MACHINE LEARNING AND APPLICATIONS GROUP

The notion and solving of known MDPs

Nikola Popović

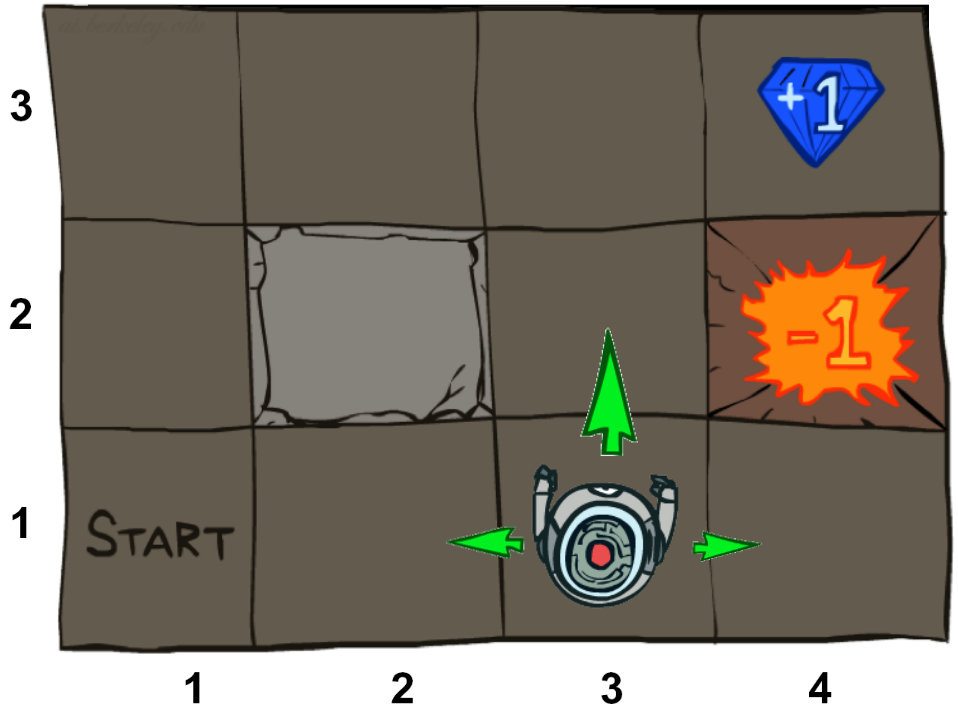
RL vs. Supervised learning

- Supervised: For each $x^{(i)}$ we know the correct $y^{(i)}$
- Sequential problems:
  - We only know how good the outcome is
  - We do not know how good each action is
  - Examples: Chess, robot control, …
- RL tries to learn which actions are good in which states, based on a lot of attempts

This slide has been taken from [3]

Example: Grid world

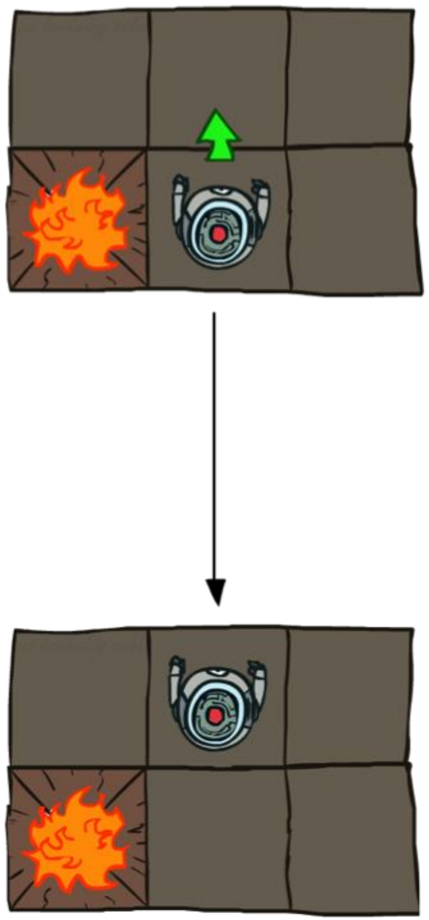
- We start at the state (1,1)
- We can move North, South, East, West
- Noisy movement:
  - 80% of the time, the action North takes the agent North (if there is no wall there)
  - 10% of the time, North takes the agent West; 10% East
  - If there is a wall in the direction the agent would have been taken, the agent stays put
- Rewards:
  - Small reward at each step (living cost)
  - Big reward at terminal states (termination cost)
- Goal: Maximize sum of rewards



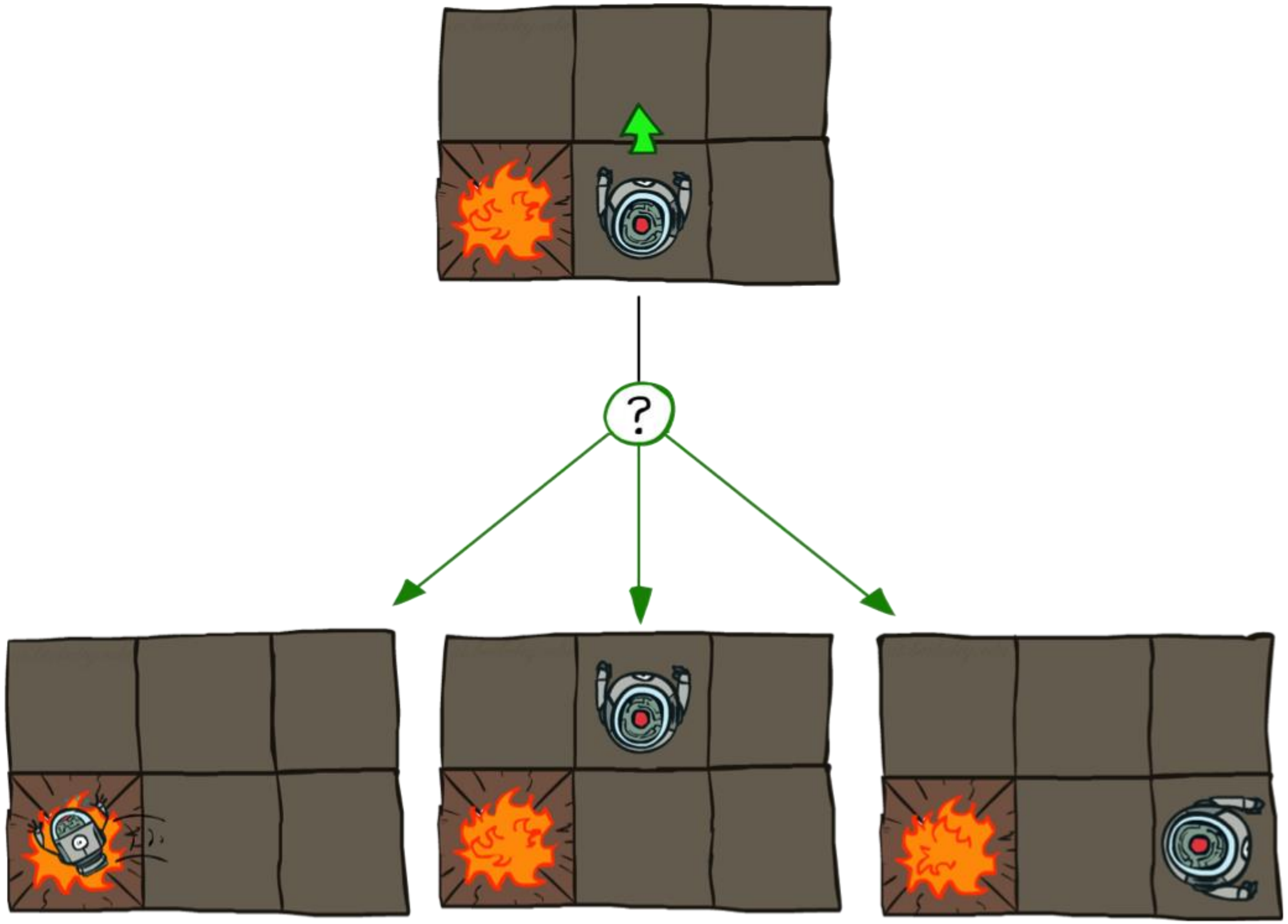
Picture taken from [1]

Grid World Actions

Deterministic Grid World



Stochastic Grid World



Pictures taken from [1]

MDP

- Sequential decision problem
- Fully observable environment
- Stochastic environment: $P(s_{t+1}|s_t, a_t)$
- Markovian transitions:
$$P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1} \dots, s_0, a_0) = P(s_{t+1}|s_t, a_t)$$
- Utility as a (discounted) sum of rewards:
$$U([s_0, s_1, s_2, \dots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots$$

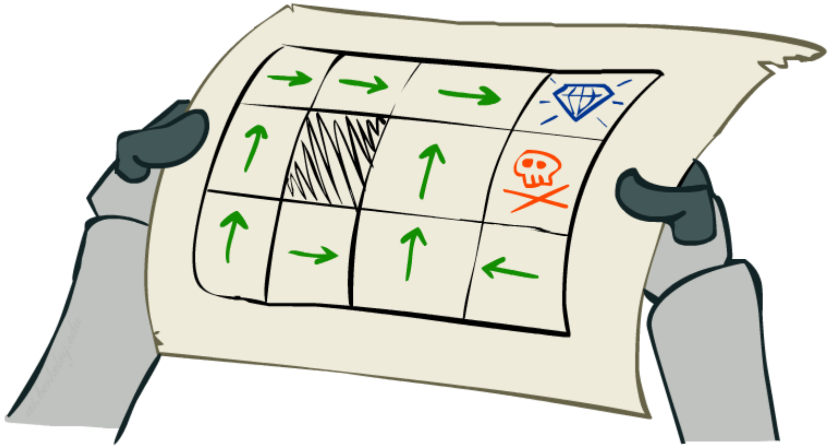
Elements of a MDP

- States s
- Actions a
- Transition model $P_{sa}(s') = P(s'|s,a)$
  - Probability that applying a in s leads to s'
- Reward function $R(s)$
  - Could also be $R(s,a,s')$
- Discount factor $\gamma \in [0,1]$

$$s_0, a_0 \xrightarrow{P_{s_0 a_0}} s_1, a_1 \xrightarrow{P_{s_1 a_1}} s_2, a_2 \xrightarrow{P_{s_2 a_2}} \ldots..$$

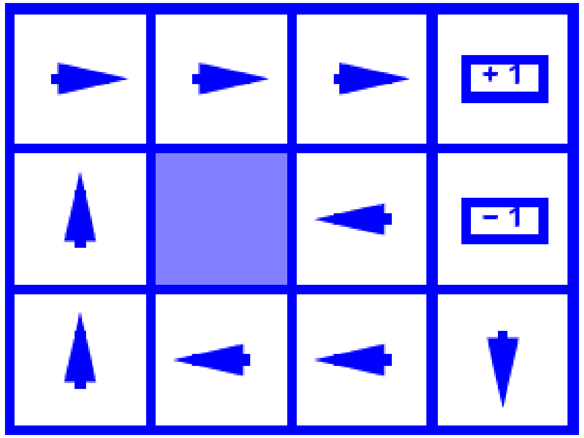
Policies

- Can a fixed sequence of states be a solution, like in classical search?
  - No, the environment is stochastic
- We should specify what the agent needs to do in every state
- Policy $\pi: S \rightarrow A$ recommends an action for every state
- Optimal policy π^* gives highest expected utility
- With π^* we can construct a simple reflex agent

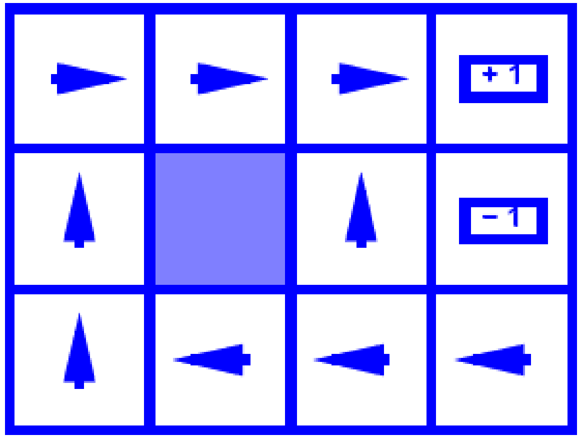


Picture taken from [1]

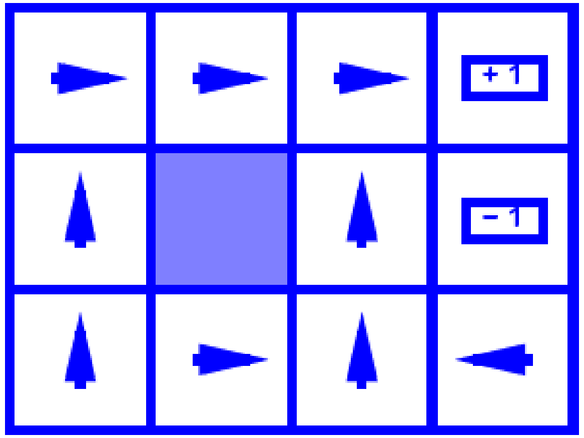
Example of optimal policies in Grid World



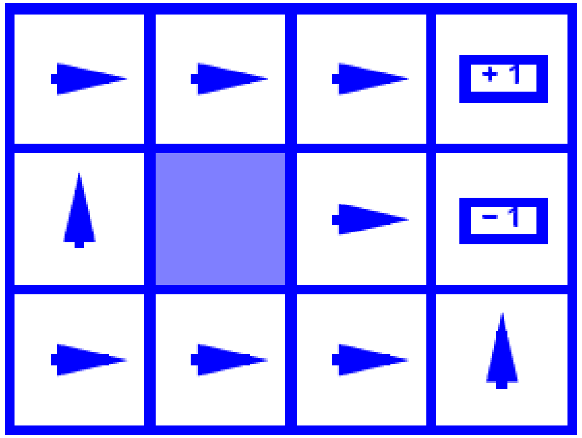
R(s) = -0.01



R(s) = -0.03



R(s) = -0.4



R(s) = -2.0

Pictures taken from [1]

Utilities over time

- Utilities evaluate sequences of states
- We will discuss infinite horizon
- With finite horizon, the optimal action in s could change over time
- If we assume stationary preferences:

$$[r, a_1, a_2, \ldots] \succ [r, b_1, b_2, \ldots] \quad \Leftrightarrow \quad [a_1, a_2, \ldots] \succ [b_1, b_2, \ldots]$$

- Then there are only 2 ways to define utilities
  - Additive utilities: $U([s_0, s_1, s_2, \ldots]) = R(s_0) + R(s_1) + R(s_2) + \cdots$
  - Discounted utilities: $U([s_0, s_1, s_2, \ldots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots$
- $\gamma \in [0,1]$ in infinite horizons:

$$U([s_0, s_1, s_2, \ldots]) = \sum_{t=0}^{\infty} \gamma^t R(s_t) \leq R_{max}/(1-\gamma)$$

Value of s using π

- Value of a state s using policy π:

$$V^\pi(s) = E\{R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots | \pi, s_0 = s\}$$

(The expected utility from s to the terminal state using π)

- **Bellman equations**:

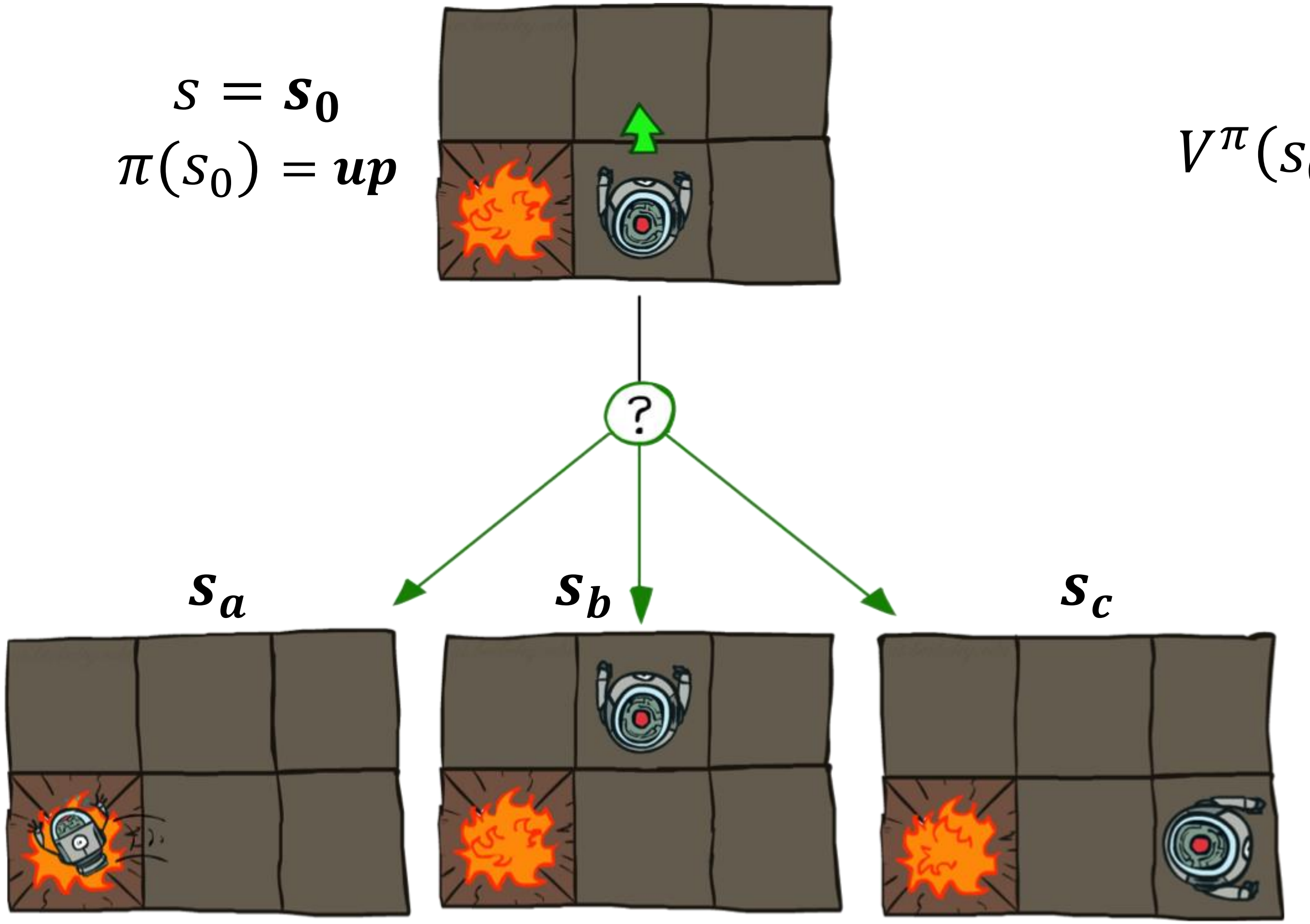
$$V^\pi(s) = E\{R(s_0) + \gamma(R(s_1) + \gamma R(s_2) + \cdots)|\pi, s_0 = s\}$$
$$= R(s) + \gamma E\{(R(s_1) + \gamma R(s_2) + \cdots)|\pi\}$$
$$= R(s) + \gamma E\{V^\pi(s_1)\}$$
$$(s, \pi(s) \xrightarrow{P_{s\pi(s)}} s')$$
$$= R(s) + \gamma \sum_{s'} P_{s\pi(s)}(s') V^\pi(s')$$

(N_s non-linear equations with N_s unknowns)

Example of a Bellman equation

Picture taken from [1]

$s = \boldsymbol{s_0}$
$\pi(s_0) = \boldsymbol{up}$



$\boldsymbol{s_a}$ $\boldsymbol{s_b}$ $\boldsymbol{s_c}$

$P_{s_0,up}(s_a) = 0.1$ $P_{s_0,up}(s_b) = 0.8$ $P_{s_0,up}(s_c) = 0.1$

$$
\begin{aligned}
V^{\pi}(s_0) &= R(s_0) + \gamma(P_{s_0,up}(s_a)V^{\pi}(s_a) \\
&\qquad + P_{s_0,up}(s_b)V^{\pi}(s_b) \\
&\qquad + P_{s_0,up}(s_c)V^{\pi}(s_c)) \\[1em]
&= R(s_0) + \gamma(0.1V^{\pi}(s_a) \\
&\qquad + 0.8V^{\pi}(s_b) \\
&\qquad + 0.1V^{\pi}(s_c))
\end{aligned}
$$

Optimal policy π^*

- Optimal value of a state s:

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

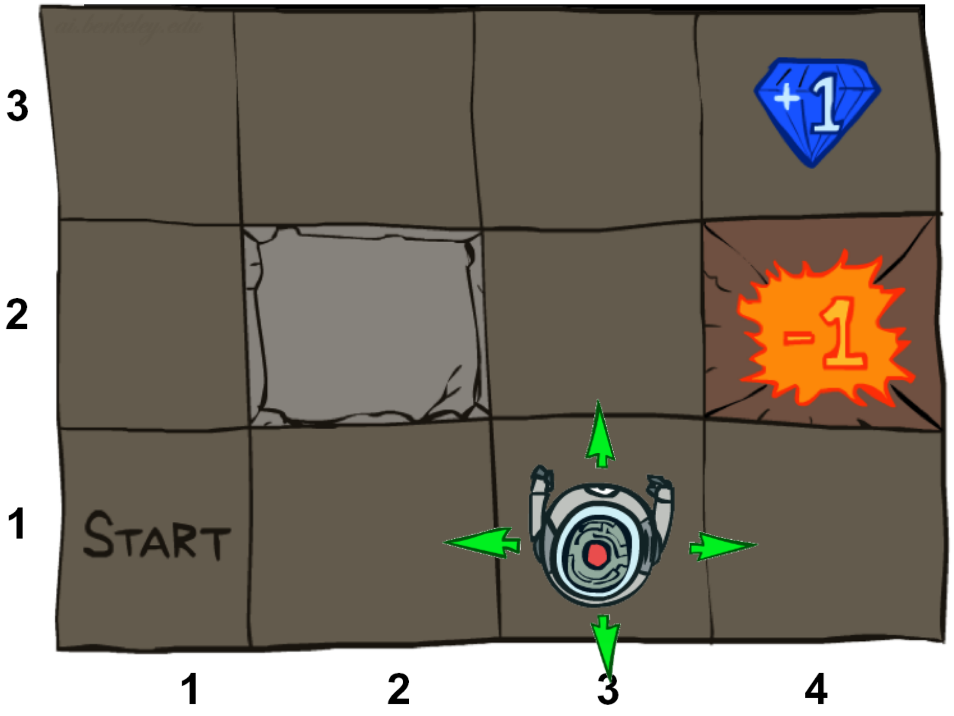
$$= R(s) + \max_{a} \gamma \sum_{s'} P_{sa}(s')V^*(s')$$

(N_s non-linear equations with N_s unknowns)

- Optimal policy:

$$\pi^*(s) = \operatorname*{argmax}_{a} \sum_{s'} P_{sa}(s')V^*(s')$$

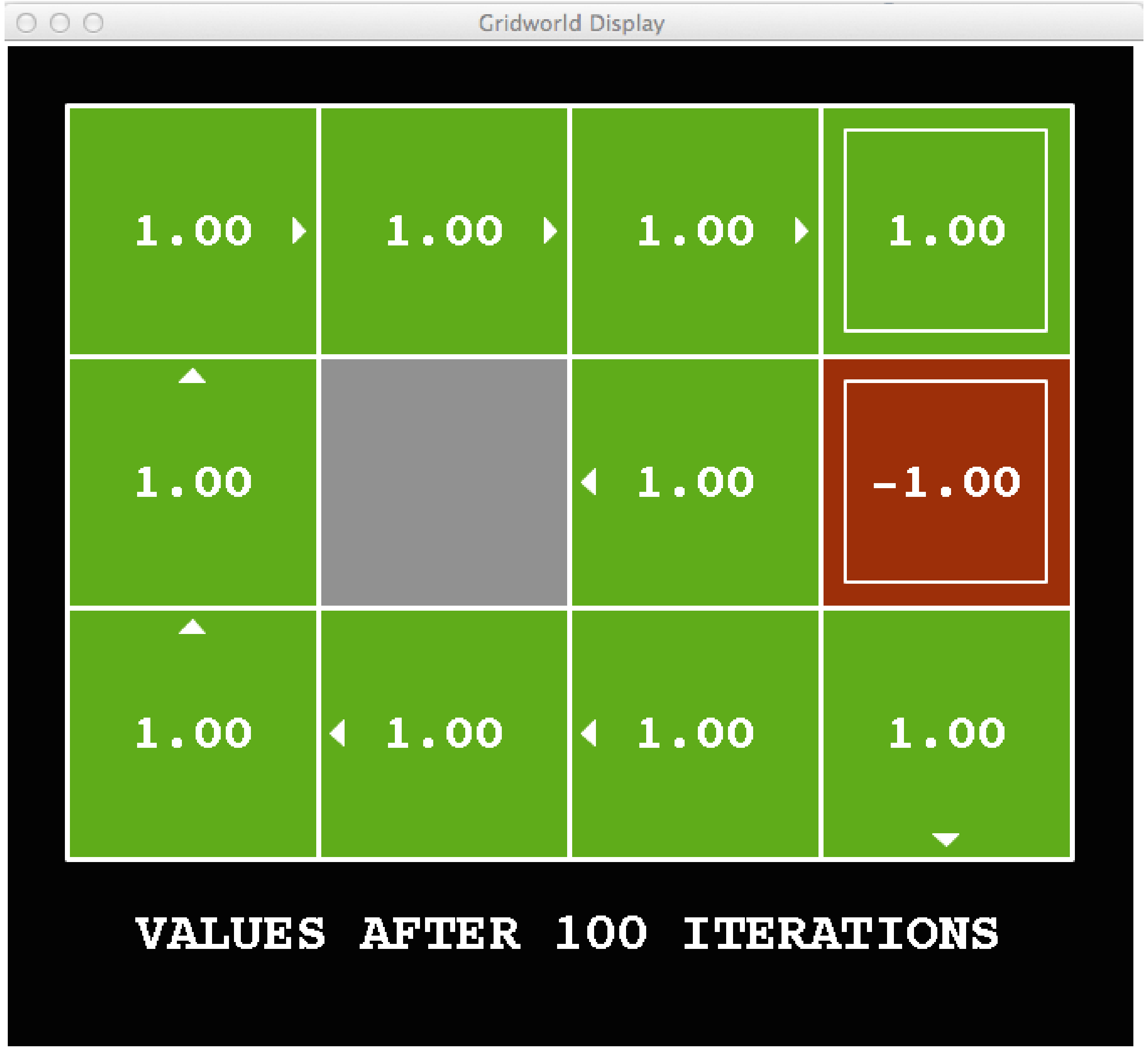
Example of a optimal value Bellman equation



Picture taken from [1]

$$V^*(s) = R(s) + \gamma \max_a [\textstyle\sum_{s'} P_{s,up}(s')V^*(s'),$$
$$\textstyle\sum_{s'} P_{s,down}(s')V^*(s'),$$
$$\textstyle\sum_{s'} P_{s,left}(s')V^*(s'),$$
$$\textstyle\sum_{s'} P_{s,right}(s')V^*(s')]$$

Example: Grid world optimal values



Picture taken from [1]

Noise = 0.2
Discount = 1
Living reward = 0

Example: Grid world optimal values

1 

Worth Now

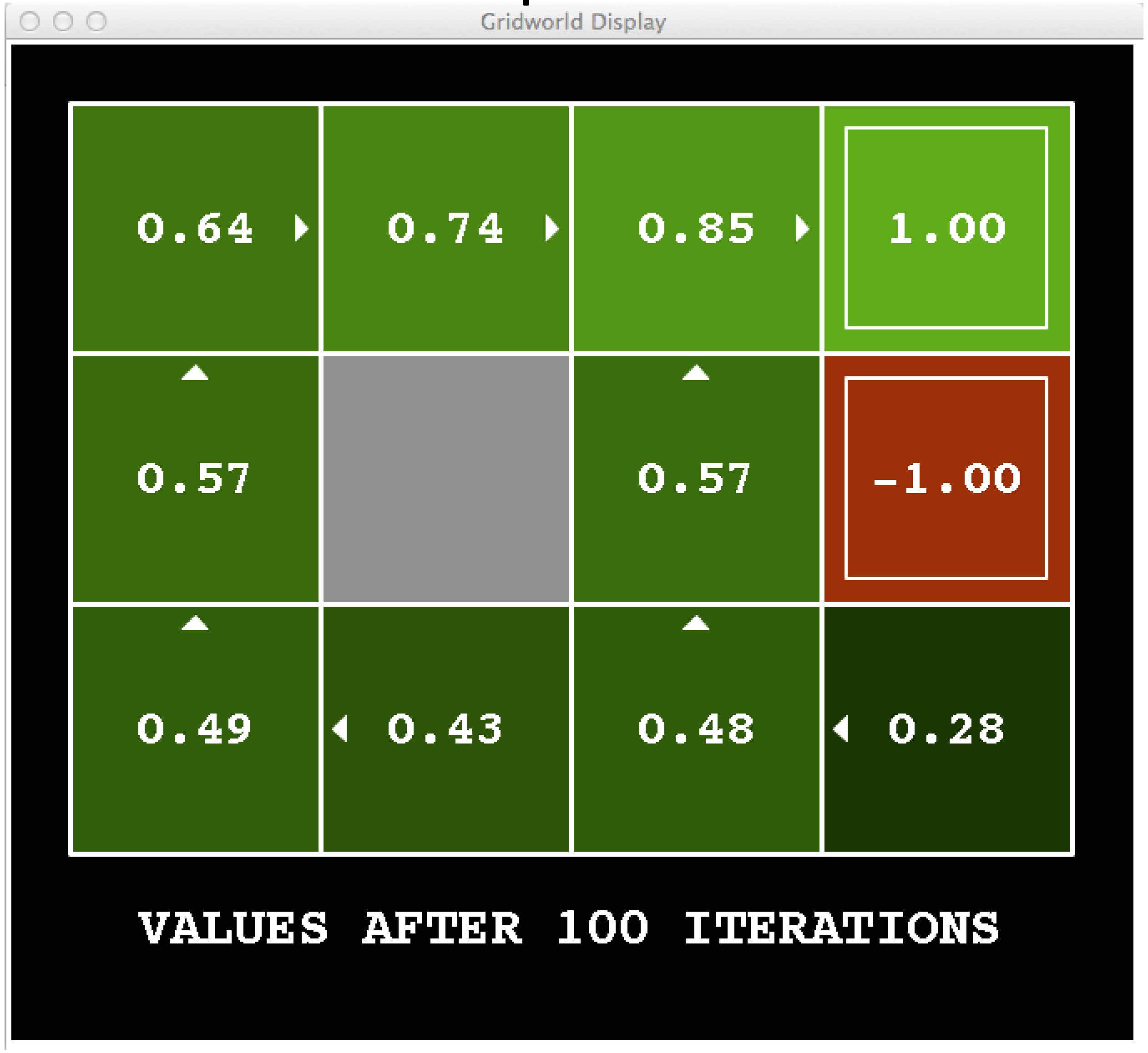
γ 

Worth Next Step

γ^2 

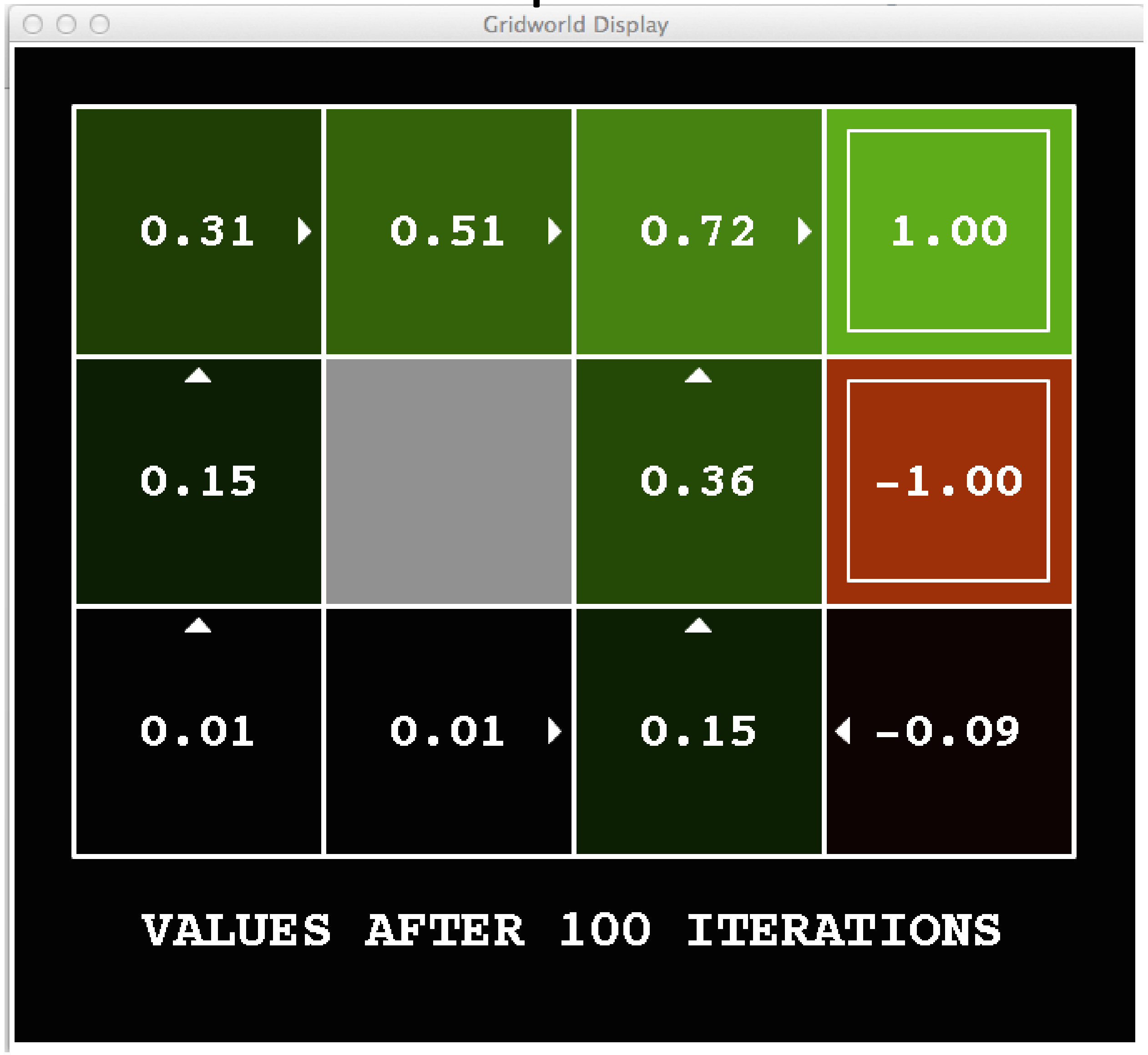
Worth In Two Steps

Pictures taken from [1]



Noise = 0.2
Discount = 0.9
Living reward = 0

Example: Grid world optimal values



Picture taken from [1]

Noise = 0.2
Discount = 0.9
Living reward = -0.1

Value iteration

- Task: For given $R(s)$, $P_{sa}(s')$ and γ compute $V^*(s)$, $\forall s$
- Assumption: finite number of states, and actions in each state
- Value iteration:
  - 0. Initialization: $V_0(s) = 0, \forall s$
  - 1. for $t = 1,2,3, \dots (untill\ convergence)$ do:

$$V_t(s) = R(s) + \gamma \max_{a'} \sum_{s'} P_{sa}(s') \, V_{t-1}(s')$$

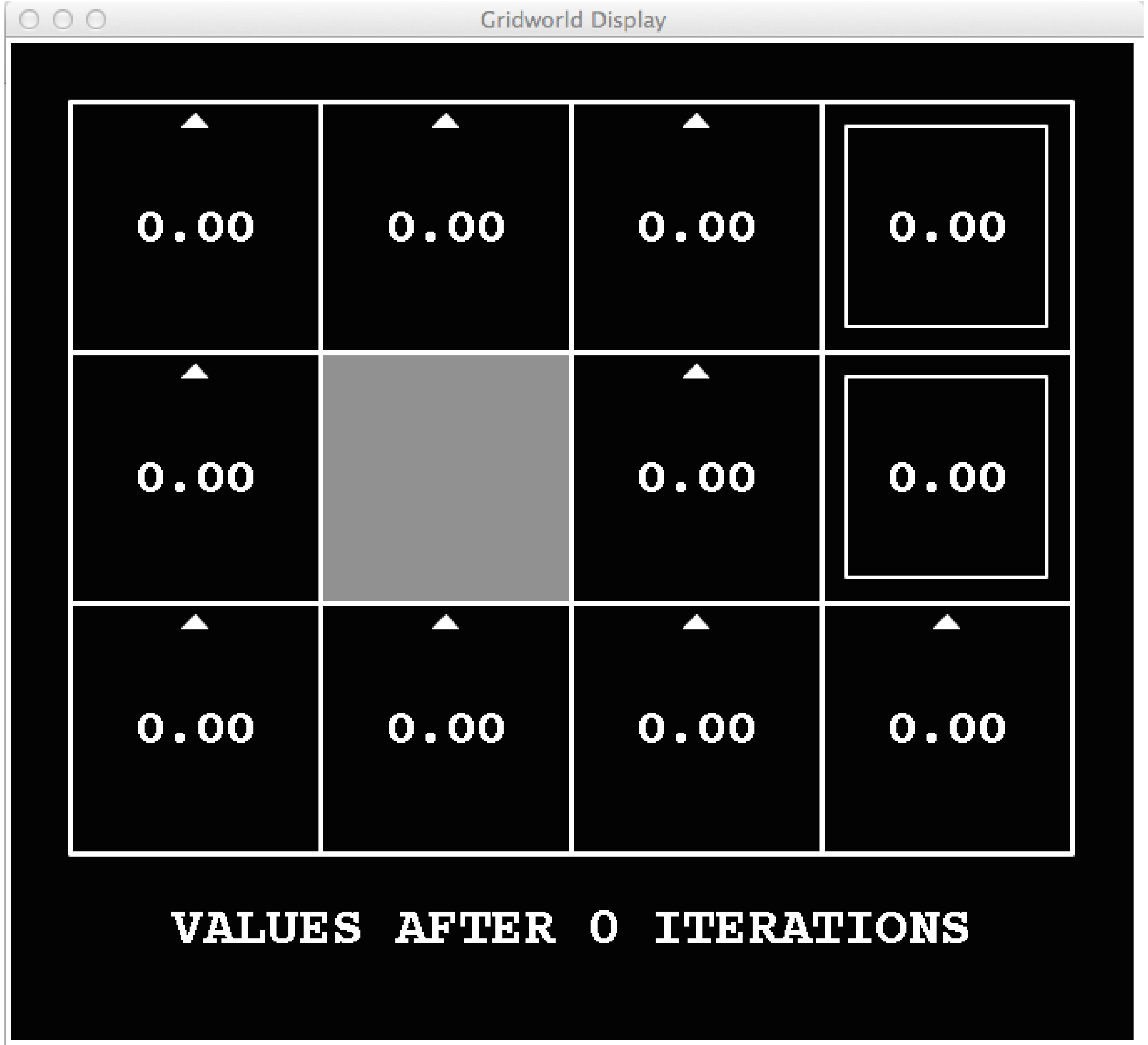
- Bellman equations have a unique solution
- Convergence: $V_t(s)$ doesn't differ much from $V_{t-1}(s)$

Value iteration

$$V_t(s) = R(s) + \gamma \max_{a'} \sum_{s'} P_{sa}(s')\, V_{t-1}(s')$$

- Complexity of each iteration: $O(|S|^2|A|)$:
  - We have to update the value of every state $s \in S$
  - For every state s: we have to take into account every action a
  - For each (s, a) pair we have to analyze all successor states s'
- Sinchronous updates: computed $V_t(s)$'s are used in the next iteration for the first time
- Asynchronous updates: Use computed $V_t(s)$'s for computing the rest of the $V_t(s)$'s

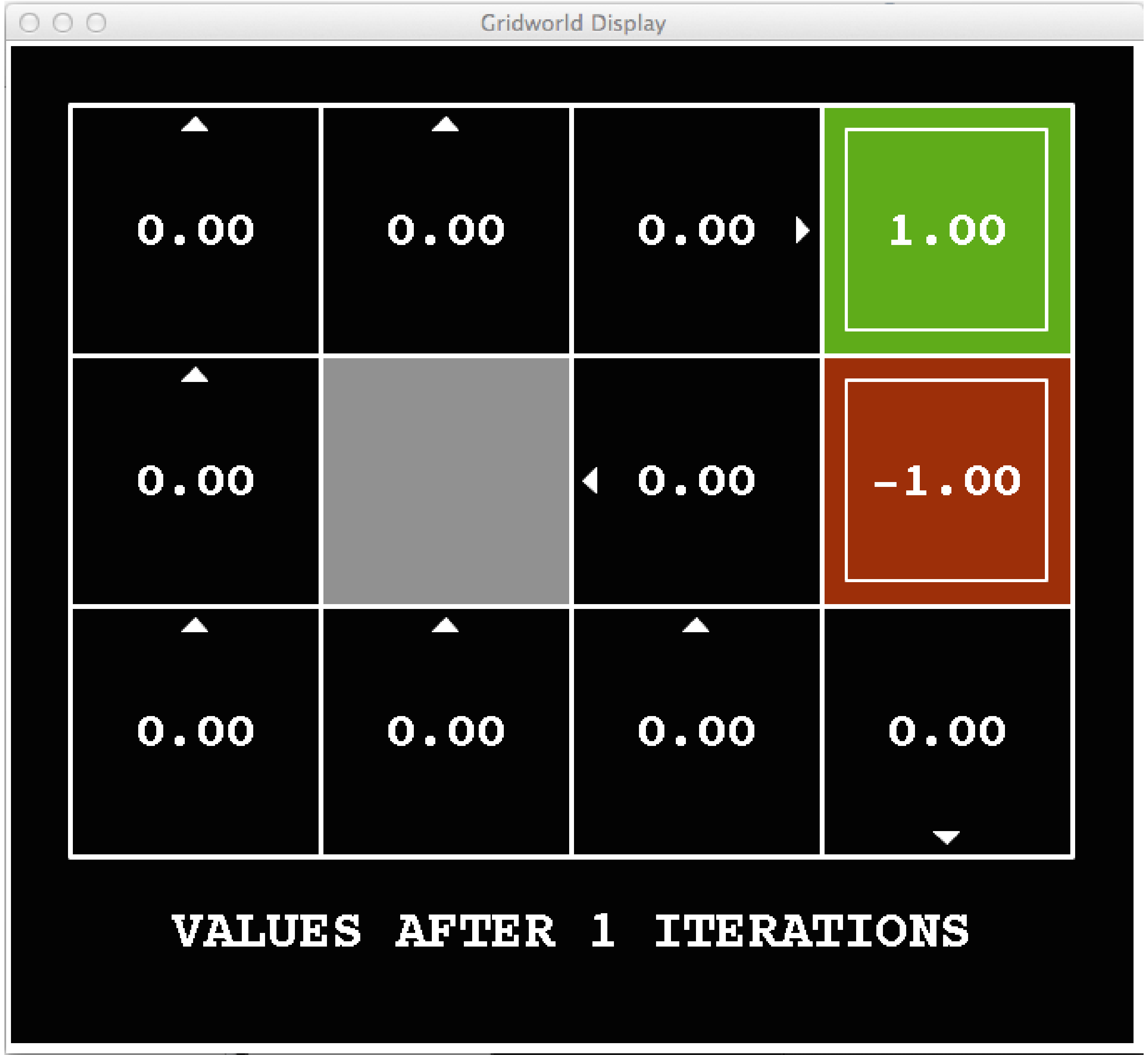
Example: Value iteration (discount + no living reward)



Picture taken from [1]

Noise = 0.2
Discount = 0.9
Living reward = 0

Example: Value iteration (discount + no living reward)



Picture taken from [1]

Noise = 0.2

Discount = 0.9

Living reward = 0

Example: Value iteration (discount + no living reward)

Noise = 0.2
Discount = 0.9
Living reward = 0

Picture taken from [1]

Example: Value iteration (discount + no living reward)

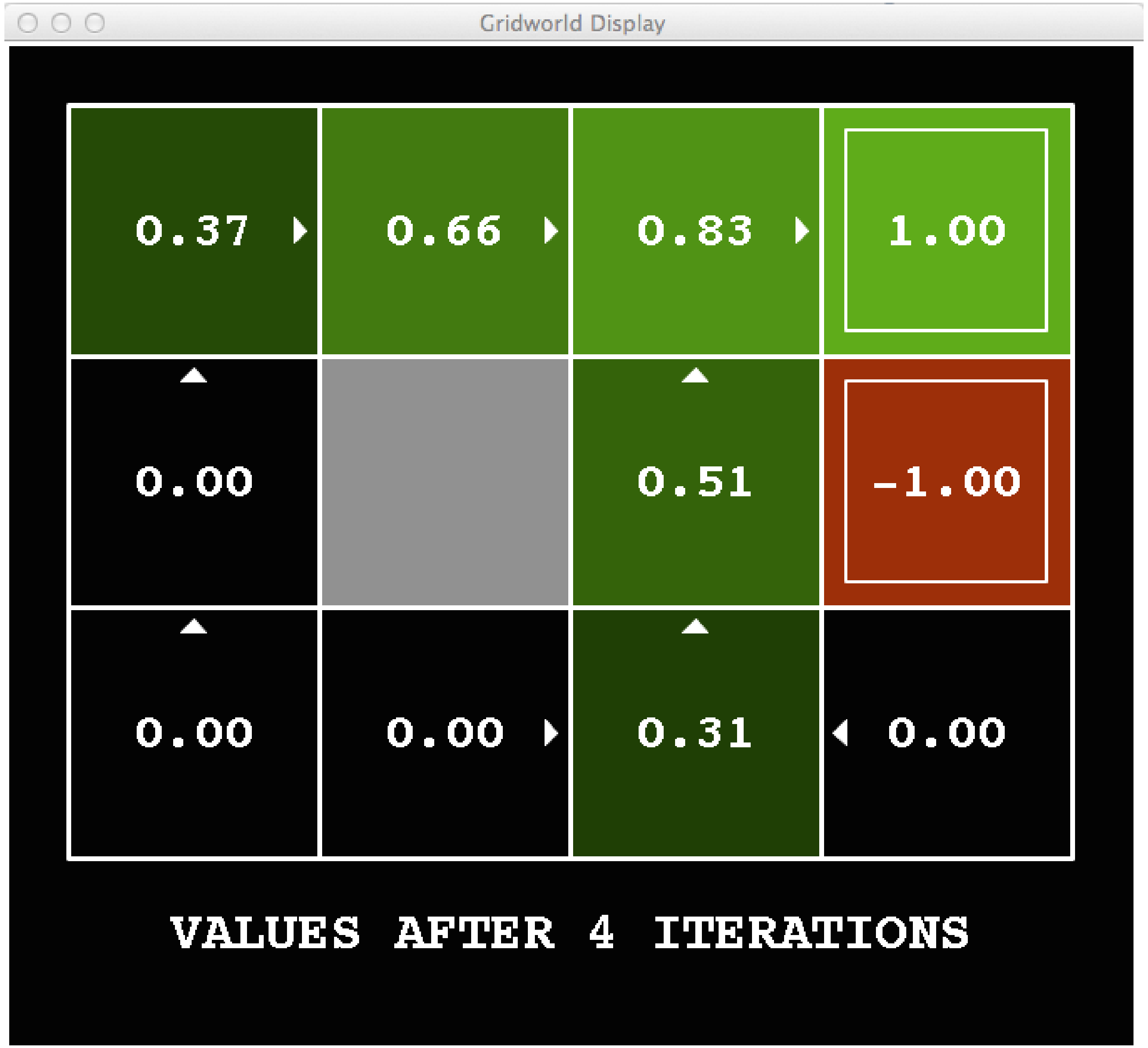
Picture taken from [1]

Noise = 0.2

Discount = 0.9

Living reward = 0

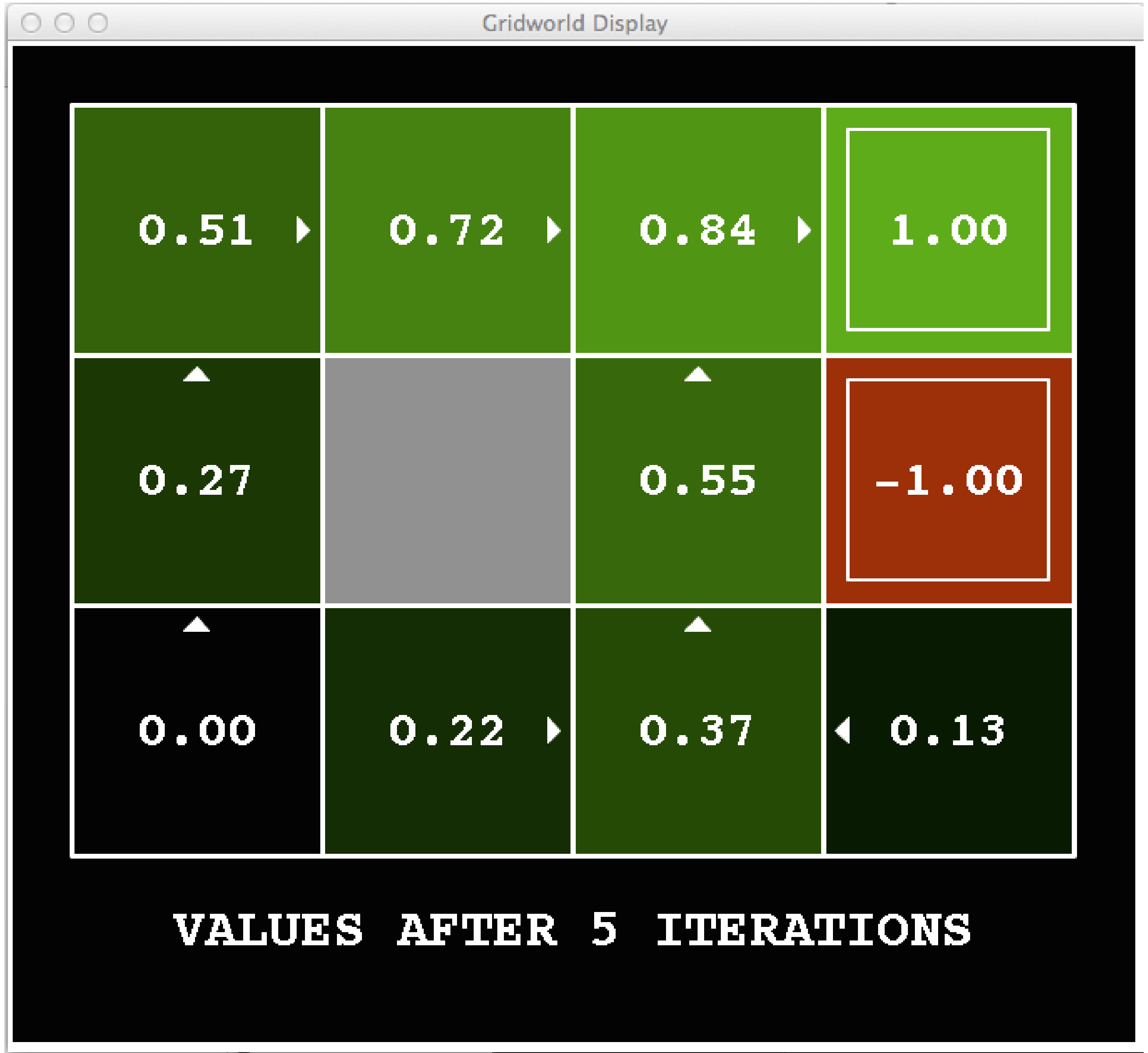
Example: Value iteration (discount + no living reward)



Picture taken from [1]

Noise = 0.2
Discount = 0.9
Living reward = 0

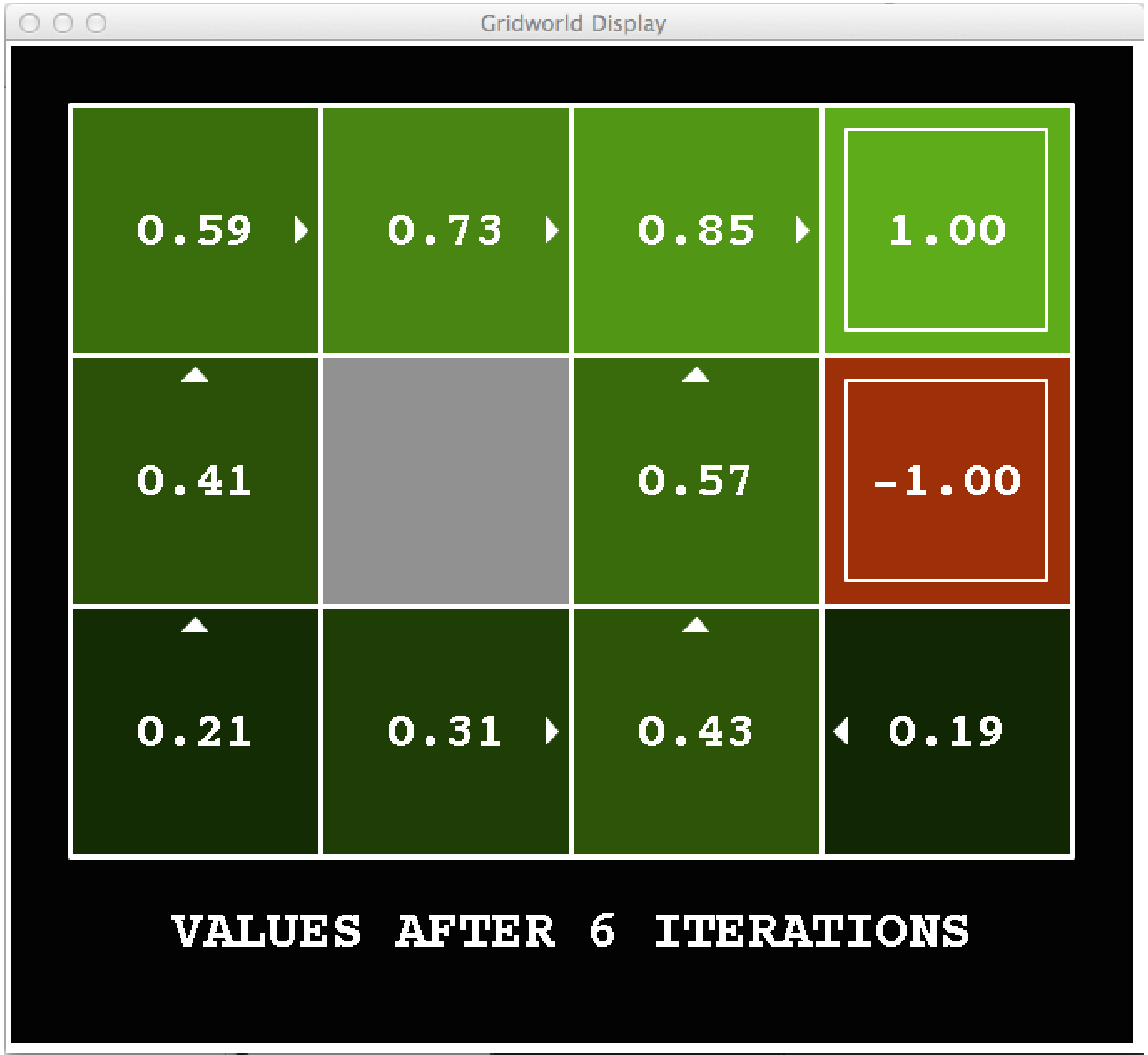
Example: Value iteration (discount + no living reward)



Picture taken from [1]

Noise = 0.2
Discount = 0.9
Living reward = 0

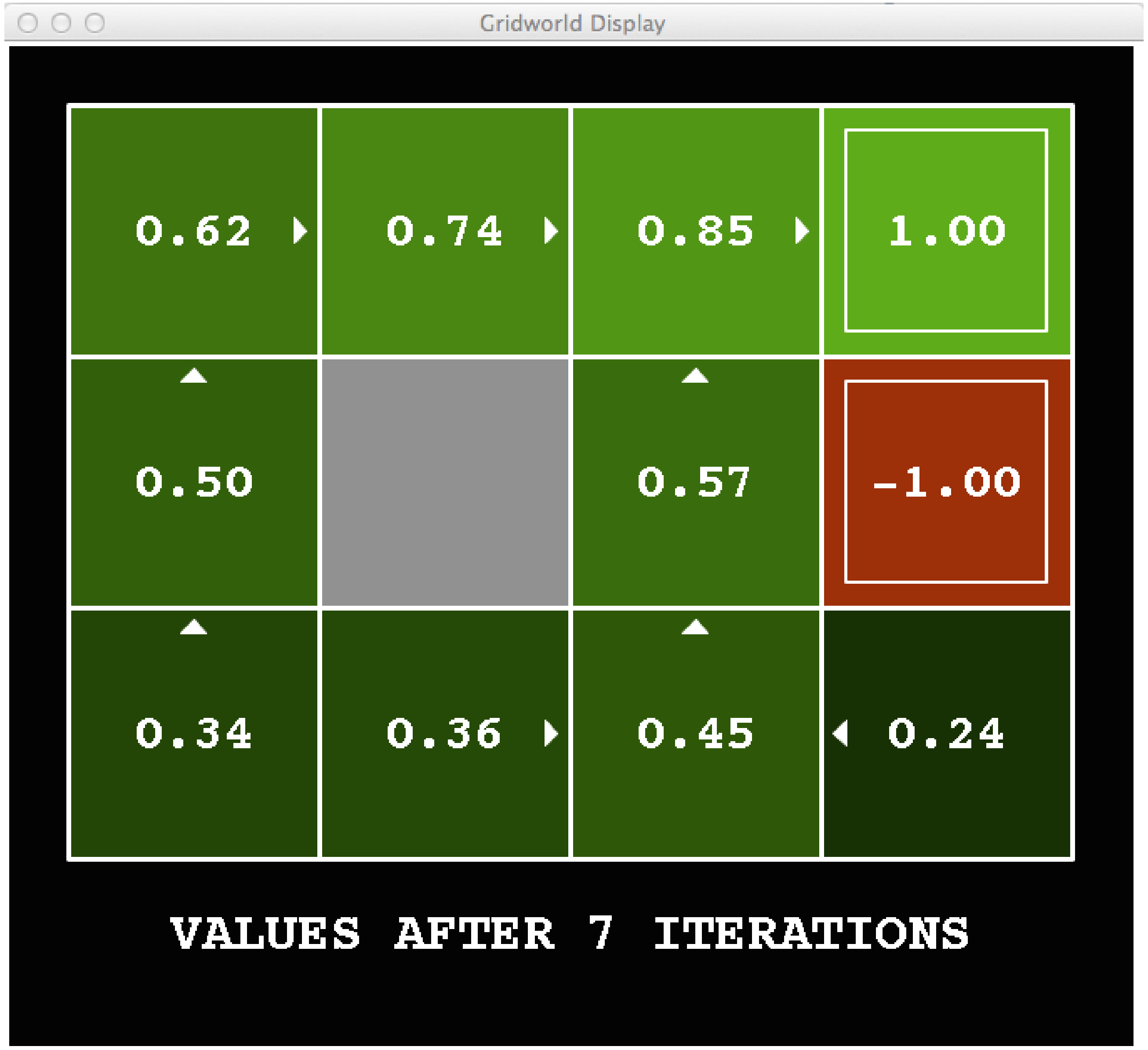
Example: Value iteration (discount + no living reward)



Picture taken from [1]

Noise = 0.2
Discount = 0.9
Living reward = 0

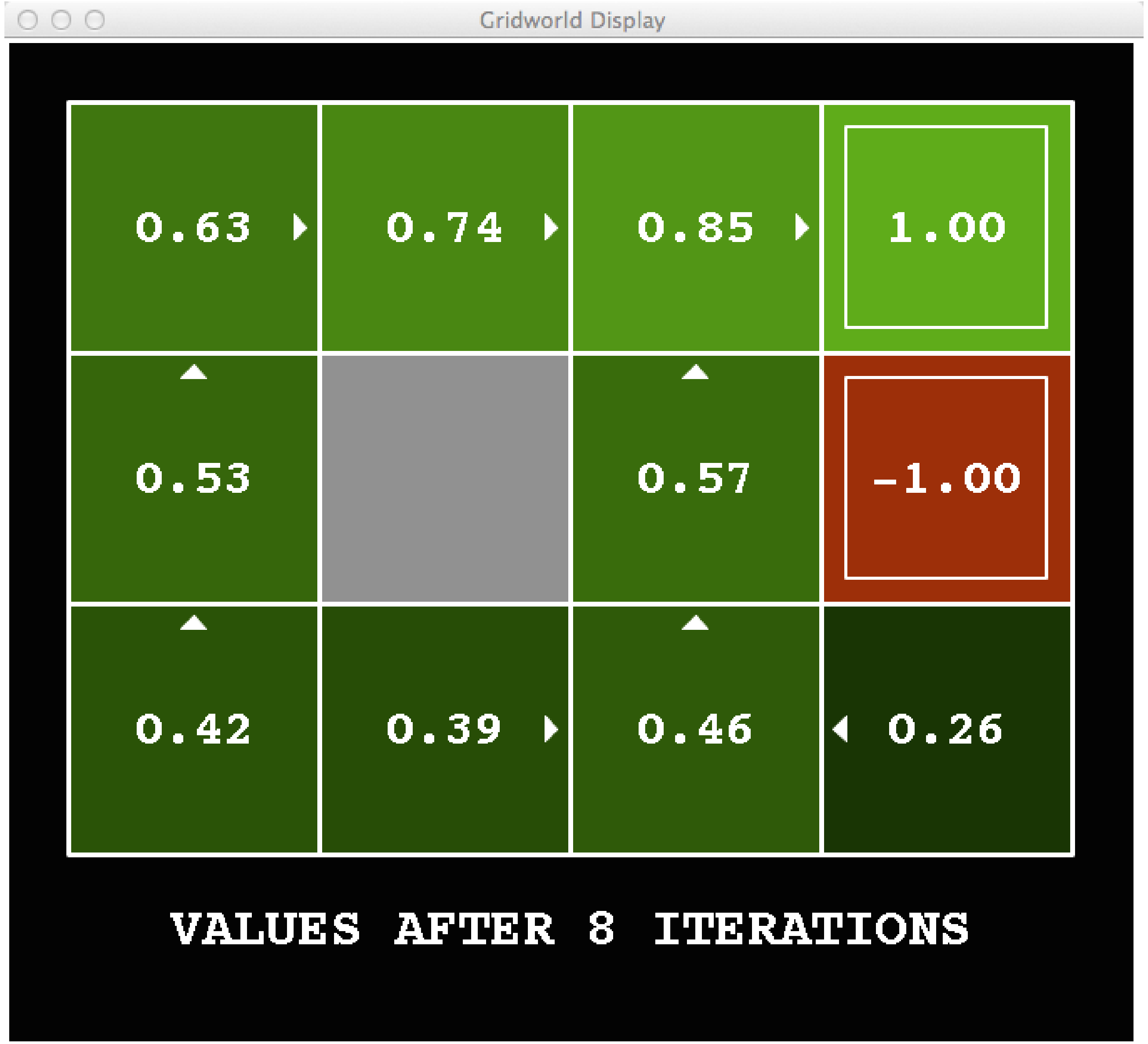
Example: Value iteration (discount + no living reward)



Picture taken from [1]

Noise = 0.2
Discount = 0.9
Living reward = 0

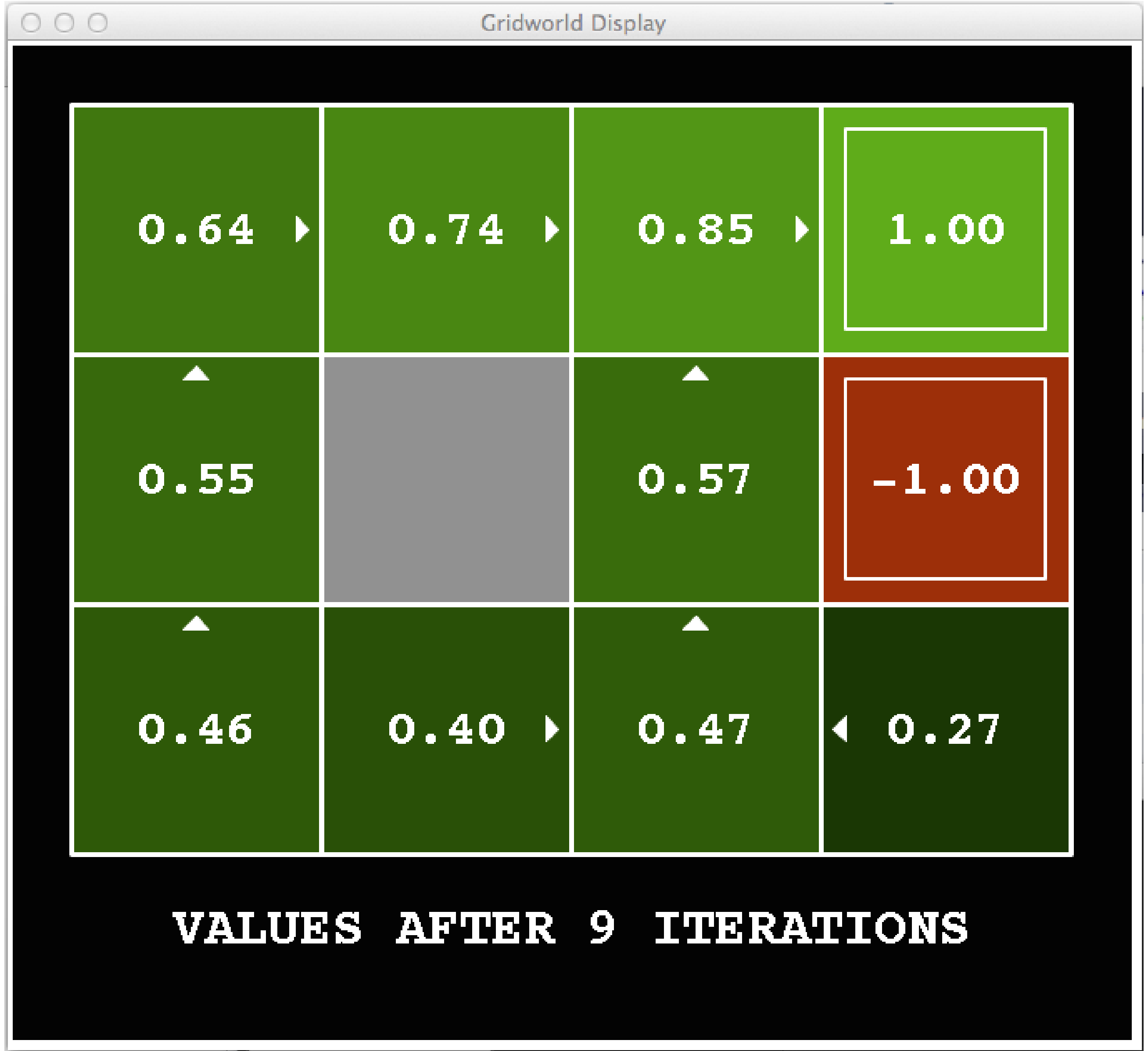
Example: Value iteration (discount + no living reward)



Picture taken from [1]

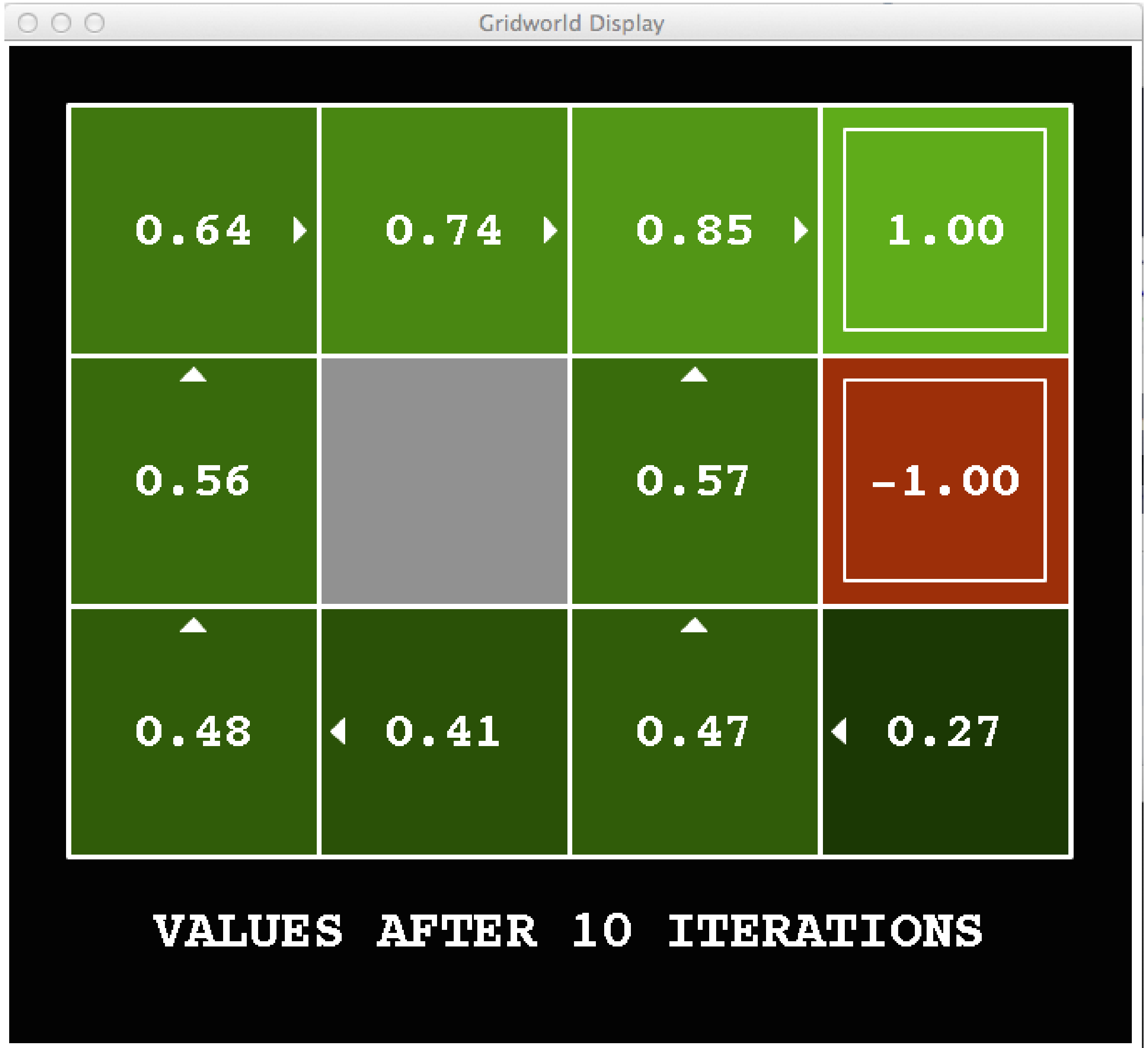
Noise = 0.2
Discount = 0.9
Living reward = 0

Example: Value iteration (discount + no living reward)



Noise = 0.2
Discount = 0.9
Living reward = 0

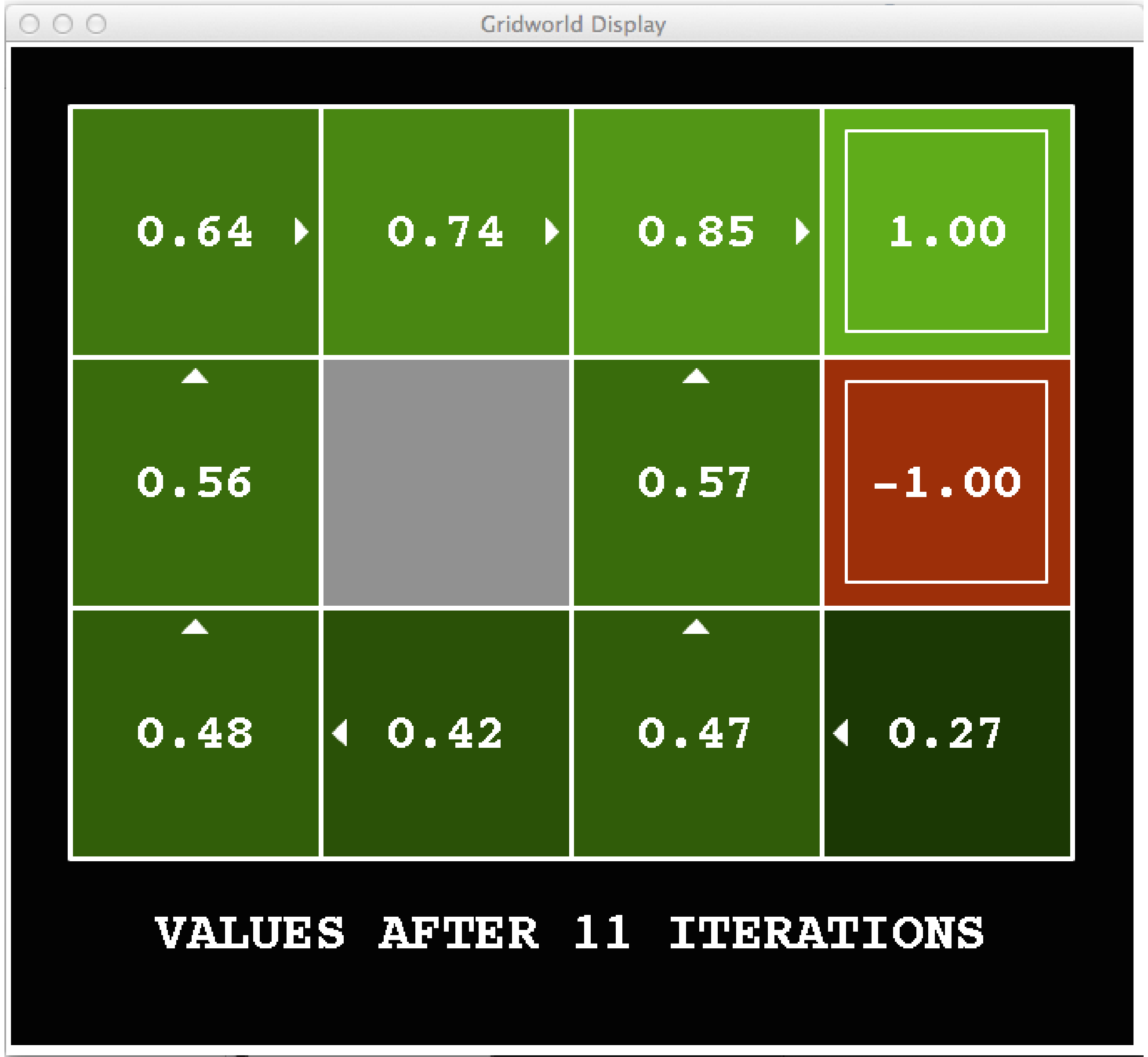
Example: Value iteration (discount + no living reward)



Picture taken from [1]

Noise = 0.2
Discount = 0.9
Living reward = 0

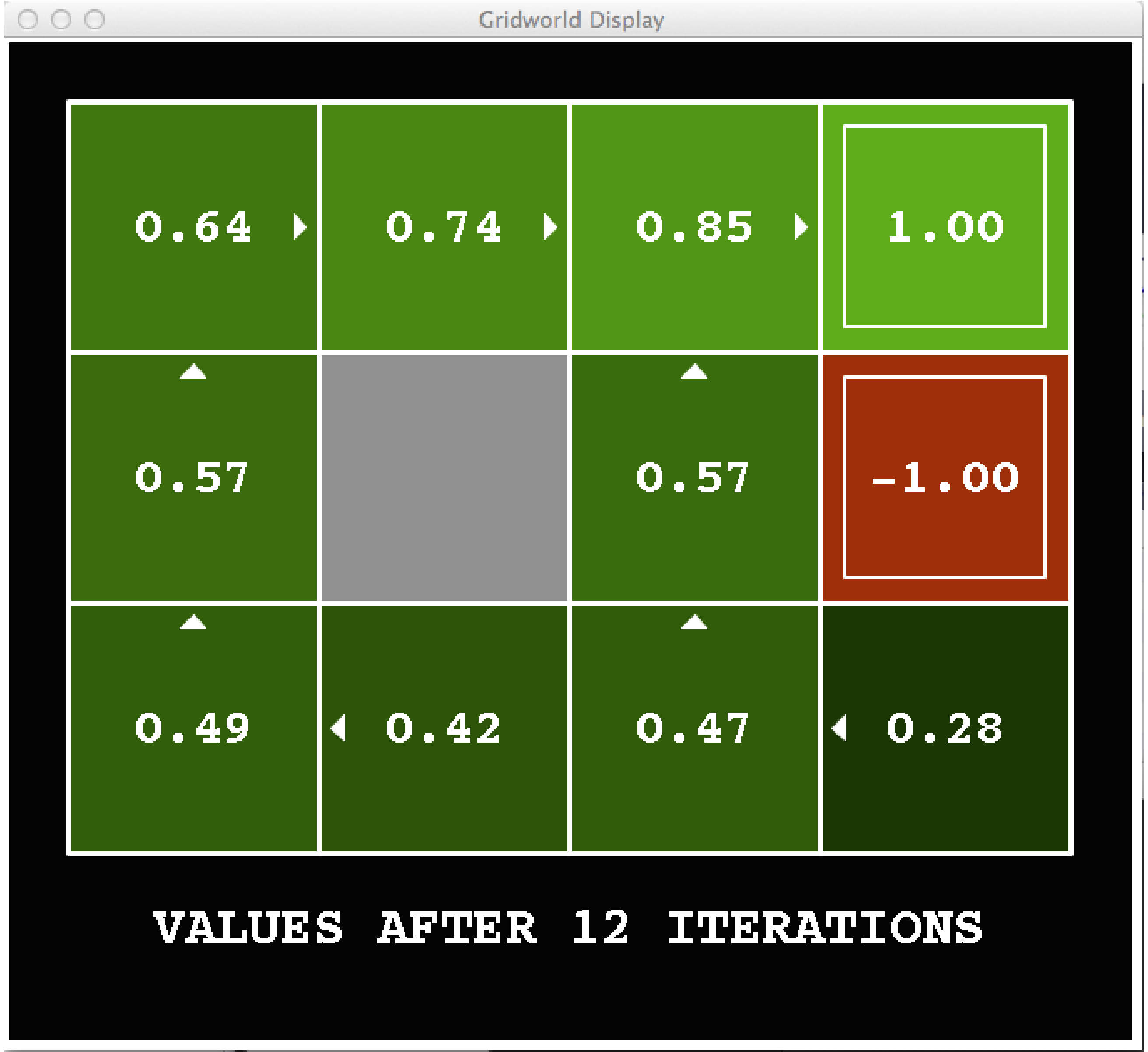
Example: Value iteration (discount + no living reward)



Picture taken from [1]

Noise = 0.2
Discount = 0.9
Living reward = 0

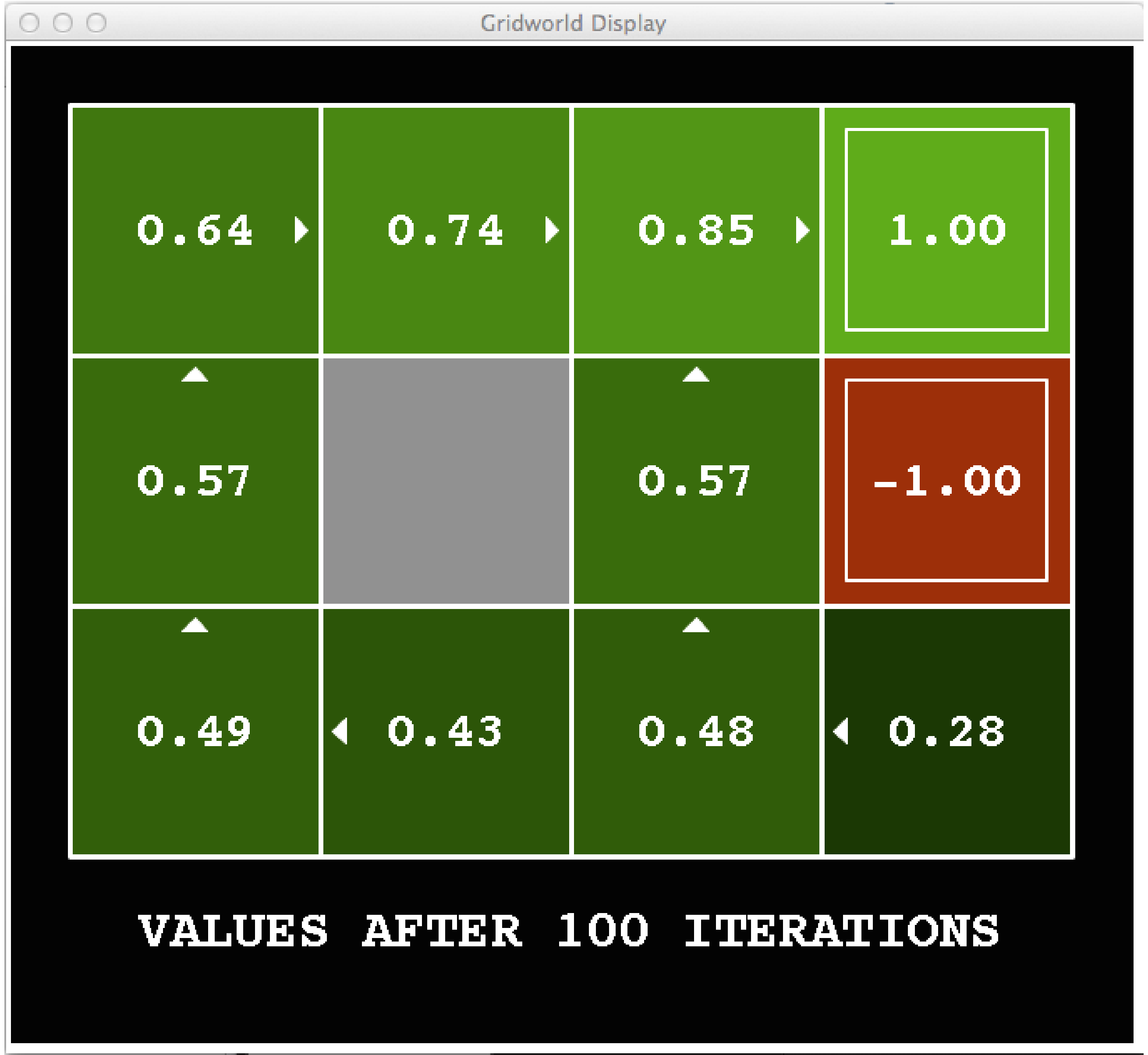
Example: Value iteration (discount + no living reward)



Picture taken from [1]

Noise = 0.2
Discount = 0.9
Living reward = 0

Example: Value iteration (discount + no living reward)



Picture taken from [1]

Noise = 0.2
Discount = 0.9
Living reward = 0

Value iteration: Convergence

- Bellman equations have a unique solution
- Interpretation: $V_t(s)$ is the optimal value if we have t moves left:
  - $V_0(s) = 0$: we cant make moves anymore
  - $V_1(s) = R(s)$: we can only collect the current reward
  - $V_2(s) = R(s) + \gamma \max_{a'} \sum_{s'} P_{sa}(s')\, R(s')$
  - …..
- Convergence:
  - Bellman update is a contraction on the space of value vectors:

$$\max_s |V_{i+1}(s) - V_{i+1}(s)'| \leq \gamma \max_s |V_i(s) - V_i(s)'|$$

$$\Downarrow$$

$$\max_s |V_{i+1}(s) - V^*(s)| \leq \gamma \max_s |V_i(s) - V^*(s)|$$

Value iterations flaws

- Its slow: $O(|S|^2|A|)$ for every iteration
- The $\max_a(\,.\,)$ rarely changes its choice of a
  - Big computational expense
- The extracted policy usually converges long before the values do

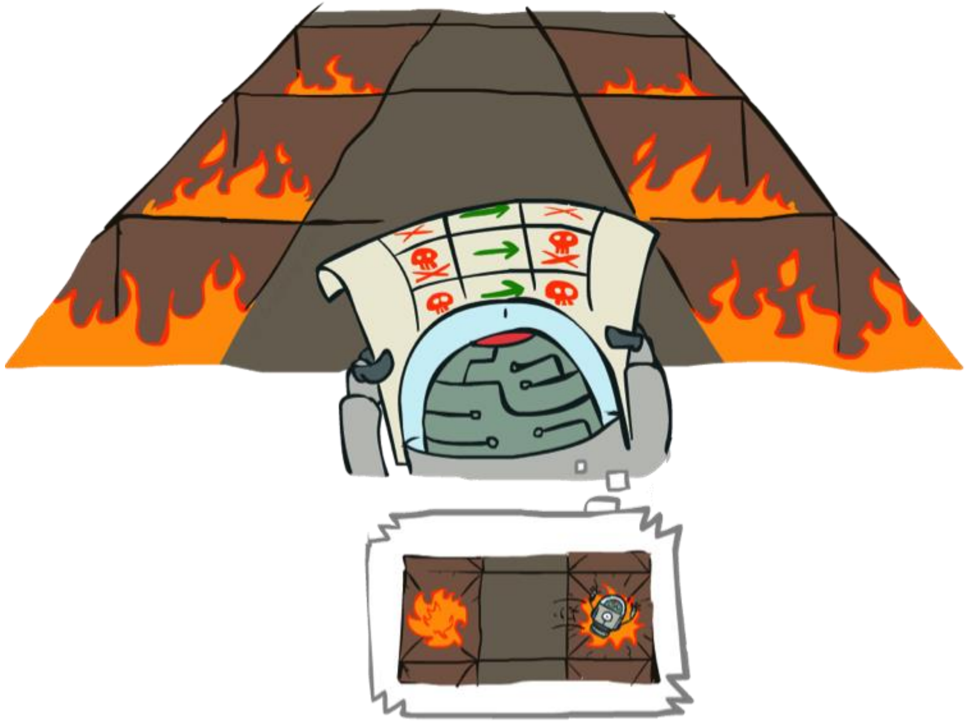
- We can also compute $V^\pi(s)$ in a similar way:

$$V_t(s) = R(s) + \gamma \sum_{s'} P_{s\pi(s)}(s')\, V_{t-1}(s')$$

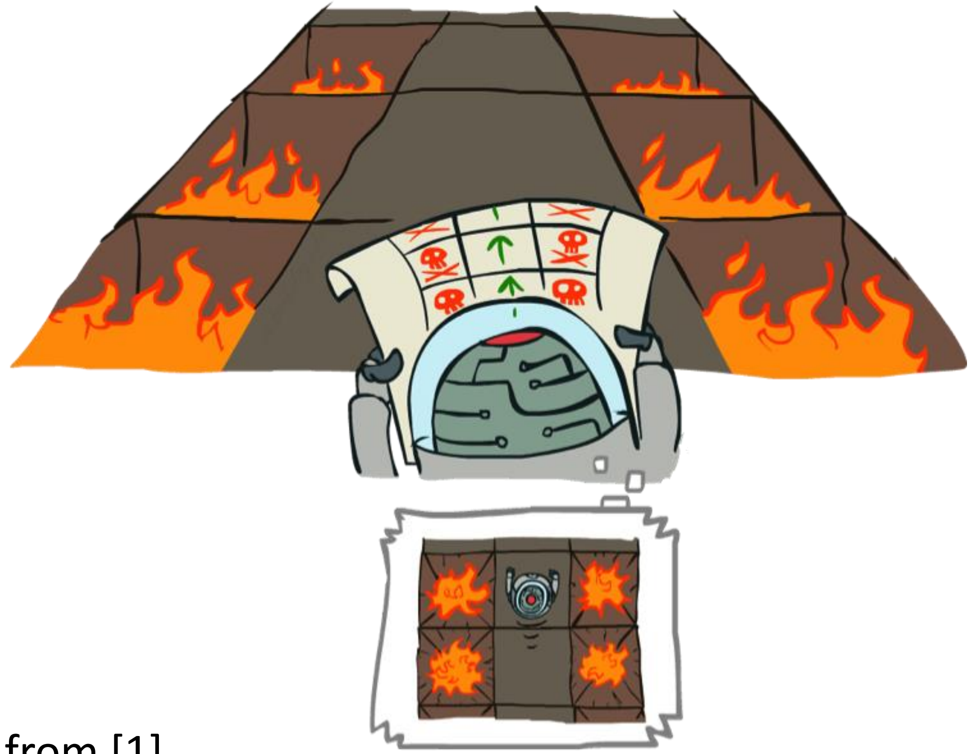
(suitable for large $|S|$)

Example: Policy evaluation

Always Go Right



Always Go Forward

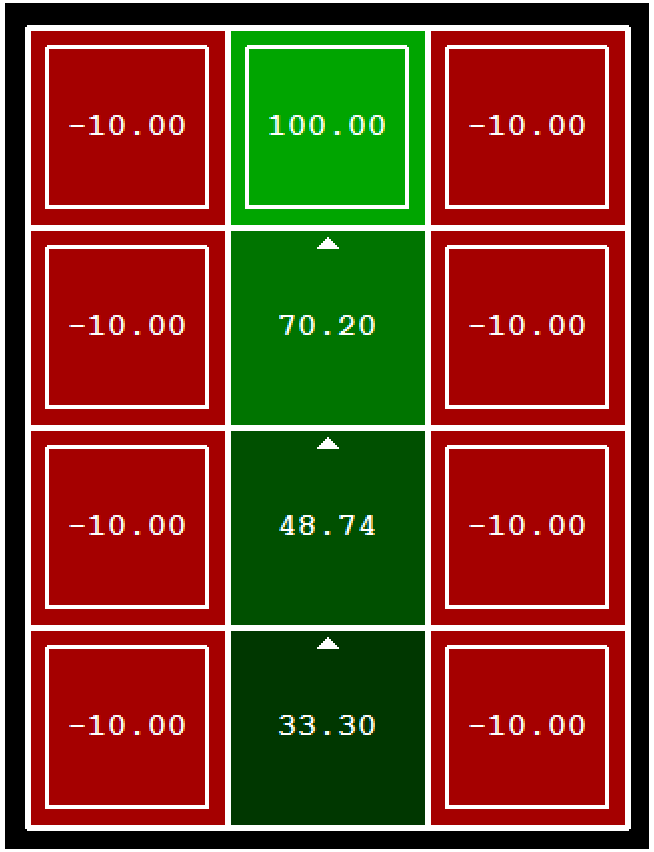


Pictures taken from [1]

Example: Policy evaluation

Always Go Right

Always Go Forward



Pictures taken from [1]

Policy iteration

- Initialization: Pick a random π_0
- for $t = 1,2,3, \dots (untill\ convergence)$ do:
  - 1. Policy evaluation:

    Calculate $V^{\pi_t}(s)$ (N_s x N_s linear system or Bellman updates)
  - 2. Policy update:

$$\pi_{t+1}(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} P_{sa}(s') V^{\pi_t}(s')$$

- Policy iteration with Bellman updates is often much more efficient than Value iteration or standard Policy iteration
- Convergence: $V^{\pi_t}(s)$ converged or if $\pi_{t+1}(s) = \pi_t\ (s)$

References

[1] UC Berkeley: CS188 Intro to AI, lecture slides, http://ai.berkeley.edu/lecture_slides.html - Lecture 8: MDP I and Lecture 9: MDP II (last visited: 11.03.2018)

[2] Stuart J. Russell and Peter Norvig, Artificial Intelligence: A Modern Approach 3rd edition, Prentice Hall, 2009.

[3] Faculty of Electrical Engineering, University of Belgrade: Statistička klasifikacija signala, lecture materials, http://automatika.etf.bg.ac.rs/images/FAJLOVI_srpski/predmeti/master_studije/SKS/09%20Ucenje%20podsticanjem.pdf (last visited: 11.03.2018)

Questions?

Thanks for the attention! ☺