Reinforcement Learning Fundamentals

Lecture 14: Value Iteration and Policy Iteration

Dr Sandeep Manjanna
Assistant Professor, Plaksha University
sandeep.manjanna@plaksha.edu.in

Announcements

- Project Proposal due at 10:00 pm tonight **23rd Feb 2024.**

In today's class…

Until now…

- Bellman Optimality:
  - Optimal Value function and Optimal Policy
- Policy Extraction

- Value Iteration
- Policy Iteration

Bellman Optimality Equation for v_*

$$v_\pi(s) = \sum_a \pi(a|s) \underbrace{\sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_\pi(s')\Big]}_{q_\pi(s,a)}, \quad \text{for all } s \in \mathcal{S},$$

The value of a state under an optimal policy must equal the expected return for the best action from that state:

$$\begin{aligned} v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a) \\ &= \max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma v_*(s')\big]. \end{aligned}$$

Similarly,

$$\begin{aligned} q_*(s,a) &= \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1},a') \,\Big|\, S_t = s, A_t = a\Big] \\ &= \sum_{s',r} p(s',r|s,a)\Big[r + \gamma \max_{a'} q_*(s',a')\Big]. \end{aligned}$$

How to solve these MDPs?

Dynamic Programming

- Stochastic dynamic programming represents the problem in the **form of a Bellman equation**.
- The aim is to **compute a policy** prescribing how to act optimally in the face of uncertainty.
- We will look at DP-based approaches to find V_π and V_*.
- Needs complete specification of system dynamics. Hence, does not scale well.

How to solve these MDPs?
Dynamic Programming

- DP is the solution method of choice for MDPs
  - Requires complete knowledge of system dynamics (transition matrix and rewards)
  - Computationally expensive
  - Curse of dimensionality
  - Guaranteed to converge!

Going from π to V_π

- We will use DP approaches to solve both **evaluation** and **control**.

Finding V_*

Value Iteration

- Turning the Bellman optimality equation into an update rule.

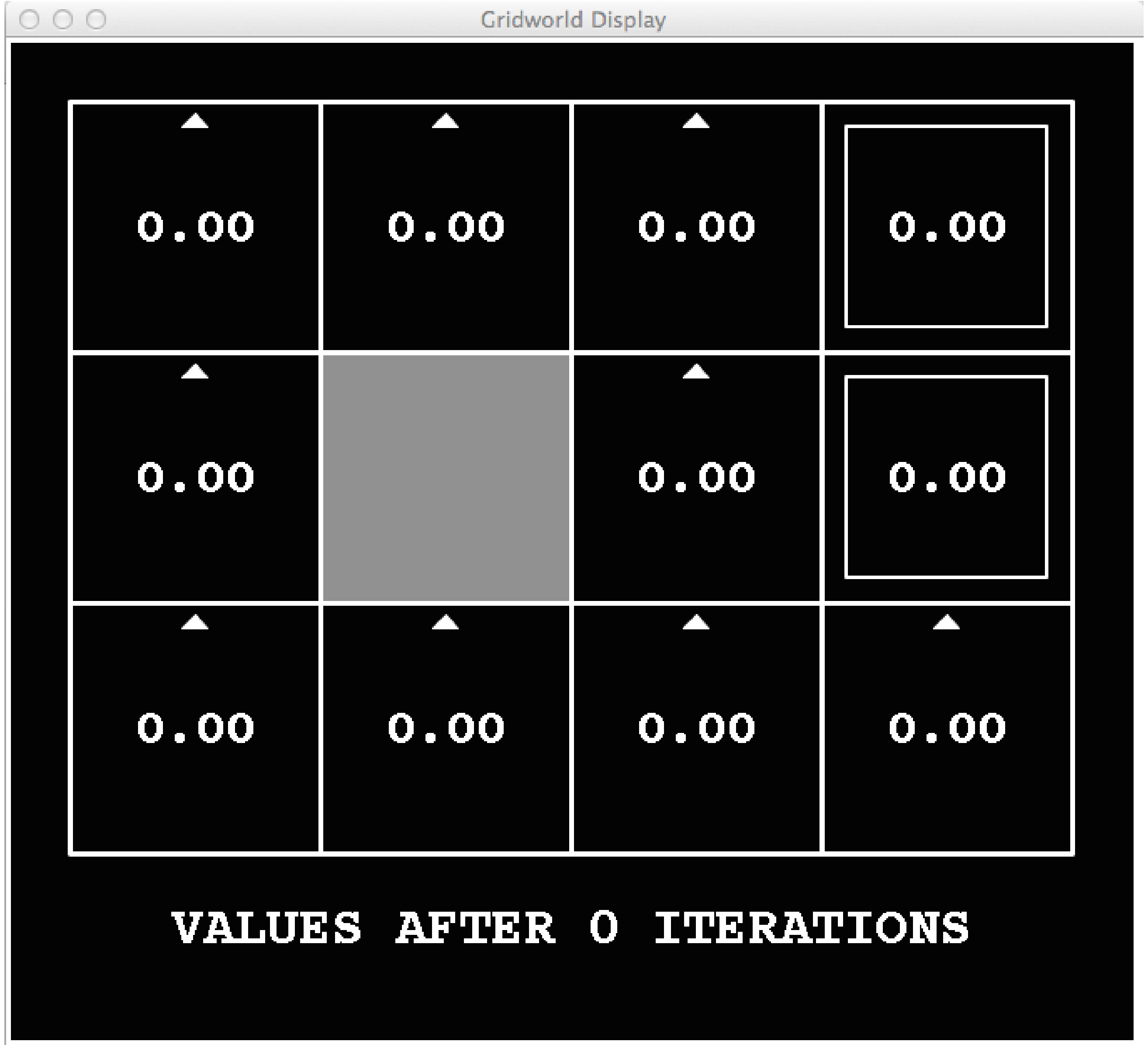
$$
\begin{aligned}
v_*(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma v_*(s')\big].
\end{aligned}
$$

$$
\begin{aligned}
v_{k+1}(s) &\doteq \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \max_a \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma v_k(s')\Big],
\end{aligned}
$$

Every iteration, we look at 1-step additional expected return.

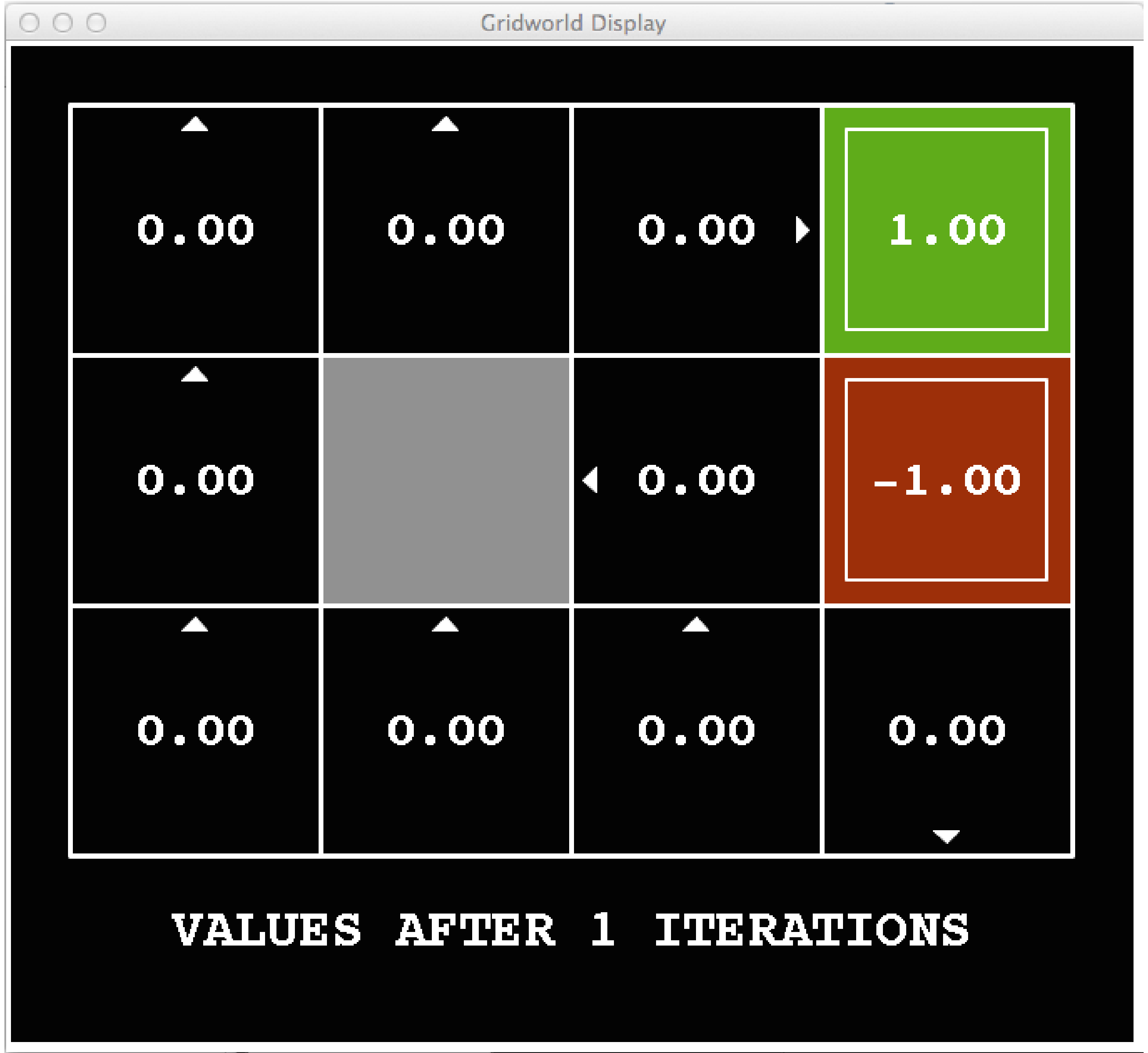
First solve for zero step problem, use that to solve the 1-step problem, in-turn use that to solve the 2-step problem…

Value Iteration Example



Noise = 0.2
Discount = 0.9
Living reward = 0

Value Iteration Example



Noise = 0.2
Discount = 0.9
Living reward = 0

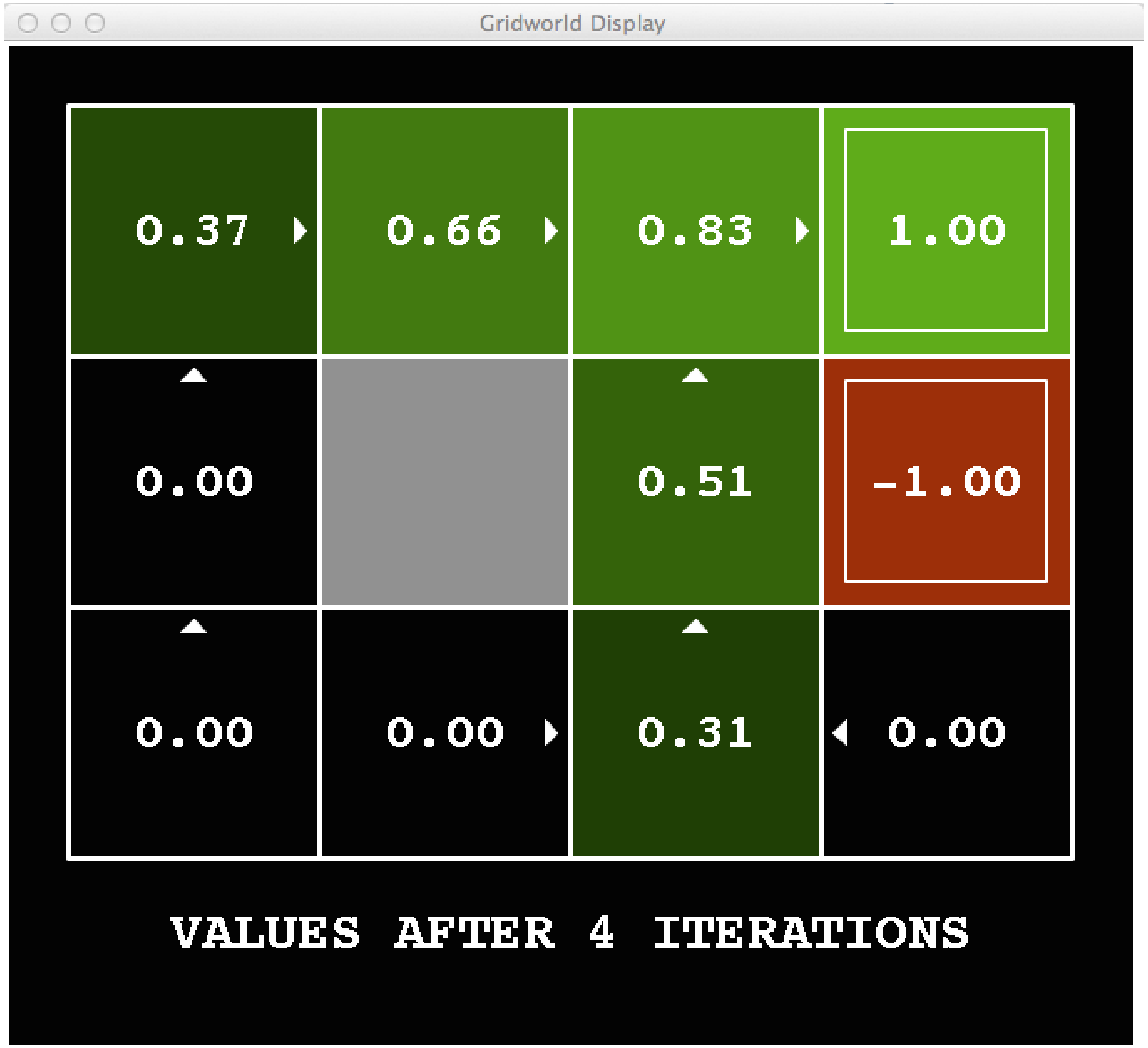
Value Iteration Example

Noise = 0.2
Discount = 0.9
Living reward = 0

Value Iteration Example

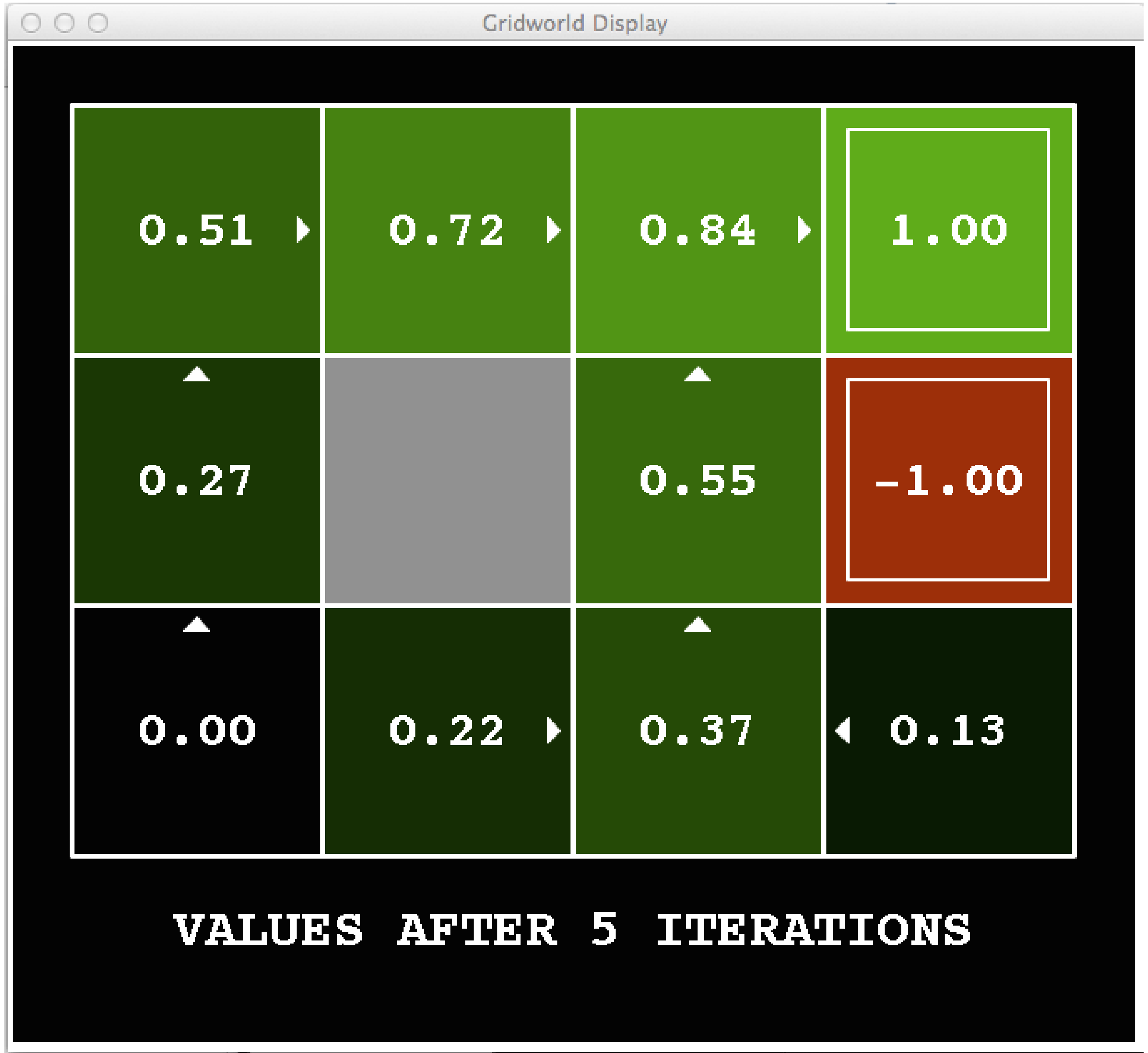
Noise = 0.2
Discount = 0.9
Living reward = 0

Value Iteration Example



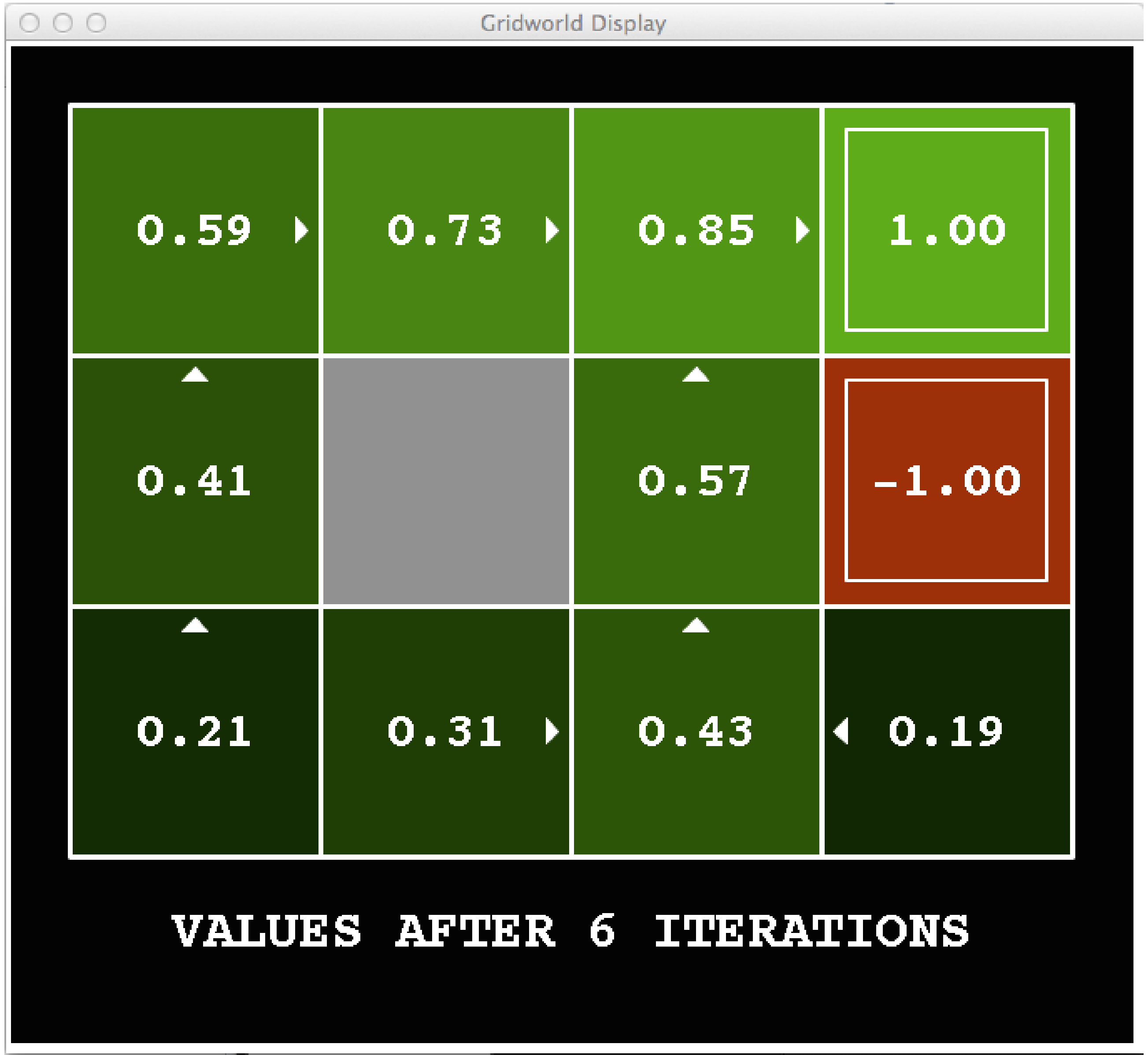
Noise = 0.2
Discount = 0.9
Living reward = 0

Value Iteration Example



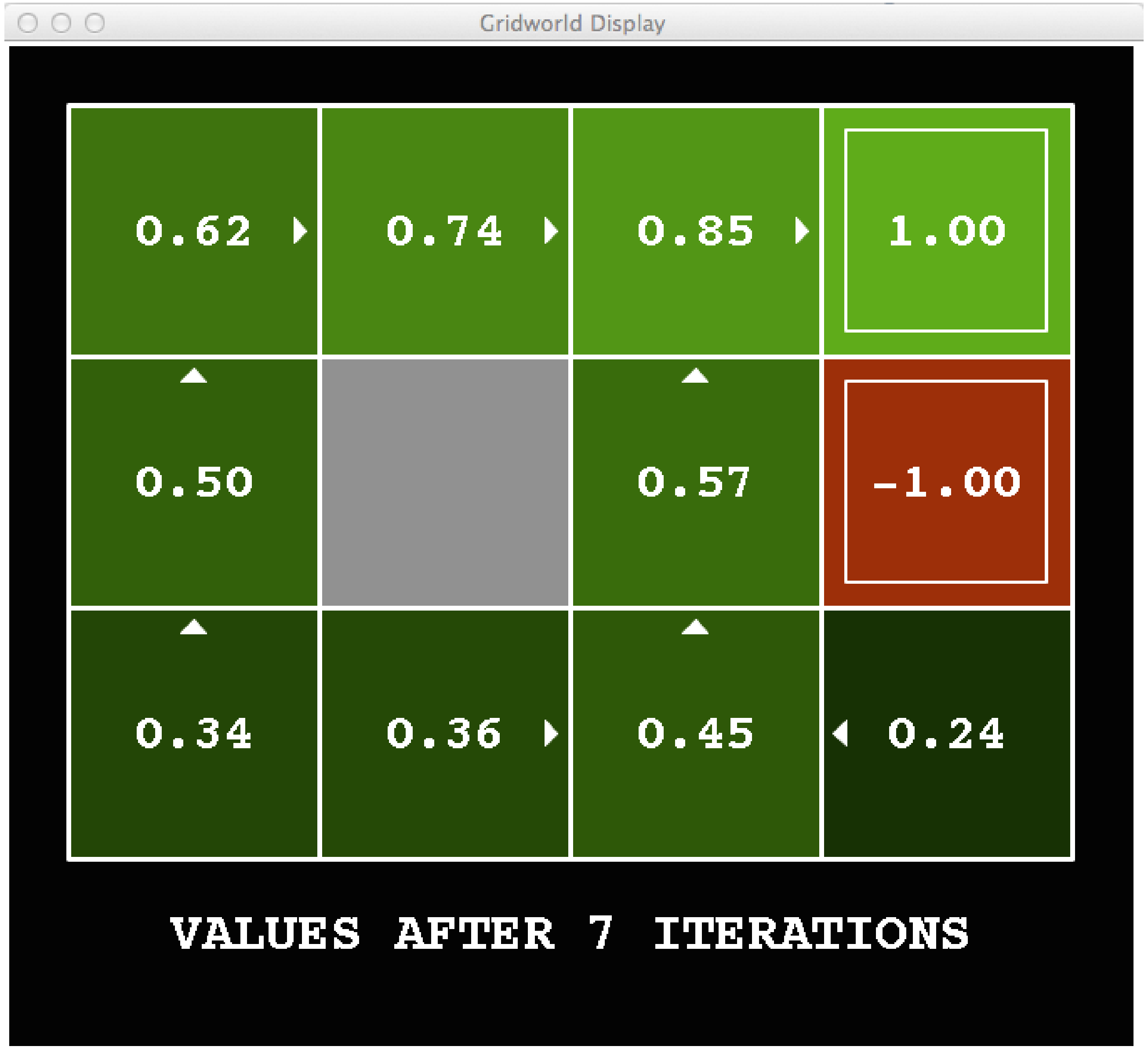
Noise = 0.2
Discount = 0.9
Living reward = 0

Value Iteration Example



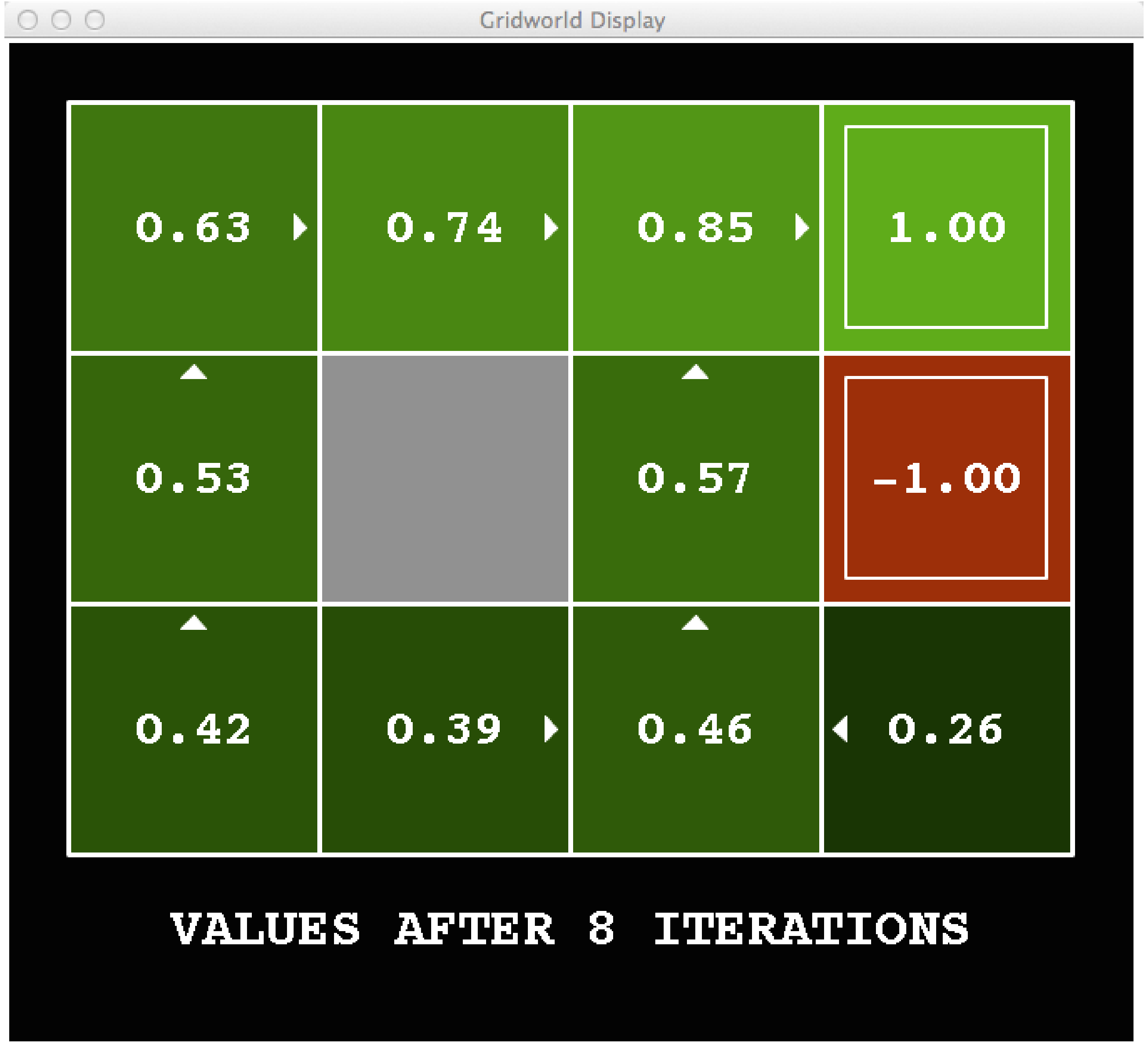
Noise = 0.2
Discount = 0.9
Living reward = 0

Value Iteration Example



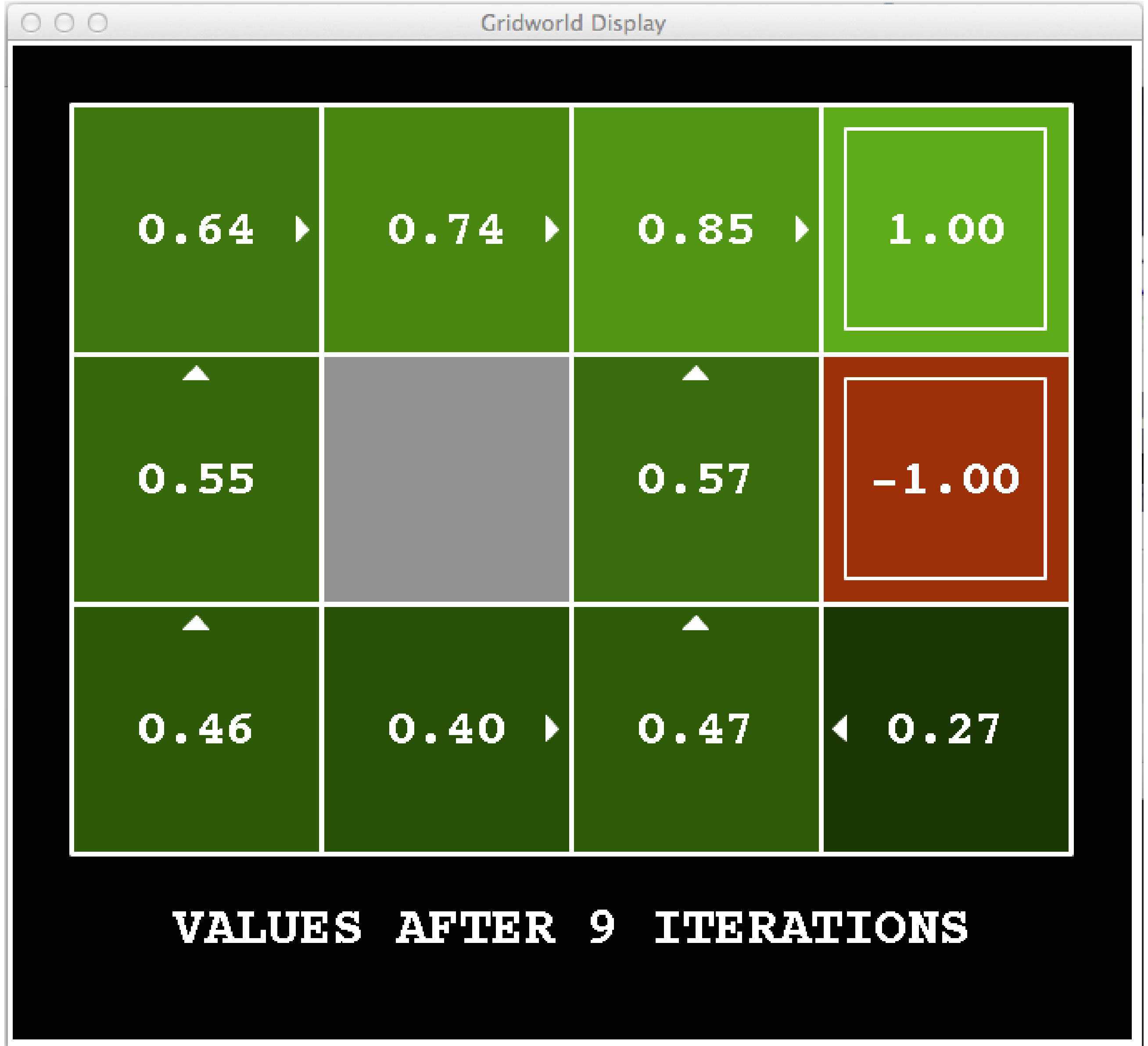
Noise = 0.2
Discount = 0.9
Living reward = 0

Value Iteration Example



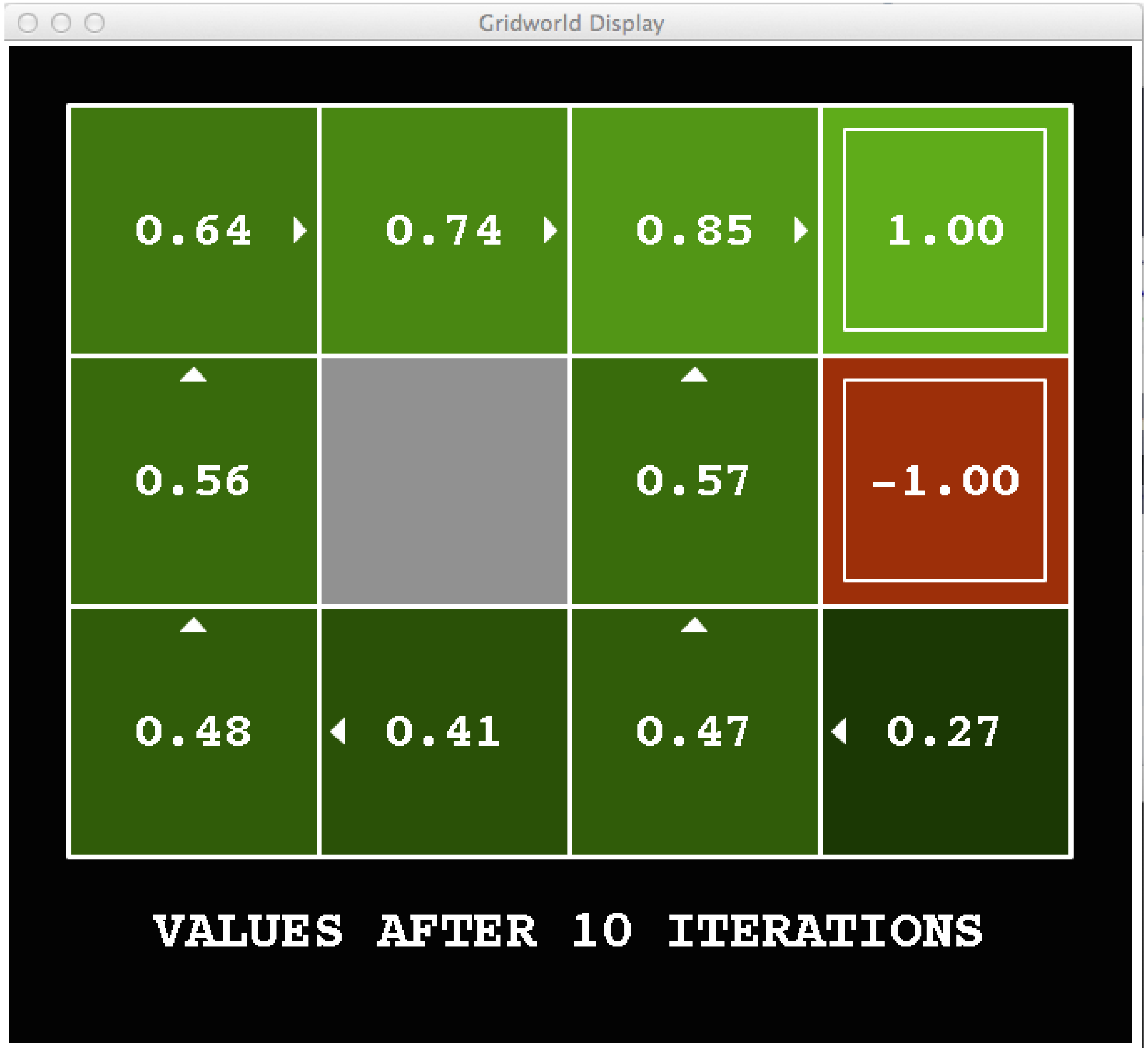
Noise = 0.2
Discount = 0.9
Living reward = 0

Value Iteration Example



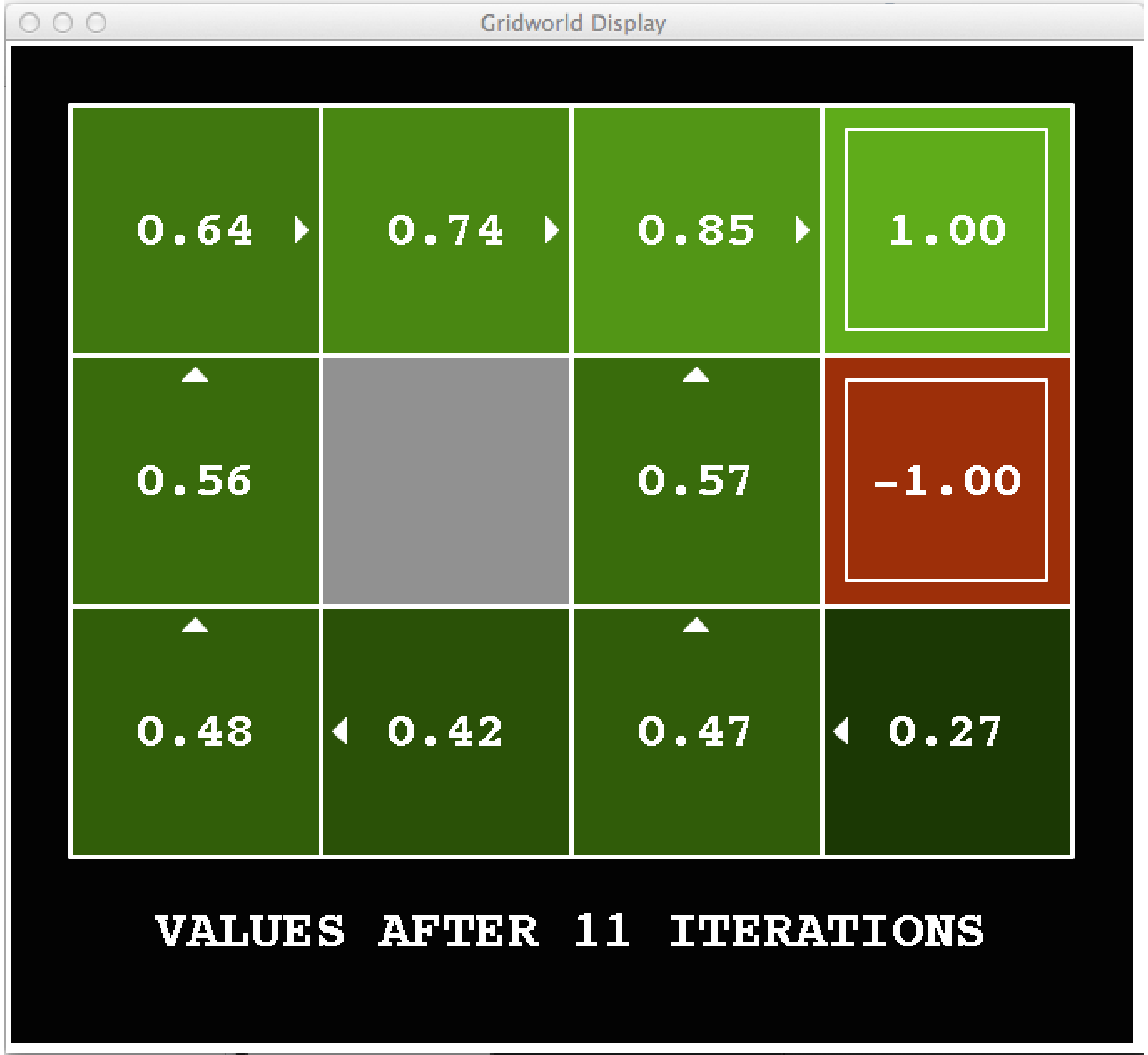
Noise = 0.2
Discount = 0.9
Living reward = 0

Value Iteration Example



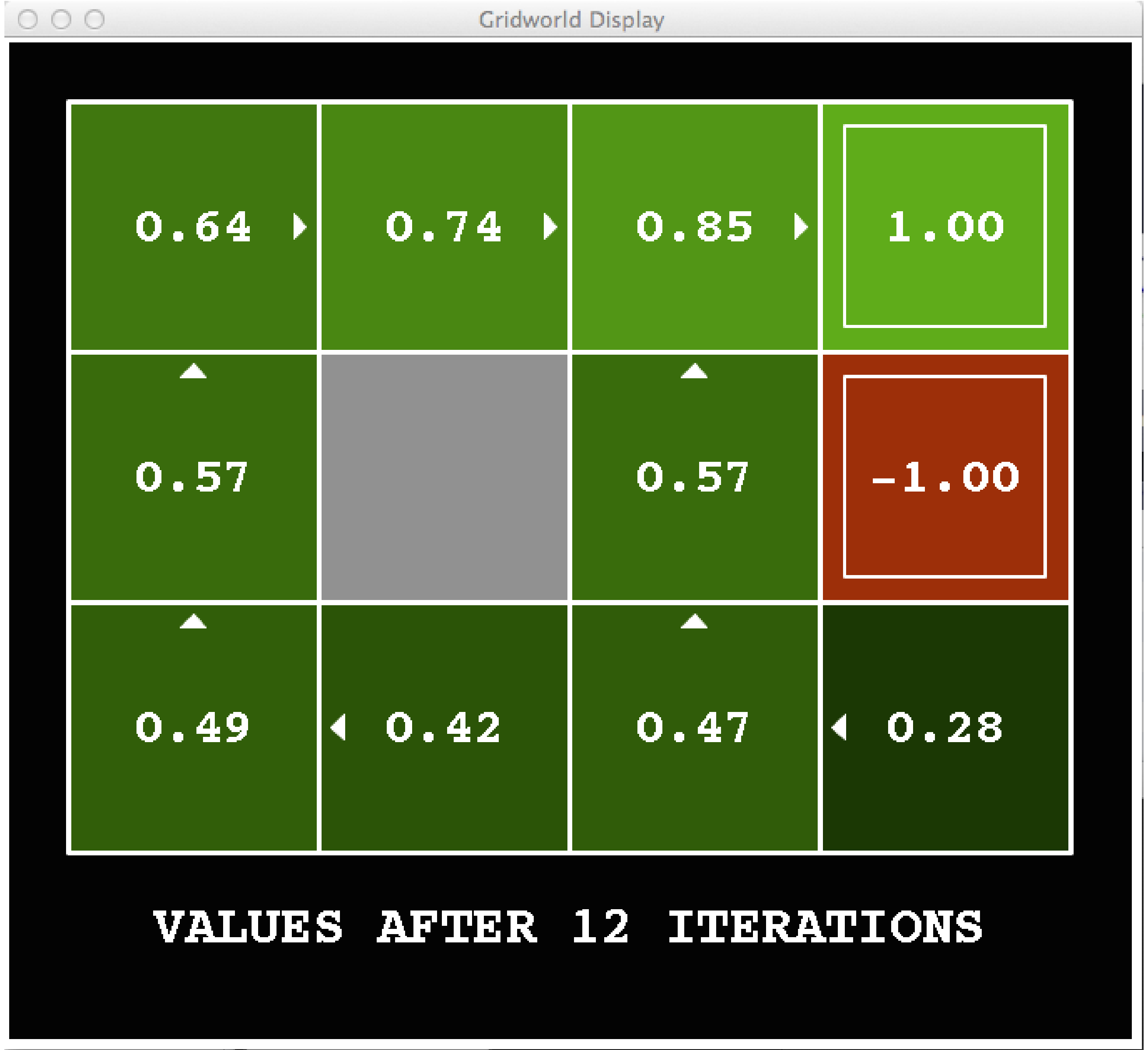
Noise = 0.2
Discount = 0.9
Living reward = 0

Value Iteration Example



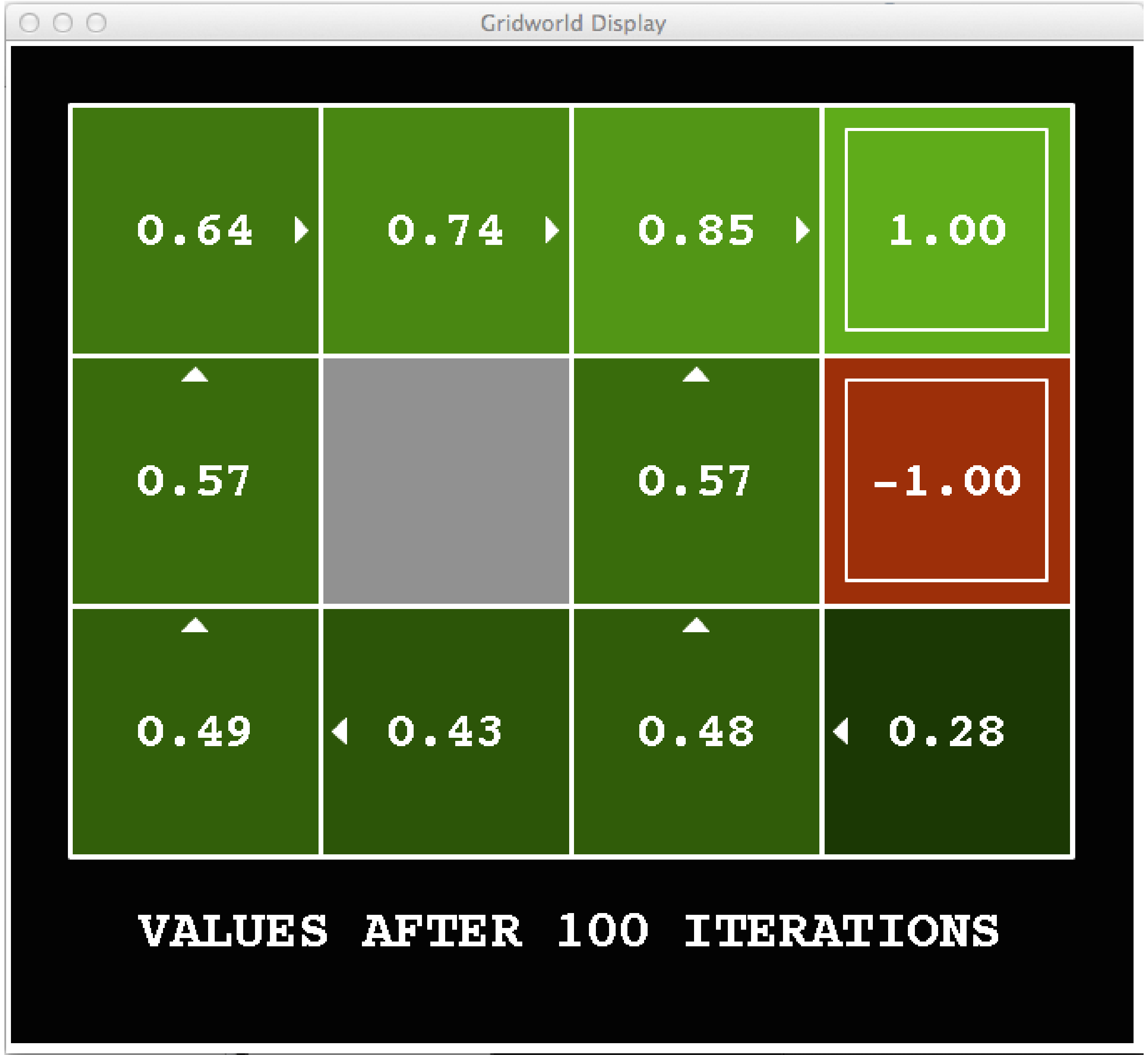
Noise = 0.2
Discount = 0.9
Living reward = 0

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

Assumption is that the ***S*** and ***A*** are small enough so that we can loop over all ***s ϵ S*** and ***a ϵ A***.

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop 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 $\Delta < \theta$

In-place updates.
It results in asynchronous computation.

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

Implement Value Iteration on a simple grid-world example and compare the effect of discount factor, size of the world, convergence threshold, reward function, and transition probability on the values achieved.