Markov Decision Processes (MDPs)

Russell and Norvig: Chapter 17.1-17.3, 21
CSE 240: Winter 2023
Lecture 15

Announcements

- Assignment 4 is posted
- Canvas Quiz Feedback not Showing?
- We will *not* have class on Tuesday March 7.
- I will go over survey feedback on Thursday.

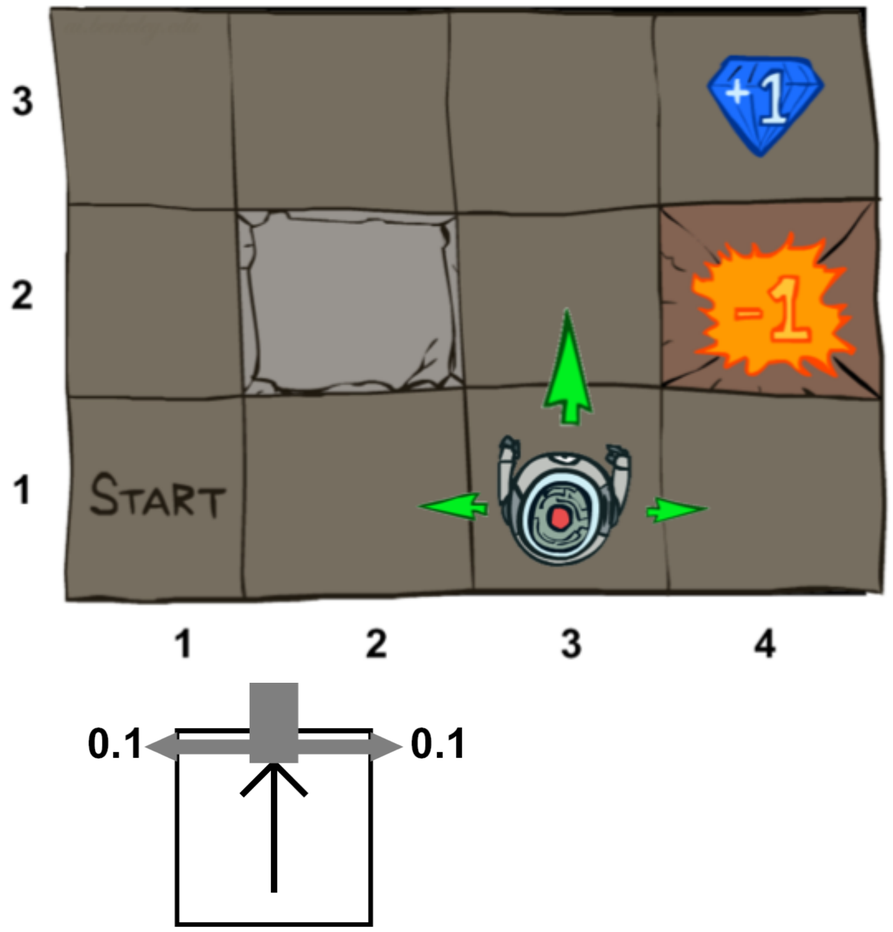
Agenda and Topics

- Assignment 4 Review and Overview
- Markov Decision Processes (MDP)
    - Discounting
    - Value Iteration
    - Bellman Equation

Assignment 4 Overview

Markov Decision Processes

- MDPs are a family of non-deterministic search problems
- An MDP is defined by:
  - A set of states $s \in S$
  - A set of actions $a \in A$
  - A transition function T(s,a,s')
    - Prob that a from s leads to s'
    - i.e., P(s' | s,a)
    - Also called the model
  - A reward function R(s, a, s')
    - Sometimes just R(s) or R(s')
  - A start state (or distribution)
  - Maybe a terminal state



What is Markov about MDPs?

- Andrey Markov (1856-1922)
- “Markov” generally means that given the present state, the future and the past are independent
- For Markov decision processes, “Markov” means:

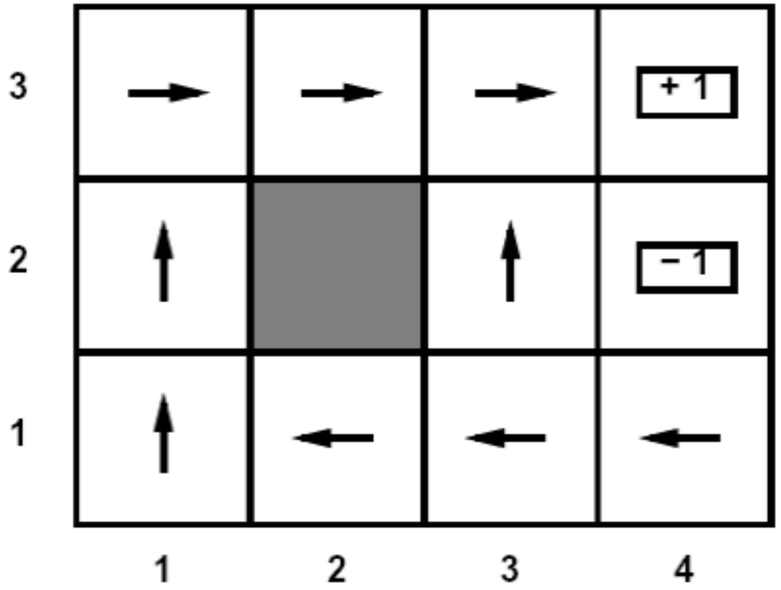
$$P(S_{t+1} = s' \mid S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1} = a_{t-1}, \ldots, S_0 = s_0, A_0 = a_0)$$
$$= P(S_{t+1} = s' | S_t = s_t, A_t = a_t)$$

Aside often use following shorthand:

$$P(s' \mid s_t, a_t, s_{t-1}, a_{t-1}, \ldots, s_0, a_0) = P(s' \mid s_t, a_t)$$

Solving MDPs

- In deterministic single-agent search problems, want an optimal plan, or sequence of actions, from start to a goal
- In an MDP, we want an optimal policy $\pi^*: S \rightarrow A$
  - A policy π gives an action for each state
  - An optimal policy maximizes expected utility if
  - An explicit policy defines a reflex agent



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

Utilities of Sequences

Utilities of Sequences

- What preferences should an agent have over reward sequences?
- More or less? [1, 2, 2] or [2, 3, 4]
- Now or later? [0, 0, 1] or [1, 0, 0]

Discounting

- It’s reasonable to maximize the sum of rewards
- It’s also reasonable to prefer rewards now to rewards later
- One solution: values of rewards decay exponentially

1

Worth Now

γ

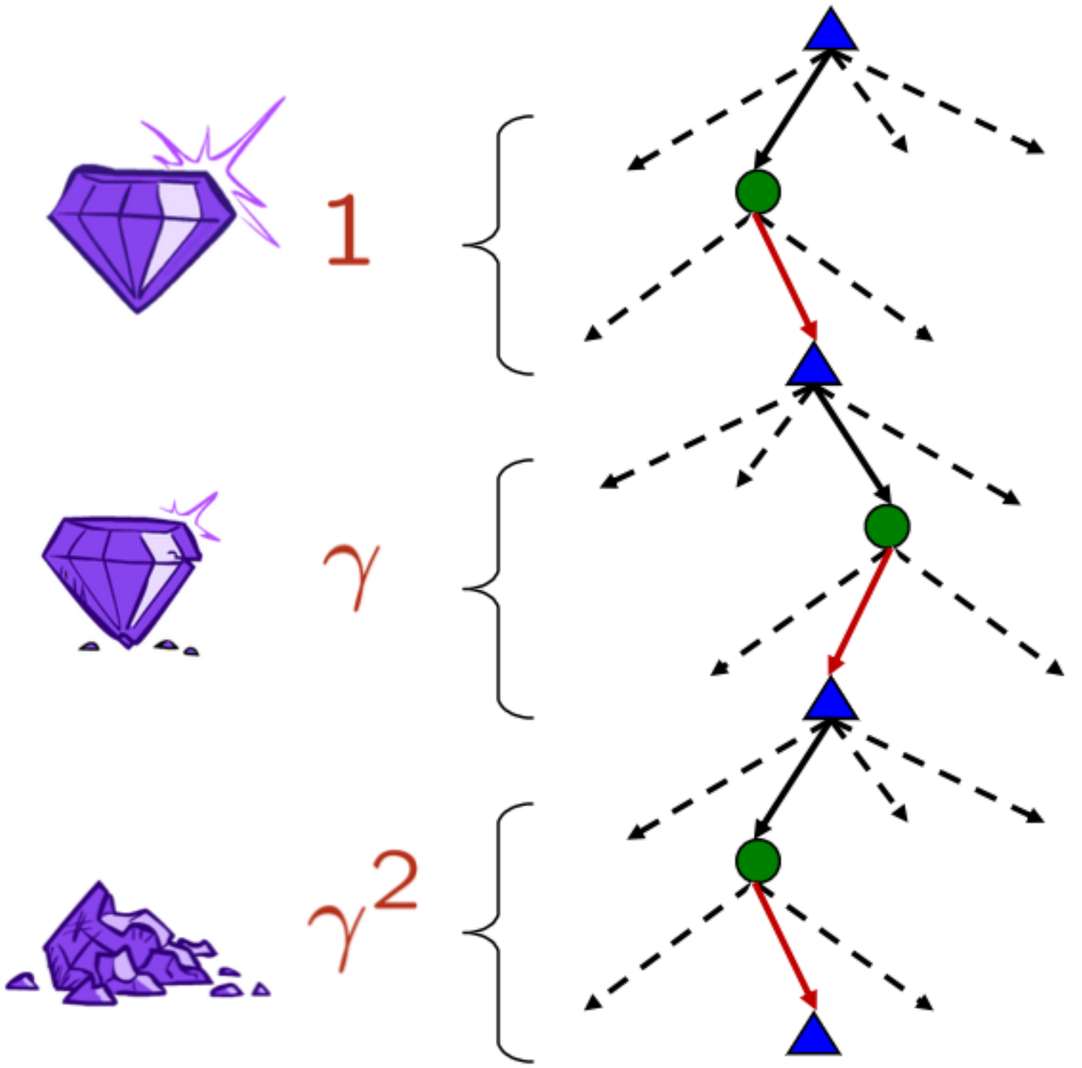
Worth Next Step

γ^2

Worth In Two Steps

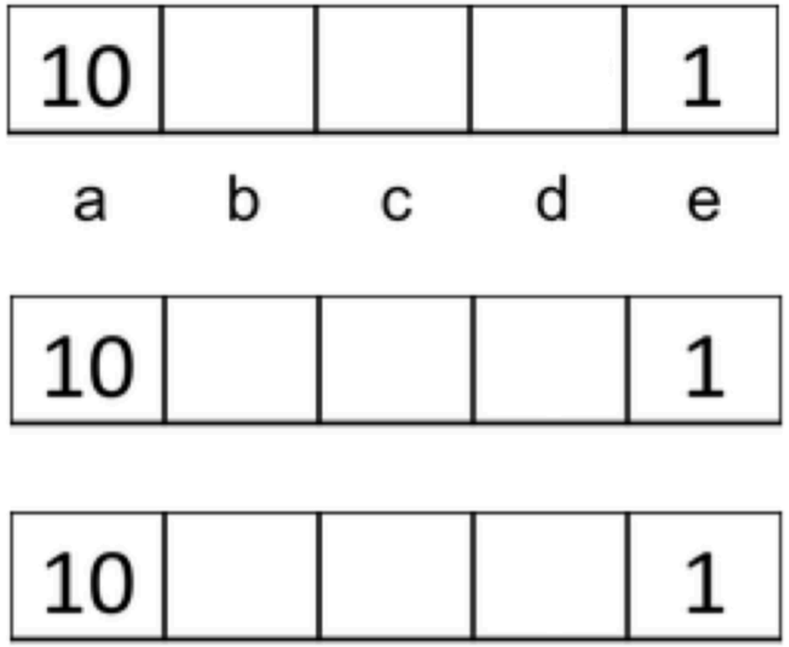
Discounting

- How to discount?
  - Each time we descend a level, we multiply in the discount once
- Why discount?
  - Think of it as a gamma chance of ending the process at every step
  - 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])



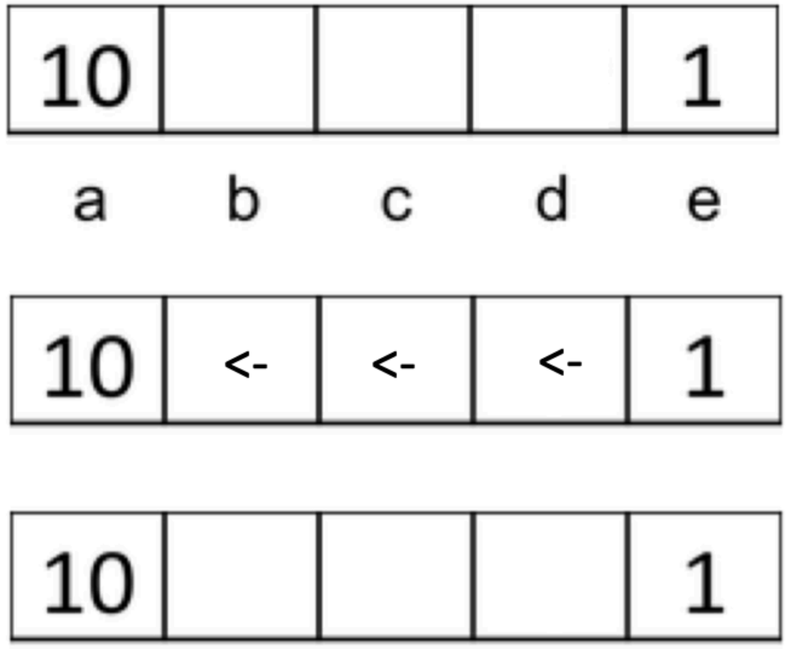
Discounting

- Actions: East, West, and Exit (only available in exit states are)
- Transitions: deterministic
- Question 1: For $\gamma = 1$, what is the optimal policy?
- Question 2: For $\gamma = 0.1$, what is the optimal policy?
- Question 3: For which γ are West and Easy equally good when in state d?



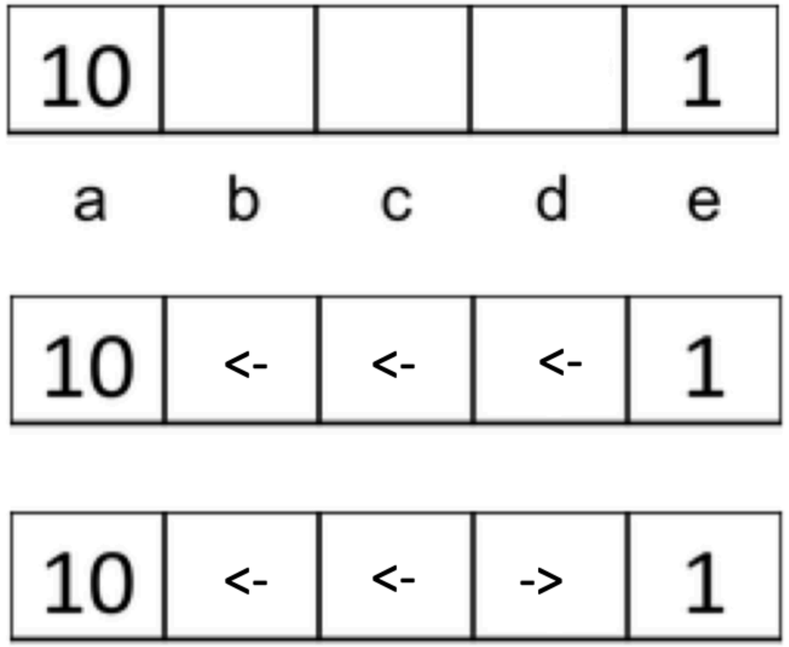
Discounting

- Actions: East, West, and Exit (only available in exit states are)
- Transitions: deterministic
- Question 1: For $\gamma = 1$, what is the optimal policy?
- Question 2: For $\gamma = 0.1$, what is the optimal policy?
- Question 3: For which γ are West and Easy equally good when in state d?



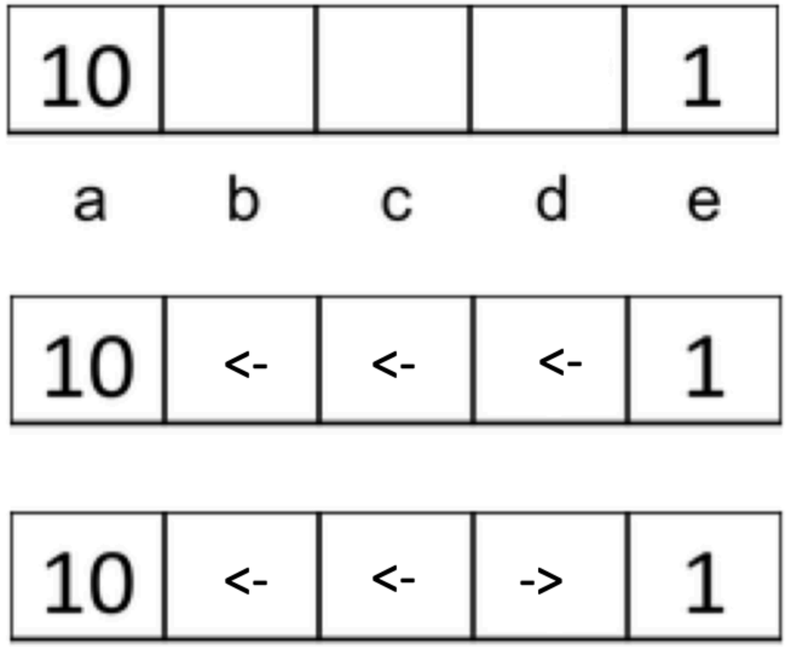
Discounting

- Actions: East, West, and Exit (only available in exit states are)
- Transitions: deterministic
- Question 1: For $\gamma = 1$, what is the optimal policy?
- Question 2: For $\gamma = 0.1$, what is the optimal policy?
- Question 3: For which γ are West and Easy equally good when in state d?



CE 15: Discounting Q3

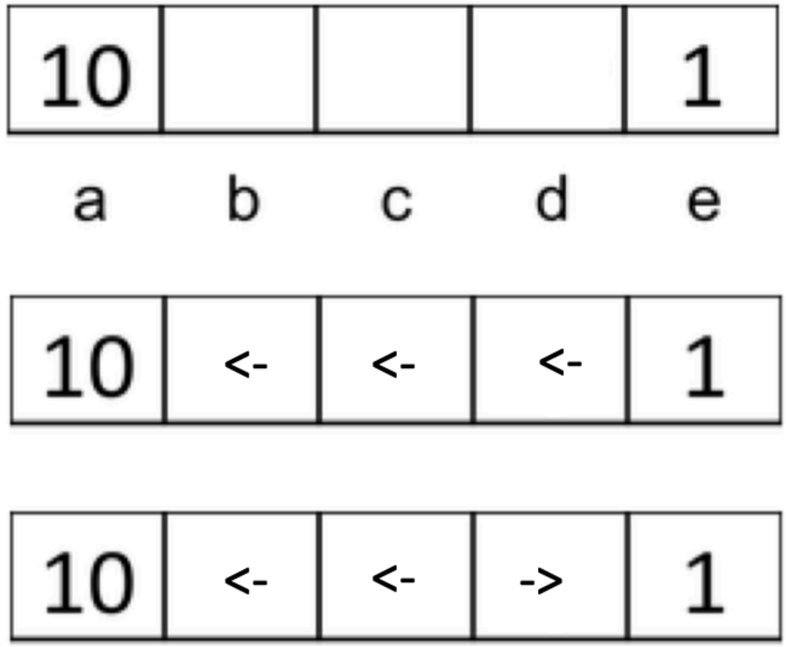
- Actions: East, West, and Exit (only available in exit states are)
- Transitions: deterministic
- Question 1: For $\gamma = 1$, what is the optimal policy?
- Question 2: For $\gamma = 0.1$, what is the optimal policy?
- **Question 3: For which γ are West and Easy equally good when in state d?**



CE 15: Discounting Q3

- Actions: East, West, and Exit (only available in exit states are)
- Transitions: deterministic
- Question 1: For $\gamma = 1$, what is the optimal policy?
- Question 2: For $\gamma = 0.1$, what is the optimal policy?
- **Question 3: For which γ are West and Easy equally good when in state d?**

$$1\gamma = 10\gamma^3$$



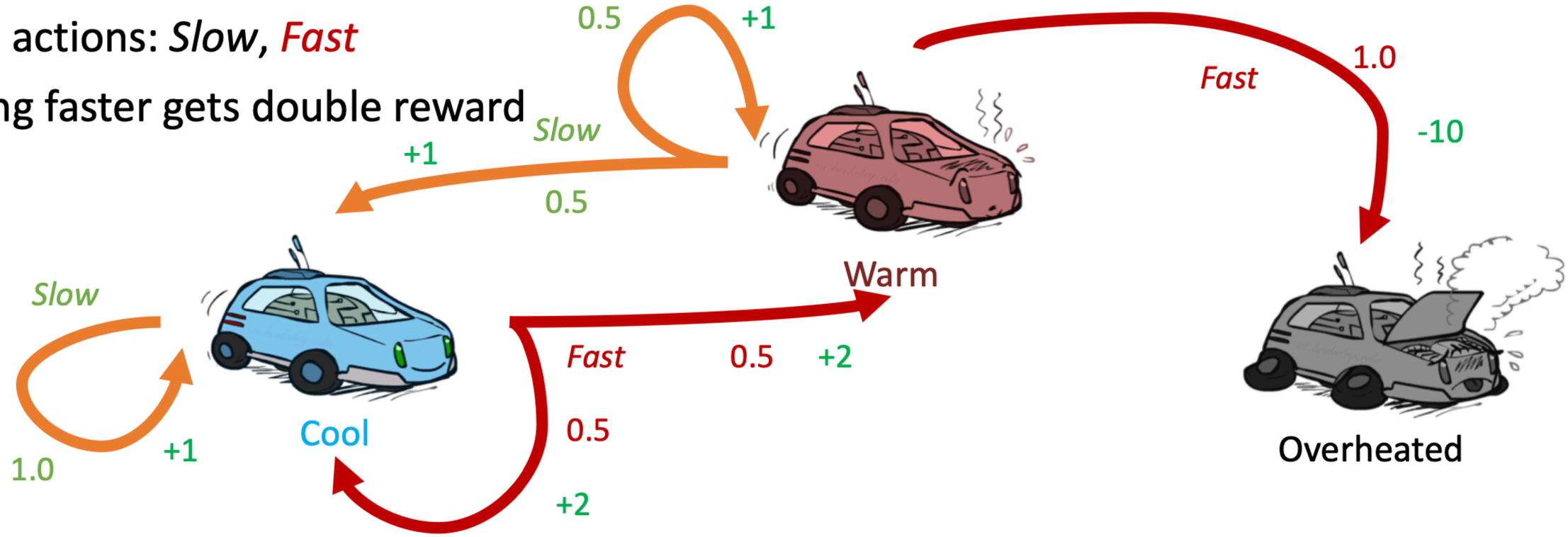
Infinite Utilities?!

- Problem: What if the game lasts forever? Do we get infinite rewards?
- Solution
  - Finite horizon (similar to depth-limited search)
    - Terminate episodes after fixed T steps (e.g., life)
    - Gives non stationary polices (π depends on time left)
  - Discounting: use $0 < \gamma < 1$
    - $U([r_0, \ldots r_\infty]) = \Sigma_{t=0}^{\infty} \gamma^t r_t \leq R_{max}/(1-\gamma)$
    - Smaller γ means smaller “horizon” - shorter term focus
  - Absorbing state: Guarantee that for every policy, a terminal state will eventually be reached (like “overheated” for racing)

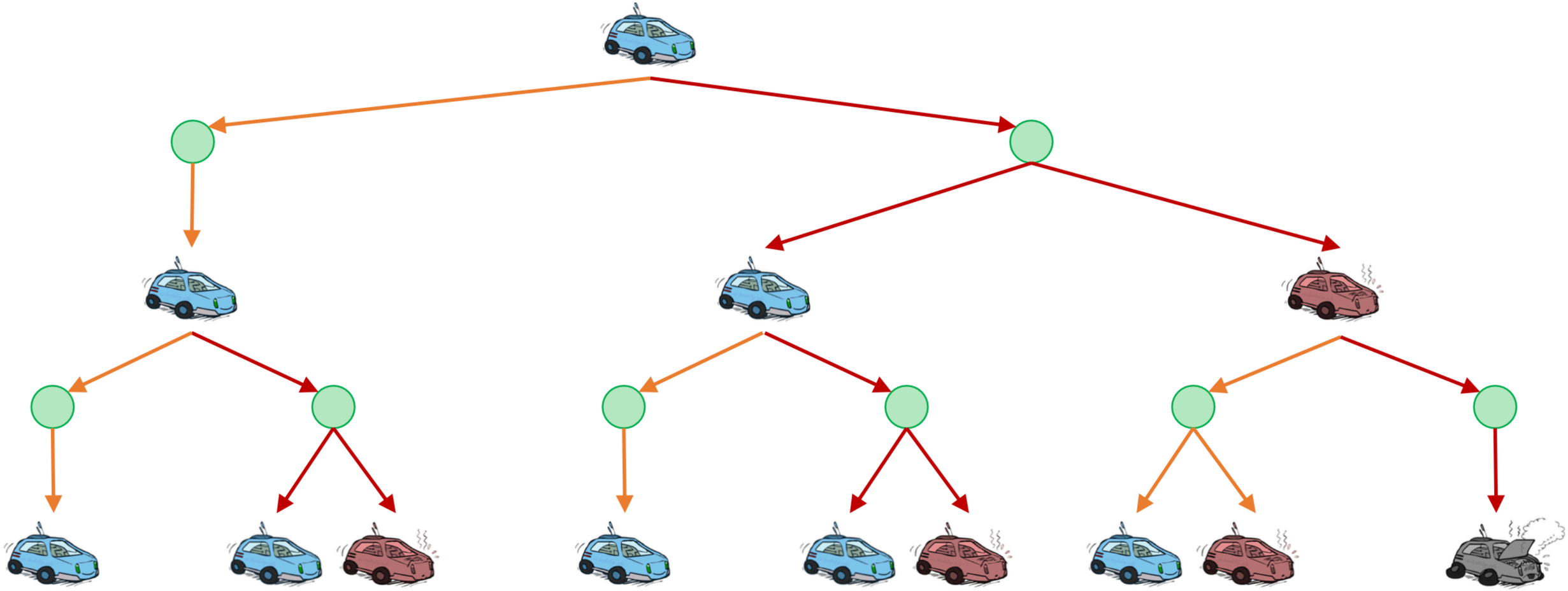
Example: Racing

Example: Racing

- A robot car wants to travel far, quickly
- Three states: Cool, Warm, Overheated
- Two actions: *Slow*, *Fast*
- Going faster gets double reward

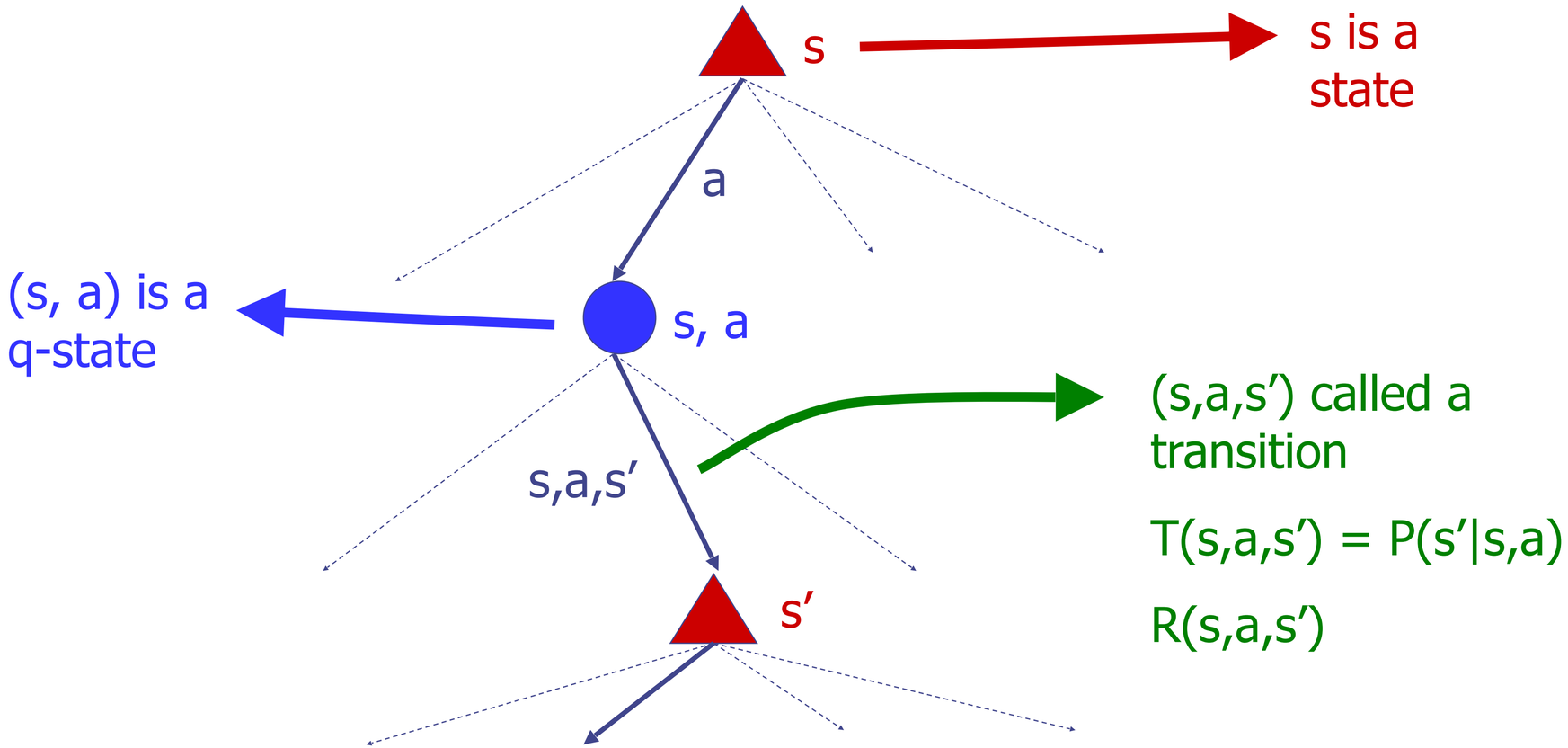


Racing Search Tree



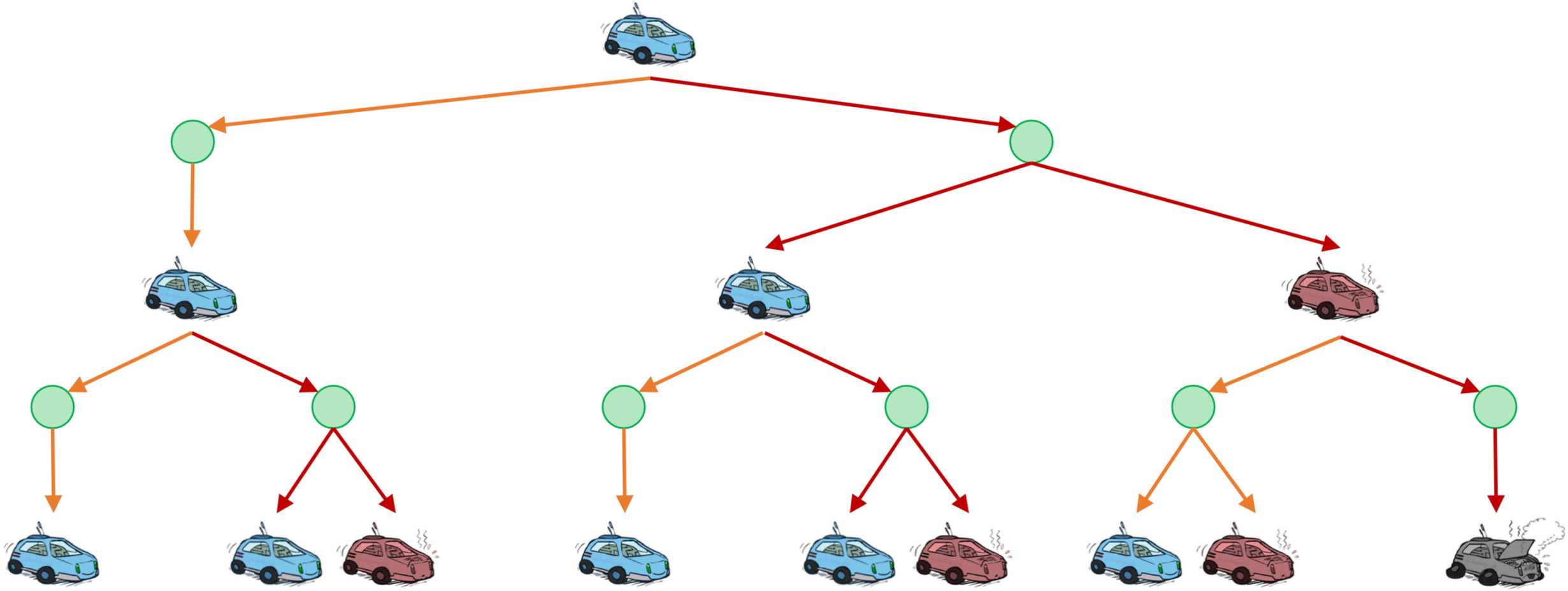
MDP Search Trees

- Each MDP state gives an expectimax-like search tree

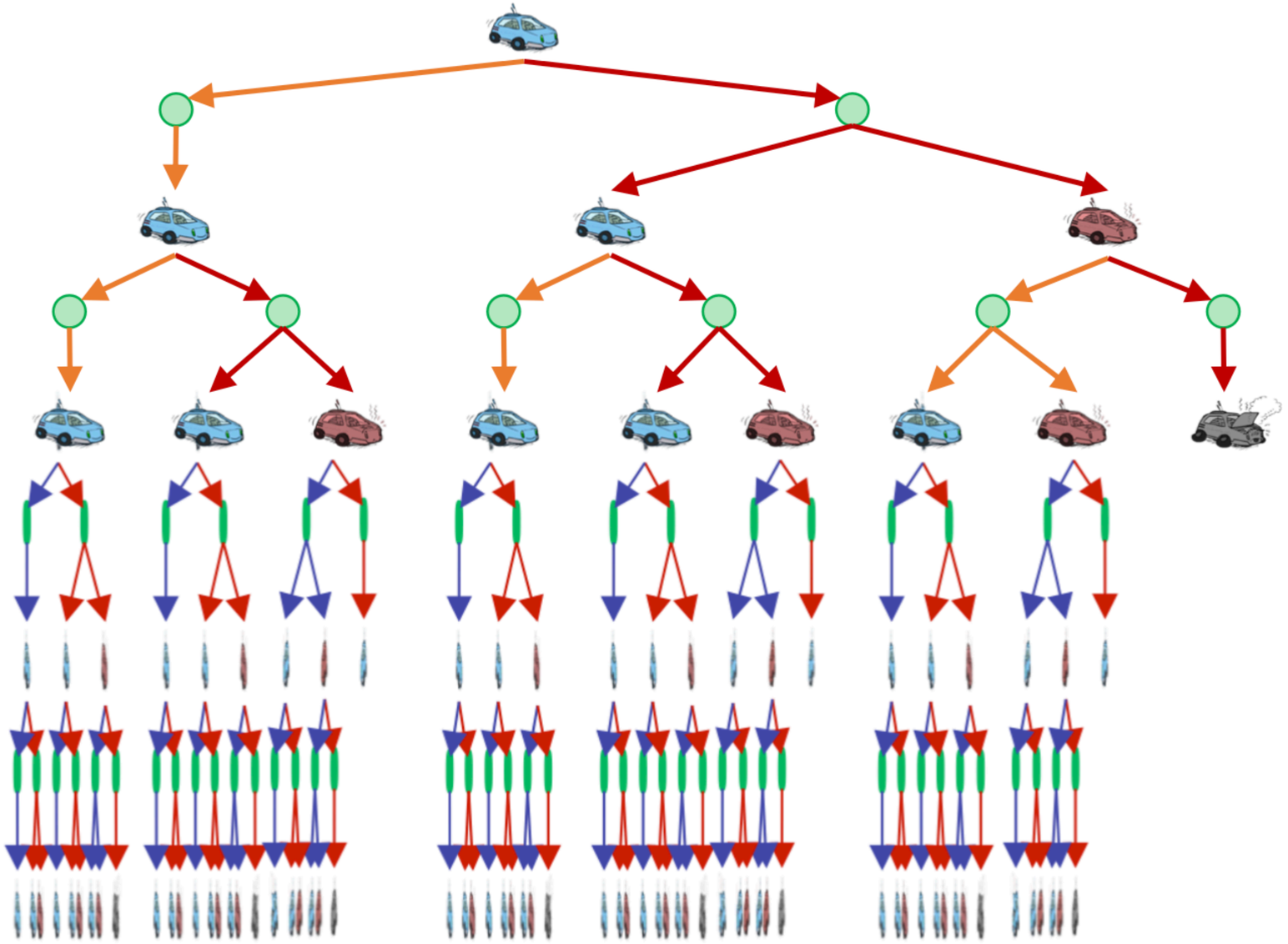


Solving MDPs

Racing Search Tree

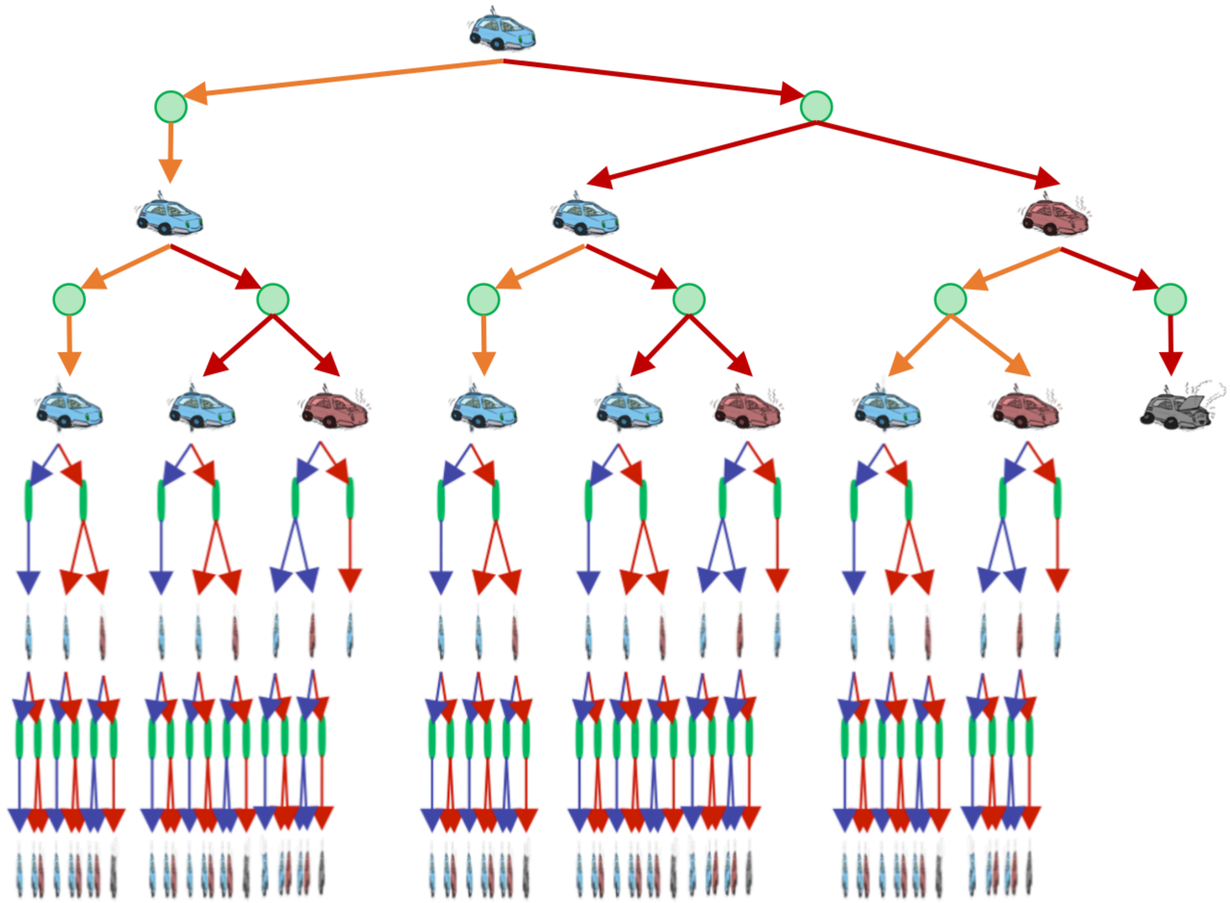


Racing Search Tree



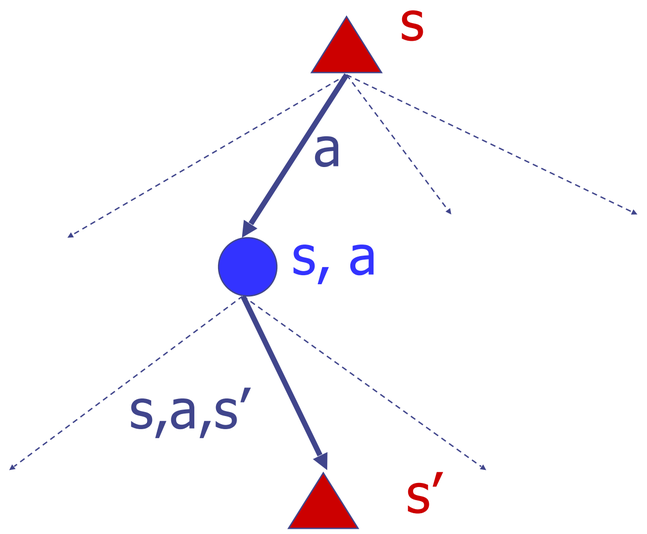
Racing Search Tree

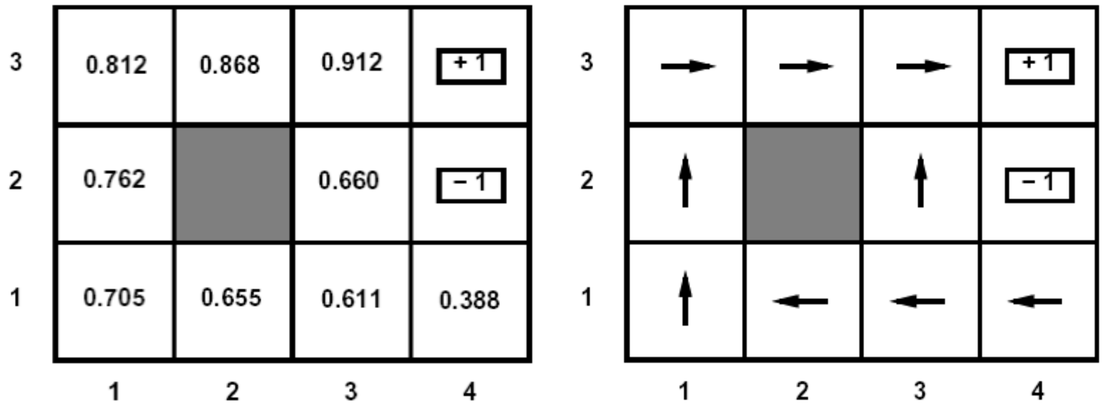
- We're doing way too much work with expectimax!
- Problem: States are repeated
  - Idea: Only compute needed quantities once
- Problem: Tree goes on forever
  - Idea: Do a depth-limited computation, but with increasing depths until change is small
  - Note: deep parts of the tree eventually don't matter if $\gamma < 1$



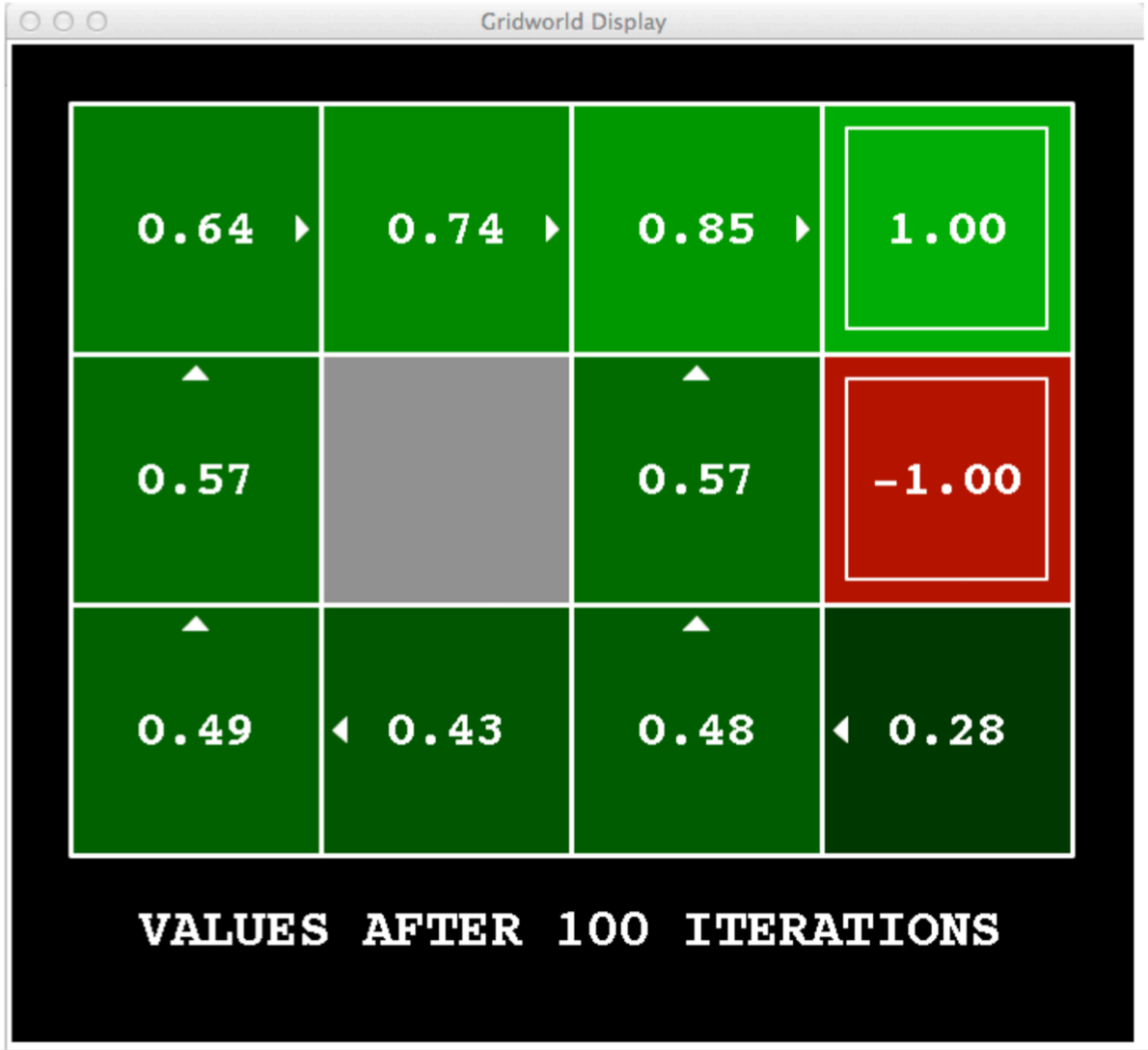
Optimal Utilities

- Define the value of a state s:
  - $V^*(s)$ = expected utility starting in s and acting optimally
- Define the value of a q-state (s,a):
  - $Q^*(s,a)$ = expected utility starting in s, taking action a and thereafter acting optimally
- Define the optimal policy:
  - $\pi^*(s)$ = optimal action from state s



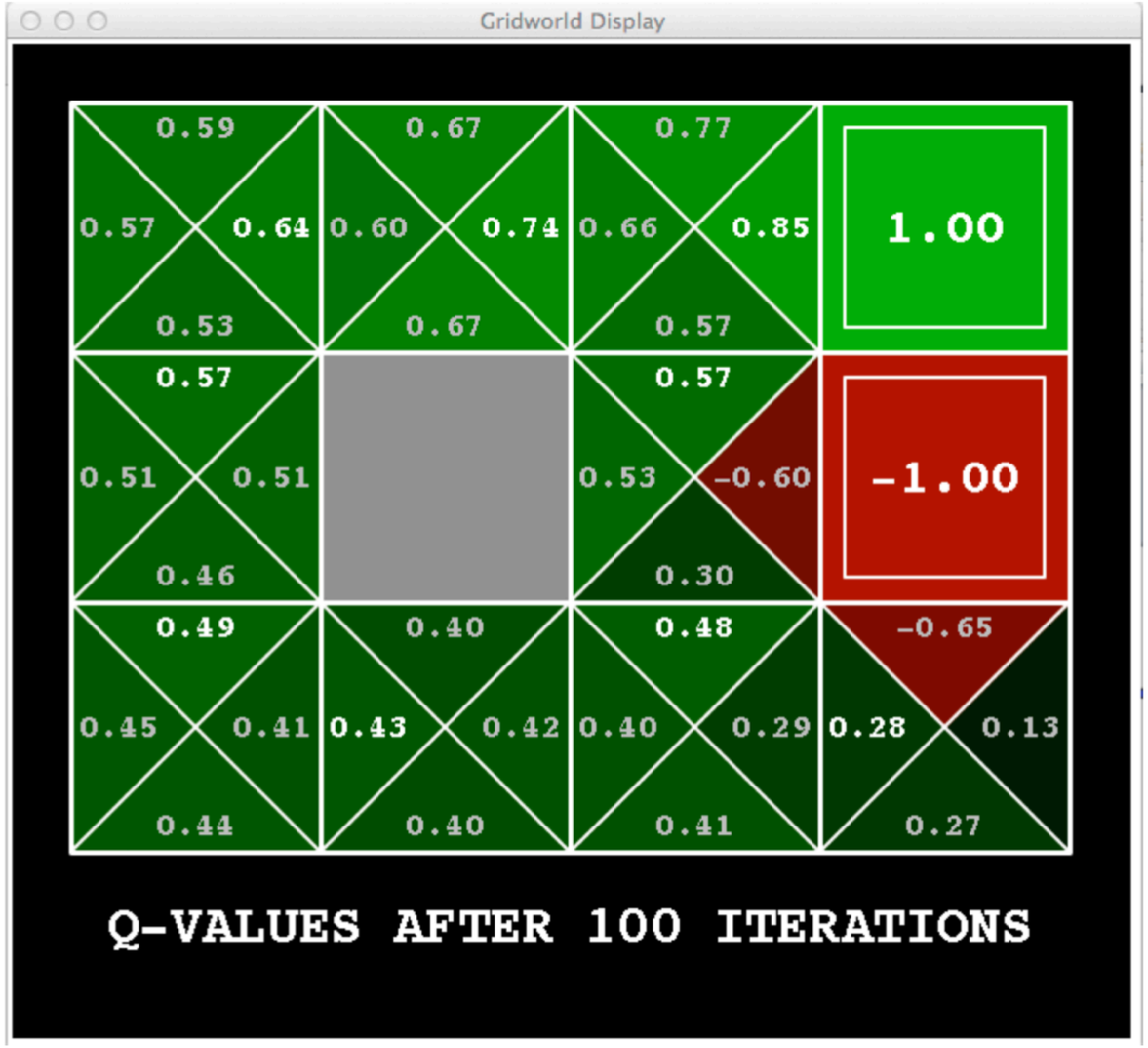


Snapshot of Demo-Gridworld V Values



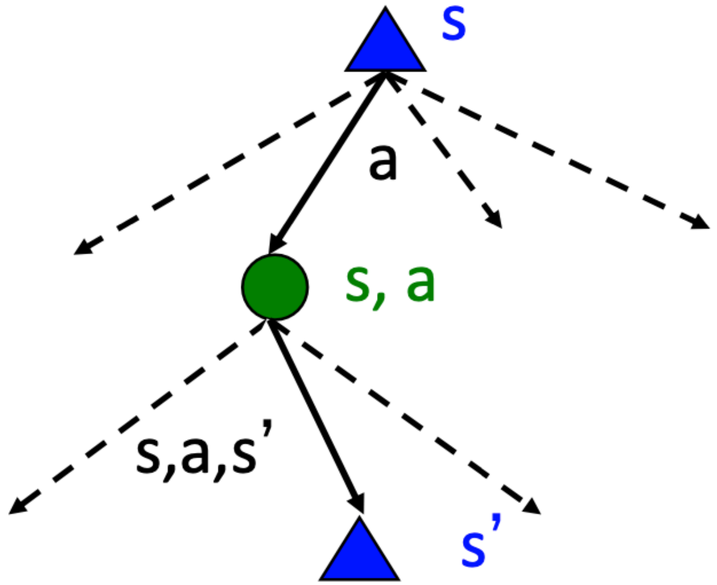
Noise = 0.2
Discount = 0.9
Living reward = 0

Snapshot of Demo-Gridworld Q Values



Noise = 0.2
Discount = 0.9
Living reward = 0

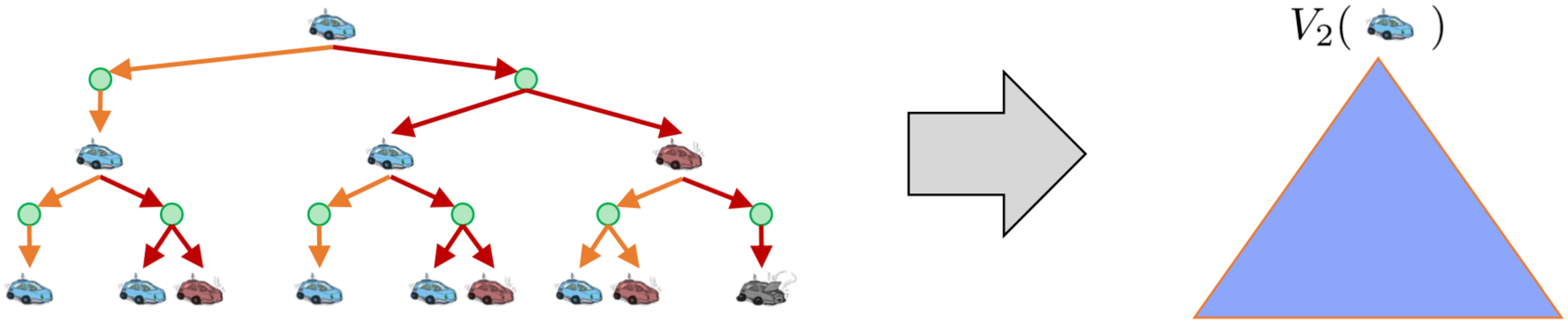
Values of States



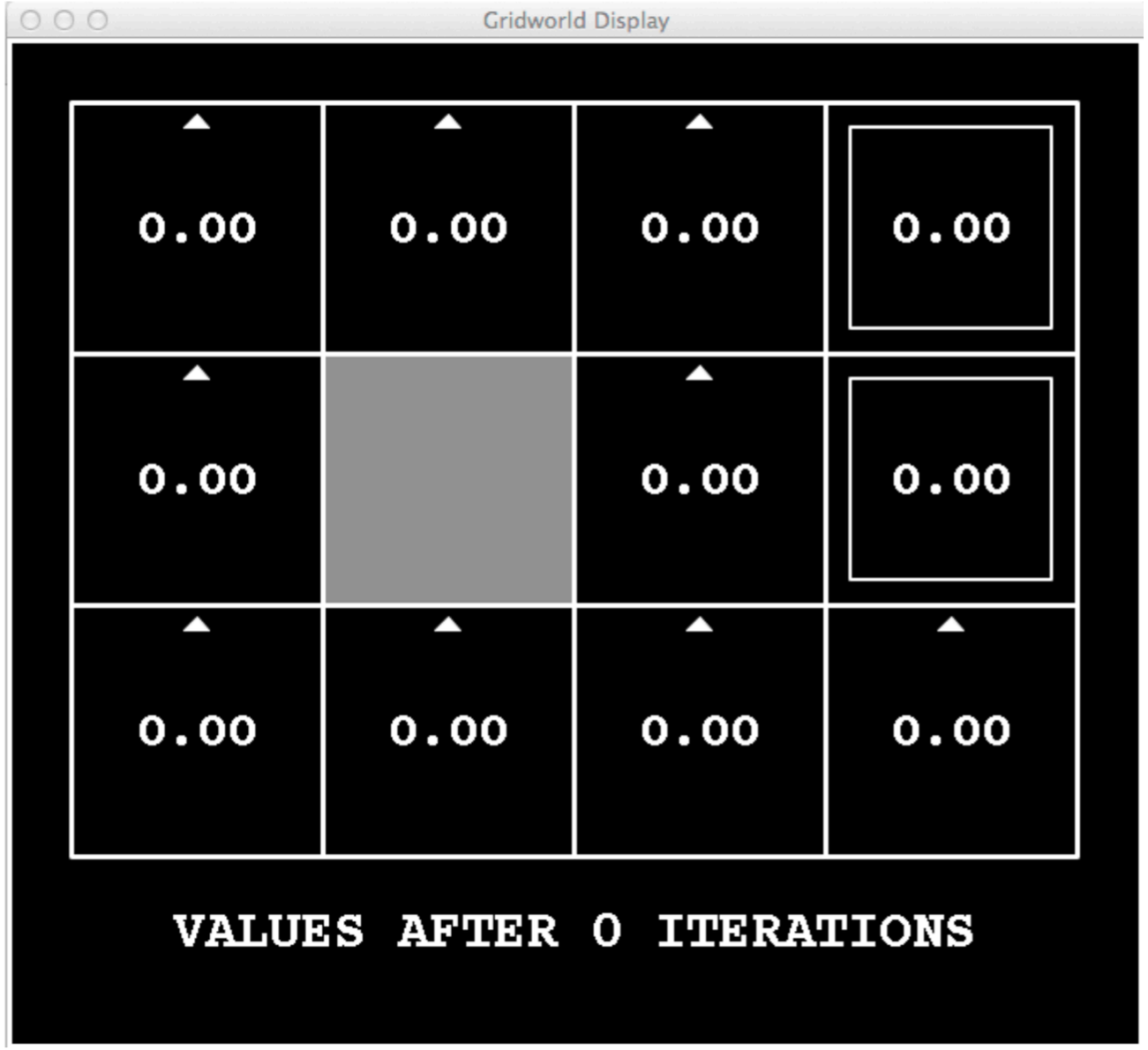
- Recursive definition of value:
  - $V^*(s) = \max_a Q^*(s, a)$
  - $Q^*(s, a) = \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V^*(s') \right]$
  - $V^*(s) = \max_a \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V^*(s') \right]$

Time-Limited Values

- Key idea: time-limited values
- Define $V_k(s)$ to be the optimal value of ’s’ if the game ends in ‘k’ more time steps
  - Equivalently, it’s what depth-k expectimax would give from ’s'

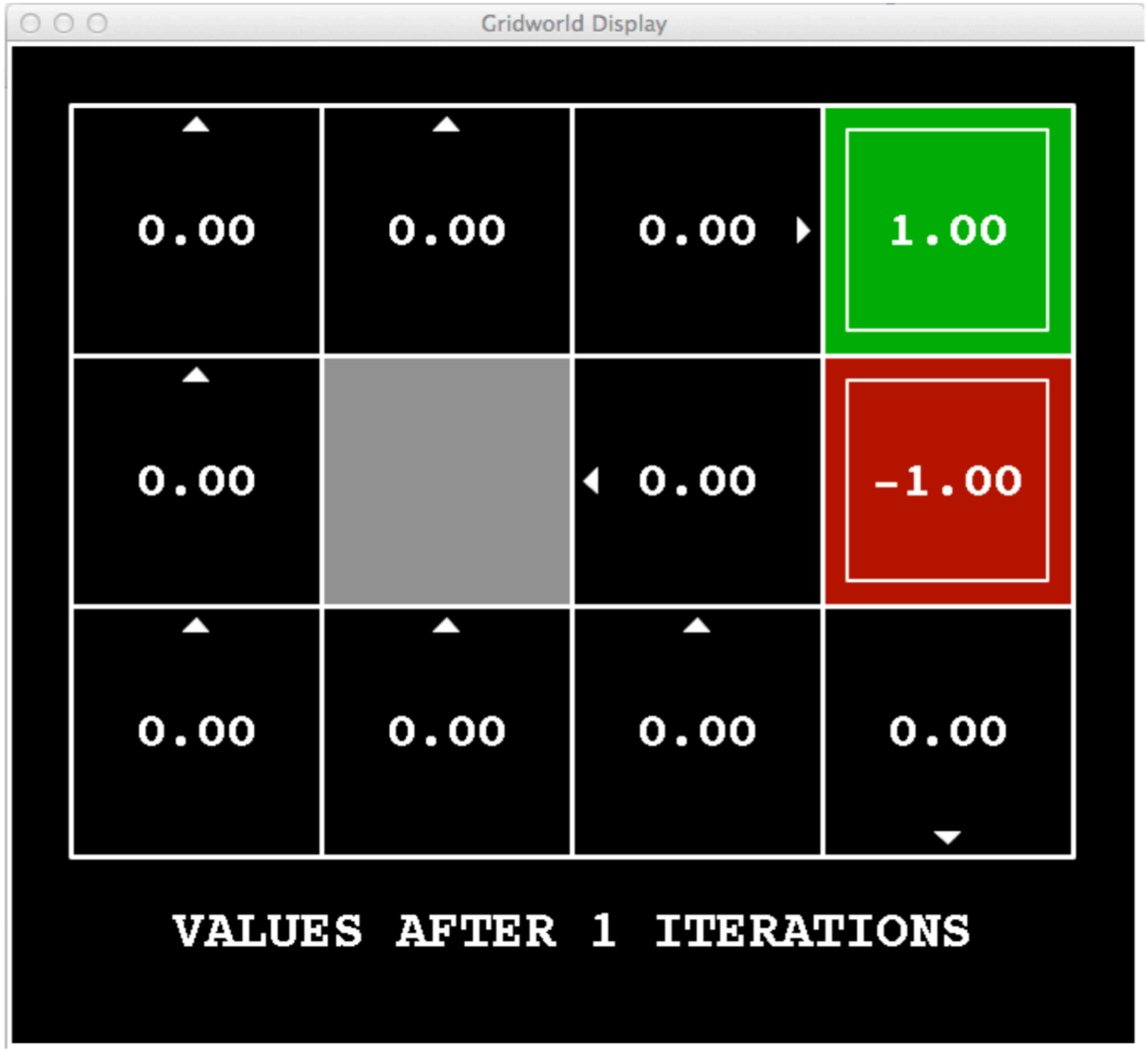


K=0



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

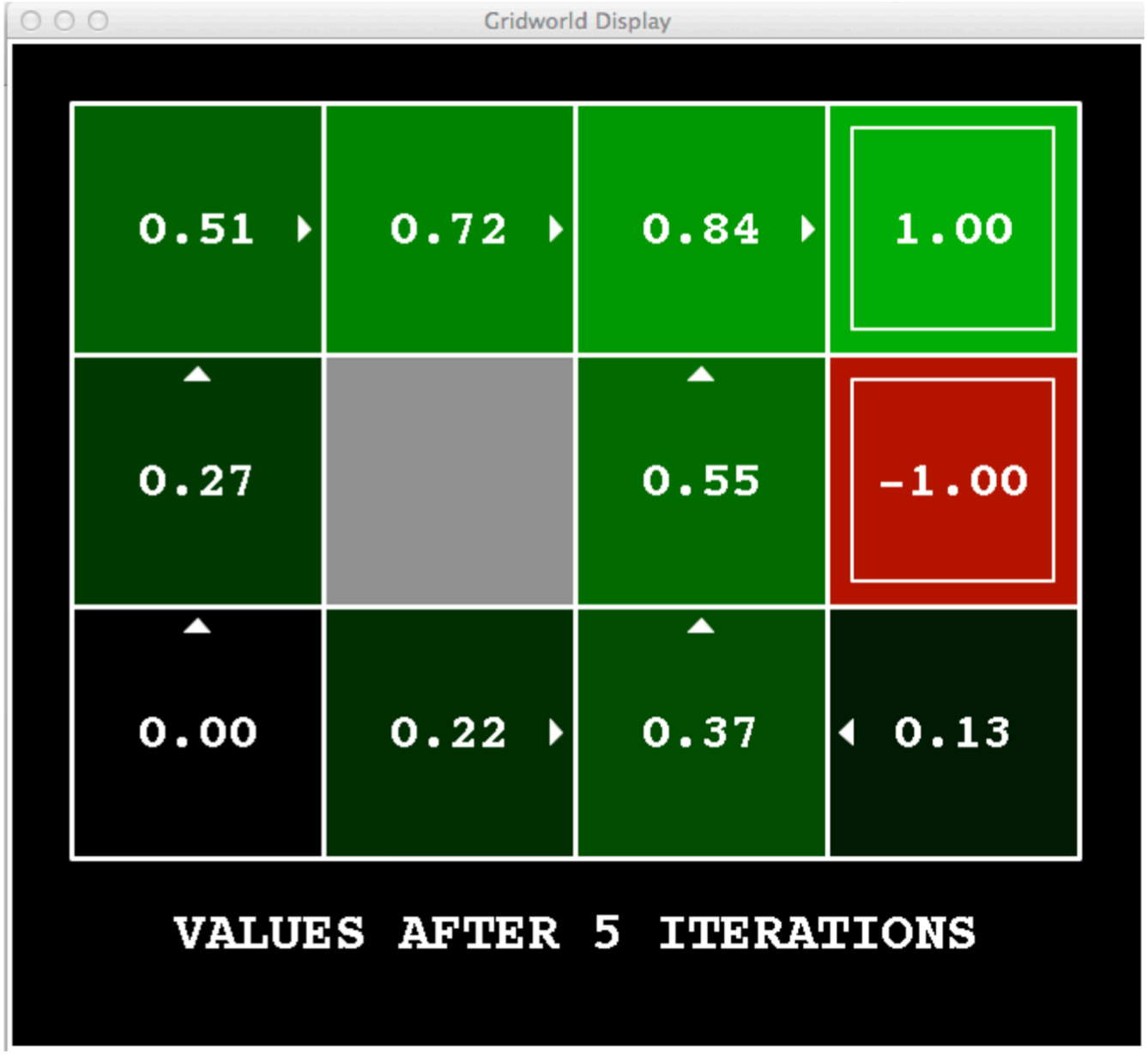
K=3

Noise = 0.2
Discount = 0.9
Living reward = 0

K=4

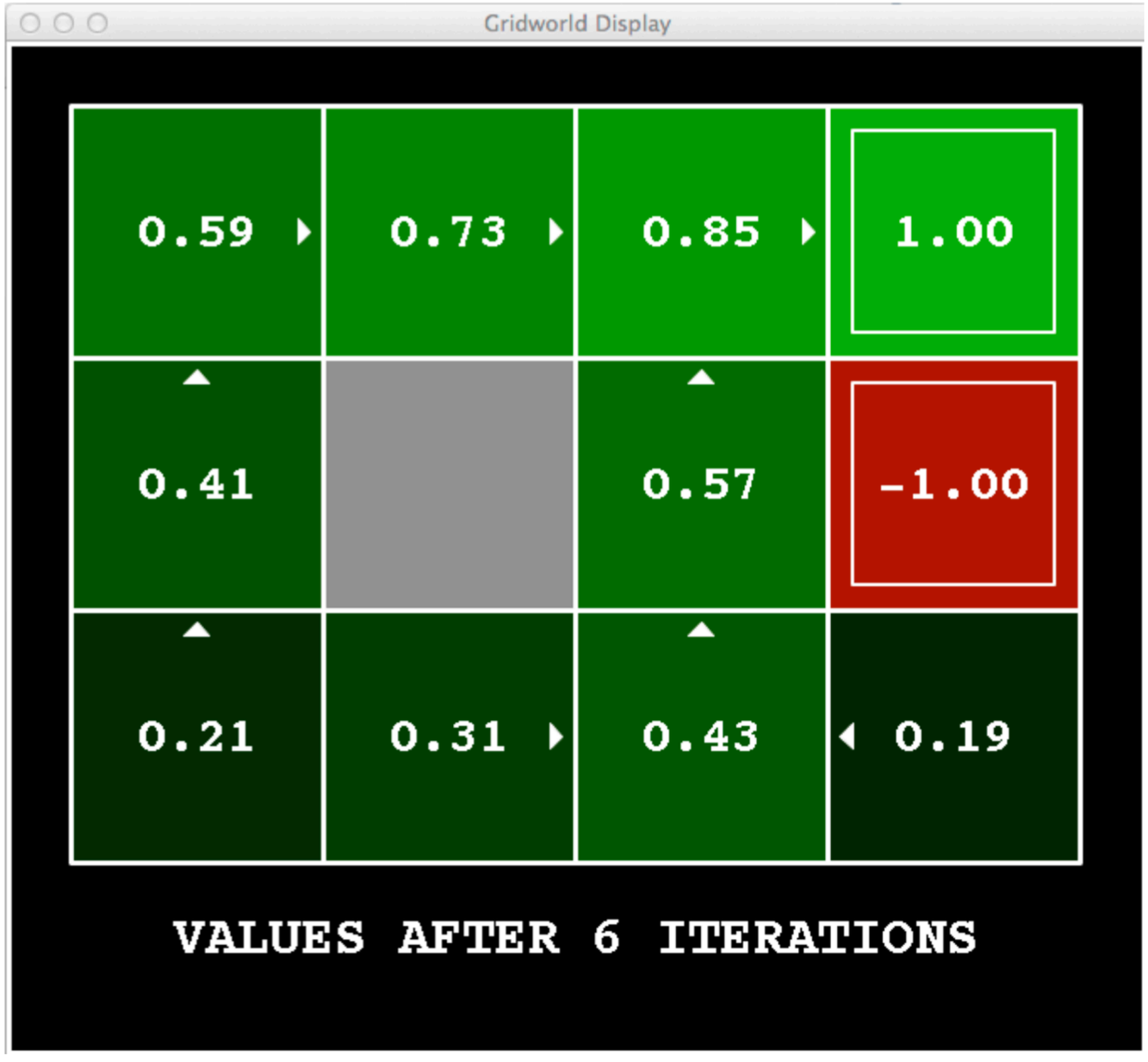
Noise = 0.2
Discount = 0.9
Living reward = 0

K=5



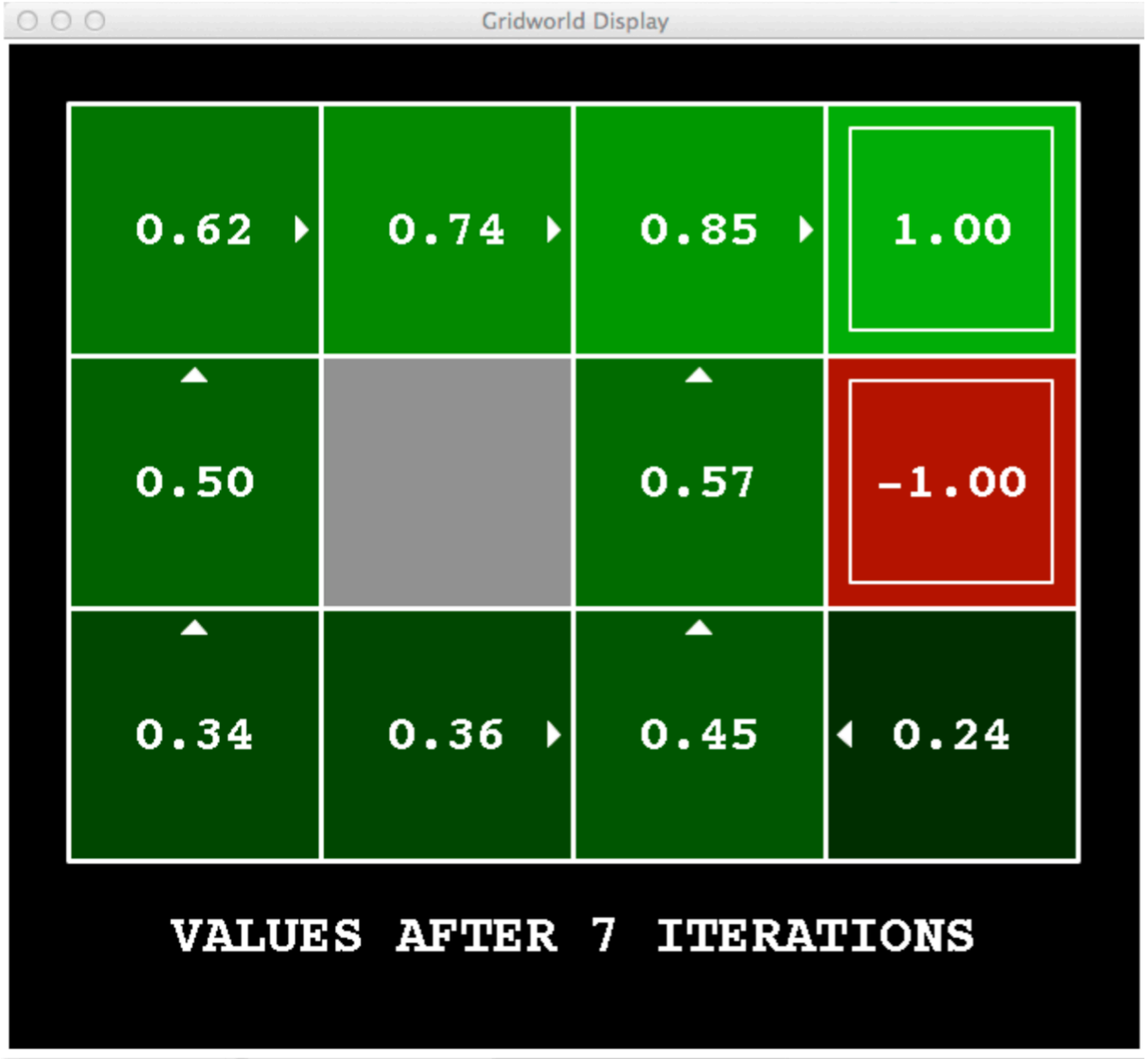
Noise = 0.2
Discount = 0.9
Living reward = 0

K=6



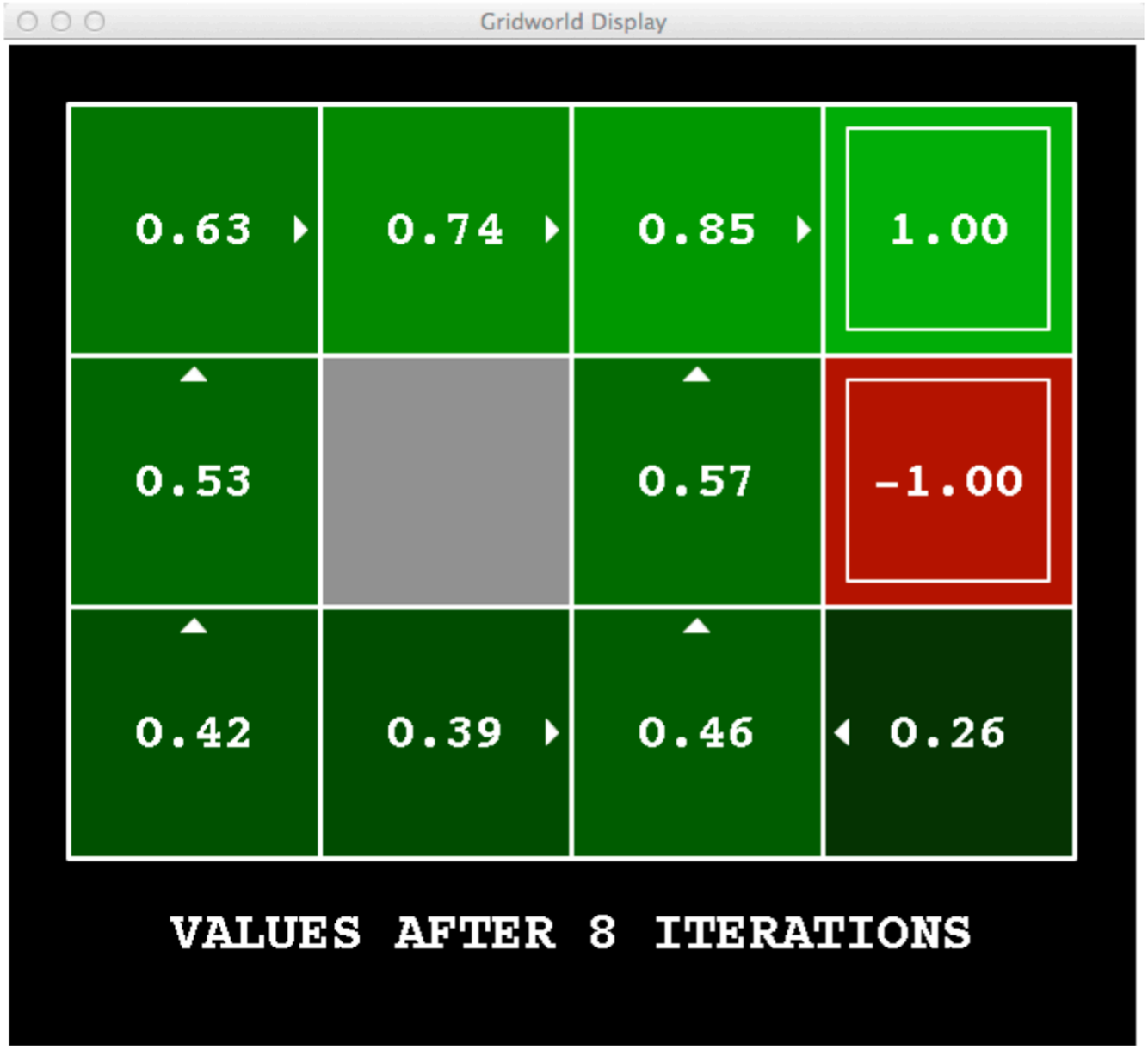
Noise = 0.2
Discount = 0.9
Living reward = 0

K=7



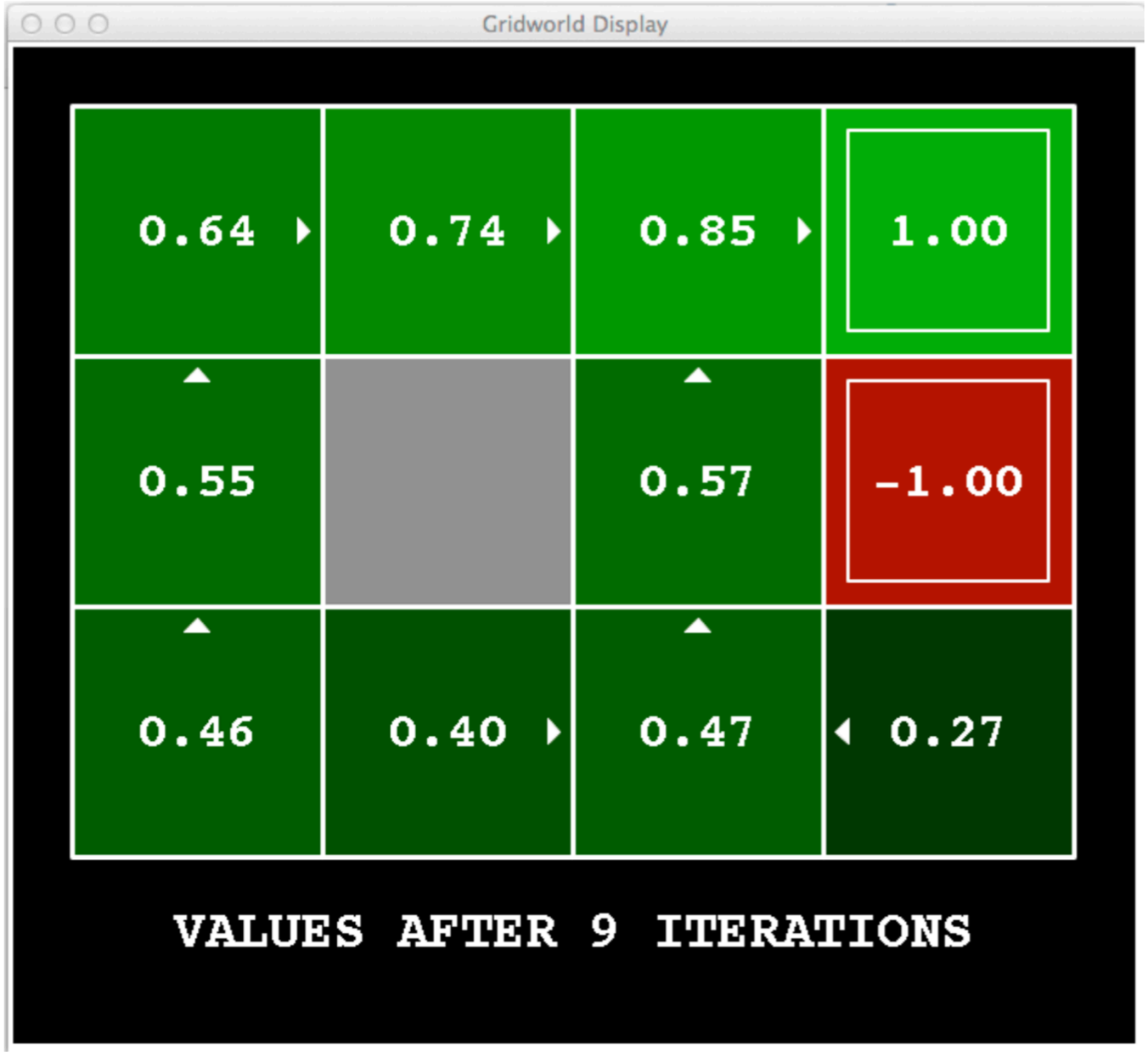
Noise = 0.2
Discount = 0.9
Living reward = 0

K=8



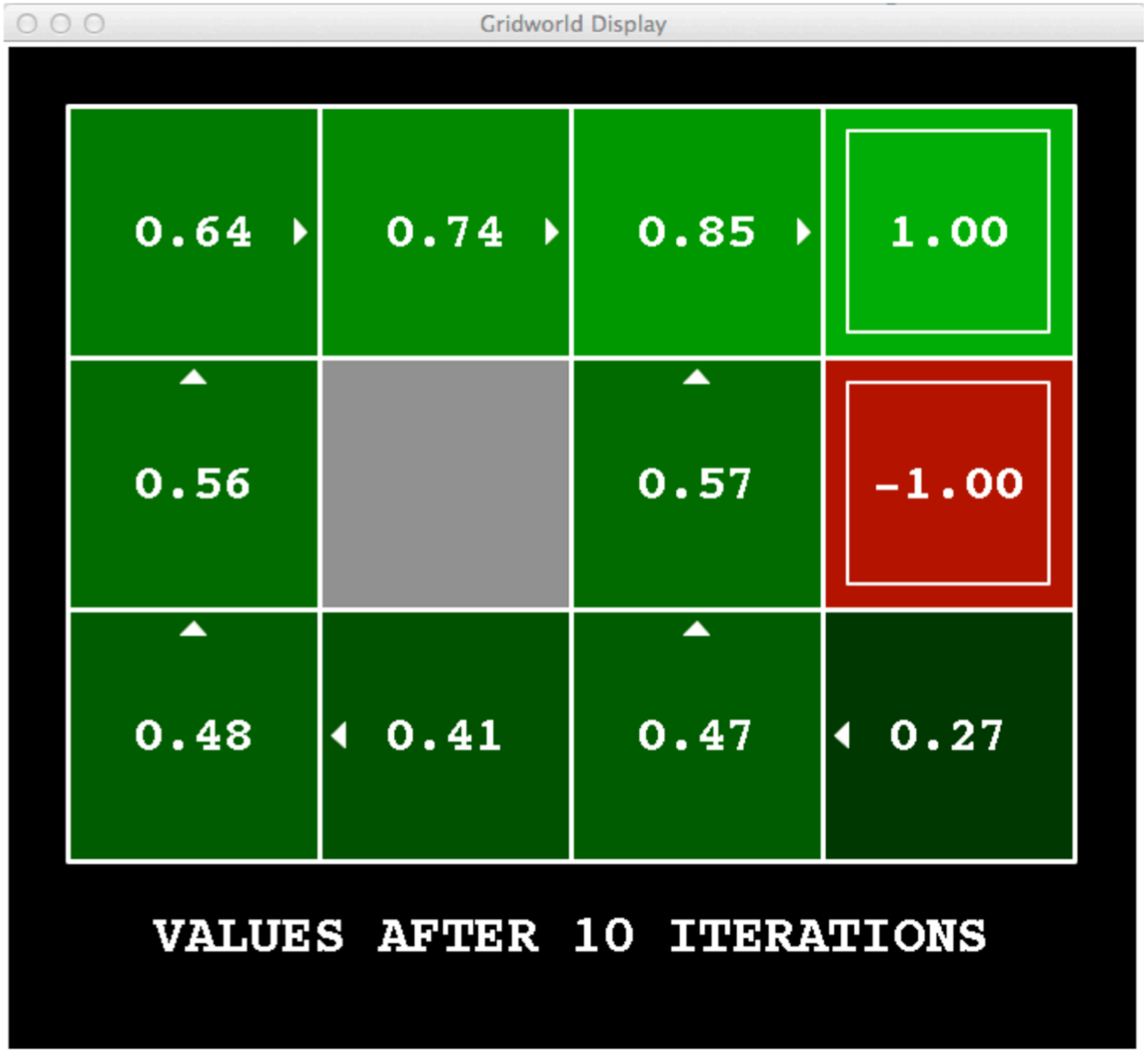
Noise = 0.2
Discount = 0.9
Living reward = 0

K=9



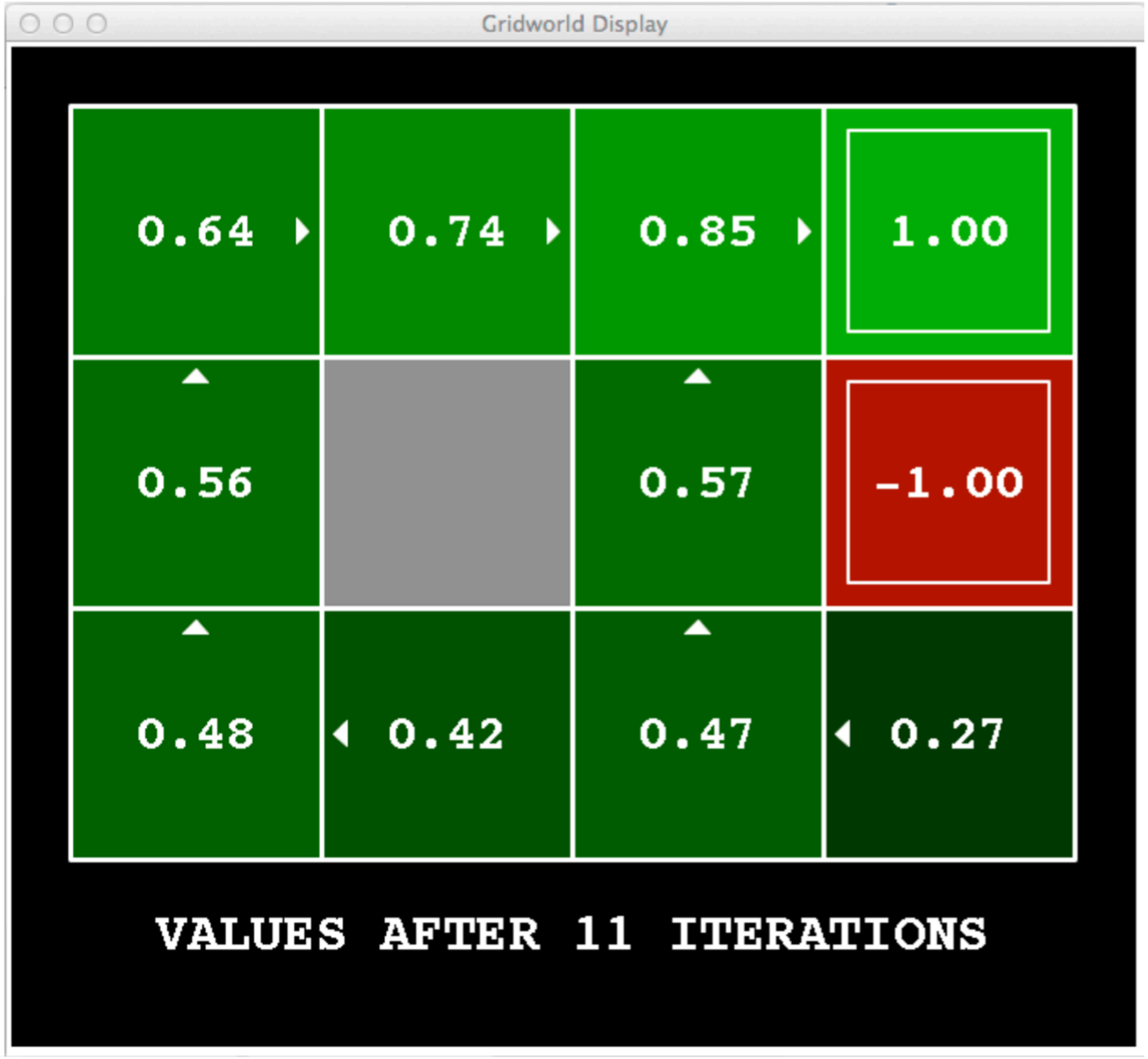
Noise = 0.2
Discount = 0.9
Living reward = 0

K=10



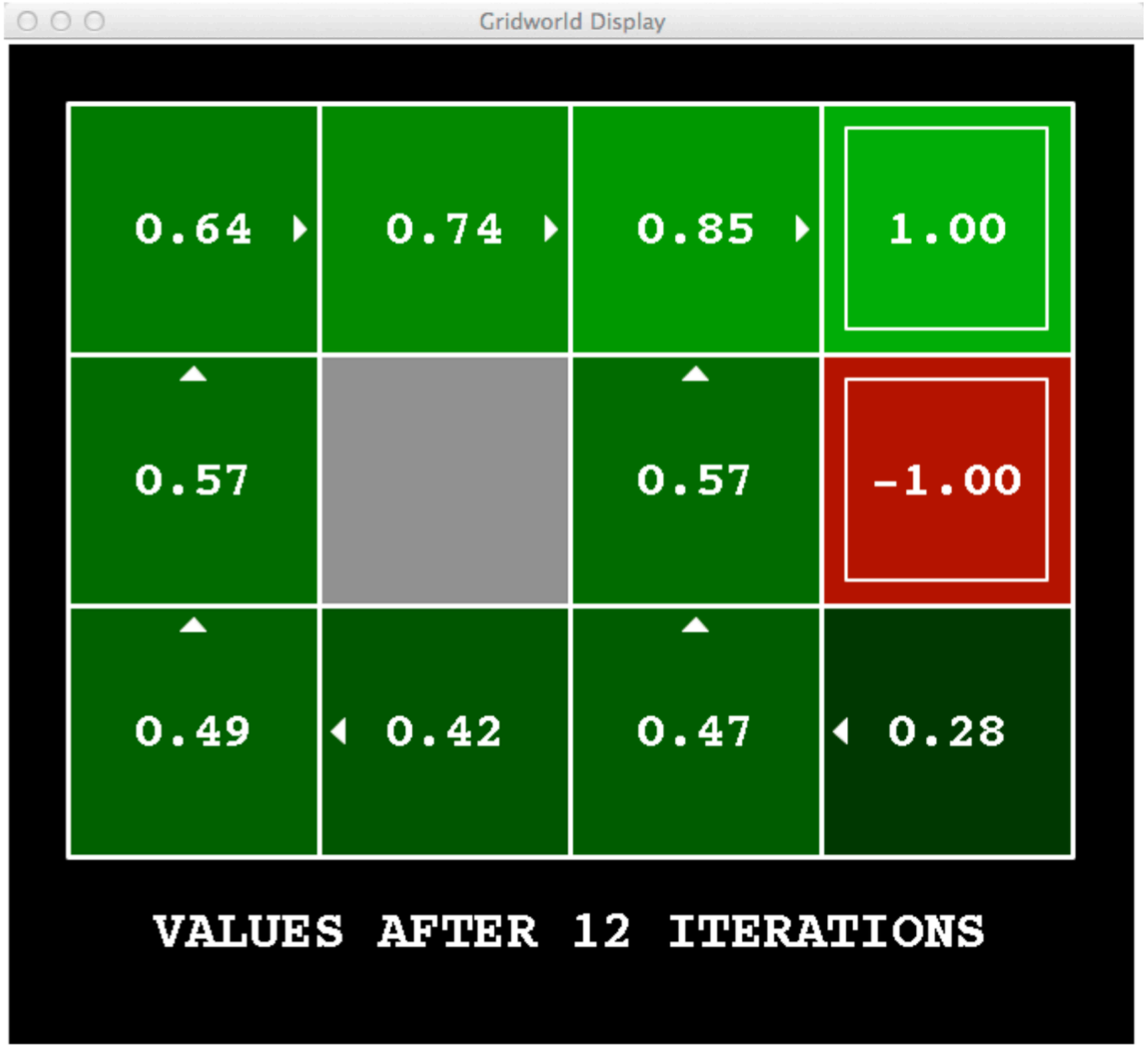
Noise = 0.2
Discount = 0.9
Living reward = 0

K=11



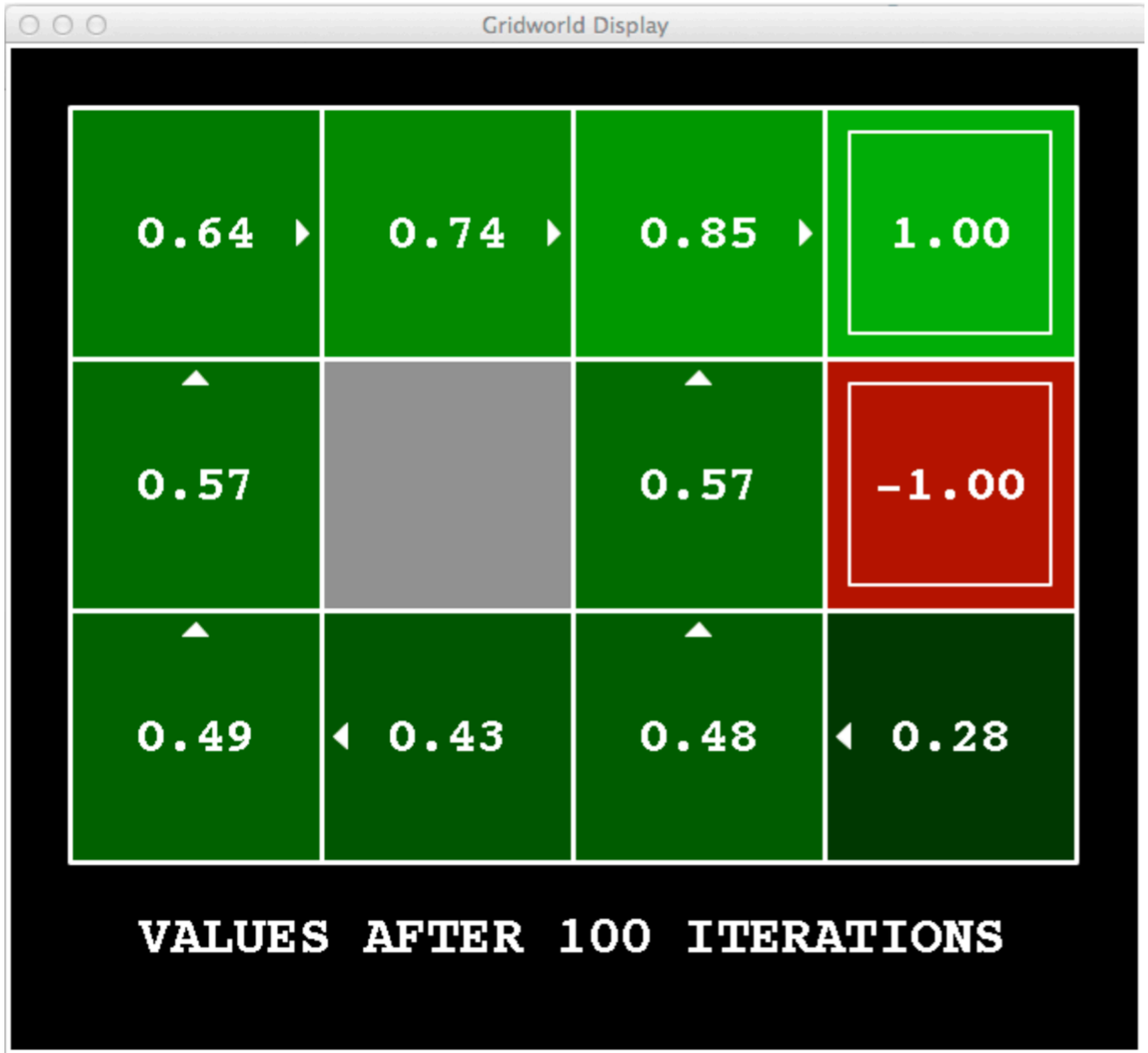
Noise = 0.2
Discount = 0.9
Living reward = 0

K=12



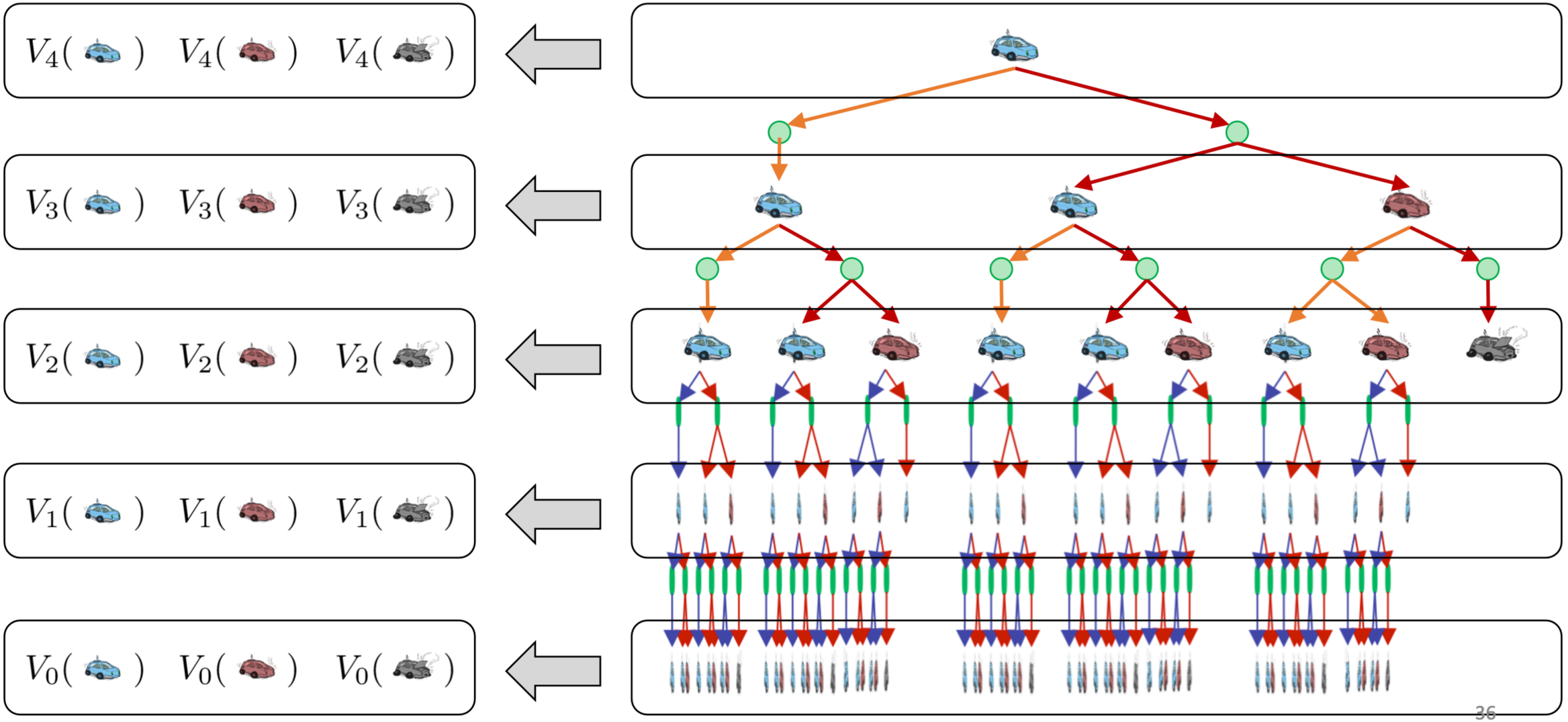
Noise = 0.2
Discount = 0.9
Living reward = 0

K=100



Noise = 0.2
Discount = 0.9
Living reward = 0

Computing Time-Limited Values



Value Iteration

Value Iteration

- Idea:
  - Start with $V_0(s) = 0$, no time steps left means an expected reward sum of zero
  - Given $V_i(s)$ values, do one ply of expectimax from each state:

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

  - Throw out old V_i values
  - This is called a value update or Bellman update
  - Repeat until convergence
  - Complexity of each iteration $O(s^2 A)$

- Theorem: will converge to unique optimal values
  - Basic idea: approximations get refined towards optimal values
  - Policy may converge long before values do

Example: Value Iteration

V_2

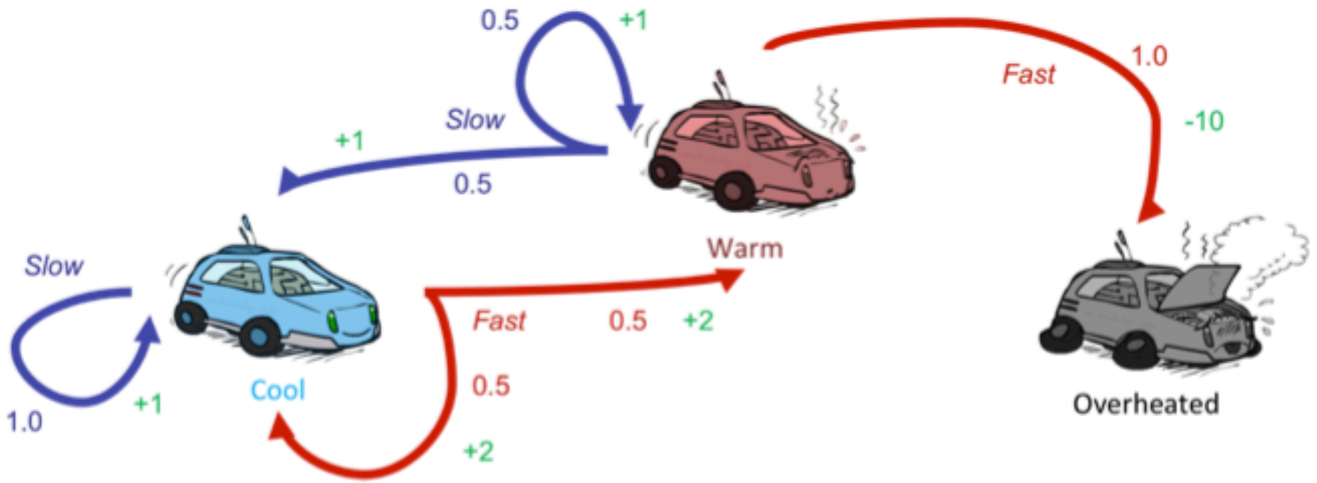
V_1

S: 1

F: $0.5 \times 2 + 0.5 \times 2 = 2$

V_0

0 0 0



Assume no discount!

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

Example: Value Iteration

V_2

V_1

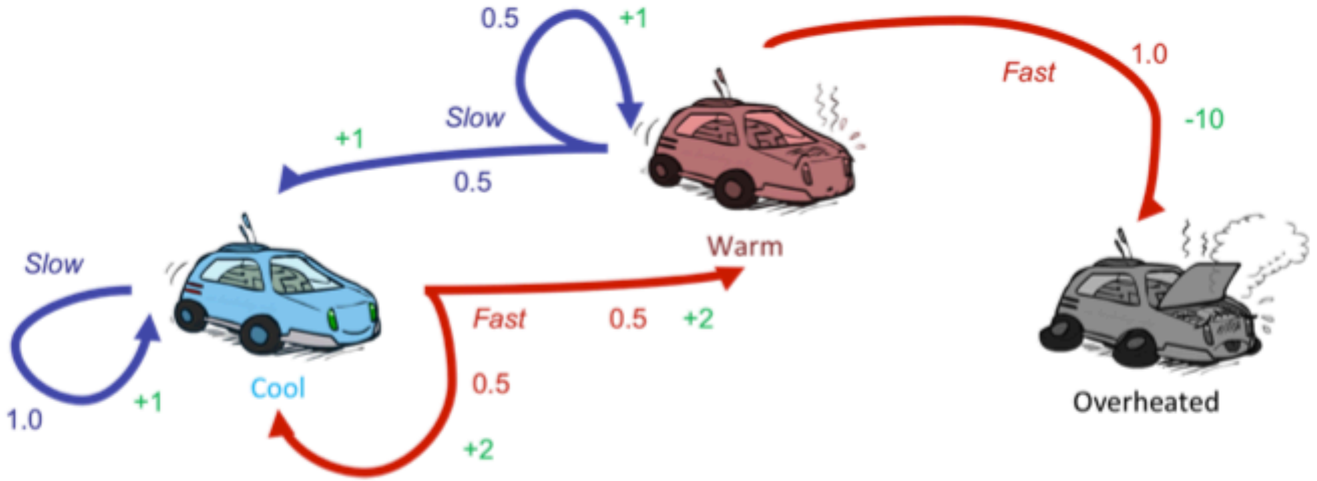
2

S: 0.5×1 + 0.5×1 = 1

F: -10

V_0

0 0 0

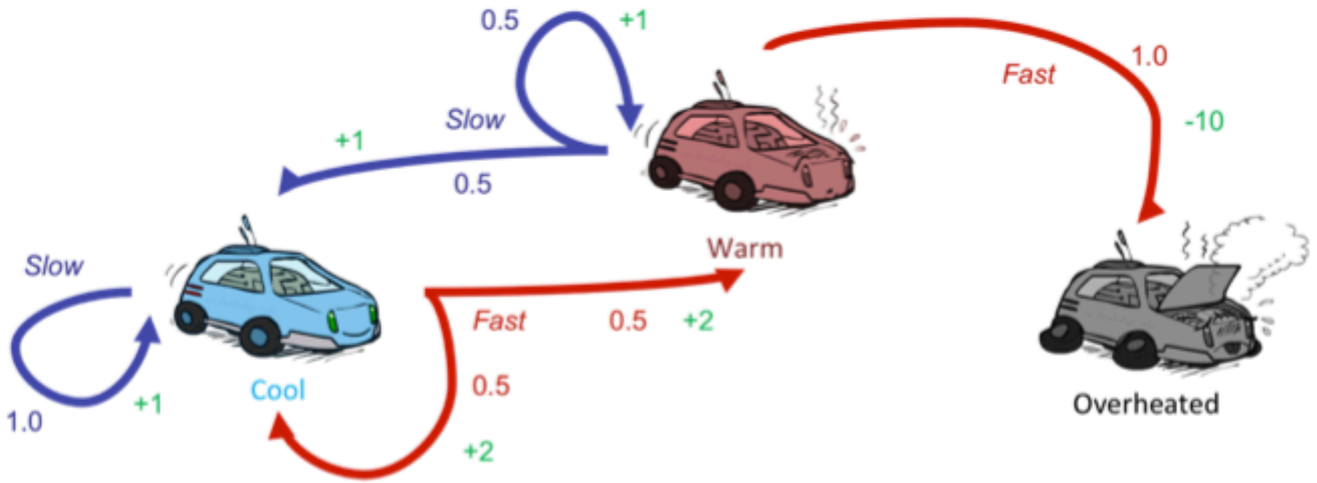


Assume no discount!

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

Example: Value Iteration

V_2			
V_1	2	1	0
V_0	0	0	0



Assume no discount!

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

Summary and Next Time

- Today
  - MDPs
    - Discounting
    - Value Iteration
    - Bellman Equations
- Next Time
  - Markov Decision Processes
  - Reinforcement Learning