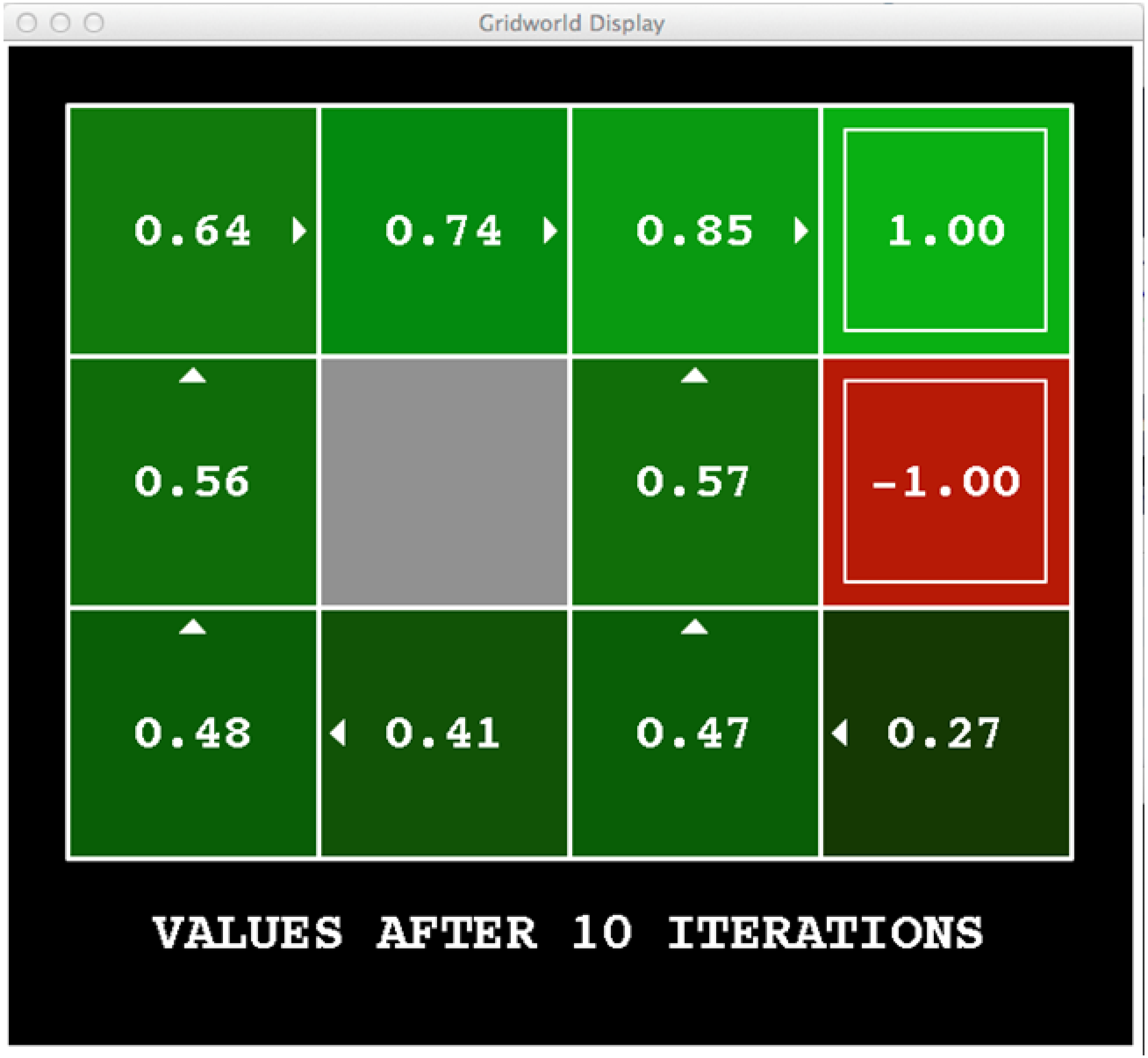
Value Iteration Example

Noise = 0.2
Discount = 0.9
Living reward = 0



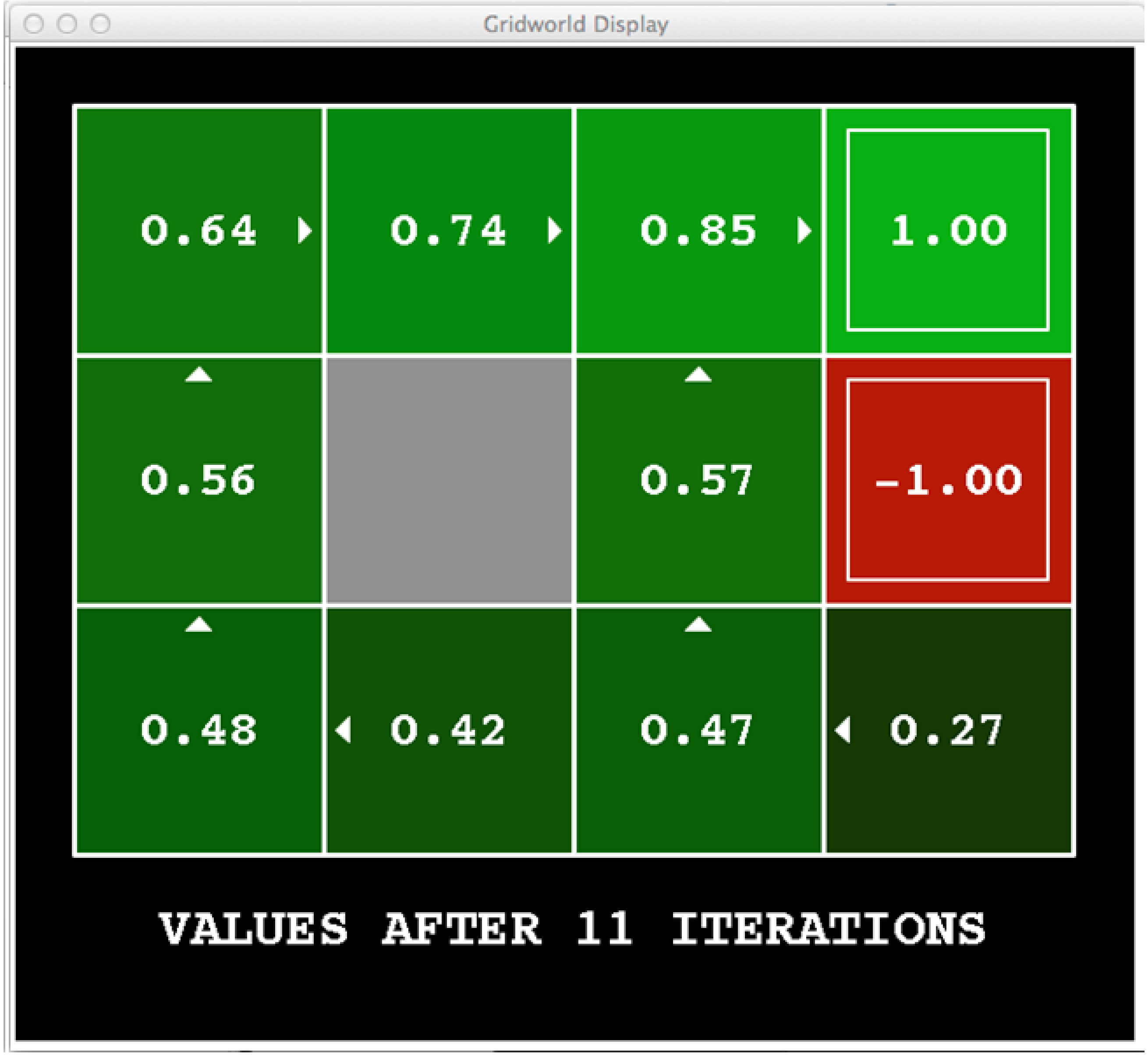
Value Iteration Example

Noise = 0.2
Discount = 0.9
Living reward = 0



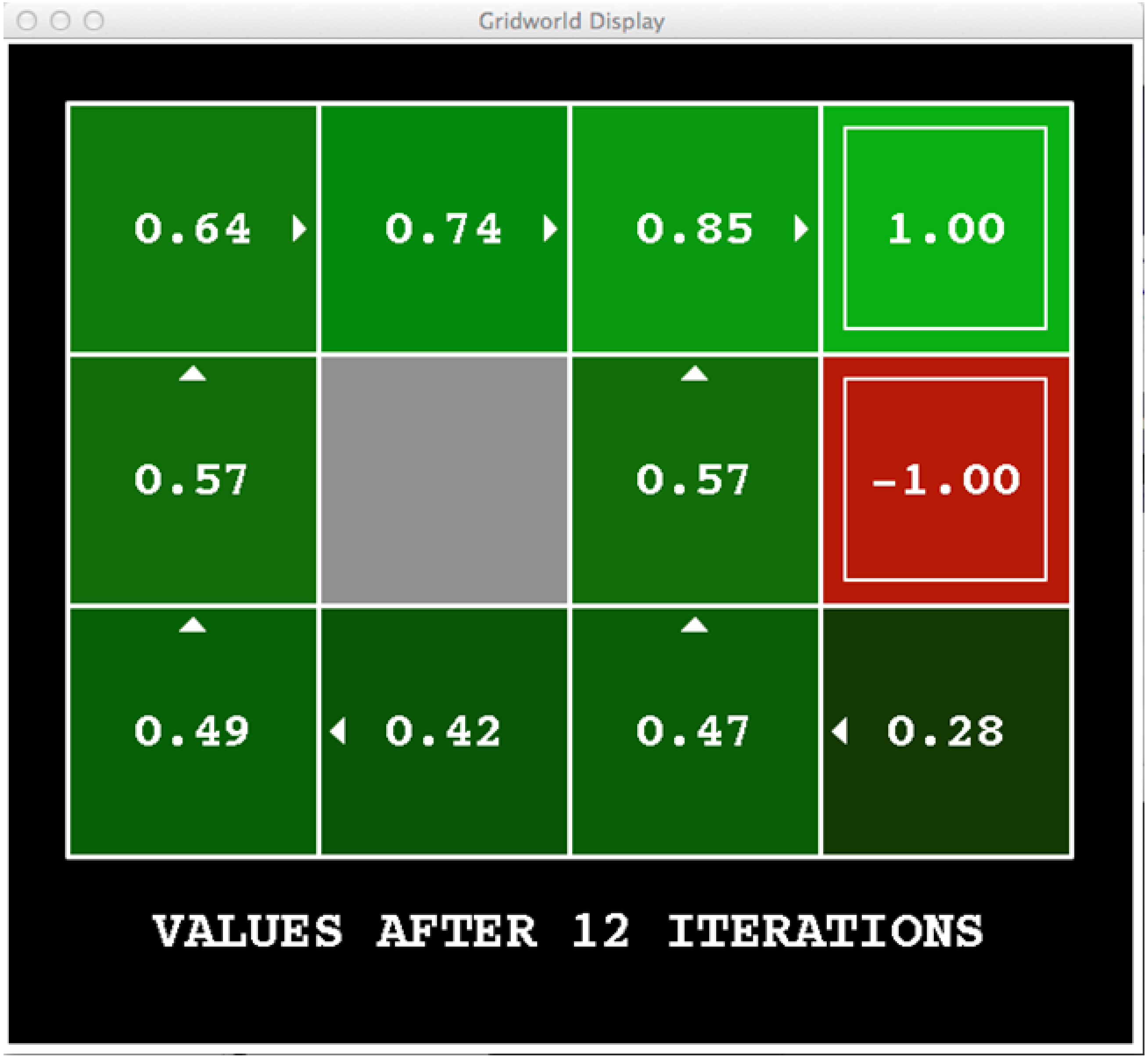
Value Iteration Example

Noise = 0.2
Discount = 0.9
Living reward = 0



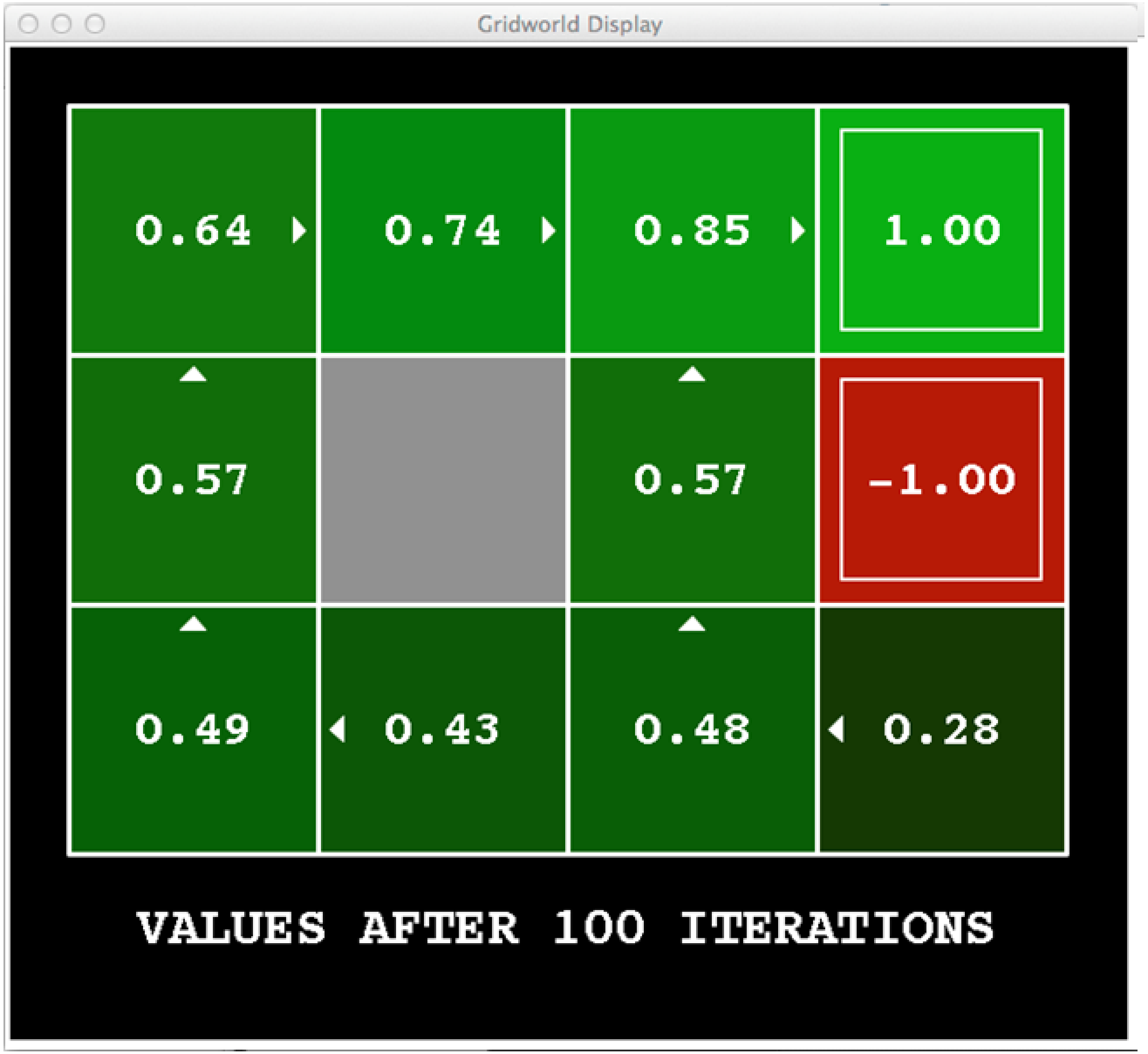
Value Iteration Example

Noise = 0.2
Discount = 0.9
Living reward = 0



Value Iteration Example

Noise = 0.2
Discount = 0.9
Living reward = 0



Value Iteration Convergence

- How do we know the V_k vectors are going to converge?
- Case 1: If the horizon has maximum depth M, then V_M holds the actual untruncated values (can think of this as an expectimax tree)
- Case 2: If the discount is less than 1

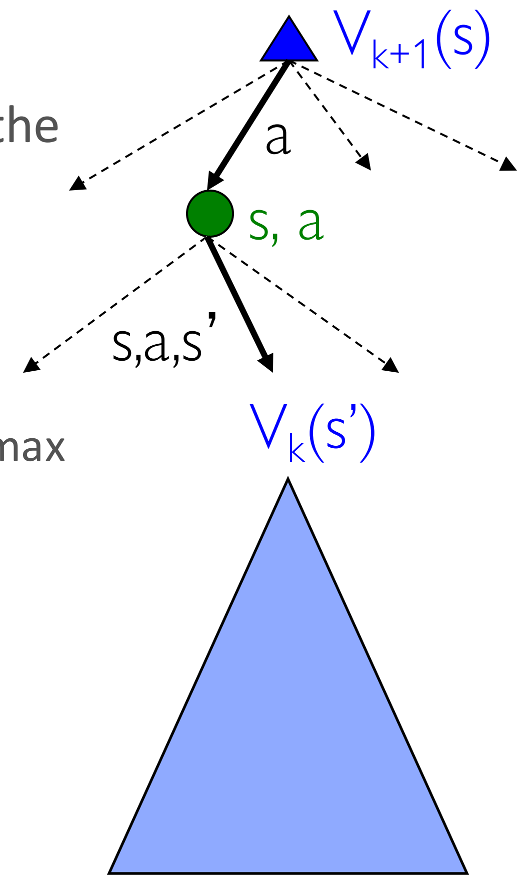
•Sketch: For any state V_k and V_{k+1} can be viewed as depth k+1 expectimax results in nearly identical search trees

•The last layer is at most all R_{MAX} and at least R_{MIN}

•But everything is discounted by γ^k that far out

– So V_k and V_{k+1} are at most $\gamma^k \max|R_{MAX} - R_{MIN}|$ different

•So as k increases, the values converge



Value Iteration Optimality

At convergence, this property must hold

$$V(s) = \max_a \sum_{s'} P(s', r|s, a)[r + \gamma V(s')]$$

Value Iteration Optimality

At convergence, this property must hold (why?)

$$V(s) = \max_a \sum_{s'} P(s', r|s, a)[r + \gamma V(s')]$$

Value Iteration Optimality

At convergence, this property must hold (why?)

$$V(s) = \max_a \sum_{s'} P(s', r|s, a)[r + \gamma V(s')]$$

Initialize array V arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)

Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s',r} p(s', r \,|\, s, a)\big[r + \gamma V(s')\big]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, $\pi \approx \pi_*$, such that
  $\pi(s) = \arg\max_a \sum_{s',r} p(s', r \,|\, s, a)\big[r + \gamma V(s')\big]$

Value Iteration Optimality

At convergence, this property must hold (why?)

$$V(s) = \max_a \sum_{s'} P(s', r|s, a)[r + \gamma V(s')]$$

What does this equation tell us about optimality of value iteration?

– we denote the *optimal* value function as: V^*

Value Iteration

$$V(s) \leftarrow \max_a \sum_{s',r} p(s', r \,|\, s, a)\big[r + \gamma V(s')\big]$$

Init: V(s) ← 0 for all s

Repeat until convergence:

For each state s:

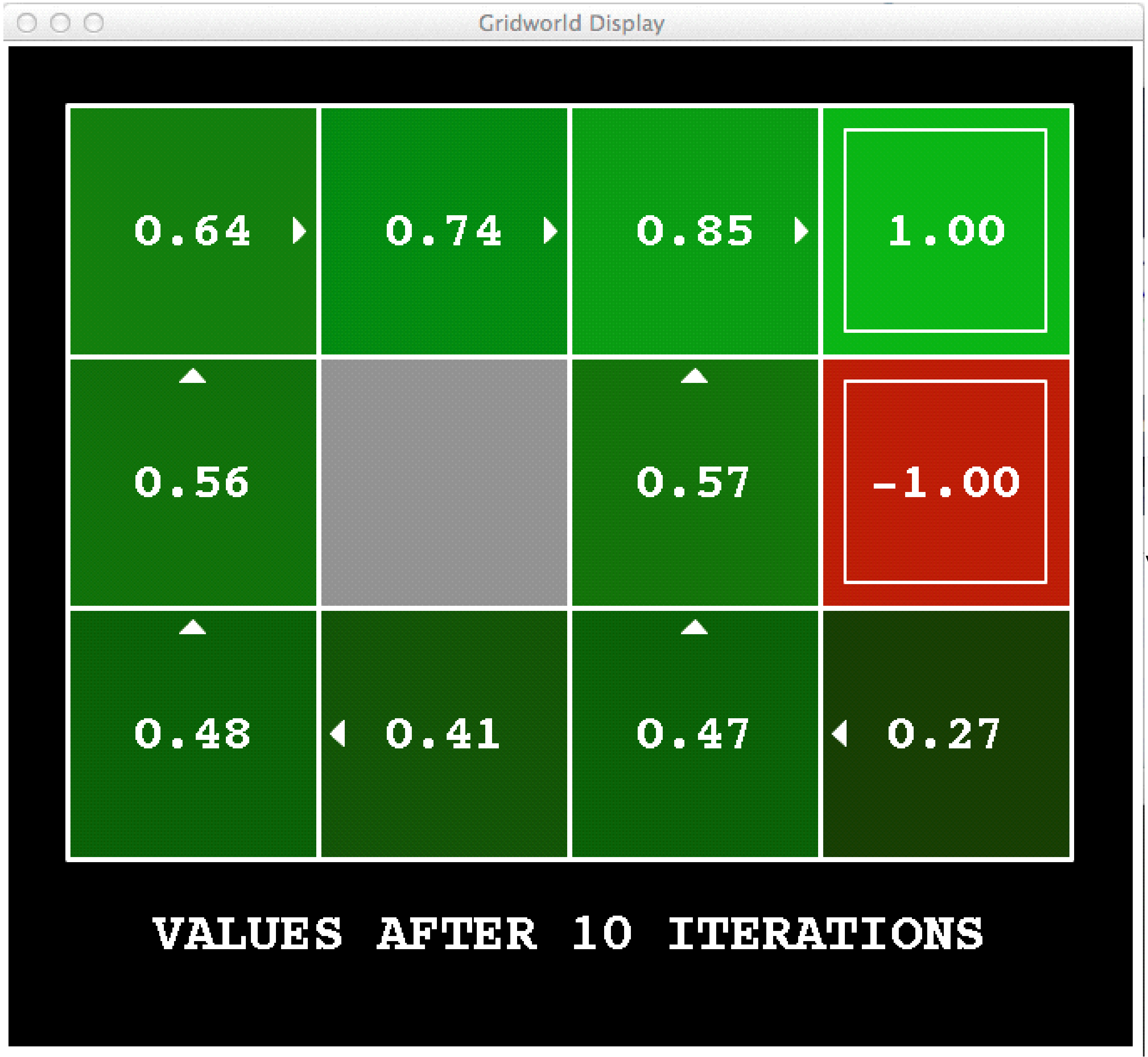
$$V(s) \leftarrow \max_a \sum_{s',r} p(s', r \,|\, s, a)\big[r + \gamma V(s')\big]$$

This performs 1 **backup** operation

- Convergence: Typically (all values do not change much)

$$\max_s |V_{k+1}(s) - V_k(s)| < \epsilon$$

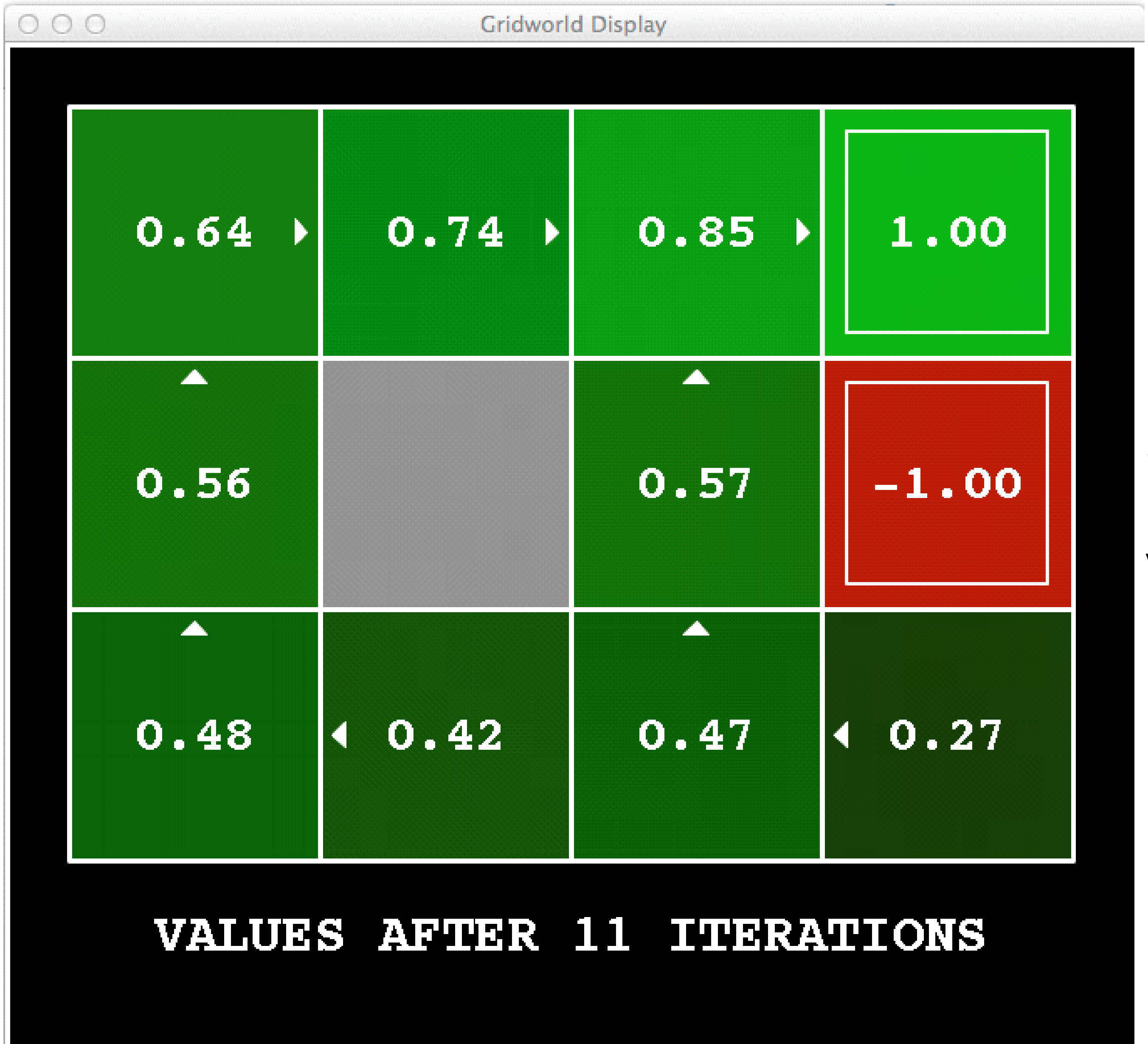
Value Iteration: Watch the policy



R(s) = 0 for non-terminal s

γ = 0.9

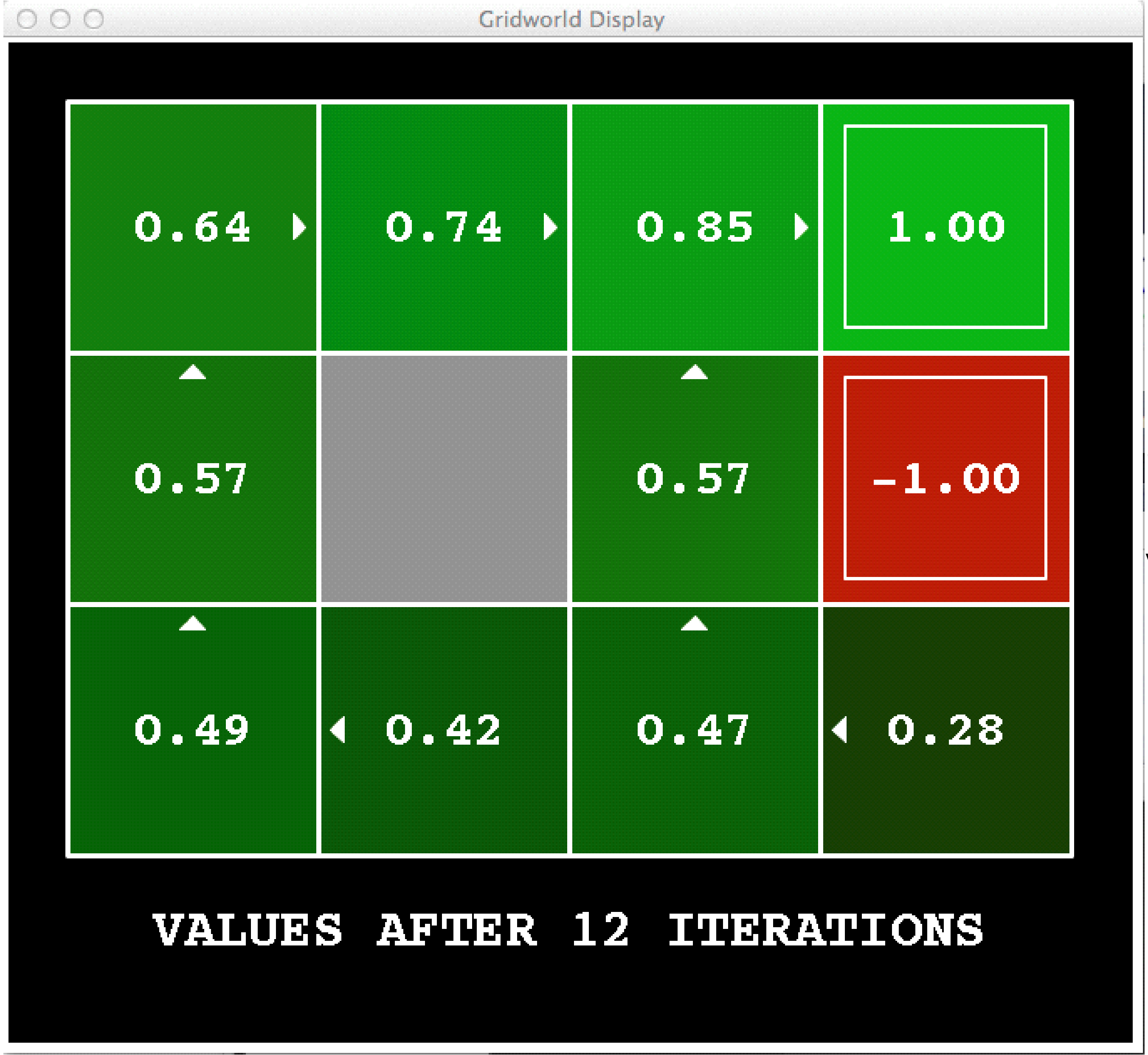
Value Iteration: Watch the policy



R(s) = 0 for non-terminal s

γ = 0.9

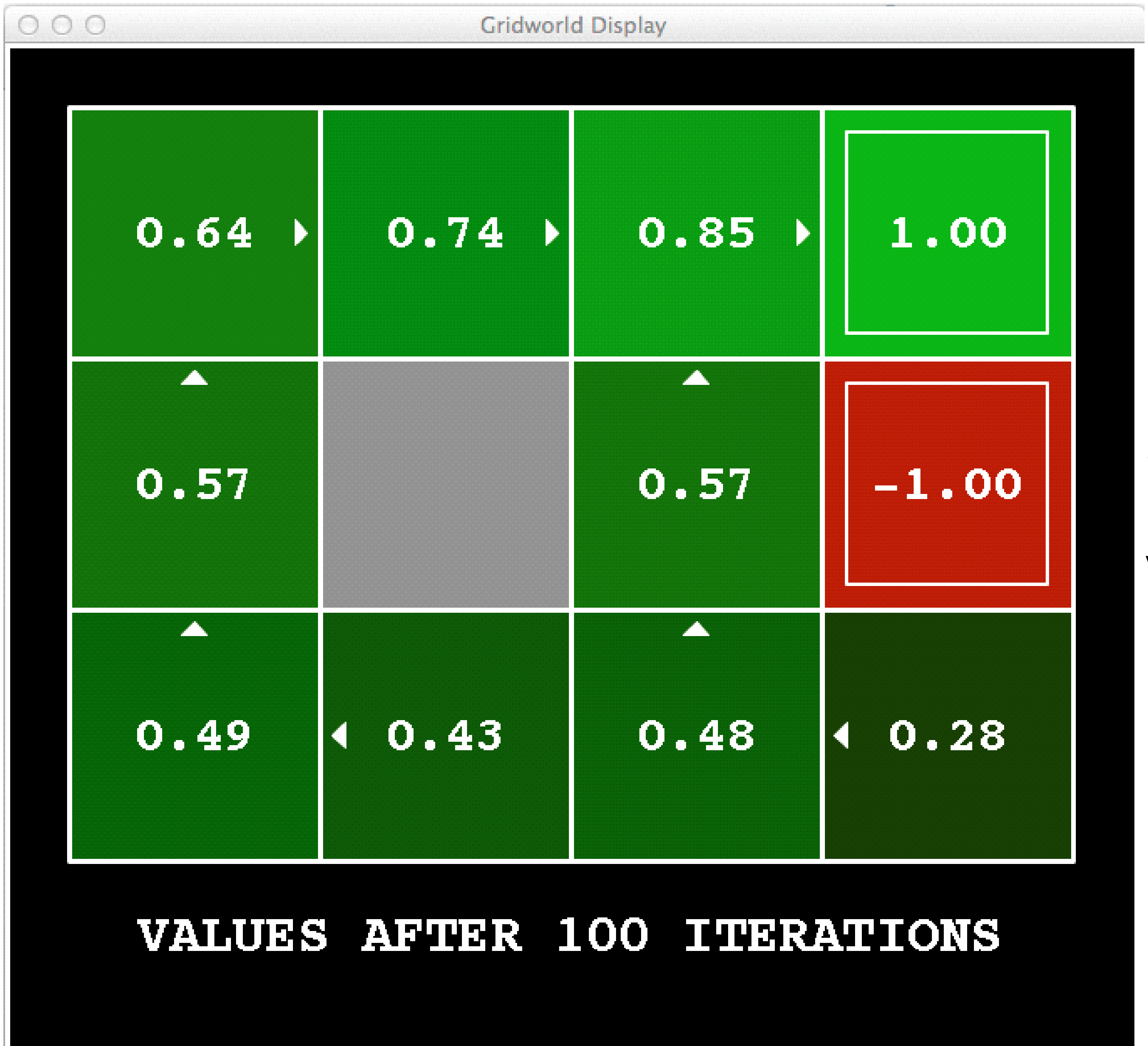
Value Iteration: Watch the policy



R(s) = 0 for non-terminal s

$\gamma = 0.9$

Value Iteration: Watch the policy



R(s) = 0 for non-terminal s

$\gamma = 0.9$

Value Iteration vs Policy Iteration

Notice anything interesting about the policy during VI?
(Look at the grid-world example again)

Policy converges <u>much faster</u> than the value function
- Getting the exact values do not matter
  if the strategy to act is the same

In <u>value</u> iteration, we iteratively compute the value function
In <u>policy</u> iteration, we iteratively compute the policy

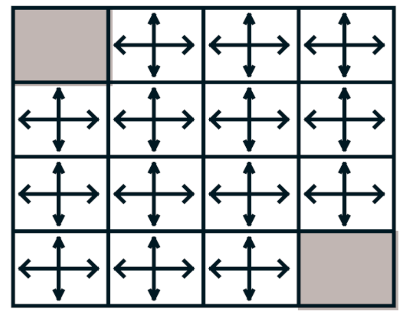
$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_*,$$

where $\xrightarrow{\text{E}}$ denotes a policy *evaluation* and $\xrightarrow{\text{I}}$ denotes a policy *improvement*

Generalized Policy Iteration

1. Initialization
   - All values = 0 , policy = random

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



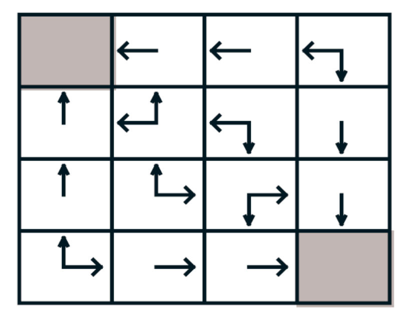
random policy

2. Iterative policy evaluation

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

(values have not converged yet,
but assume they are close at this point)

3. Policy improvement

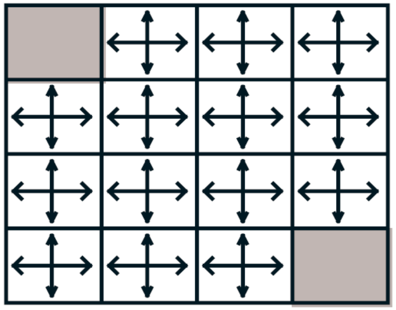


Generalized Policy Iteration

1. Initialization

- All values = 0 , policy = random

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



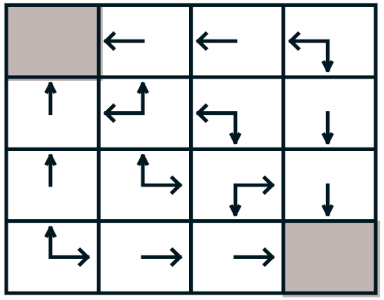
random policy

2. Iterative policy evaluation

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

3. Policy improvement



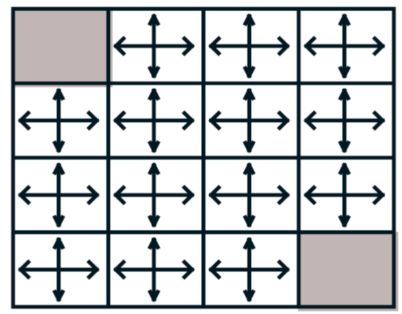
If we waited long enough, we get this value function

Generalized Policy Iteration

1. Initialization

- All values = 0 , policy = random

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



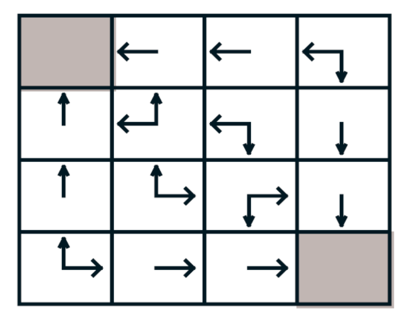
random policy

2. Iterative policy evaluation

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

3. Policy improvement



If we waited long enough, we get this value function

- The greedy policy for both is the same!

Variation: Generalized policy iteration

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop: **for some number of iterations k**
   $\Delta \leftarrow 0$
   Loop for each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r \,|\, s, \pi(s)) \big[r + \gamma V(s')\big]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   ~~until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)~~

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   old-action $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \,|\, s, a) \big[r + \gamma V(s')\big]$
   If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Variation: Generalized policy iteration

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop: **for some number of iterations k**
   $\Delta \leftarrow 0$
   Loop for each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r \,|\, s, \pi(s))\big[r + \gamma V(s')\big]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   ~~until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)~~

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   old-action $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \,|\, s, a)\big[r + \gamma V(s')\big]$
   If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

If k = 1, equivalent to value iteration

Variation: Asynchronous dynamic programming

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop: **for some number of iterations k**
   - $\Delta \leftarrow 0$
   - Loop ~~for each $s \in \mathcal{S}$:~~ **for some states s**
     - $v \leftarrow V(s)$
     - $V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s))\big[r + \gamma V(s')\big]$
     - $\Delta \leftarrow \max(\Delta, |v - V(s)|)$

   ~~until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)~~

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   - *old-action* $\leftarrow \pi(s)$
   - $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
   - If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*

   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

State-update order and relative update frequency can be arbitrary, as long as all states visited infinitely often

Policy Iteration convergence

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

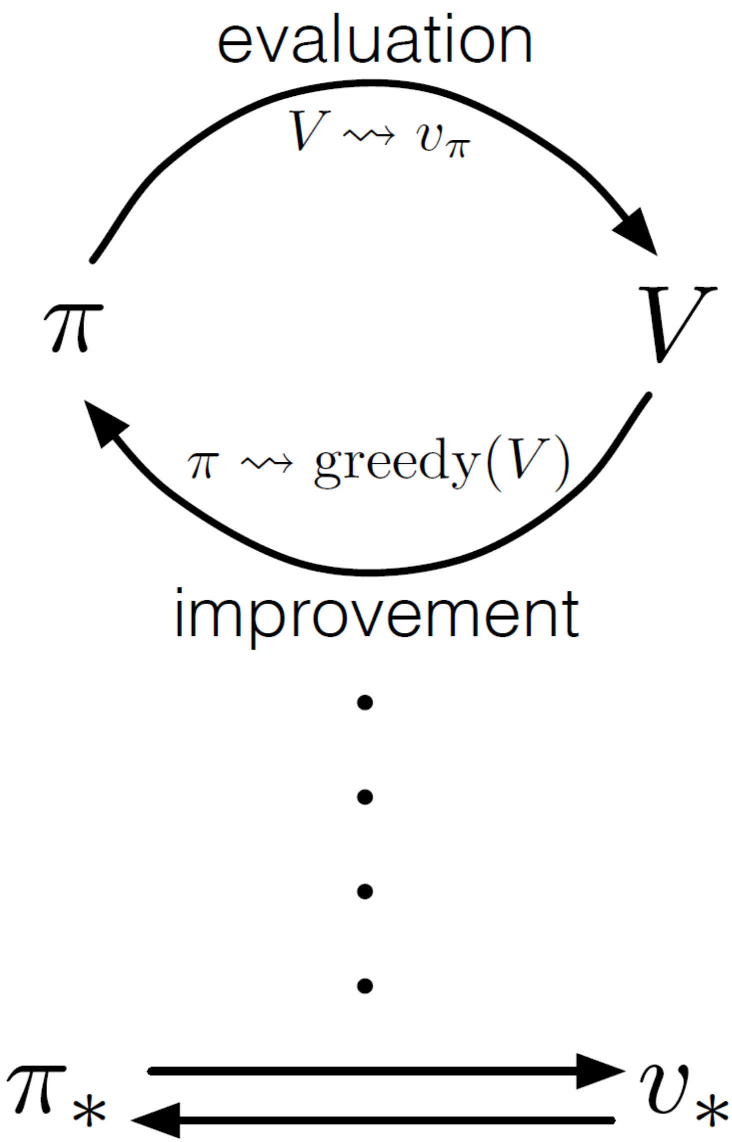
1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
   $\quad \Delta \leftarrow 0$
   $\quad$ Loop for each $s \in \mathcal{S}$:
   $\quad\quad v \leftarrow V(s)$
   $\quad\quad V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s)) \big[r + \gamma V(s')\big]$
   $\quad\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $\quad$ *old-action* $\leftarrow \pi(s)$
   $\quad \pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a) \big[r + \gamma V(s')\big]$
   $\quad$ If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Guaranteed to converge to π*

Generalized (and Asynchronous) Policy Iteration



"Almost all reinforcement learning methods are well described as GPI. [GPI = Generalized Policy Iteration]

That is, all have identifiable
policies and value functions,

with the policy
always being improved
with respect to the value function

and the value function
always being driven toward
the value function for the policy,

as suggested by the diagram."

Computational efficiency of VI and PI

- To find an optimal policy is polynomial in the number of states…
- BUT, the number of states is often astronomical, e.g., often growing exponentially with the number of state variables (what Bellman called "the curse of dimensionality")
- In practice, classical DP can be applied to problems with a few millions of states
- Asynchronous DP can be applied to larger problems, and is appropriate for parallel computation
- It is surprisingly easy to come up with MDPs for which DP methods are not practical

- Nevertheless, VI and PI are the basis for many more scalable algorithms that we will discuss next

Summary: Dynamic programming

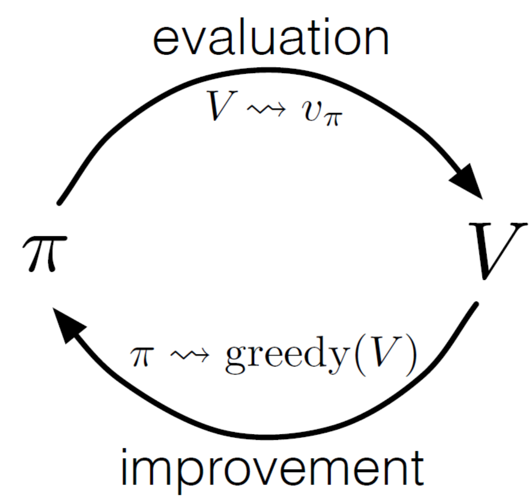
- Value iteration

$$V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V(s')\right]$$

- Policy evaluation

$$V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s))\left[r + \gamma V(s')\right]$$

- Policy iteration



"Almost all reinforcement learning methods are well described as generalized policy iteration."

$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_*,$$

where $\xrightarrow{\text{E}}$ denotes a policy *evaluation* and $\xrightarrow{\text{I}}$ denotes a policy *improvement*

Next time

- Monte Carlo methods
- Getting rid of the assumption of a known model (i.e., P(s'|s,a) and R(s,a))