LECTURE 19: MDPS

CS 6140: Machine Learning

Chris Amato and Andrea Baisero
(some slides from Mykel Kochenderfer and Rob Platt)

TODAY

- Markov decision processes (MDPs)
  - Formulation
  - Solution

QUANTIFYING UNCERTAINTY WITH PROBABILITIES

- Generally we may use probabilities due to
- Ignorance
  - E.g., we don't know our opponent
- Laziness
  - It is too difficult or takes too much effort to model the event in detail
- The event is inherently random

MAKING DECISIONS UNDER UNCERTAINTY

- Going to the airport: A_X means leaving X minutes before the flight
- Suppose I believe the following:
- $P(A_{25}$ gets me there on time|...) = 0.04
- $P(A_{90}$ gets me there on time|...) = 0.70
- $P(A_{120}$ gets me there on time|...) = 0.95
- $P(A_{1440}$ gets me there on time|...) = 0.9999
- Which action to choose?

MAKING DECISIONS UNDER UNCERTAINTY

- Suppose I believe the following:
- $P(A_{25}$ gets me there on time|...) = 0.04
- $P(A_{90}$ gets me there on time|...) = 0.70
- $P(A_{120}$ gets me there on time|...) = 0.95
- $P(A_{1440}$ gets me there on time|...) = 0.9999
- Which action to choose?
- Depends on my preferences for missing flight vs. airport cuisine, etc.
- Utility theory is used to represent and infer preferences
- Decision theory = utility theory + probability theory

MAKING DECISIONS UNDER UNCERTAINTY

- Rational decision making requires reasoning about one's *uncertainty* and *objectives*

- Previous section focused on uncertainty

- This section will discuss how to make rational decisions based on a *probabilistic model* and *utility function*

- Last week, we focused on single step decisions, now we will consider sequential decision problems

MAXIMIZING EXPECTED UTILITY (MEU)

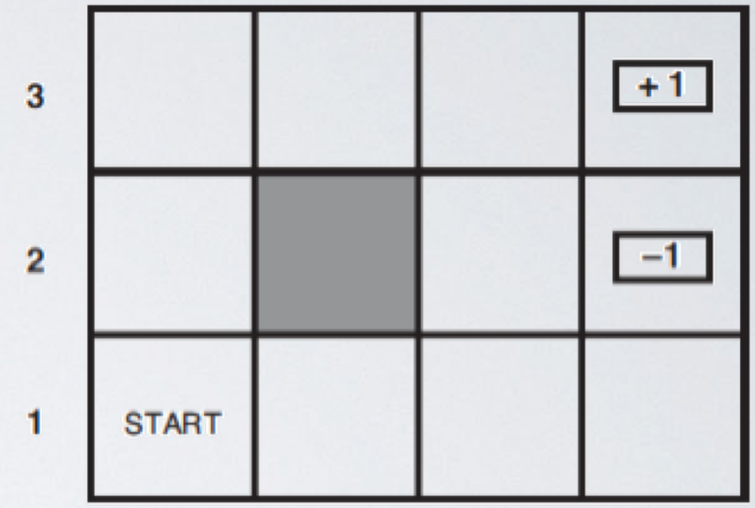
- Theorem (Ramsey, 1931; von Neumann and Morgenstern, 1944): Given preferences satisfying the constraints there exists a real-valued function U such that
  - $U(A) \geq U(B) \Leftrightarrow A \succsim B$
  - $U(A) > U(B) \Leftrightarrow A \succ B$
  - $U(A) = U(B) \Leftrightarrow A \sim B$
  - $U([p_1, s_1; \ldots; p_n, s_n]) = \sum_i p_i U(s_i)$
- *MEU principle:* Choose the action that maximizes expected utility

DIFFERENT APPROACH IN SEQUENTIAL DECISION MAKING

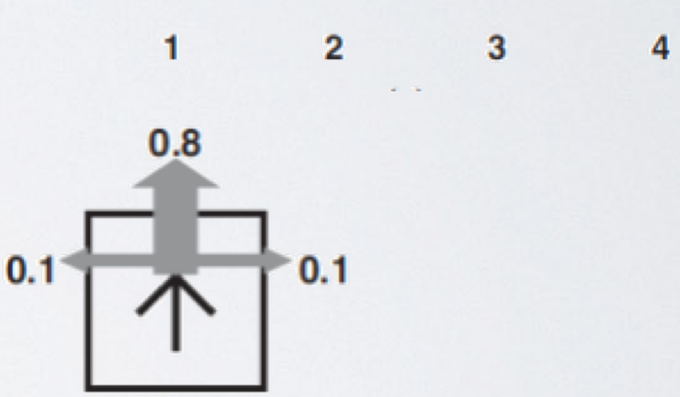
- Now we consider agents who use a "Policy"
  - A strategy that determines what action to take in any state
  - Assuming unreliable action outcomes

EXAMPLE: GRID WORLD

- A maze-like problem
  - The agent lives in a grid
  - Walls block the agent's path



- Non-deterministic movement: actions are not reliable
  - 80% of the time, the action has intended effect (if there is no wall there)
  - 10% of the time, takes the agent to the left of intended direction and 10% to the right
  - If there is a wall in the direction the agent would have been taken, the agent stays put

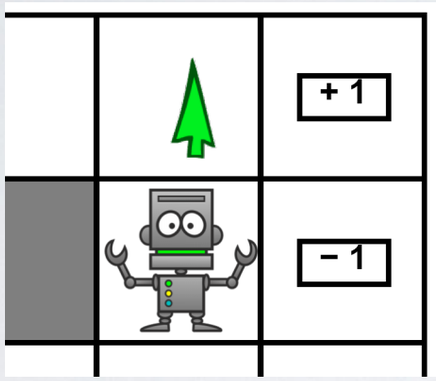


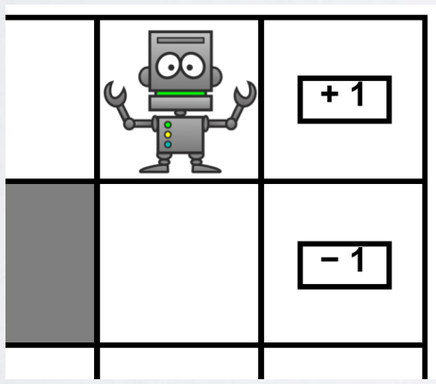
- This agent receives rewards each time step
  - Small "living" reward each step (can be negative)
  - Big rewards come at the end (good +1 or bad -1)

- Goal: maximize sum of rewards

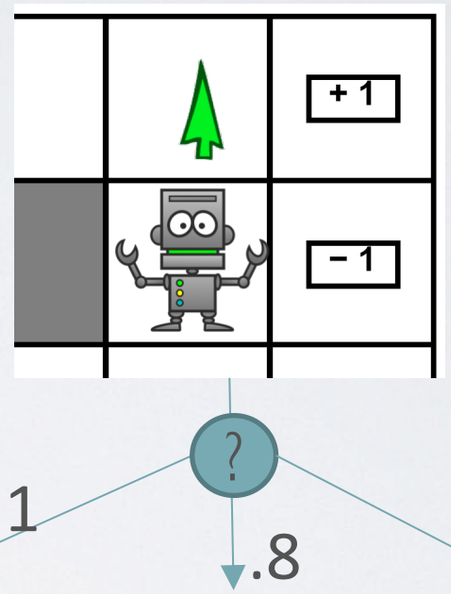
IMPACT OF NON-DETERMINISM

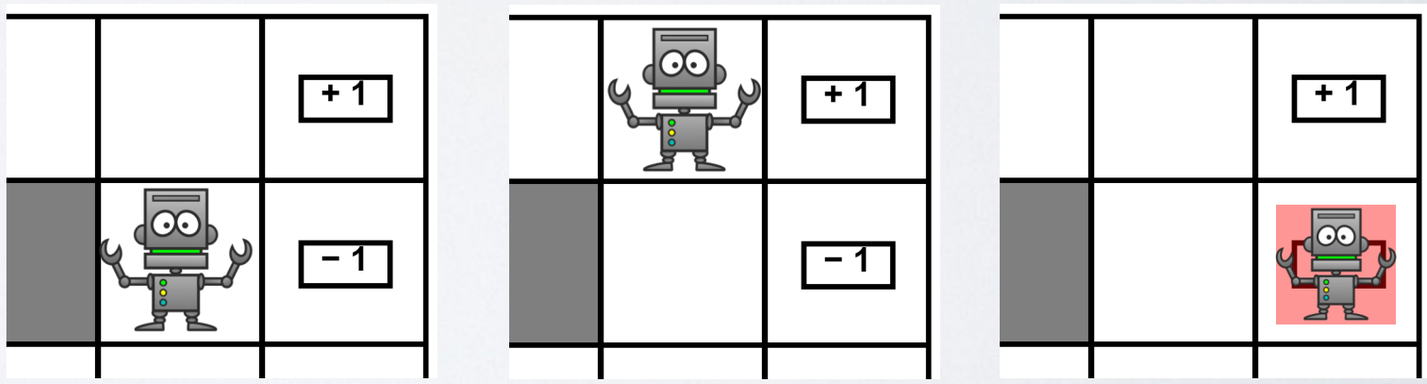
Deterministic World



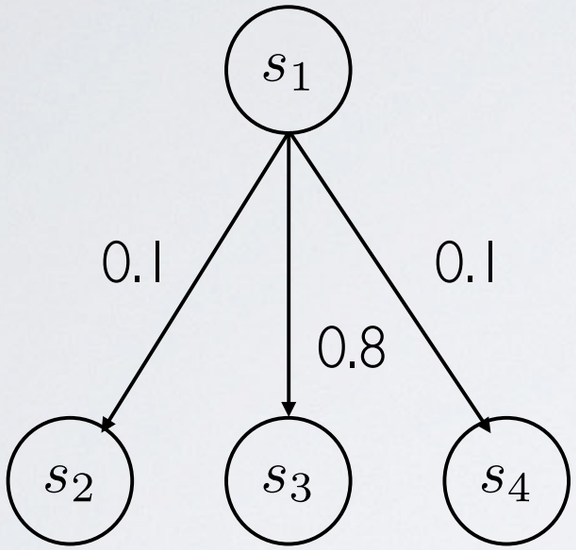


Non-Deterministic

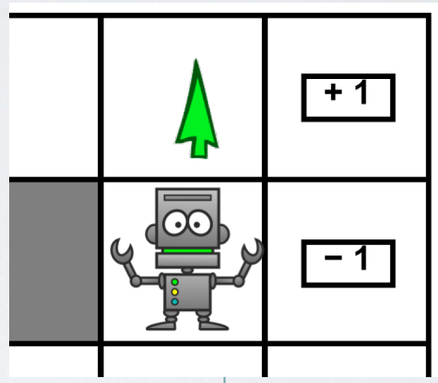


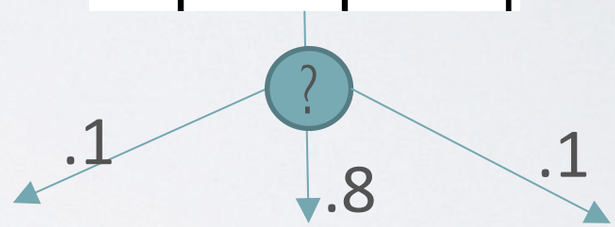


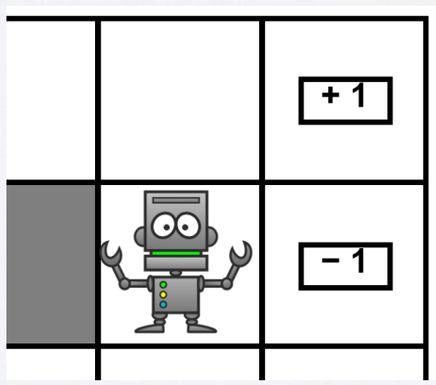
TRANSITION FUNCTION

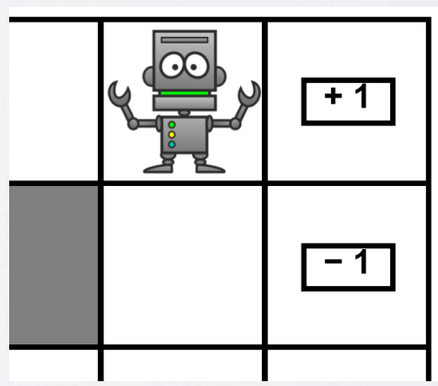


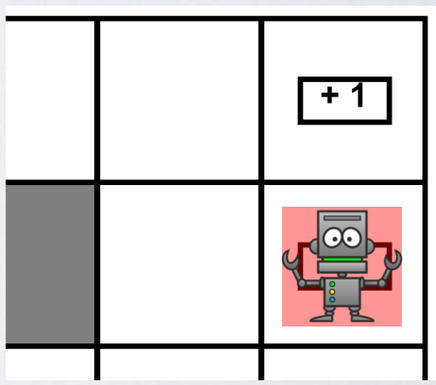
Probabilistic (or stochastic) transitions









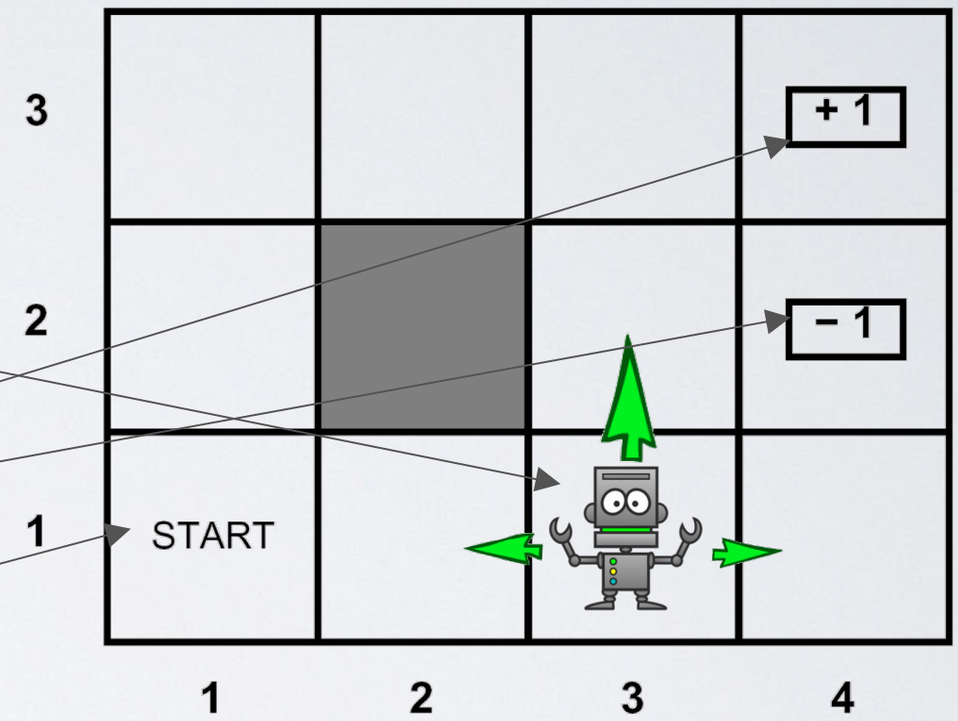


Transition probabilities:

s'	$P(s' \mid s_1, a)$
s_2	0.1
s_3	0.8
s_4	0.1

MARKOV DECISION PROCESSES

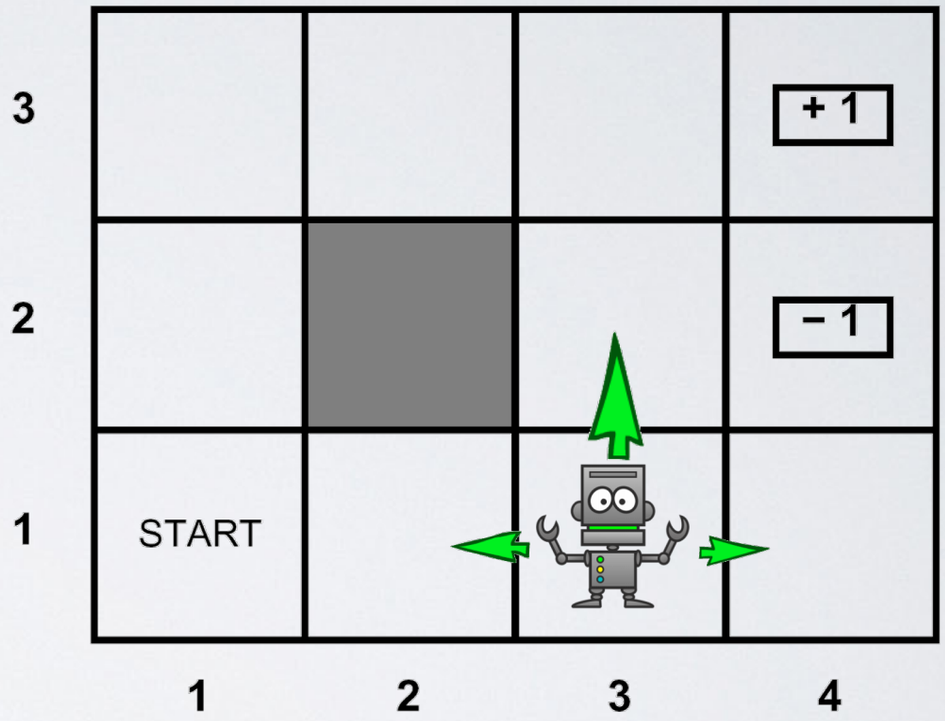
- An MDP is defined by:
  - A set of states s ∈ S
  - A set of actions a ∈ A
  - A transition function T(s, a, s')
    - Returns probability that action a from state s leads to s', i.e., P(s'| s, a)
  - A reward function R(s, a, s')
    - Sometimes just R(s) or R(s')
  - A start state
  - Maybe terminal 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

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)
- MDPs are non-deterministic (probabilistic) search problems



ANDREY MARKOV & MARKOV DECISION PROCESSES (MDP)

- "Markovian Property"
  - Given the present state, the future and the past are independent
- For Markov decision processes, "Markov" means action outcomes depend only on the current state

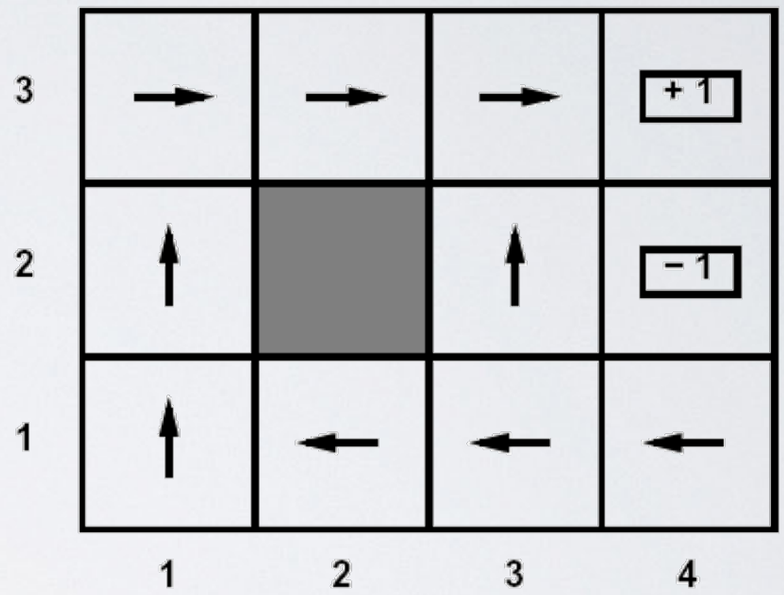
$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1}, \ldots S_0 = s_0)$$
$$=$$
$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t)$$

- This is just like Markov models

Andrey Markov (1856-1922)

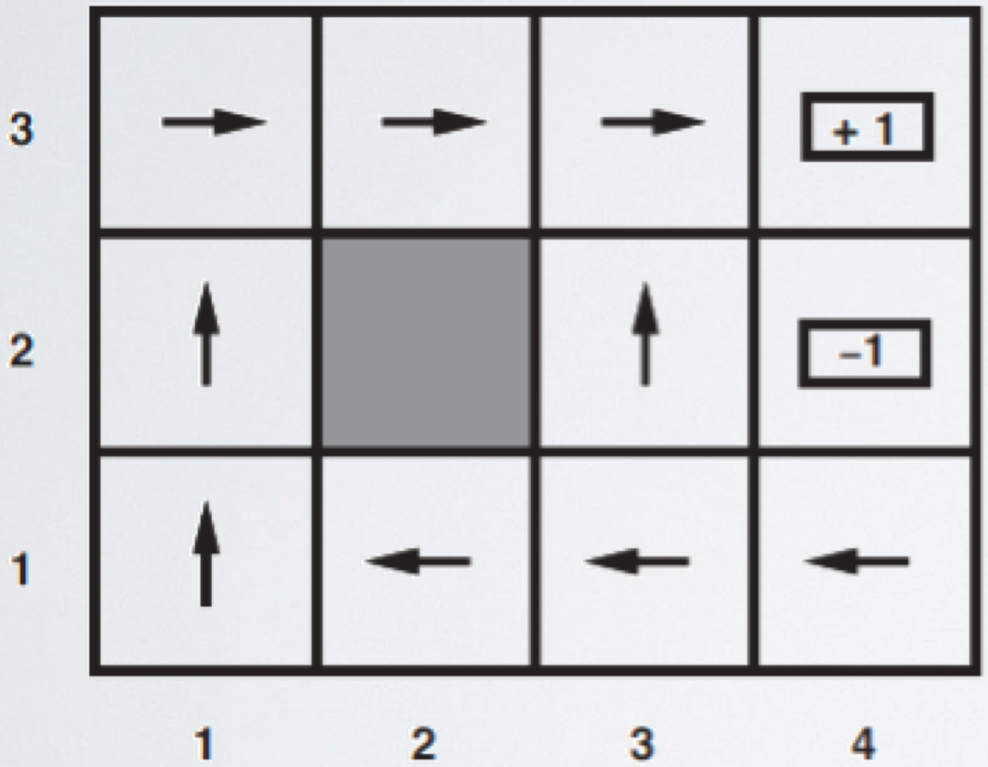
POLICIES

- 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 utility if followed

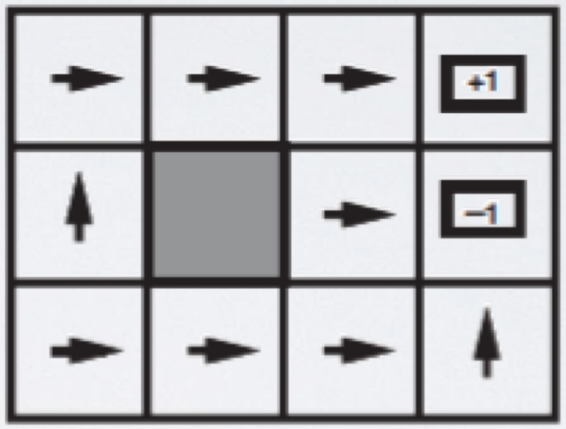


Optimal policy when R(s, a, s') = -0.03 for all non-terminals s (cost of living)

DIFFERENT OPTIMAL POLICIES

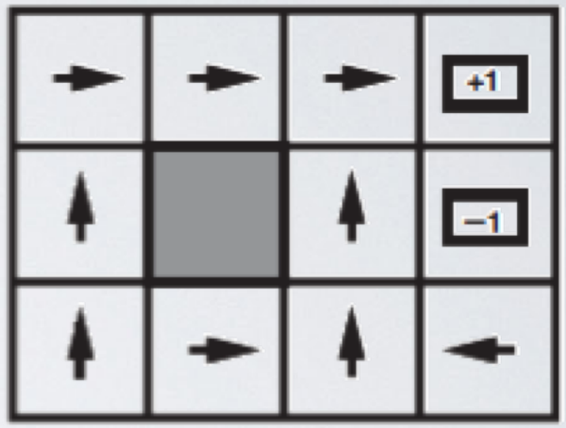


R(s) = -0.03
For s nonterminal



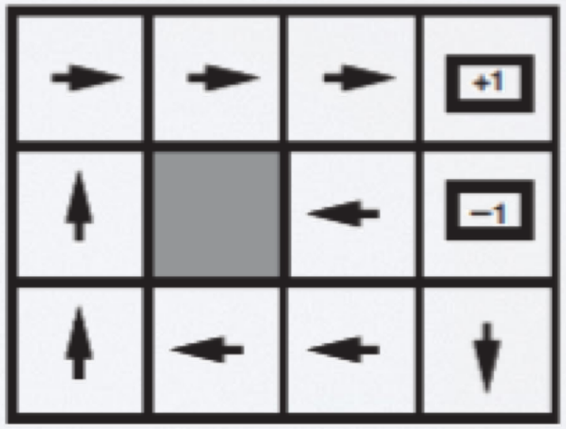
$R(s) < -1.6284$

Life is painful – head to nearest exit



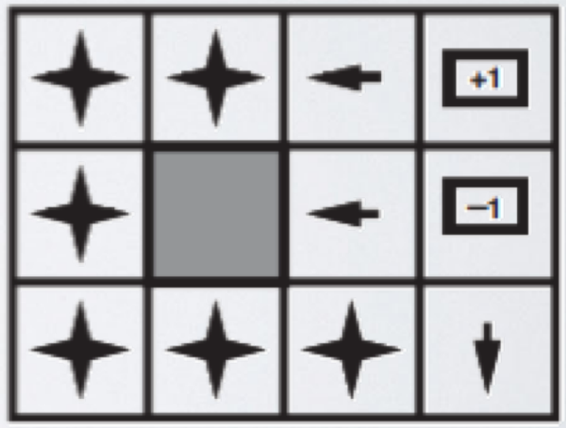
$-0.4278 < R(s) < -0.0850$

Life is somewhat dreary – take risks to +1



$-0.0221 < R(s) < 0$

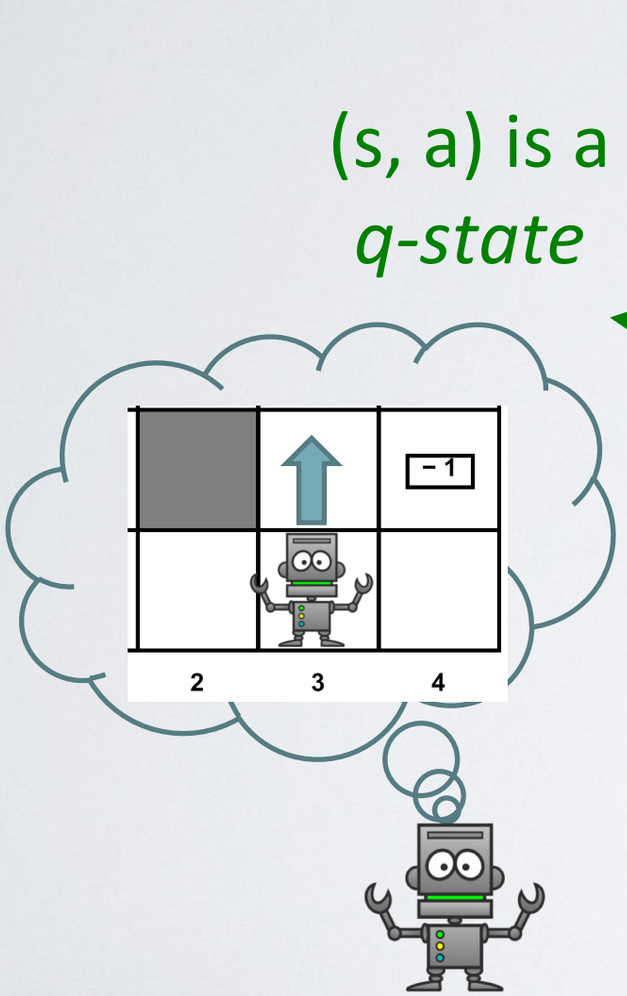
Life is slightly dreary – take no risks
(notice heading towards wall)



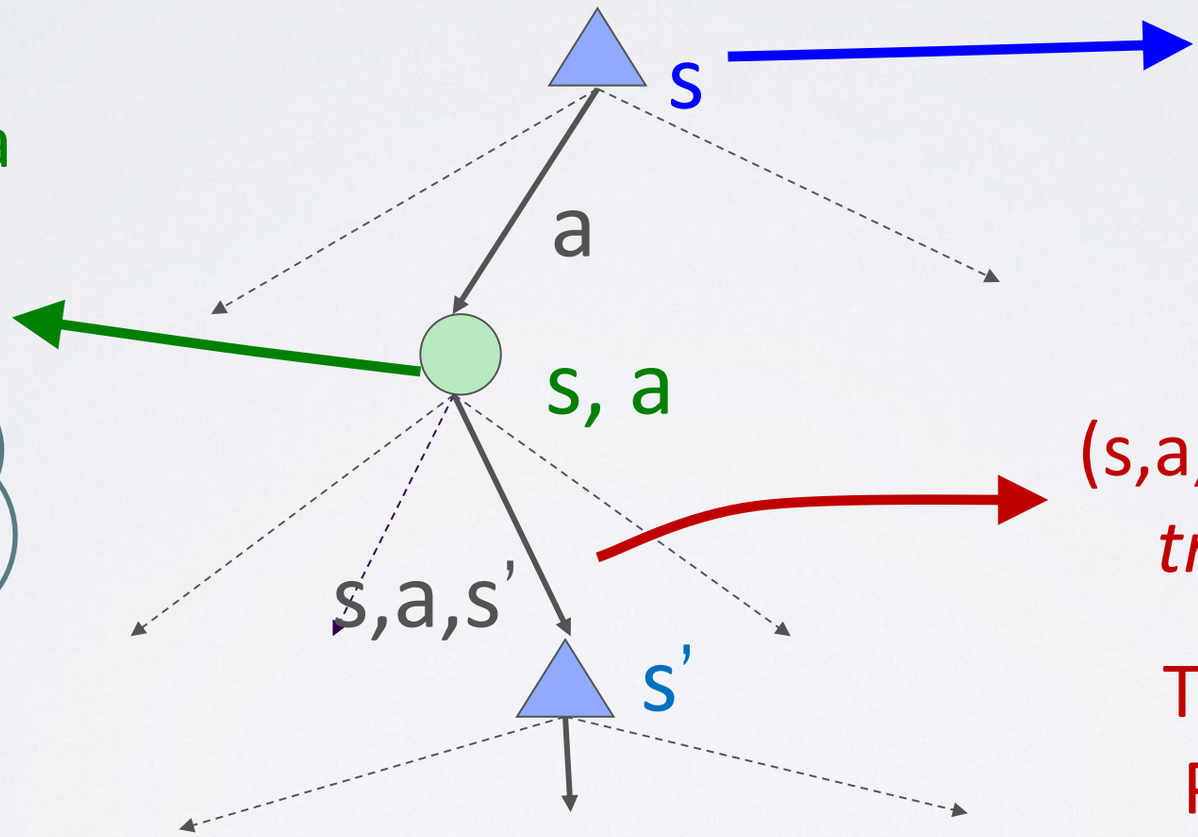
$R(s) > 0$

Life positively enjoyable – avoid exits

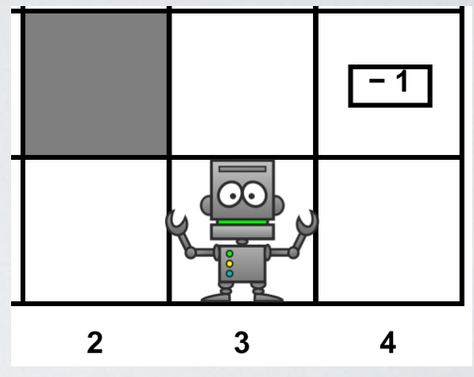
MDP CAST AS SEARCH TREES



(s, a) is a
q-state



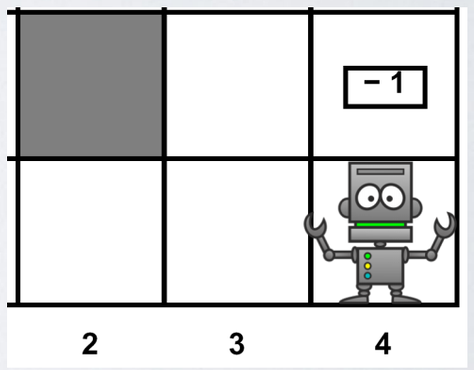
s is a
state



(s,a,s') called a
transition

T(s,a,s') =
P(s'|s,a)

R(s,a,s')



INCREMENTAL REWARD: UTILITIES OF SEQUENCES

- What preferences should an agent have over reward sequences?

- More or less? [2, 3, 4] or [1, 2, 2]

- Now or later? [1, 0, 0] or [0, 0, 2]

NEW: DISCOUNTING, A TREAT NOW OR MORE LATER

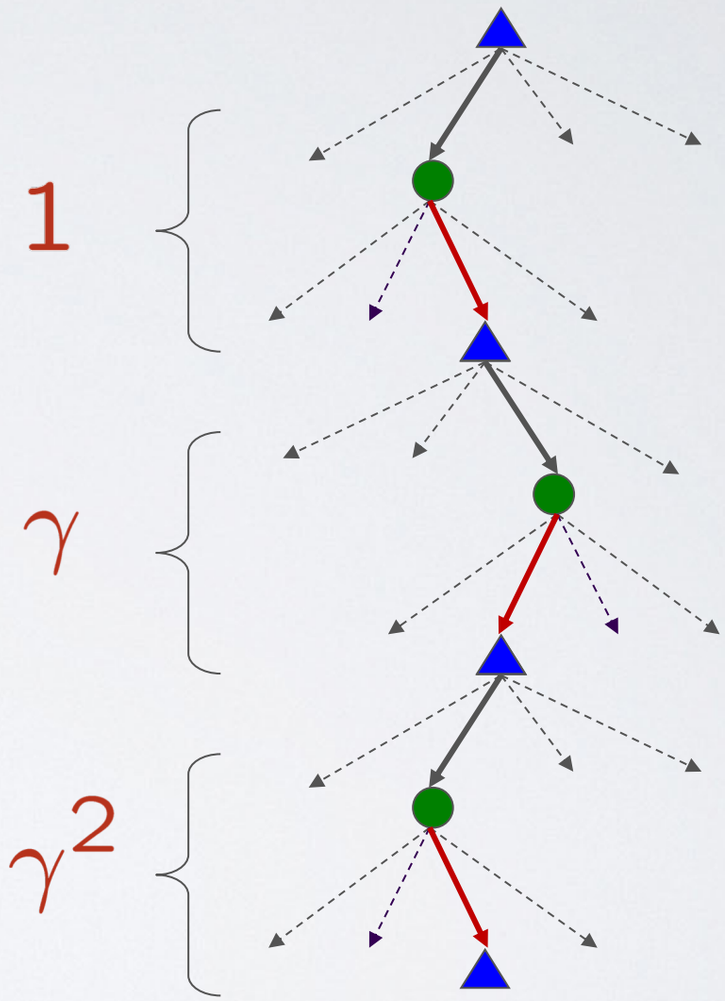
NEW: DISCOUNTING

- Reasonable to maximize the sum of rewards
- Also reasonable to prefer rewards now to rewards later
  - Money earns interest
    - A dollar today is worth more than in the future
  - Agent may die
    - Probability of death = $1 - \gamma$, $0 < \gamma \leq 1$
  - You're hungry
  - Mathematically tractable (will see soon)
- One solution: values of rewards decay exponentially, $0 < \gamma \leq 1$

Worth Now	Worth Next Step	Worth In Two Steps
1	γ	γ^2

DISCOUNTING

- How to discount?
  - Each step we extend into future, we multiply in the discount once
- Why discount?
  - Sooner rewards probably do have higher utility than later rewards
  - Also helps our algorithms converge
- Example: discount of 0.5
  - U([1,2,3]) = 1*1 + 0.5*2 + 0.25*3
  - U([1,2,3]) < U([3,2,1])



STATIONARY PREFERENCES

- Theorem: if we assume stationary preferences:

$$[a_1, a_2, \ldots] \succ [b_1, b_2, \ldots]$$

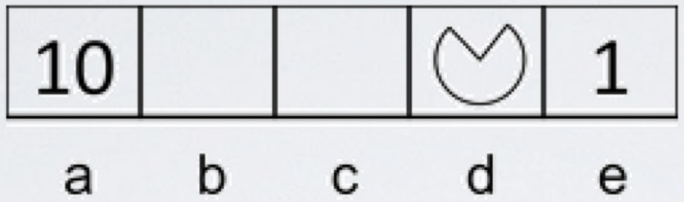
$$\Updownarrow$$

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

- Then: there are only two ways to define utilities
  - Additive utility: $U([r_0, r_1, r_2, \ldots]) = r_0 + r_1 + r_2 + \cdots$
  - Discounted utility: $U([r_0, r_1, r_2, \ldots]) = r_0 + \gamma r_1 + \gamma^2 r_2 \cdots$

QUIZ: DISCOUNTING

- Given:

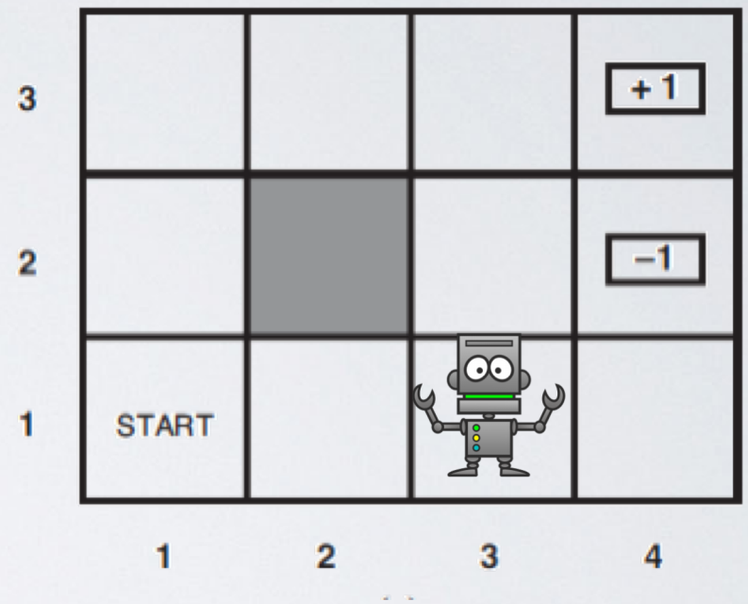


  - Actins: East, West, and Exit (only available in exit states a, e)
  - Transitions: deterministic

- Quiz 1: For $\gamma = 1$, what is the optimal policy?
- Quiz 2: For $\gamma = 0.1$, what is the optimal policy?
- Quiz 3: For which γ are West and East equally good when in state d?
  - $10\gamma^2 = 1$

UTILITIES OVER TIME: FINITE OR INFINITE HORIZON?

- If there is fixed time, N, after which nothing can happen, what should an agent do?
  - E.g., if N=3, Bot must head directly for +1 state
  - If N =100, can take safe route
- So with finite horizon, optimal action changes over time
  - Optimal policy is nonstationary
  - (π depends on time left)



MODELS OF OPTIMAL BEHAVIOR

- In the *finite-horizon* model, agent should optimize expected reward for the next *H* steps:

$$\mathbb{E}\left(\sum_{t=0}^{H} r_t\right)$$

- In the *infinite-horizon* discounted model agent should optimize:

$$\mathbb{E}\left(\sum_{t=0}^{\infty} \gamma^t r_t\right)$$

- Discount factor $0 \leq \gamma < 1$ can be thought of as an interest rate (reward now is worth more than reward in the future)

INFINITE UTILITIES?!

- Problem: What if the game lasts forever? Do we get infinite rewards?
- Solutions:
  - Discounting: use $0 < \gamma < 1$

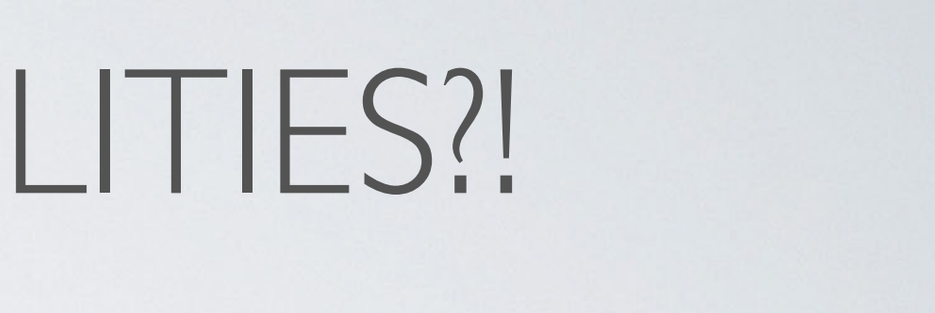
$$U([r_0, \ldots r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_t \leq R_{\max}/(1-\gamma)$$

    - Smaller γ means small shorter term focus

Geometric series

If Rmax = 1/2 and discount = ½

1 + ½ + ¼ +1/8 + 1/16 ….

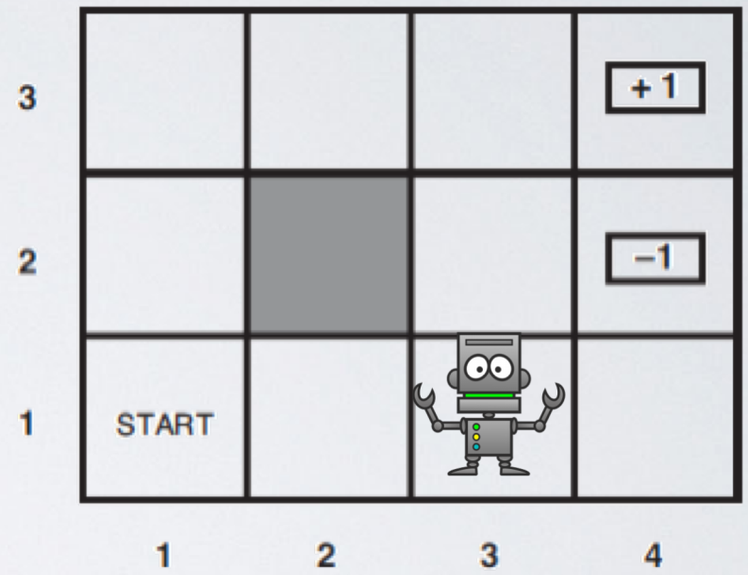


- Absorbing state: guarantee that for every policy, a terminal state will eventually be reached
  - Called a proper policy - won't consider for now

CHOOSING A REWARD FUNCTION

<u>A few possibilities:</u>

- All reward on goals
- Negative reward everywhere except terminal states
- Gradually increasing reward as you approach the goal

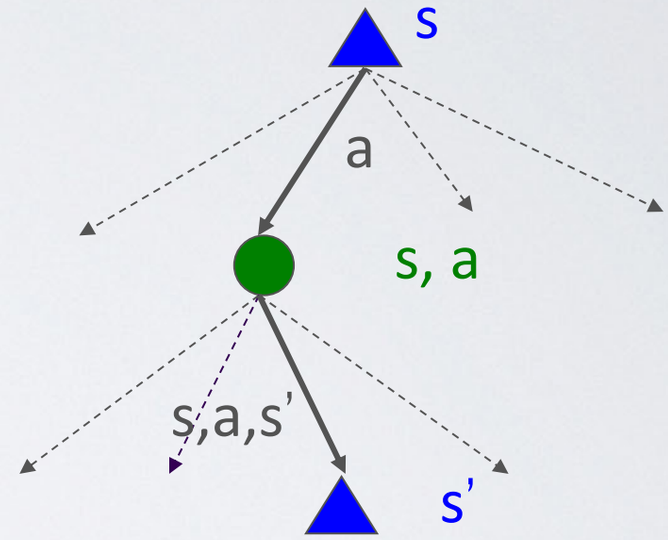


<u>In general:</u>

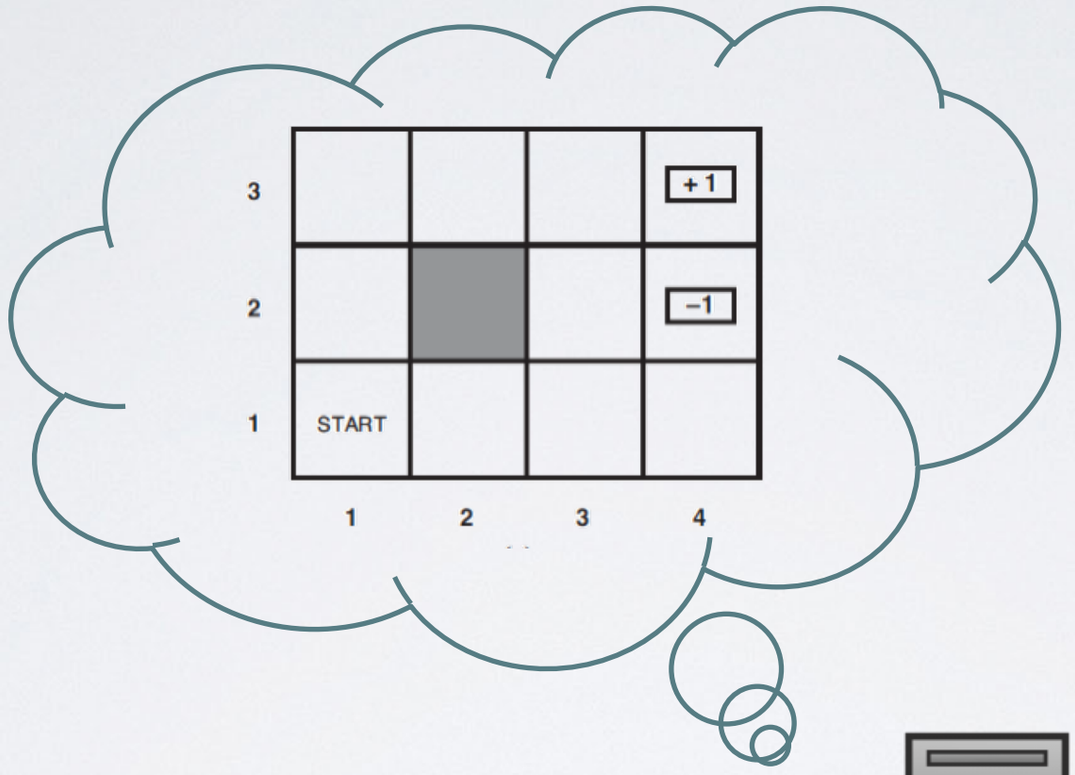
- Reward can be whatever you want

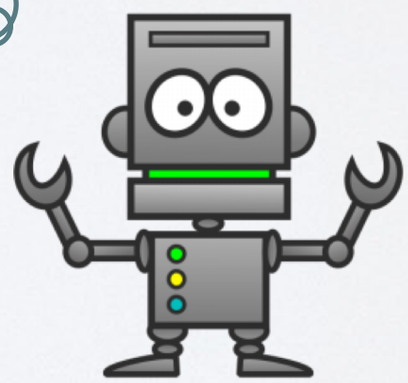
SO FAR: DEFINING MDPS

- Markov decision processes:
  - Set of states S
  - Start state s_0
  - Set of actions A
  - Transitions P(s'|s,a) (or T(s,a,s'))
  - Rewards R(s,a,s') (and discount γ)
- MDP calculations so far:
  - Policy = Choice of action for each state
  - Utility = sum of (discounted) rewards



SOLVING MDPS





SOLVING MDPS

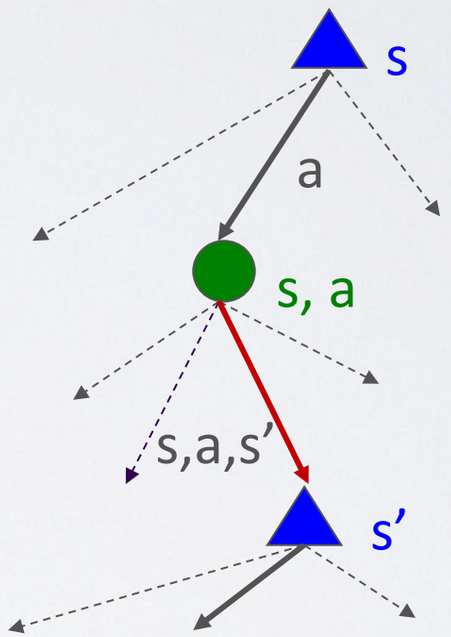
Want an optimal Policy

- Policy gives an action for each state
- Maximizes expected sum of rewards

OPTIMAL QUANTITIES

- The value (utility) of a state s:
  $V^*(s)$ = expected utility starting in s and acting optimally

- The value (utility) of a q-state (s,a):
  $Q^*(s,a)$ = expected utility starting out having taken action a from state s and (thereafter) acting optimally



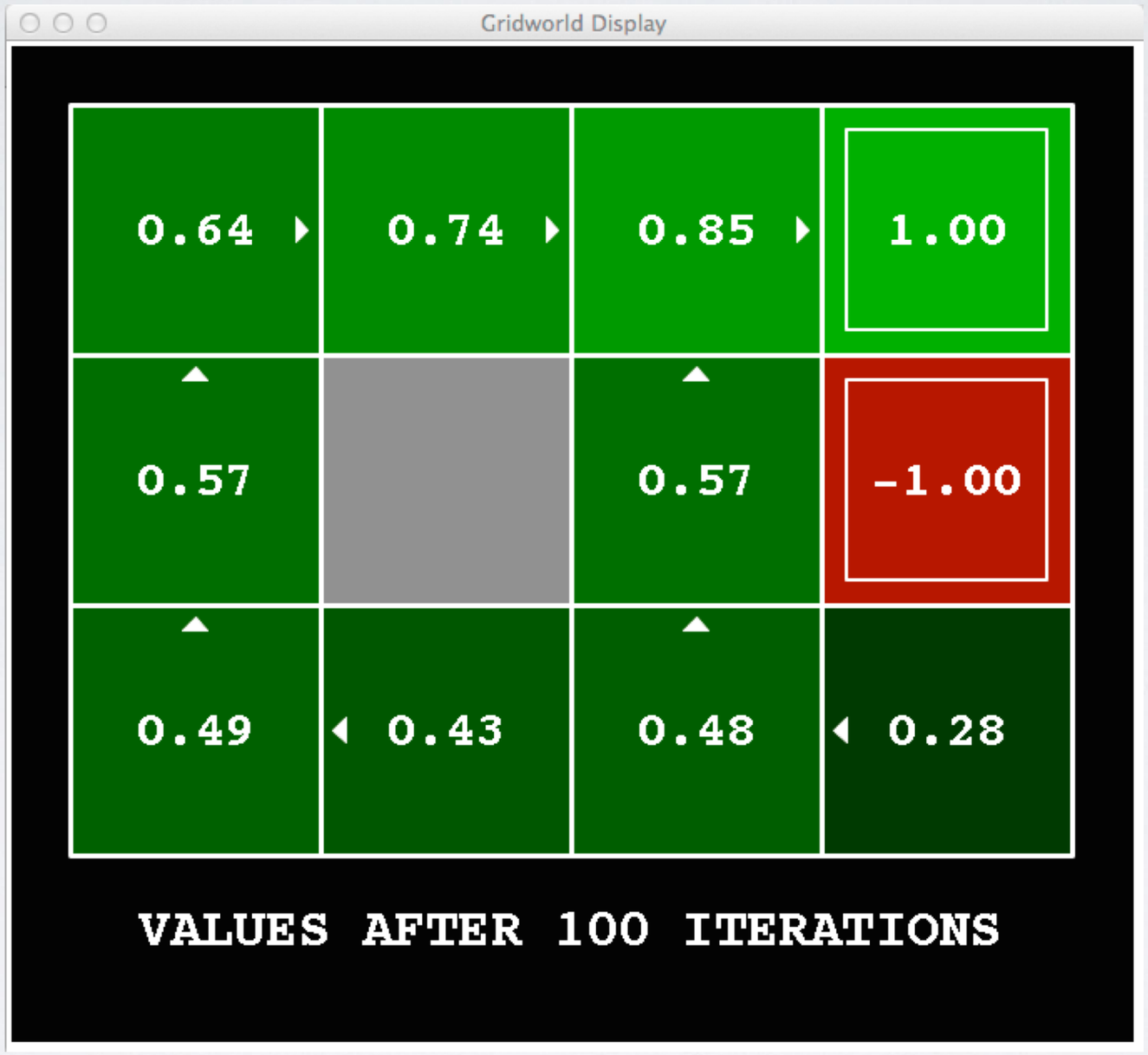
s is a *state*

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

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

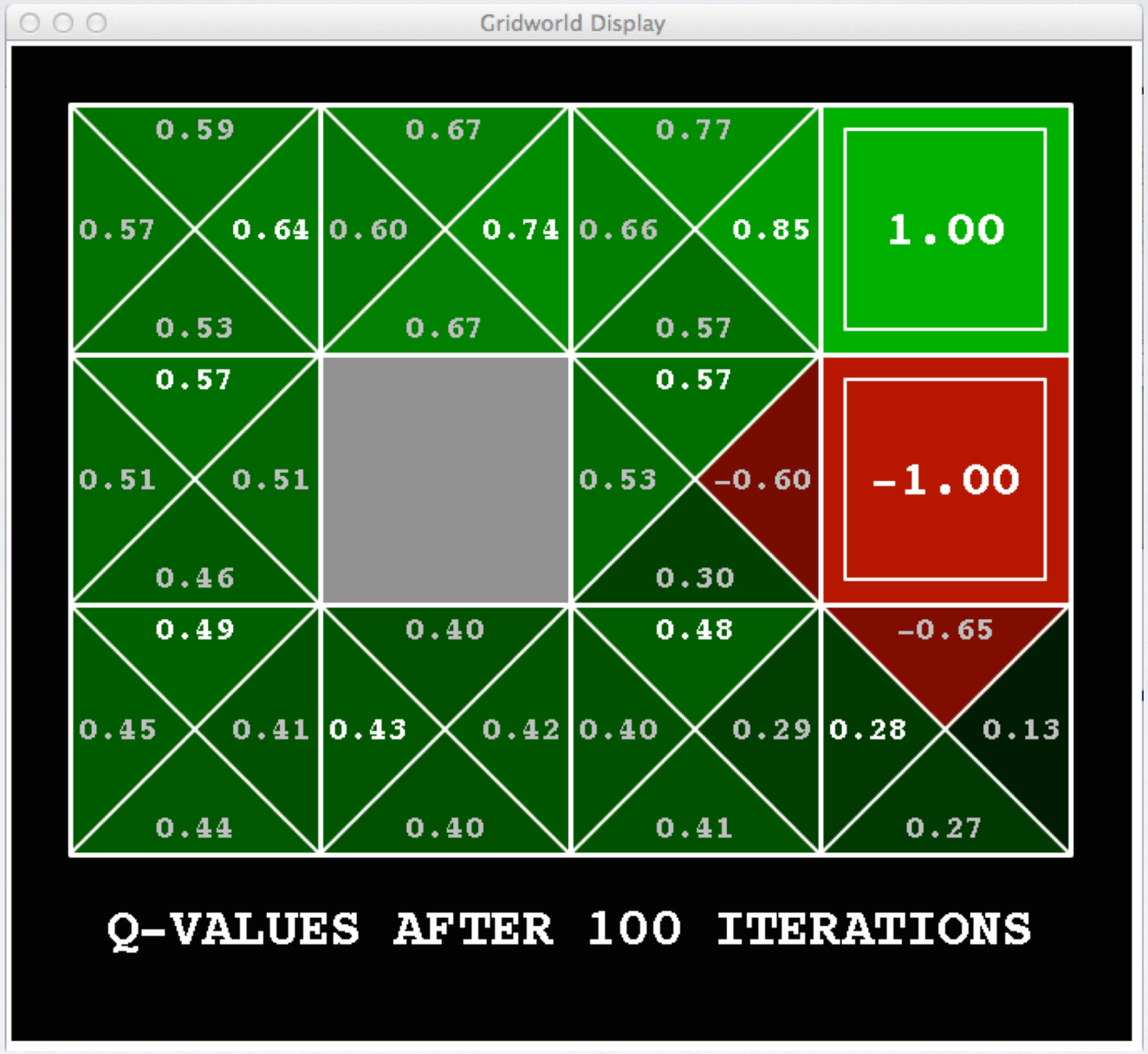
- The optimal policy:
  $\pi^*(s)$ = optimal action from state s

GRIDWORLD V VALUES



Noise = 0.2
Discount = 0.9
Living reward = 0

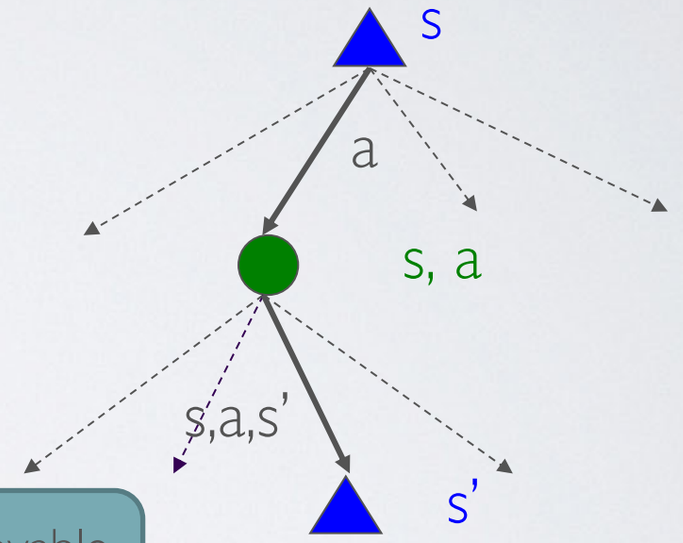
GRIDWORLD Q VALUES



Noise = 0.2
Discount = 0.9
Living reward = 0

VALUES OF STATES

- Fundamental operation: compute the (expectimax) value of a state
  - Expected utility under optimal action
  - Average sum of (discounted) rewards



- Recursive definition of value:

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

Max value achievable from state s

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

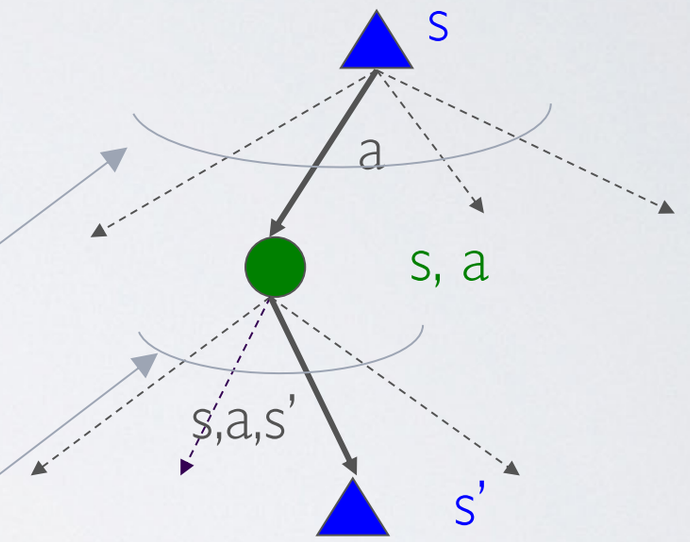
Value achievable from state s if you take action a and then act

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

Combine

VALUES OF STATES

- Fundamental operation: compute the (expectimax) value of a state
  - Expected utility under optimal action
  - Average sum of (discounted) rewards



- Recursive definition of value:

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

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

Probability of transition to s'

Reward for transition

Future reward if acting optimally discounted by gamma

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

BELLMAN EQUATION

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

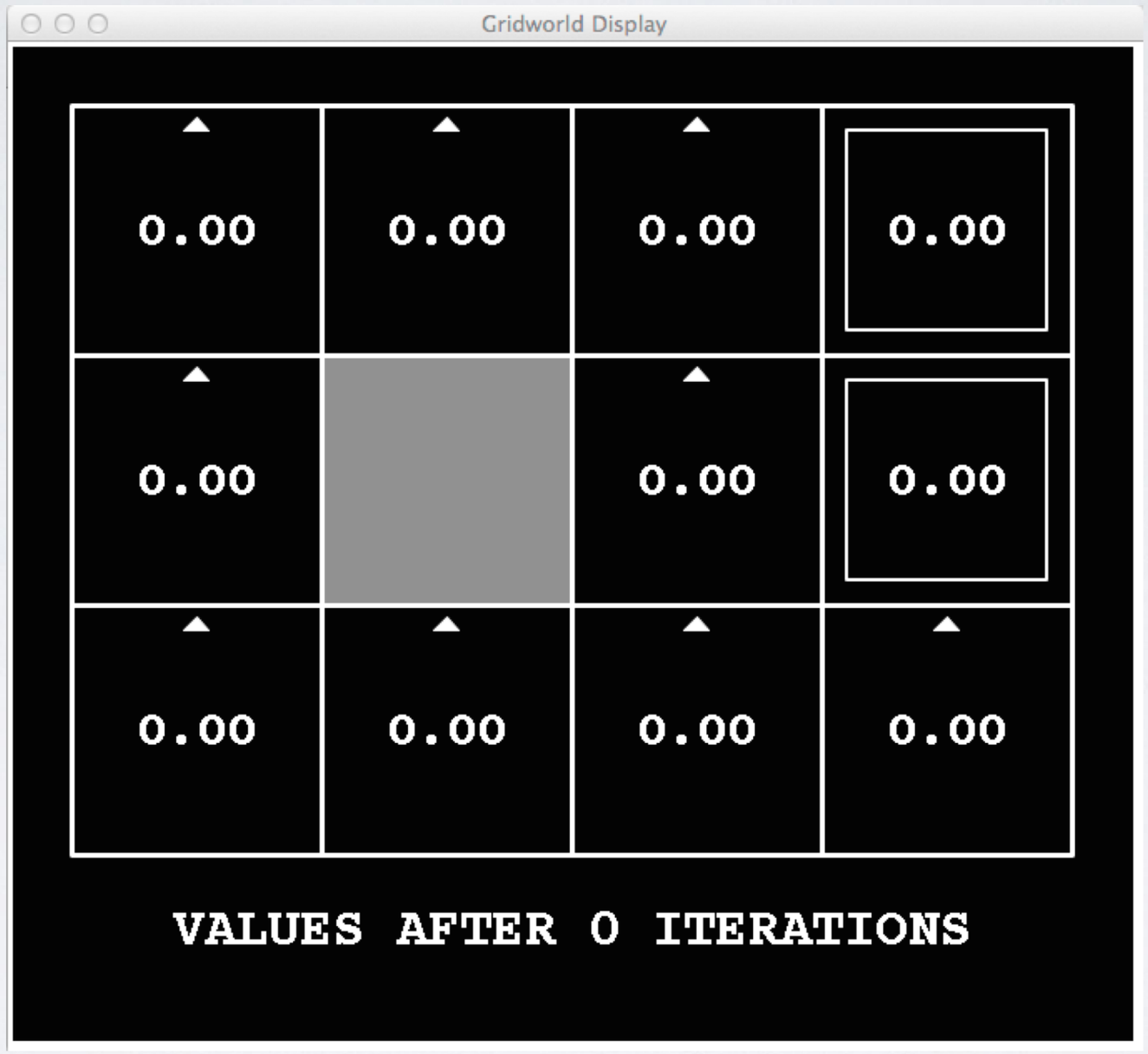
Richard Bellman
1920 –1984

- With this equation, Bellman introduced dynamic programming in 1953

VALUE ITERATION

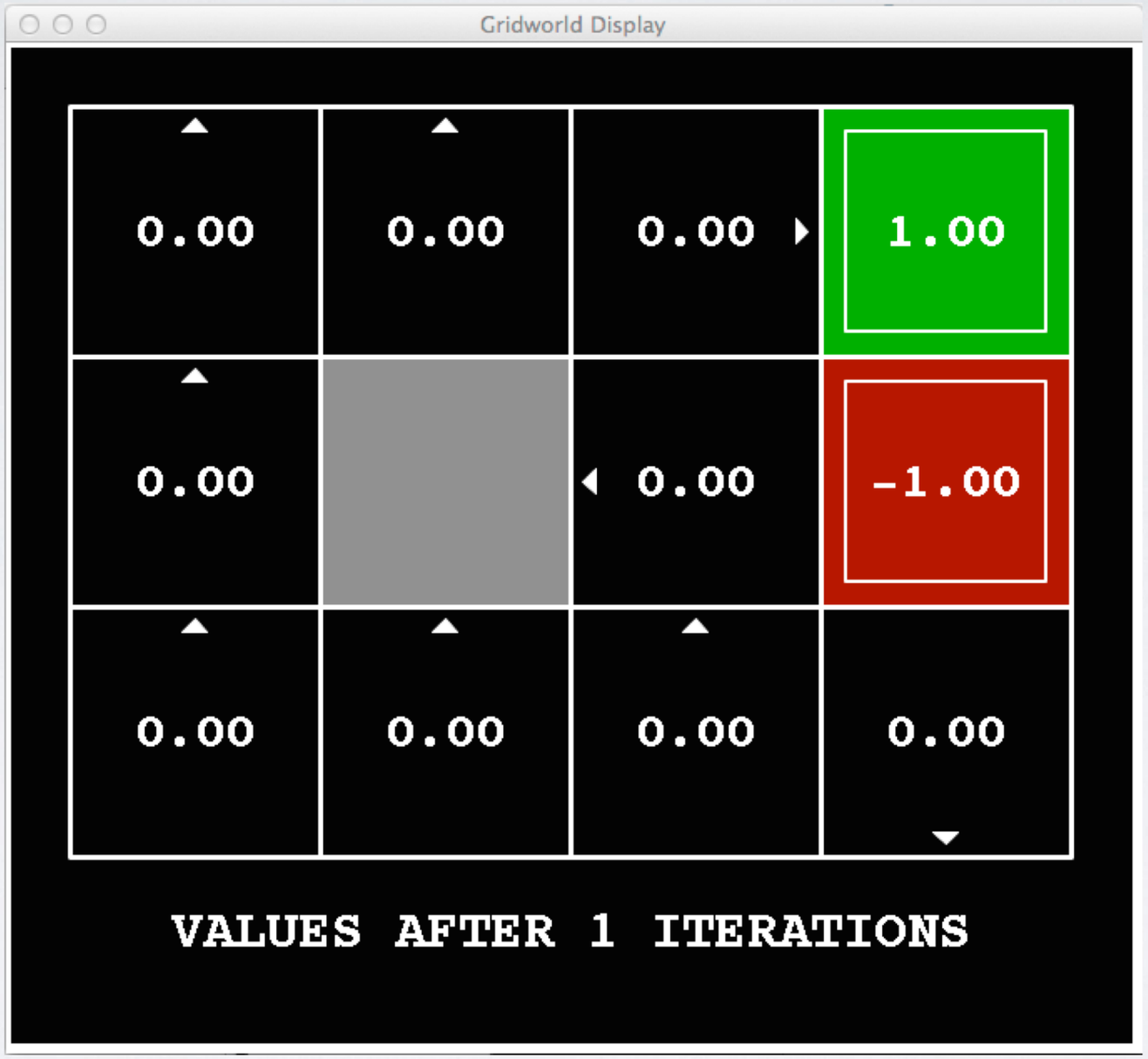
- Build up V*(s) in Layers
- Starting at $V_0(s)$
- Until convergence
  - Max within some ε

K=0



Noise = 0.2
Discount = 0.9
Living reward = 0

K=1



Noise = 0.2
Discount = 0.9
Living reward = 0

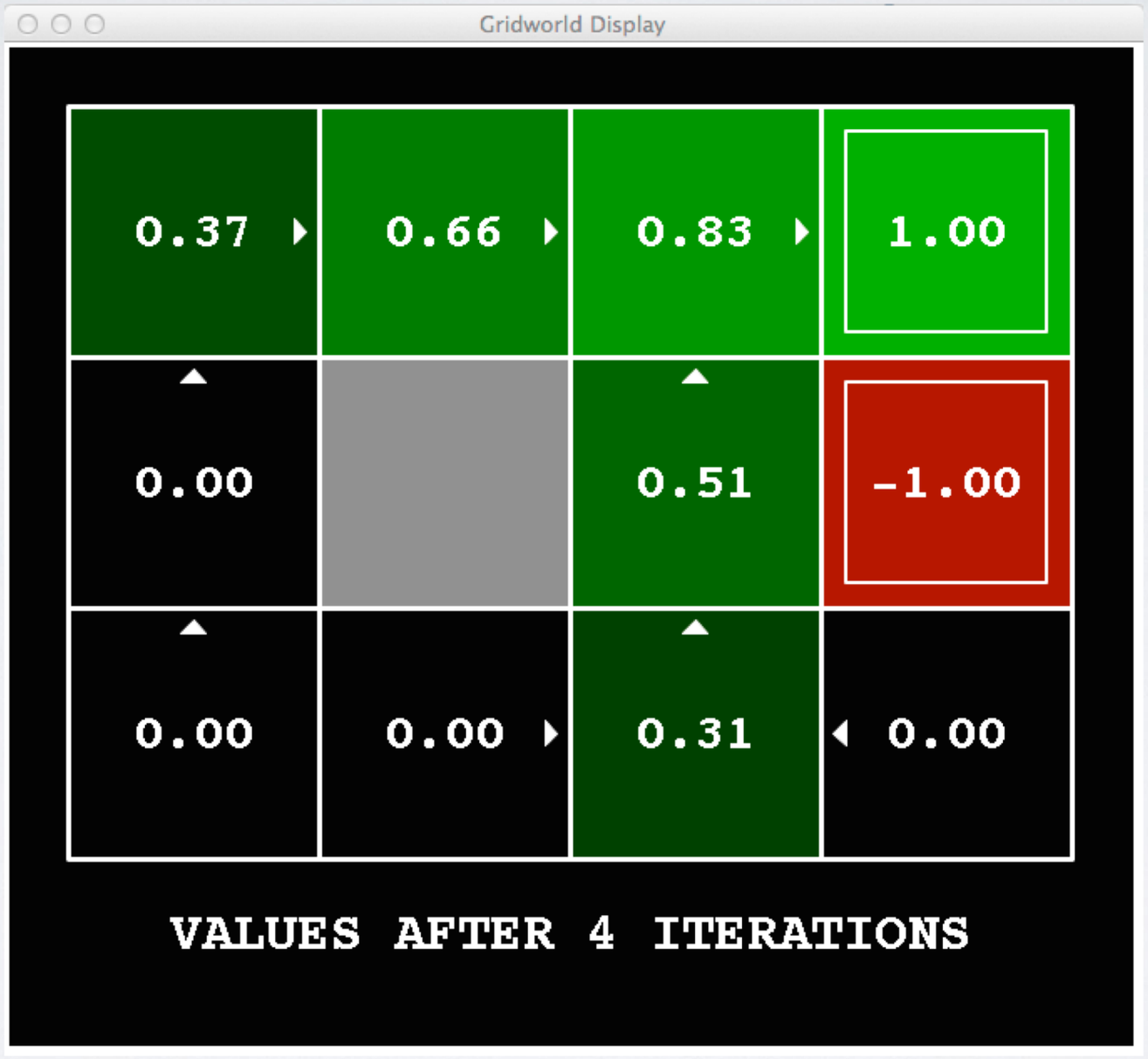
K=2

Noise = 0.2
Discount = 0.9
Living reward = 0

K=3

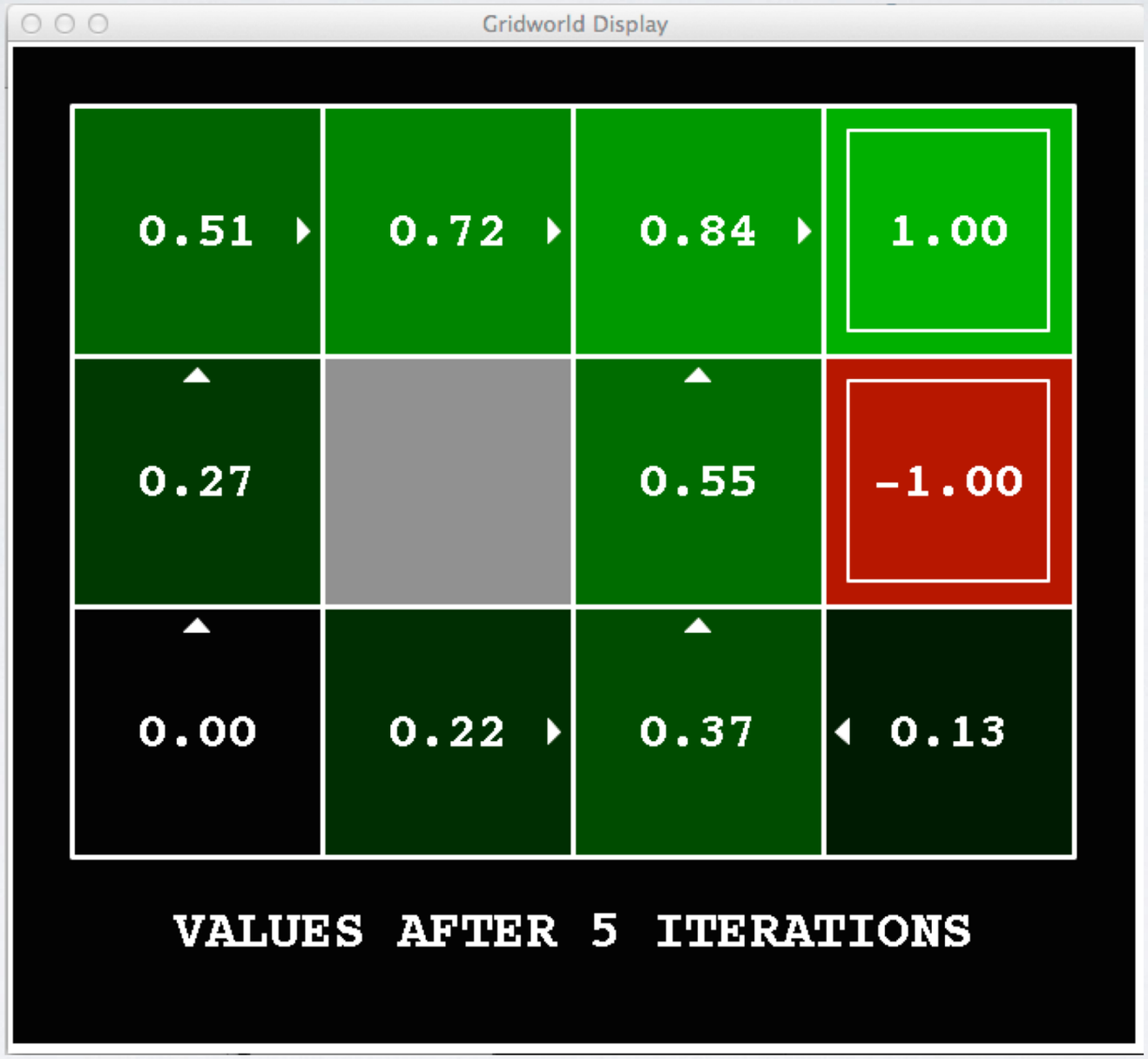
Noise = 0.2
Discount = 0.9
Living reward = 0

K=4



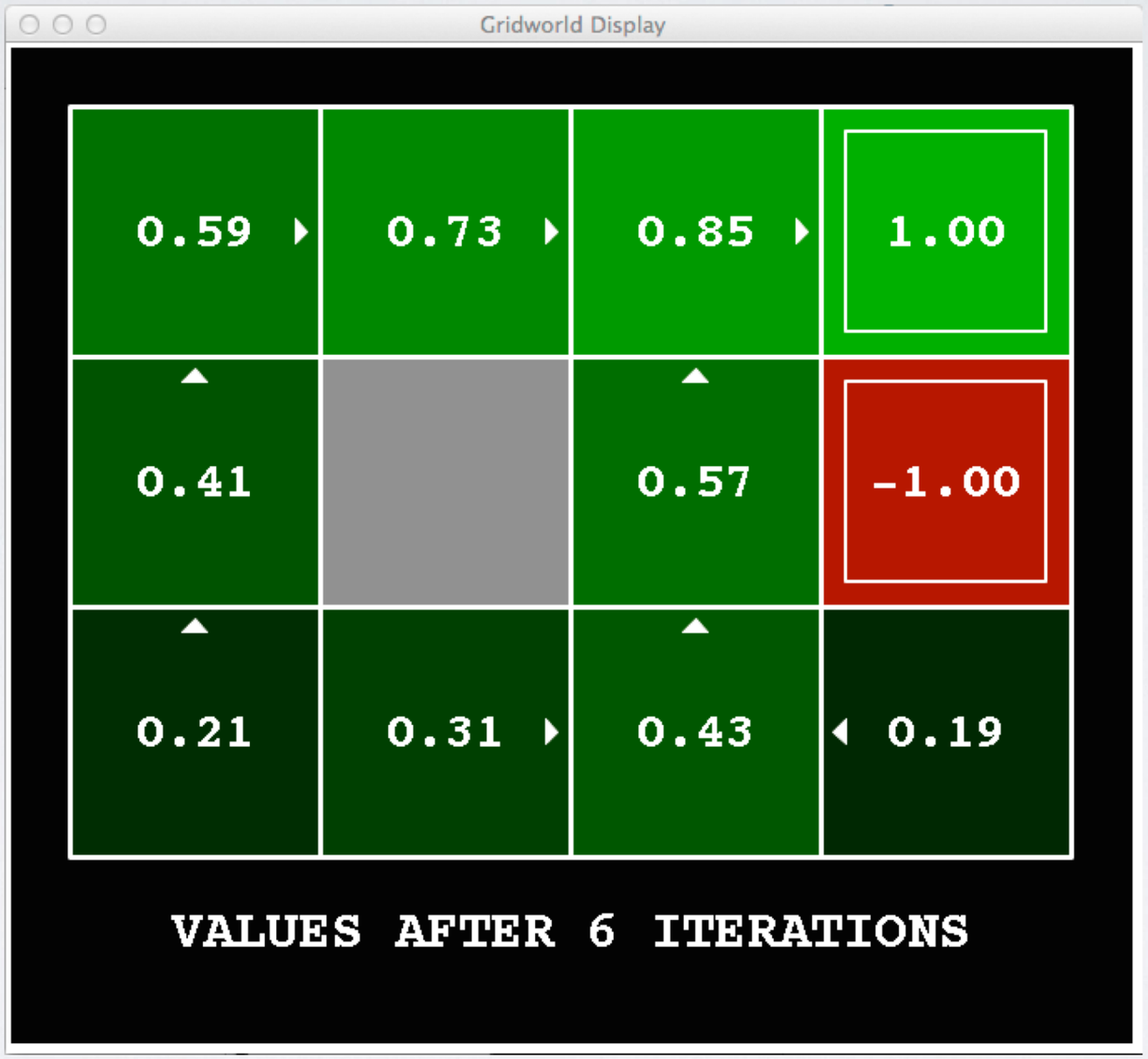
Noise = 0.2
Discount = 0.9
Living reward = 0

K=5



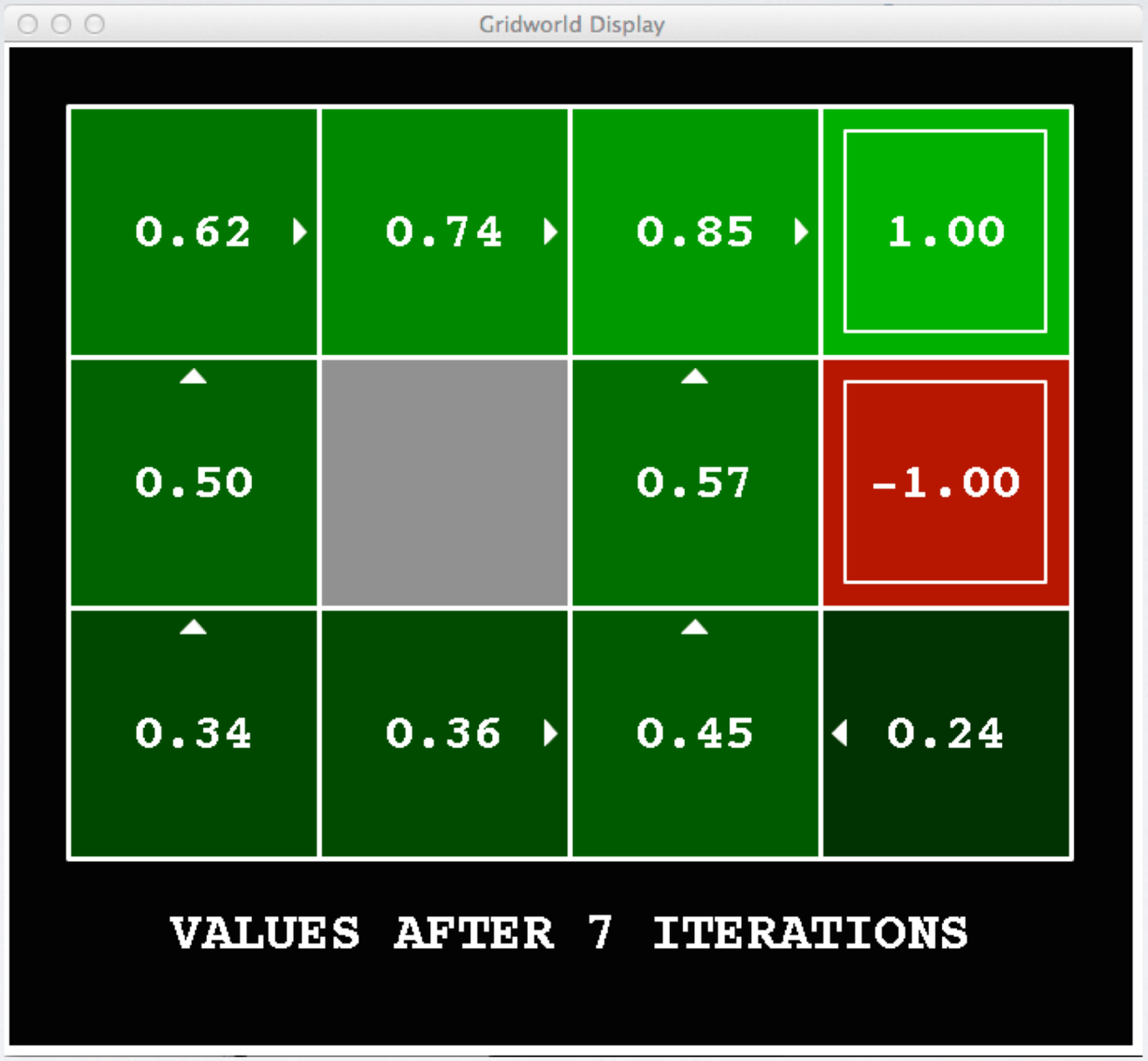
Noise = 0.2
Discount = 0.9
Living reward = 0

K=6



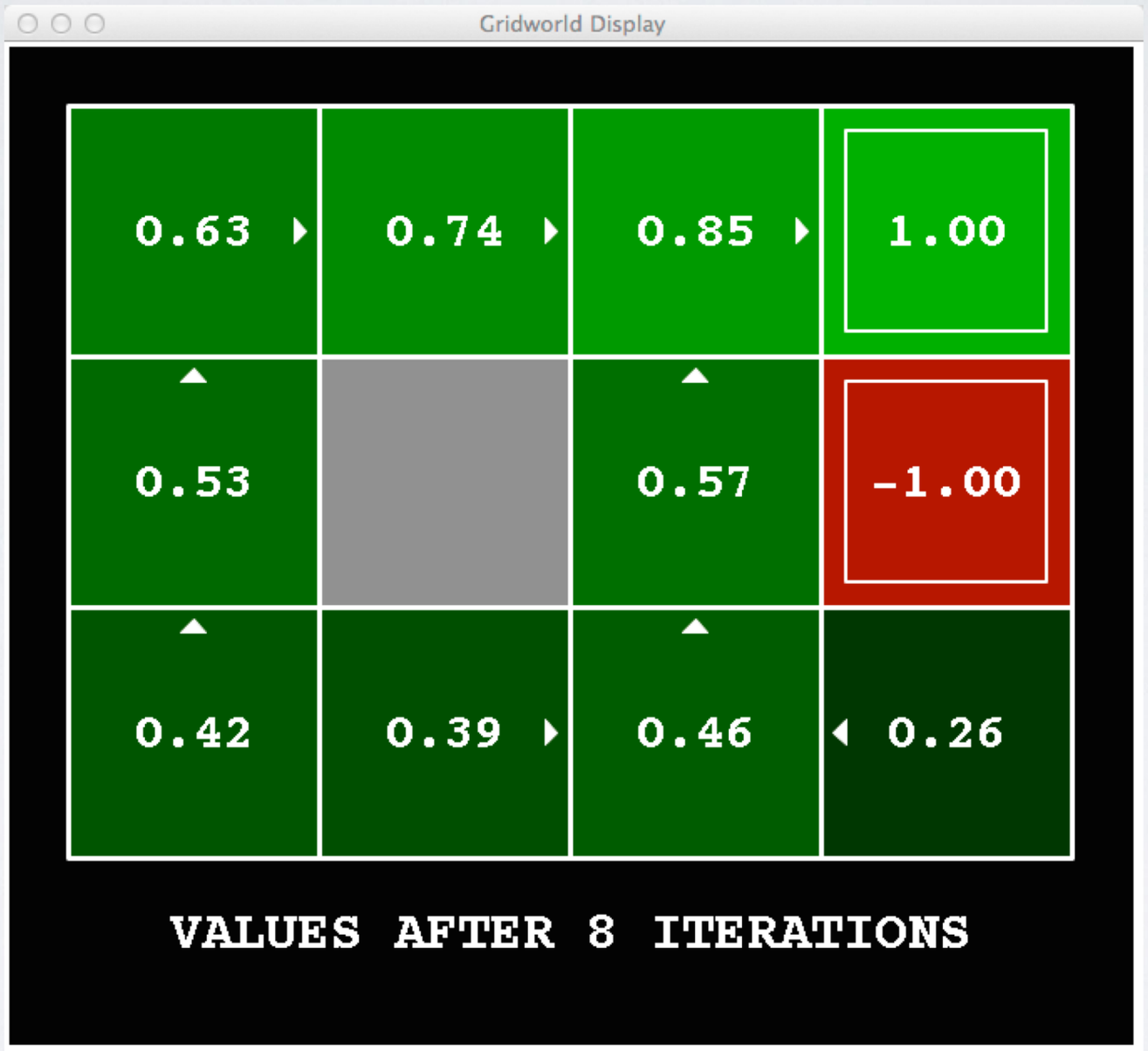
Noise = 0.2
Discount = 0.9
Living reward = 0

K=7



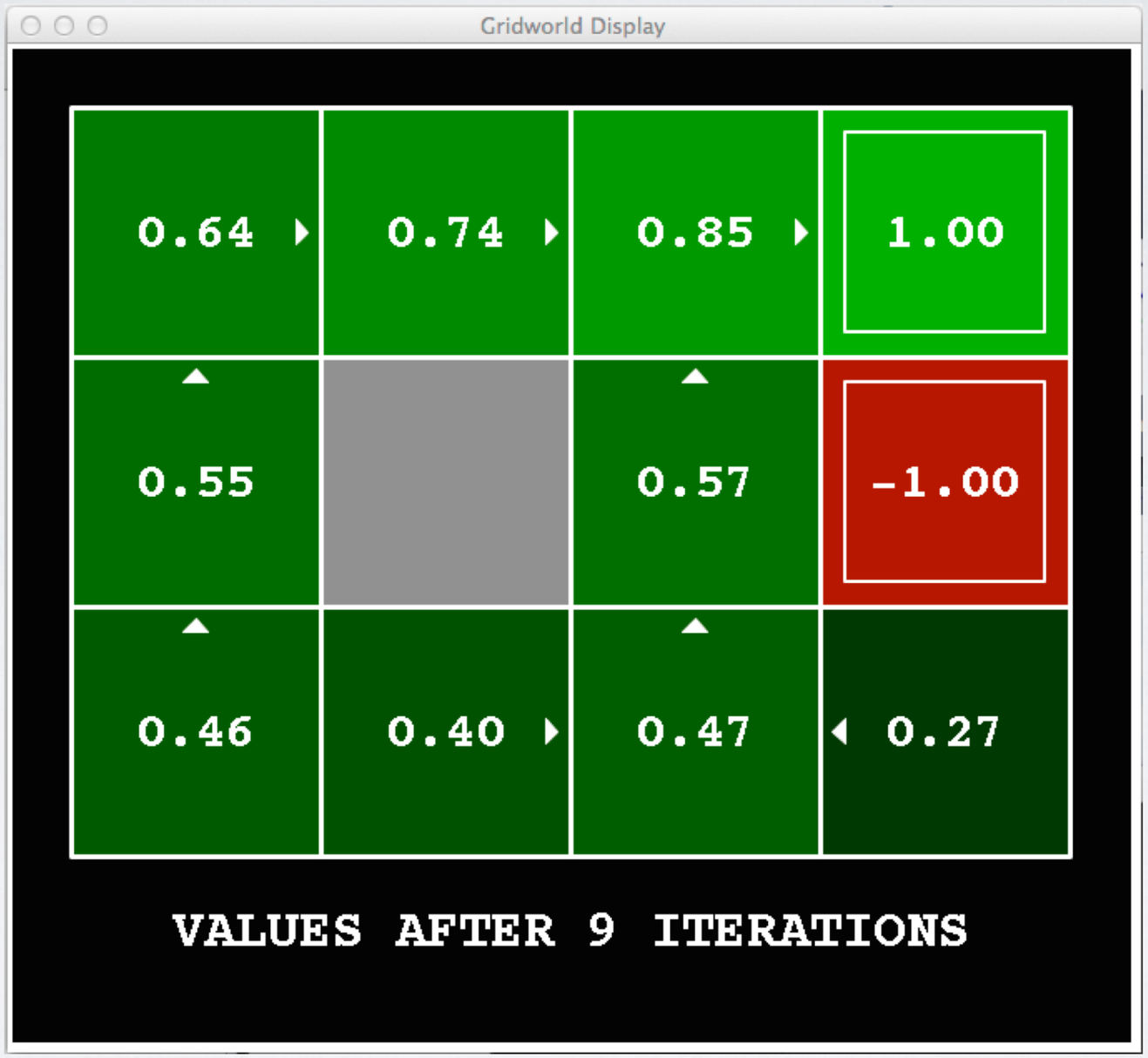
Noise = 0.2
Discount = 0.9
Living reward = 0

K=8



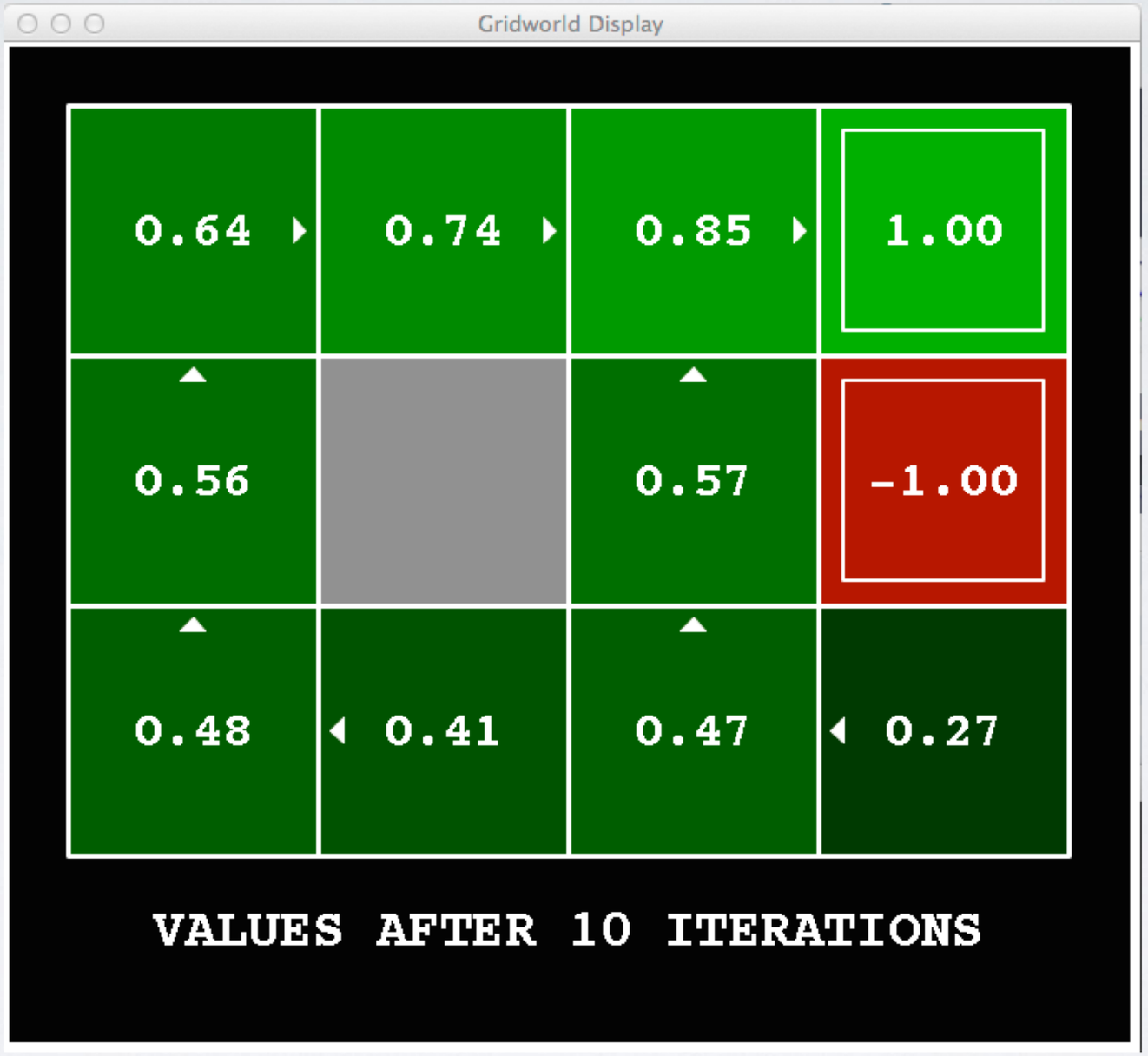
Noise = 0.2
Discount = 0.9
Living reward = 0

K=9



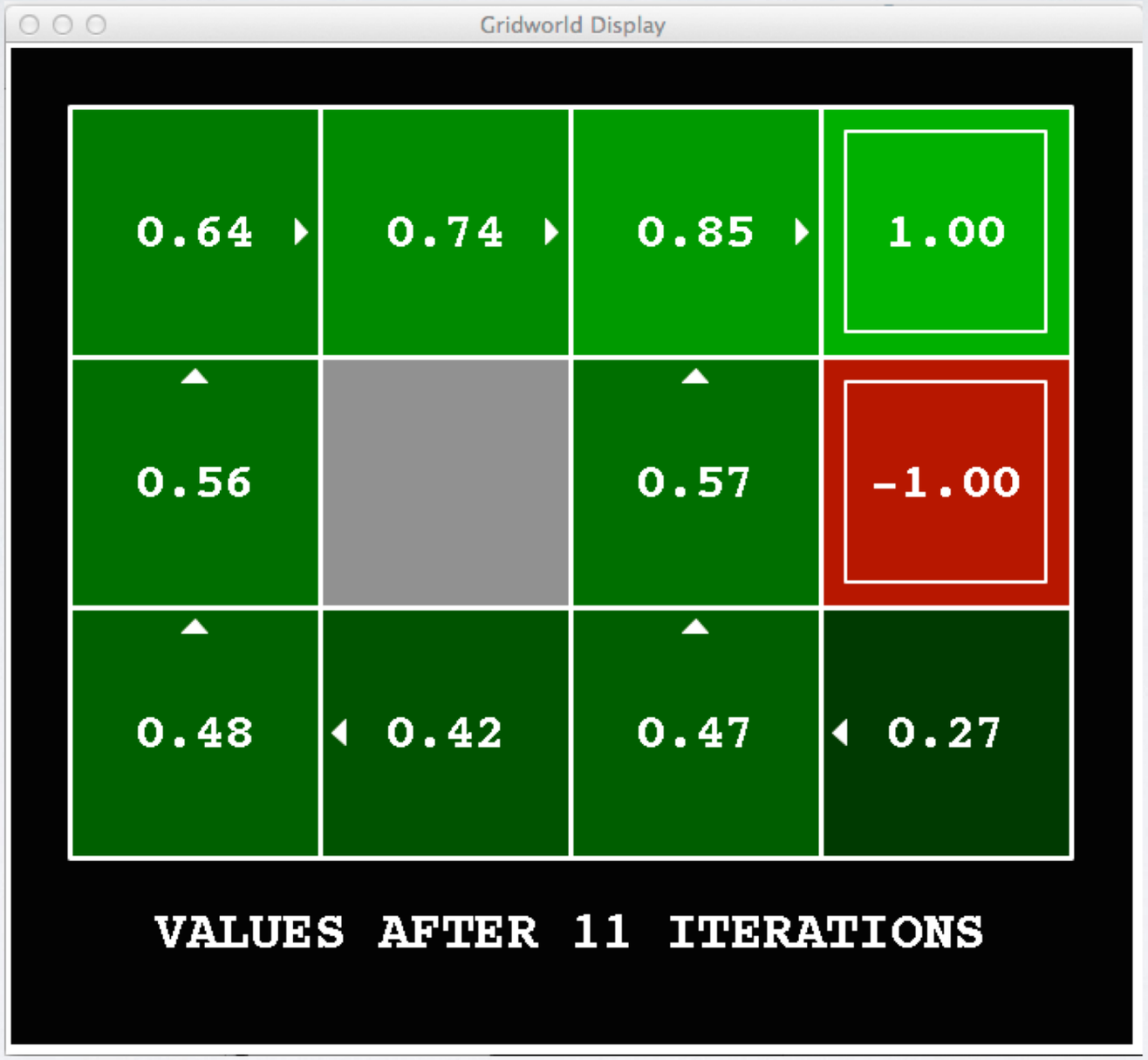
Noise = 0.2
Discount = 0.9
Living reward = 0

K=10



Noise = 0.2
Discount = 0.9
Living reward = 0

K=11

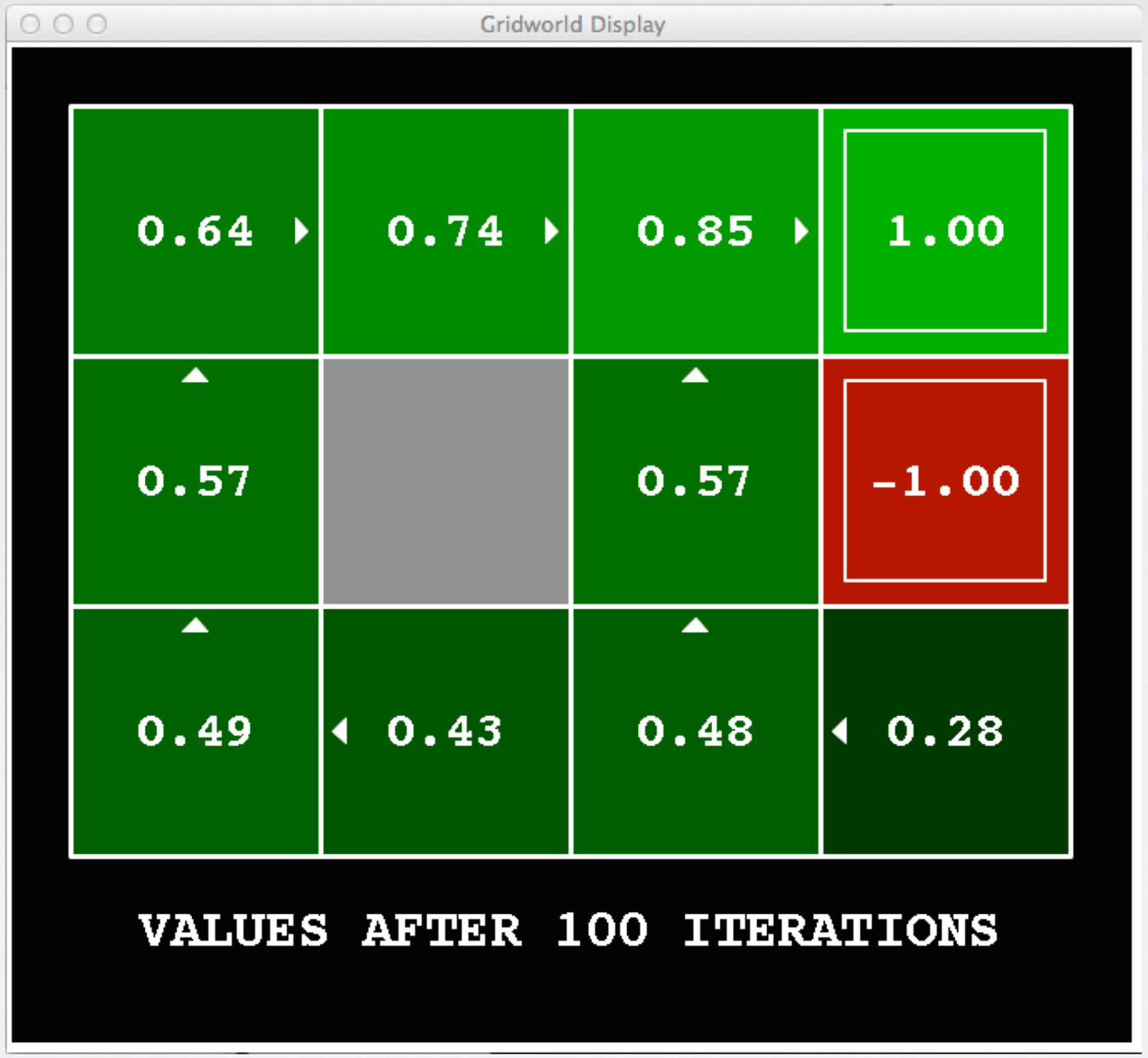


Noise = 0.2
Discount = 0.9
Living reward = 0

K=12

Noise = 0.2
Discount = 0.9
Living reward = 0

K=100

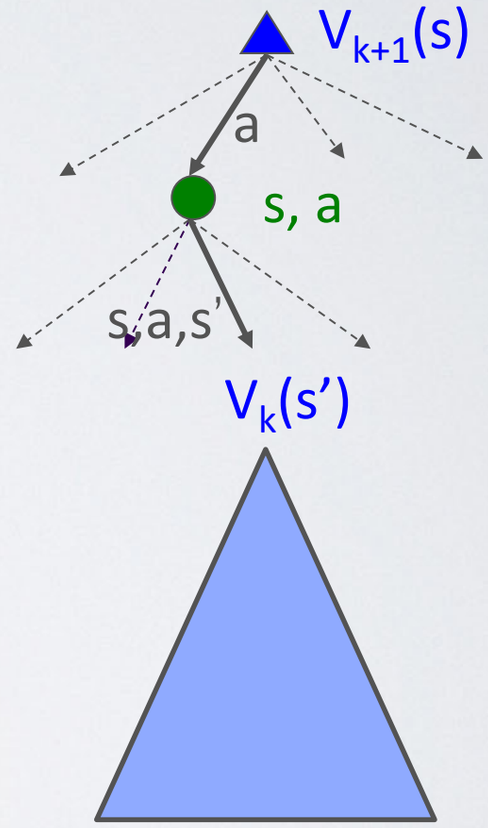


Noise = 0.2
Discount = 0.9
Living reward = 0

VALUE ITERATION

- Start with $V_0(s) = 0$: no time steps left means an expected reward sum of zero
- Given vector of $V_k(s)$ values, do one "backup" from each state:

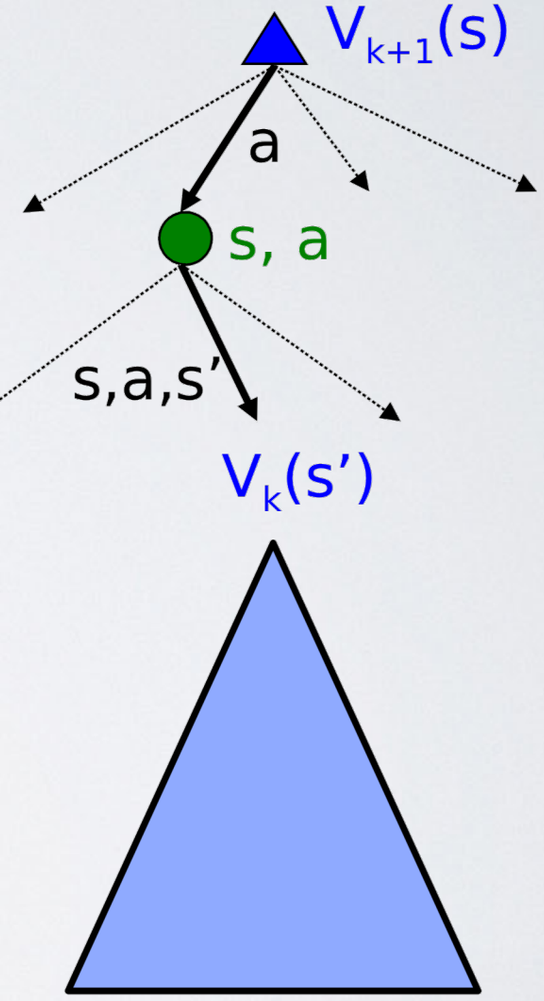
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right]$$



- Repeat until convergence
  - Max_s ($|V_{n+1}(s) - V_n(s)|$) < ε (where ε depends on γ)
- Complexity of each iteration: $O(S^2A)$
- Theorem: will converge to unique optimal values
  - Basic idea: approximations get refined towards optimal values
  - Policy may converge long before values do

VALUE ITERATION

Value of *s* at *k* timesteps to go: $V_k(s)$



<u>Value iteration:</u>

1. initialize $V_0(s) = 0$

2. $V_1(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_0(s')\right]$

3. $V_2(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_1(s')\right]$

4. ....

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

EXAMPLE: VALUE ITERATION

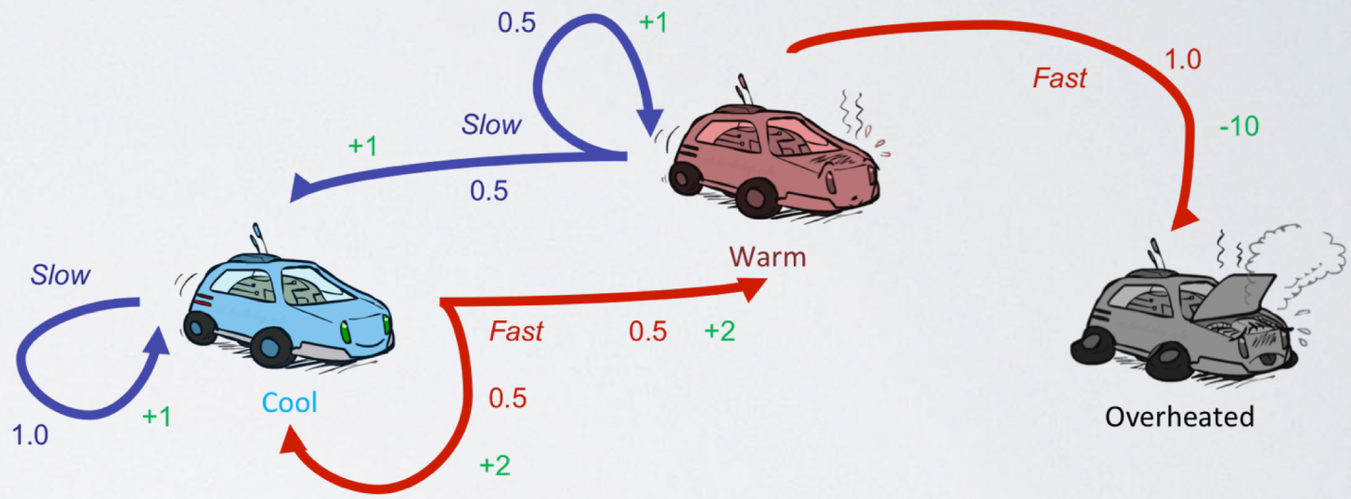
V_2	3.5	2.5	0

S=1.0 [1 + V_1(c)]
F=.5 [2+ V_1(c)]+.5[2+V_1(w)]

V_1	2	1	0

S=1.0 [1 + V_0(c)]
F=.5[2+V_0(c)] + .5[2 + V_0(w)]

V_0	0	0	0

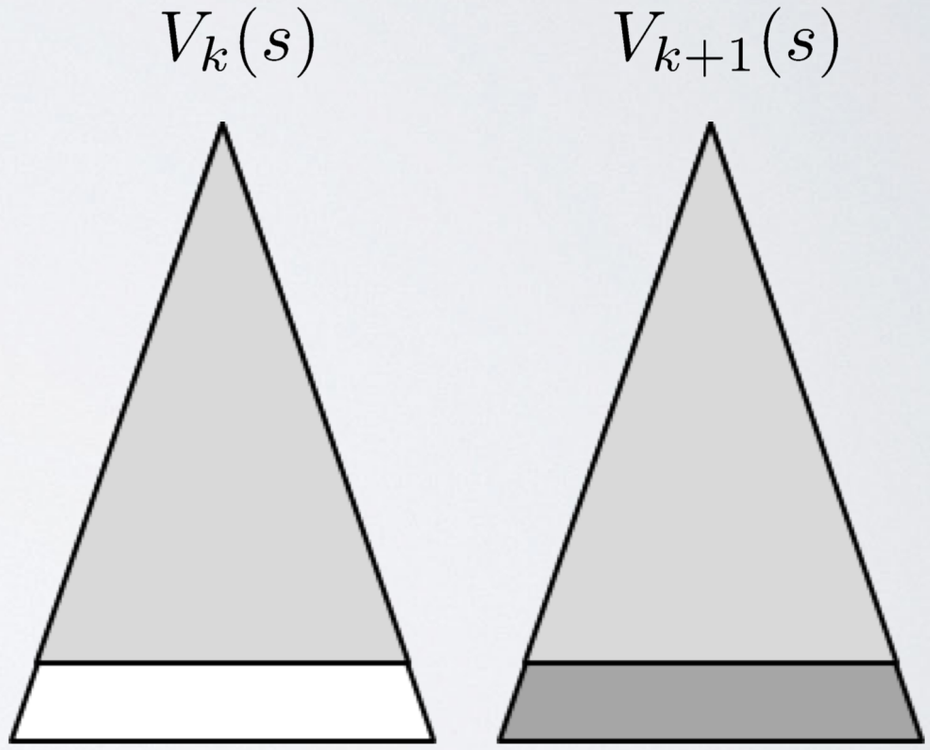


Assume no discount!

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

CONVERGENCE

- How do we know the V_k vectors are going to converge?
- Case 1: If the tree has maximum depth M, then V_M holds the actual untruncated values
- 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 value results in nearly identical search trees
- The difference is that on the bottom layer, V_{k+1} has actual rewards while V_k has zeros
- That last layer is at best all R_{MAX}
- It is at worst R_{MIN}
- But everything is discounted by γ^k that far out
- So V_k and V_{k+1} are at most γ^k max|R| different
- So as k increases, the values converge



BELLMAN EQUATIONS AND THE OPTIMAL VALUES

- Bellman equations characterize the optimal values:

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

- Value iteration computes them:

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

- Value iteration is just a fixed point solution method
  … though the V_k vectors are also interpretable as time-limited values

COMPUTING A POLICY

Suppose that we have run value iteration and now have a pretty good approximation of V*

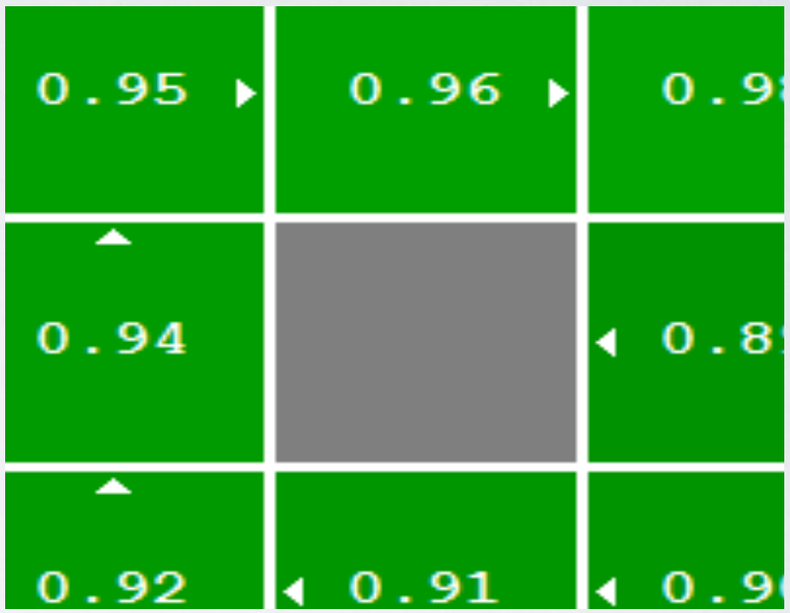
$$V^*(s) = \max_a \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V^*(s') \right]$$

How do we compute the optimal policy?

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

The optimal policy is implied by the optimal value function...

COMPUTING A POLICY



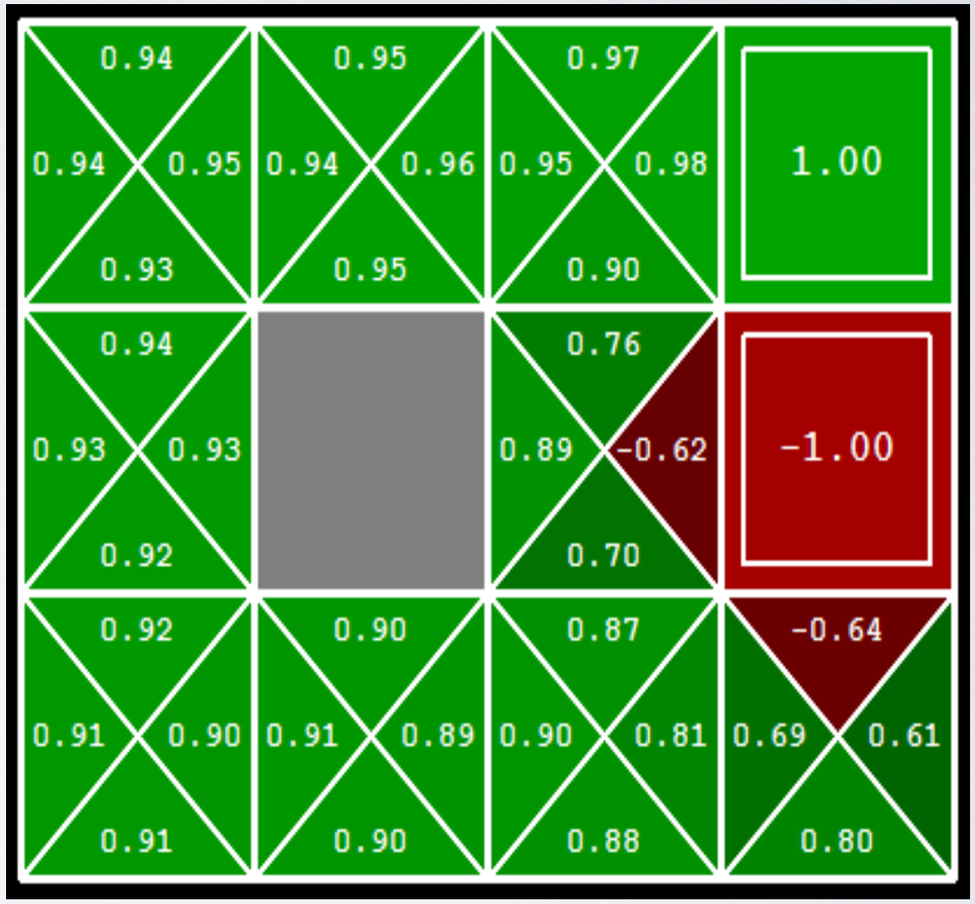
Given values calculated using value iteration, do one step of value calculation:

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

The optimal policy is implied by the optimal value function...

A POLICY FROM Q-VALUES

- Let's imagine we have the optimal q-values:



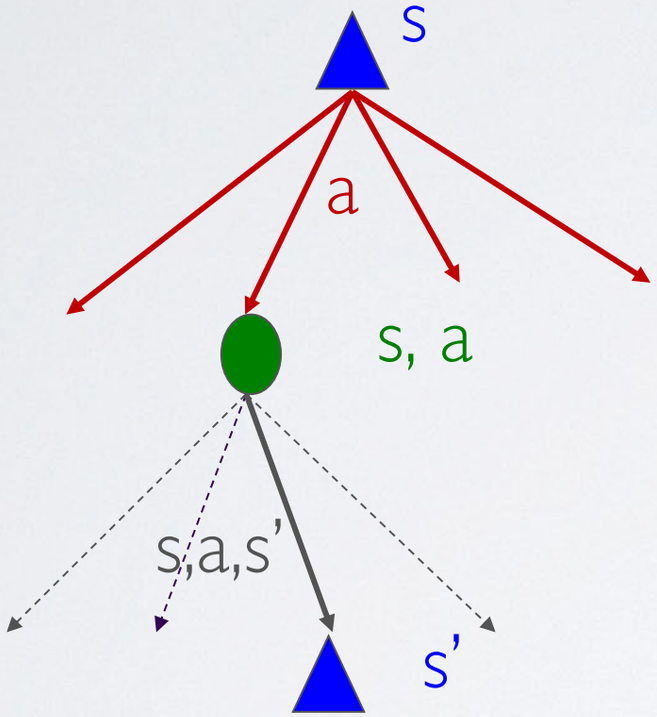
- How should we act?
- Completely trivial to decide!

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

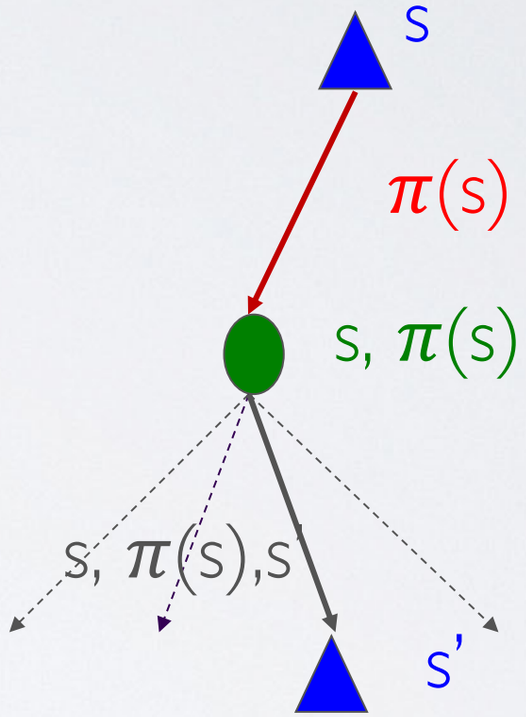
- Important lesson: actions are easier to select from q-values than values!

FIXED POLICIES

Do the optimal action



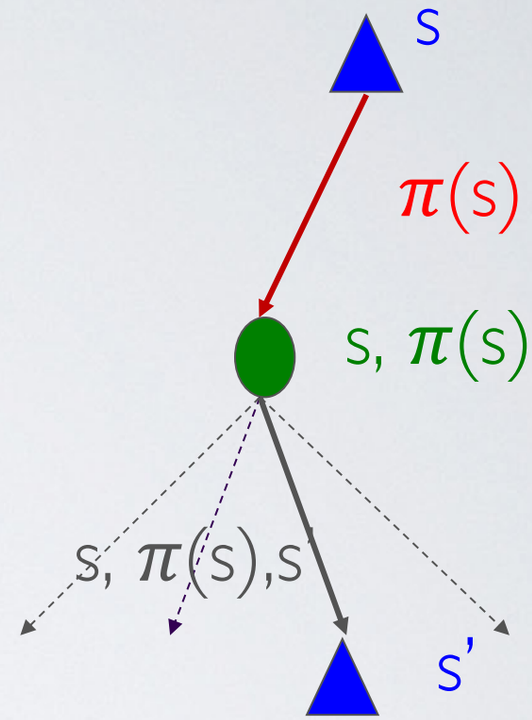
Do what π says to do



- Our evaluation trees max over all actions to compute the optimal values
- If we fixed some policy $\pi(s)$, then the tree would be simpler – only one action per state
  - … though the tree's value would depend on which policy we fixed

EVALUATING FIXED POLICIES

- Another basic operation: compute the utility of a state s under a fixed (generally non-optimal) policy



- Define the utility of a state s, under a fixed policy π:

V^π(s) = expected total discounted rewards starting in s and following π

- Recursive relation (one-step look-ahead / Bellman equation):

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

EVALUATING FIXED POLICIES

- How do we calculate the V's for a fixed policy π?
- Idea 1: Incrementally compute expected utility after *k* steps of executing π (like value iteration)

$$V_0^\pi(s) = 0$$

$$V_1^\pi(s) = R(s, \pi(s))$$

$$\dots$$

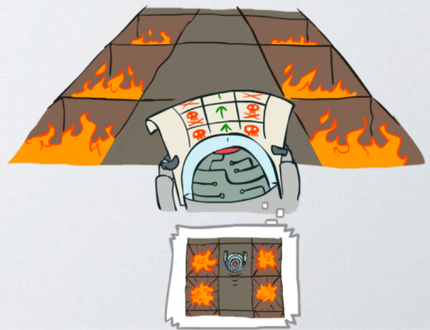
$$V_k^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s'} T(s, \pi(s), s') V_{k-1}^\pi(s)$$

- Dynamic programming as iterative evaluation
- Efficiency: $O(S^2)$ per iteration
- Idea 2: Without the maxes, the Bellman equations are just a linear system
  - Solve with Matlab (or your favorite linear system solver)

EXAMPLE: POLICY EVALUATION

Always Go Right

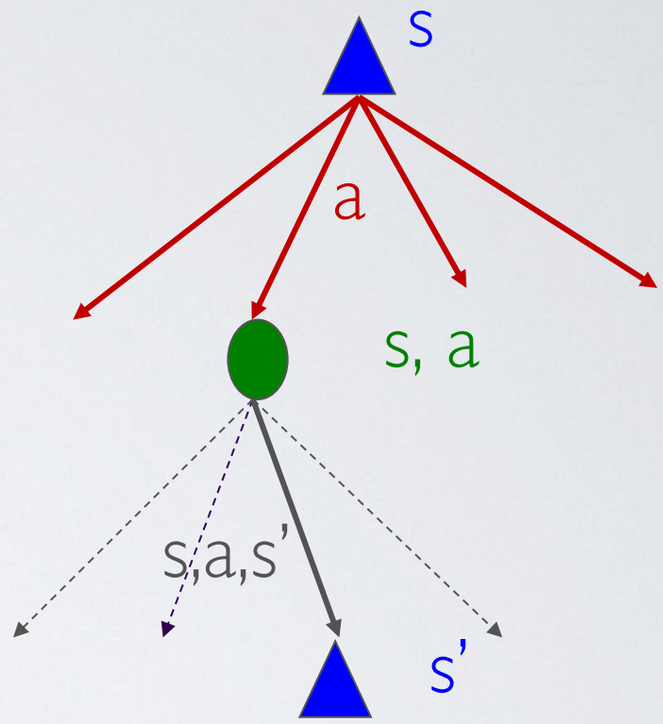
Always Go Forward



PROBLEMS WITH VALUE ITERATION

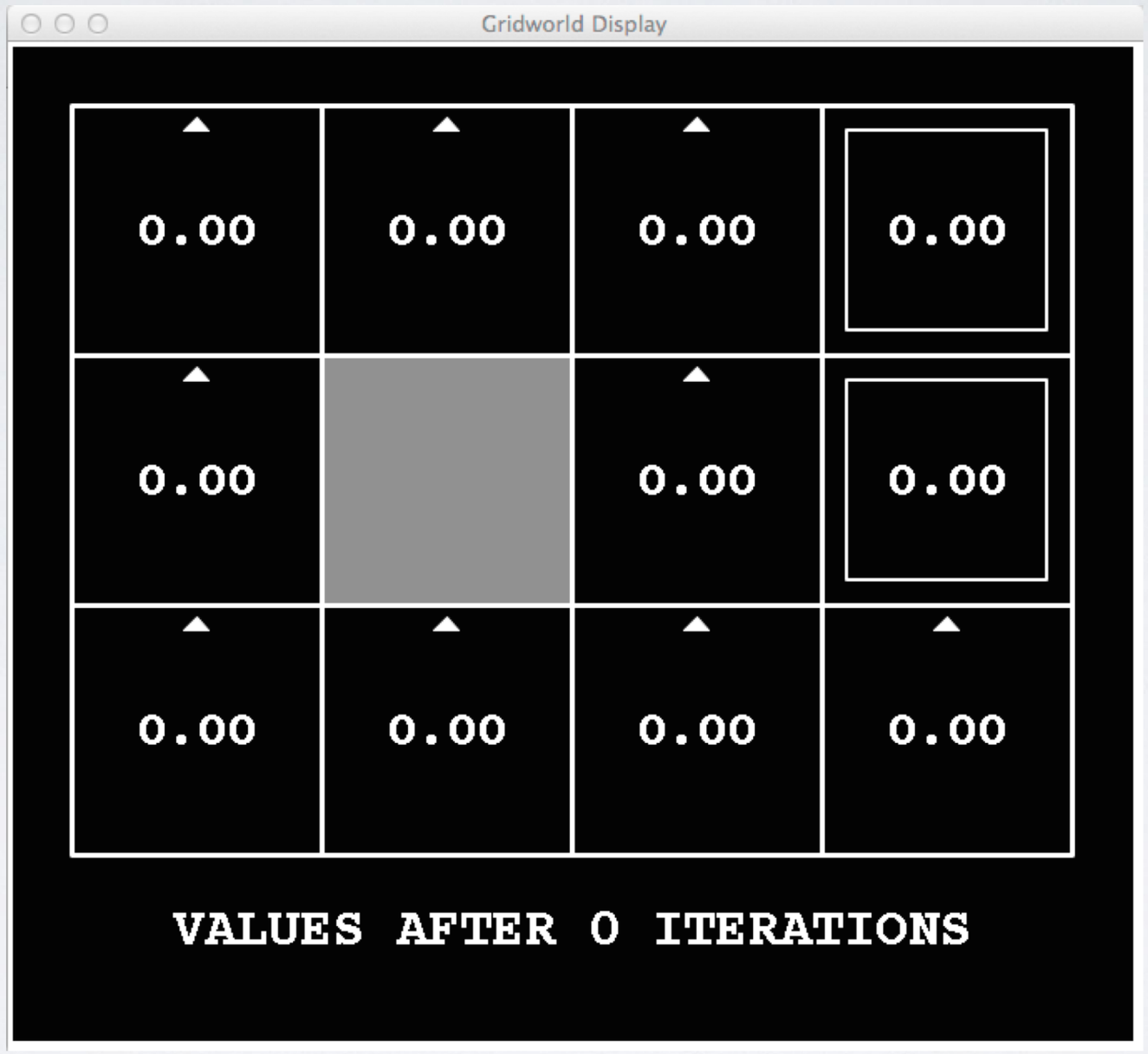
- Value iteration repeats the Bellman updates:

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



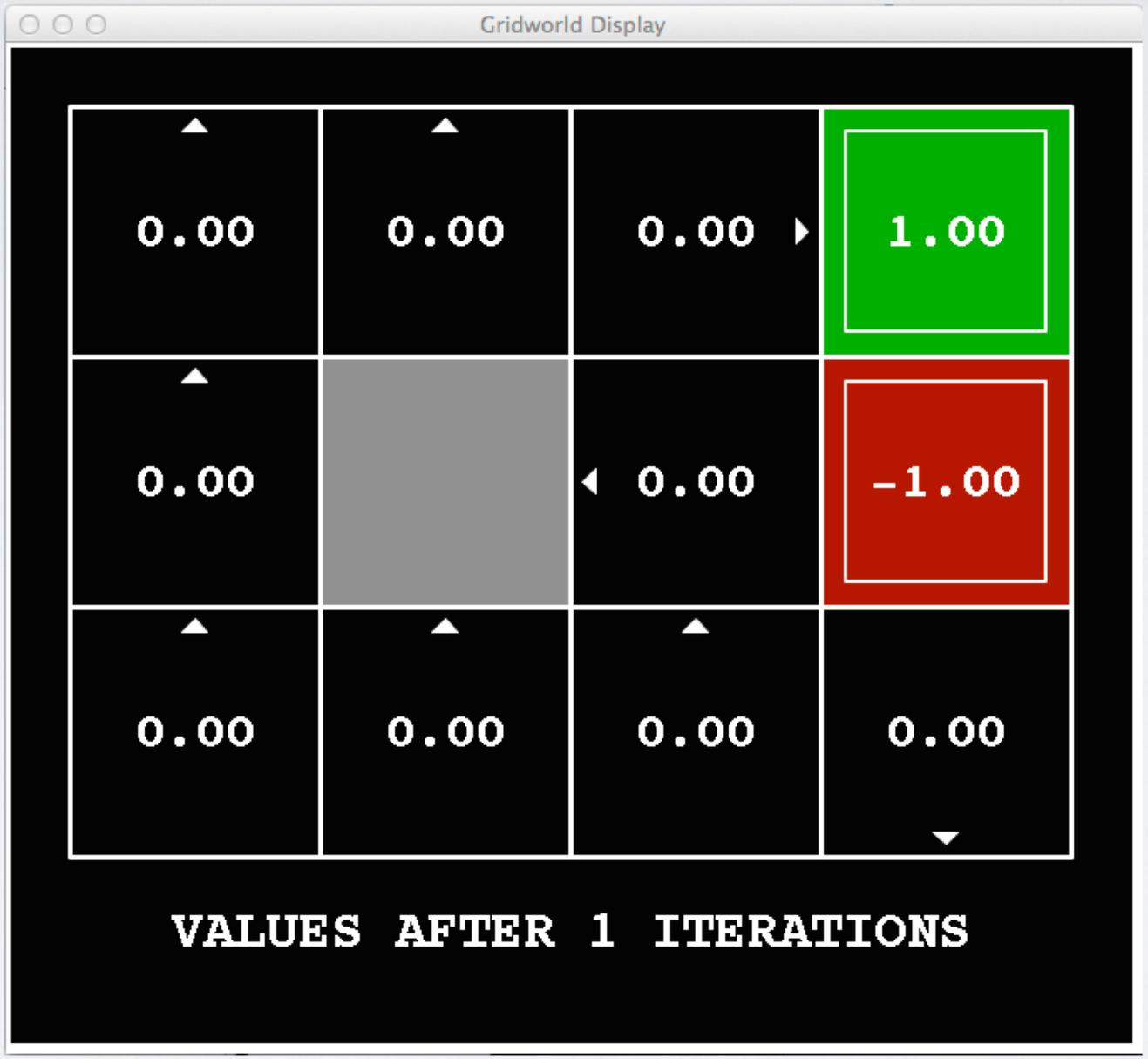
- Problem 1: It's slow – $O(S^2A)$ per iteration
- Problem 2: The "max" at each state rarely changes
- Problem 3: The policy often converges long before the values

K=0



Noise = 0.2
Discount = 0.9
Living reward = 0

K=1



Noise = 0.2
Discount = 0.9
Living reward = 0

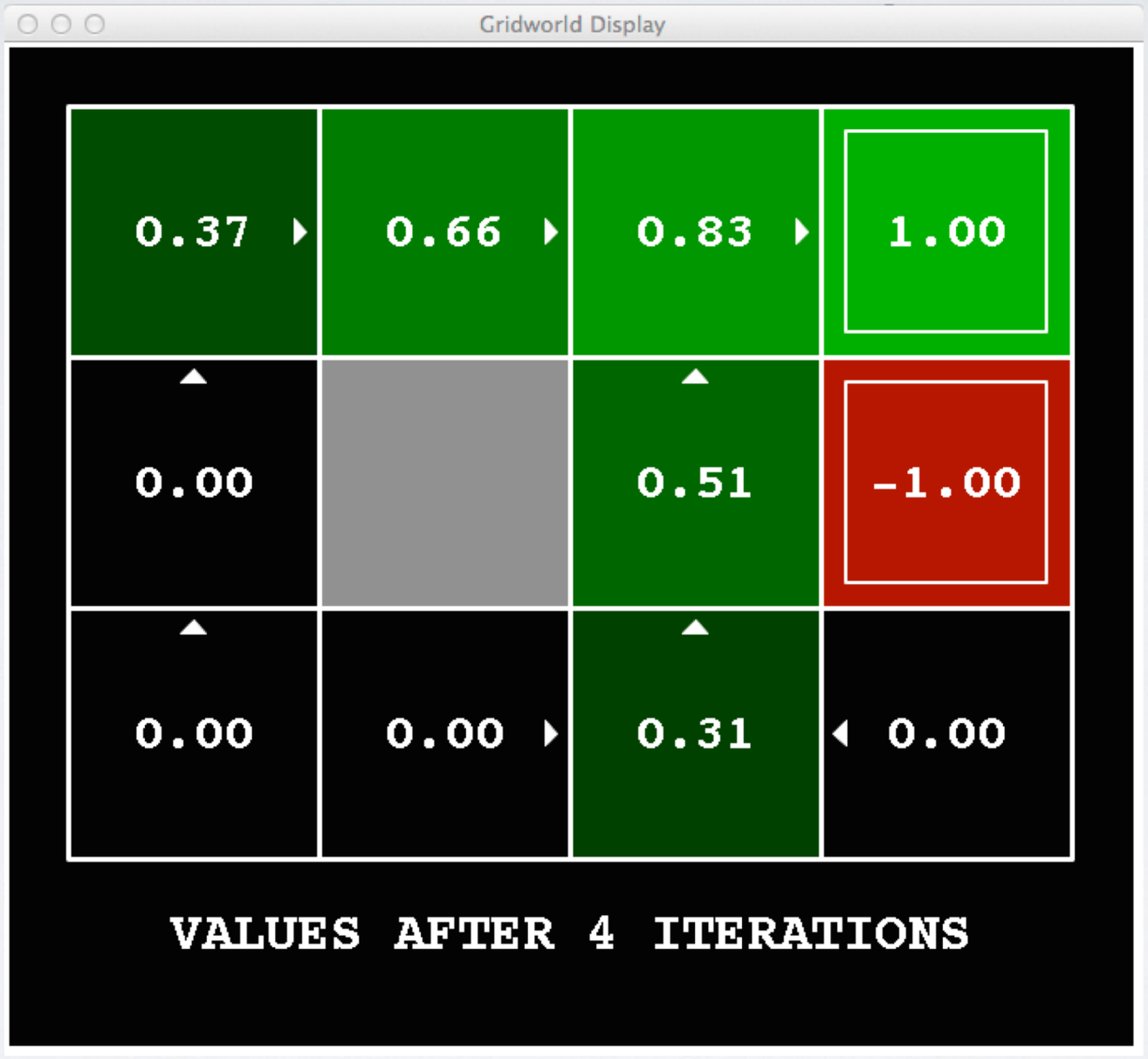
K=2

Noise = 0.2
Discount = 0.9
Living reward = 0

K=3

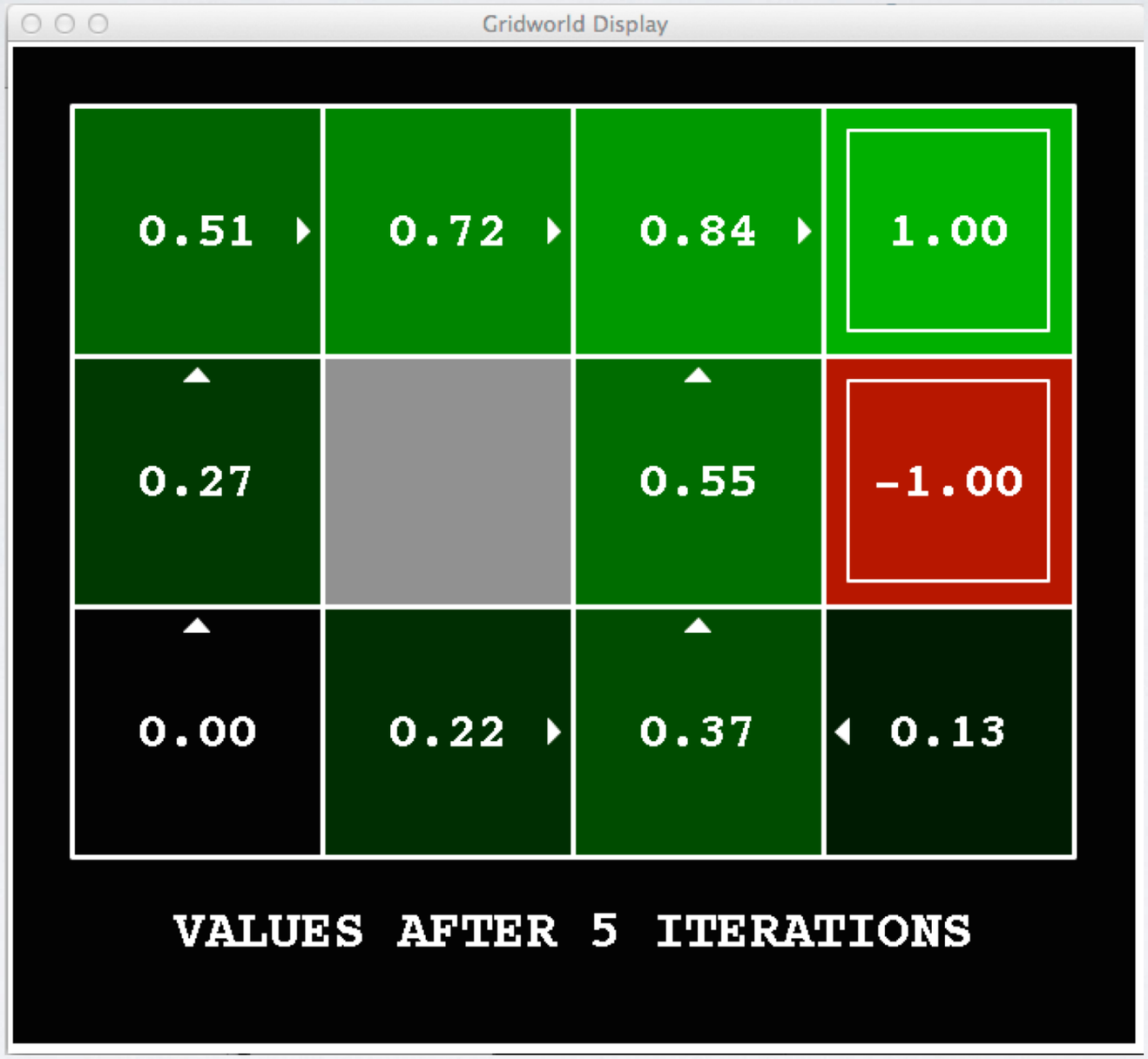
Noise = 0.2
Discount = 0.9
Living reward = 0

K=4



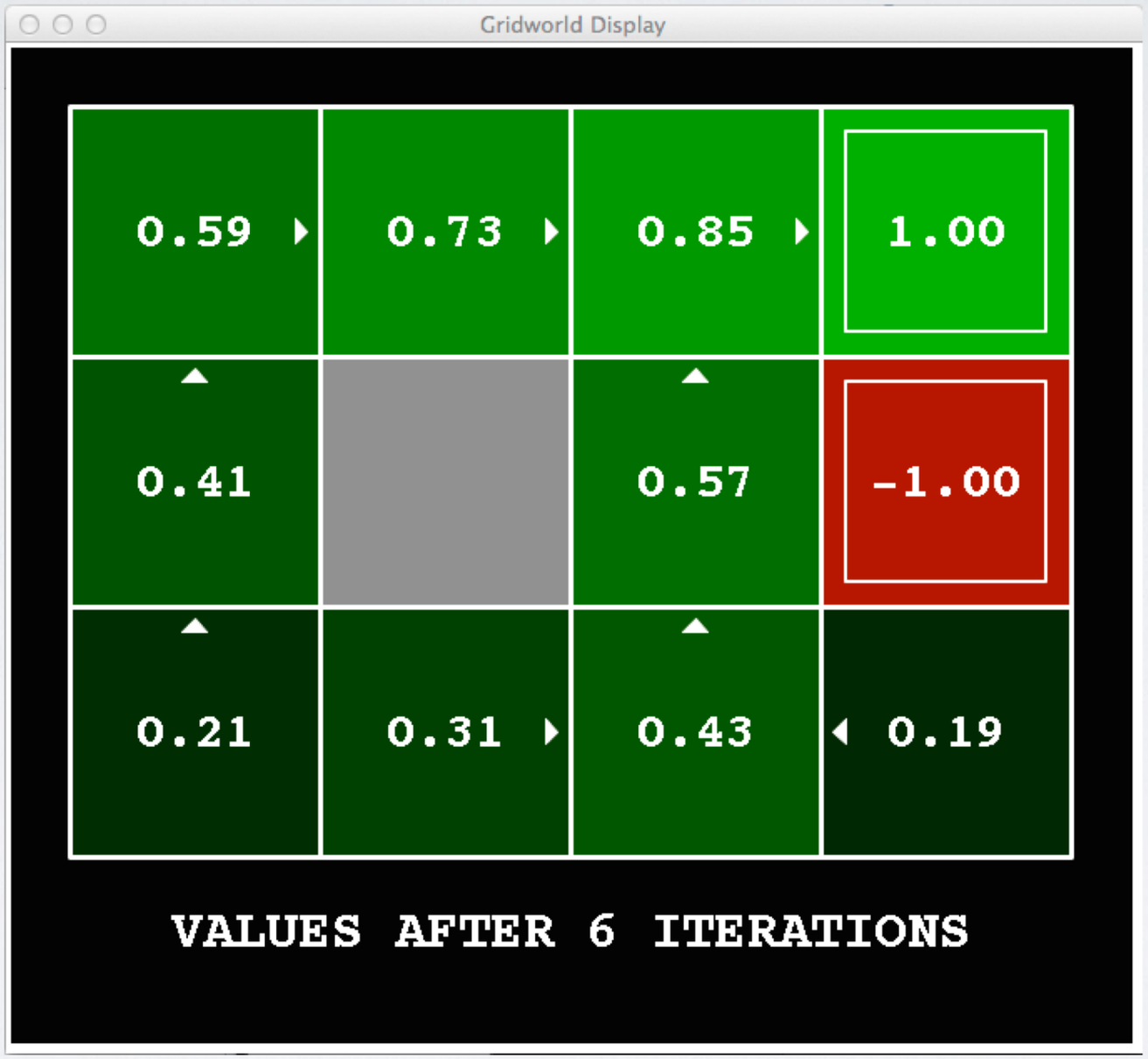
Noise = 0.2
Discount = 0.9
Living reward = 0

K=5



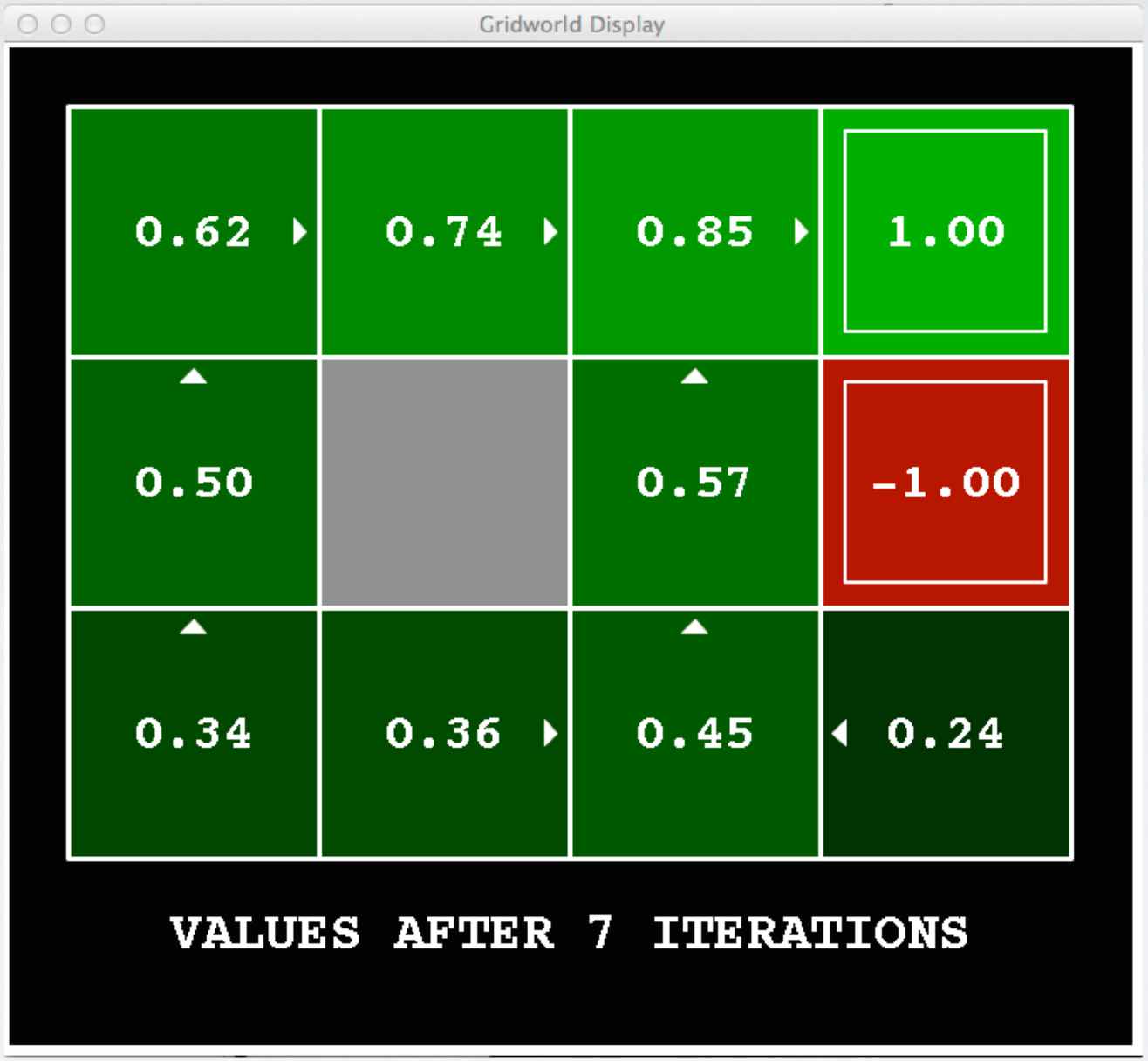
Noise = 0.2
Discount = 0.9
Living reward = 0

K=6



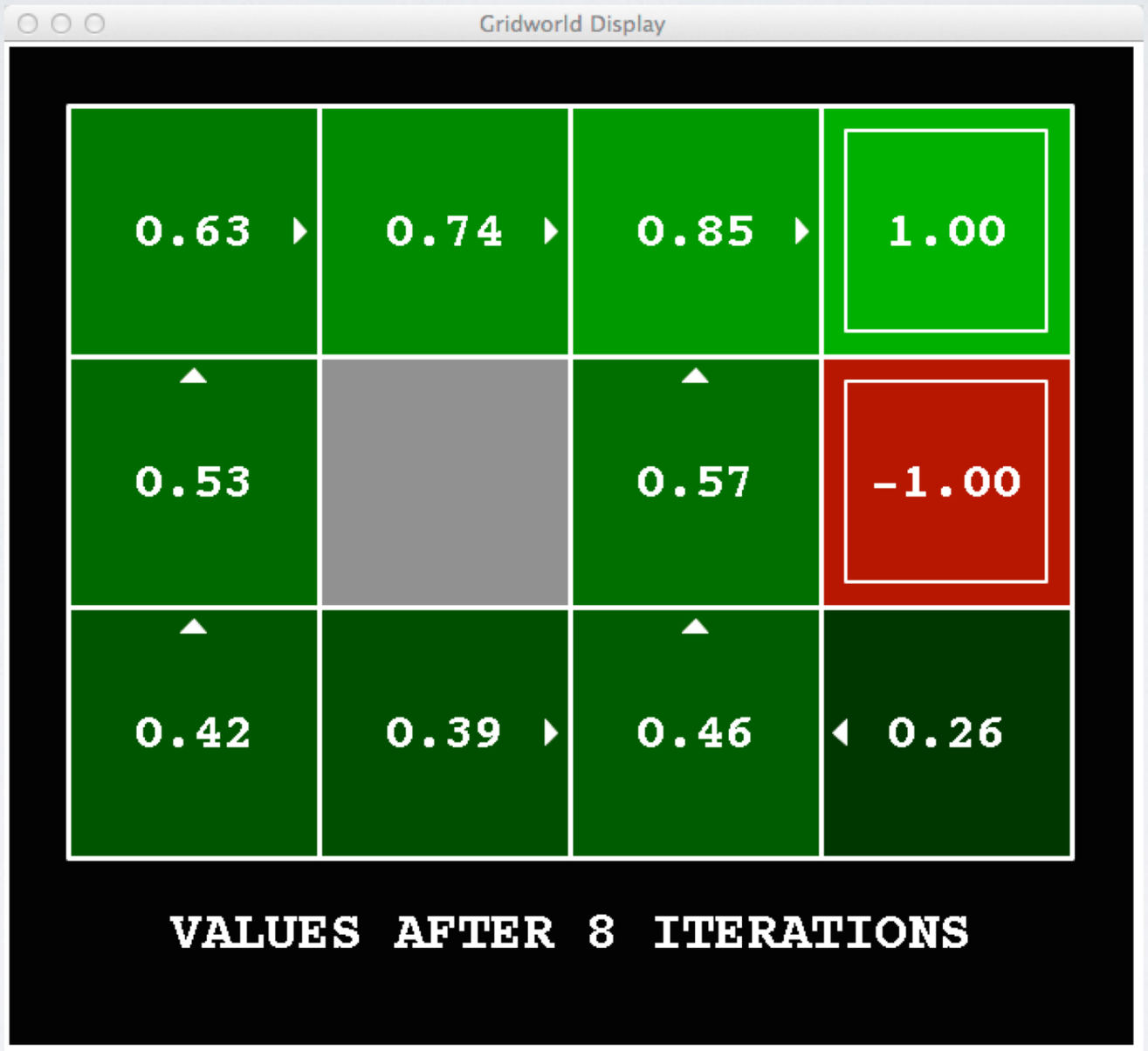
Noise = 0.2
Discount = 0.9
Living reward = 0

K=7



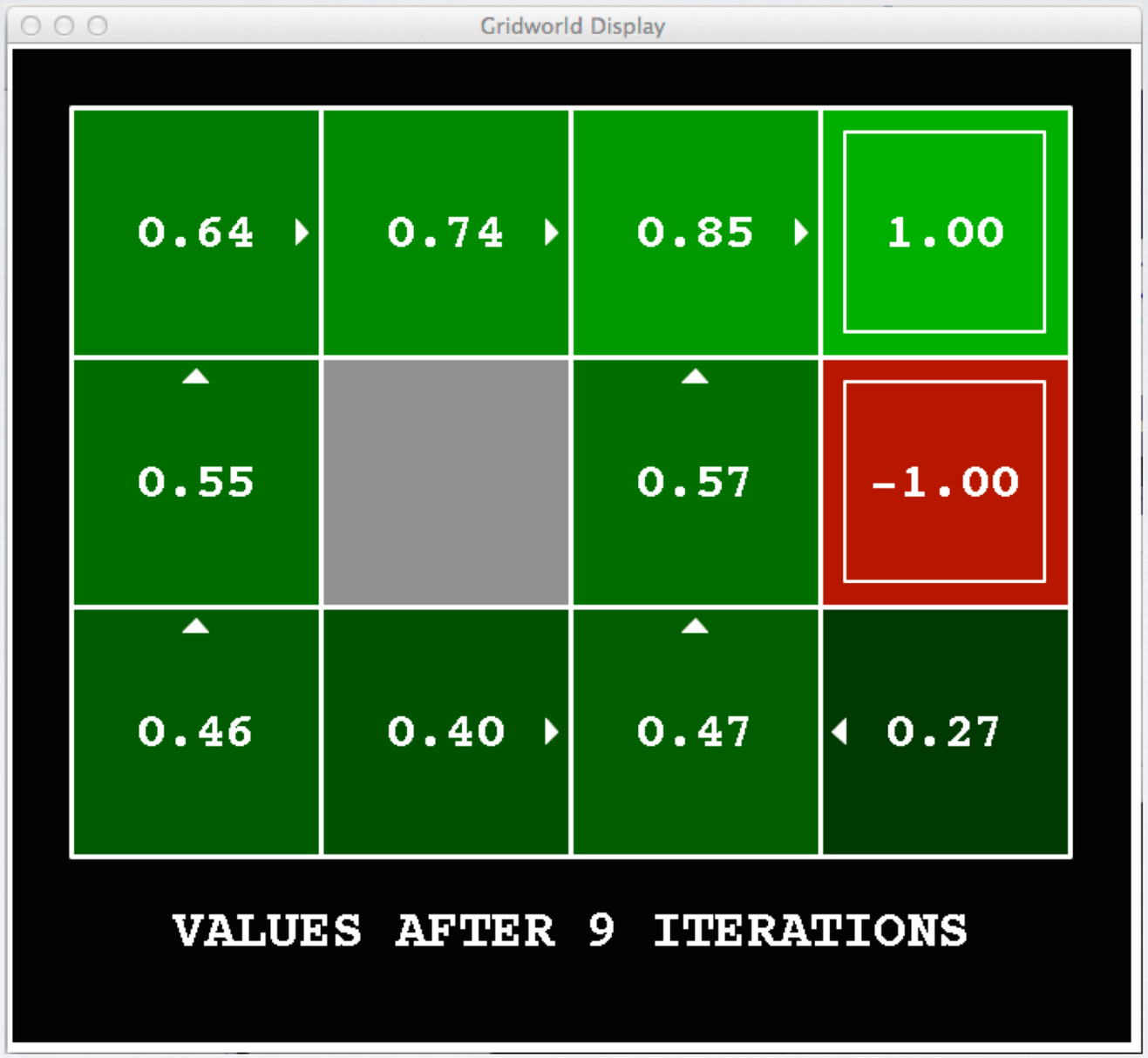
Noise = 0.2
Discount = 0.9
Living reward = 0

K=8



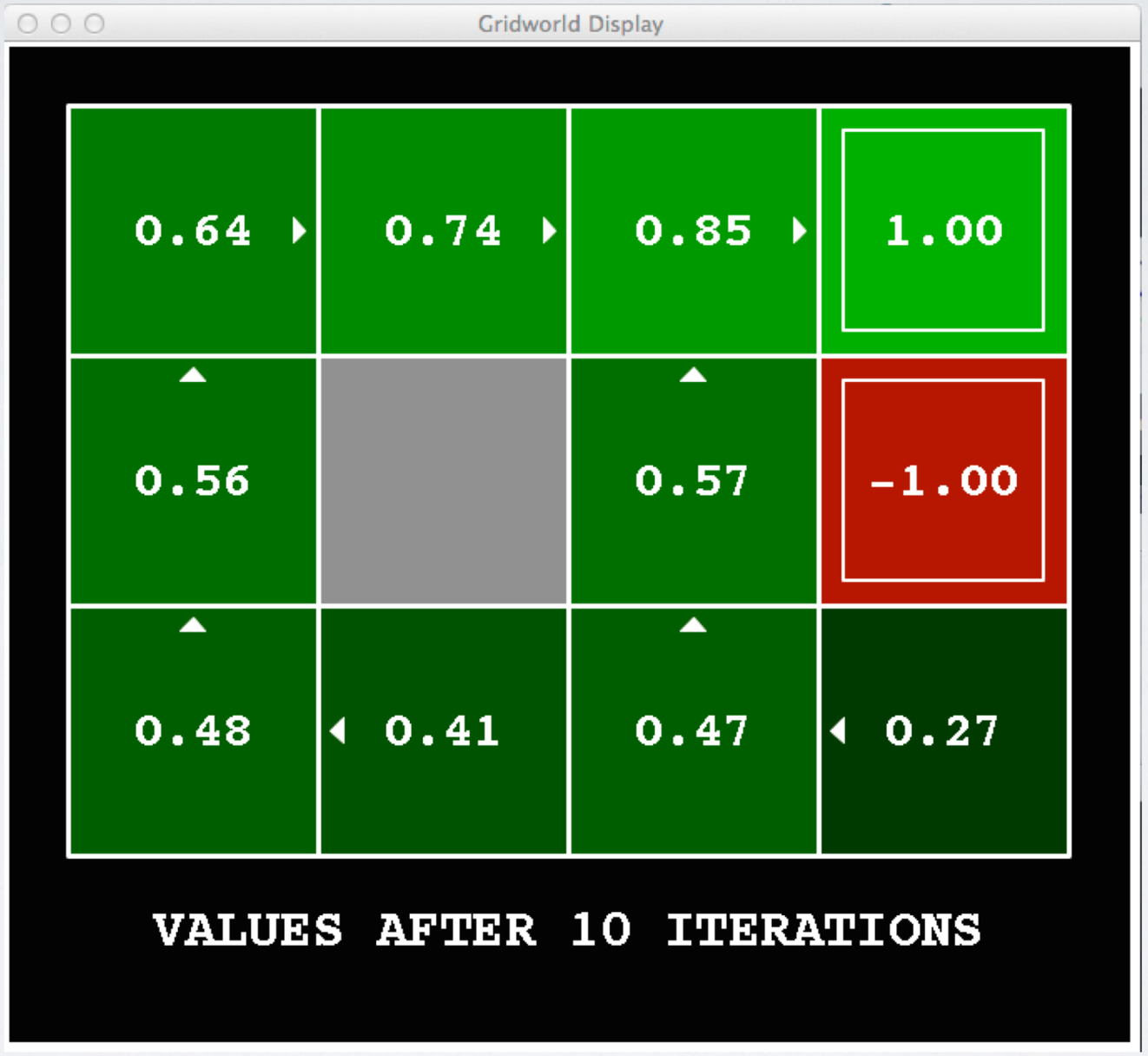
Noise = 0.2
Discount = 0.9
Living reward = 0

K=9



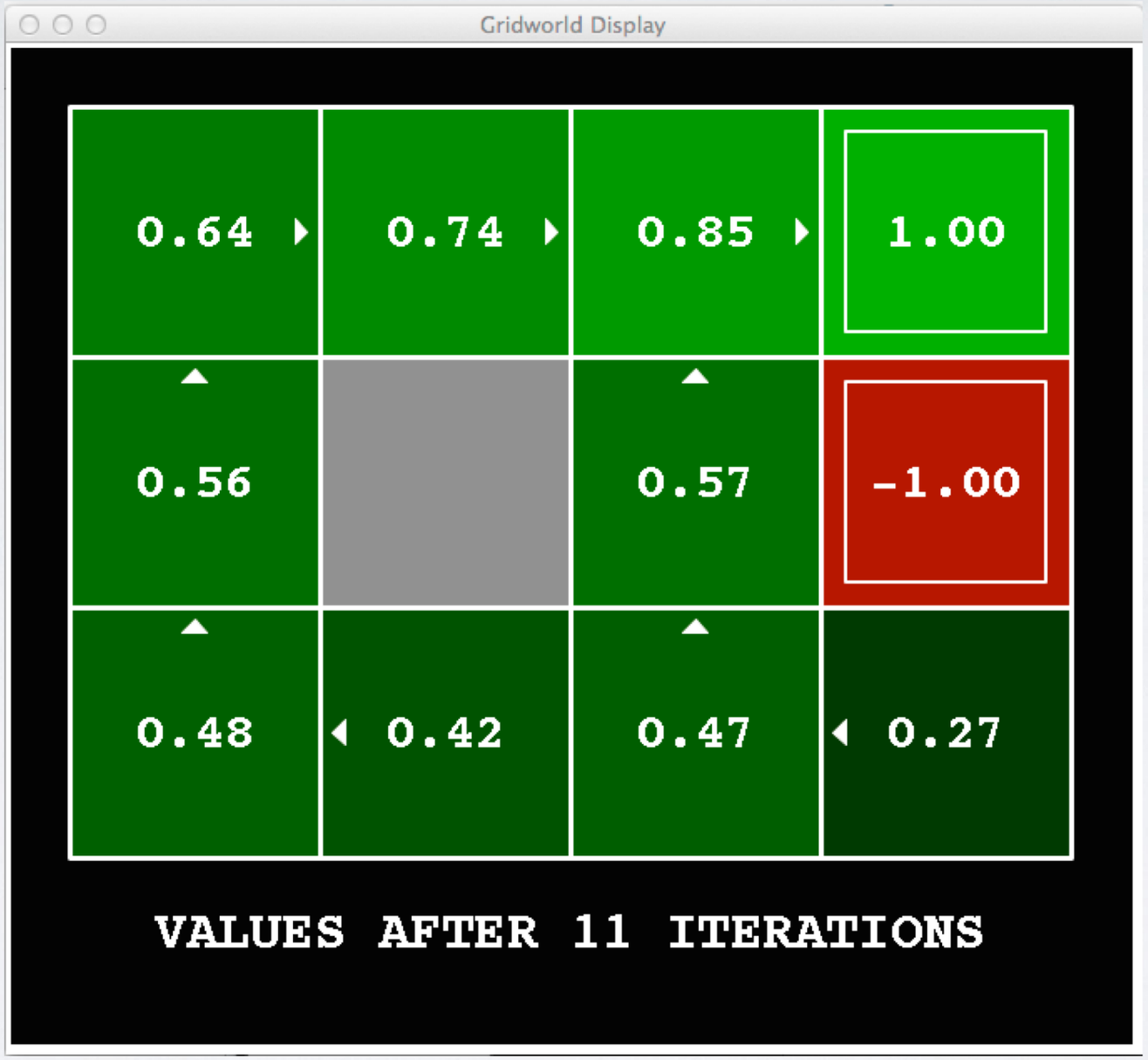
Noise = 0.2
Discount = 0.9
Living reward = 0

K=10



Noise = 0.2
Discount = 0.9
Living reward = 0

K=11

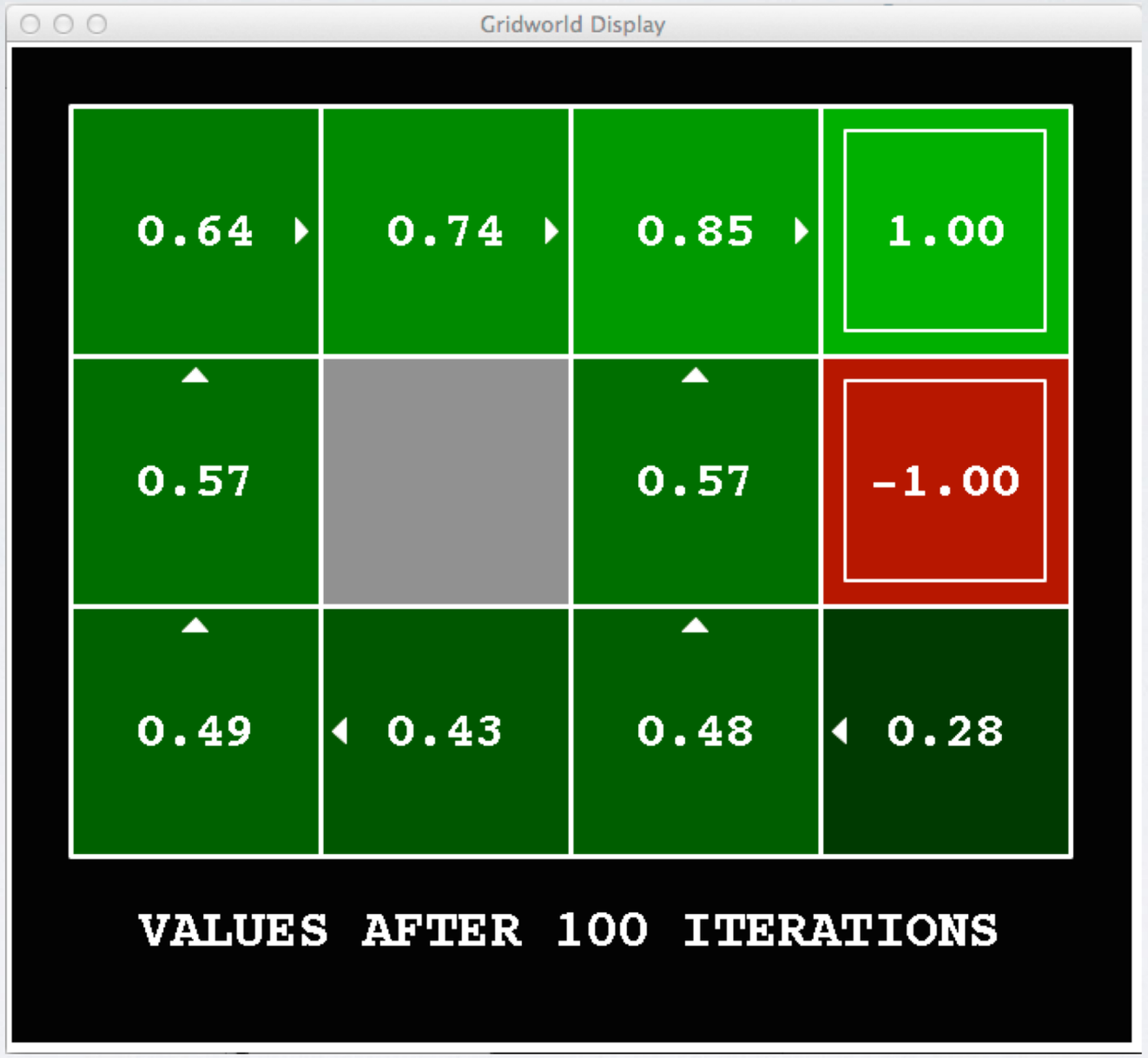


Noise = 0.2
Discount = 0.9
Living reward = 0

K=12

Noise = 0.2
Discount = 0.9
Living reward = 0

K=100



Noise = 0.2
Discount = 0.9
Living reward = 0

POLICY ITERATION

- Alternative approach for optimal values:
  - Step 1: Policy evaluation: calculate utilities for some fixed policy (not optimal utilities!) until convergence
  - Step 2: Policy improvement: update policy using one-step look-ahead with resulting converged (but not optimal!) utilities as future values
  - Repeat steps until policy converges
- This is policy iteration
  - It's still optimal!
  - Can converge (much) faster under some conditions

POLICY ITERATION

- Algorithm:
  - $\pi \leftarrow$ an arbitrary initial policy
  - repeat until no change in π
    - compute utilities given π
    - update π as if utilities were correct (i.e., local MEU)

$$argmax_a R(s,a) + \gamma \sum_{s'} T(s,\pi(s),s')V^{\pi}(s') \; = \pi(s)\,?$$

- To compute utilities given a fixed π (policy evaluation)

$$V^{\pi(s)} = R\big(s,\pi(s)\big) + \gamma \sum_{s'} T(s,\pi(s),s')V^{\pi}(s') \quad \text{for all } s$$

MODIFIED POLICY ITERATION

- Policy iteration often converges in few iterations, but each is expensive
- Idea: use a few steps of value iteration (but with π fixed) starting from the value function produced the last time to produce an approximate value determination step.
- Often converges much faster than pure VI or PI
- Leads to much more general algorithms where Bellman value updates and Howard policy updates can be performed locally in any order
- Reinforcement learning algorithms operate by performing such updates based on the observed transitions made in an initially unknown environment

NEXT TIME

- More solving of MDPs!