CSE 573 p: Artificial Intelligence

Hanna Hajishirzi

Markov Decision Processes

slides adapted from
Dan Klein, Pieter Abbeel ai.berkeley.edu
And Dan Weld, Luke Zettlemoyer

Announcements

- HW1: Jan 28.
- PS2: Feb 4th
- Next class: Vote?
- Project proposals: Feb 11th
- Paper review: Feb 18th

Project Proposal

- Project proposals: Feb 11th
  - Pick projects close to you interests, or select from here: list of potential projects. Your final project can also be a re-implementation of one of the recent papers from Robotics/ AI/ML/NLP/Computer vision conferences.
- The project proposal is a 1-page summary of the project topic, motivation, definition, dataset, and resources. It should also include the milestones, detailed experiment plan, and the timeline to complete each milestone.

Paper Review

- Paper review:
  - 1. Describe what problem or question this paper addresses, and the main contributions that it makes towards a solution or answer. a. Problem/Question:
  - b. Solution/approach:
  - c. Contributions (list at least two):
- 2. Evaluate the paper in terms of novelty, significance, and empirical results. 3. Describe the main strengths you see in the paper. 4. Describe critiques and weaknesses you see in the paper.

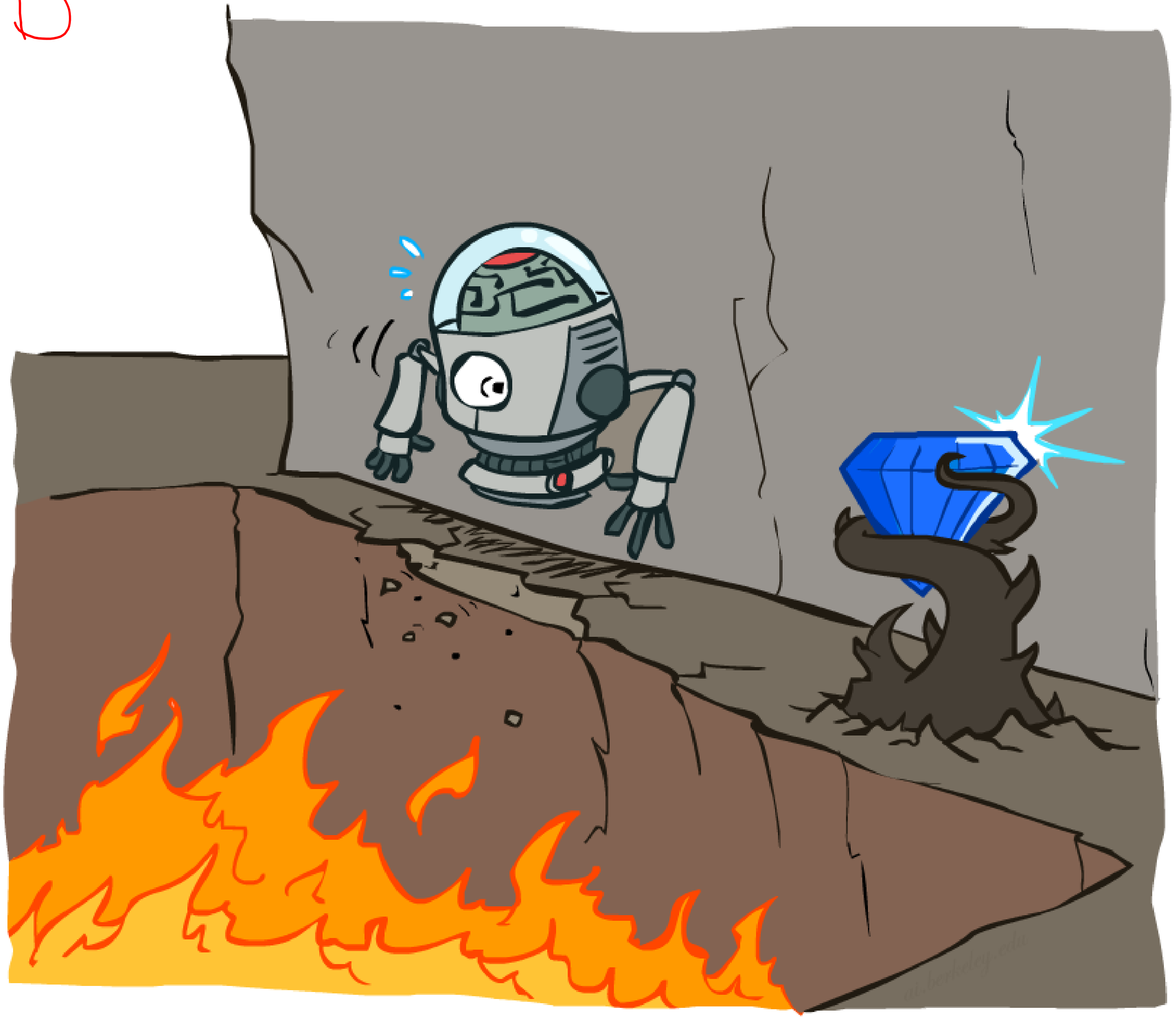
Review and Outline

- Adversarial Games
  - Minimax search
  - α-β search
  - Evaluation functions
  - Multi-player, non-0-sum
- Stochastic Games
  - Expectimax
  - Markov Decision Processes
  - Reinforcement Learning

Non-Deterministic Search

A

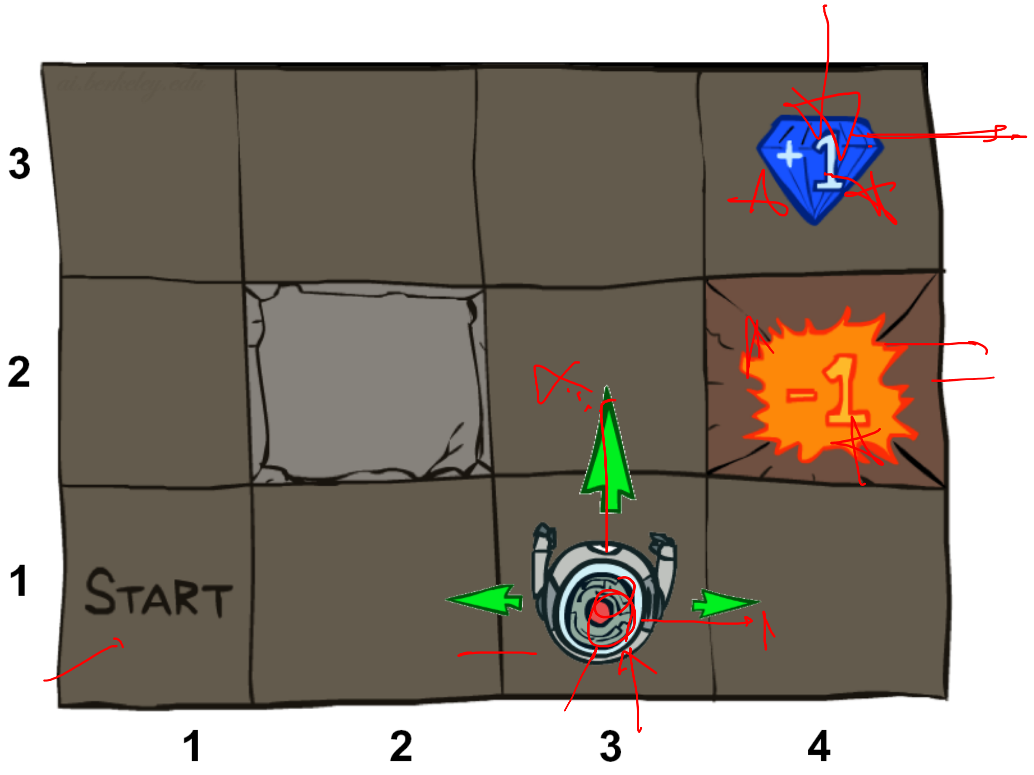
B

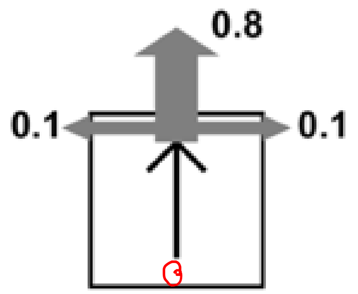


Step —— goal

Example: Grid World

- A maze-like problem
  - The agent lives in a grid
  - Walls block the agent’s path
- Noisy movement: 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)
  - Big rewards come at the end (good or bad)
- Goal: maximize sum of rewards

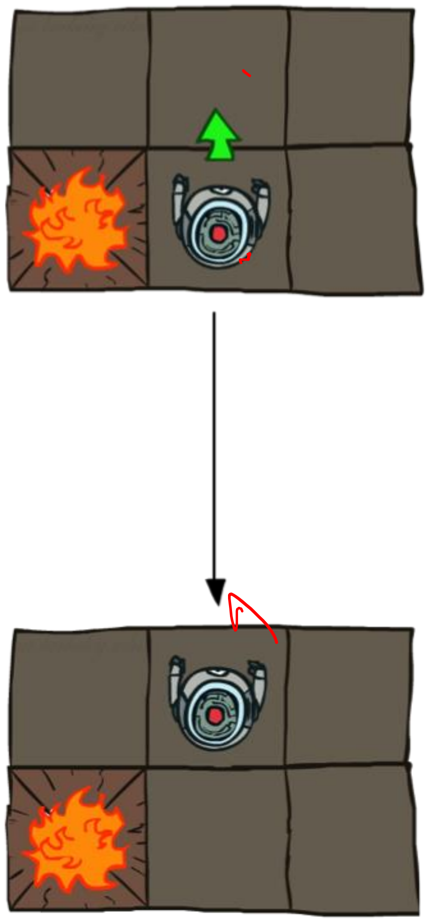




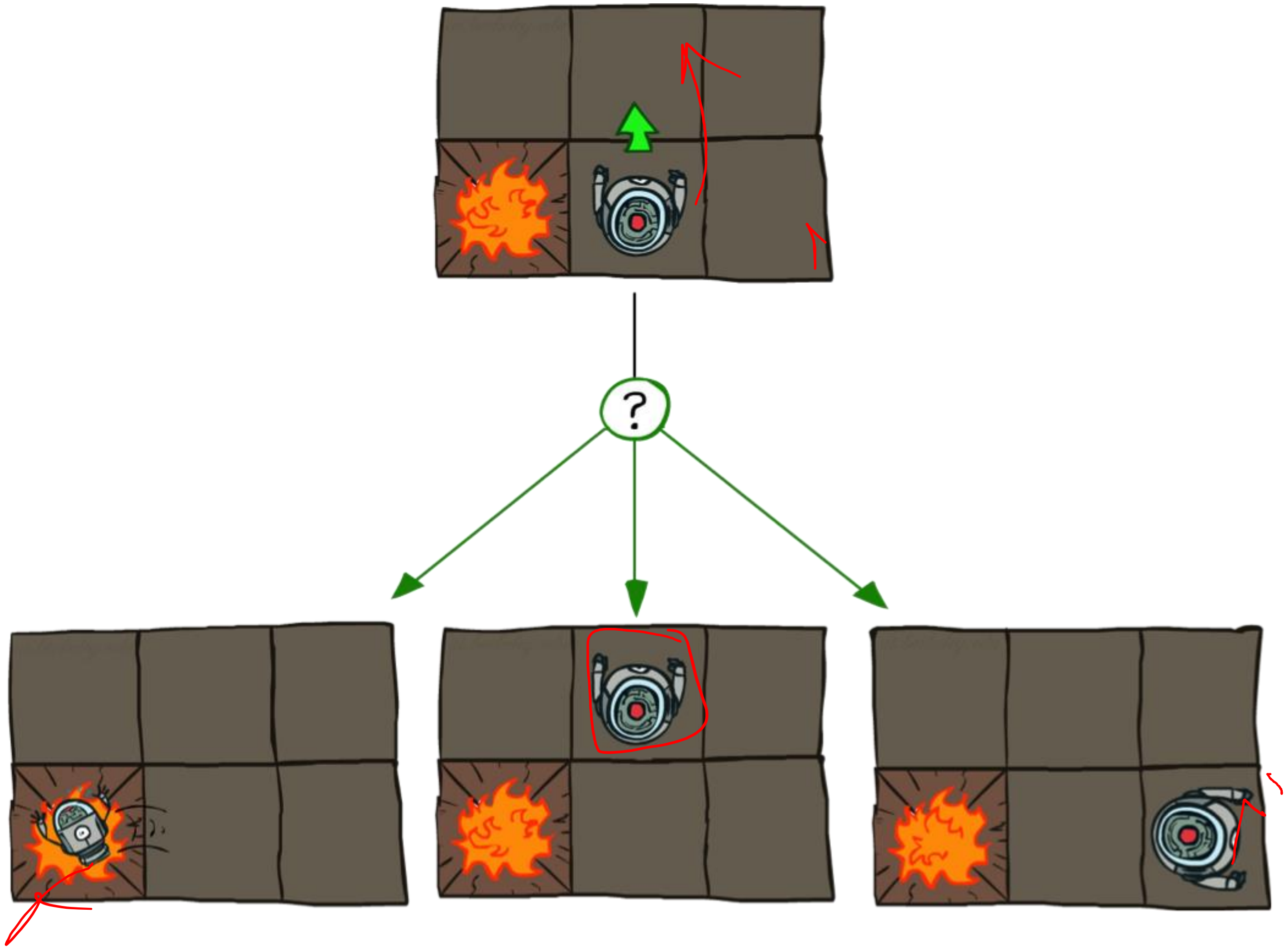
-0.01

Grid World Actions

Deterministic Grid World



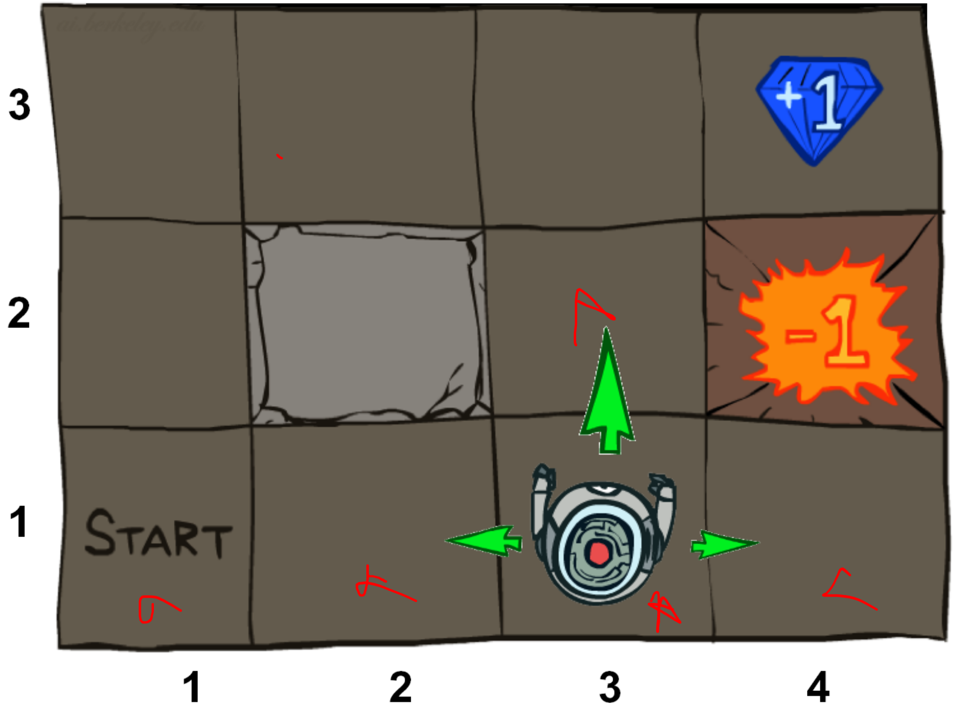
Stochastic Grid World



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

P(s'|s,a)



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

…

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

…

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

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

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

…

Tis a Big Table!

11 X 4 x 11 = 484 entries

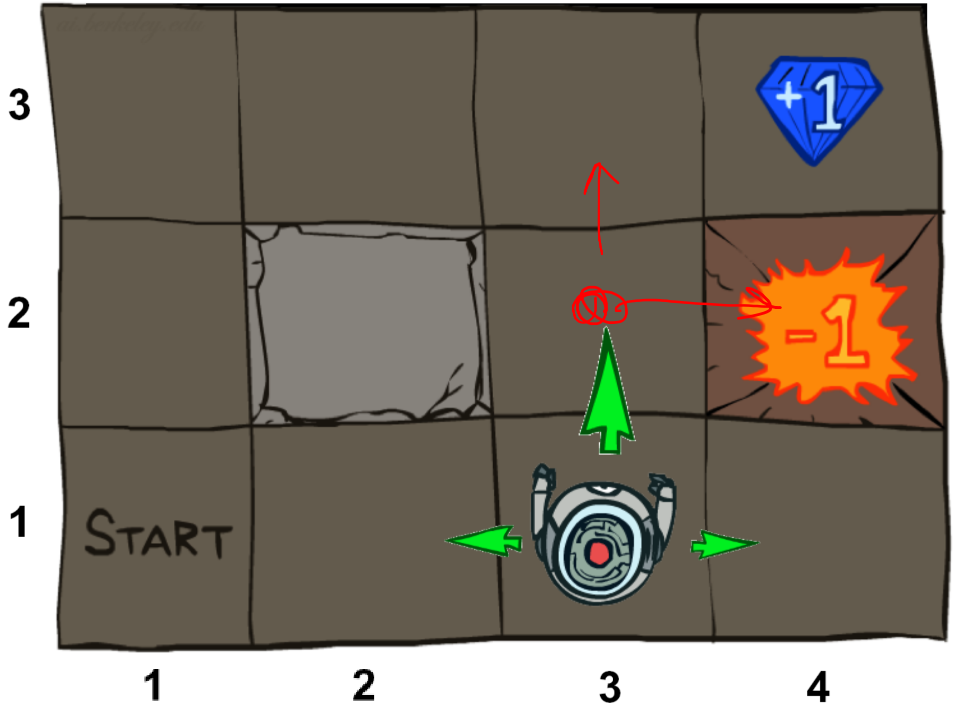
|S| x |A| x |S|

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 $R(s, a, s')$
    - Sometimes just $R(s)$ or $R(s')$

$R(s, a)$



…

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

…

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

…

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

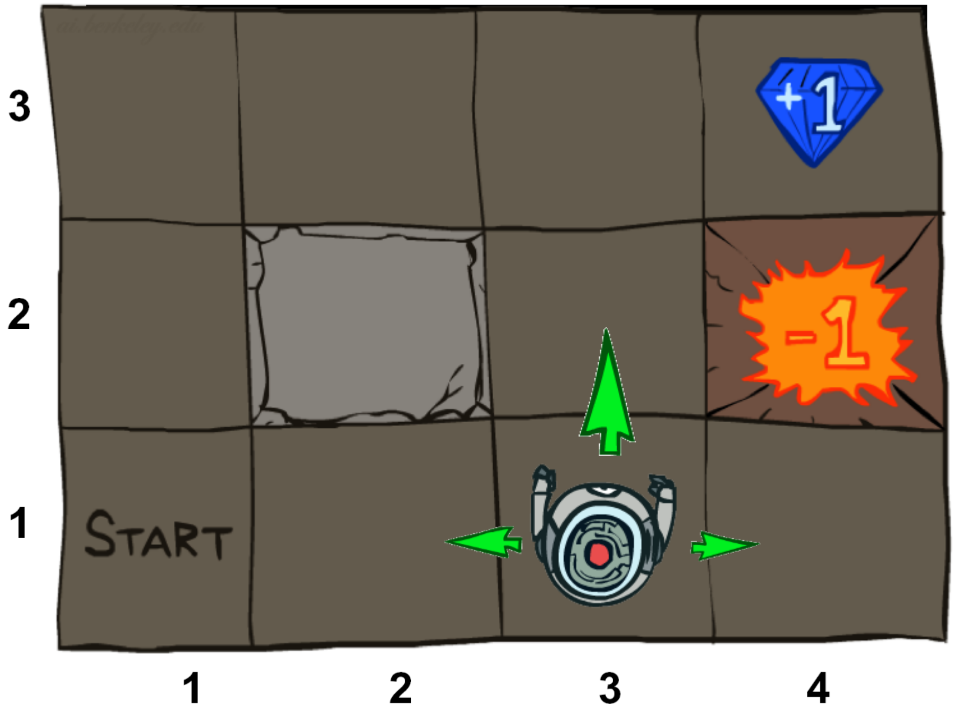
Cost of breathing

R is also a Big Table!

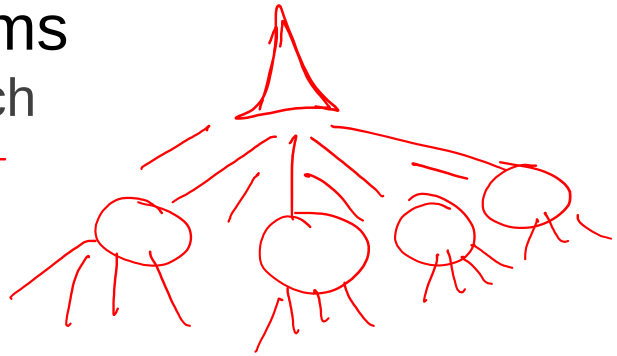
For now, we also give this to the agent

Markov Decision Processes

- An MDP is defined by:
  - A set of states s ∈ S
  - A set of actions a ∈ 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



- MDPs are non-deterministic search problems
  - One way to solve them is with expectimax search
  - We’ll have a new tool soon



What is Markov about MDPs?

- “Markov” generally means that 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

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

$$=$$

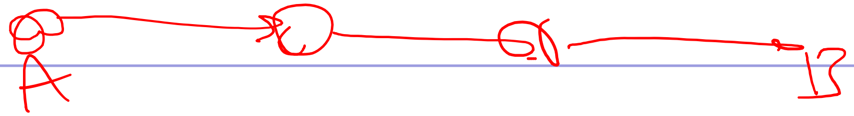
$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t)$$

- This is just like search, where the successor function could only depend on the current state (not the history)

Andrey Markov
(1856-1922)

Policies

S → → → ↗ G



- In deterministic single-agent search problems, we wanted an optimal plan, or sequence of actions, from start to a goal

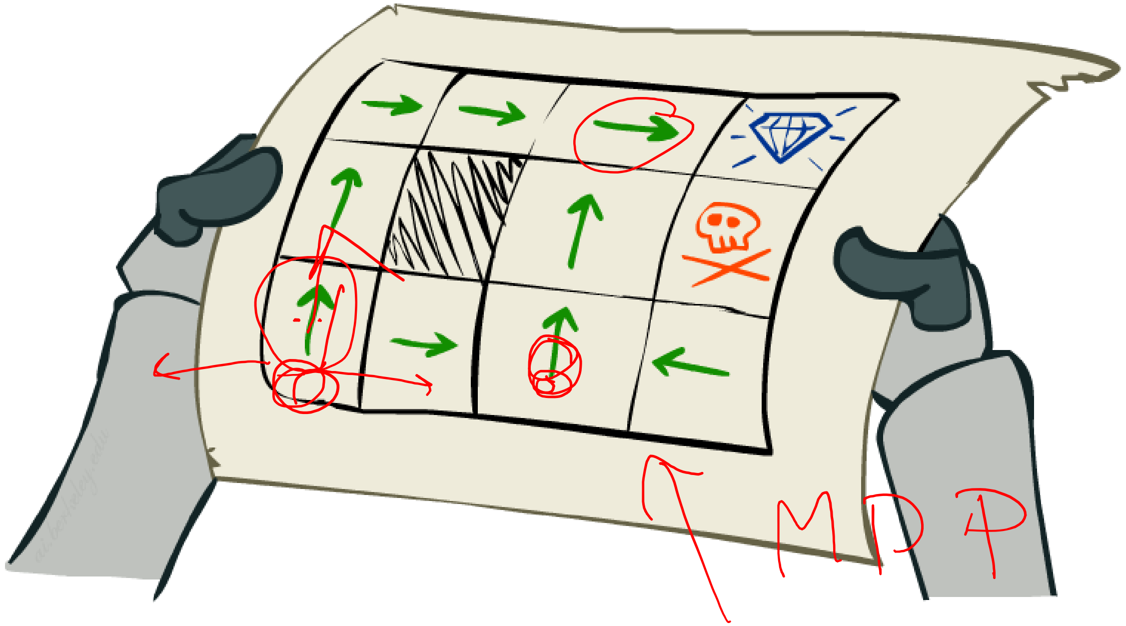
(S, A, T, R)

- For MDPs, we want an optimal

  policy π*: S → A

  π*

  - A policy π gives an action for each state
  - An optimal policy is one that maximizes expected utility if followed
  - An explicit policy defines a reflex agent

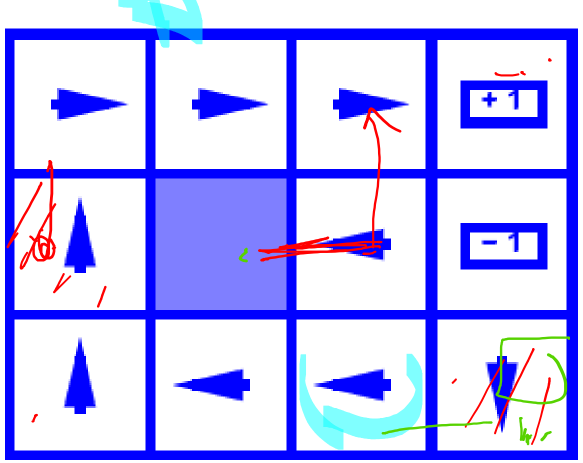


Optimal policy when R(s, a, s’) = -0.4 for all non-terminals s

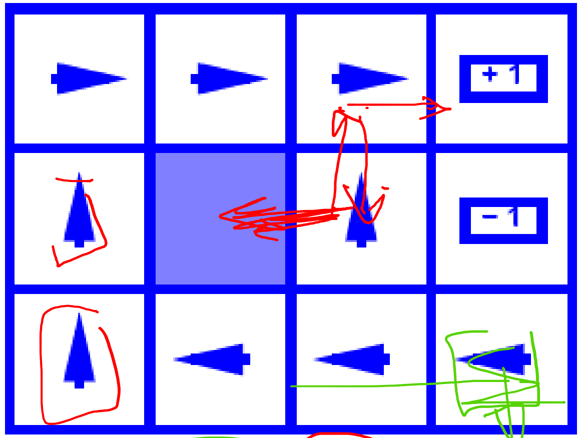
Optimal Policies

$T(s,a,s') <$ 0.8
0.1
0.1

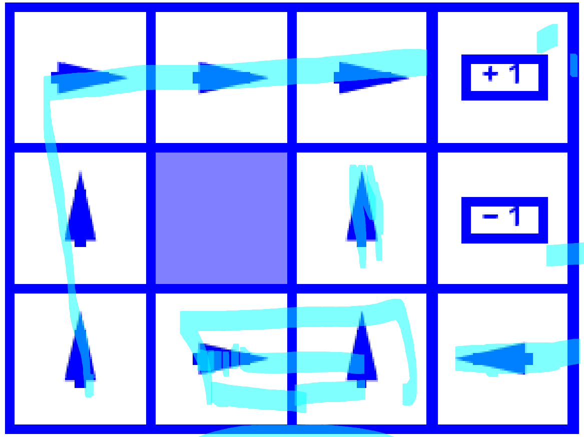
0.8
0.1
0.1



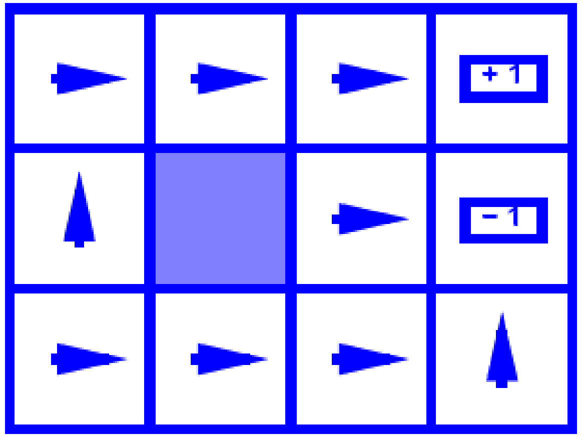
R(s) = -0.01



R(s) = -0.03

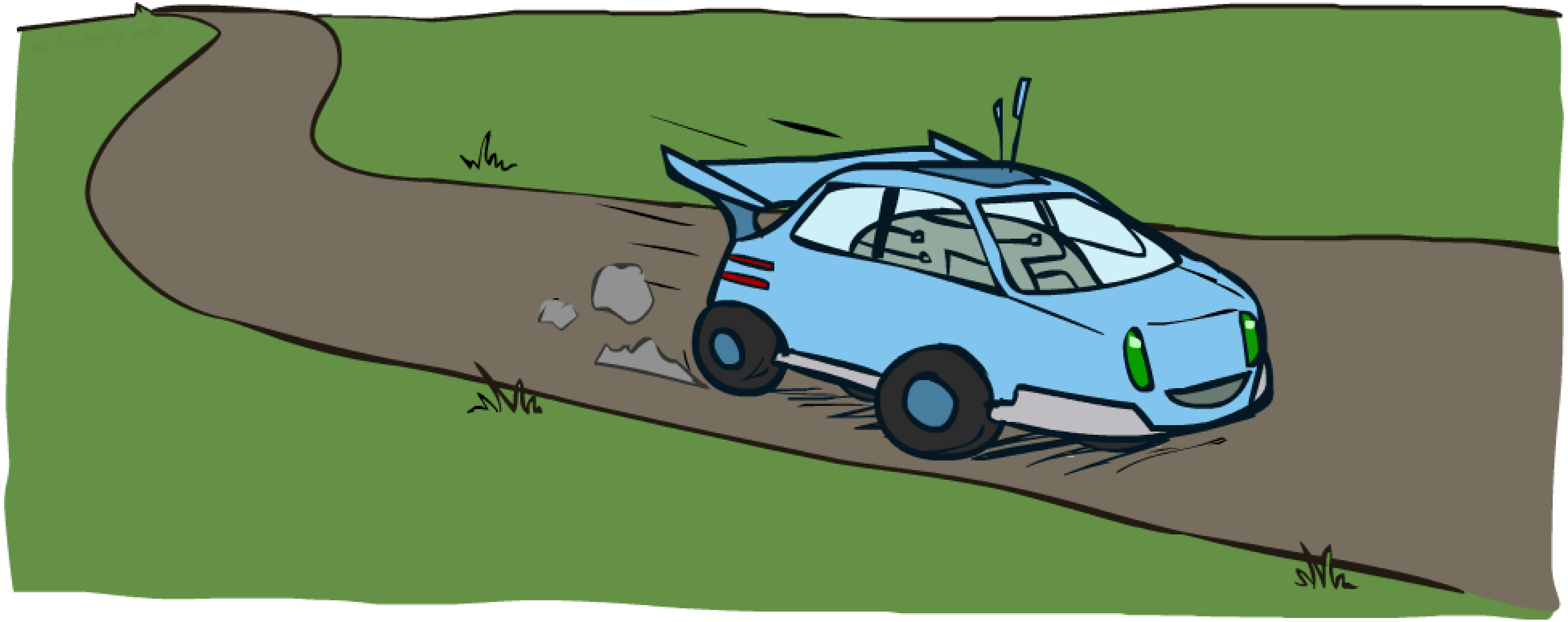


R(s) = -0.4



R(s) = -2.0

Example: Racing



Example: Racing

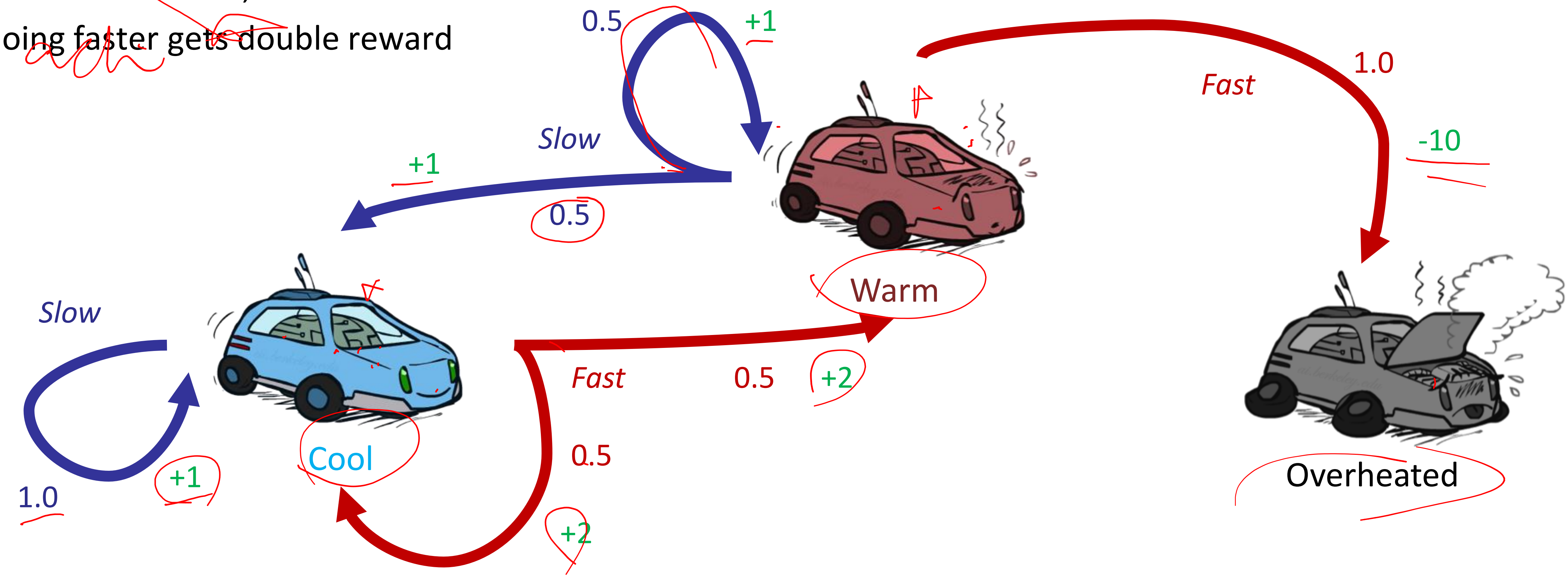
T(s, a, s')

- A robot car wants to travel far, quickly
- Three states: Cool, Warm, Overheated

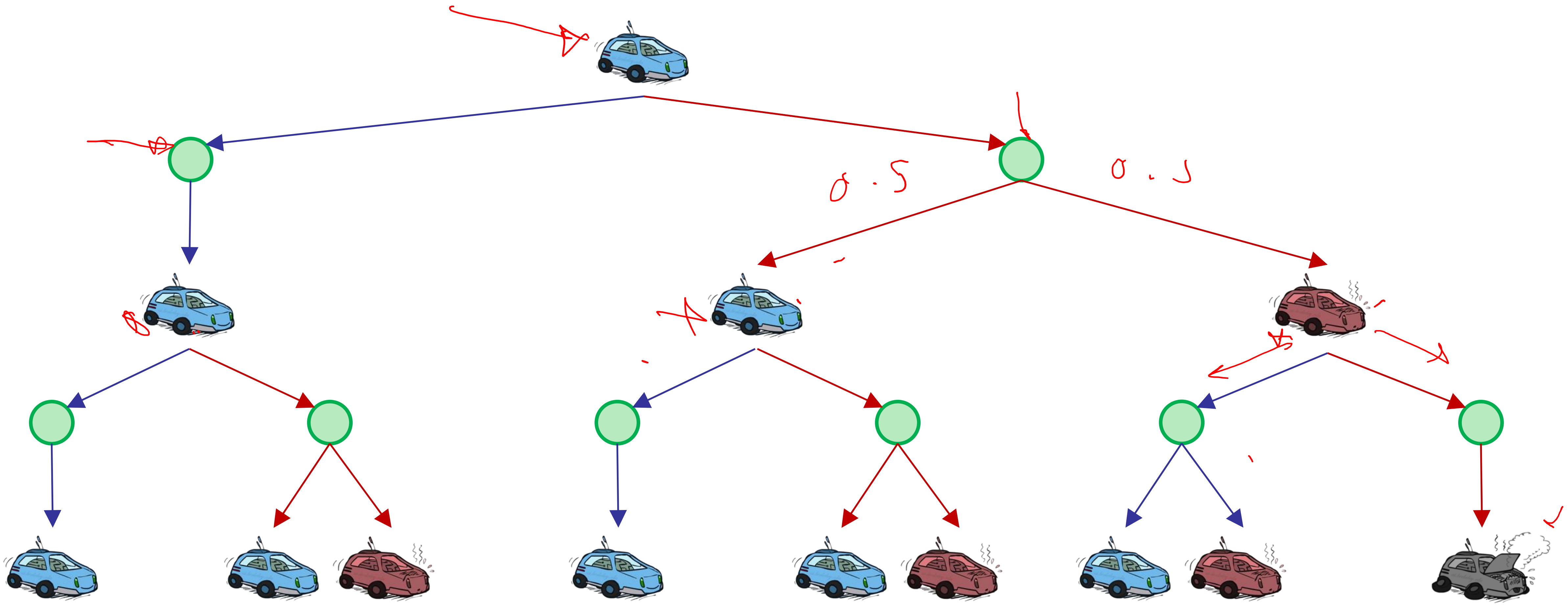
3 states

- Two actions: *Slow*, *Fast*
- Going faster gets double reward

2 actions

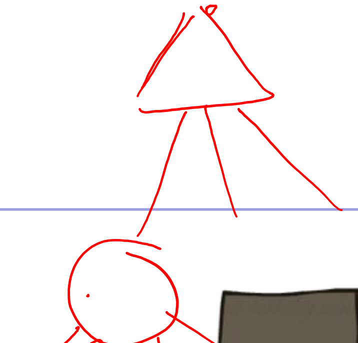


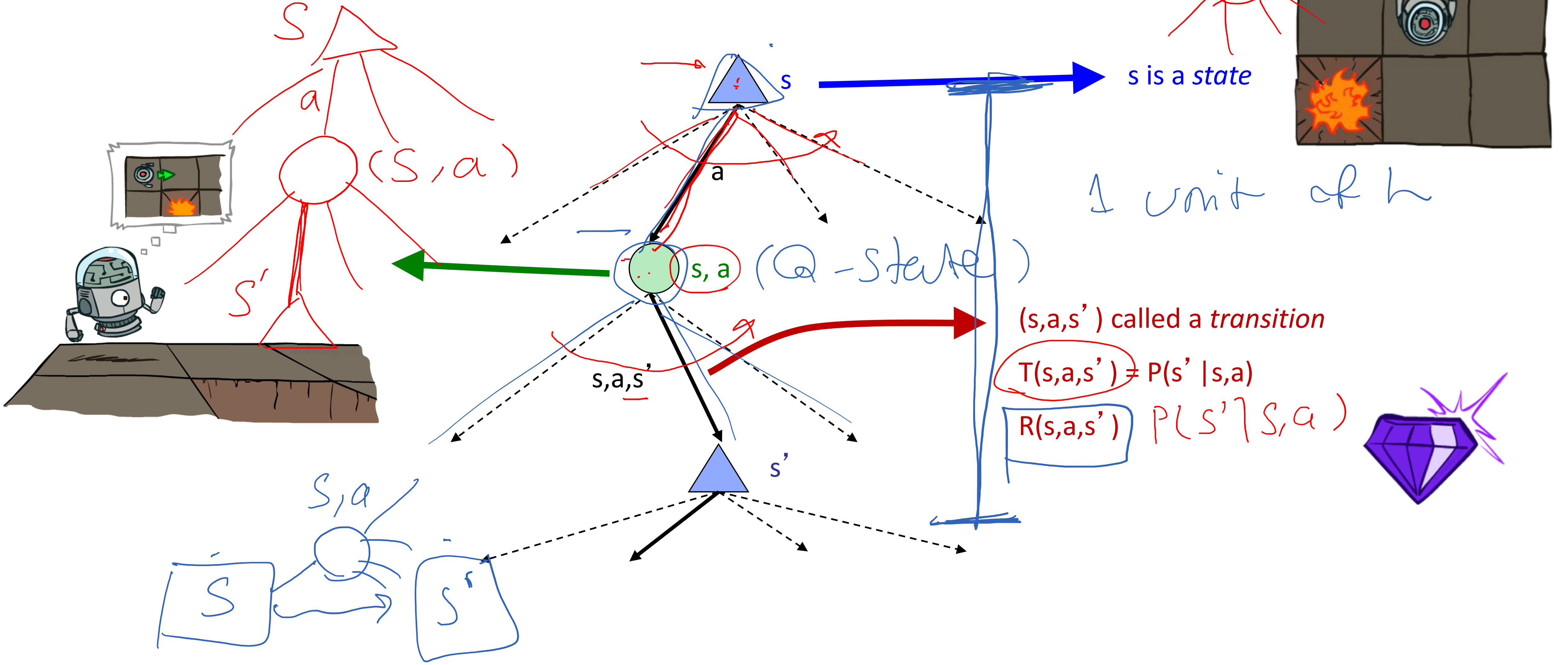
Racing Search Tree



MDP Search Trees

- Each MDP state projects an expectimax-like search tree





s is a *state*

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

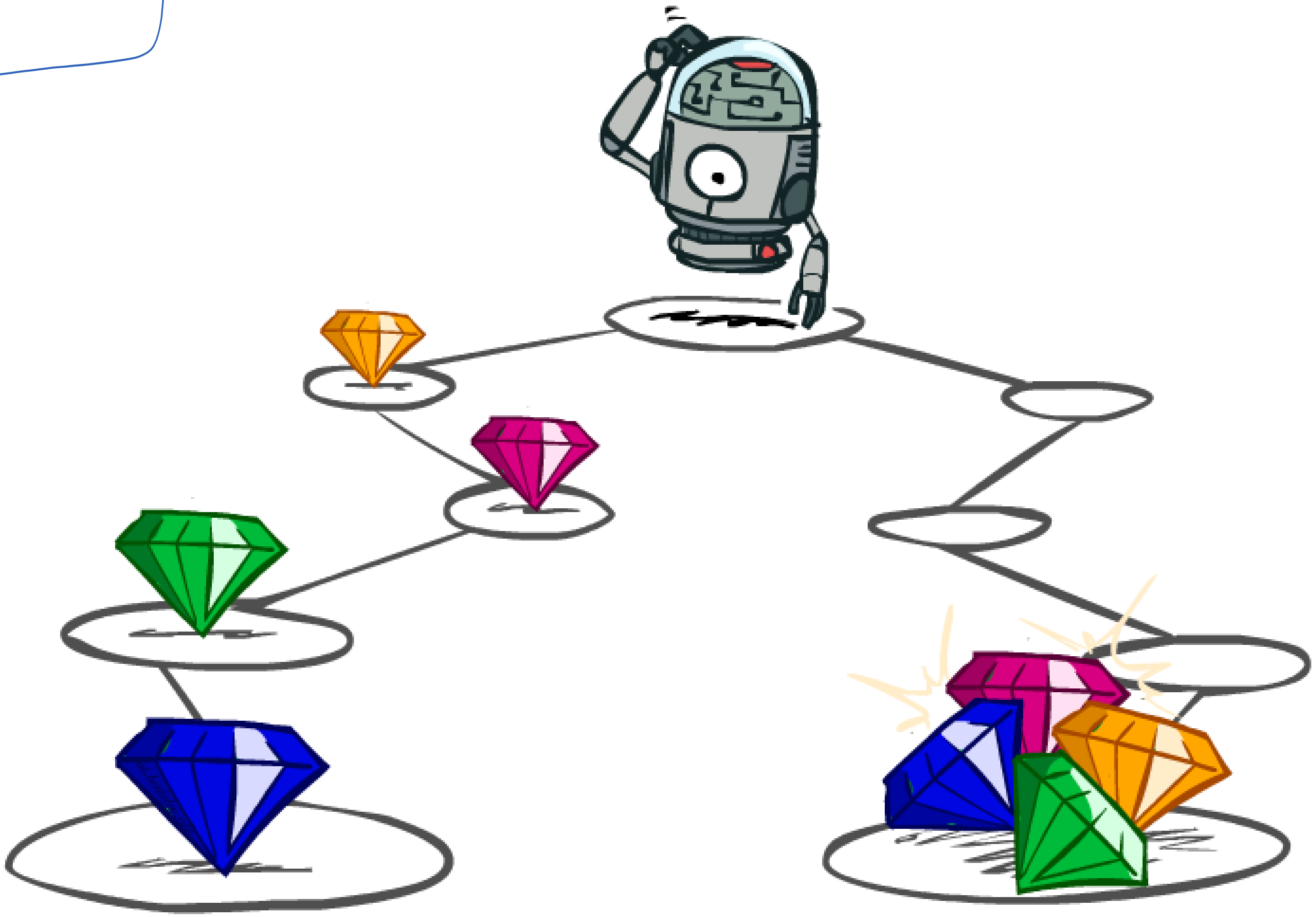
(s,a,s') called a *transition*

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

R(s,a,s')

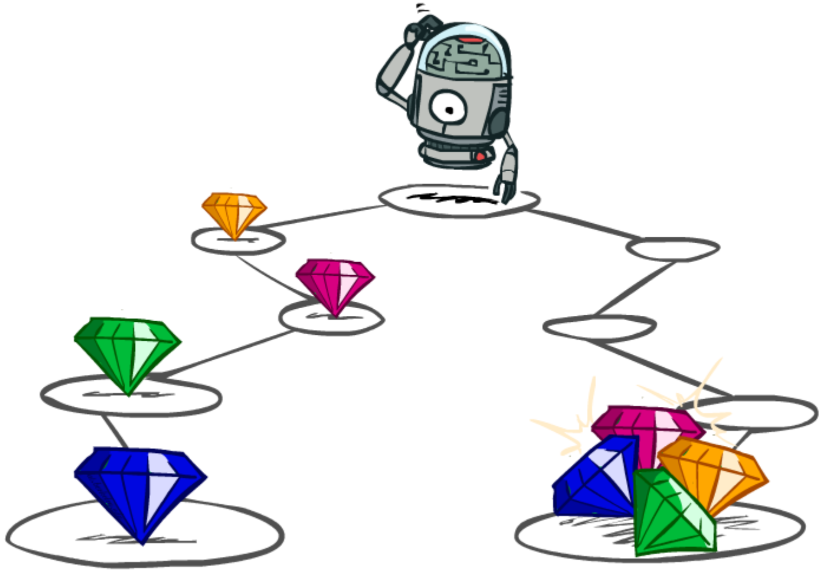
Utilities of Sequences

$\boxed{R(s,a,s')} = -0.01$



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

$0 < \gamma \leq 1$

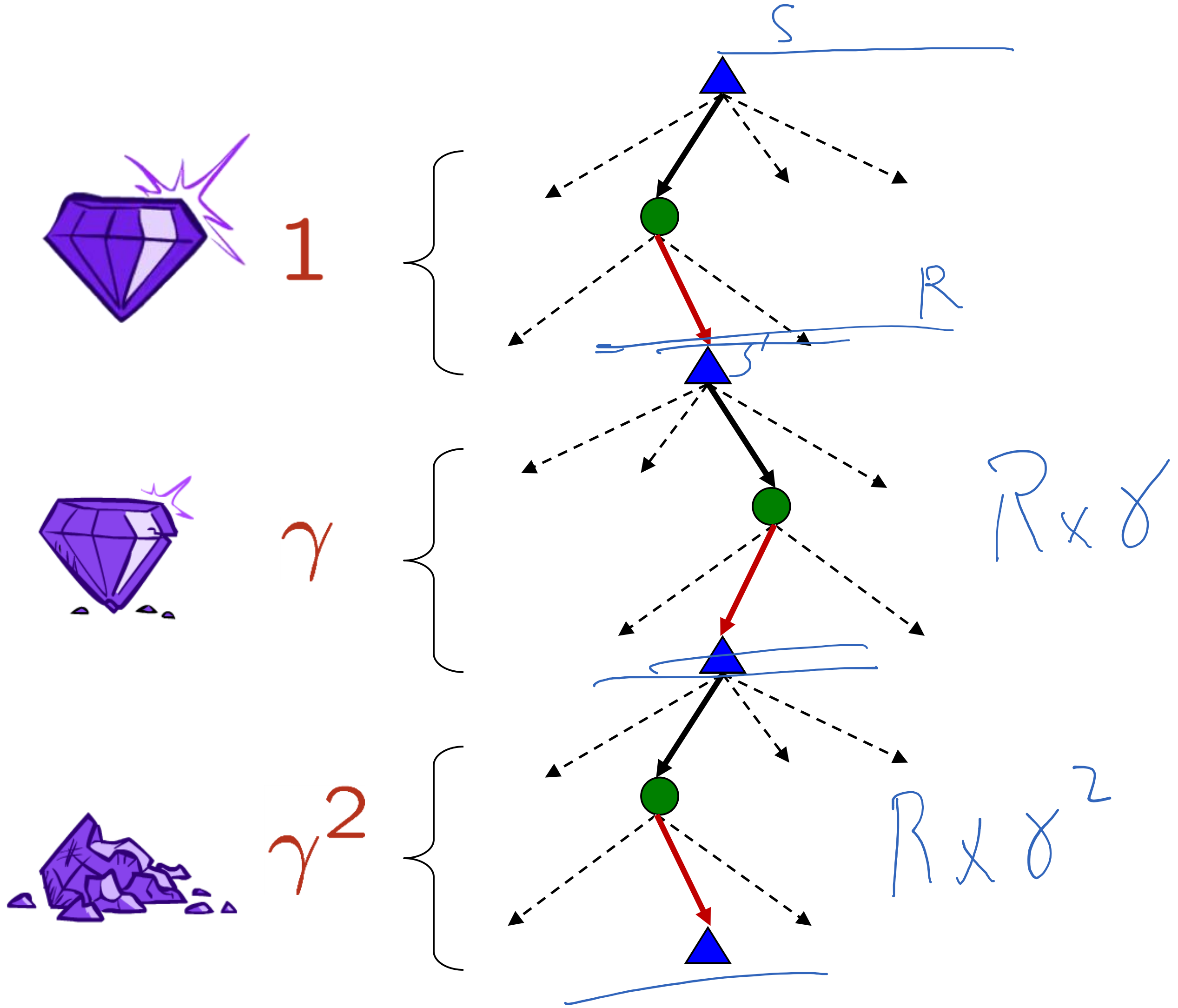
$R(s,a,s')$

Discounting

$R(s,a,s') = 0, -0.01$

- 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])

$1 \times 1 + \gamma \times 2 + \gamma^2 \times 3$



Quiz: Discounting

- Given:

a	b	c	d	e
10				1

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

- Quiz 1: For $\gamma = 1$, what is the optimal policy?

10	<-	<-	<-	1

- Quiz 2: For $\gamma = 0.1$, what is the optimal policy?

10	<-	<-	->	1

6

- Quiz 3: For which γ are West and East equally good when in state d?

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

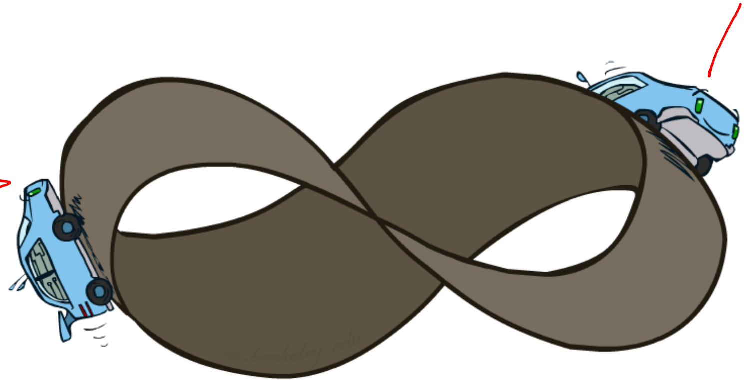
left: $0 + \gamma \times 0 + \gamma^2 \times 0 + \gamma^3 \cdot 10$

Right: $0 + \gamma \times 1$

Infinite Utilities?!

- Problem: What if the game lasts forever? Do we get infinite rewards?

- Solutions:
  - Finite horizon: (similar to depth-limited search)
    - Terminate episodes after a fixed T steps (e.g. life)
    - Policy π depends on time left



  - Discounting: use $0 < \gamma < 1$

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

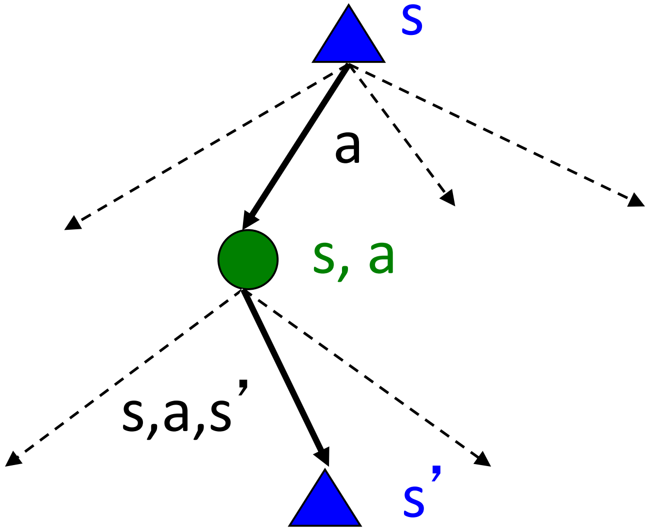
$$r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots + \gamma^\infty \cdot r_\infty$$

    - 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)

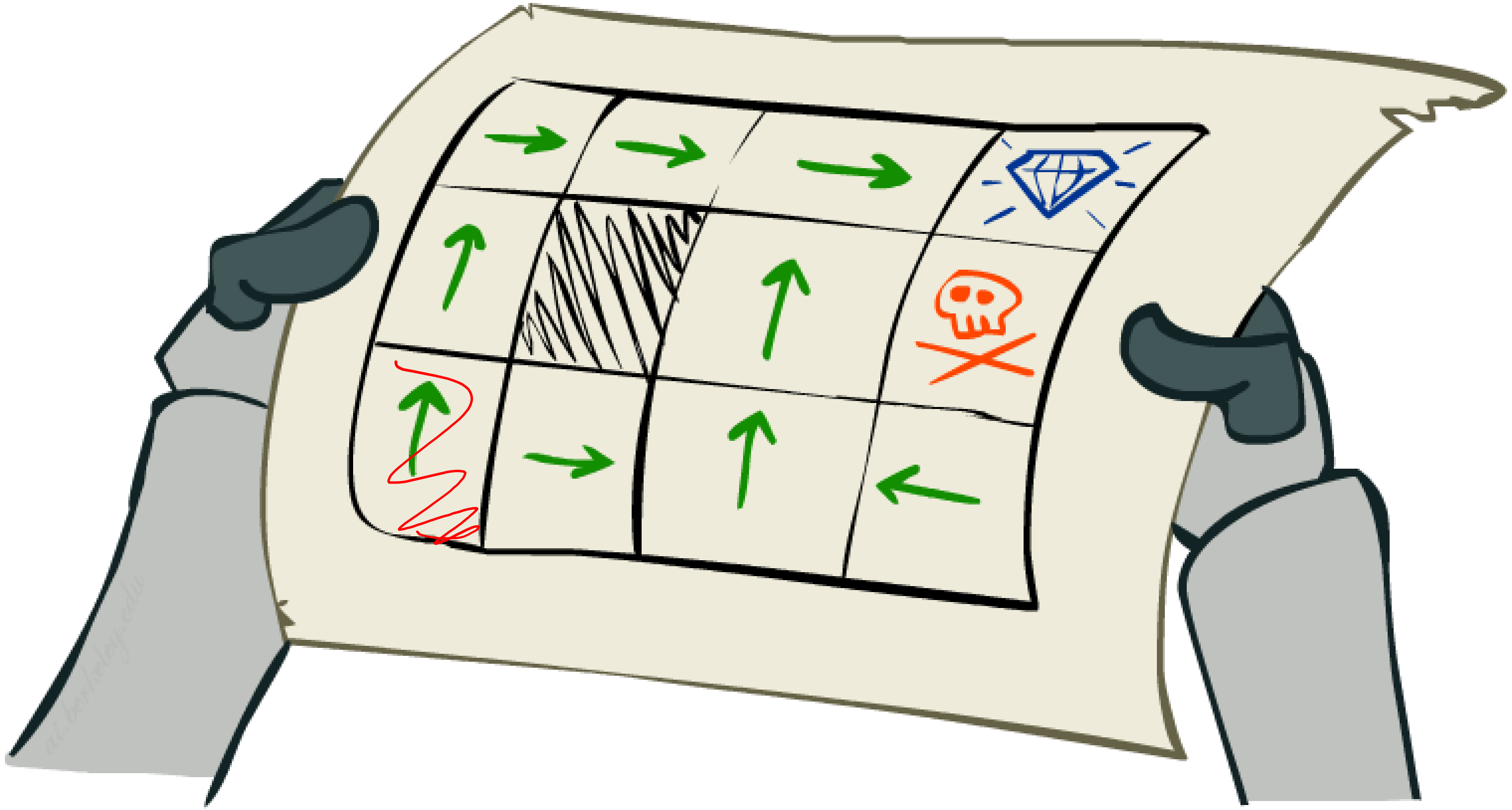
Recap: Defining MDPs

- 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)



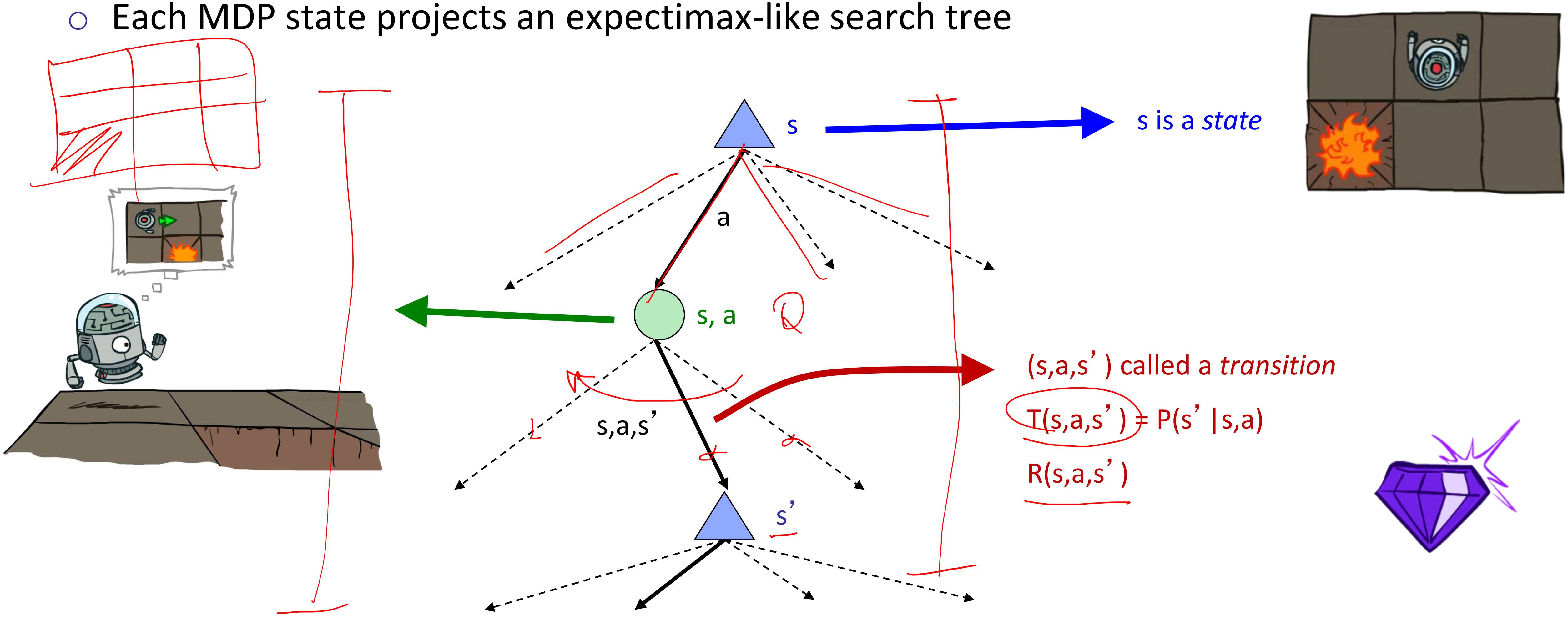
- MDP quantities so far:
  - Policy = Choice of action for each state
  - Utility = sum of (discounted) rewards

Solving MDPs



MDP Search Trees

- Each MDP state projects an expectimax-like search tree



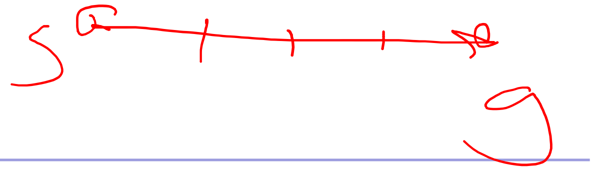
s is a *state*

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

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

$R(s,a,s')$

Optimal Quantities



- The value (utility) of a state s:

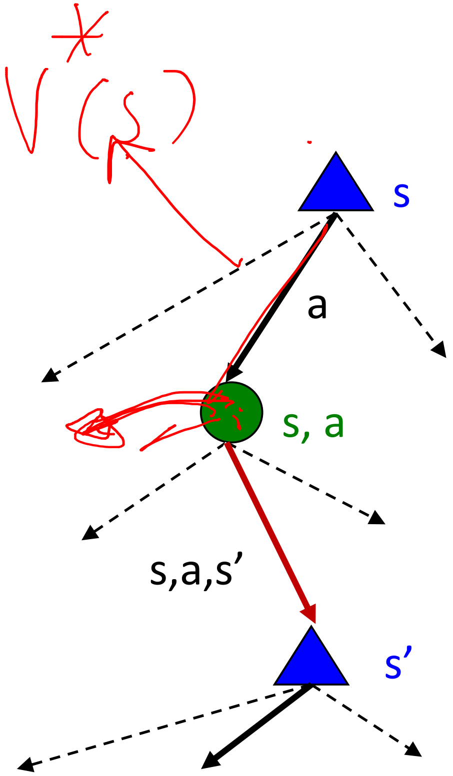
  $V^*(s)$ = expected utility starting in s and acting optimally

- 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*

Snapshot Gridworld V Values

0.64 ▸	0.74 ▸	0.85 ▸	1.00
▴ 0.57		▴ 0.57	-1.00
▴ 0.49	◂ 0.43	▴ 0.48	◂ 0.28

VALUES AFTER 100 ITERATIONS

0.9 × 1

1

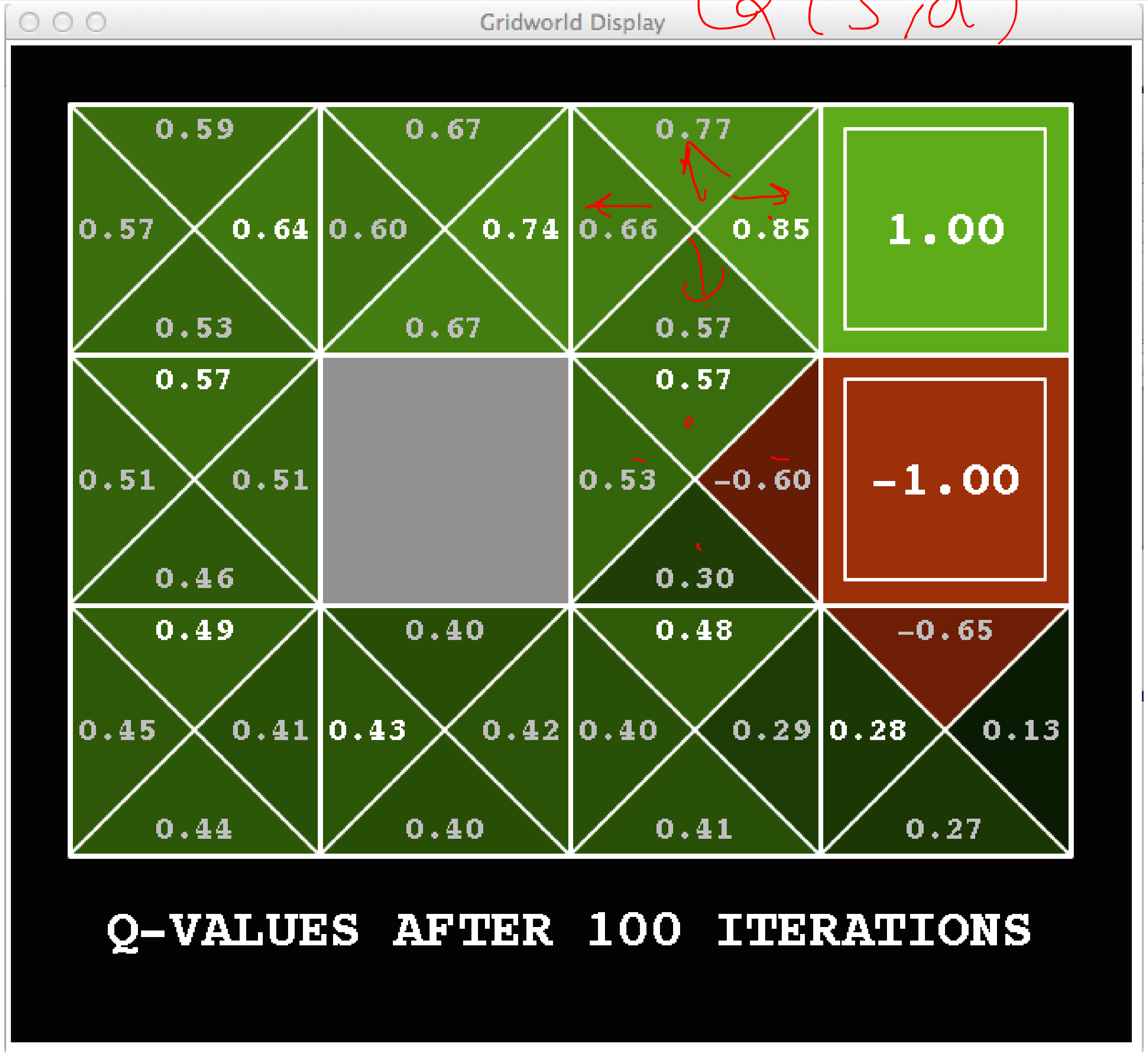
$V^*(s)$

0.8 0.1 0.1

Noise = 0.2
Discount = 0.9
Living reward = 0

Snapshot of Gridworld Q Values

Q(s,a)



Q-VALUES AFTER 100 ITERATIONS

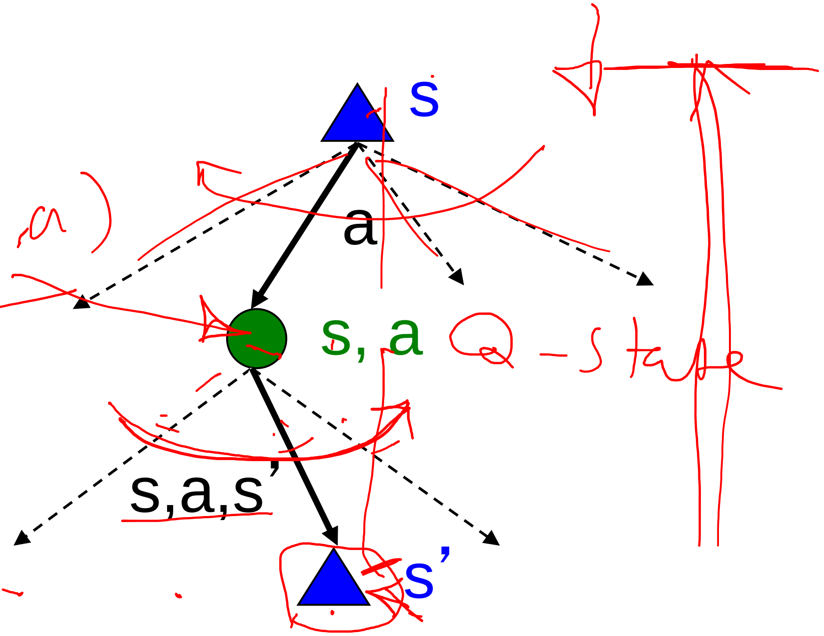
Noise = 0.2
Discount = 0.9
Living reward = 0

Values of States (Bellman Equations)

- Fundamental operation: compute the (expectimax) value of a state
  - Expected utility under optimal action
  - Average sum of (discounted) rewards
  - This is just what expectimax computed!

V(s)

Q*(s,a)



- Recursive definition of value:

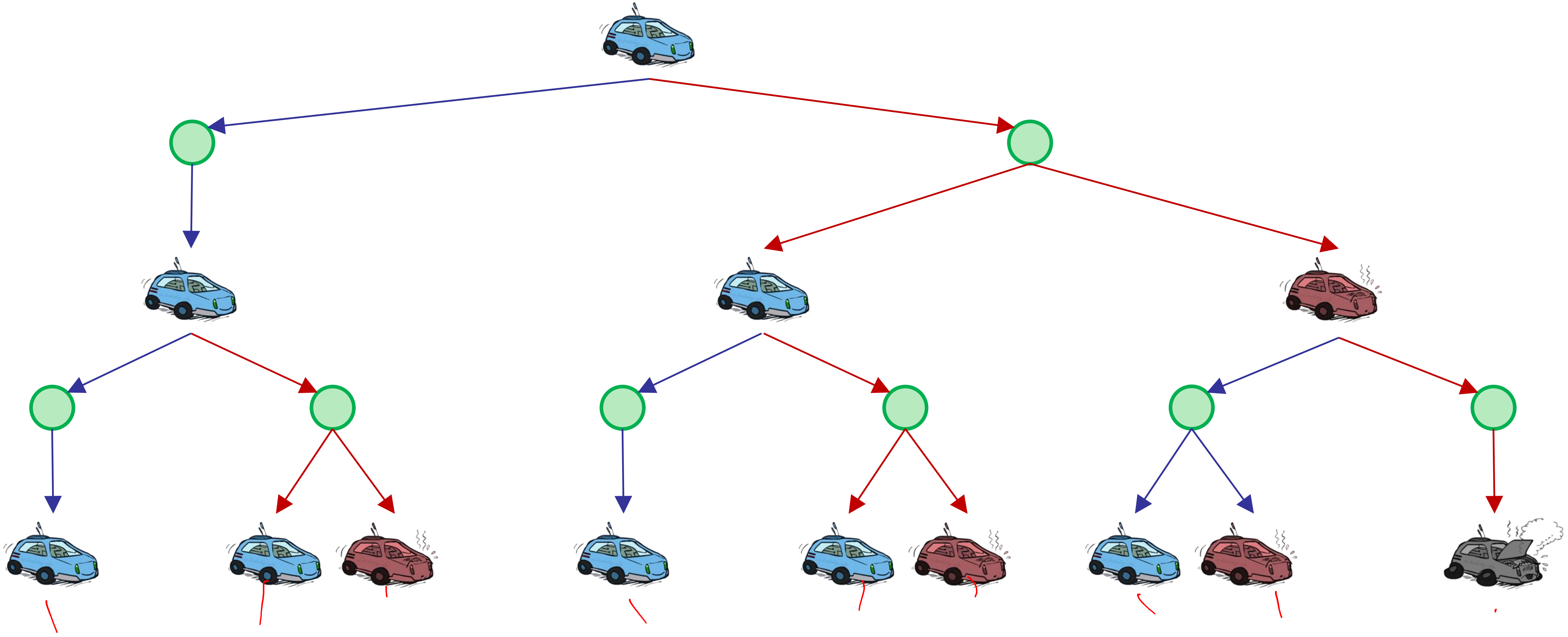
$$V^*(s) = \max_a Q^*(s,a)$$

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

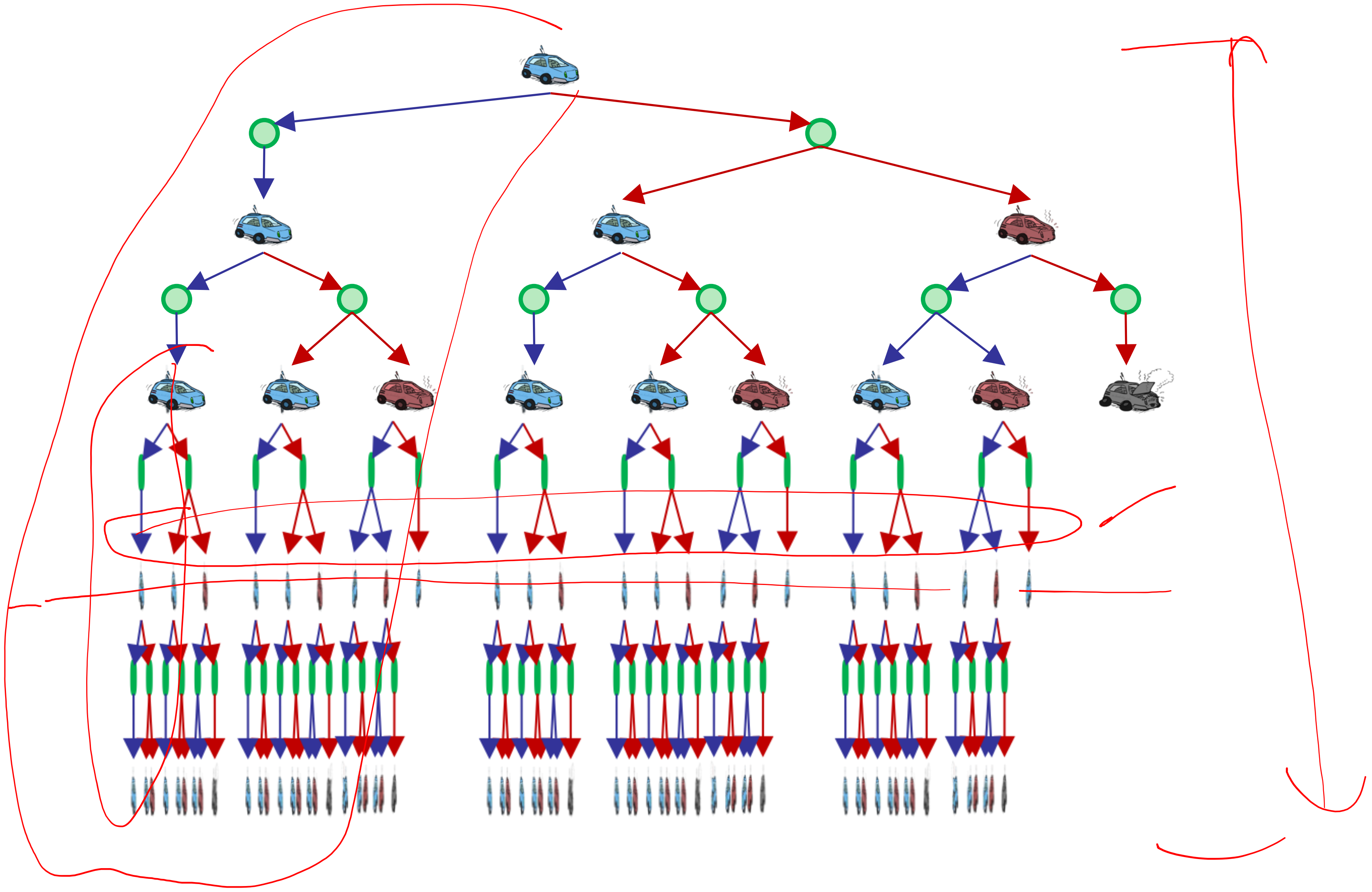
$$V^*(s) = \max_a \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V^*(s')\right]$$

γ

Racing Search Tree

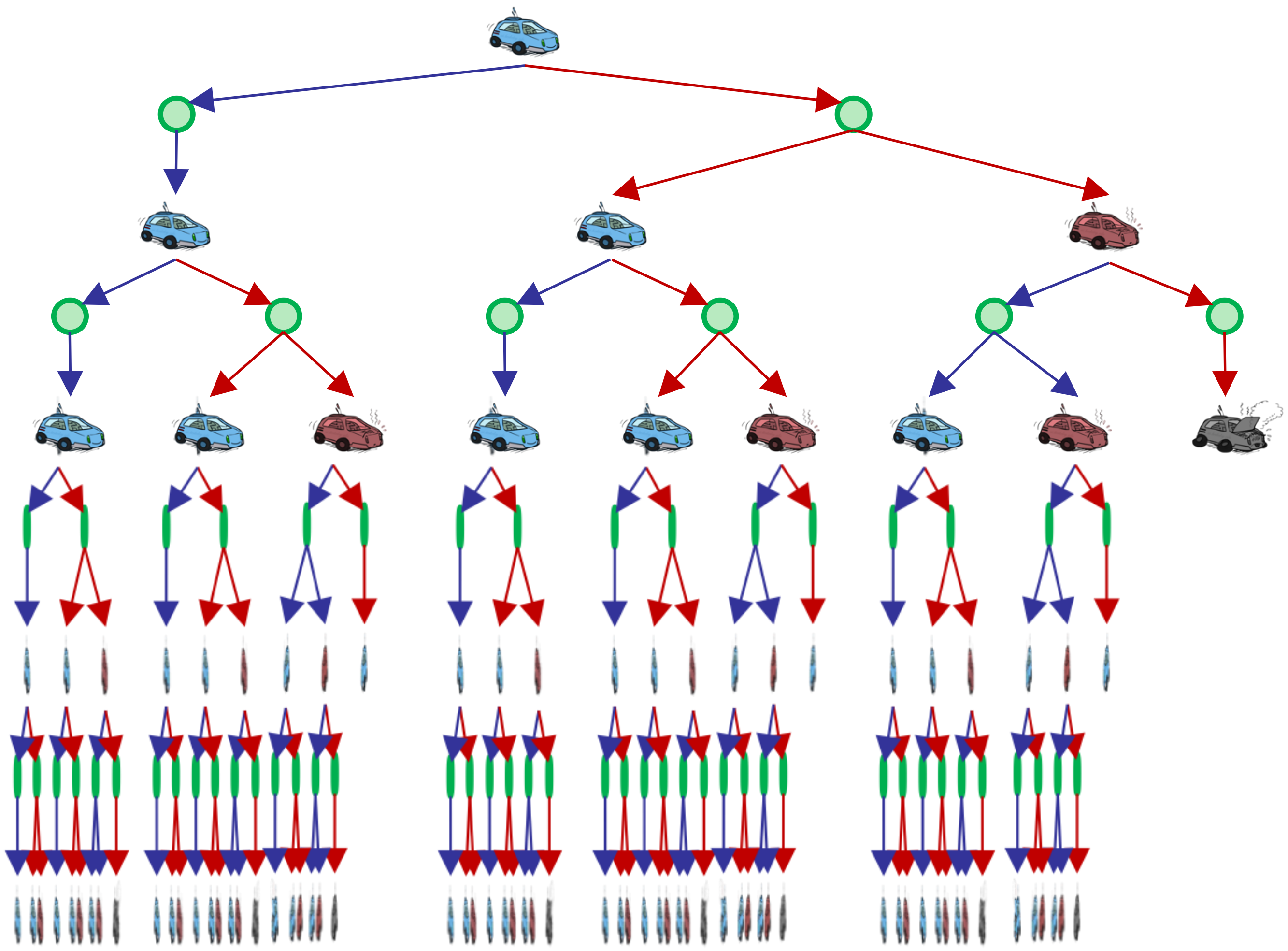


Racing Search Tree

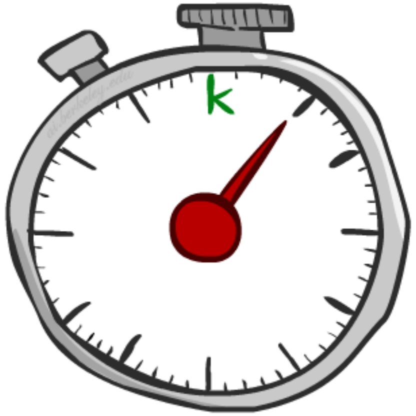


Racing Search Tree

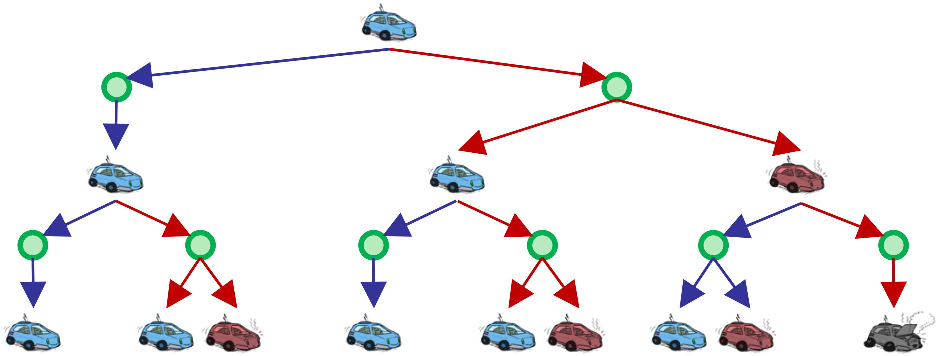
- We're doing way too much work with expectimax!
- Problem: States are repeated
  - Idea quantities: Only compute needed once
- Problem: Tree goes on forever
  - Idea: Do a depth-limited computation, but with increasing depths until change is small
  - Note: deep parts of the tree

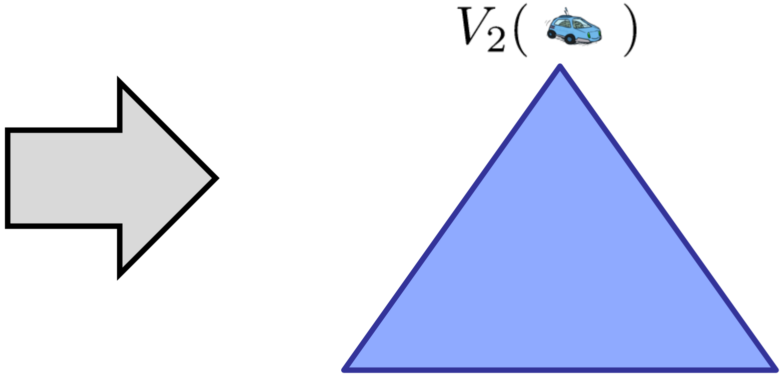


Time-Limited Values



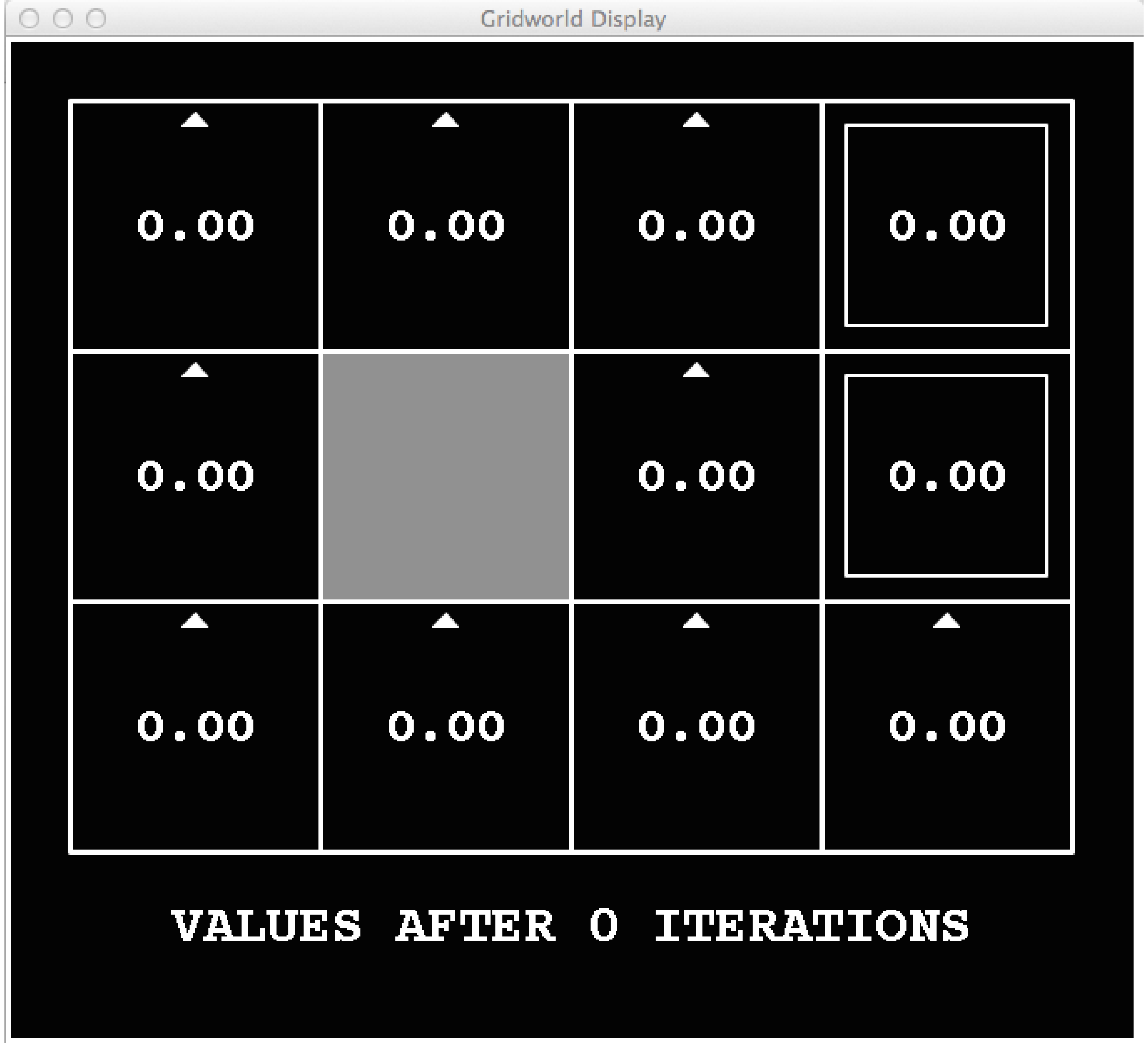
- Key idea: time-limited values
- Define $V_k(s)$ to be the optimal value of s if the game ends in k more time steps
  - Equivalently, it's what a depth-k expectimax would give from s





[Demo – time-limited values (L8D6)]

k=0

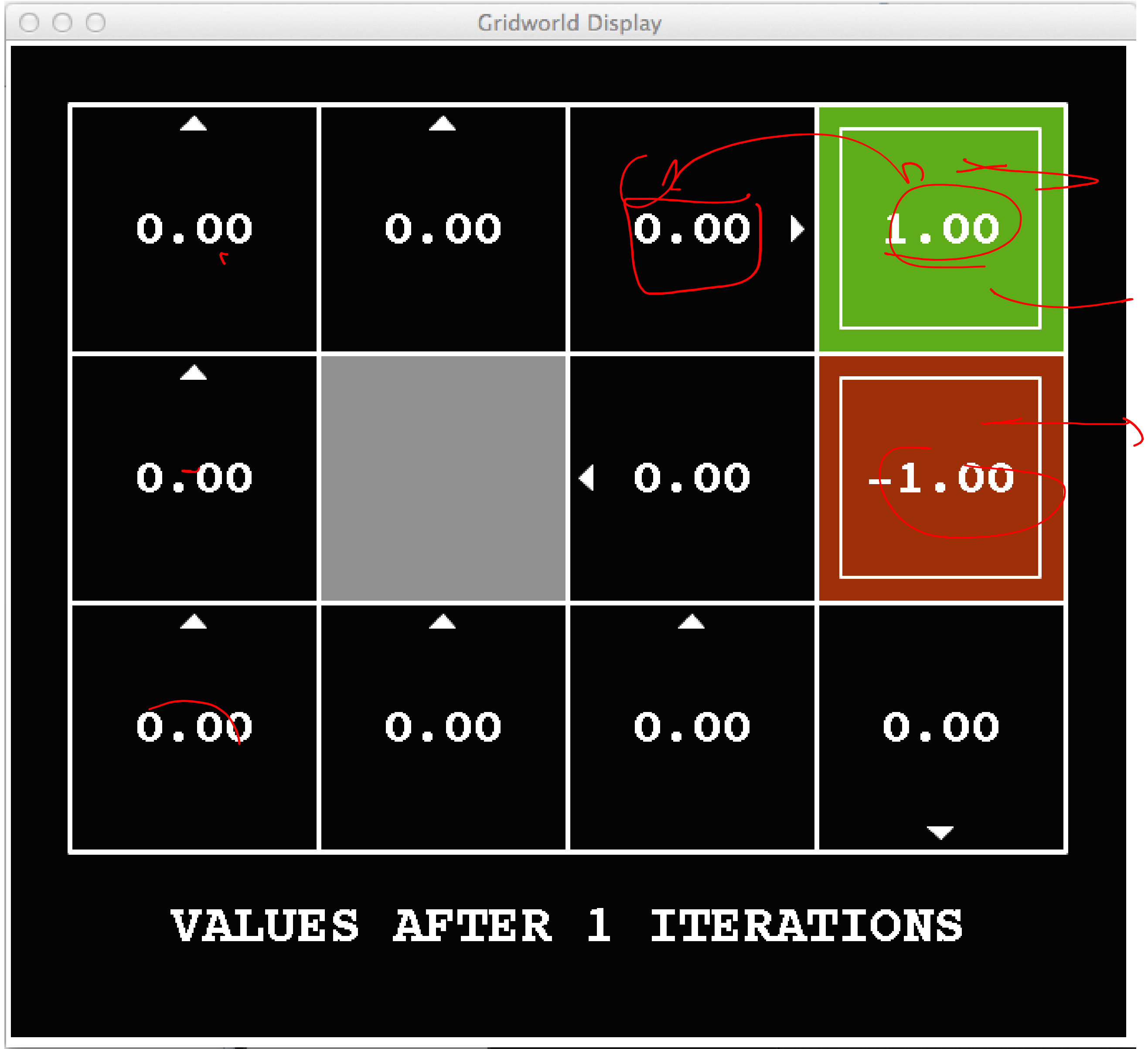


0.00	0.00	0.00	0.00
0.00		0.00	0.00
0.00	0.00	0.00	0.00

VALUES AFTER 0 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

k=1


Gridworld Display

0.00	0.00	0.00	1.00
0.00		0.00	-1.00
0.00	0.00	0.00	0.00

VALUES AFTER 1 ITERATIONS

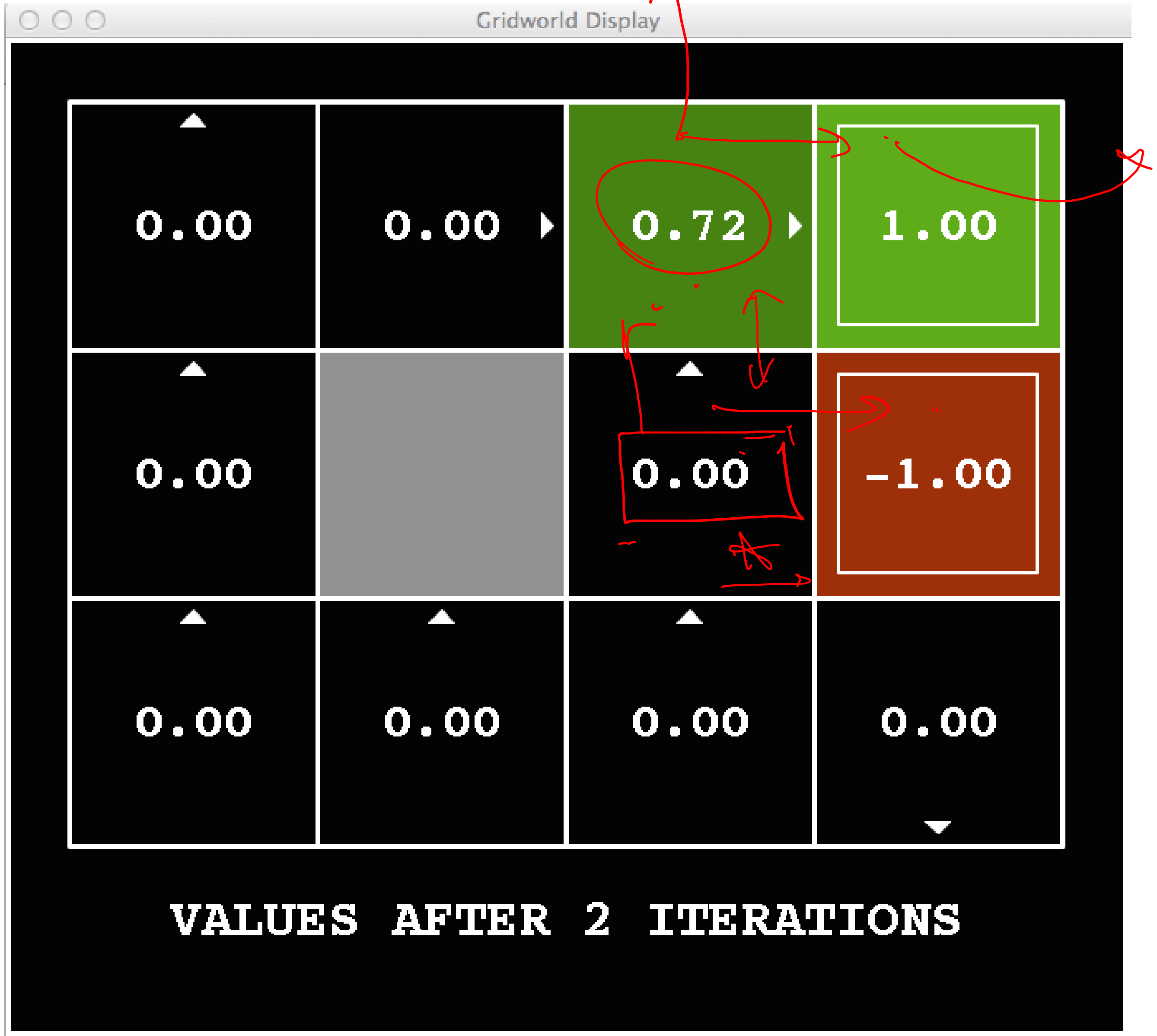
Noise = 0.2
Discount = 0.9
Living reward = 0

$T[R + \gamma \times 1]$

k=2

$V_k(s)$

optimal value



Gridworld Display

▲ 0.00	0.00 ▶	0.72 ▶	1.00
▲ 0.00		▲ 0.00	-1.00
▲ 0.00	▲ 0.00	▲ 0.00	0.00 ▼

VALUES AFTER 2 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

k=3

$T(s,a,s')[R+\gamma V^*(s')]$

Rt

Lft

$(R+\gamma$ [illegible] $)$

S

$0.8 \times 0.9 \times 1$

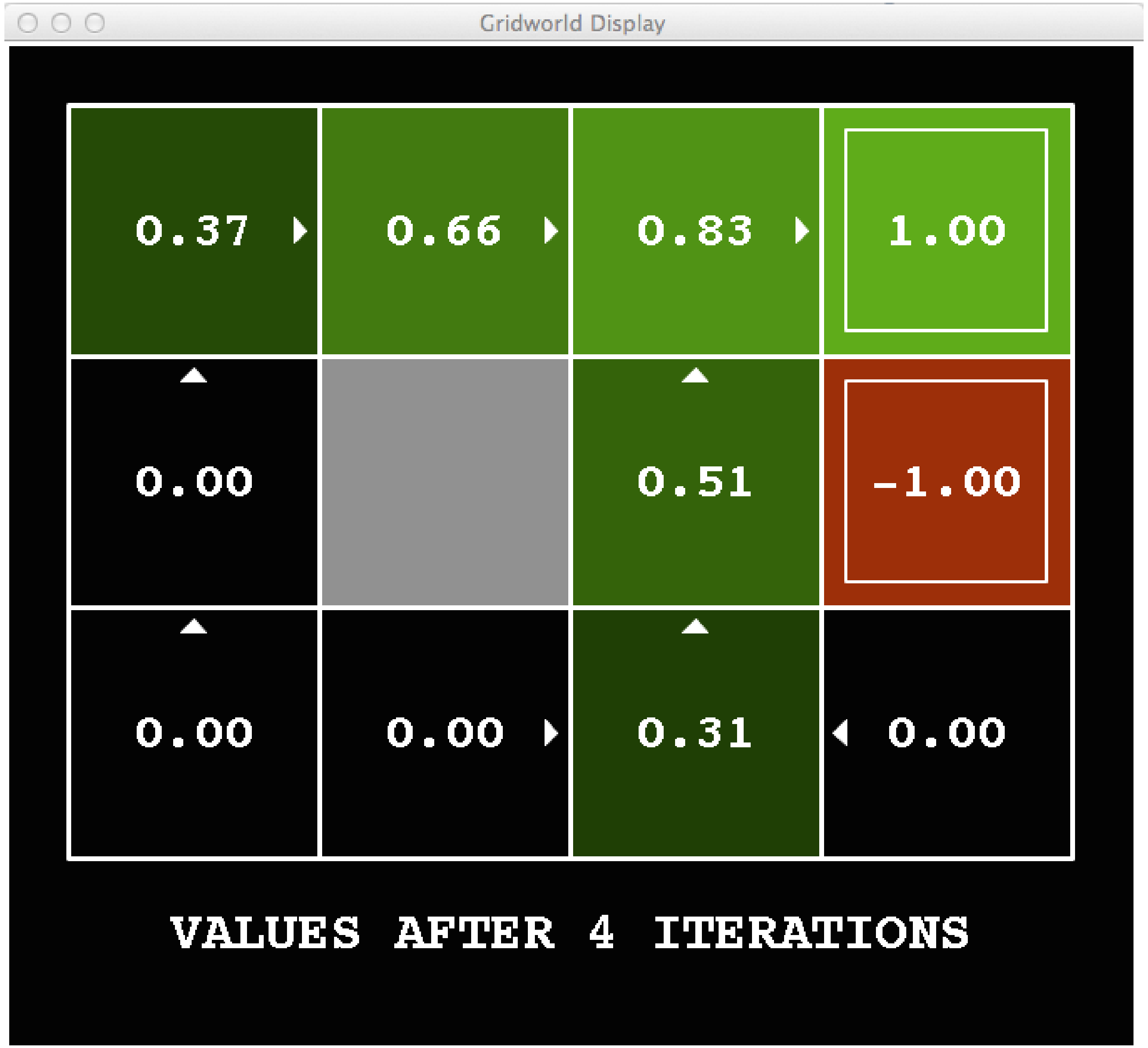
$+0.1\{0+\gamma\times 0.72\}$

0.00 ▸	0.52 ▸	0.78 ▸	1.00
▴ 0.00		▴ 0.43	-1.00
▴ 0.00	▴ 0.00	▴ 0.00	0.00 ▾

VALUES AFTER 3 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

k=4

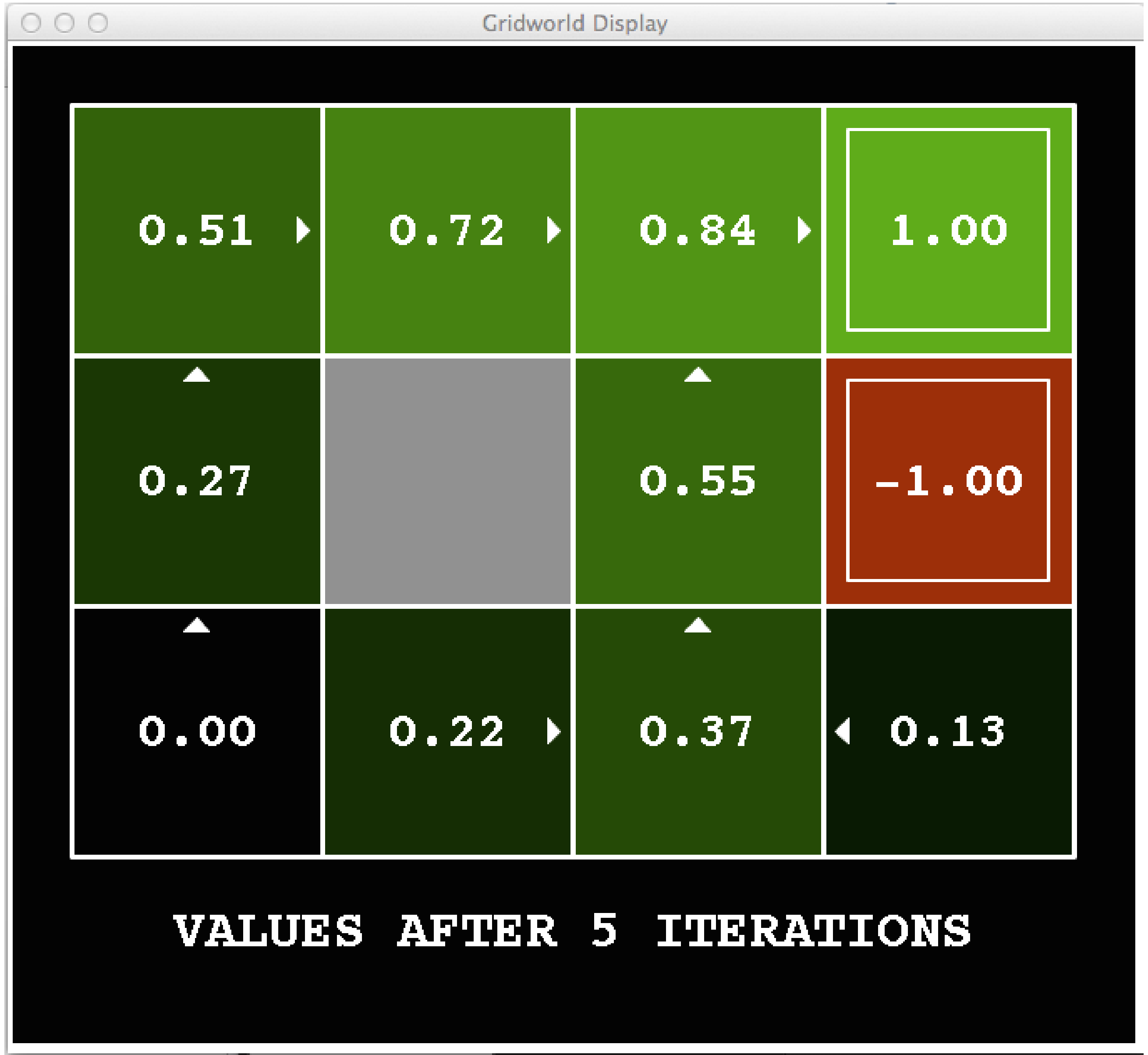


0.37 ▸	0.66 ▸	0.83 ▸	1.00
▴ 0.00		▴ 0.51	-1.00
▴ 0.00	0.00 ▸	▴ 0.31	◂ 0.00

VALUES AFTER 4 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

k=5



0.51 ▸	0.72 ▸	0.84 ▸	1.00
▴ 0.27		▴ 0.55	-1.00
▴ 0.00	0.22 ▸	▴ 0.37	◂ 0.13

VALUES AFTER 5 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

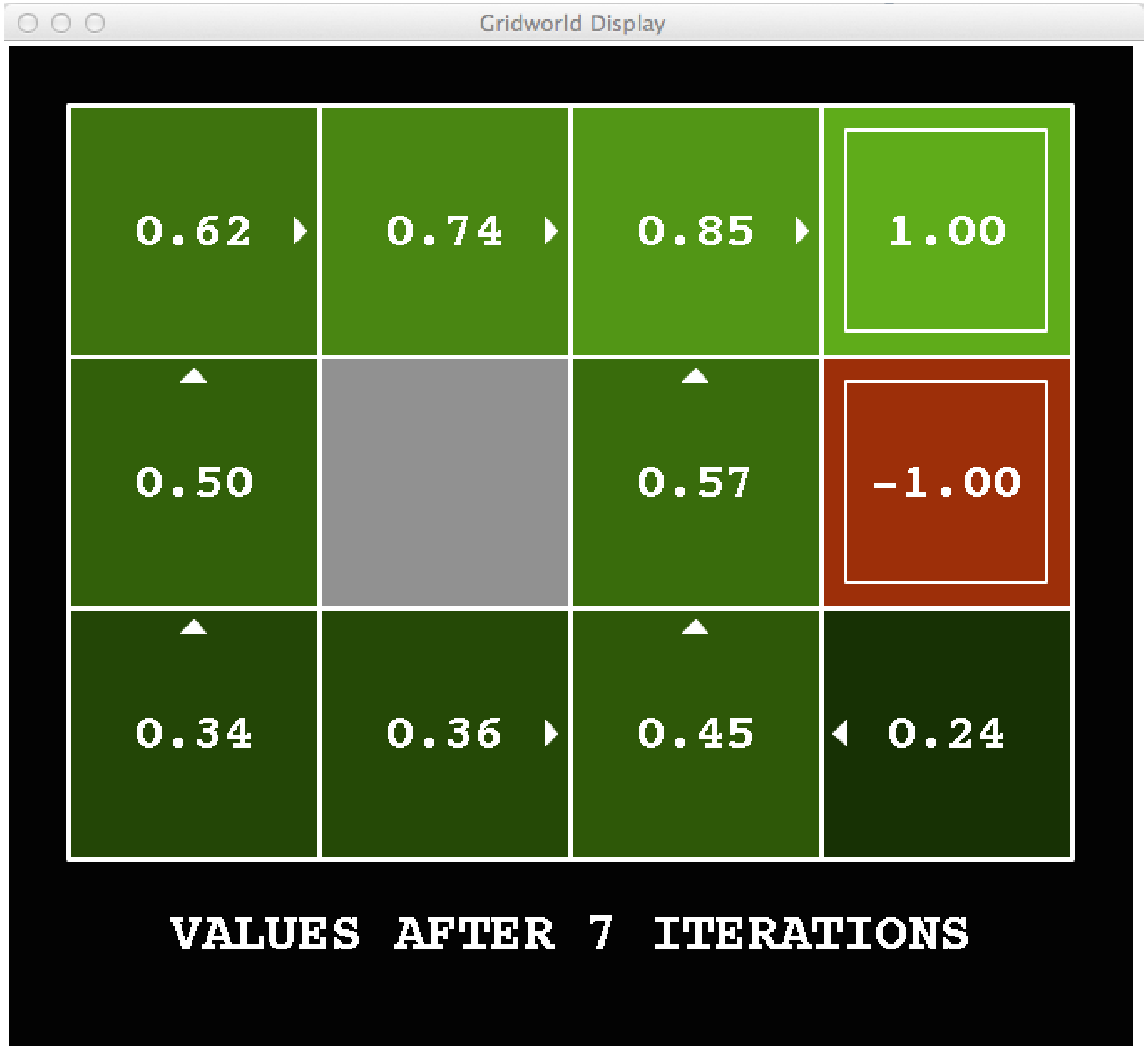
k=6

0.59 ▶	0.73 ▶	0.85 ▶	1.00
▲ 0.41		▲ 0.57	-1.00
▲ 0.21	0.31 ▶	▲ 0.43	◀ 0.19

VALUES AFTER 6 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

k=7

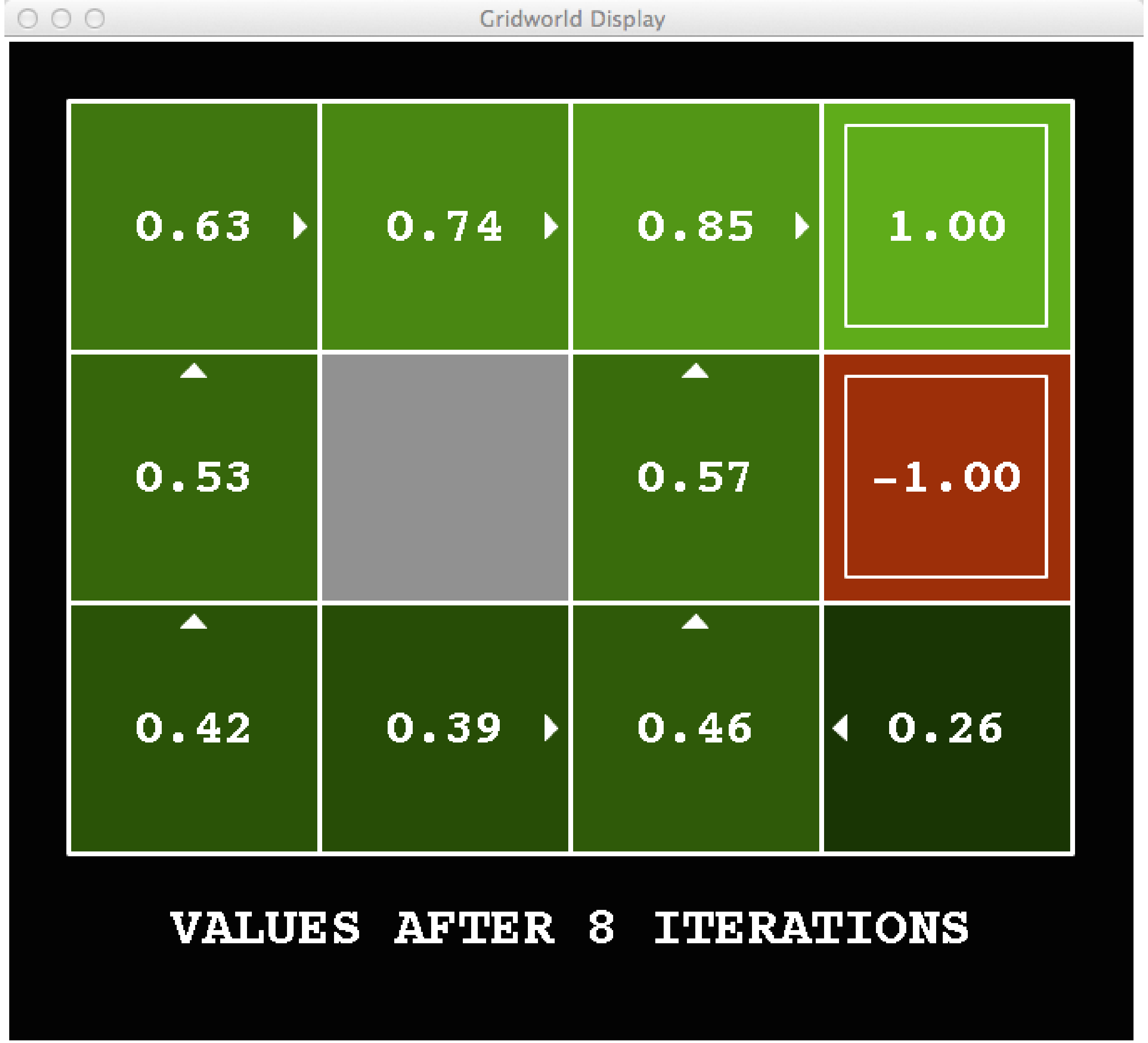


0.62 ▸	0.74 ▸	0.85 ▸	1.00
▴ 0.50		▴ 0.57	-1.00
▴ 0.34	0.36 ▸	▴ 0.45	◂ 0.24

VALUES AFTER 7 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

k=8

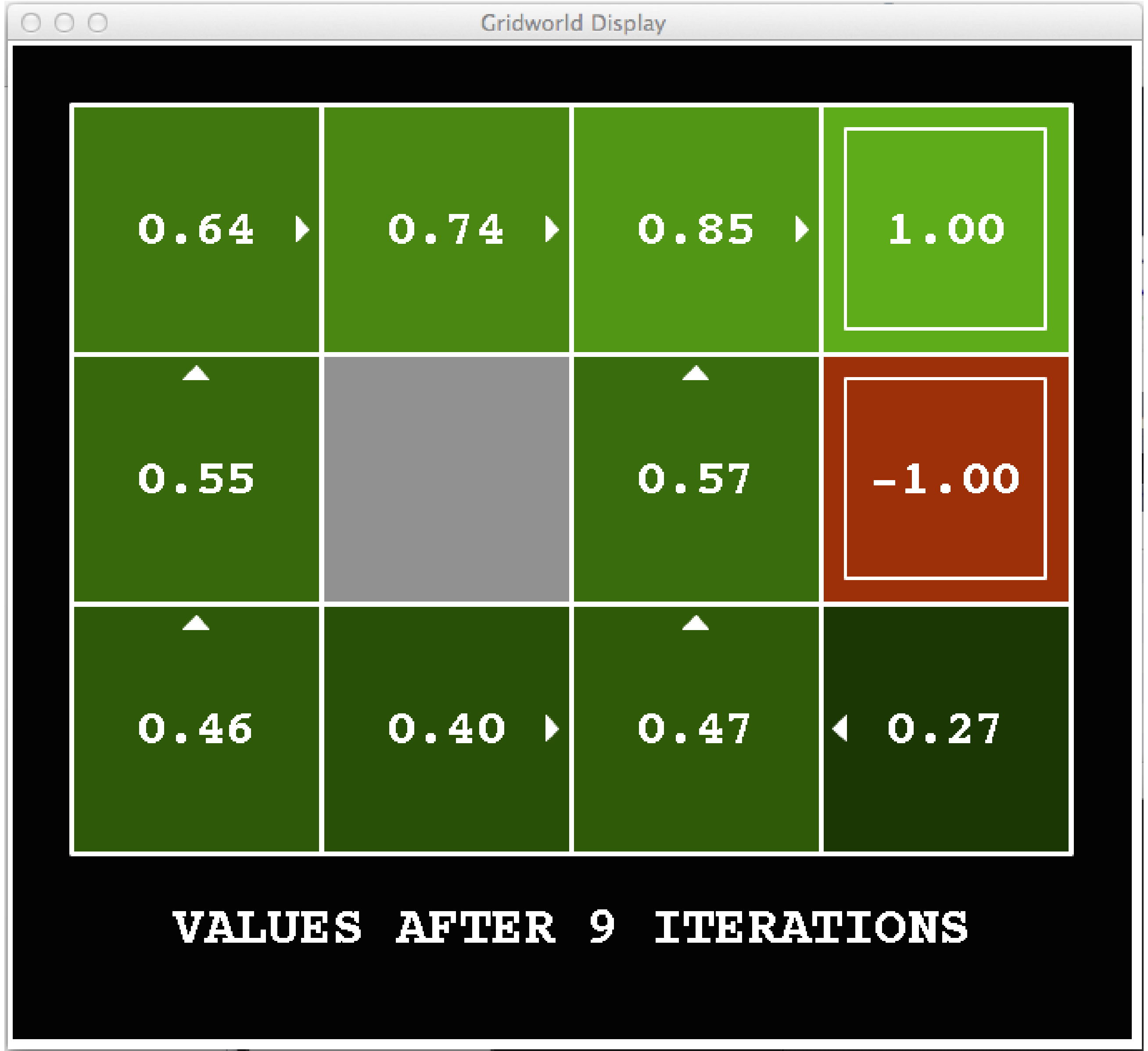


0.63 ▸	0.74 ▸	0.85 ▸	1.00
▴ 0.53		▴ 0.57	-1.00
▴ 0.42	0.39 ▸	▴ 0.46	◂ 0.26

VALUES AFTER 8 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

k=9

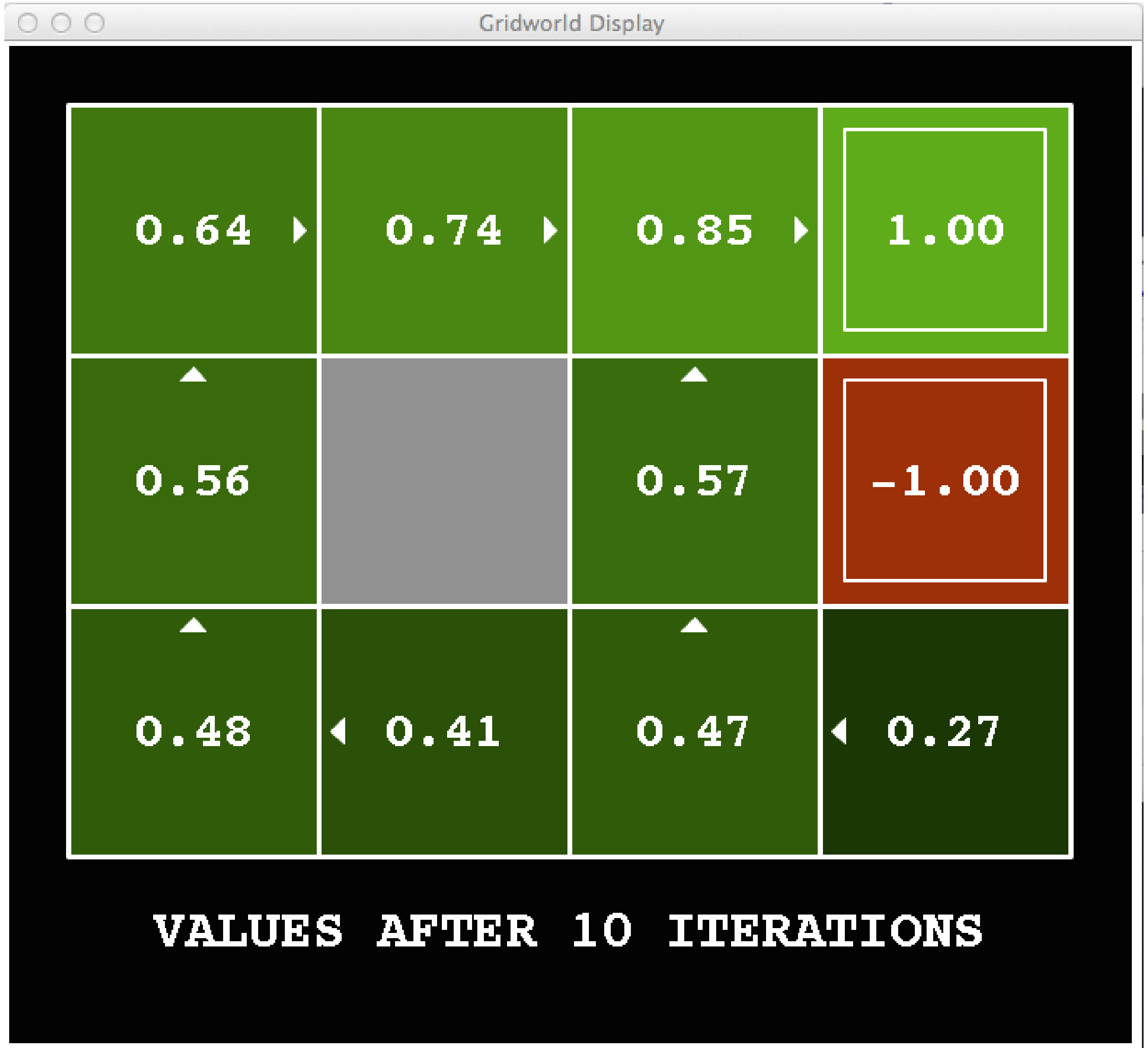


0.64 ▶	0.74 ▶	0.85 ▶	1.00
▲ 0.55		▲ 0.57	-1.00
▲ 0.46	0.40 ▶	▲ 0.47	◀ 0.27

VALUES AFTER 9 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

k=10

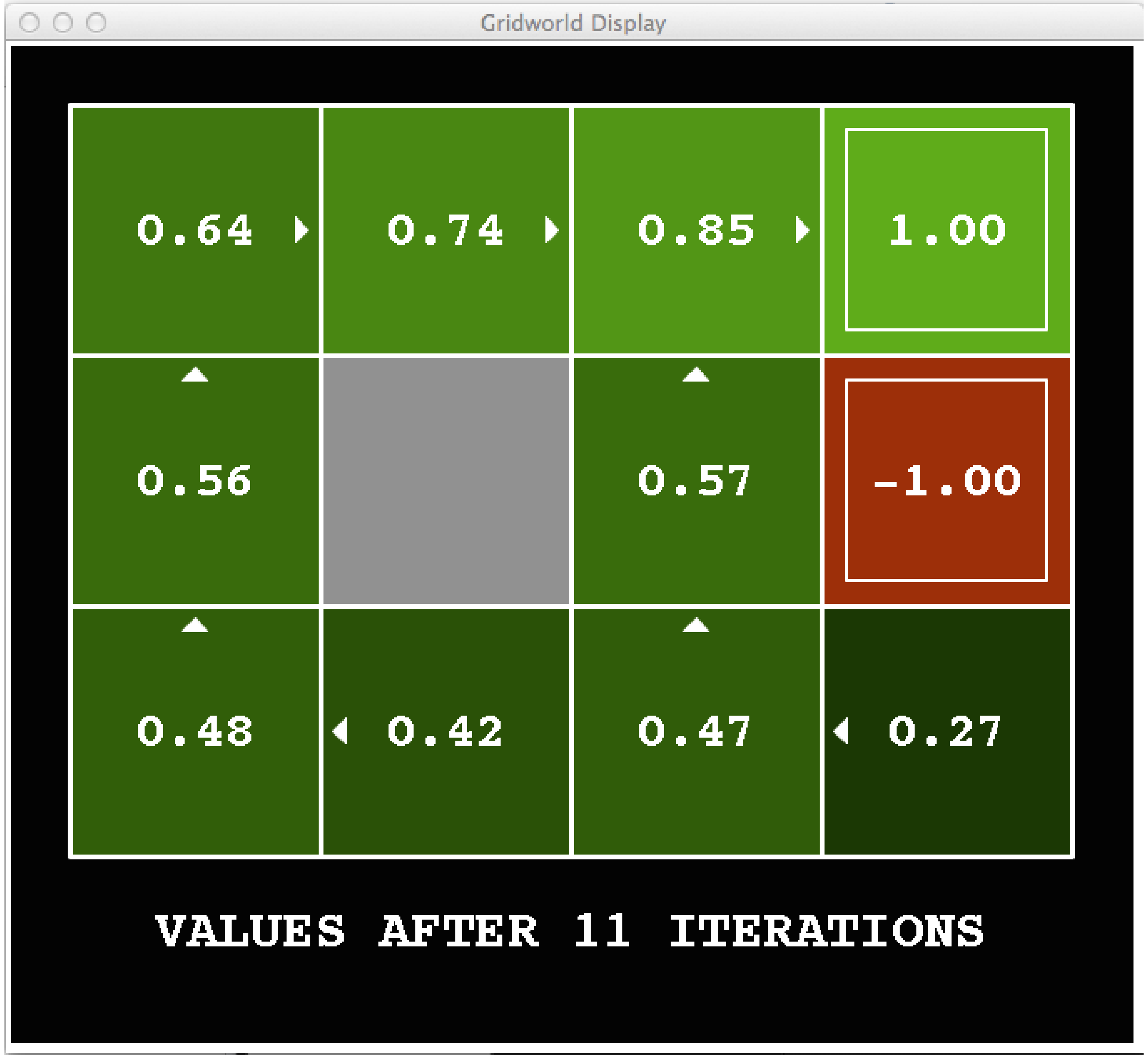


0.64 ▶	0.74 ▶	0.85 ▶	1.00
▲ 0.56		▲ 0.57	-1.00
▲ 0.48	◀ 0.41	▲ 0.47	◀ 0.27

VALUES AFTER 10 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

k=11

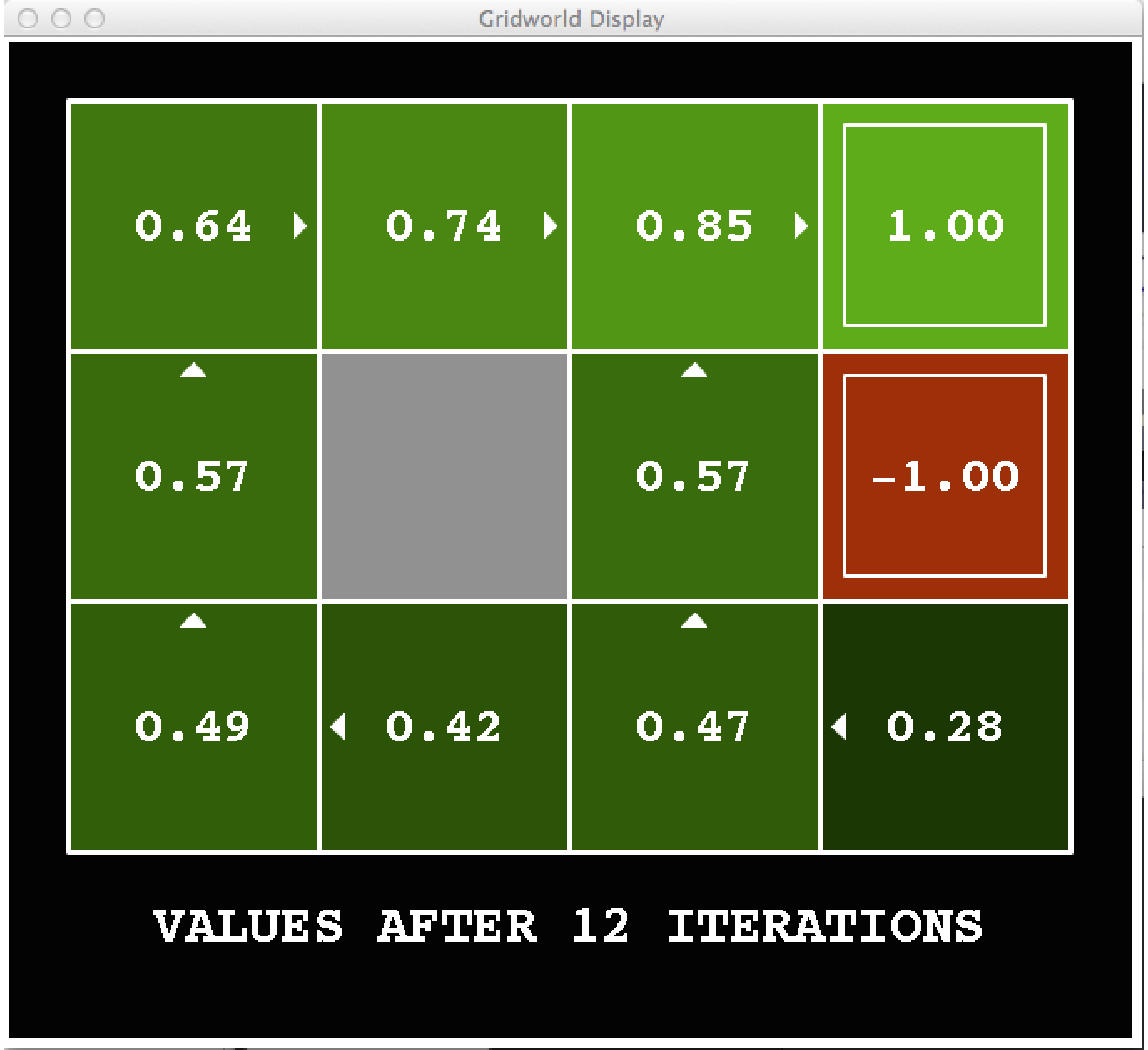


0.64 ▸	0.74 ▸	0.85 ▸	1.00
▴ 0.56		▴ 0.57	-1.00
▴ 0.48	◂ 0.42	▴ 0.47	◂ 0.27

VALUES AFTER 11 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

k=12

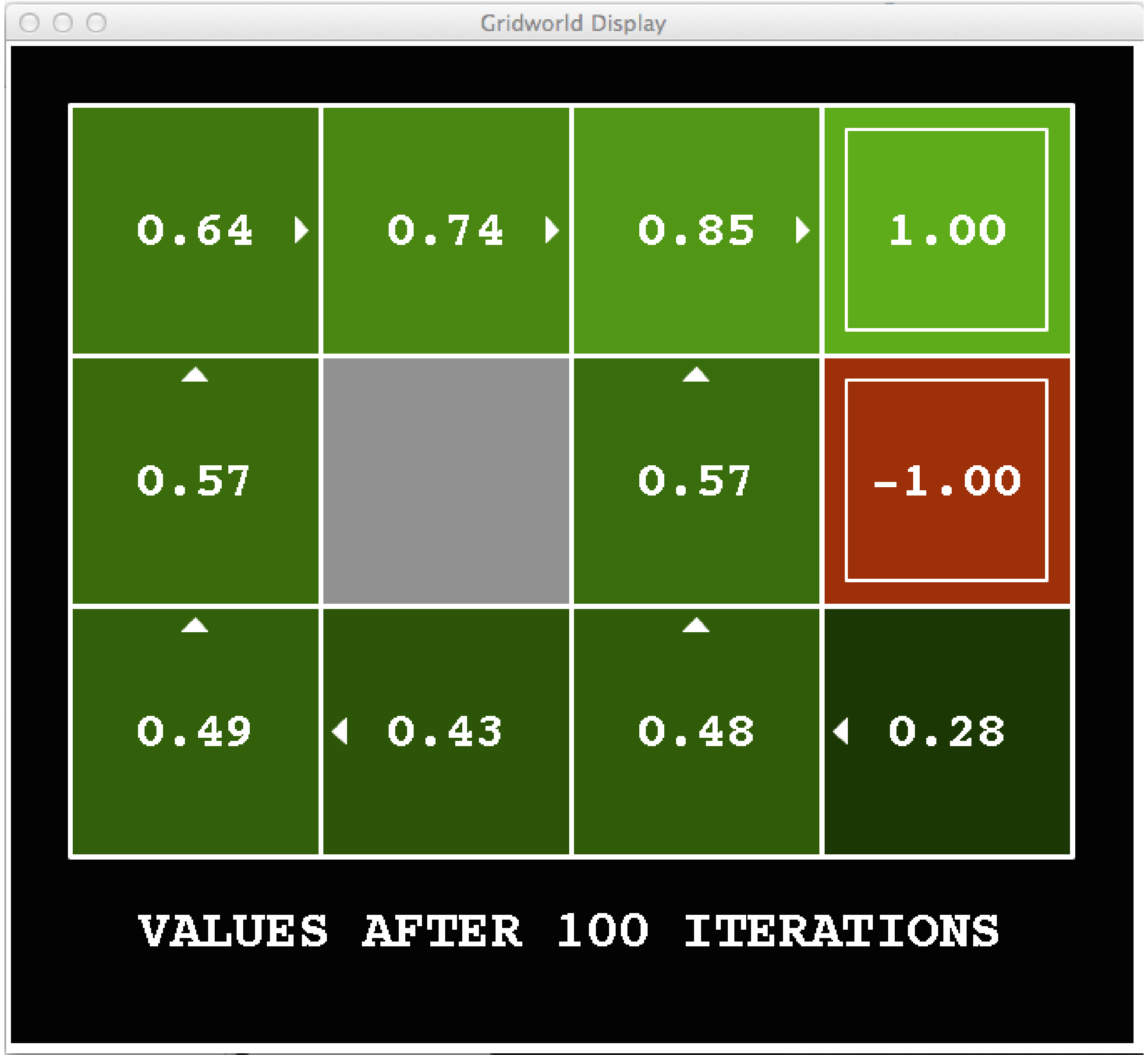


0.64 ▶	0.74 ▶	0.85 ▶	1.00
▲ 0.57		▲ 0.57	-1.00
▲ 0.49	◀ 0.42	▲ 0.47	◀ 0.28

VALUES AFTER 12 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

k=100

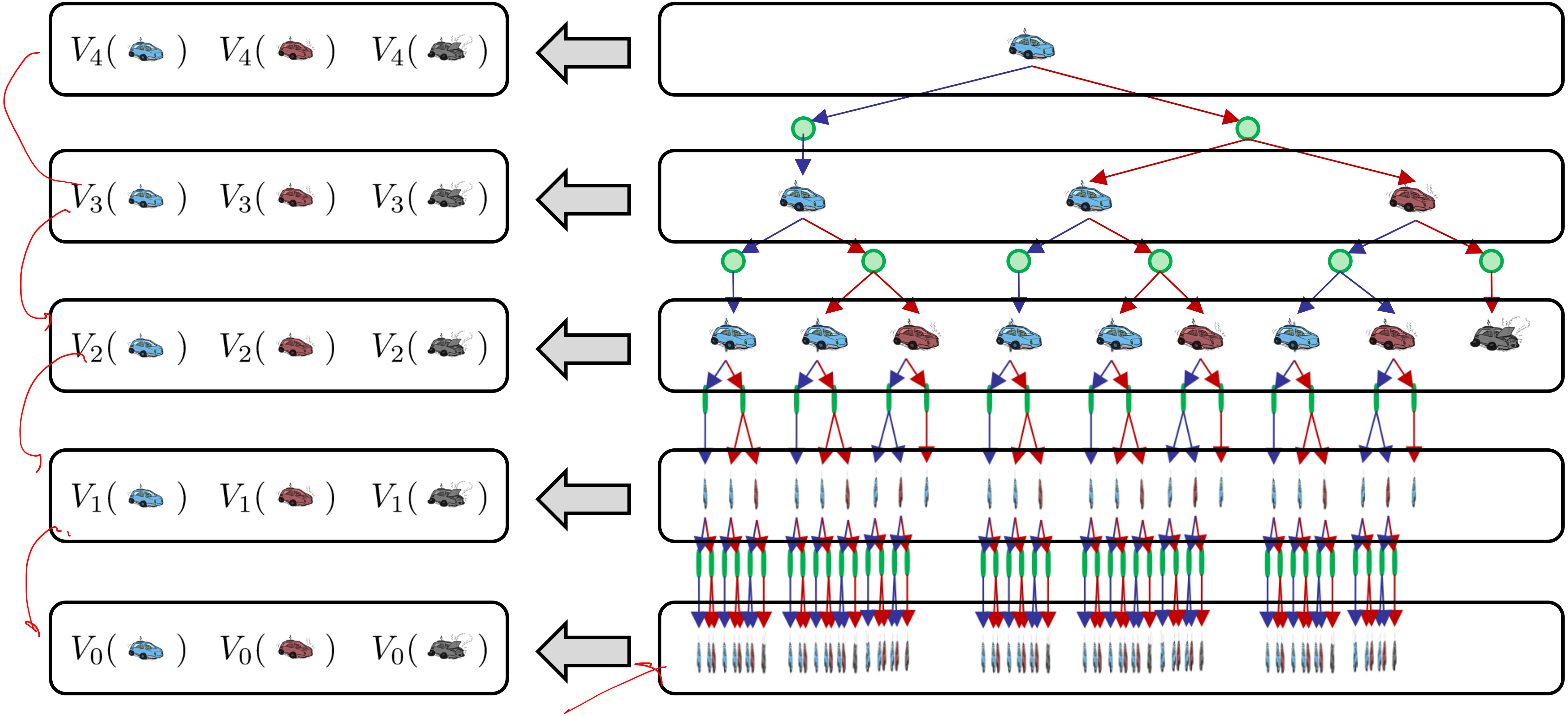


0.64 ▶	0.74 ▶	0.85 ▶	1.00
▲ 0.57		▲ 0.57	-1.00
▲ 0.49	◀ 0.43	▲ 0.48	◀ 0.28

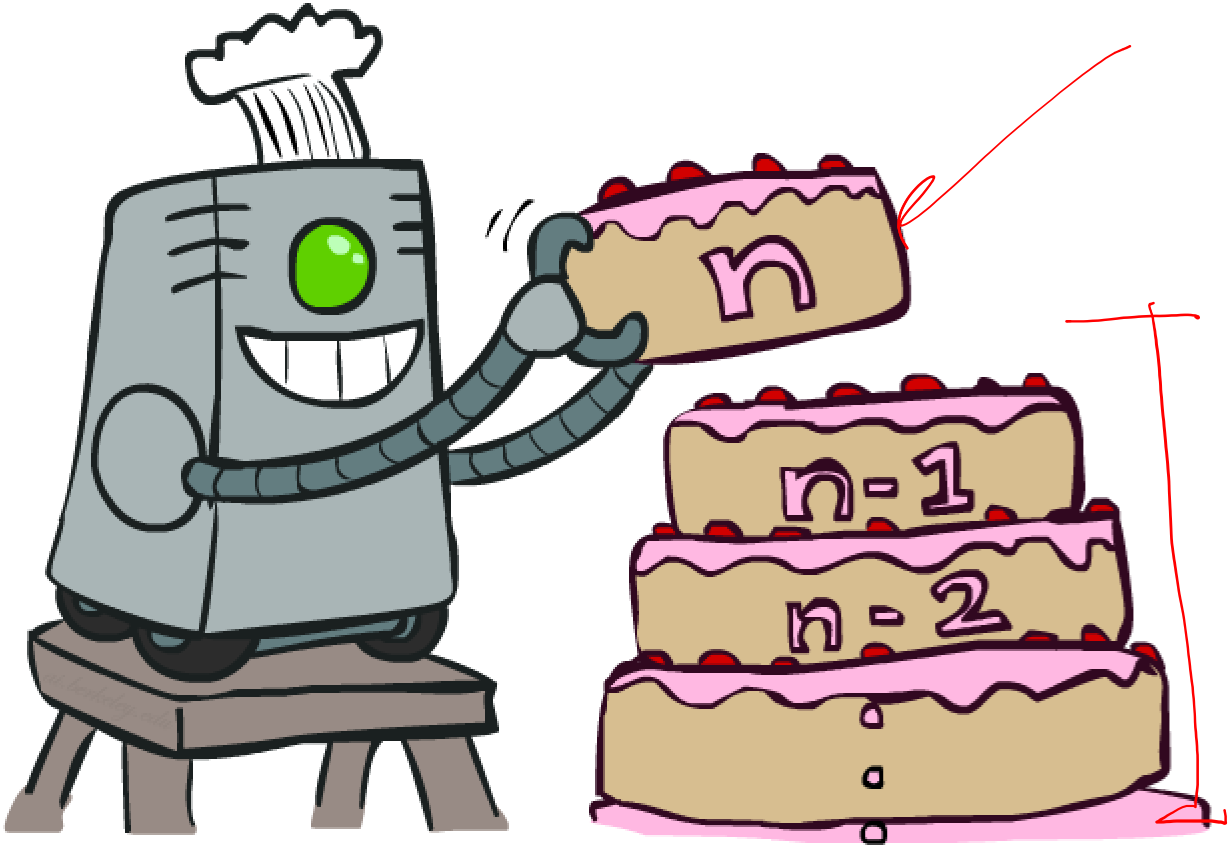
VALUES AFTER 100 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

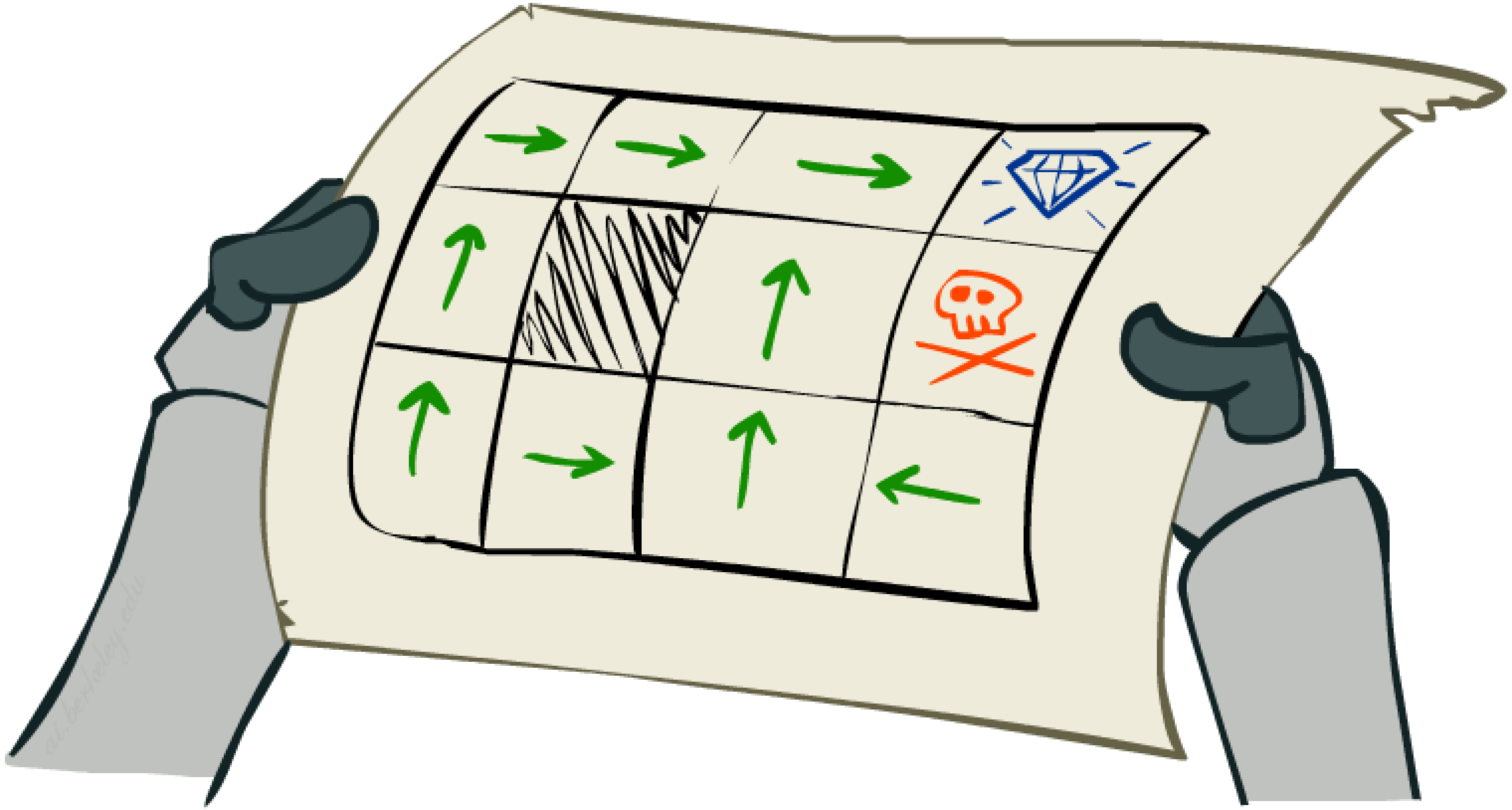
Computing Time-Limited Values



Value Iteration



Solving MDPs



Value Iteration

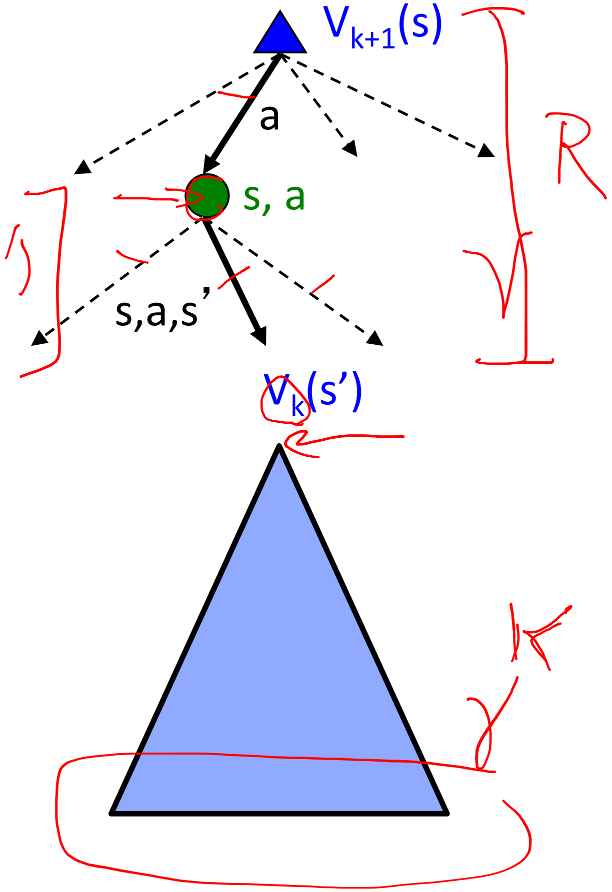
- Start with $V_0(s) = 0$: no time steps left means an expected reward sum of zero
- Given vector of $V_k(s)$ values, do one ply of expectimax from each state:

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

- Repeat until convergence

$$V_{K+1}(S) \leftarrow \max_a \sum_{S'} T(S,a,S')\left[R(S,a,S') + \gamma V_K(S')\right]$$

- 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



Example: Value Iteration

V_2

V_1

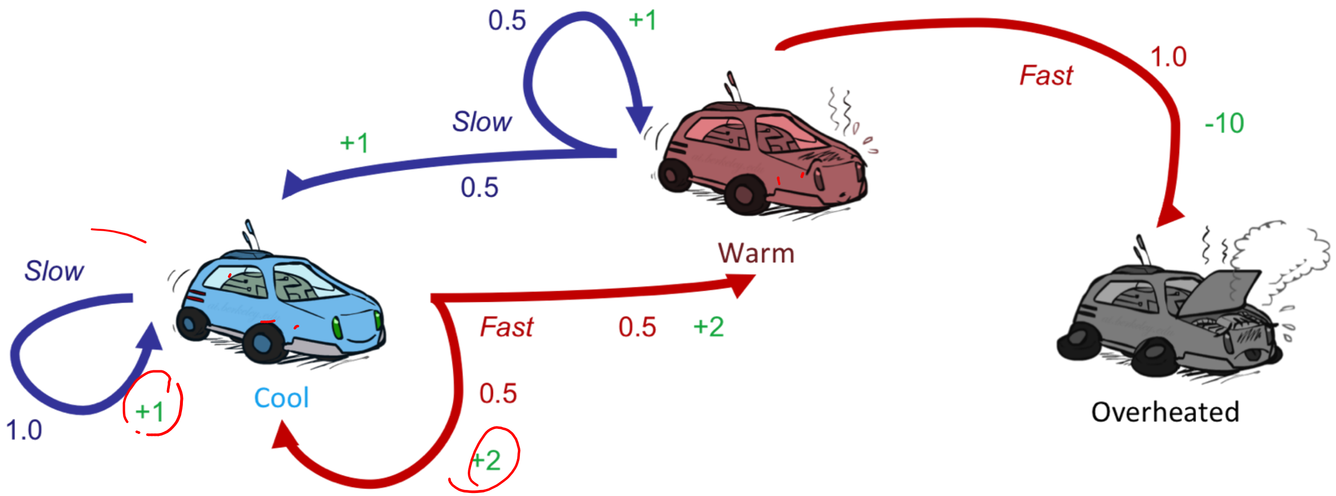
S: 1

F: .5*2+.5*2=2

V_0

0 0

0



Assume no discount!

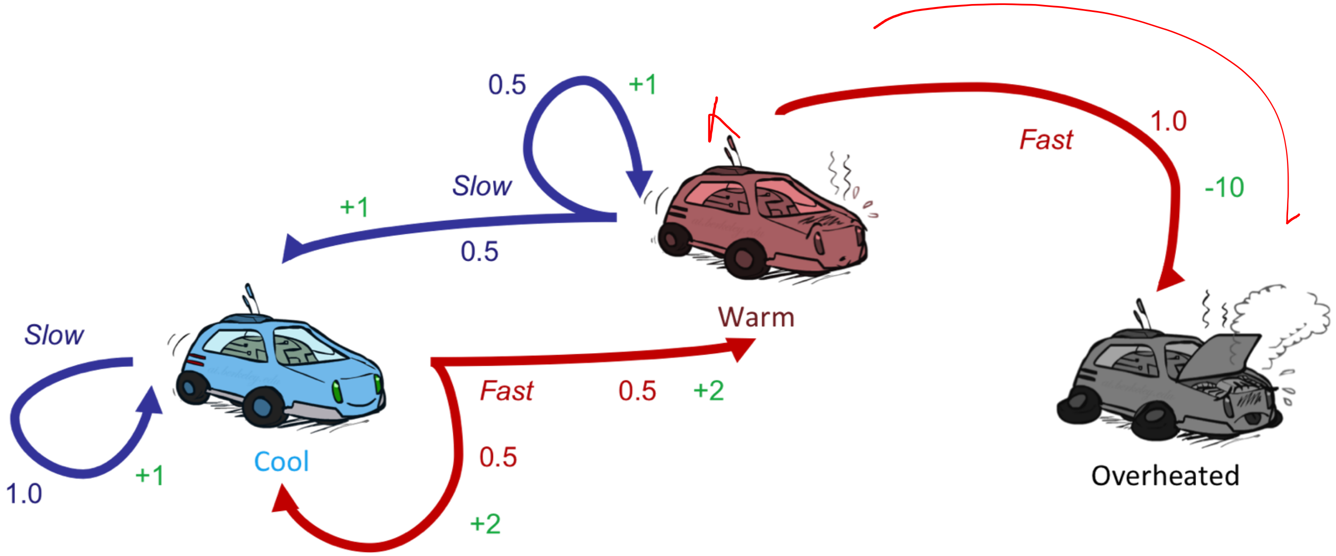
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right]$$

Example: Value Iteration

V_2

V_1 2 S: .5*1+.5*1=1
F: -10

V_0 0 0
0



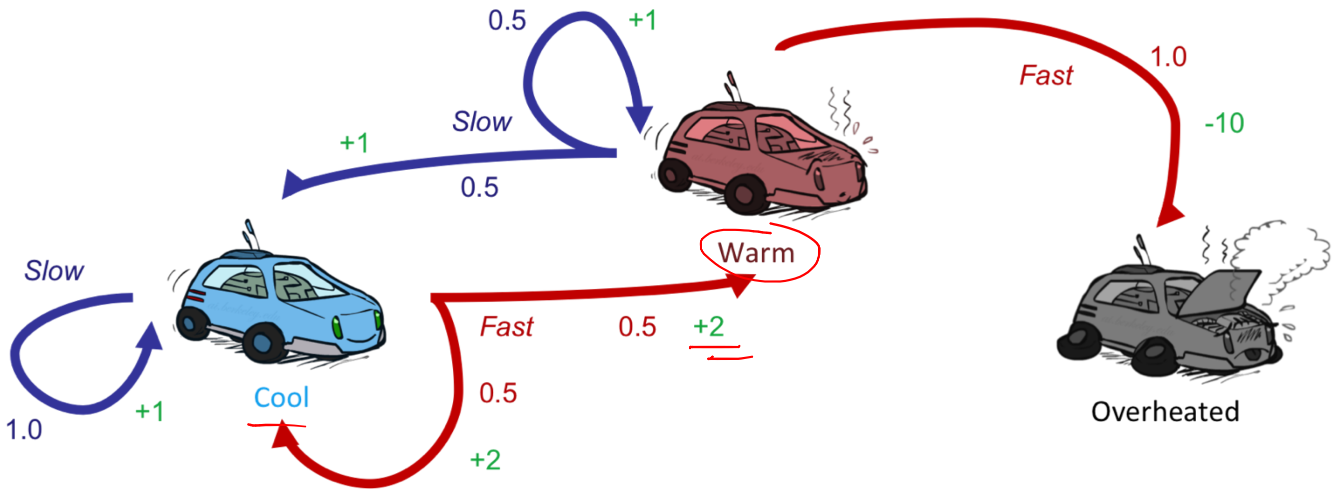
Assume no discount!

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

Example: Value Iteration

$f: 0.5[2+\gamma \times 2]+0.5[2+\gamma \times 1]=3.5$

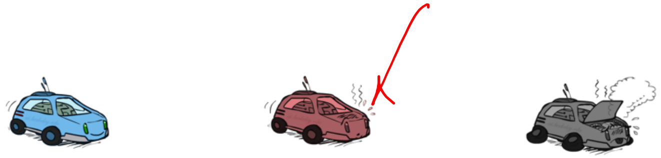
V_2			
V_1	2	1	0
V_0	0 0	0	



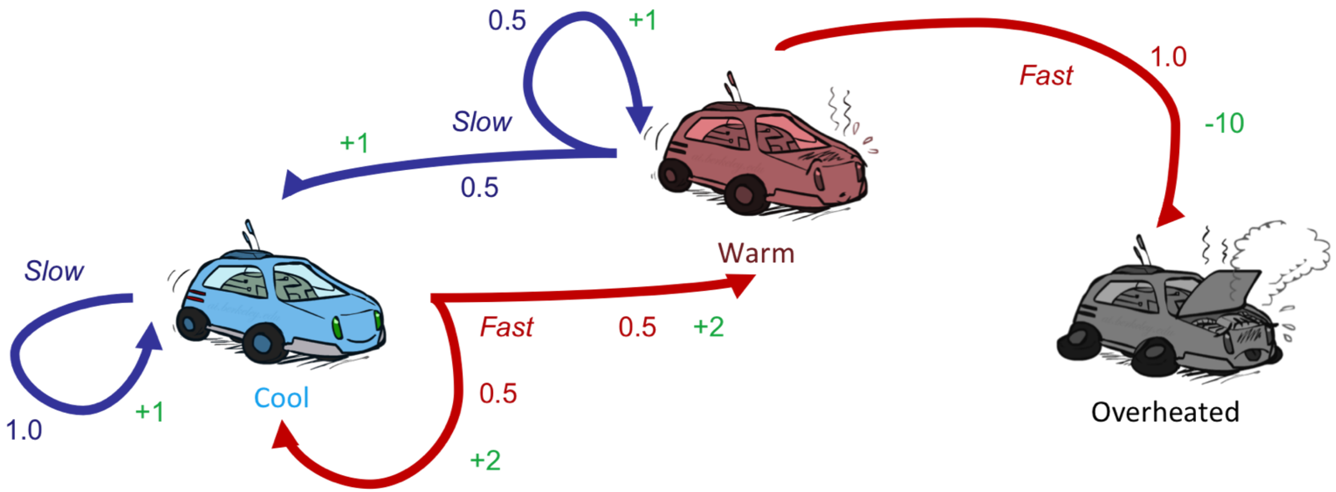
Assume no discount!

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

Example: Value Iteration



V_2	S: 1+2=3 F: .5*(2+2)+.5*(2+1)=3.5		
V_1	2	1	0
V_0	0	0	0



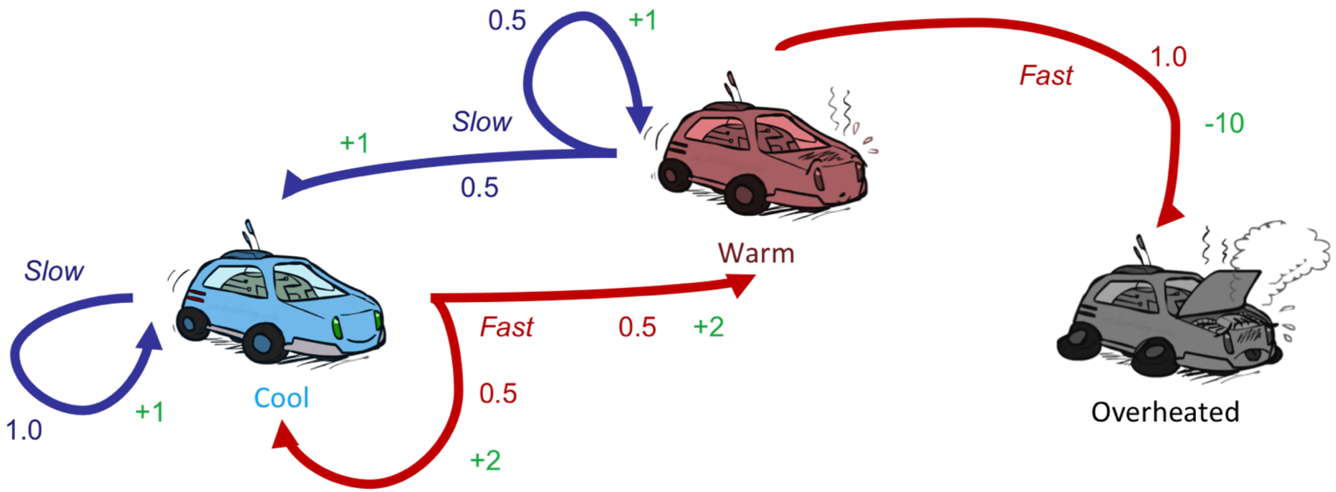
Assume no discount!

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

Example: Value Iteration

V_2	3.5	2.5	0
V_1	2	1	0
V_0	0	0	

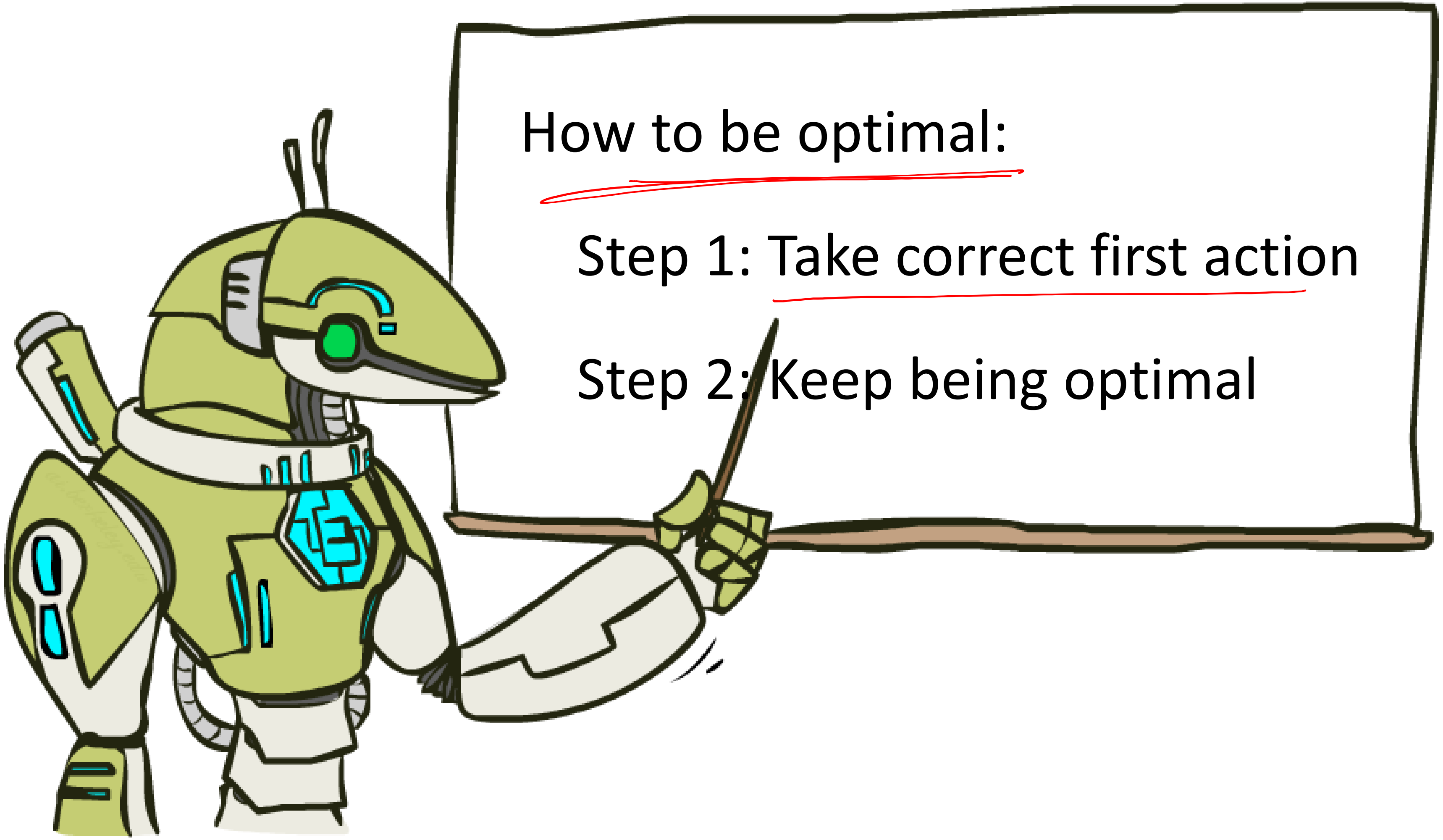
0



Assume no discount!

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

The Bellman Equations



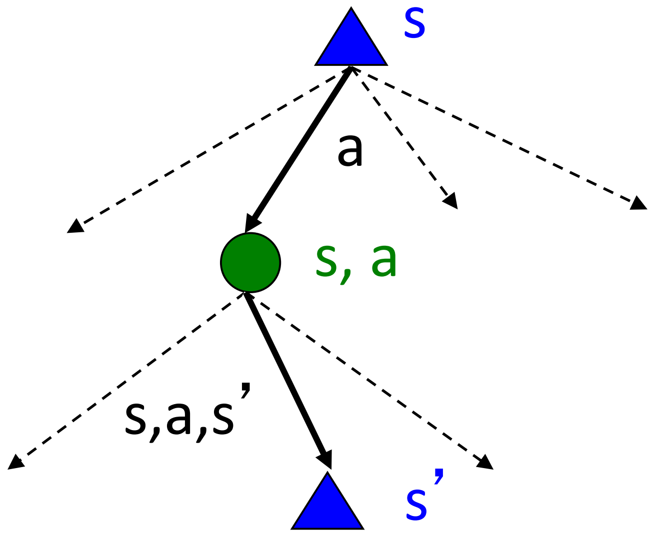
How to be optimal:

Step 1: Take correct first action

Step 2: Keep being optimal

The Bellman Equations

- Definition of "optimal utility" via expectimax recurrence gives a simple one-step lookahead relationship amongst optimal utility values



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

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

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

- These are the Bellman equations, and they characterize optimal values in a way we'll use over and over

Value Iteration

- Bellman equations characterize the optimal values:

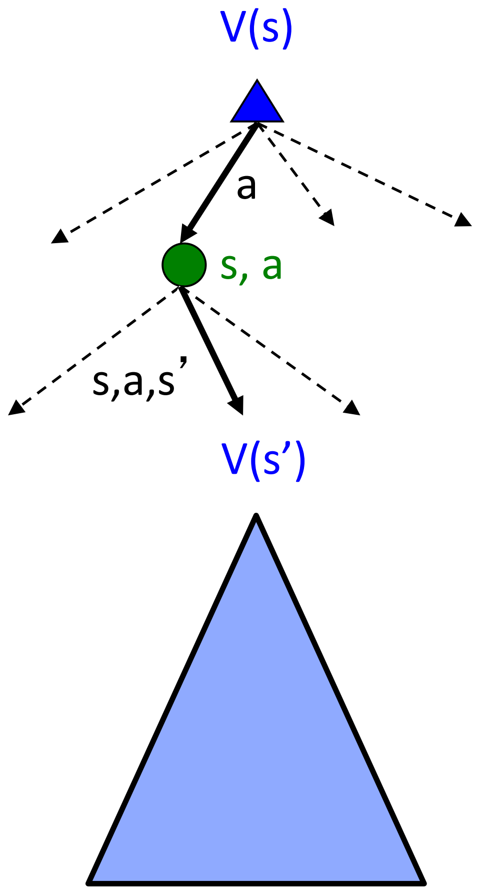
$$V^*(s) \boxed{=} \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s') \right]$$

- Value iteration computes them:

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

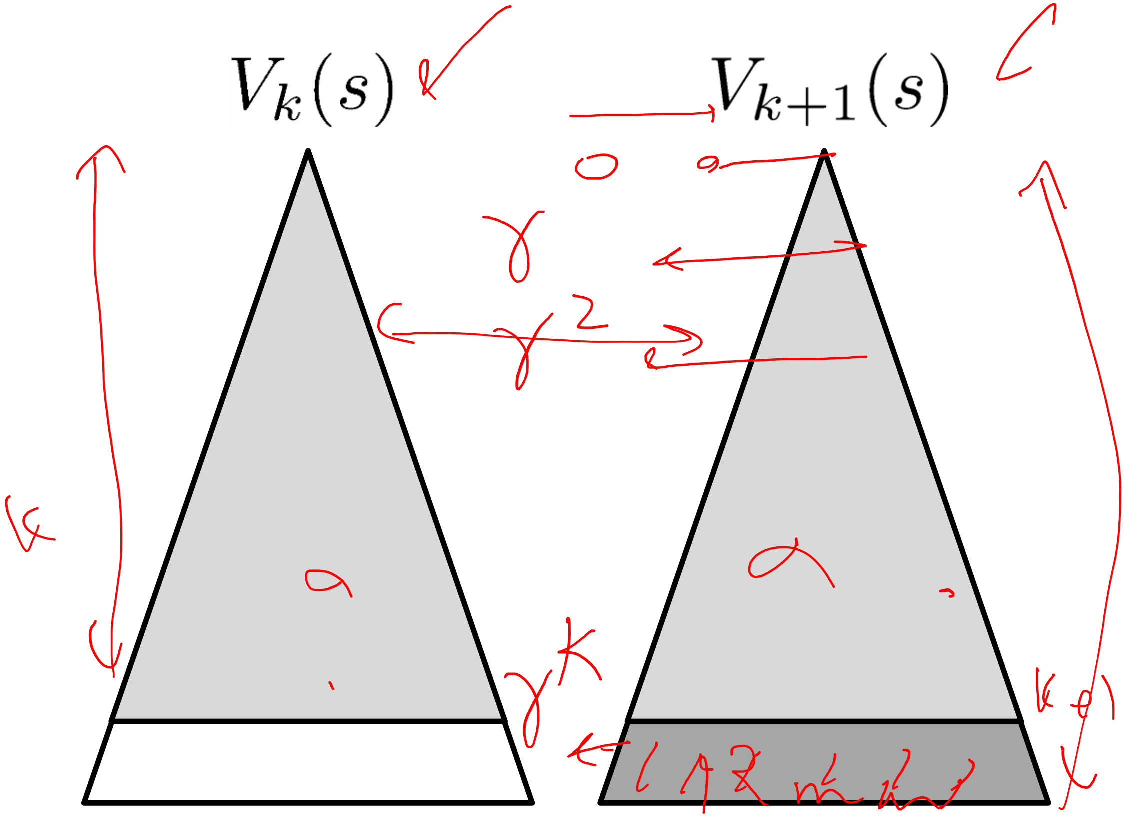
$V_0(s) = 0$

- Value iteration is just a fixed point solution method
  - … though the $\underline{V_k \text{ vectors}}$ are also interpretable as time-limited values



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 expectimax 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

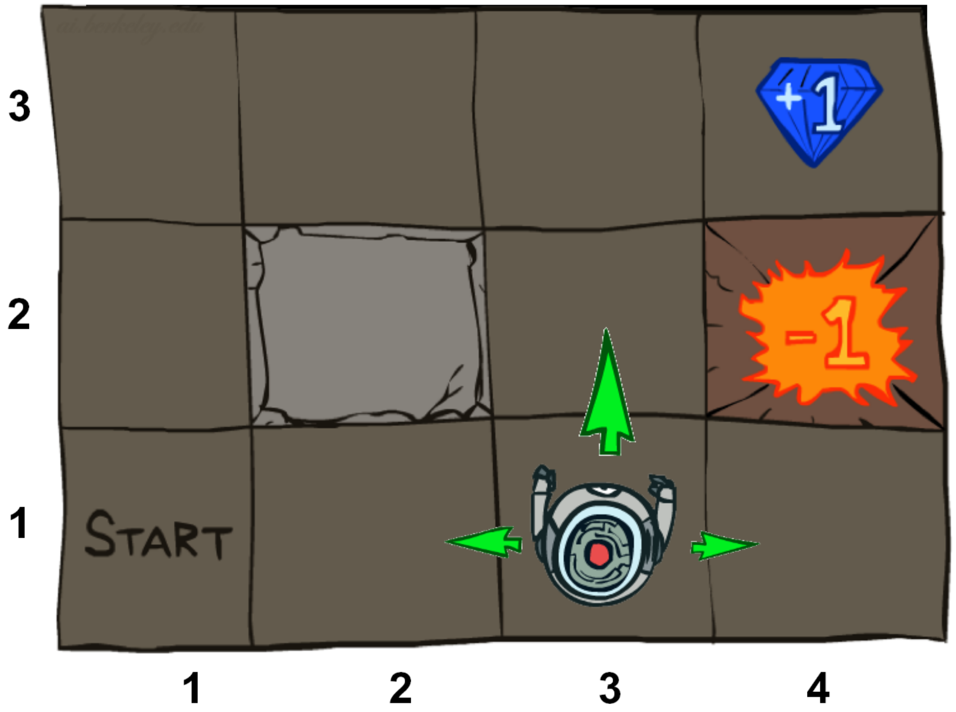


$\gamma^k \times |R_{max}|$

R_{min}

Recap: 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')
    - 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
- MDPs are non-deterministic search problems
  - One way to solve them is with expectimax search
  - We'll have a new tool soon



Announcements

- PS2: Feb 4th
- Next class: Vote?
- Project proposals: Feb 11th
- Paper review: Feb 18th

Recap: MDPs

- Search problems in uncertain environments
  - Model uncertainty with transition function
  - Assign utility to states. How? Using reward functions

  - Decision making and search in MDPs <-- Find a sequence of actions that maximize expected sum of rewards
    - Value of a state
    - Q-Value of a state
    - Policy for a state

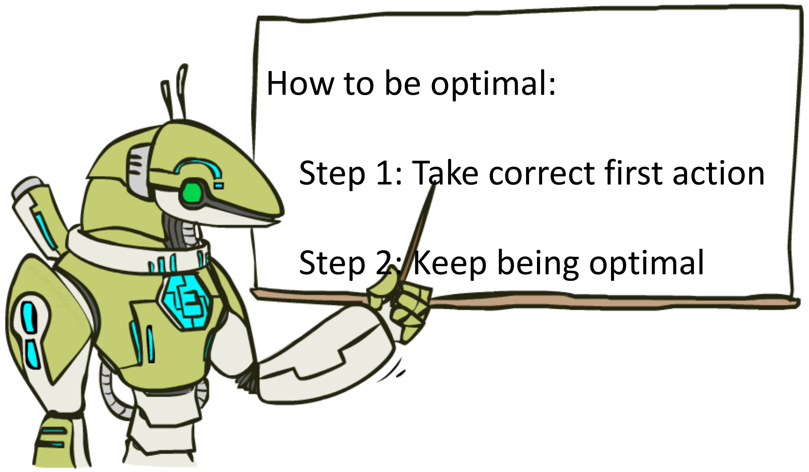
The Bellman Equations

- Definition of “optimal utility” via expectimax recurrence gives a simple one-step lookahead relationship amongst optimal utility values

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

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

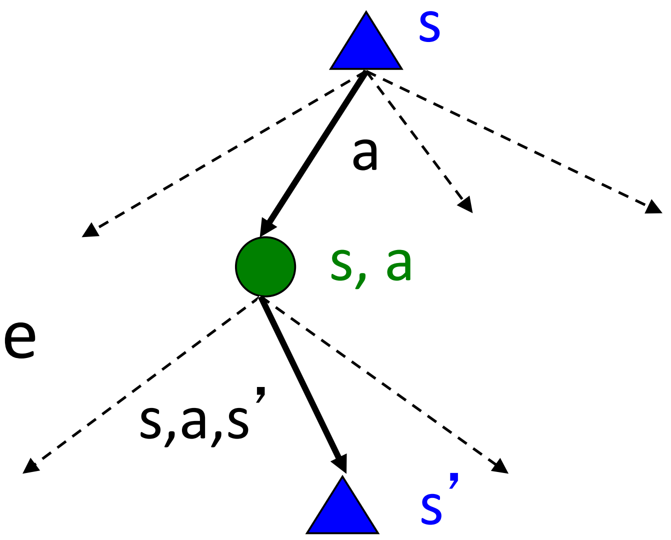
$$V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s') \right]$$



How to be optimal:

Step 1: Take correct first action

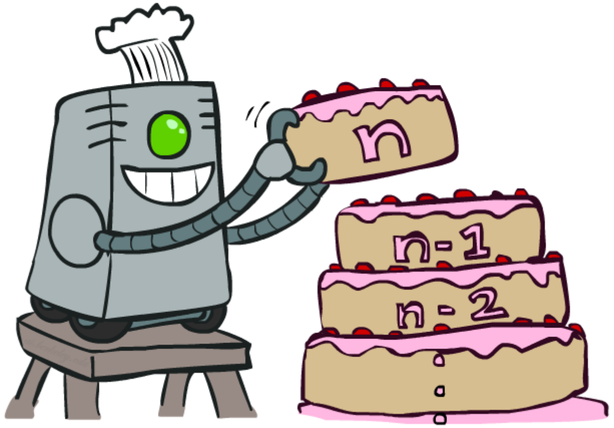
Step 2: Keep being optimal



- These are the Bellman equations, and they characterize optimal values in a way we’ll use over and over

Solving MDPs

- Finding the best policy → mapping of actions to states
- So far, we have talked about one method
    - Value iteration: computes the **optimal** values of states



Value Iteration

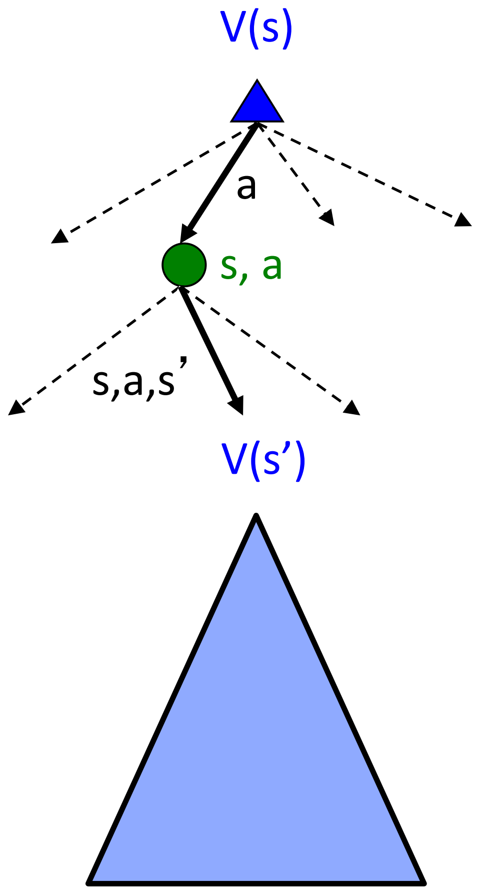
- Bellman equations characterize the optimal values:

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

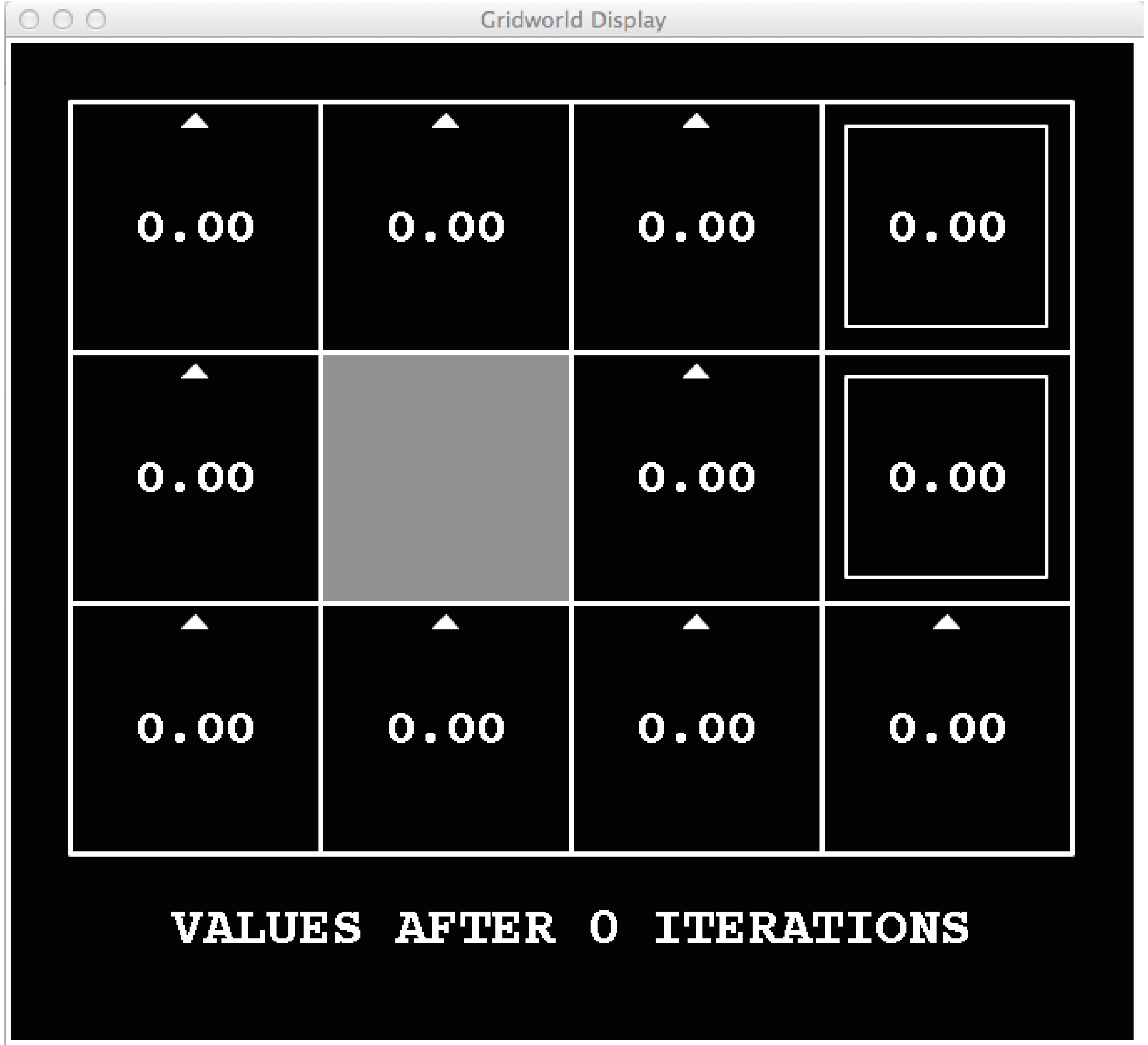
- Value iteration computes them:

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

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



k=0

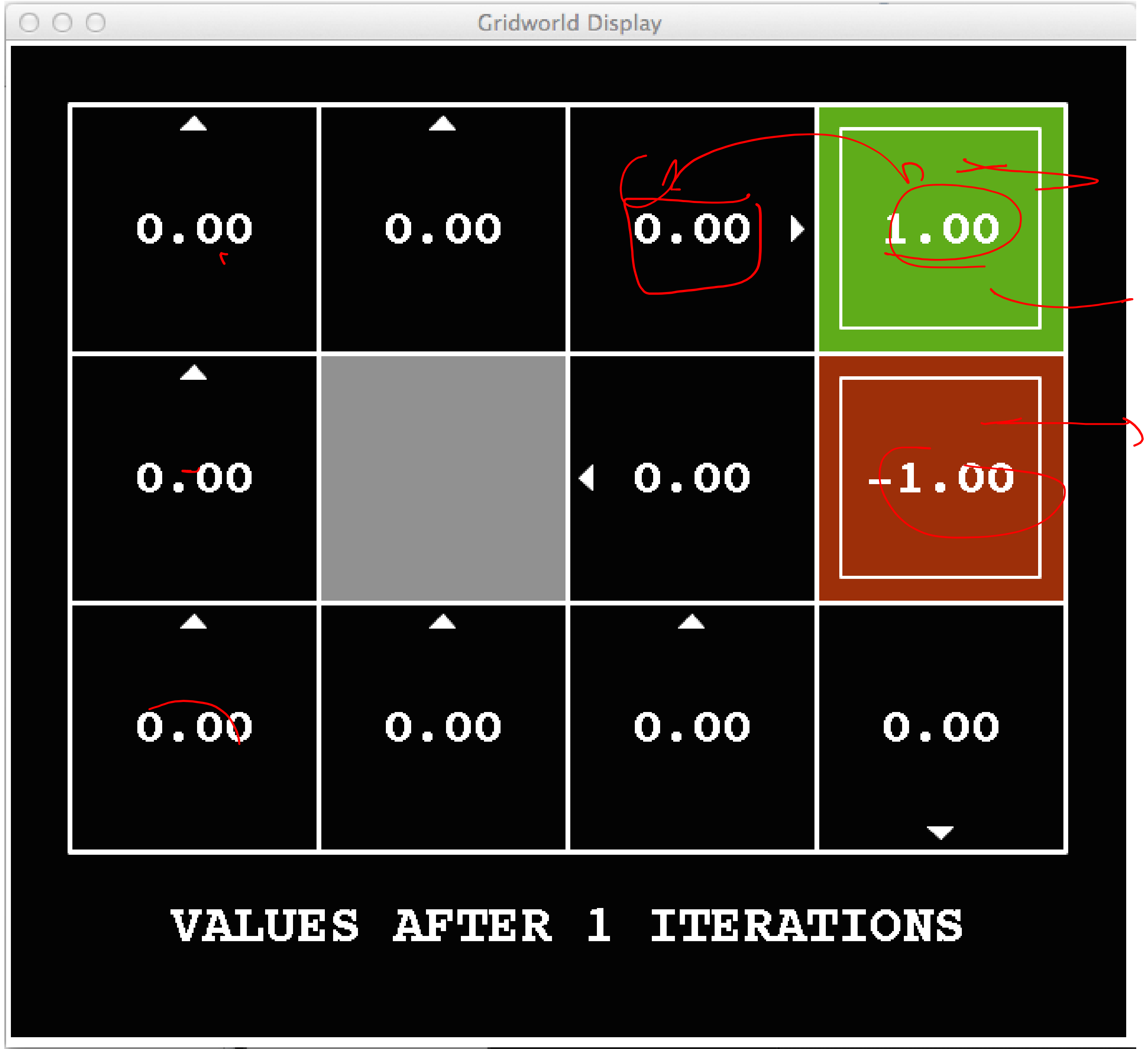


▲ 0.00	▲ 0.00	▲ 0.00	0.00
▲ 0.00		▲ 0.00	0.00
▲ 0.00	▲ 0.00	▲ 0.00	▲ 0.00

VALUES AFTER 0 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

k=1



▲ 0.00	▲ 0.00	0.00 ▶	1.00
▲ 0.00	▨	◀ 0.00	-1.00
▲ 0.00	▲ 0.00	▲ 0.00	0.00 ▼

VALUES AFTER 1 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

k=2

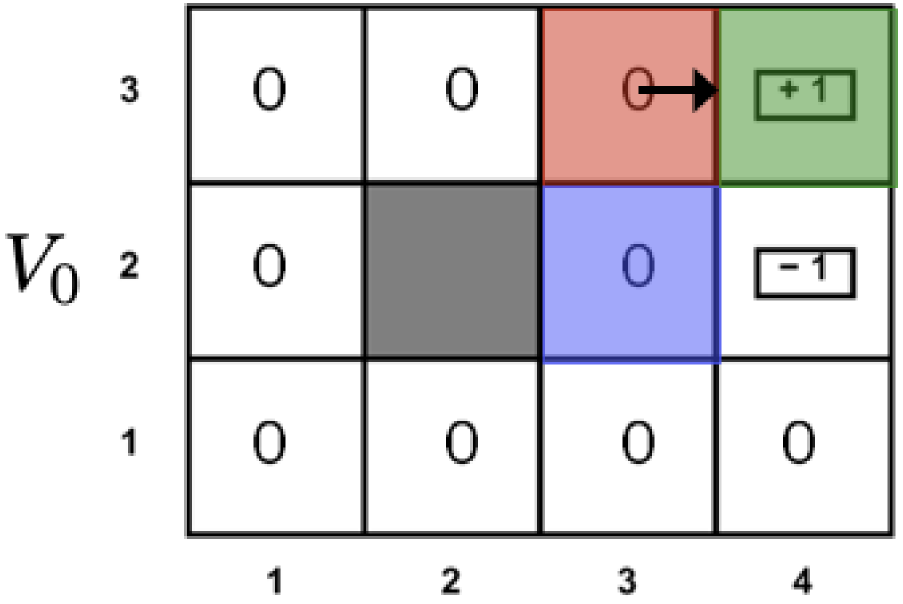
▲ 0.00	0.00 ▶	0.72 ▶	1.00
▲ 0.00		▲ 0.00	-1.00
▲ 0.00	▲ 0.00	▲ 0.00	0.00 ▼

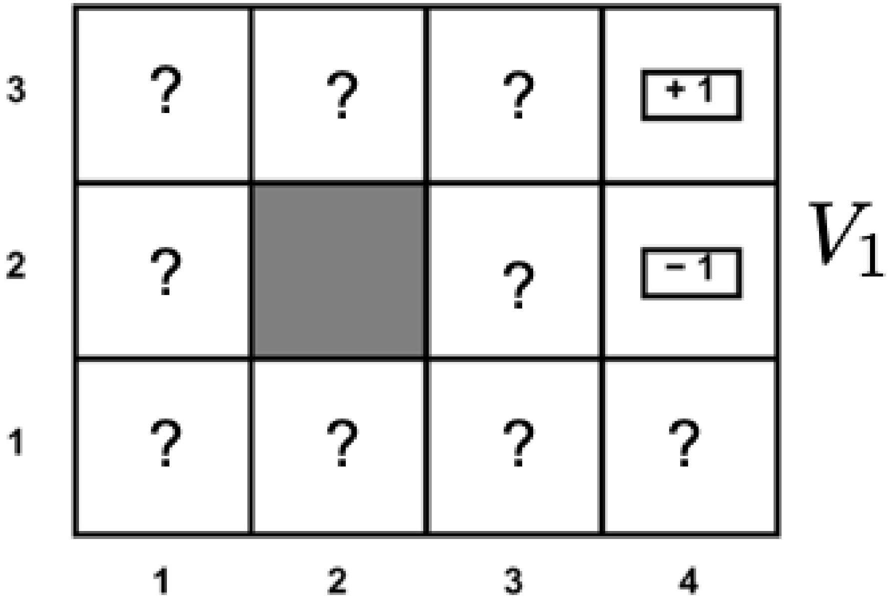
VALUES AFTER 2 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

Bellman Updates

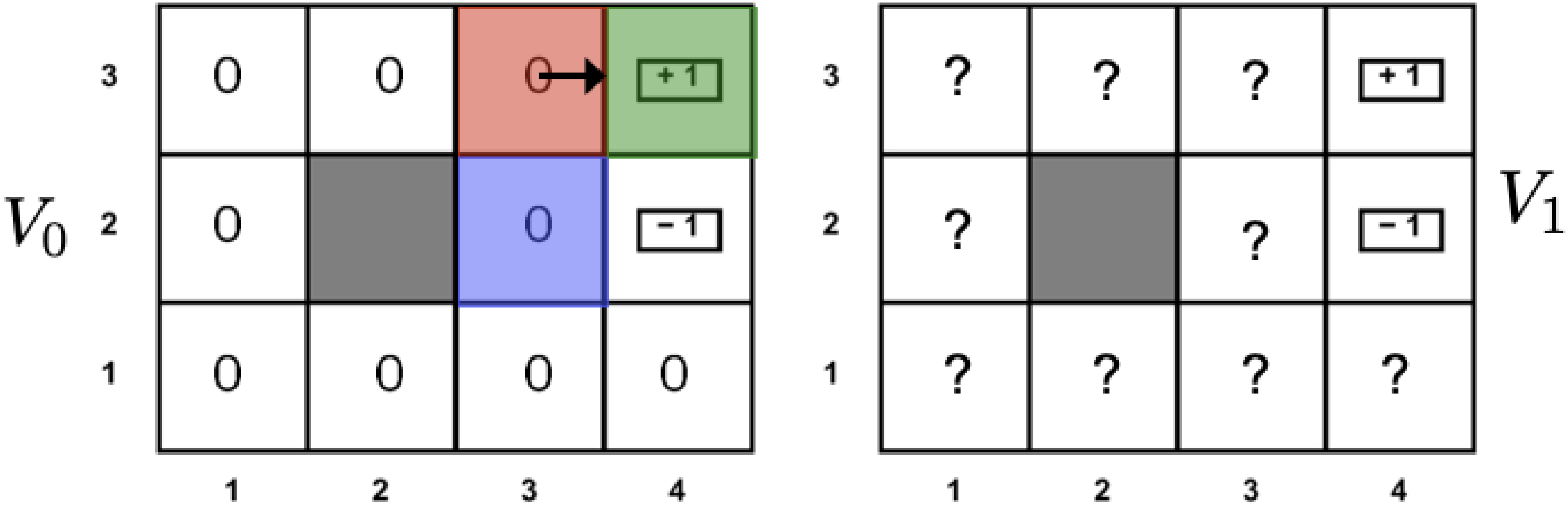
Example: γ=0.9, living reward=0, noise=0.2





Bellman Updates

Example: γ=0.9, living reward=0, noise=0.2



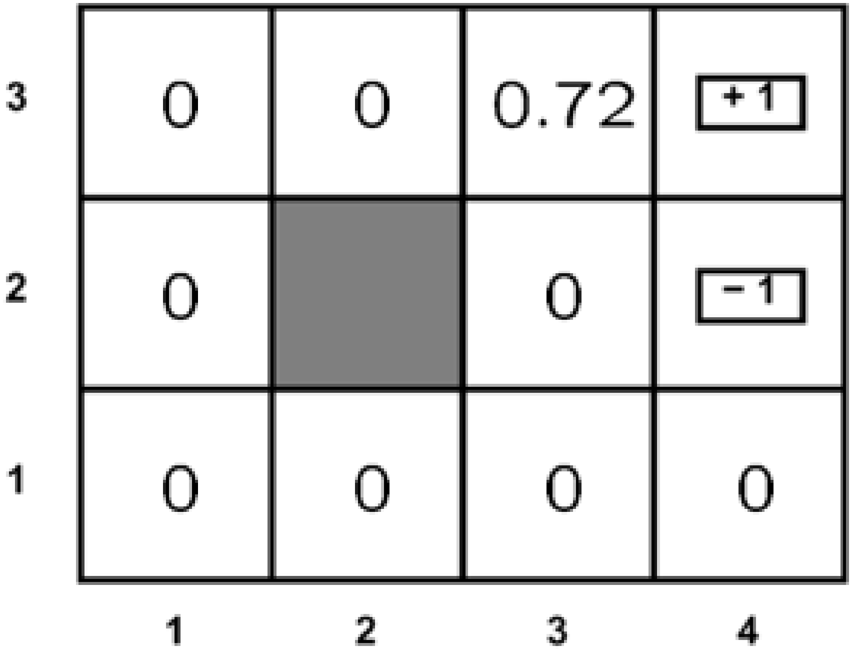
$$V_{i+1}(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_i(s')\right] = \max_a Q_{i+1}(s,a)$$

$$Q_1(\langle 3,3\rangle, \text{right}) = \sum_{s'} T(\langle 3,3\rangle, \text{right}, s') \left[R(\langle 3,3\rangle, \text{right}, s') + \gamma V_i(s')\right]$$

$$= 0.8 * [0.0 + 0.9 * 1.0] + 0.1 * [0.0 + 0.9 * 0.0] + 0.1 * [0.0 + 0.9 * 0.0]$$

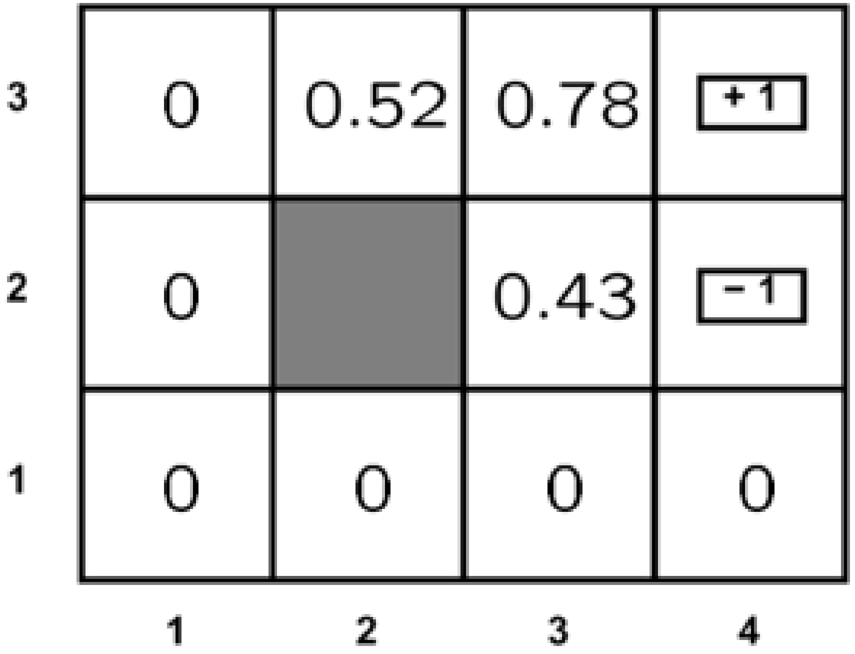
Example: Value Iteration

V_1



3	0	0	0.72	+1
2	0		0	-1
1	0	0	0	0
	1	2	3	4

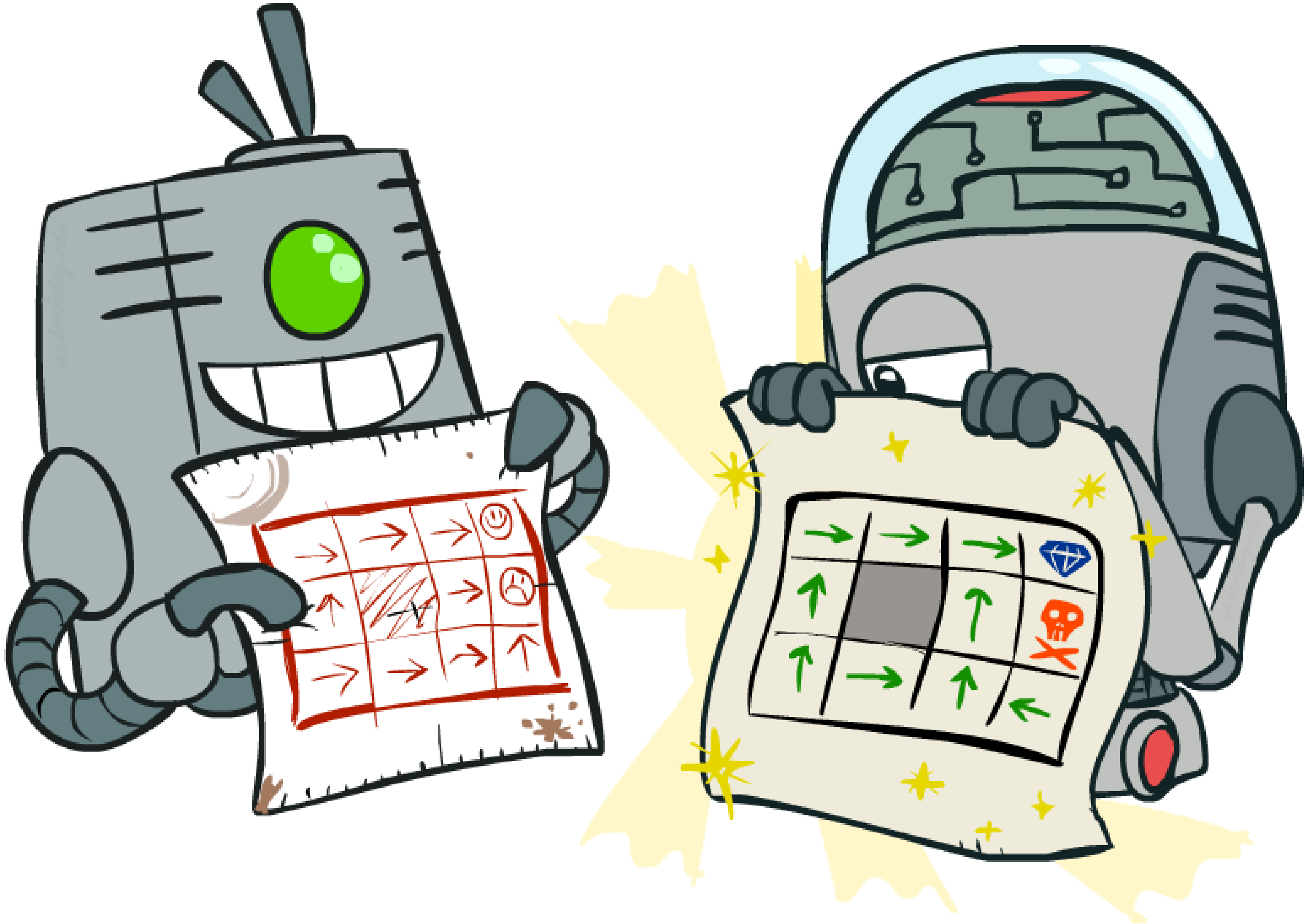
V_2



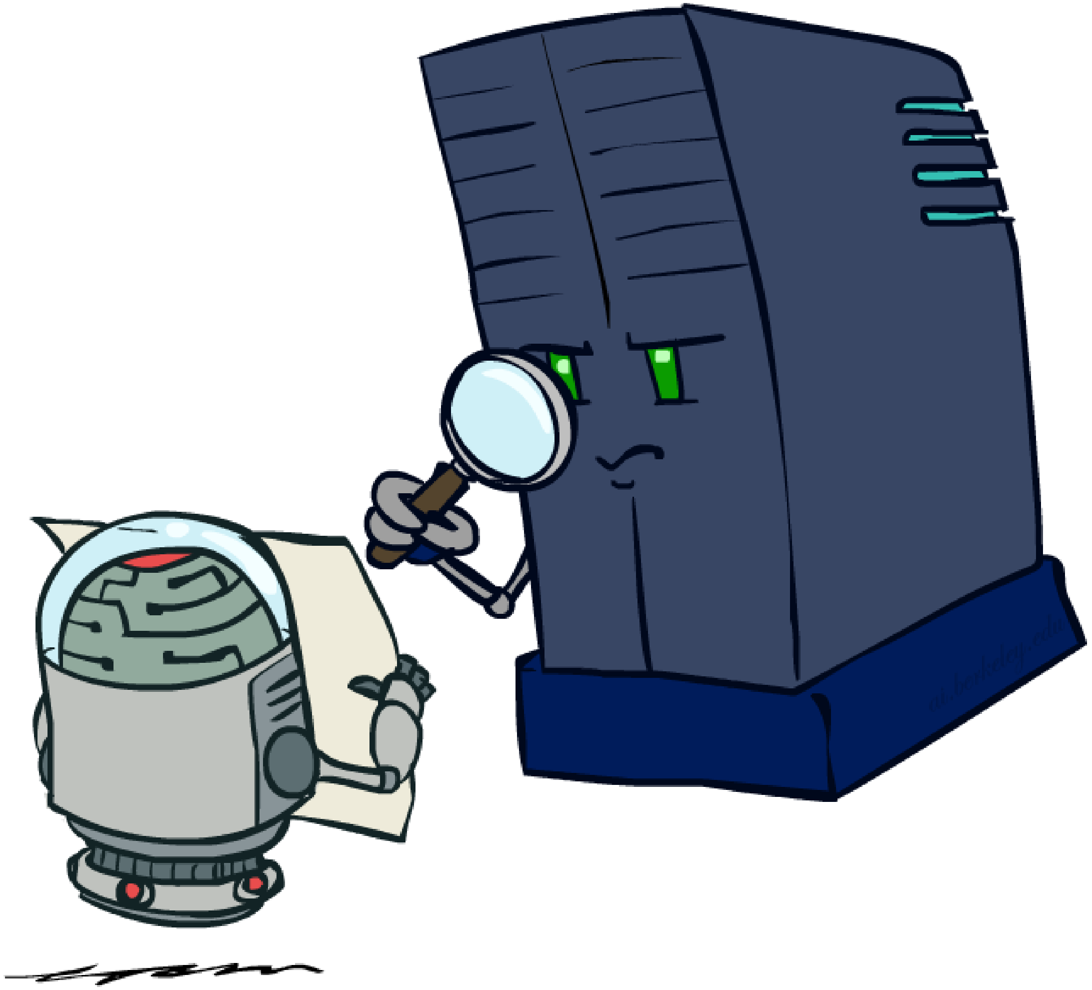
3	0	0.52	0.78	+1
2	0		0.43	-1
1	0	0	0	0
	1	2	3	4

- Information propagates outward from terminal states and eventually all states have correct value estimates

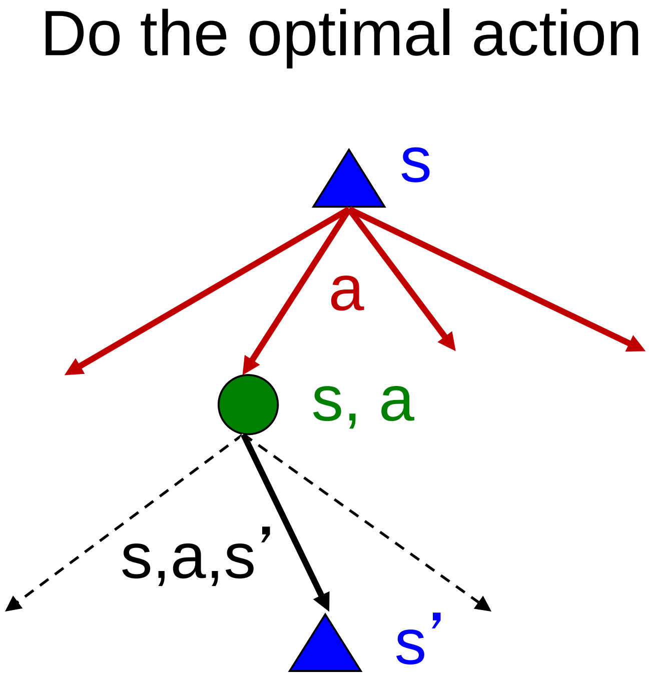
Policy Methods

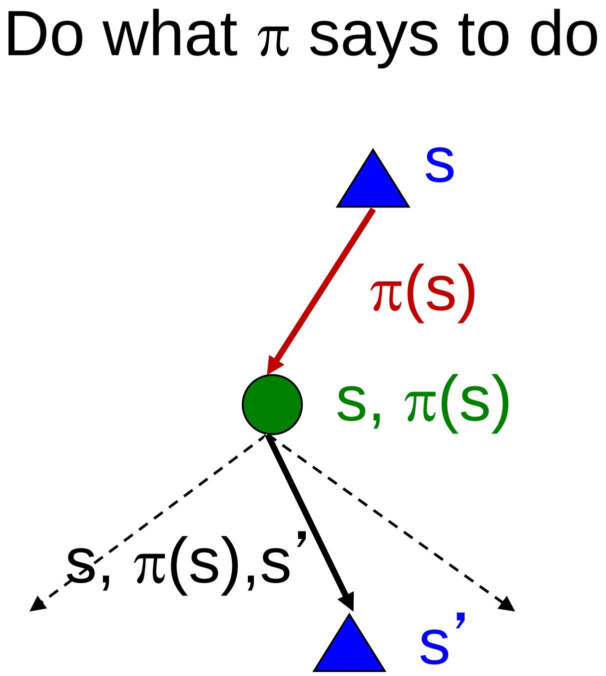


Policy Evaluation



Fixed Policies





- Expectimax 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

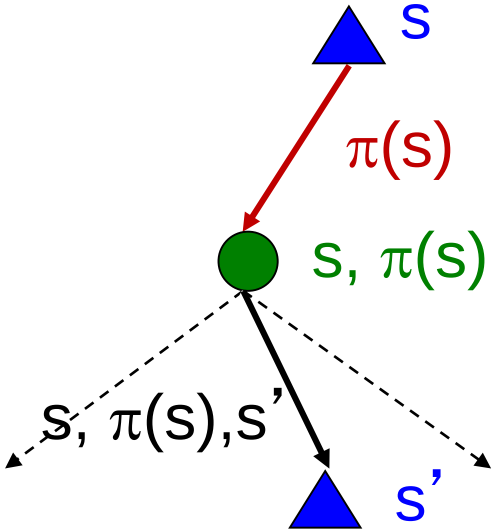
Utilities for a Fixed Policy

- 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:
  - $V^{\pi}(s)$ = expected total discounted rewards starting in s and following π

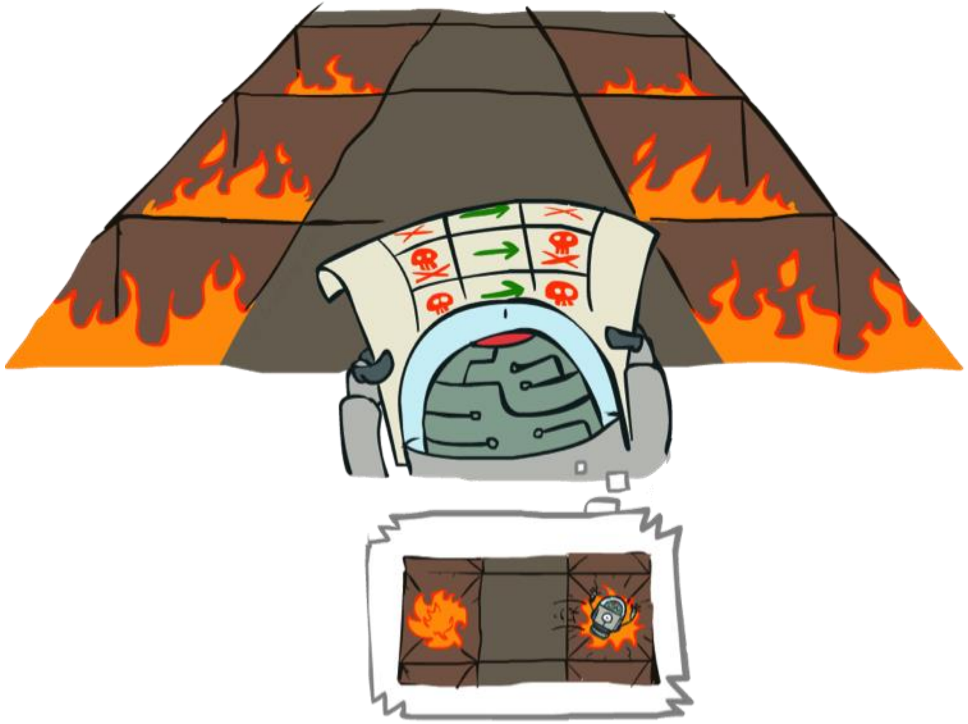
- Recursive relation (one-step look-ahead / Bellman equation):

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

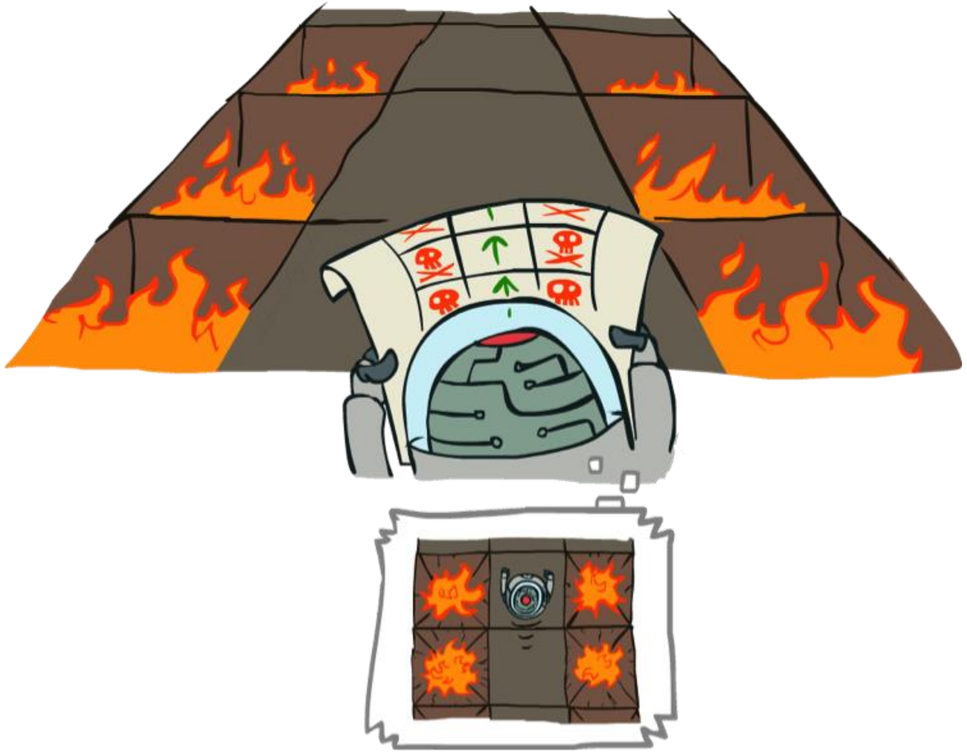


Example: Policy Evaluation

Always Go Right



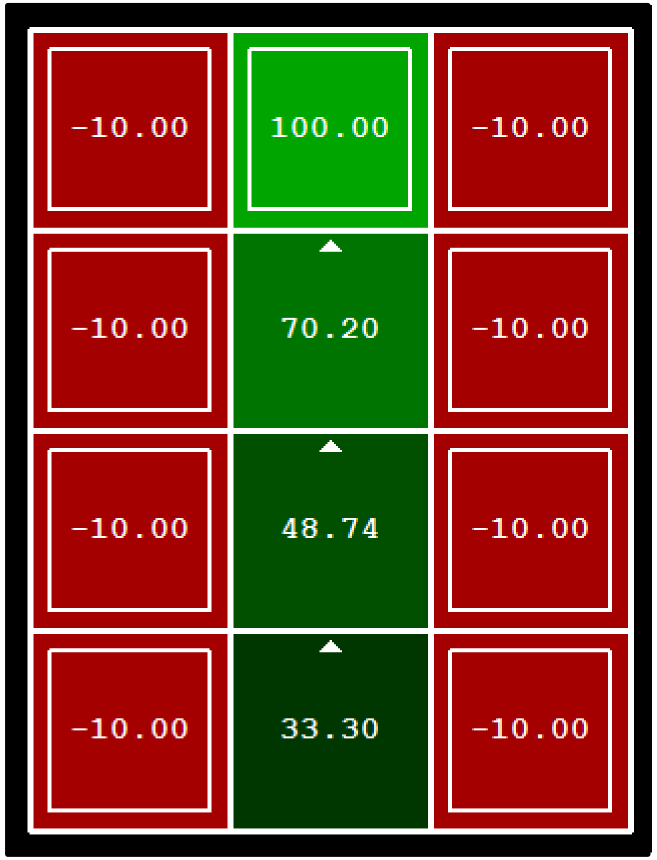
Always Go Forward



Example: Policy Evaluation

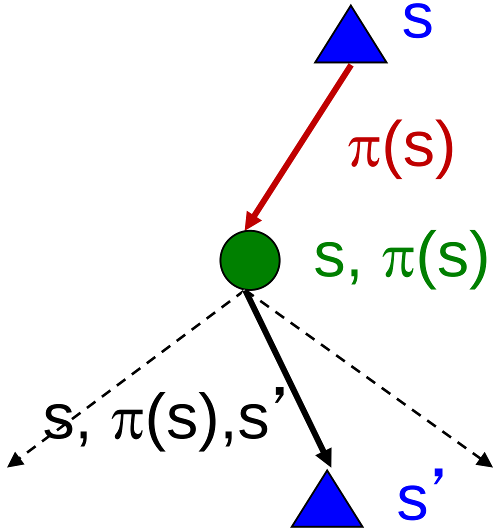
Always Go Right

Always Go Forward



Policy Evaluation

- How do we calculate the V's for a fixed policy π?



- Idea 1: Turn recursive Bellman equations into updates (like value iteration)

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

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

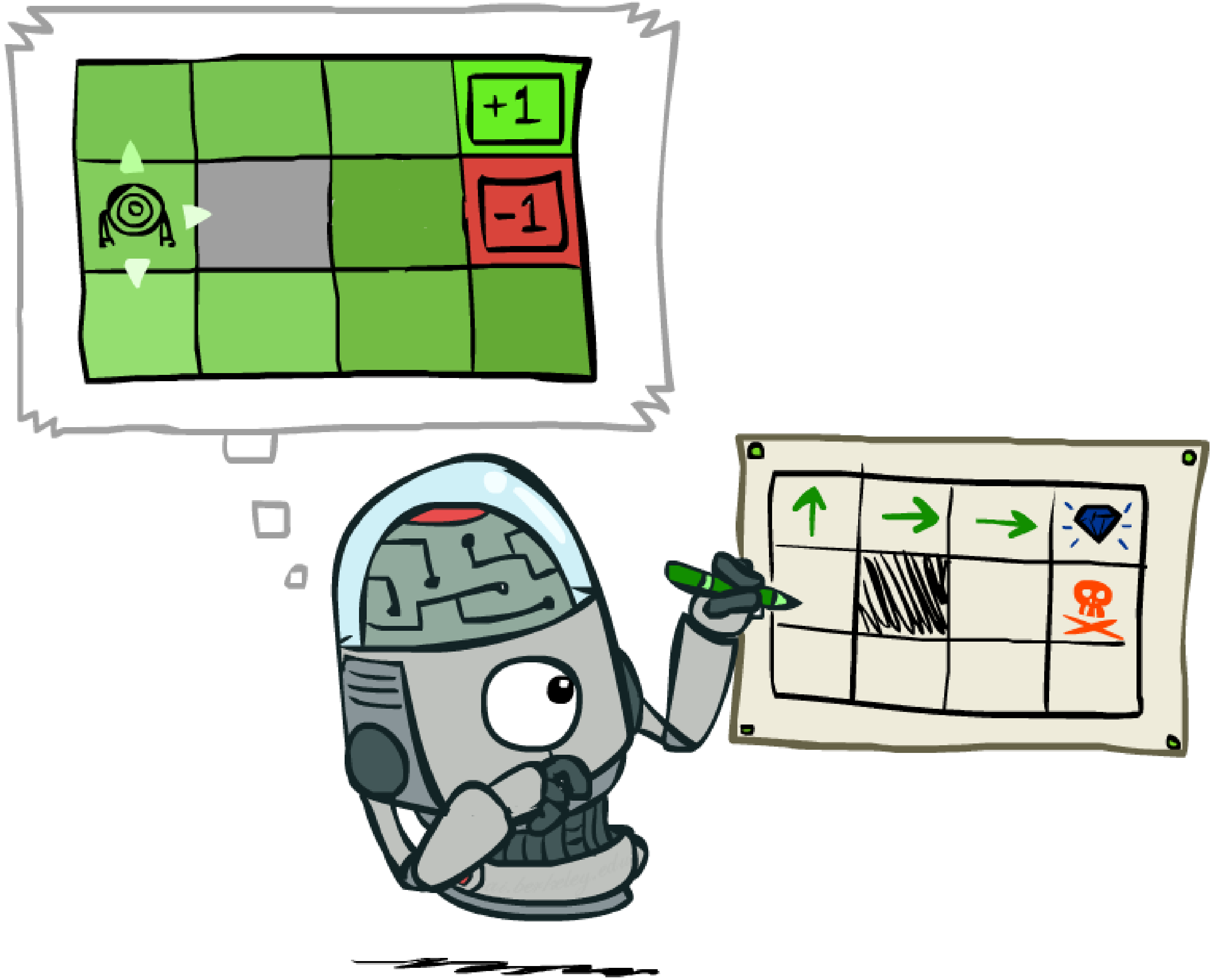
- Efficiency: O(S^2) per iteration

- Idea 2: Without the maxes, the Bellman equations are just a linear system
  - Solve with Matlab (or your favorite linear system solver)

Let’s think…

- Take a minute, think about value iteration and policy evaluation
    - Write down the biggest questions you have about them.

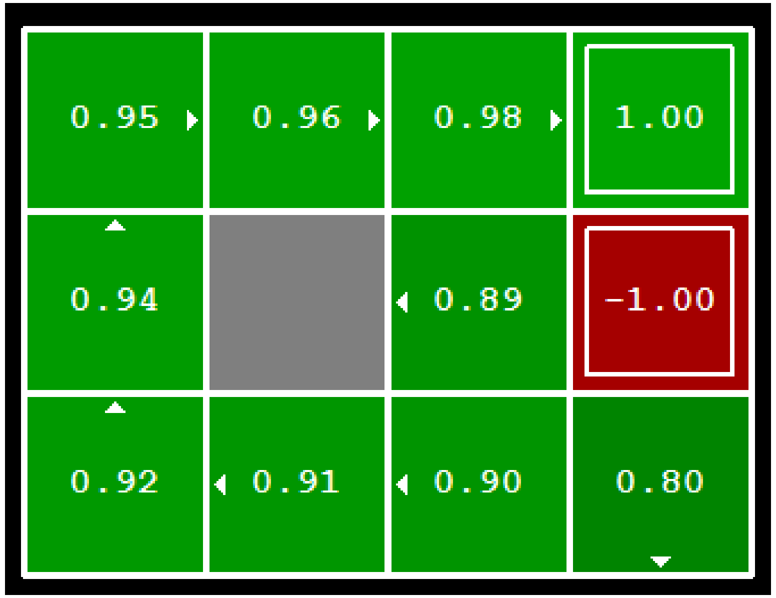
Policy Extraction



Computing Actions from Values

- Let’s imagine we have the optimal values V*(s)

- How should we act?
  - It’s not obvious!



0.95 ▸	0.96 ▸	0.98 ▸	1.00
▴ 0.94		◂ 0.89	-1.00
▴ 0.92	◂ 0.91	◂ 0.90	0.80 ▾

- We need to do a mini-expectimax (one step)

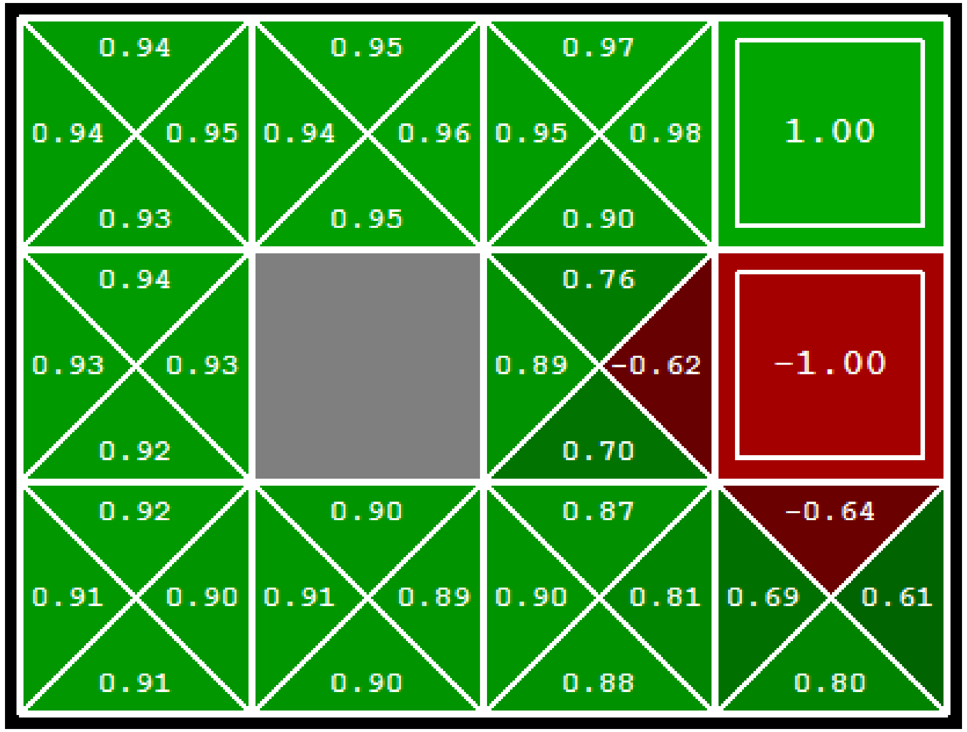
$$\pi^*(s) = \arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

- This is called policy extraction, since it gets the policy implied by the values

Computing Actions from Q-Values

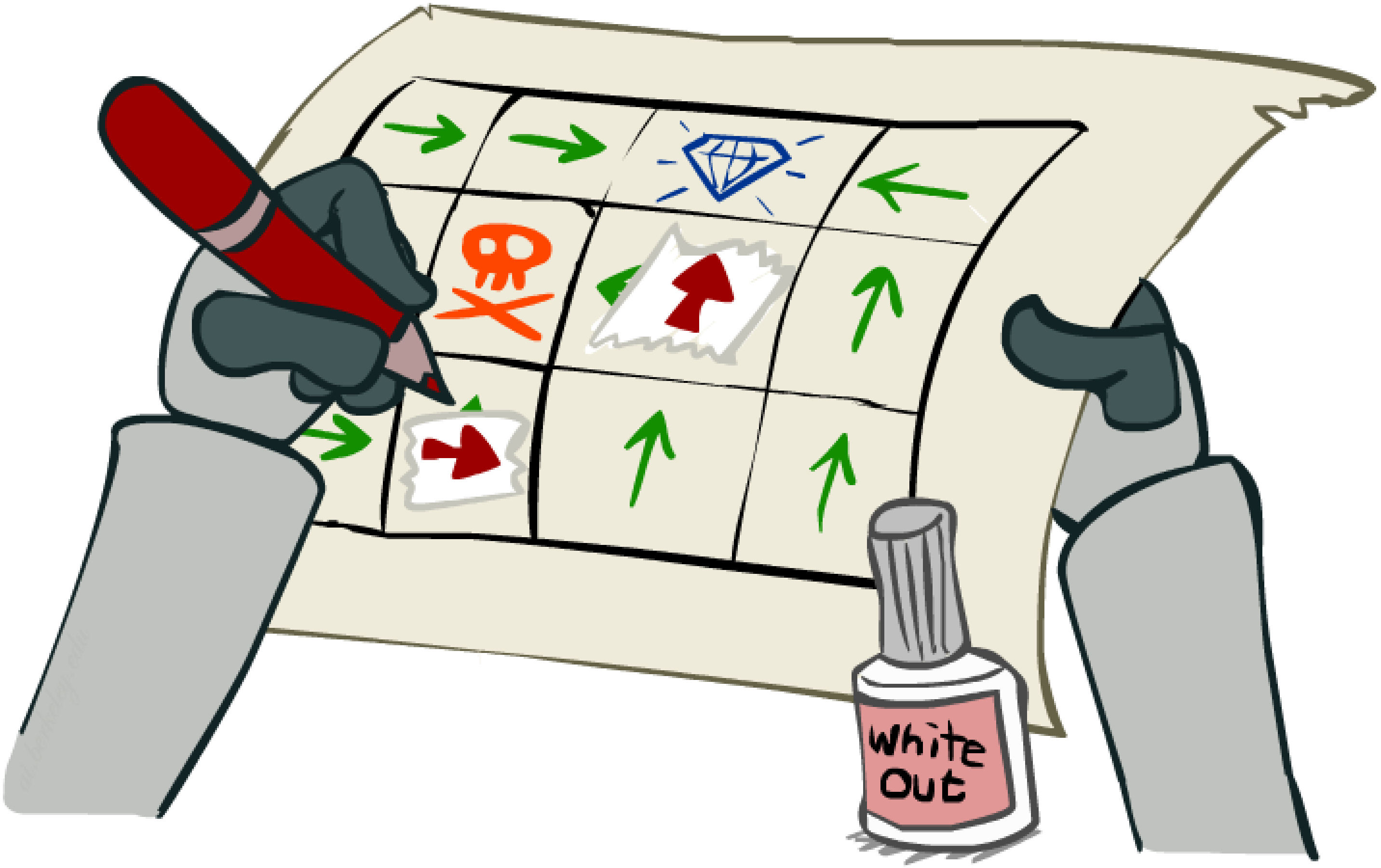
- Let's imagine we have the optimal q-values:
- How should we act?
  - Completely trivial to decide!

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



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

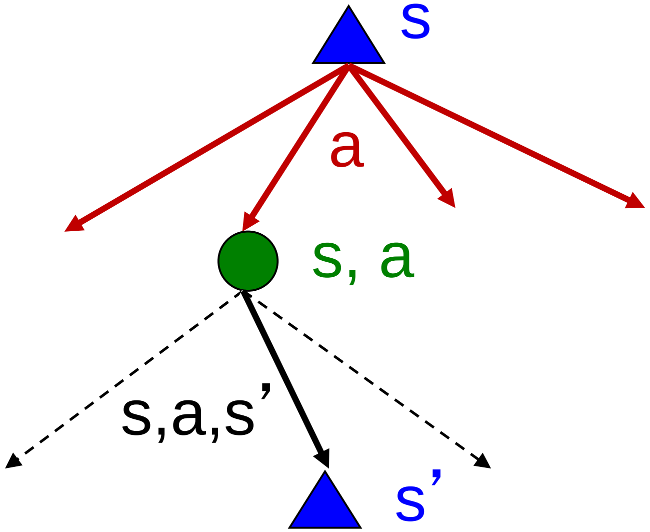
Policy Iteration



Problems with Value Iteration

- Value iteration repeats the Bellman updates:

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

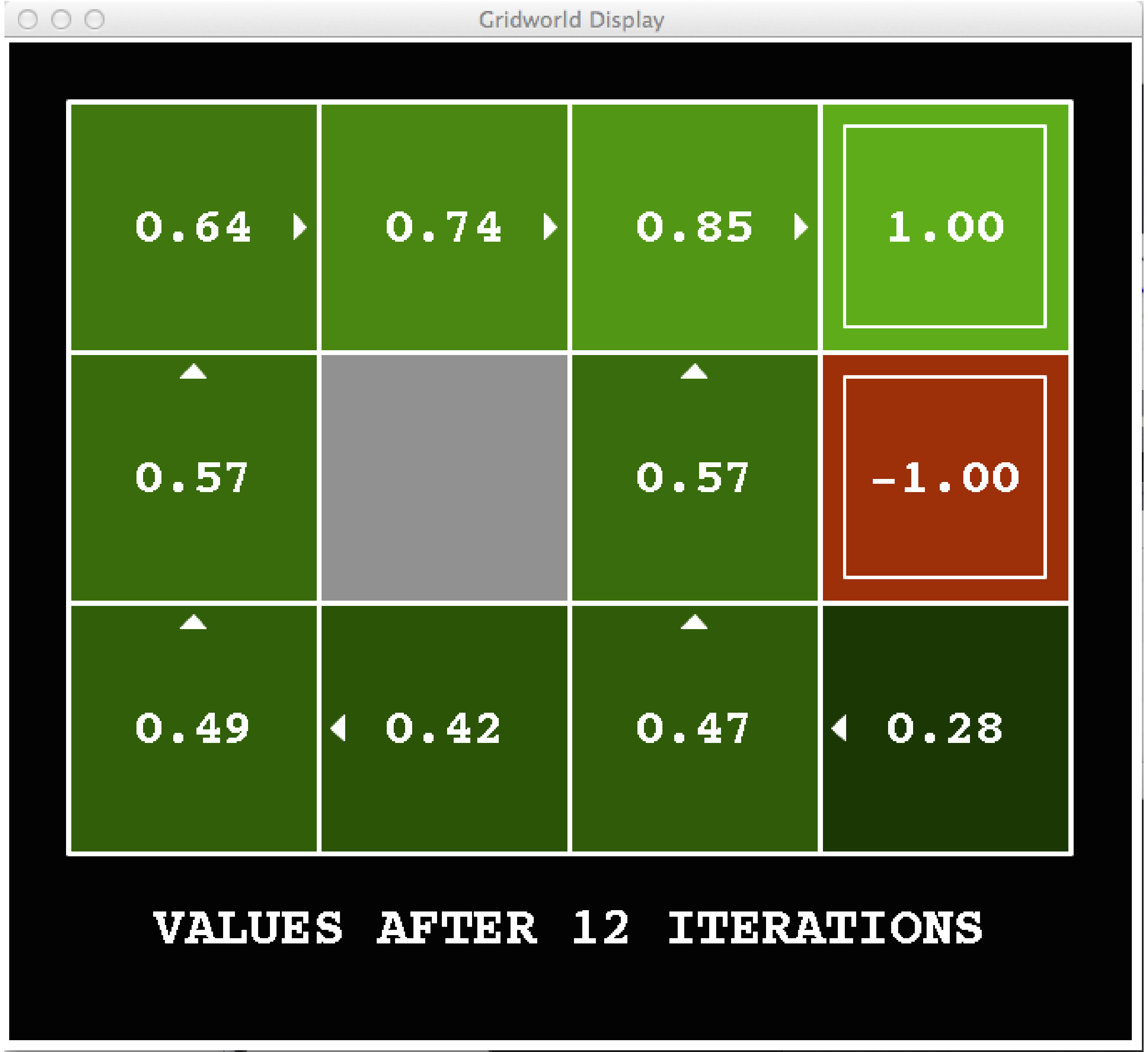


- Problem 1: It's slow – O(S^2A) per iteration

- Problem 2: The "max" at each state rarely changes

- Problem 3: The policy often converges long before the values

k=12

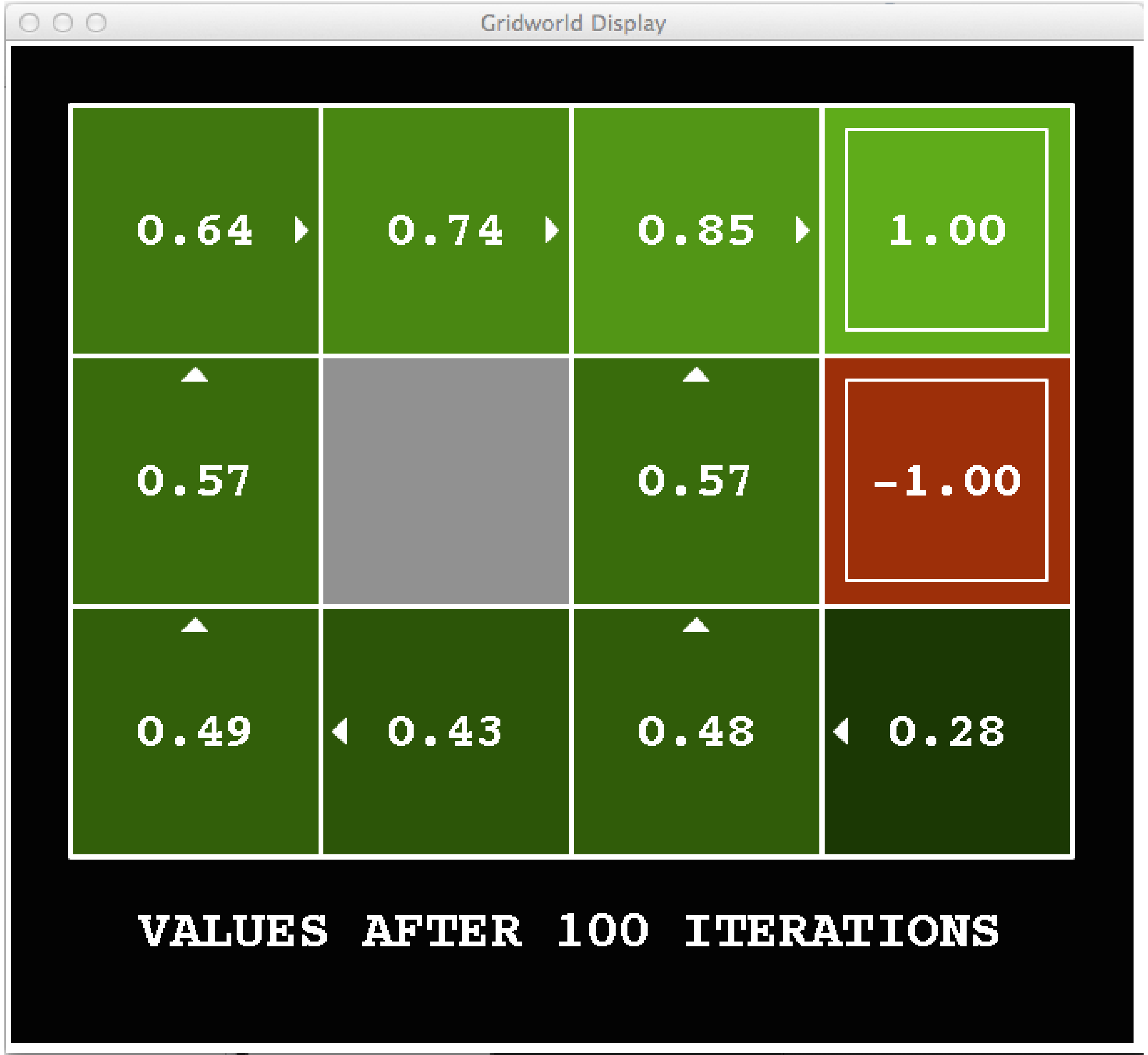


0.64 ▸	0.74 ▸	0.85 ▸	1.00
▲ 0.57		▲ 0.57	-1.00
▲ 0.49	◂ 0.42	▲ 0.47	◂ 0.28

VALUES AFTER 12 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

k=100



0.64 ▶	0.74 ▶	0.85 ▶	1.00
▲ 0.57		▲ 0.57	-1.00
▲ 0.49	◀ 0.43	▲ 0.48	◀ 0.28

VALUES AFTER 100 ITERATIONS

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

- Evaluation: For fixed current policy π, find values with policy evaluation:
  - Iterate until values converge:

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

- Improvement: For fixed values, get a better policy using policy extraction
  - One-step look-ahead:

$$\pi_{i+1}(s) = \arg\max_a \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V^{\pi_i}(s')\right]$$

Comparison

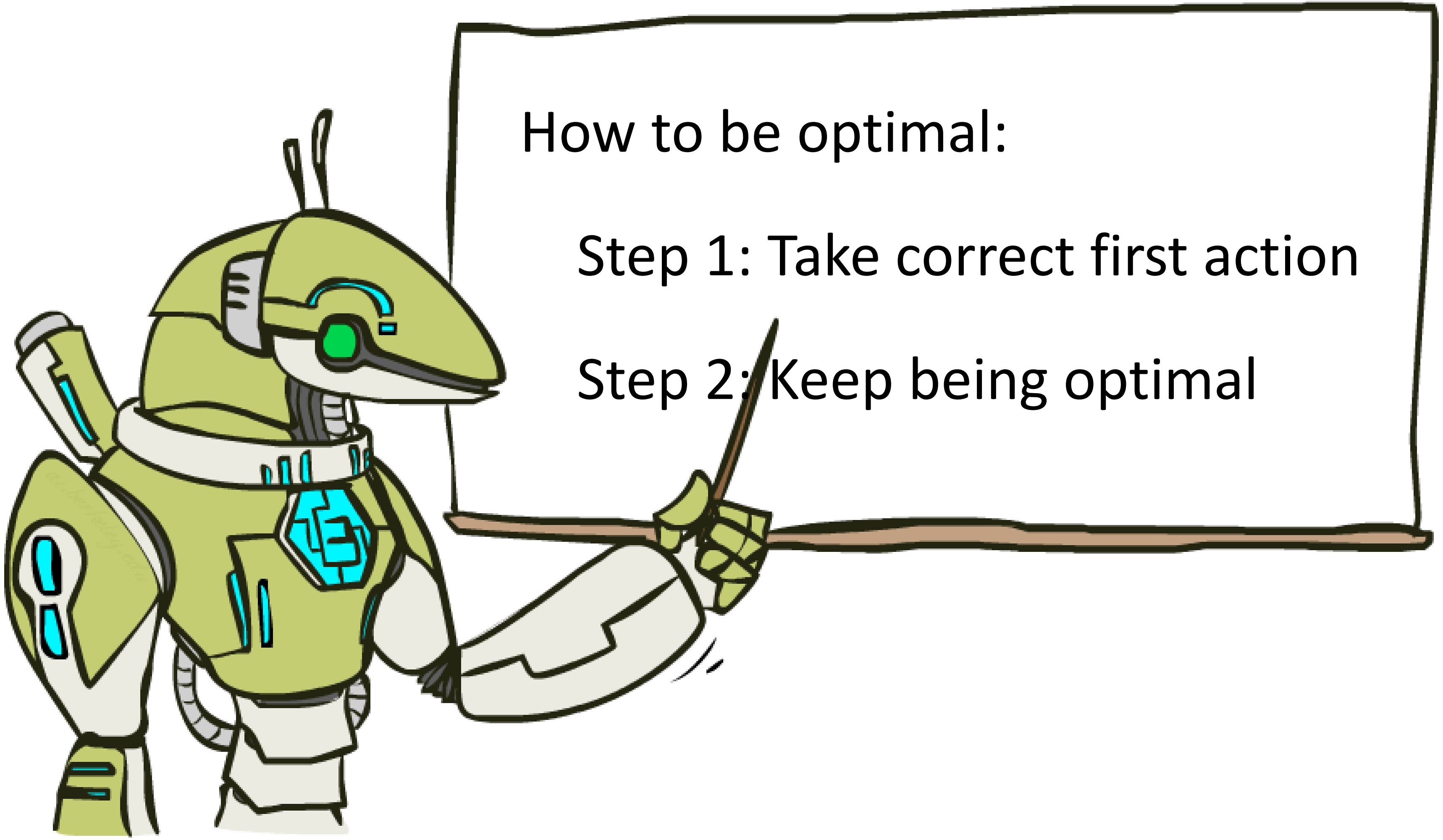
- Both value iteration and policy iteration compute the same thing (all optimal values)
- In value iteration:
  - Every iteration updates both the values and (implicitly) the policy
  - We don’t track the policy, but taking the max over actions implicitly recomputes it
- In policy iteration:
  - We do several passes that update utilities with fixed policy (each pass is fast because we consider only one action, not all of them)
  - After the policy is evaluated, a new policy is chosen (slow like a value iteration pass)
  - The new policy will be better (or we’re done)
- Both are dynamic programs for solving MDPs

Summary: MDP Algorithms

- So you want to….
  - Compute optimal values: use value iteration or policy iteration
  - Compute values for a particular policy: use policy evaluation
  - Turn your values into a policy: use policy extraction (one-step lookahead)

- These all look the same!
  - They basically are – they are all variations of Bellman updates
  - They all use one-step lookahead expectimax fragments
  - They differ only in whether we plug in a fixed policy or max over actions

The Bellman Equations



How to be optimal:

Step 1: Take correct first action

Step 2: Keep being optimal

Next Topic: Reinforcement Learning!