COL333/671: Introduction to AI

Semester I, 2021

Markov Decision Processes

Rohan Paul

Outline

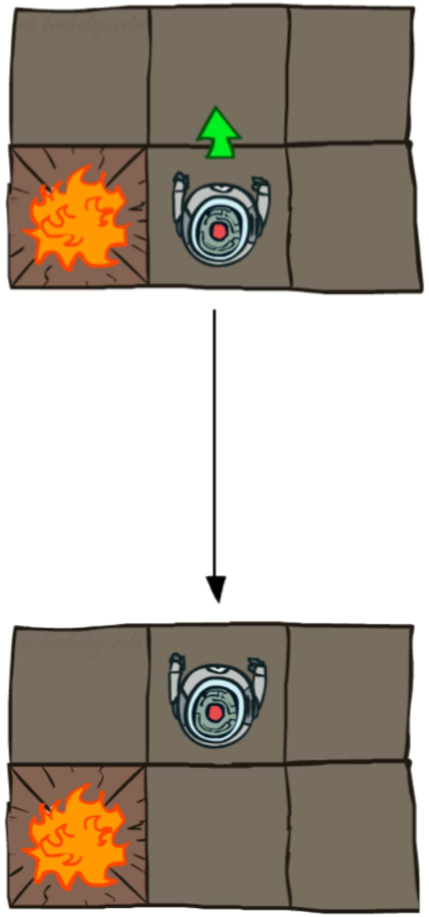
- Last Class
  - Utilities and Probabilities
- This Class
  - Markov Decision Processes
- Reference Material
  - Please follow the notes as the primary reference on this topic. Supplementary reading on topics covered in class from AIMA Ch 17 sections 17.1 – 17.3.

Acknowledgement

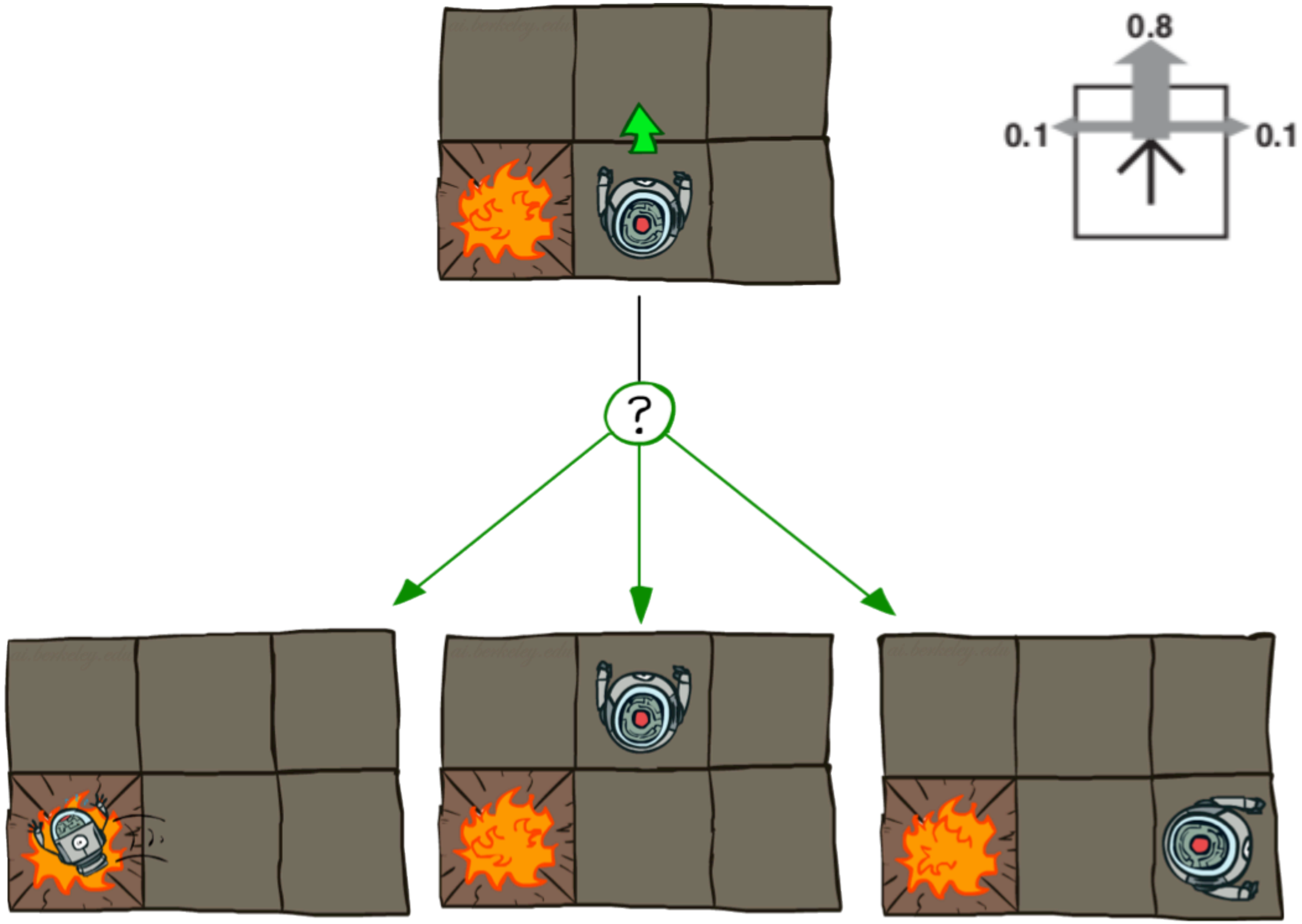
These slides are intended for teaching purposes only. Some material has been used/adapted from web sources and from slides by Doina Precup, Dorsa Sadigh, Percy Liang, Mausam, Dan Klein, Anca Dragan, Nicholas Roy, Emilio Frazzoli and others.

Deterministic vs. Stochastic Actions

Deterministic Action Outcomes



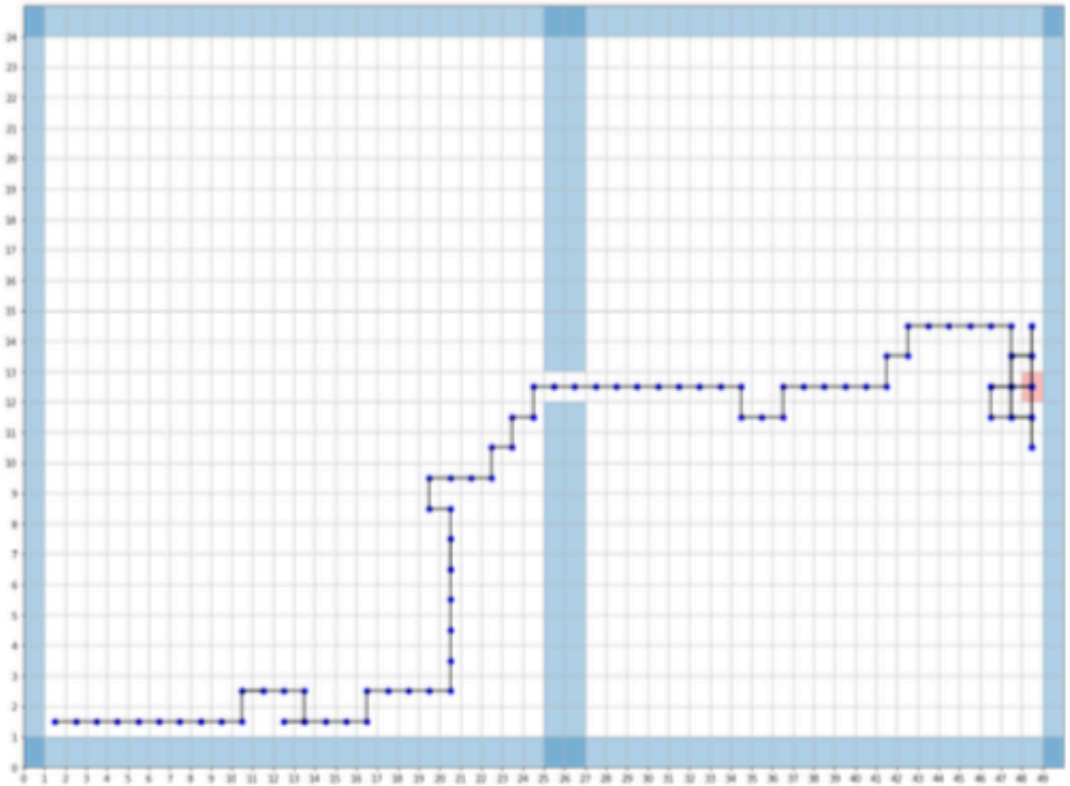
Stochastic Action Outcomes



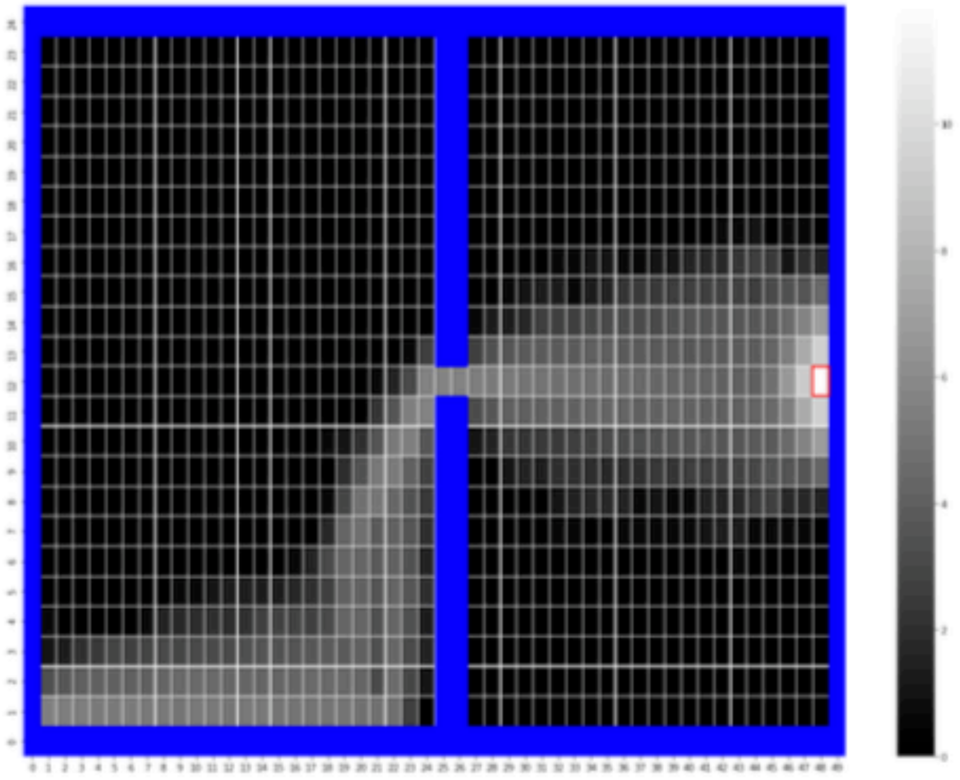
Need to plan for contingencies

Example: Sample paths through an MDP

A sample path through the MDP

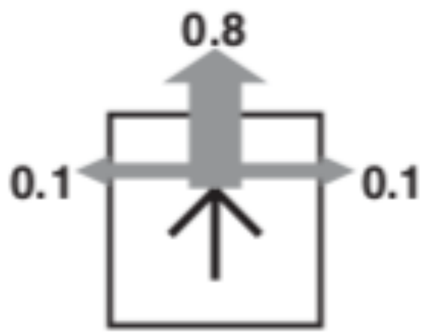


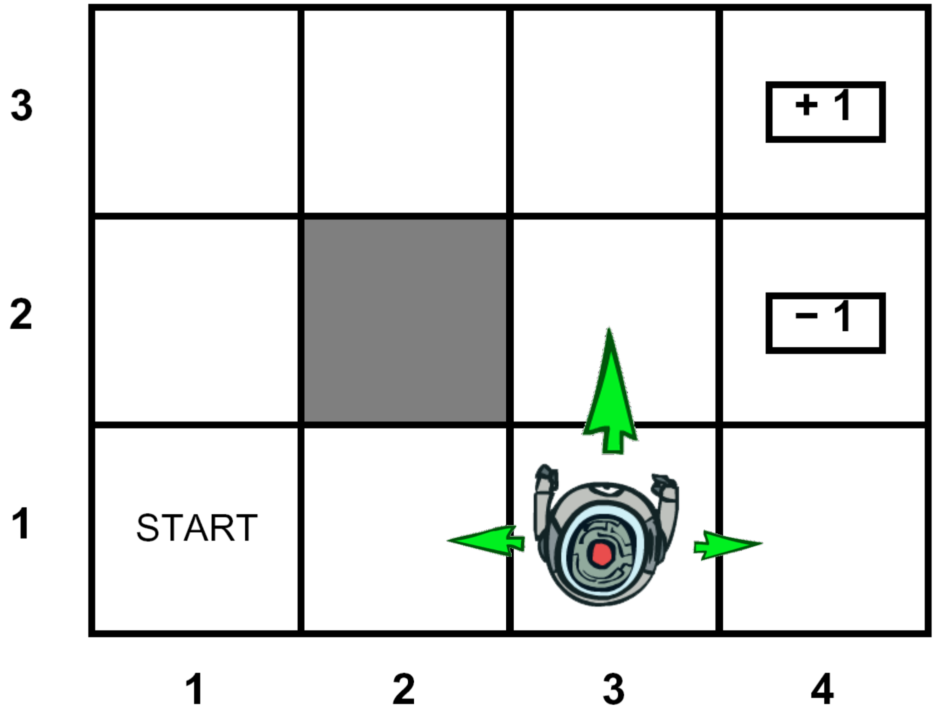
State visitations after multiple runs



Grid World Example

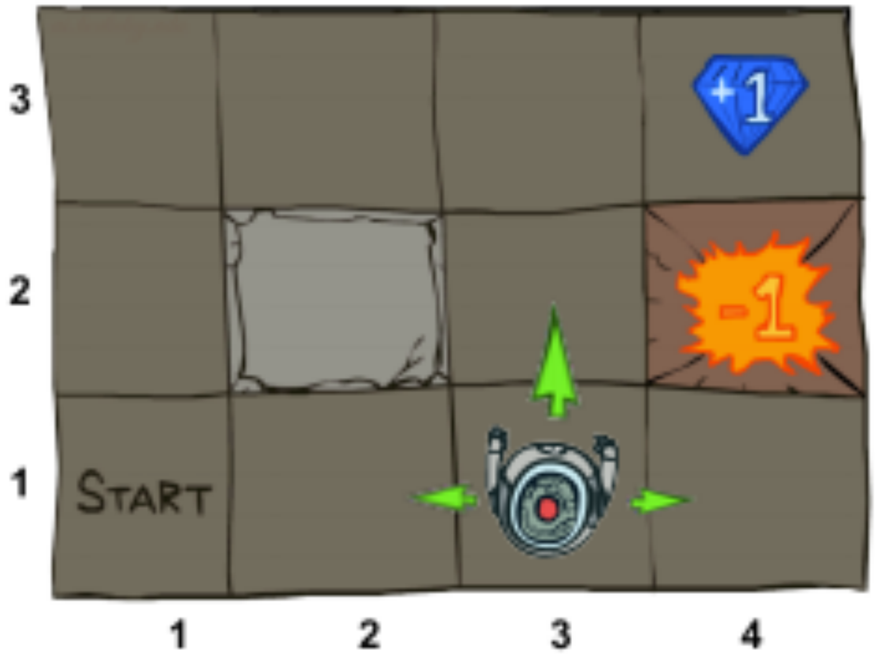
- Actions do not always go as planned
  - 80% of the time, the action North takes the agent North (if there is no wall there)
  - 10% of the time, North takes the agent West; 10% East
  - If there is a wall in the direction the agent would have been taken, the agent stays put
- The agent receives rewards each time step
  - Small “living” reward each step (can be negative)
  - Large rewards come at the end (negative or positive)
- Overall Goal: maximize sum of rewards
  - Fundamentally a sequential decision-making problem.
  - Taking an action now can have an impact later.





Markov Decision Processes

- 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')
    - Probability that a from s leads to s', i.e., P(s' | s, a)
    - Also called the model or the dynamics



$T(s_{11}, E, \ldots$

$\ldots$

$T(s_{31}, N, s_{11}) = 0$

$\ldots$

$T(s_{31}, N, s_{32}) = 0.8$

$T(s_{31}, N, s_{21}) = 0.1$

$T(s_{31}, N, s_{41}) = 0.1$

$\ldots$

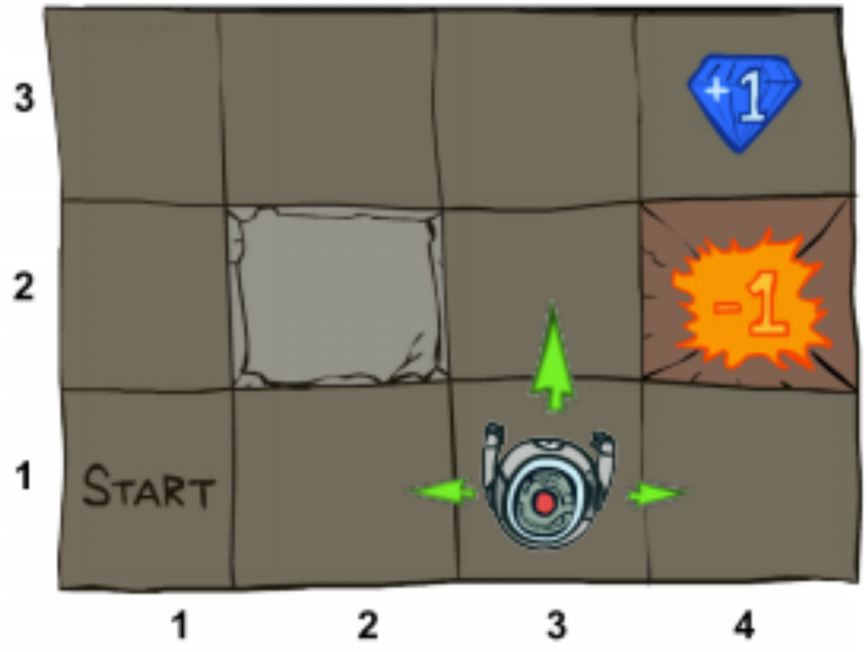
T is a Big Table!

11 X 4 x 11 = 484 entries

For now, we give this as input to the agent

Markov Decision Processes

- 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')
    - Probability that a from s leads to s', i.e., P(s' | s, a)
    - Also called the model or the dynamics
  - A reward function R(s, a, s')



...

$R(s_{32}, N, s_{33}) = -0.01$

...

$R(s_{32}, N, s_{42}) = -1.01$

...

$R(s_{33}, E, s_{43}) = 0.99$

Cost of breathing

R is also a Big Table!

For now, we also give this to the agent

Markov Decision Processes

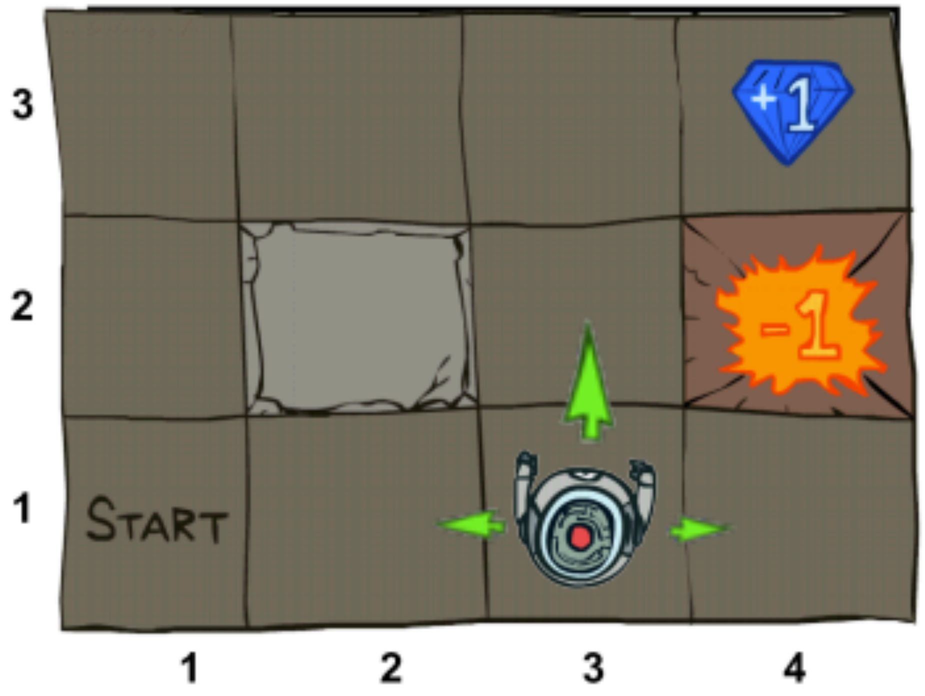
- An MDP is defined by:
  - A set of states s in S
  - A set of actions a in A
  - A transition function T(s, a, s')
    - Probability that a from s leads to s', i.e., P(s'| s, a)
    - Also called the model or the dynamics
  - A reward function R(s, a, s')
    - Sometimes just R(s) or R(s')

...

$R(s_{33}) = -0.01$

$R(s_{42}) = -1.01$

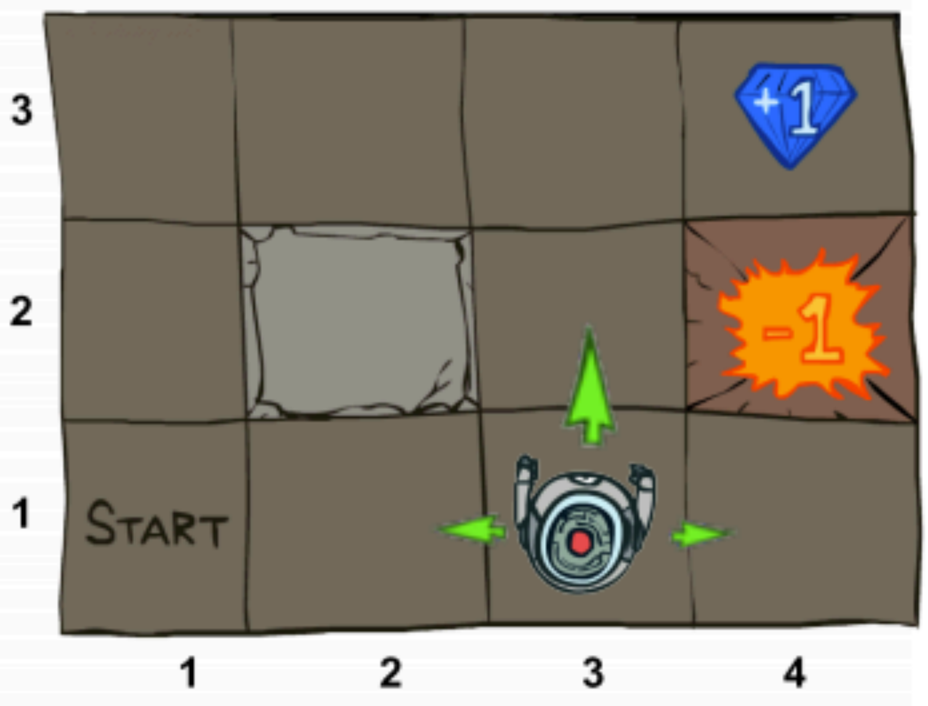
$R(s_{43}) = 0.99$



Note: two notations are followed in literature. One in which rewards are associated with states and in the other rewards are associated with state transitions. Both the notations are equivalent and accepted.

Markov Decision Processes

- 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')
    - Probability that a from s leads to s', i.e., P(s' | s, a)
    - Also called the model or the dynamics
  - A reward function R(s, a, s')
    - Sometimes just R(s) or R(s')
  - A start state
  - Maybe a terminal state

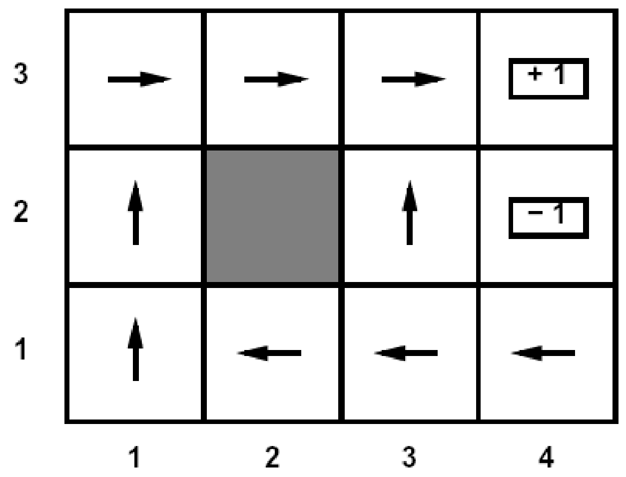


Policies

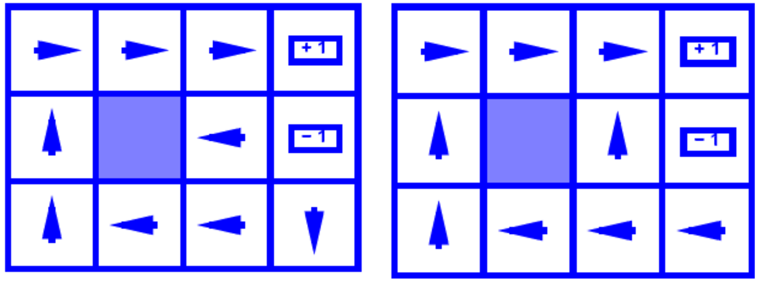
- Deterministic single-agent search problems
  - We determined the optimal plan, or sequence of actions, from start to a goal

- 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 the expected utility if followed**
  - The agent arrives at a state and looks up the action according to the policy.

- Note: there can be many policies, we are to determine the optimal one.



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



There can be other policies that prescribe different actions in a state.

Markov Assumption in MDPs

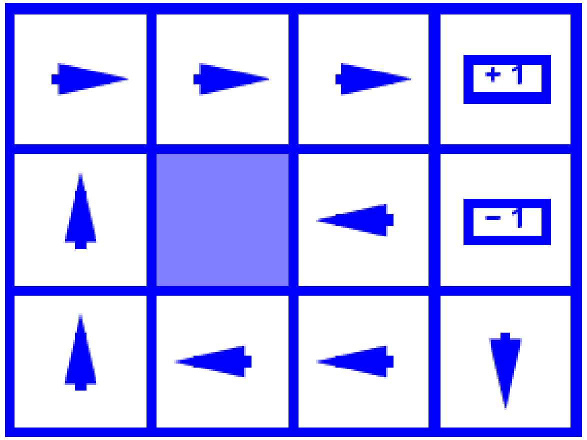
- "Markov"
  - Given the present state, the future and the past are independent
- For Markov decision processes
  - "Markov" means action outcomes depend only on the current state
  - The next state depends on the action and the current state.
  - The past actions taken the past states encountered do not affect the next state.

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

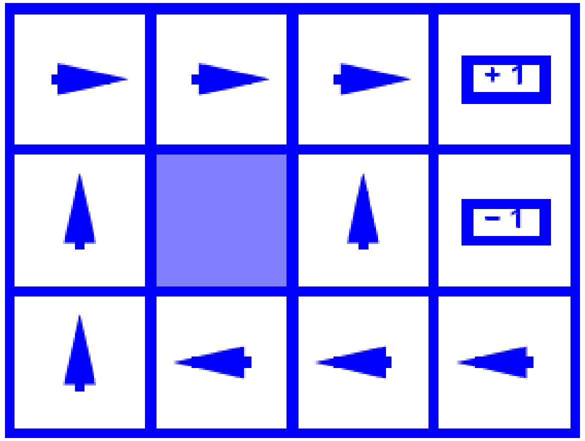
$$= \quad P(S_{t+1} = s' | S_t = s_t, A_t = a_t)$$

Optimal policies for different living rewards

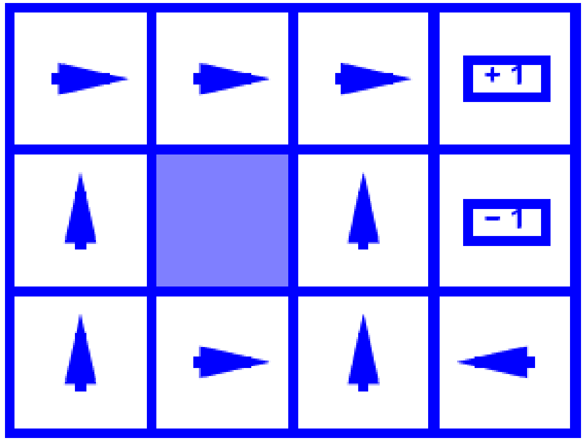
- Interpret reward as the cost of breathing (living reward).
- The value of R(s) balances the risk and reward that the agent takes.



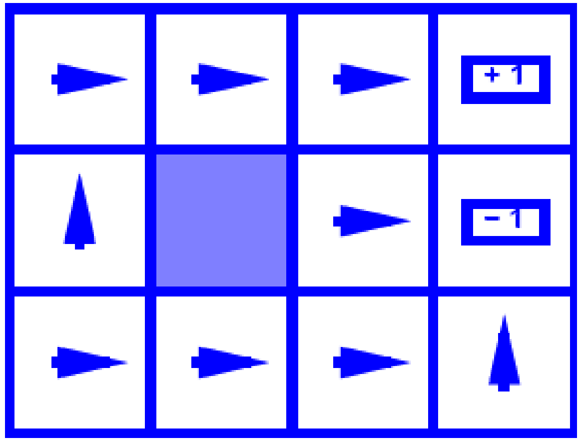
R(s) = -0.01



R(s) = -0.03



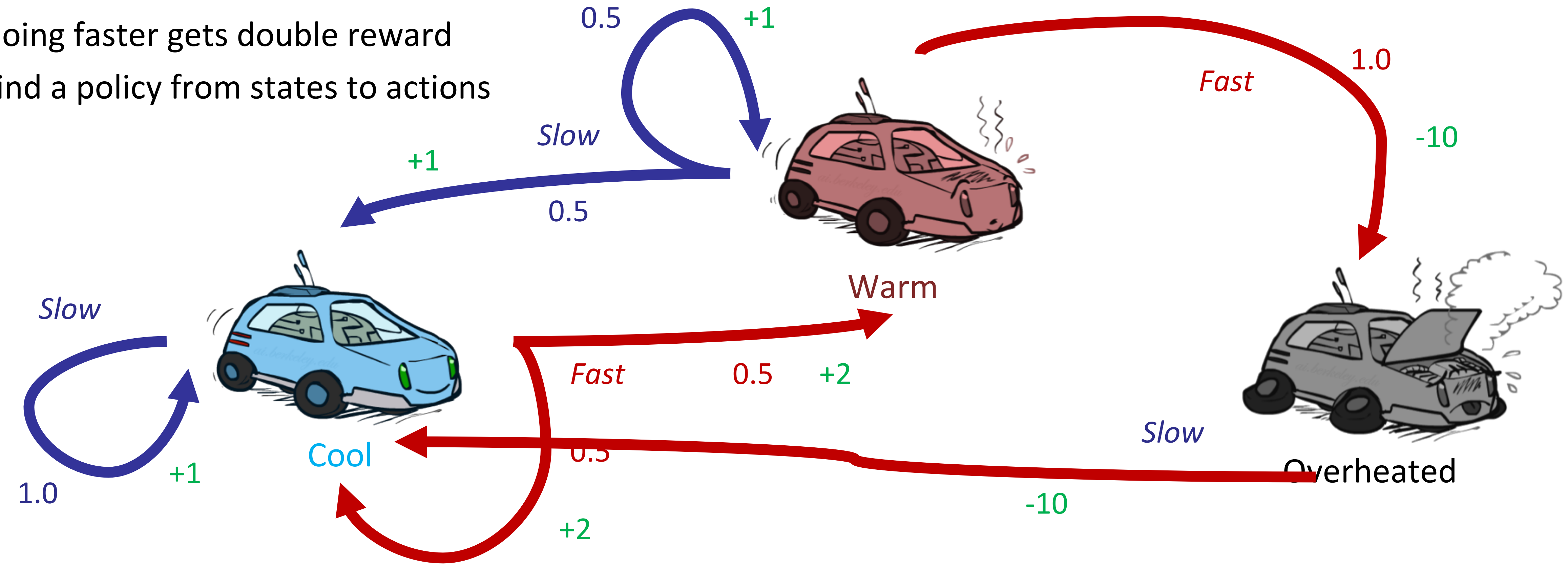
R(s) = -0.4



R(s) = -2.0

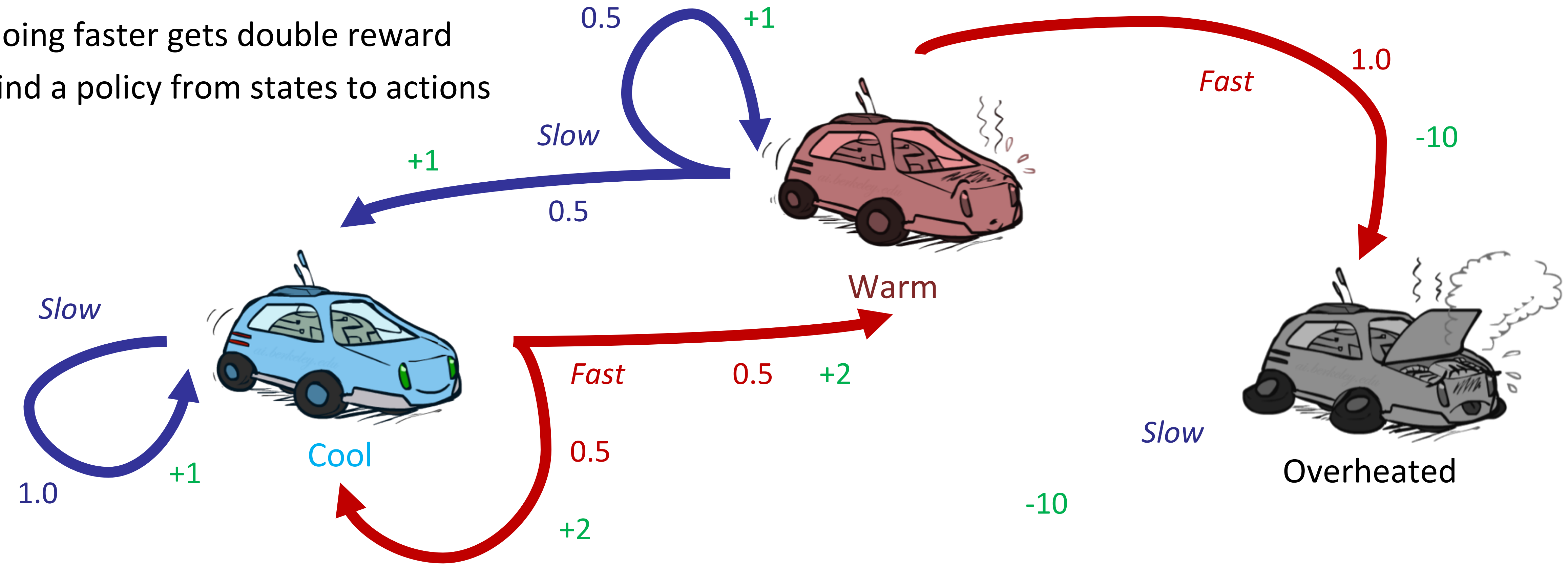
Example: Racing Car

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



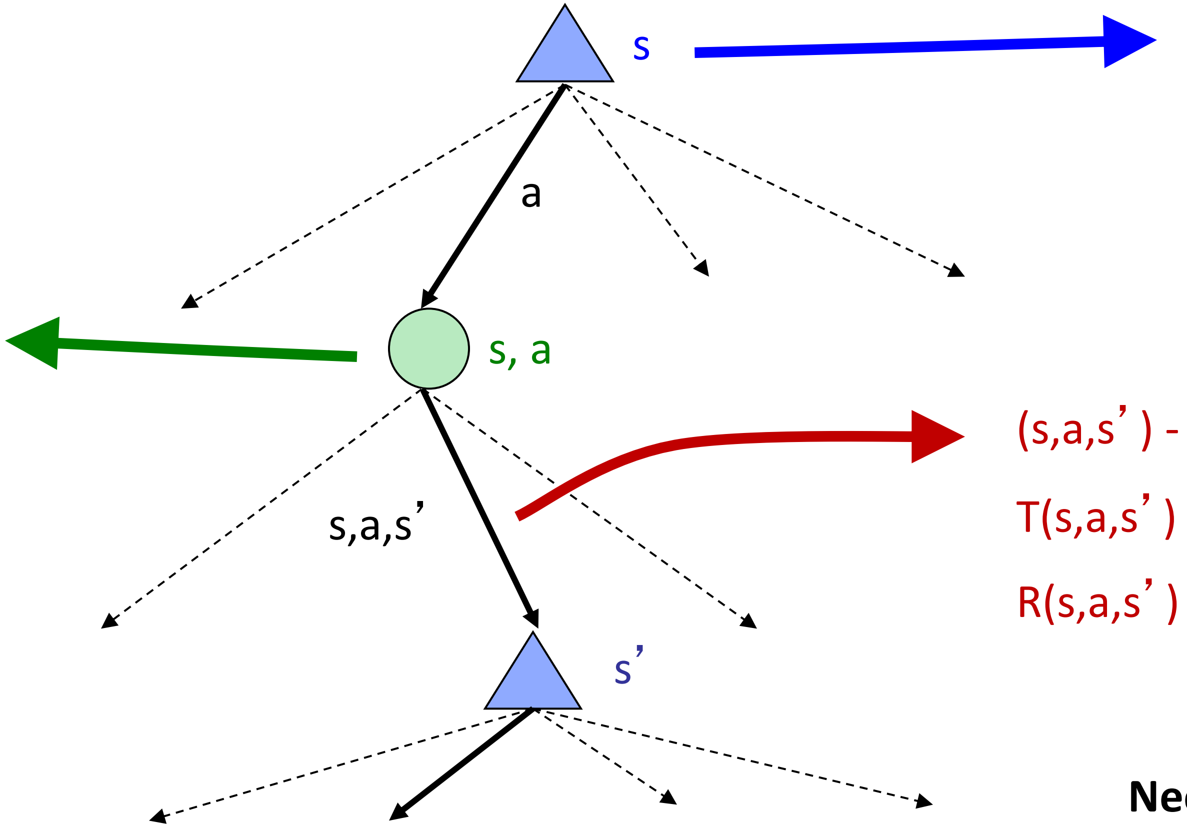
Example: Racing Car

- A robot car wants to travel far, quickly
- Three states: Cool, Warm, Overheated (terminal state)
- Two actions: *Slow*, *Fast*
- Going faster gets double reward
- Find a policy from states to actions



MDP as a Search Tree

Target: Need to find the optimal policy. The one that maximizes the expected utility if followed.



s is a *state*

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

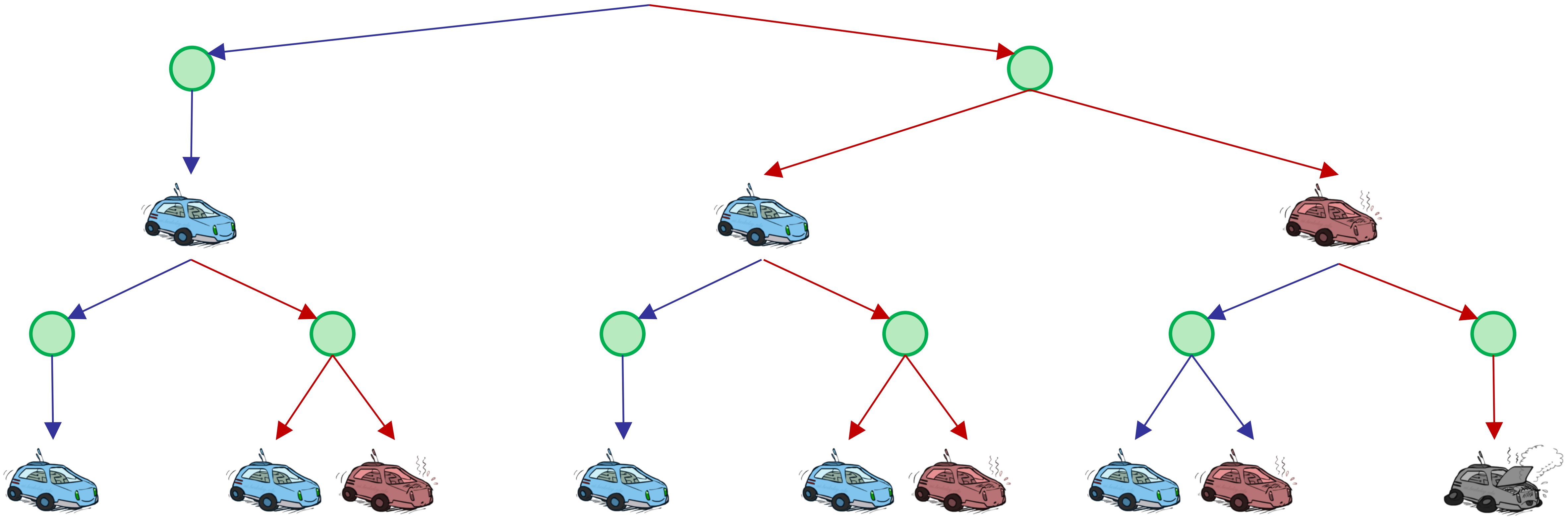
(s,a,s’) - *Transition*

T(s,a,s’) = P(s’ |s,a)

R(s,a,s’) – *Reward the agent gets*

Need a way to calculate the utility of a sequence of states!

Example: Racing Car



Visualizing an MDP as an Expectimax tree. Imagining the consequences of actions into the future.

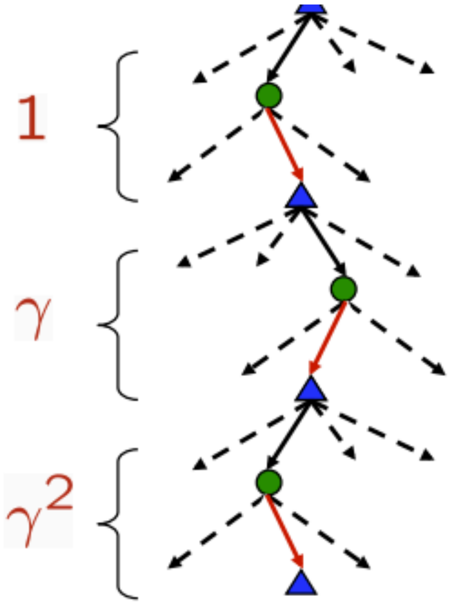
Utility of Reward Sequences

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]

Maximize the sum of rewards
Prefer rewards now to rewards later - discounting

- How to discount?
  - Each time we descend a level, 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])



Assigning Utilities to Reward Sequences

- **Additive utility:** $U([r_0, r_1, r_2, \ldots]) = r_0 + r_1 + r_2 + \cdots$
- **Discounted utility:** $U([r_0, r_1, r_2, \ldots]) = r_0 + \gamma r_1 + \gamma^2 r_2 \cdots$

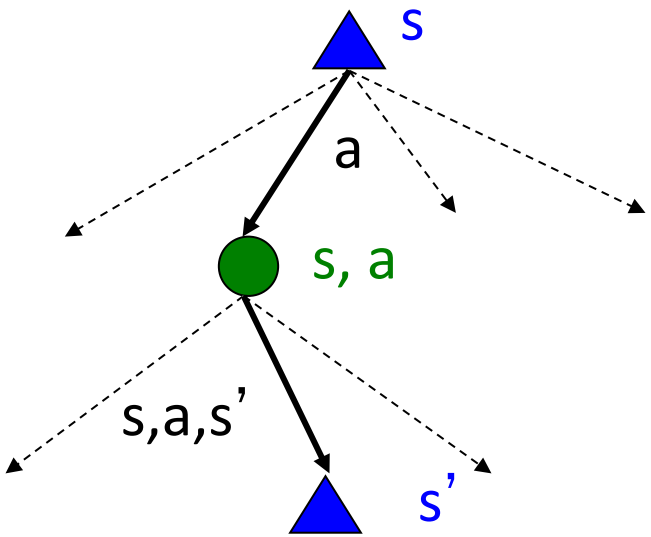
Discounting appears to be a good model for both animal and human preferences over time. Computationally, helps us converge utilities of infinite sequences.

With discounted rewards, the utility of an infinite sequence is finite.

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

MDP Formulation

- **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 quantities:**
  - Policy = Choice of action for each state
  - Utility = sum of (discounted) rewards
- ***Next: How to solve the MDP?***
  - *How to determine the optimal policy?*



Optimal Quantities

- **The value (utility) of a state s:**

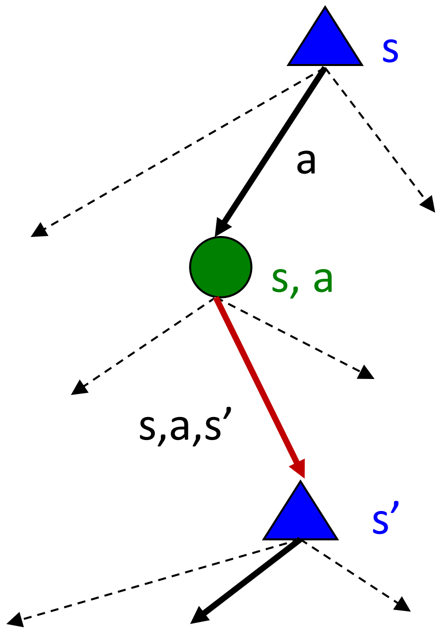
  $V^*(s)$ = expected utility starting in s and acting optimally

- **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

- **The optimal policy:**

  $\pi^*(s)$ = optimal action from state s



s is a *state*

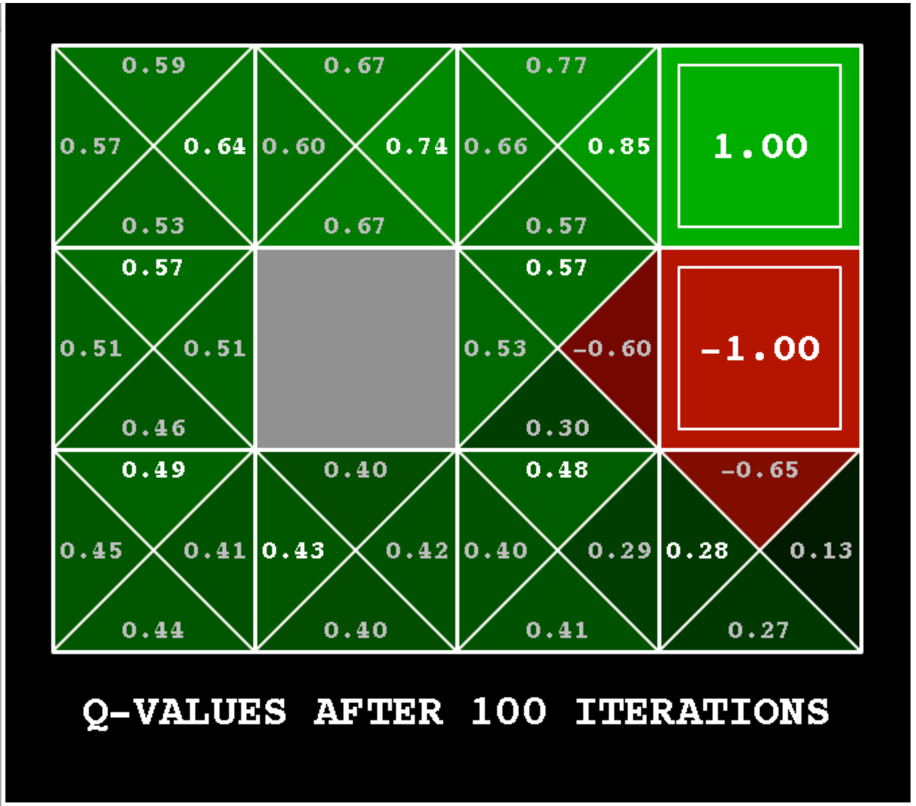
(s, a) is a *q-state*

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

Value Function Example

Value (utility) of states V(s) for all states

Value (utility) of a q-states Q(s,a) for all states and all actions at each state.



Noise = 0.2
Discount = 0.9
Living reward = 0

Value of States

- **Values of states are related to each other.**
- **Fundamental Operation**
  - Expected utility under optimal action for this state. What is the best we can do from this state?
  - Average sum of (discounted) rewards

- **Recursive definition of value:**

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

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

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

Bellman Equations

- Definition of **"optimal"** utility gives a simple **one-step** lookahead relationship amongst optimal values.

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

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

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

- The utility of a state is the **immediate reward** for that state plus the **expected discounted utility of the next state** assuming that the agent is acting **optimally**.

Value Iteration

- Calculate the **utility of each state** and then **use the state utility** to **select an optimal action** in each state.

- Bellman equations **characterize** the optimal values:

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

- 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 a **fixed-point solution** method

Value Iteration Algorithm

- **Start with $V_0(s) = 0$**

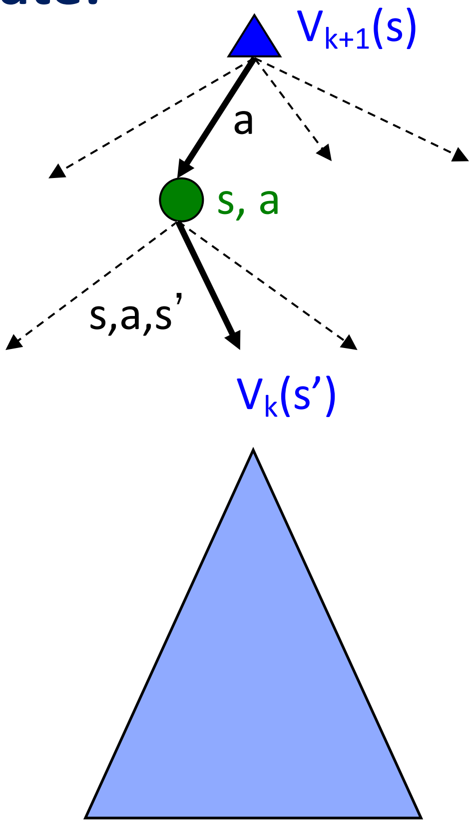
- **Given vector of $V_k(s)$ values, do one ply of expectimax from each state:**

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

- **Repeat until convergence**
  - Determine by looking at the max. change in utility of any state in an iteration.

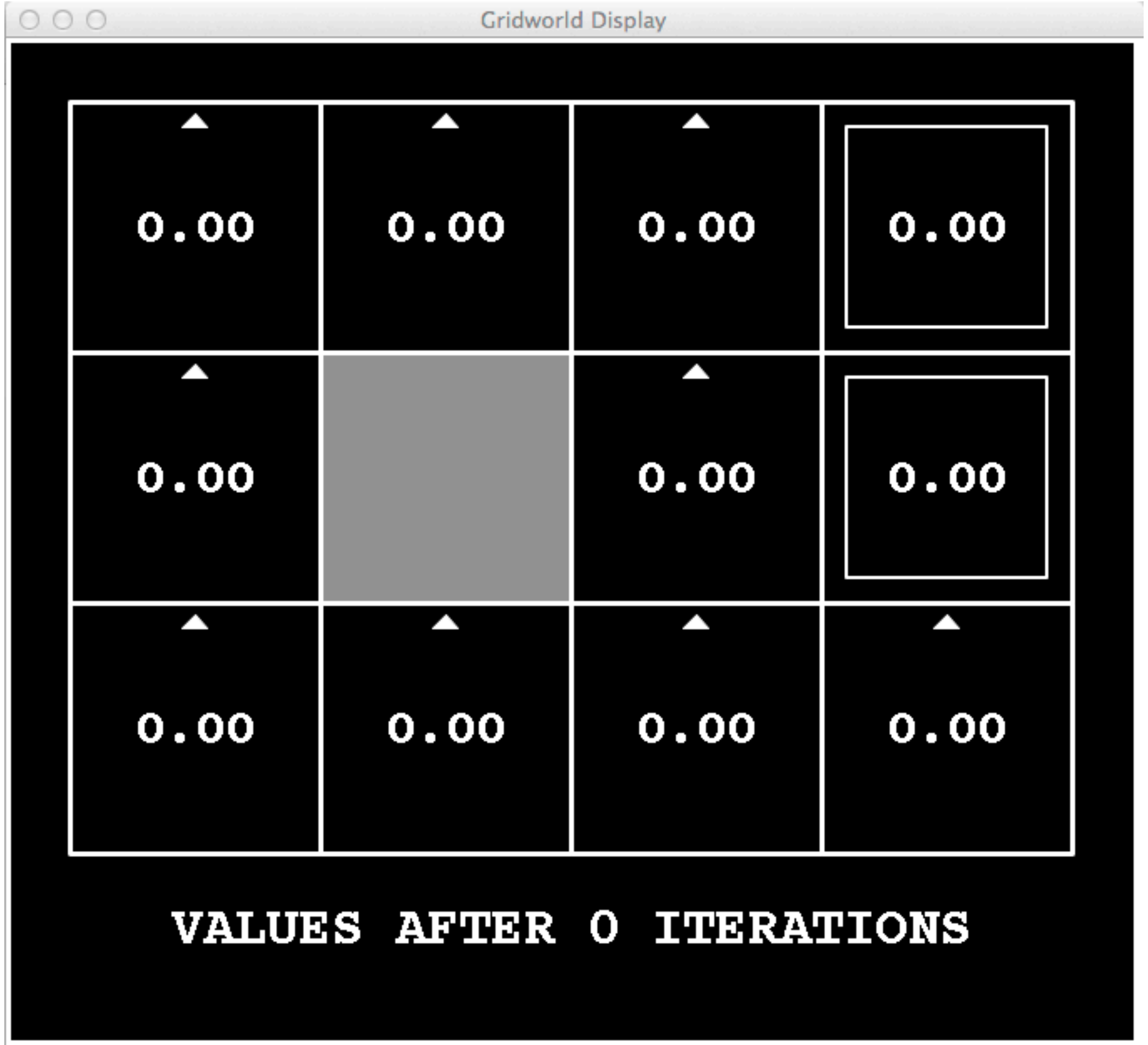
- Complexity of each iteration: **$O(S^2A)$**

- Theorem: will converge to unique optimal values
  - the start state does not matter



Value Iteration

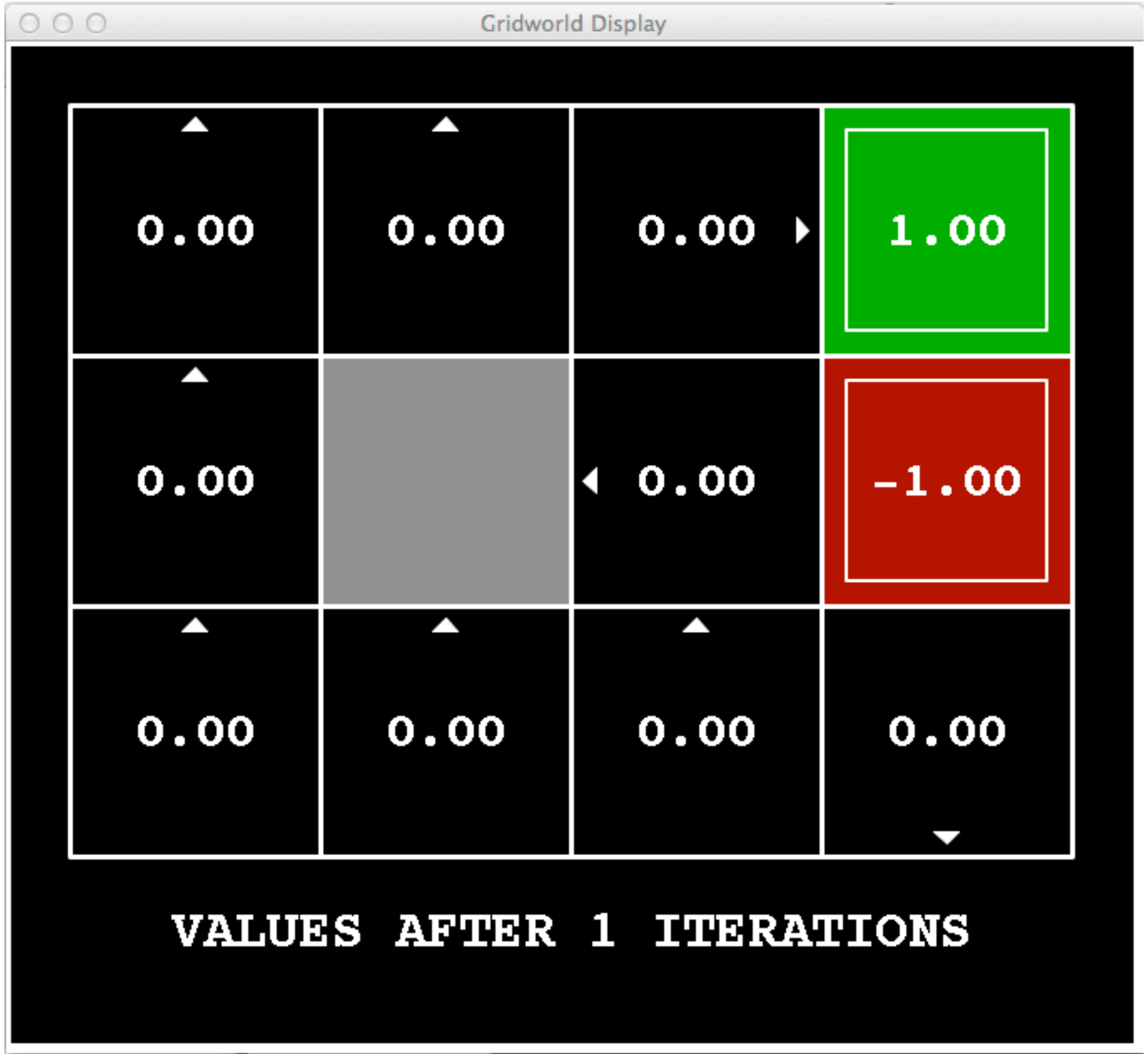
k=0



Noise = 0.2
Discount = 0.9
Living reward = 0

Value Iteration

k=1



In the first iteration the terminal states reflect the reward.

Noise = 0.2
Discount = 0.9
Living reward = 0

Value Iteration

k=2

Adjacent states start to get updated.

Noise = 0.2
Discount = 0.9
Living reward = 0

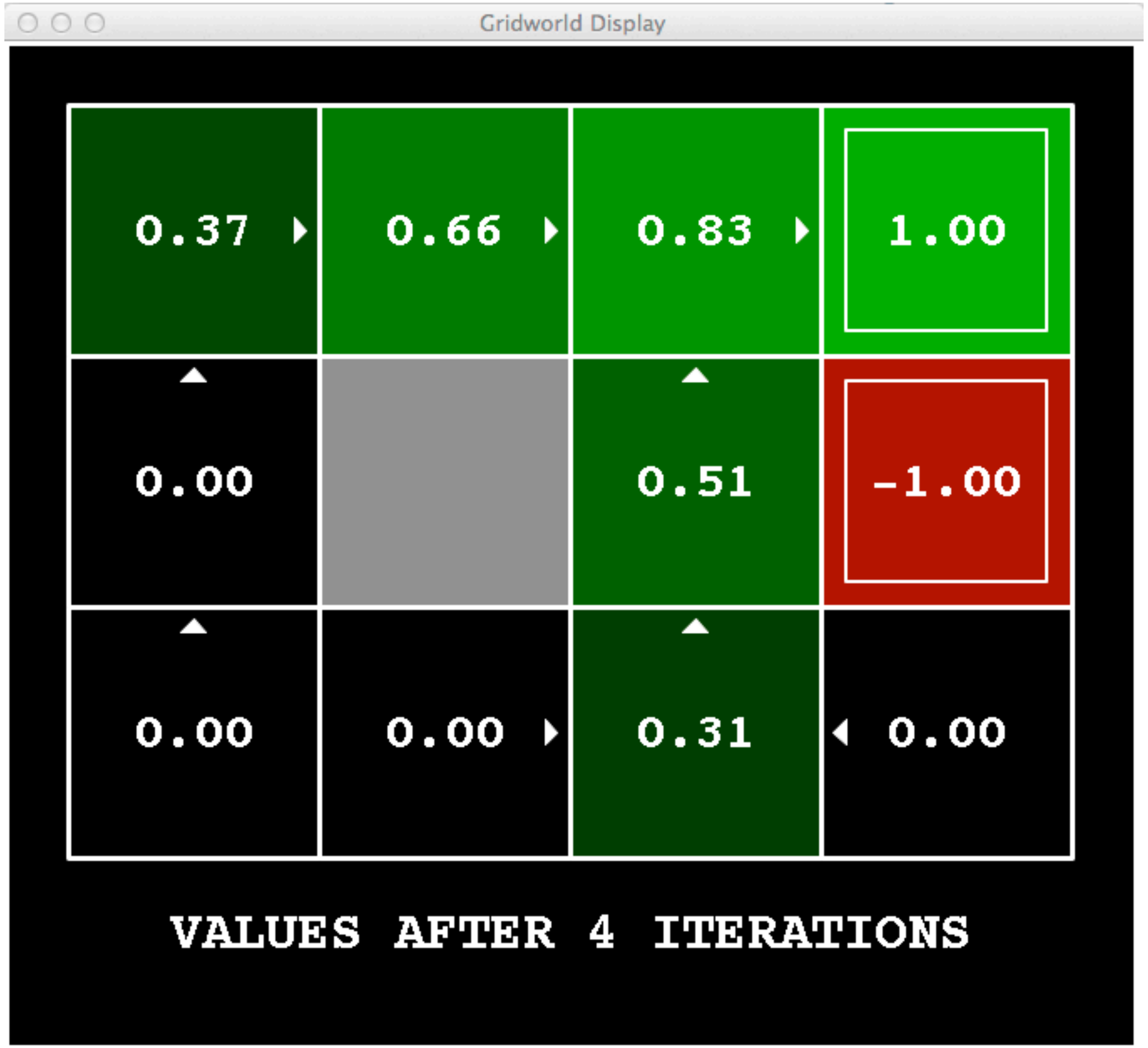
Value Iteration

k=3

Noise = 0.2
Discount = 0.9
Living reward = 0

Value Iteration

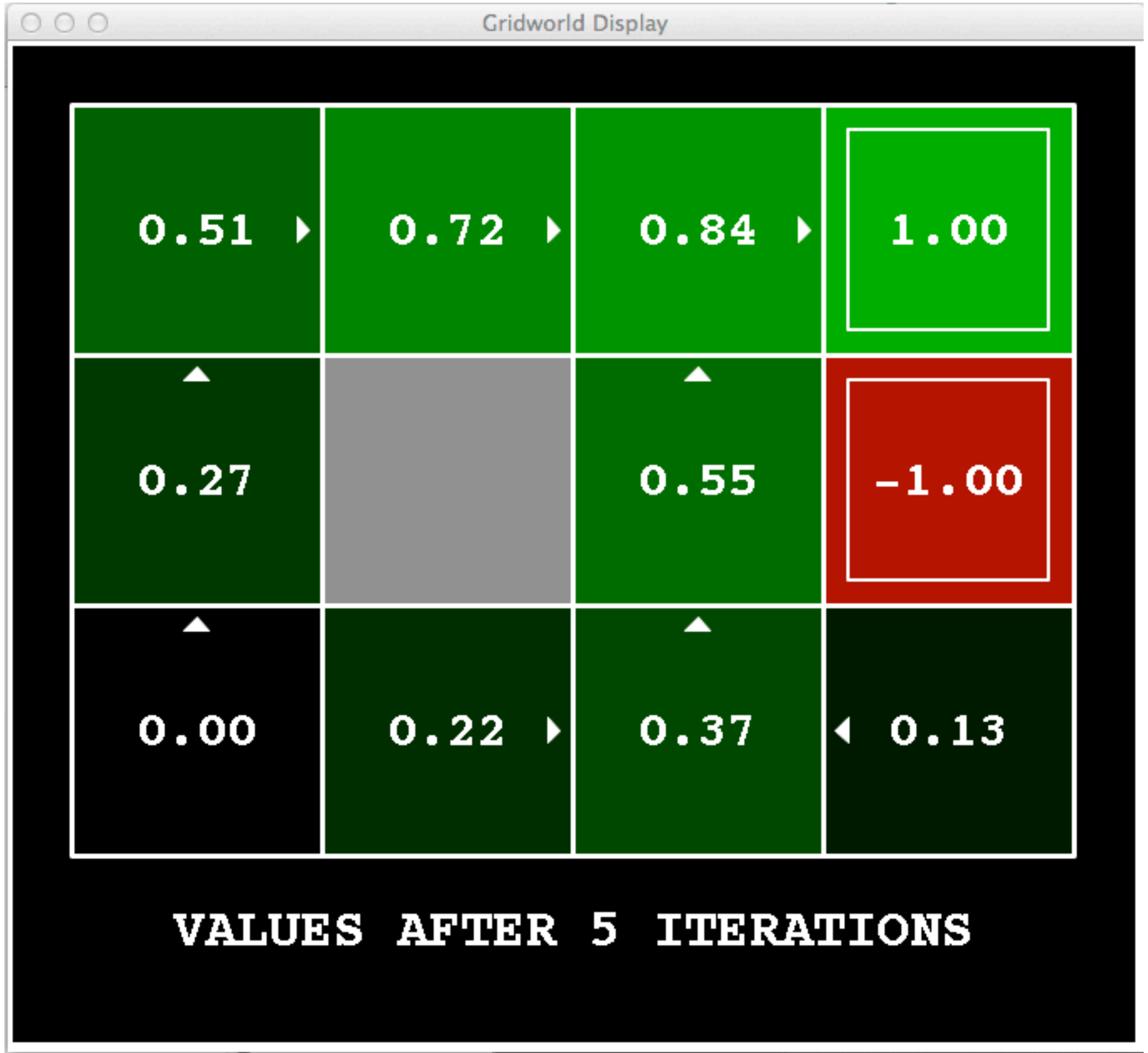
k=4



Noise = 0.2
Discount = 0.9
Living reward = 0

Value Iteration

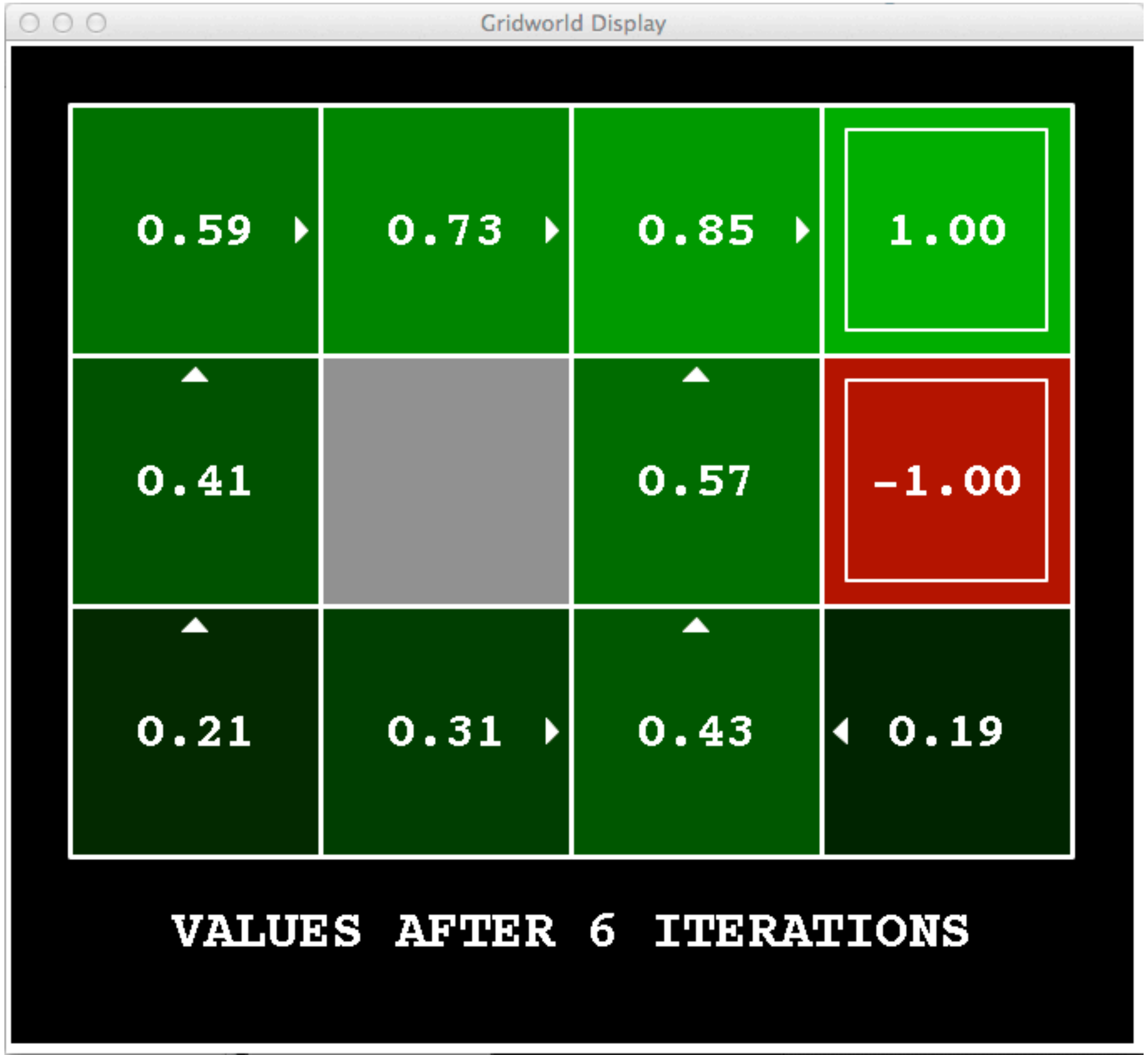
k=5



Noise = 0.2
Discount = 0.9
Living reward = 0

Value Iteration

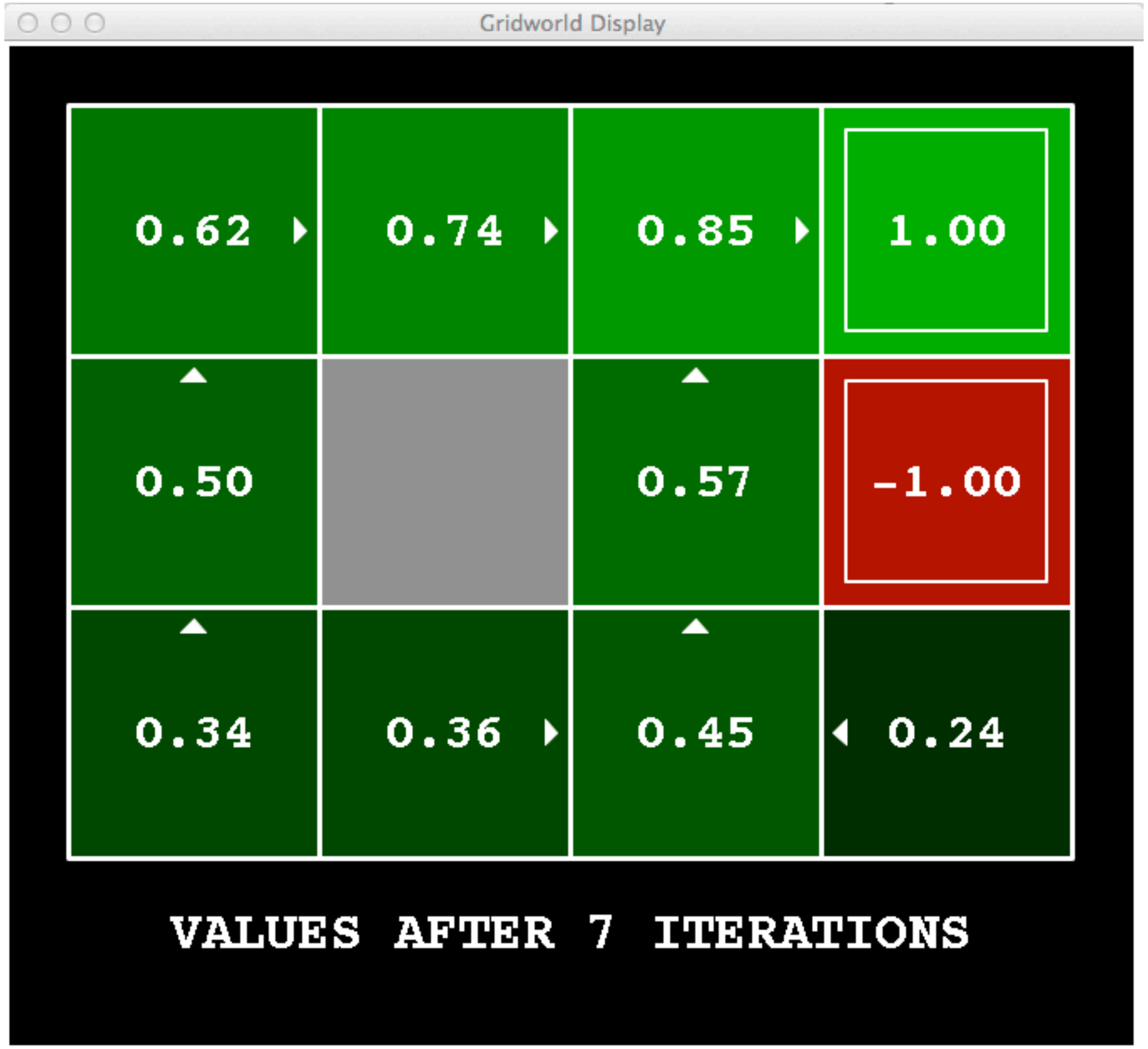
k=6



Noise = 0.2
Discount = 0.9
Living reward = 0

Value Iteration

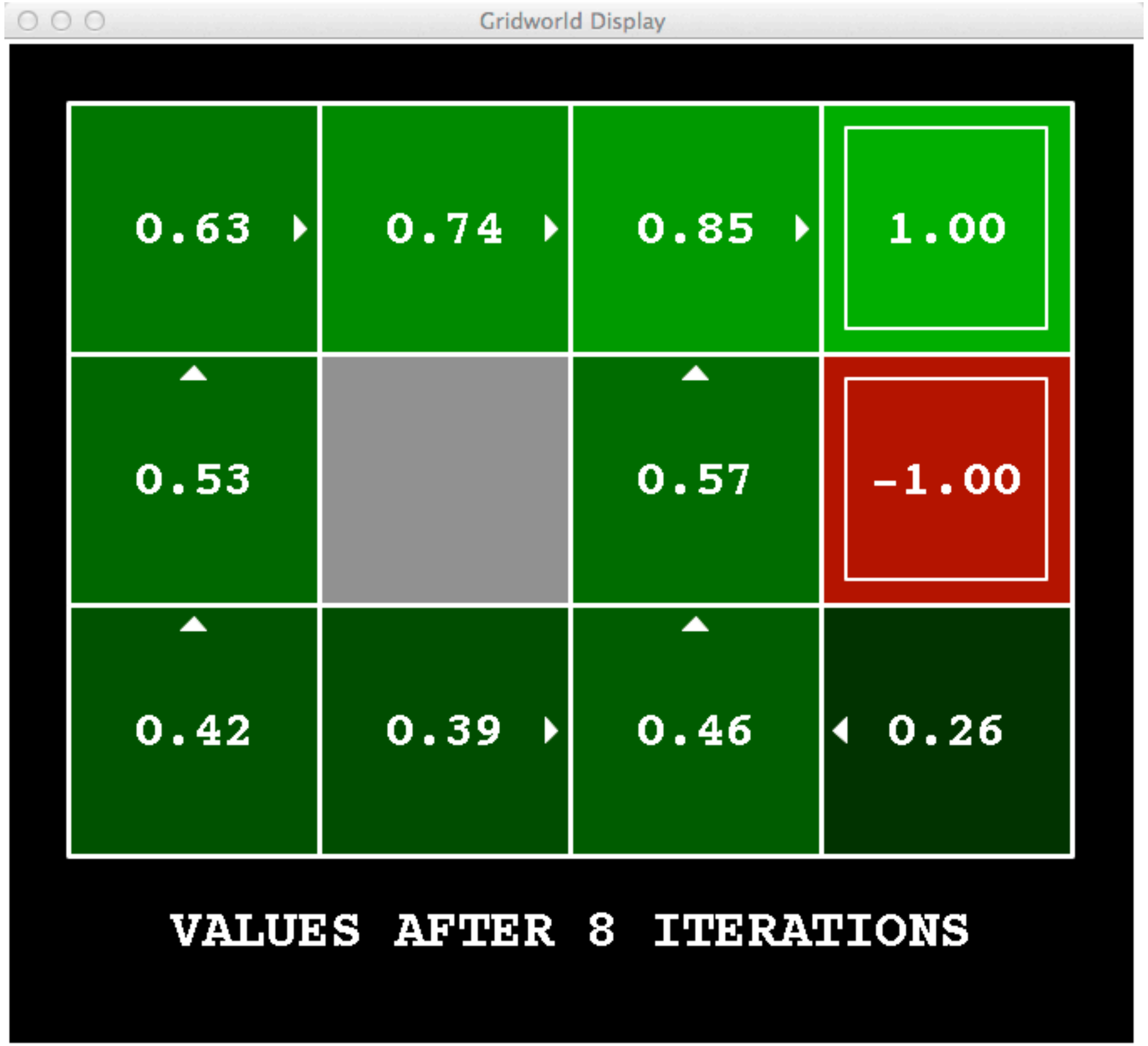
k=7



Noise = 0.2
Discount = 0.9
Living reward = 0

Value Iteration

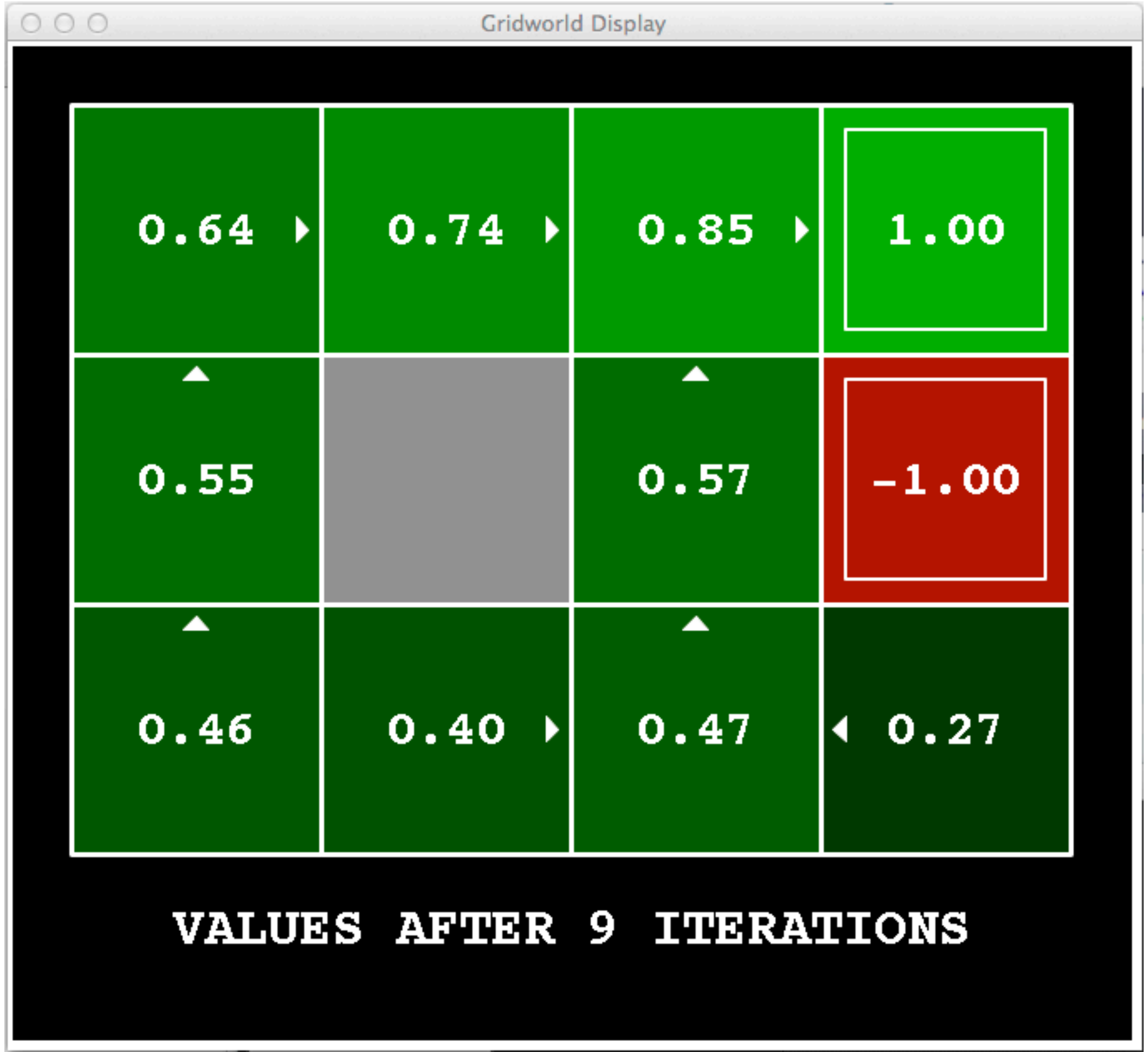
k=8



Noise = 0.2
Discount = 0.9
Living reward = 0

Value Iteration

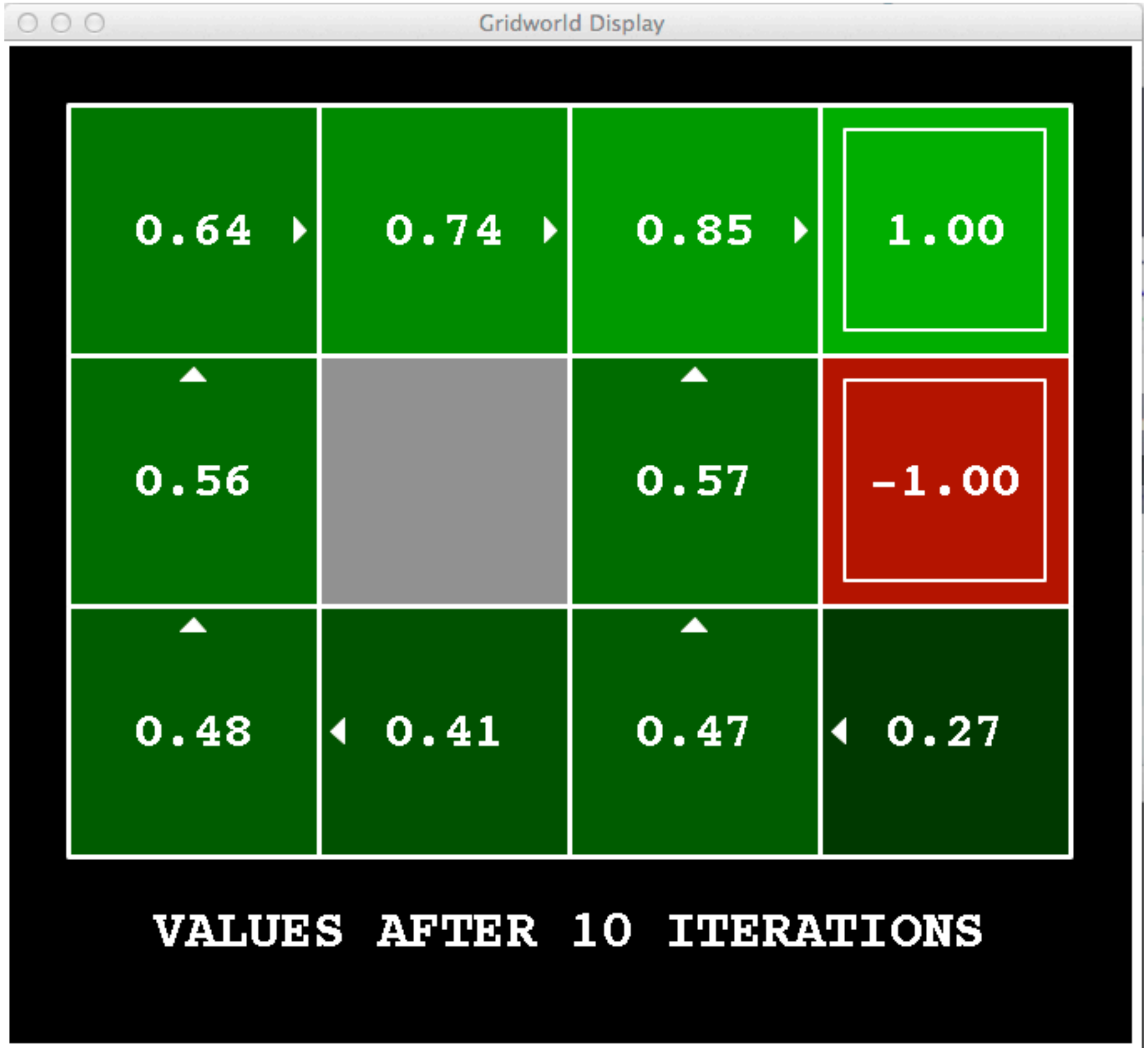
k=9



Noise = 0.2
Discount = 0.9
Living reward = 0

Value Iteration

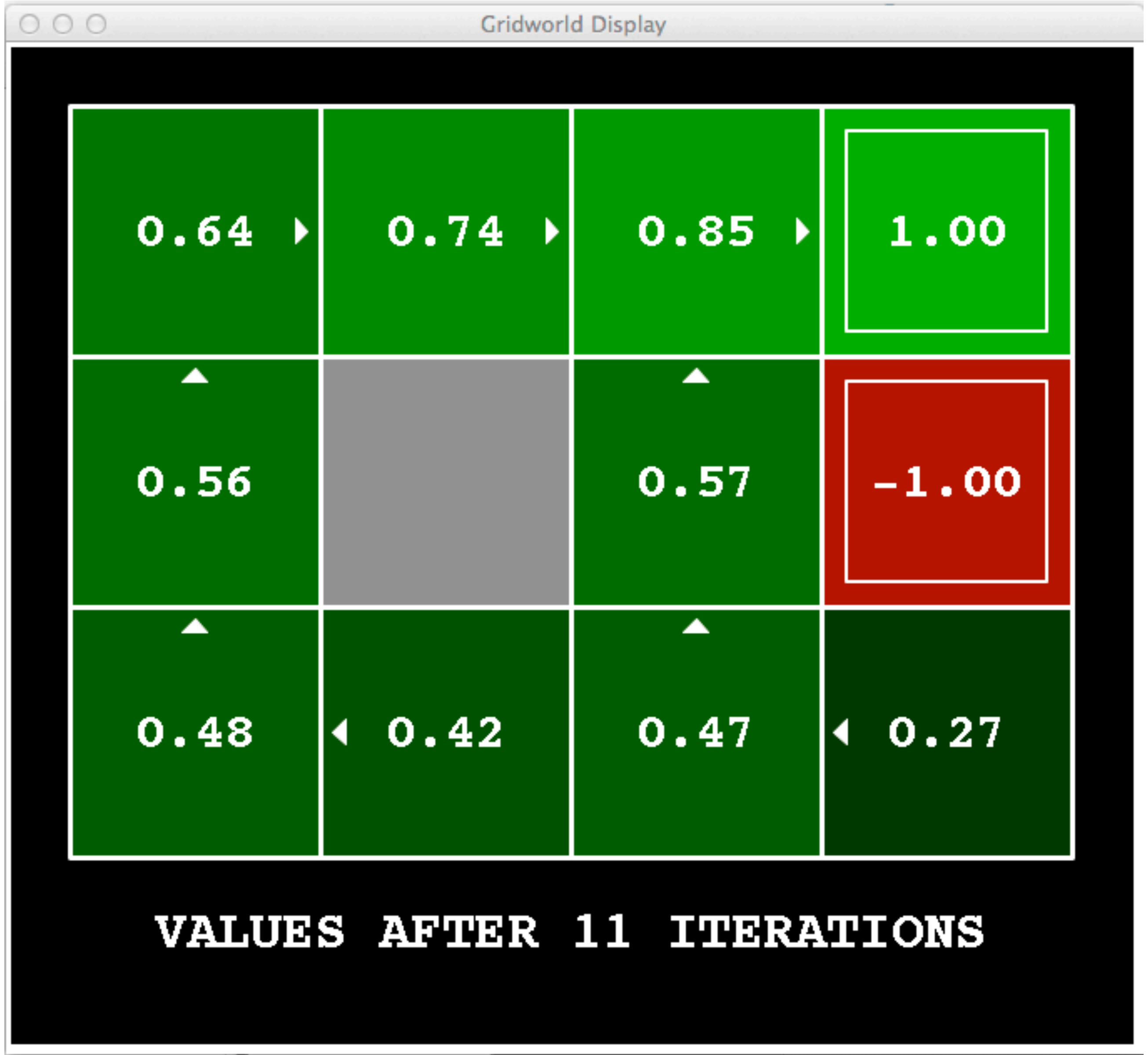
k=10



Noise = 0.2
Discount = 0.9
Living reward = 0

Value Iteration

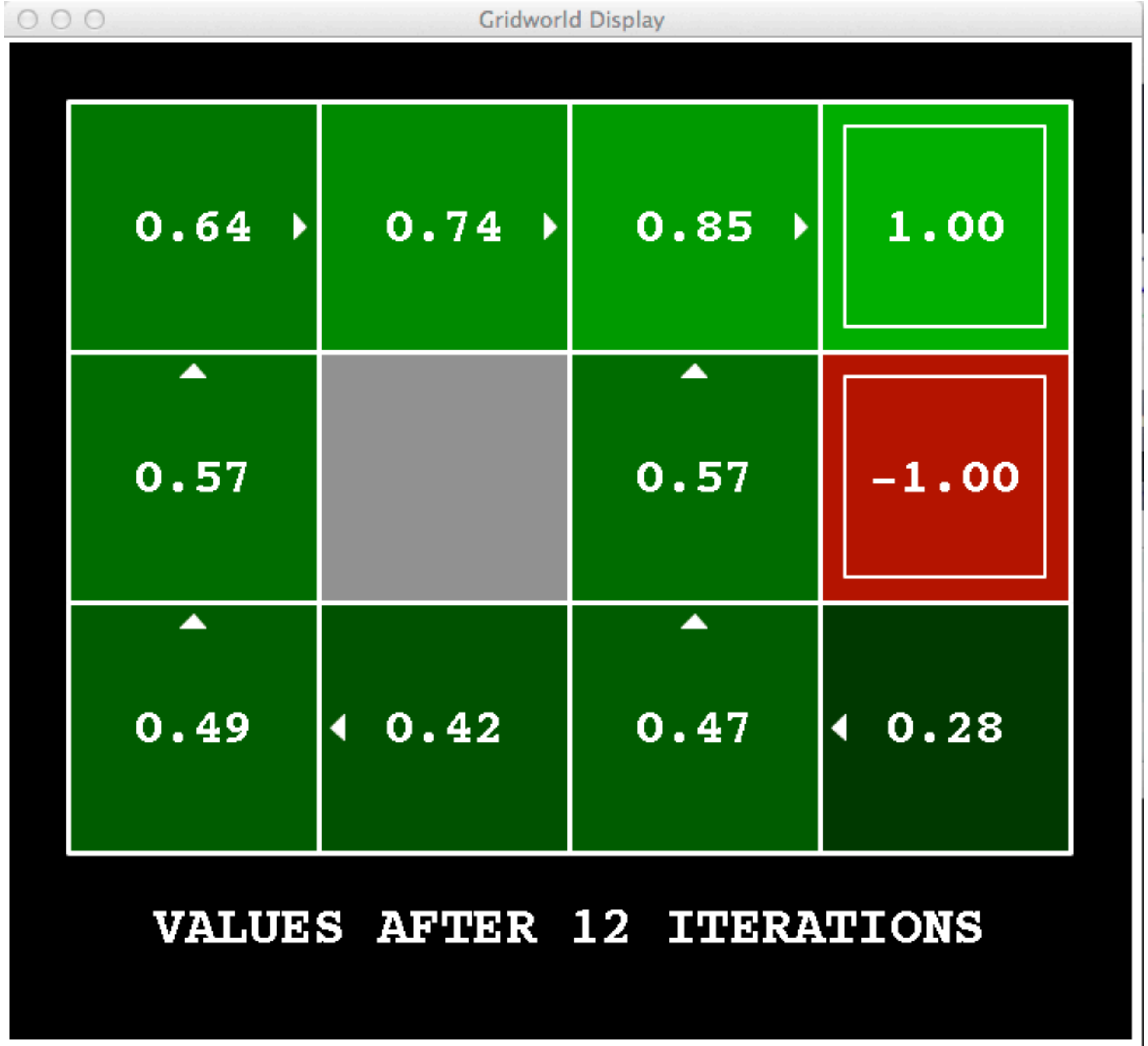
k=11



Noise = 0.2
Discount = 0.9
Living reward = 0

Value Iteration

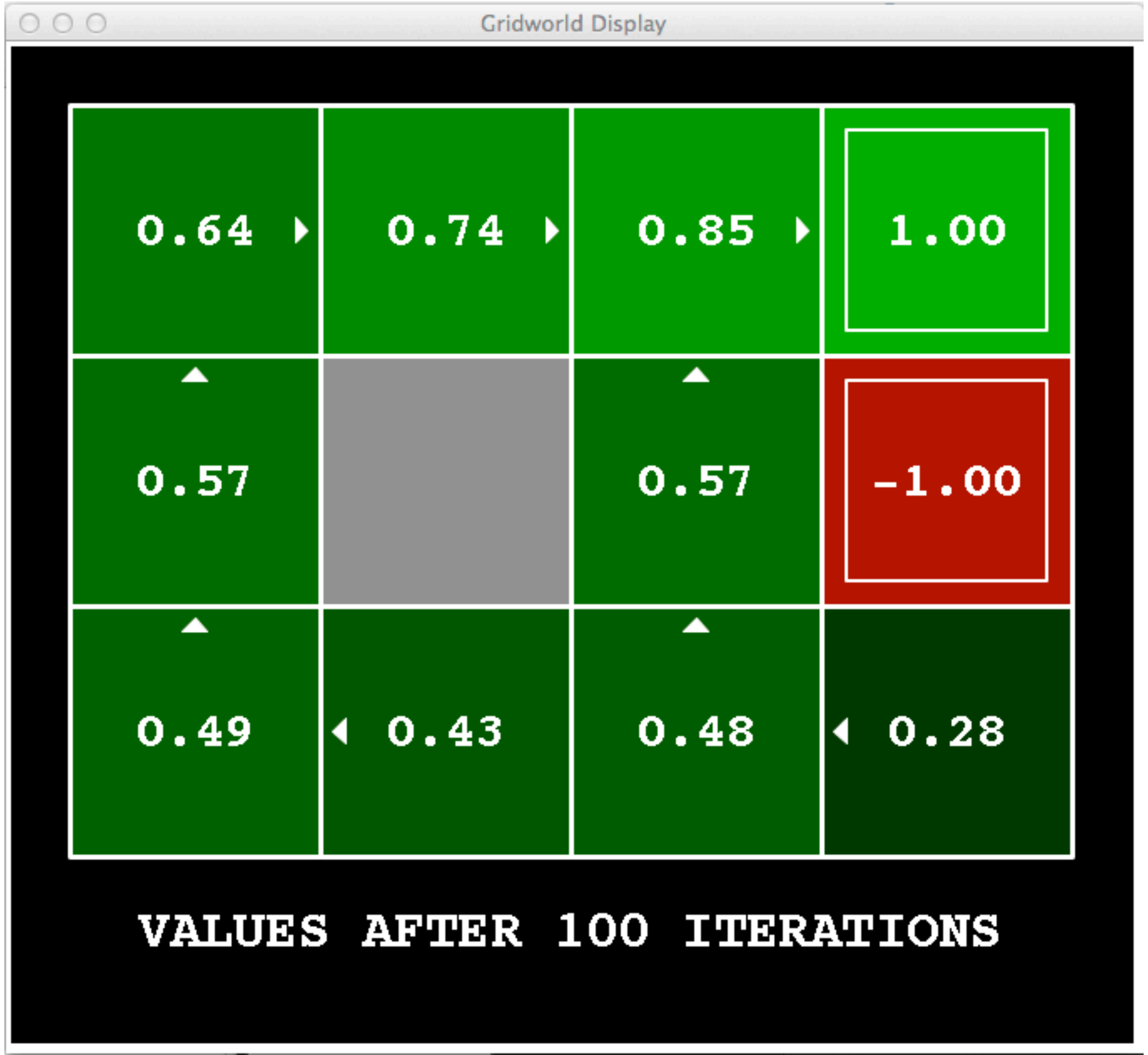
k=12



Noise = 0.2
Discount = 0.9
Living reward = 0

Value Iteration

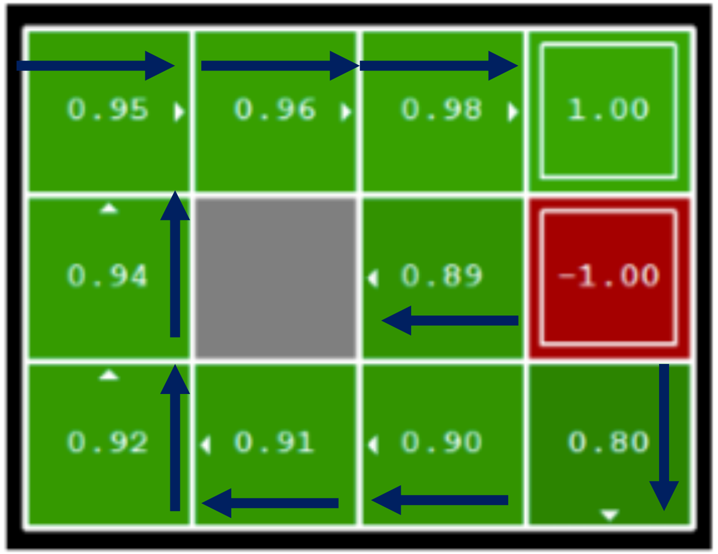
k=100



Noise = 0.2
Discount = 0.9
Living reward = 0

Extracting Policy from the Optimal Value Function

- **Our goal is to determine the policy for the MDP**
- **Step I: Estimate the optimal values V*(s)**
  - Through Value Iteration algorithm
- **Step II: Policy Extraction**
  - Obtain the policy implied by the values (using 1-step look ahead).
  - Use the policy to act in the environment.



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