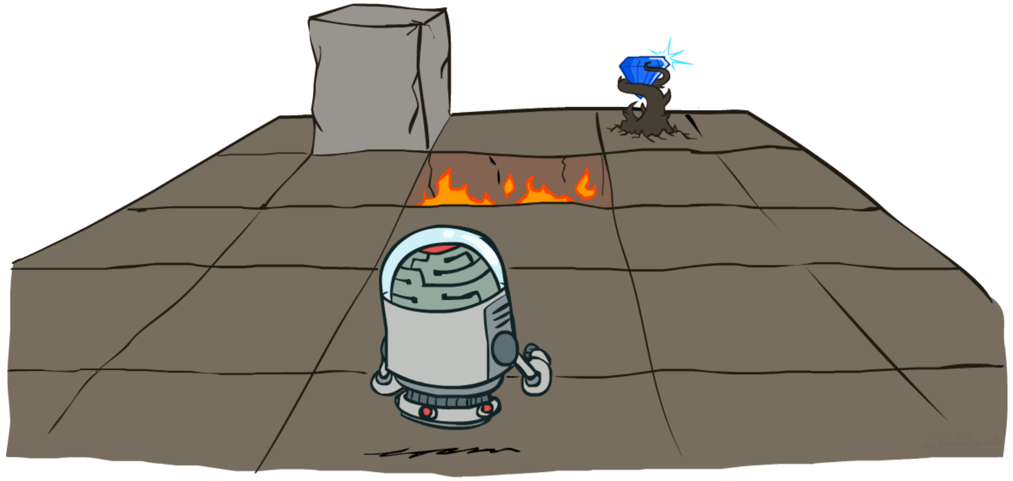
# Artificial Intelligence - INFOF311

## Markov decision processes

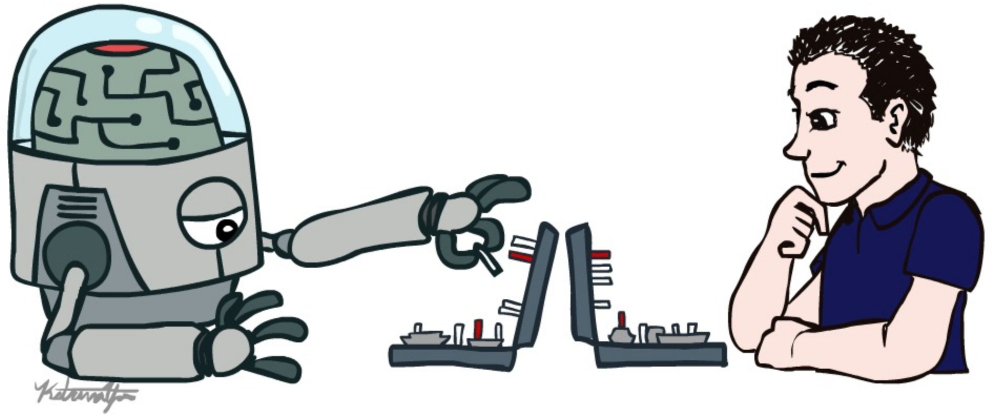


Instructor : Tom Lenaerts

## Acknowledgement

We thank Stuart Russell for his generosity in allowing us to use the slide set of the UC Berkeley Course CS188, Introduction to Artificial Intelligence. These slides were created by Dan Klein, Pieter Abbeel and Anca Dragan for CS188 Intro to AI at UC Berkeley.  All CS188 materials are available at http://ai.berkeley.edu.

CHAI Center for Human-Compatible Artificial Intelligence



The slides for INFOF311 are slightly modified versions of the slides of the spring and summer CS188 sessions in 2021 and 2022

## Aims of the course

**4 main themes :**

Part 1 : Search and planning (uninformed and informed search, local search, game and adversarial search, …)

Part 2: Probabilistic reasoning (Bayesian network, hidden Markov models, filtering, decision networks…)

**Part 3: Decision making with uncertainty (MDP, reinforcement learning, …)**

Part 4: Machine learning (naïve bayes, perceptrons, regression, neural networks, …)

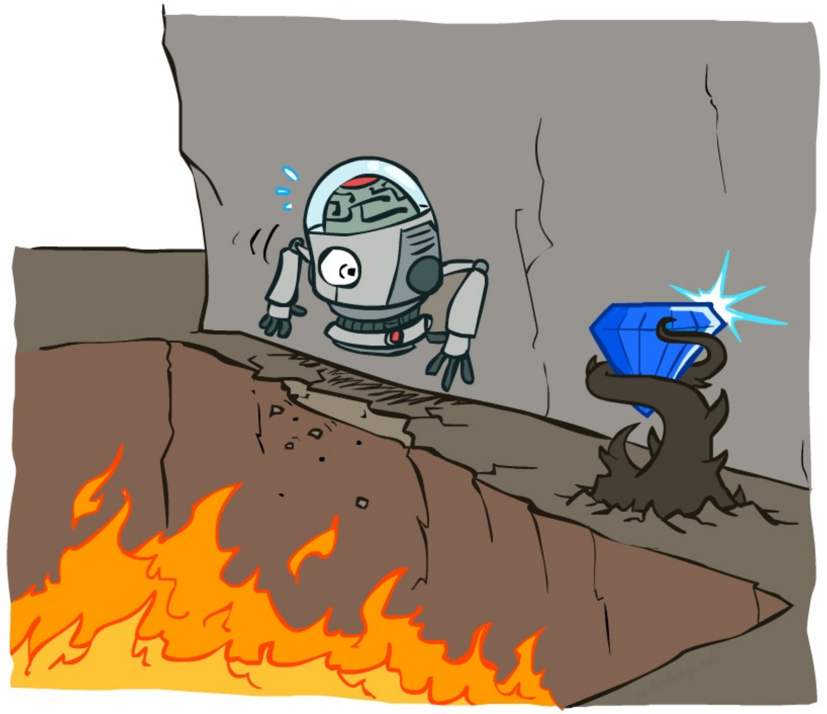
# Sequential decisions under uncertainty

So far, decision problem is one-shot --- concerning only one action

Sequential decision problem: agent's utility depends on a sequence of actions

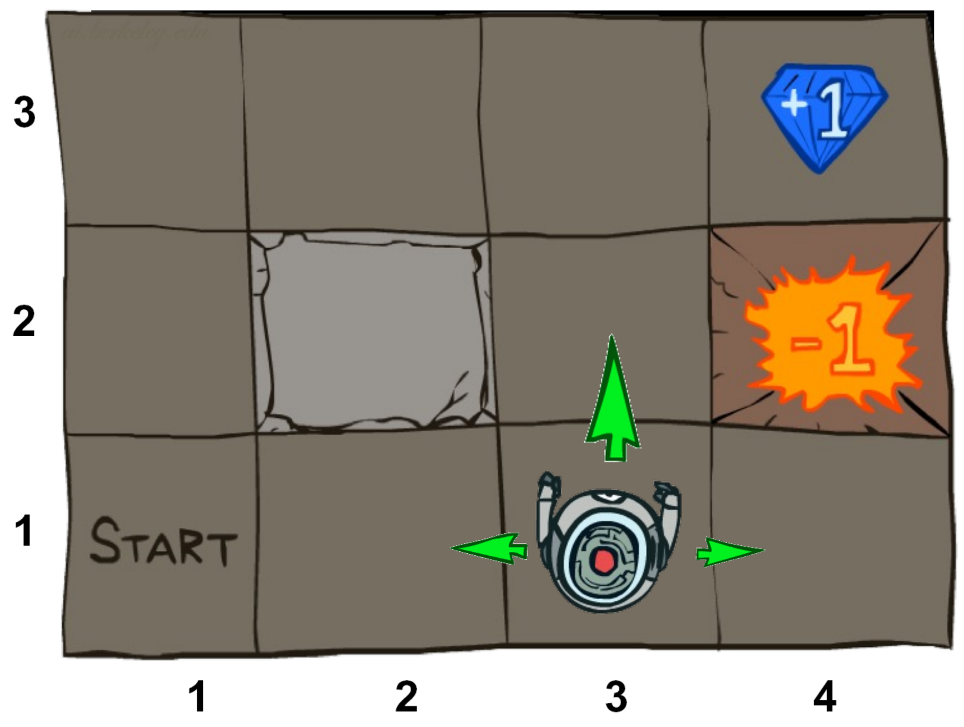
But the environment is stochastic !

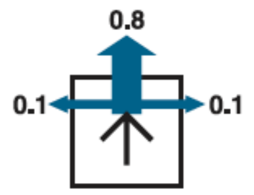
**Non-deterministic search**



# Example: Grid World

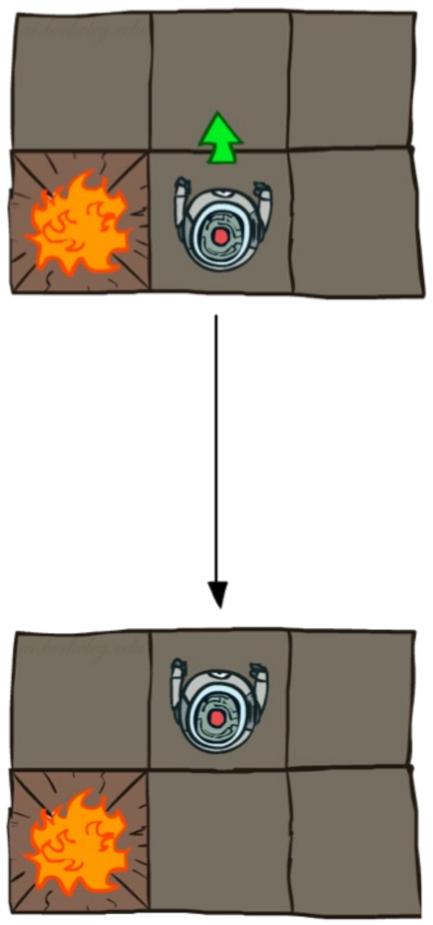
- A maze-like problem
  - The agent lives in a grid
  - Walls block the agent's path
- Noisy movement: actions do not always go as planned
  - 80% of the time, the action North takes the agent North (if there is no wall there)
  - 10% of the time, North takes the agent West; 10% East
  - If there is 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 r each step (can be negative)
  - Big rewards come at the end (good or bad)
- Goal: maximize sum of rewards



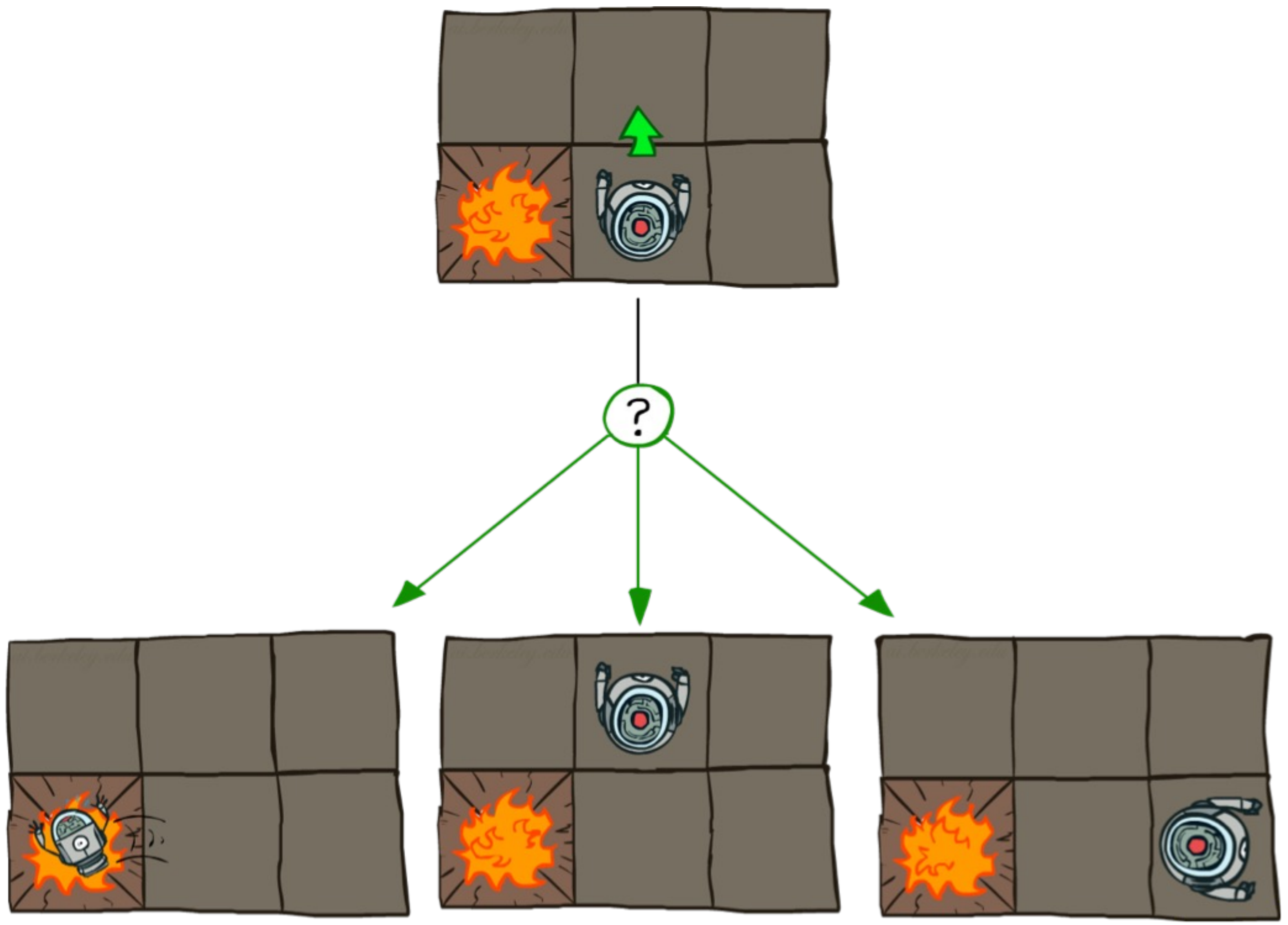


# Grid World Actions

Deterministic Grid World

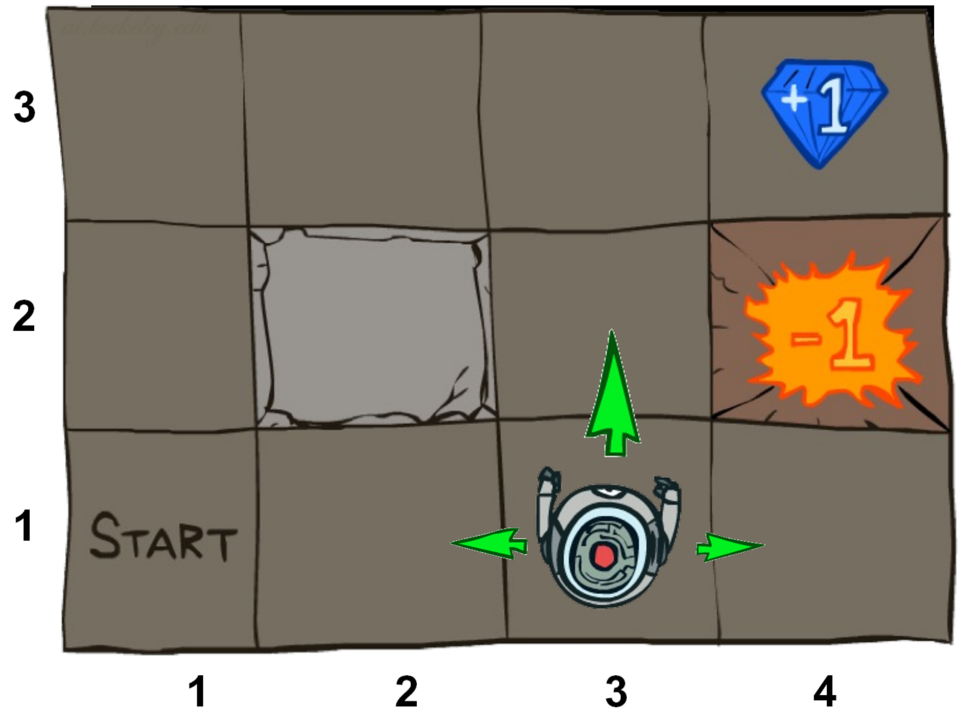


Stochastic Grid World



# Markov Decision Process (MDP)

- An MDP is defined by:
  - A set of states $s \in S$
  - A set of actions $a \in A$
  - A transition model $T(s, a, s')$
    - Probability that $a$ from $s$ leads to $s'$, i.e., $P(s' | s, a)$
  - A reward function $R(s, a, s')$ for each transition
  - A start state
  - Possibly a terminal state (or absorbing state)
  - Utility function which is additive (discounted) rewards



- MDPs are fully observable but probabilistic search problems

[Demo – gridworld manual intro (L8D1)]

# Markov Decision Process (MDP)

- “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

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

$$=$$

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

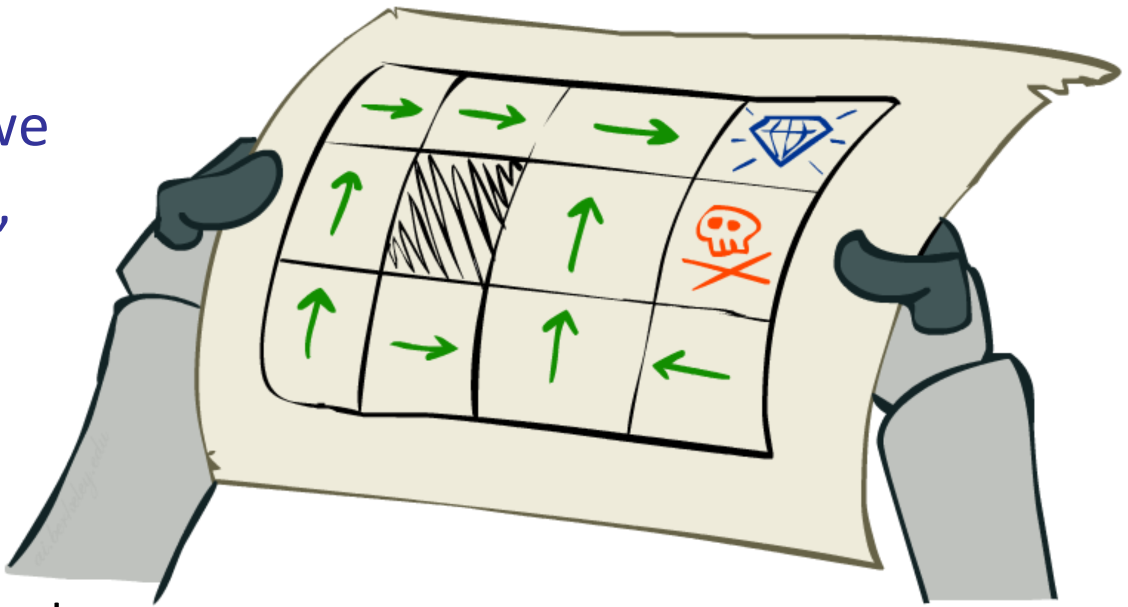
- This is just like search, where the successor function could only depend on the current state (not the history)

Andrey Markov
(1856-1922)

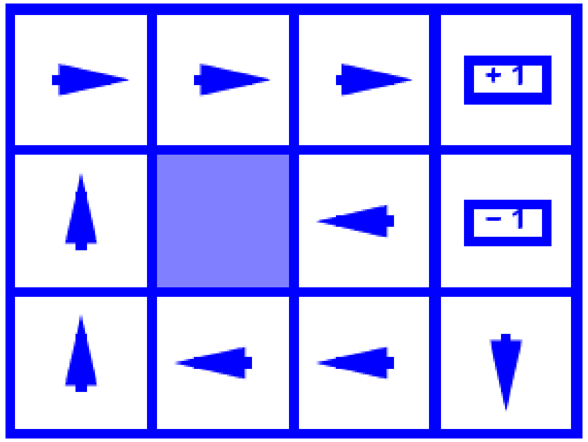
# Policies

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

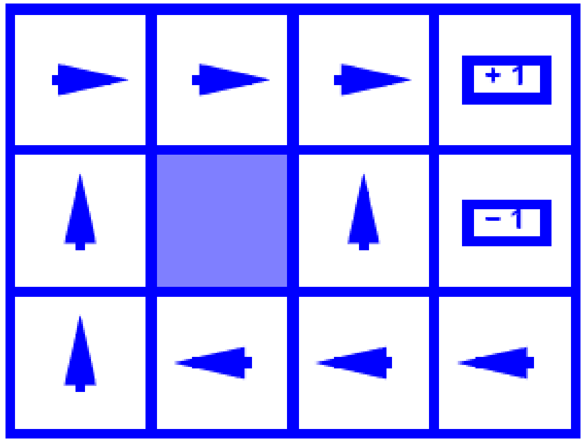
- For MDPs, we want an optimal ***policy*** $\pi^*: S \rightarrow A$
  - A policy $\pi$ gives an action for each state
  - An optimal policy maximizes expected utility, if followed
  - An explicit policy defines a reflex agent



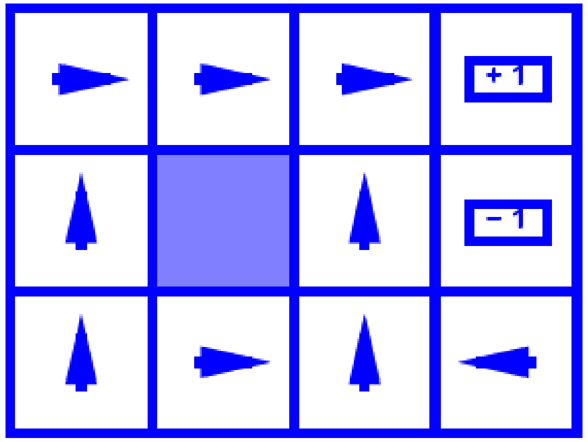
# Optimal Policies



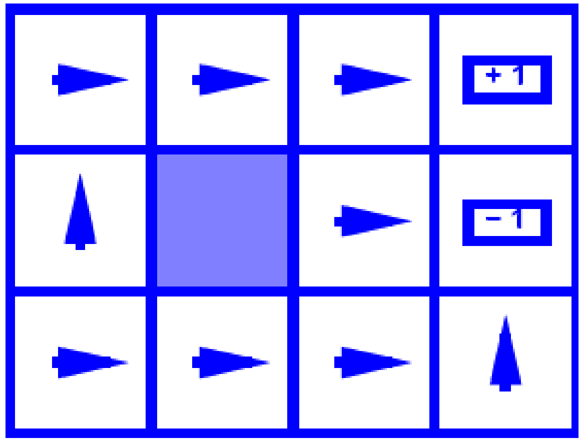
R(s) = -0.01



R(s) = -0.03

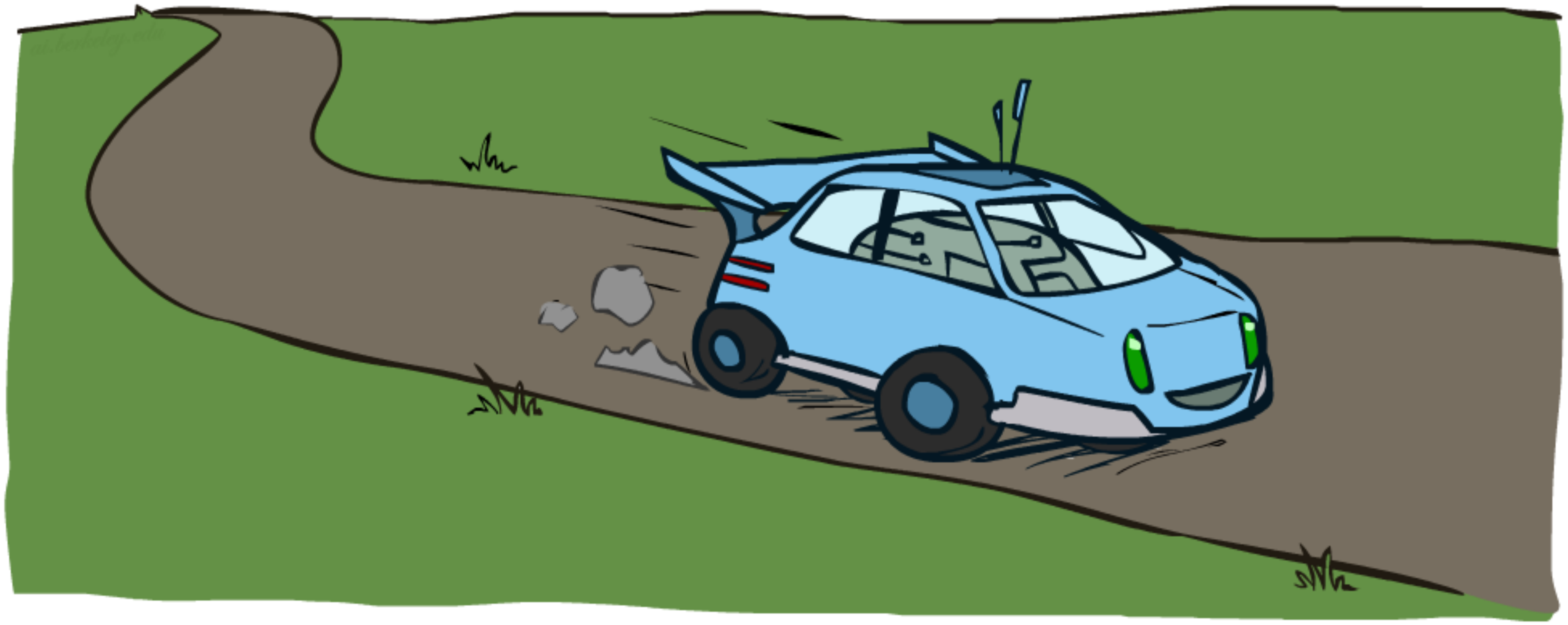


R(s) = -0.4



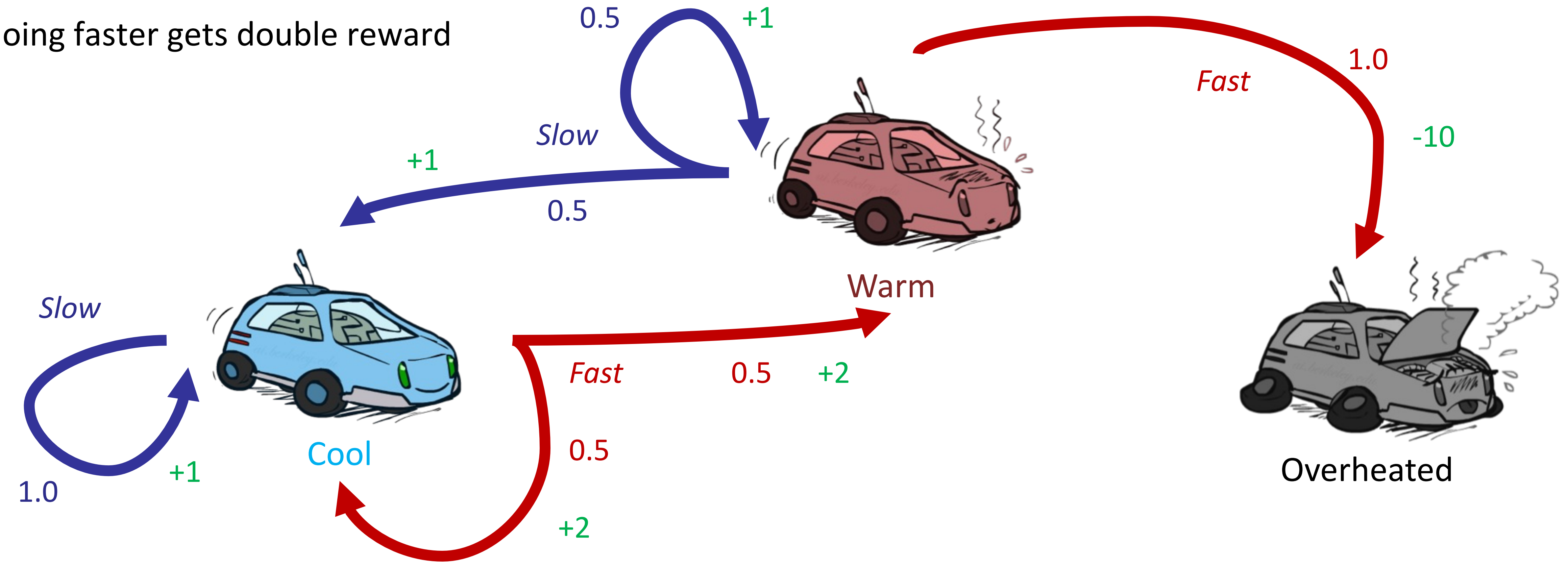
R(s) = -2.0

## Example: Racing

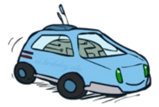
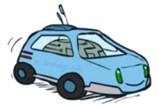
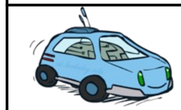
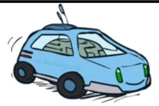
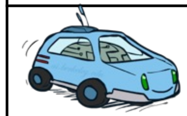
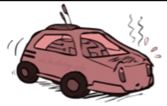
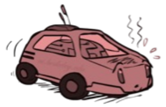
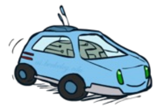
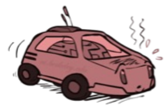
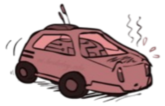
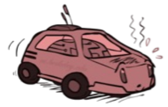
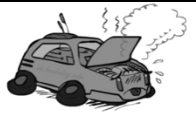
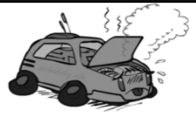
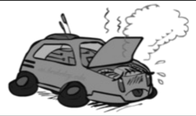


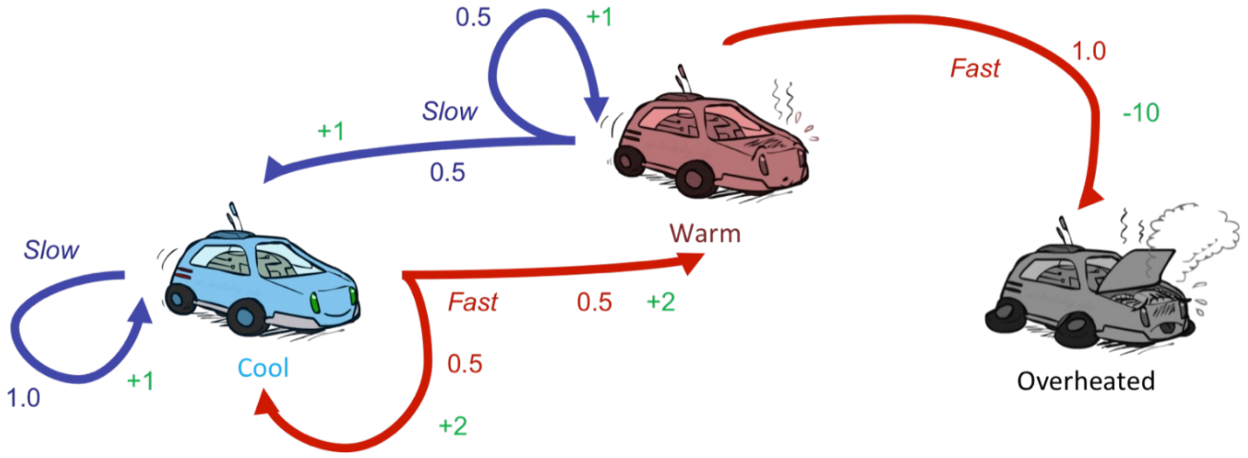
# Example: Racing

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

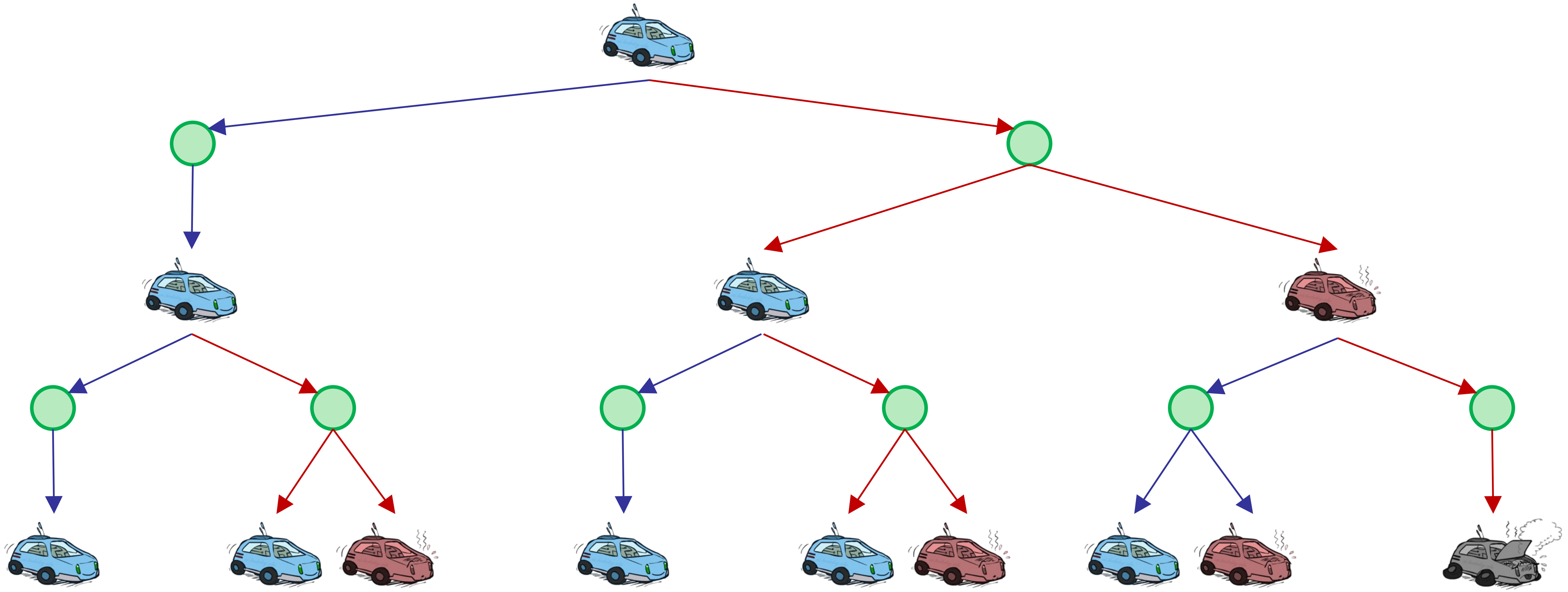


# Example: Racing

| s | a | s' | T(s,a,s’) | R(s,a,s’) |
|---|---|---|---|---|
|  | Slow |  | 1.0 | +1 |
|  | Fast |  | 0.5 | +2 |
|  | Fast |  | 0.5 | +2 |
|  | Slow |  | 0.5 | +1 |
|  | Slow |  | 0.5 | +1 |
|  | Fast |  | 1.0 | –10 |
|  | (end) |  | 1.0 | 0 |

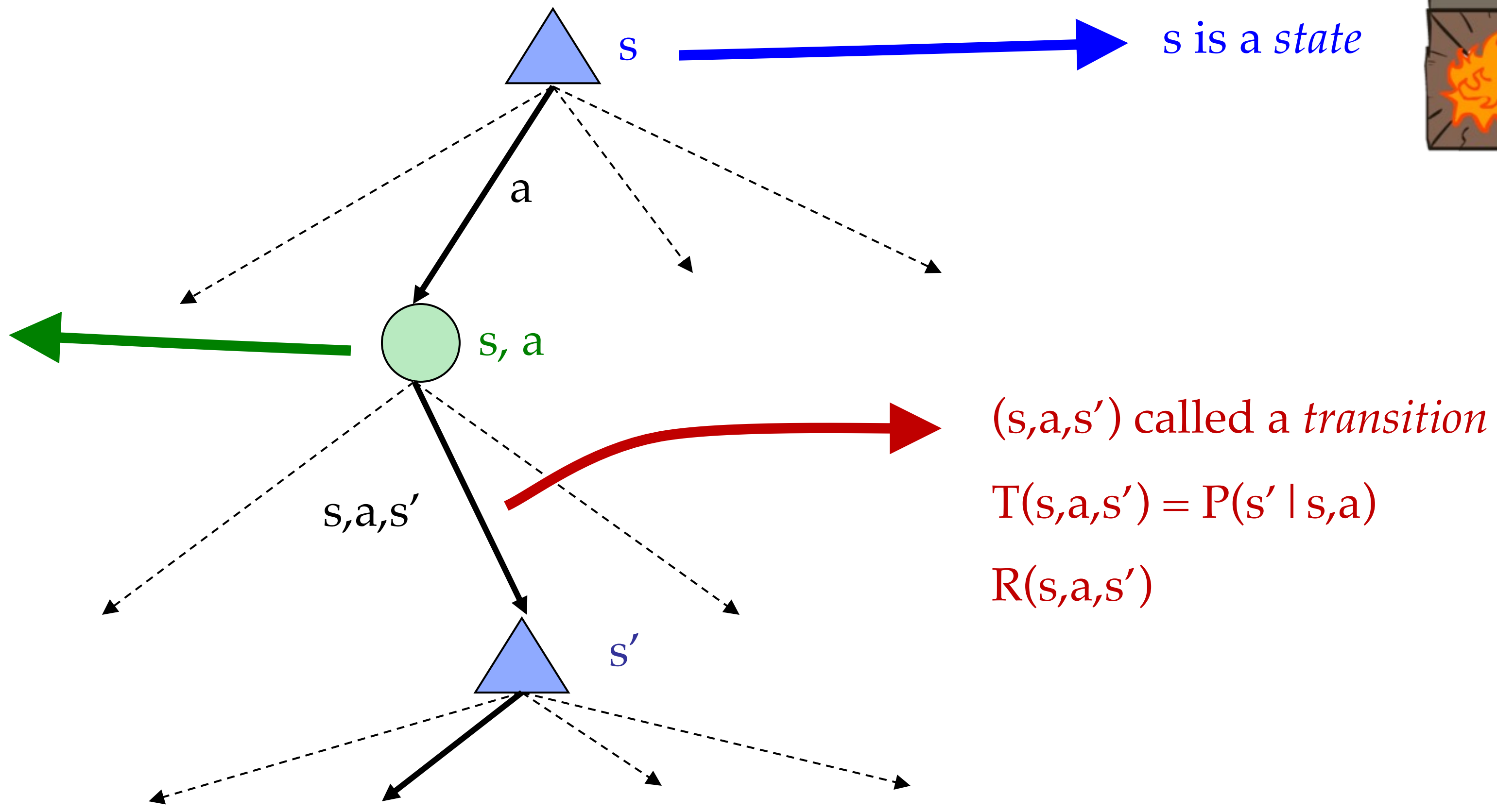


# Racing Search Tree



# MDP Search Trees

- Each MDP state projects an expectimax-like search tree

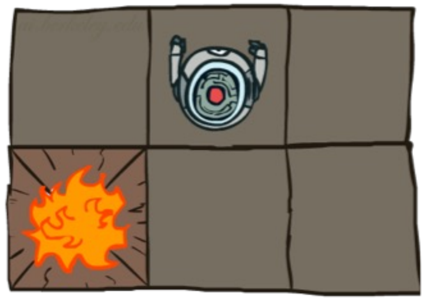


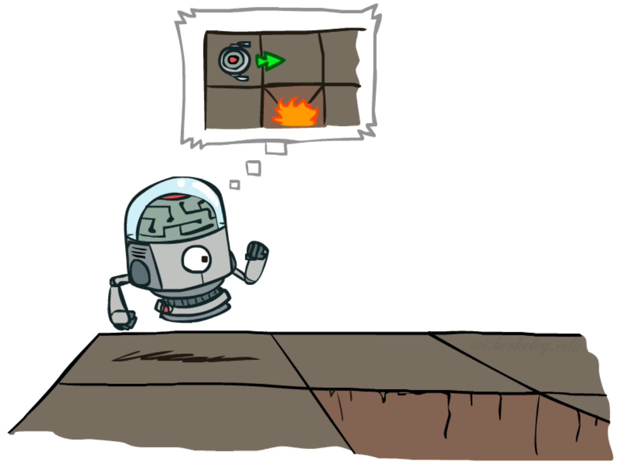
s is a *state*

(s,a,s′) called a *transition*

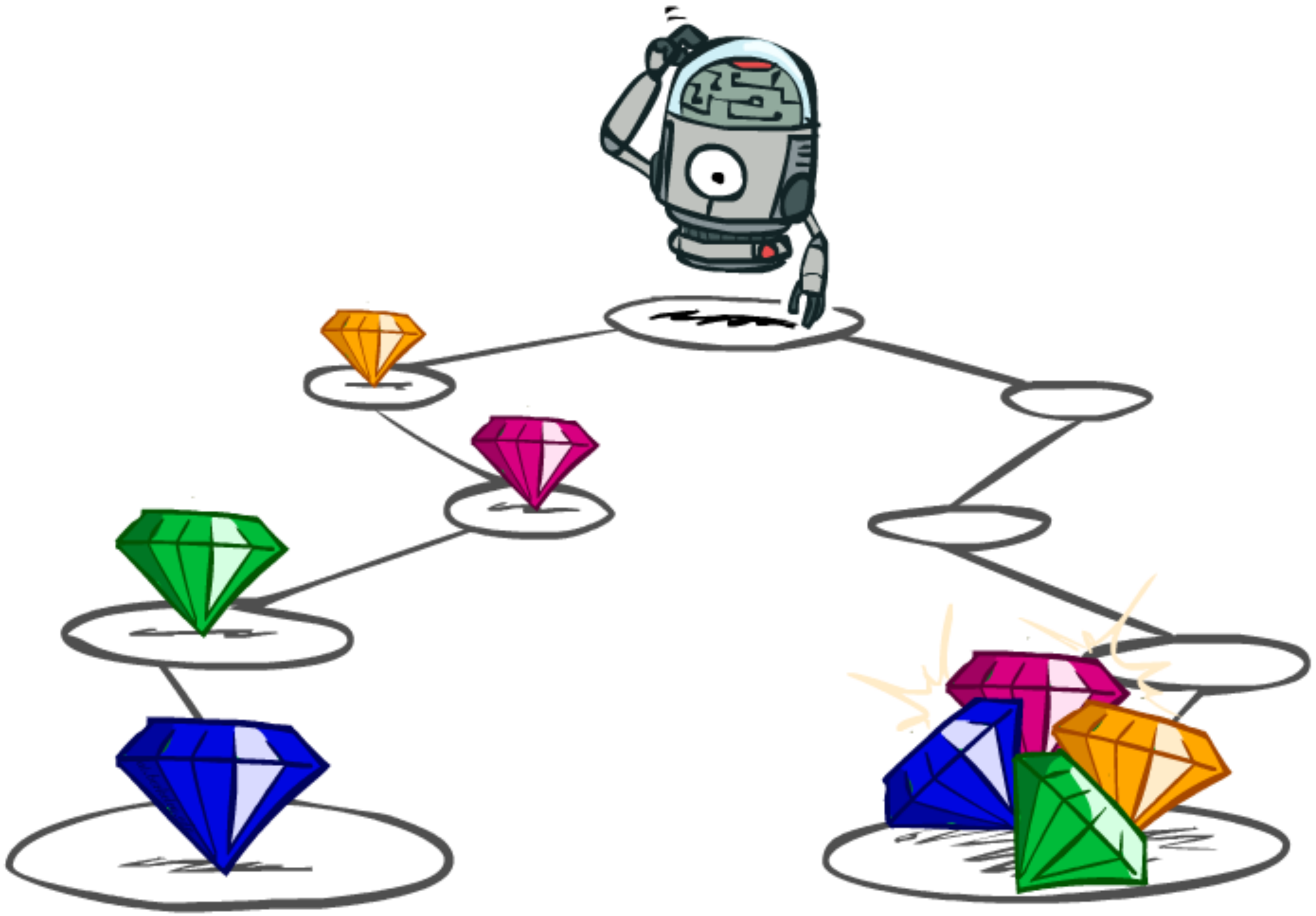
$T(s,a,s') = P(s' \mid s,a)$

$R(s,a,s')$



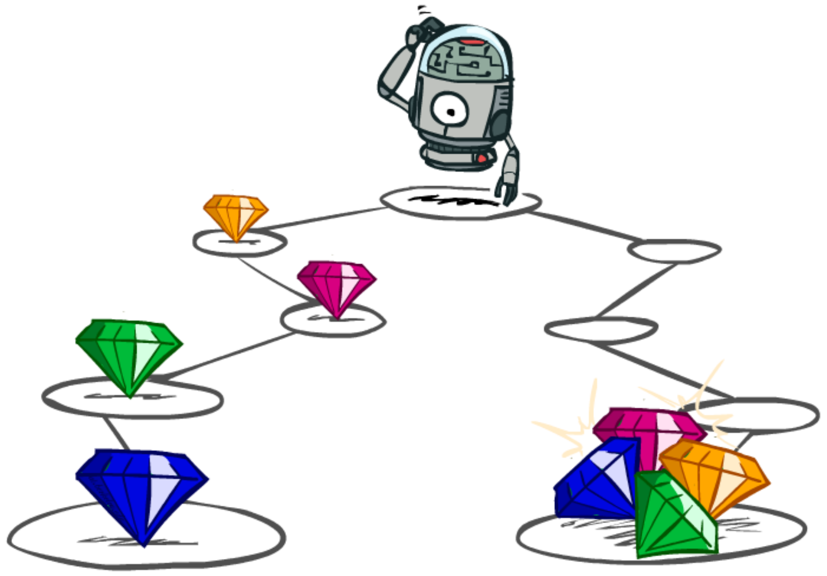


# Utilities of Sequences



# 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

$\gamma$

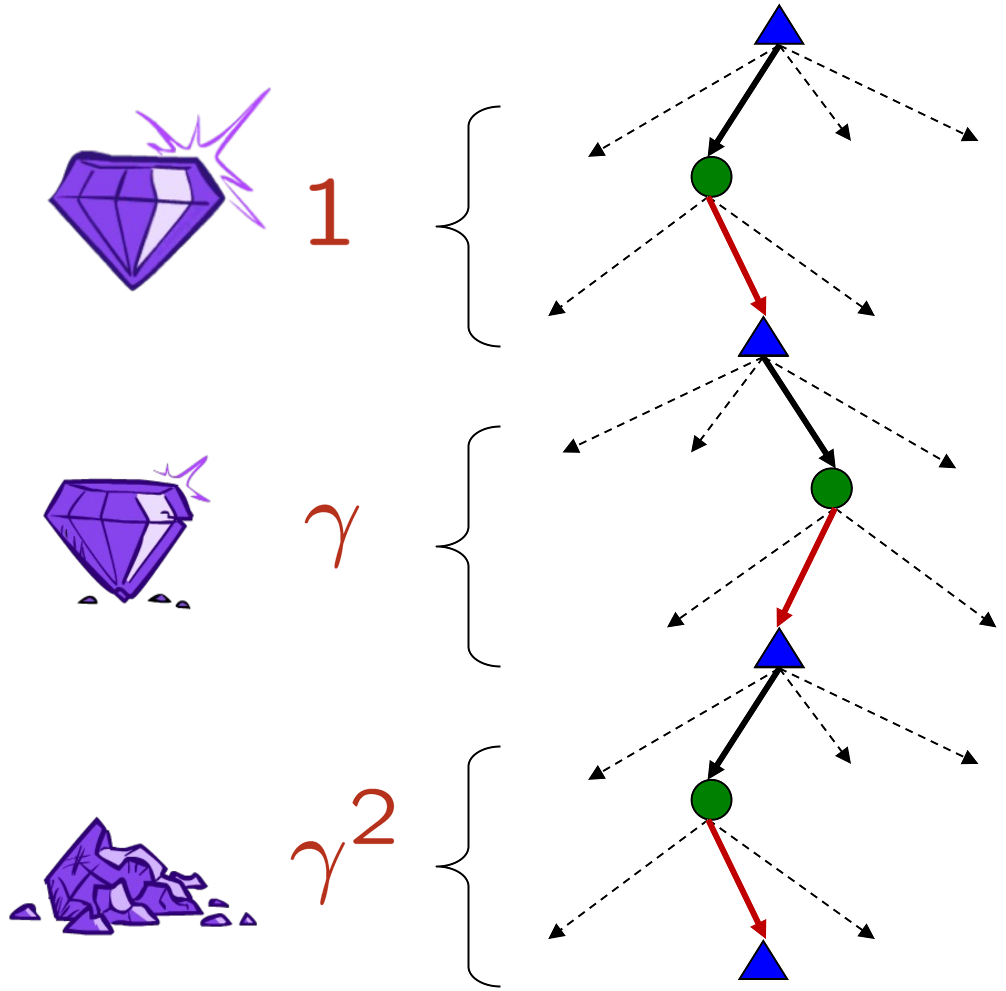
Worth Next Step

$\gamma^2$

Worth In Two Steps

# Discounting

- How to discount?
  - Each time we descend a level, we multiply in the discount once
- Why discount?
  - Reward now is better than later
  - Can also think of it as a 1-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])



# Quiz: Discounting

- Given:

|  |  |  |  |  |
|---|---|---|---|---|
| 10 |  |  |  | 1 |
| a | b | c | d | e |

  - 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?

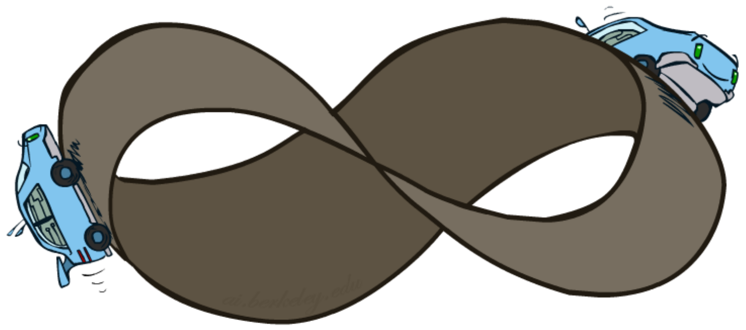
|  |  |  |  |  |
|---|---|---|---|---|
| 10 | <- | <- | -> | 1 |

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

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

# Infinite Utilities?!

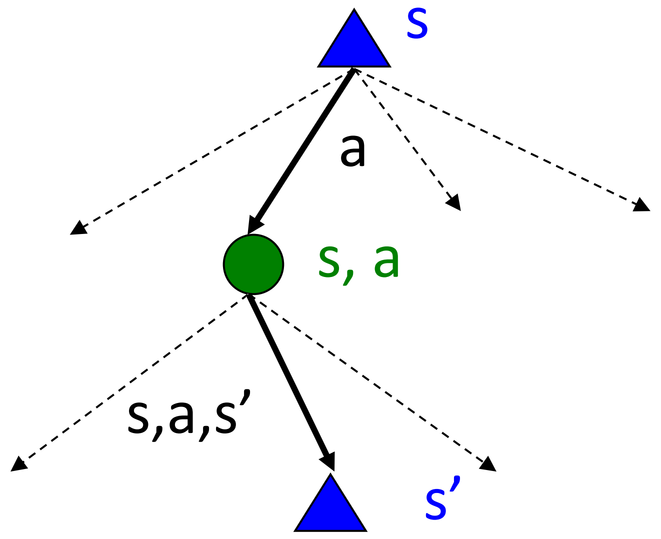
- Problem: What if the game lasts forever? Do we get infinite rewards?
- Solutions:
  - Finite horizon: (similar to depth-limited search)
    - Terminate episodes after a fixed T steps (e.g. life)
    - Gives nonstationary policies ($\pi$ depends on time left)



- Discounting with $\gamma$ solves the problem of infinite reward streams!
  - Geometric series: $1 + \gamma + \gamma^2 + \ldots = 1/(1 - \gamma)$
  - Assume rewards bounded by $\pm R_{\max}$
  - Then $r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots$ is bounded by $\pm R_{\max}/(1 - \gamma)$
- Absorbing state: guarantee that for every policy, a terminal state will eventually be reached (like “overheated” for racing)

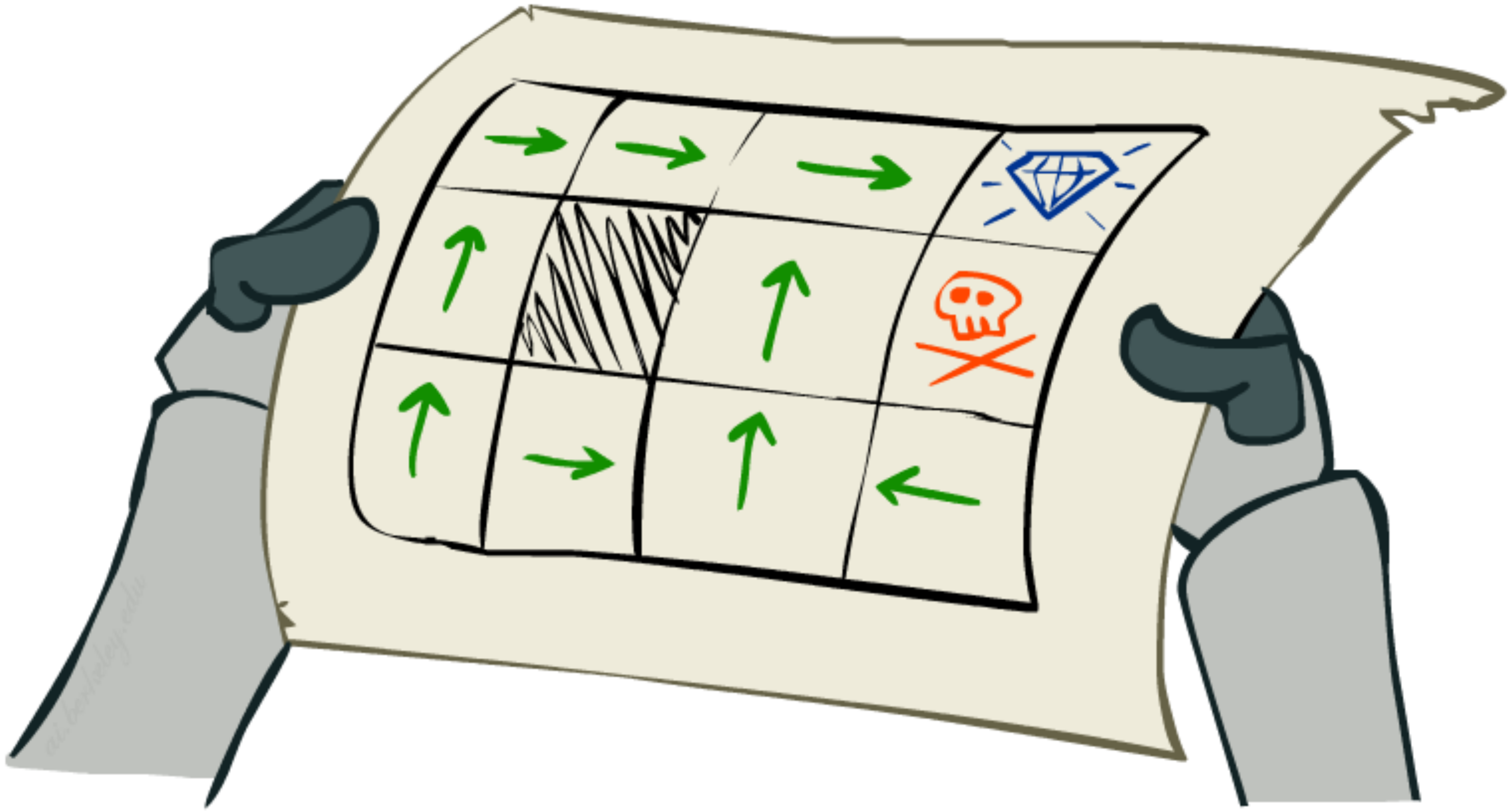
# Recap: Defining MDPs

- 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 $\gamma$)

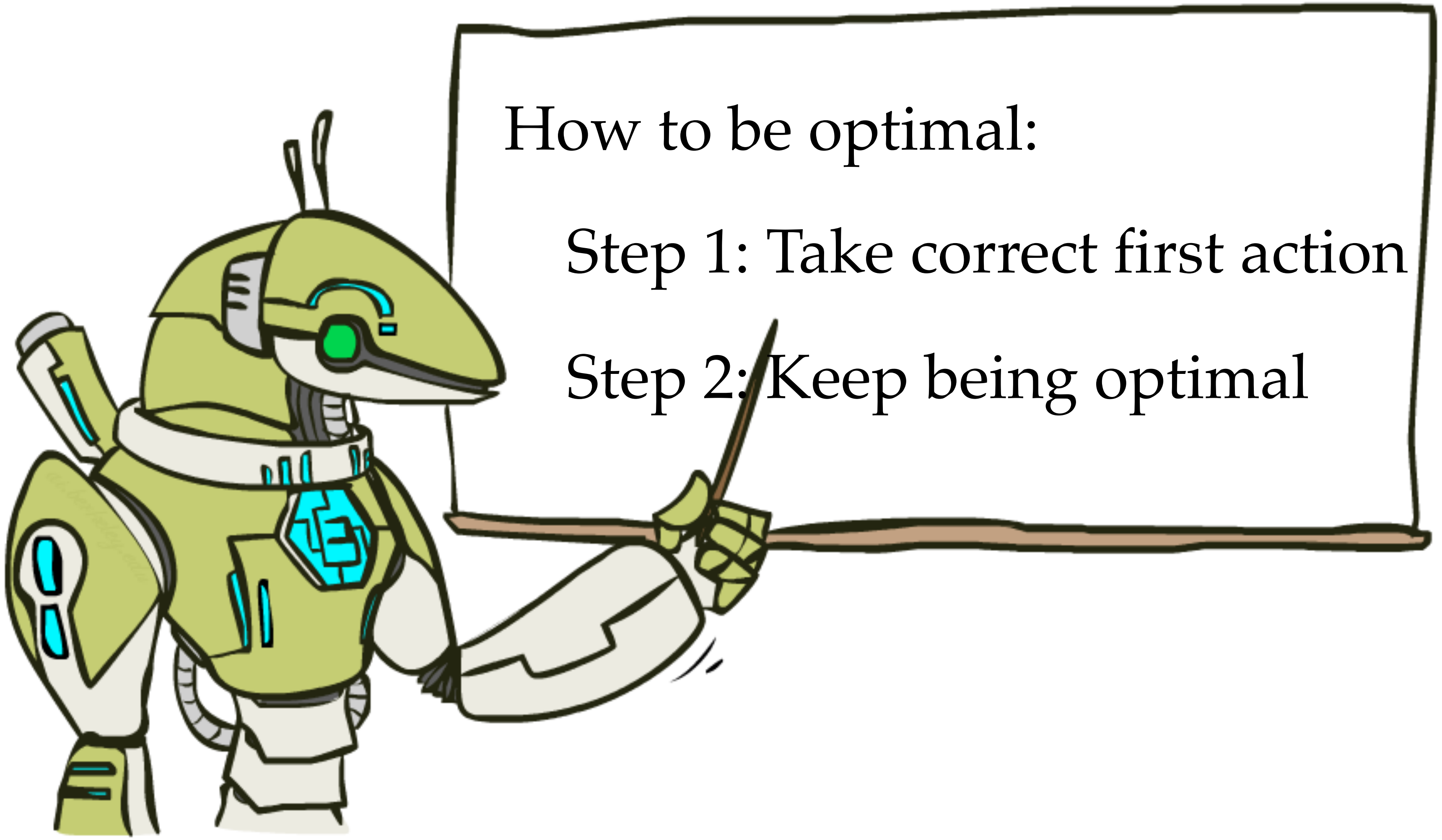


- MDP quantities so far:
  - Policy = Choice of action for each state
  - Utility = sum of (discounted) rewards

## Solving MDPs



## The Bellman Equations



# Optimal Quantities

- The value (utility) of a state s:

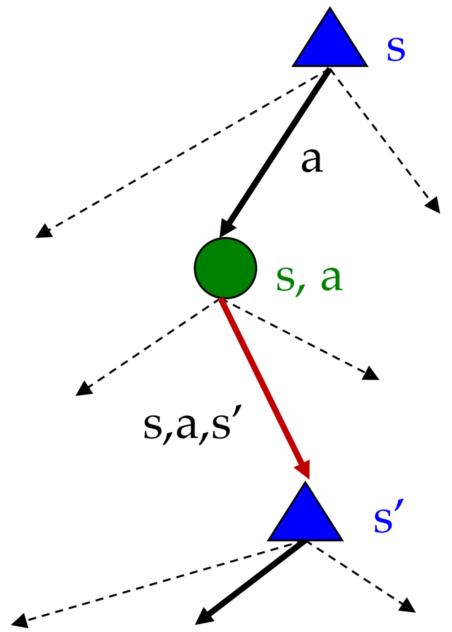
  $V^*(s)$ = expected utility starting in s and acting optimally

- 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

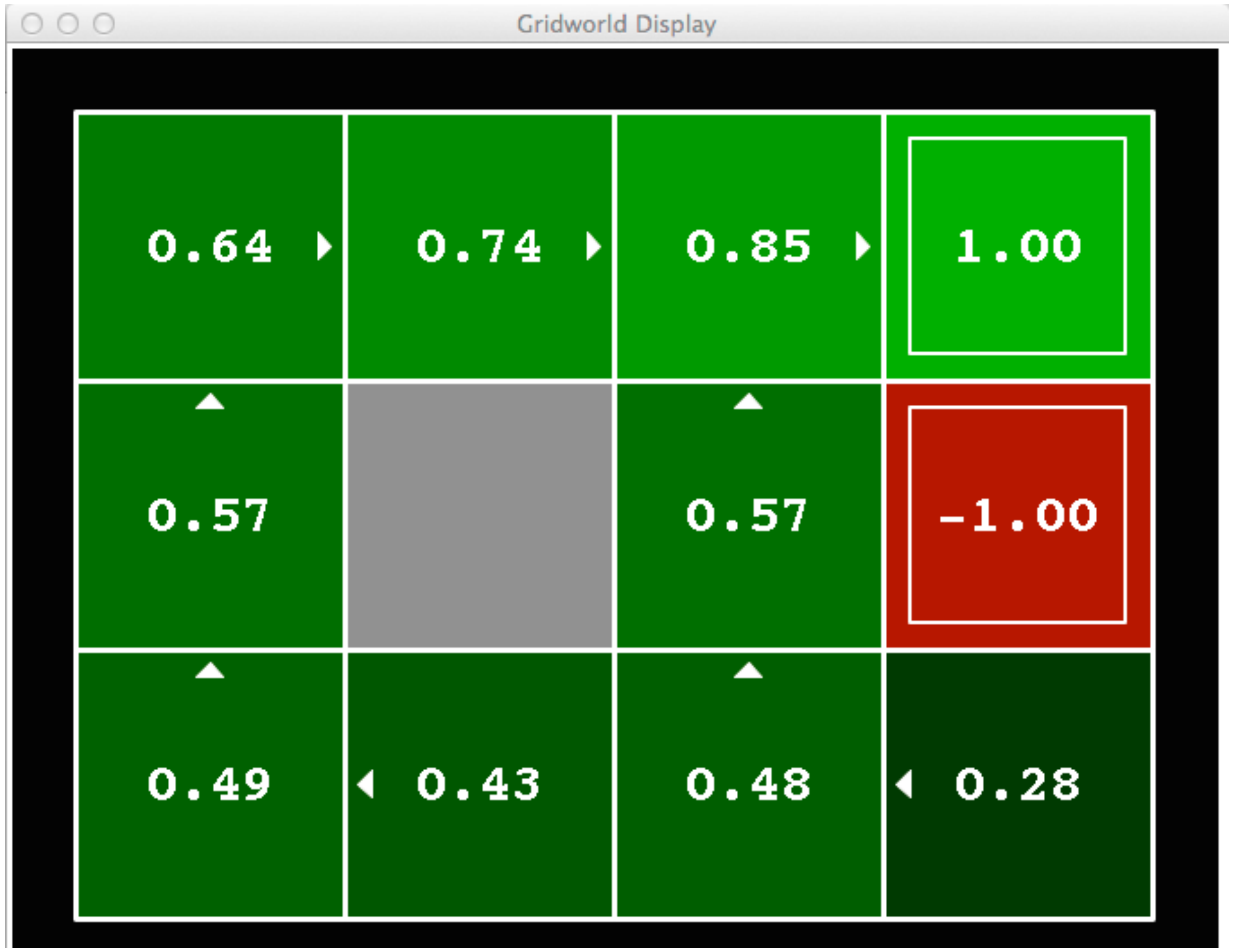


s is a *state*

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

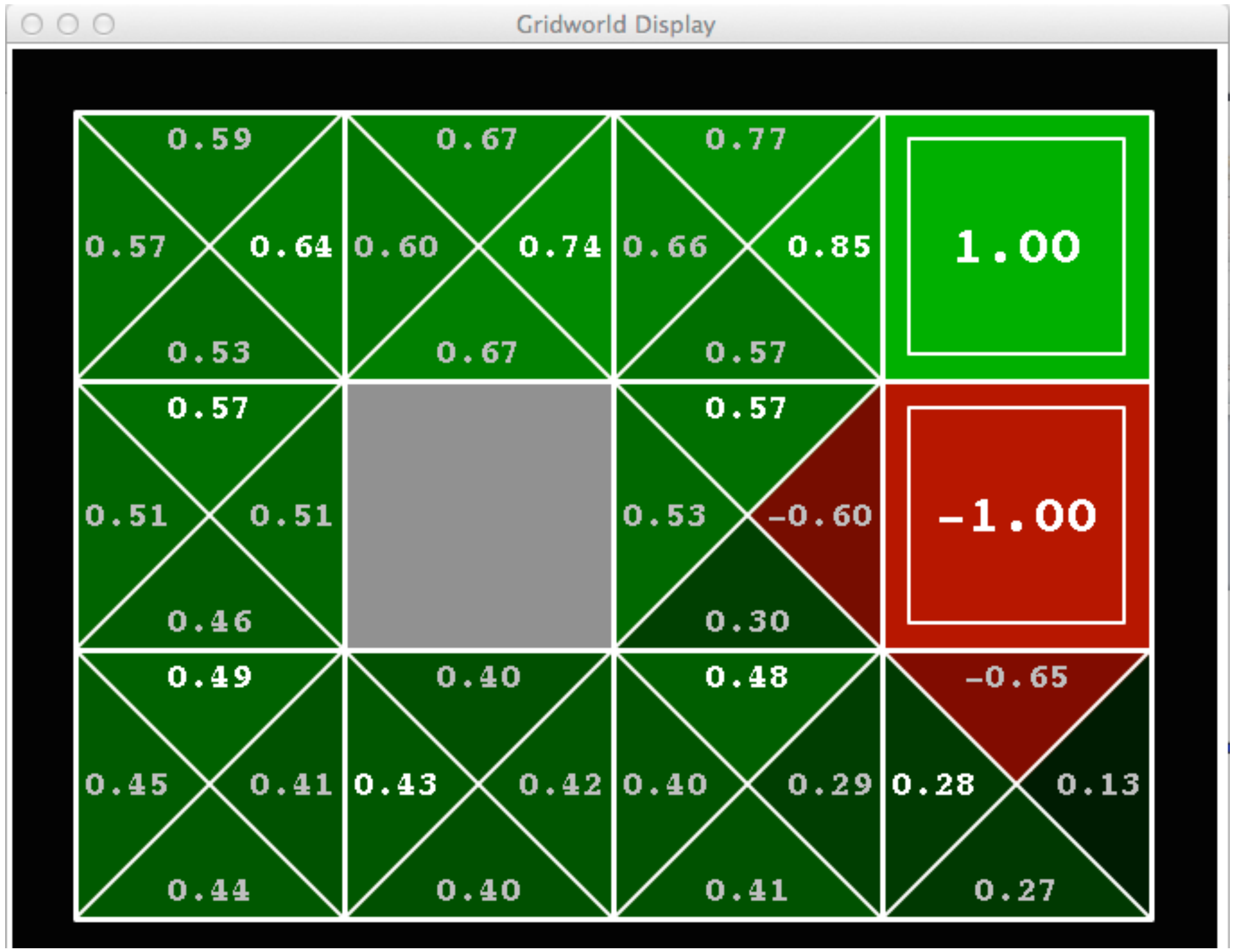
(s,a,s') is a *transition*

# Gridworld V* Values



Noise = 0.2
Discount = 0.9
Living reward = 0

# Gridworld Q* Values



Noise = 0.2
Discount = 0.9
Living reward = 0

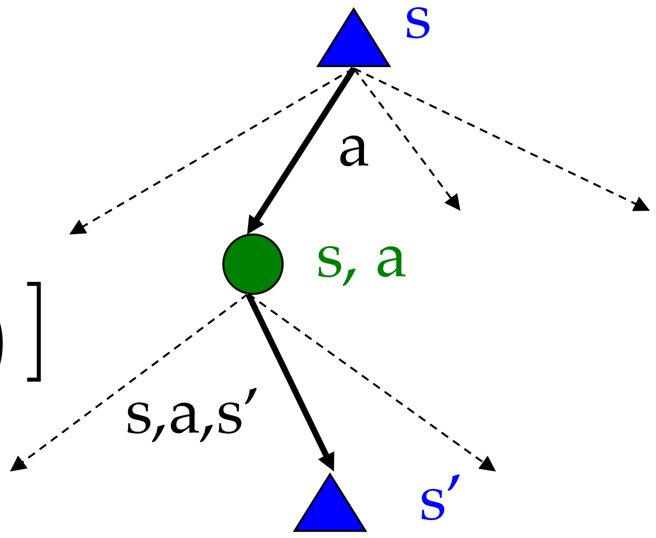
## Values of States

- Recursive definition of value:

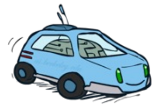
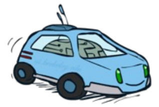
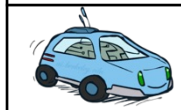
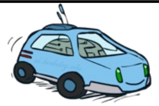
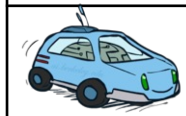
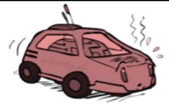
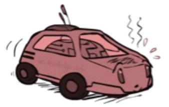
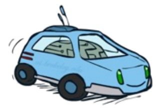
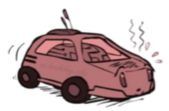
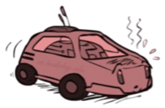
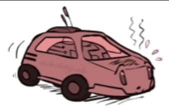
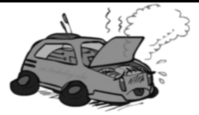
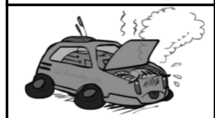
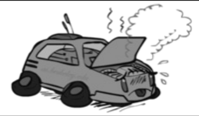
$$V^*(s) = \max_a Q^*(s,a)$$

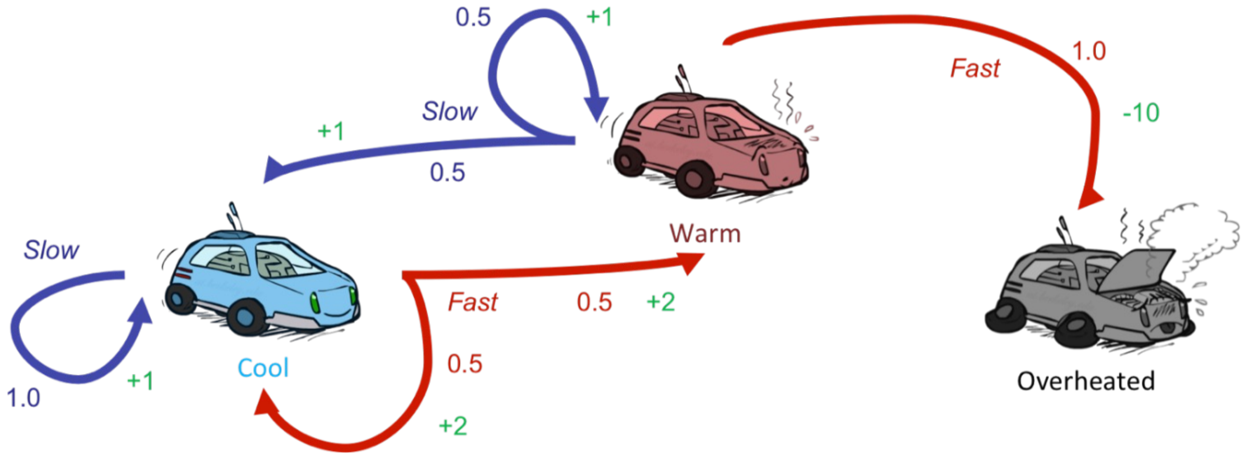
$$Q^*(s,a) = \sum_{s'} T(s,a,s')\left[ R(s,a,s') + \gamma V^*(s') \right]$$

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

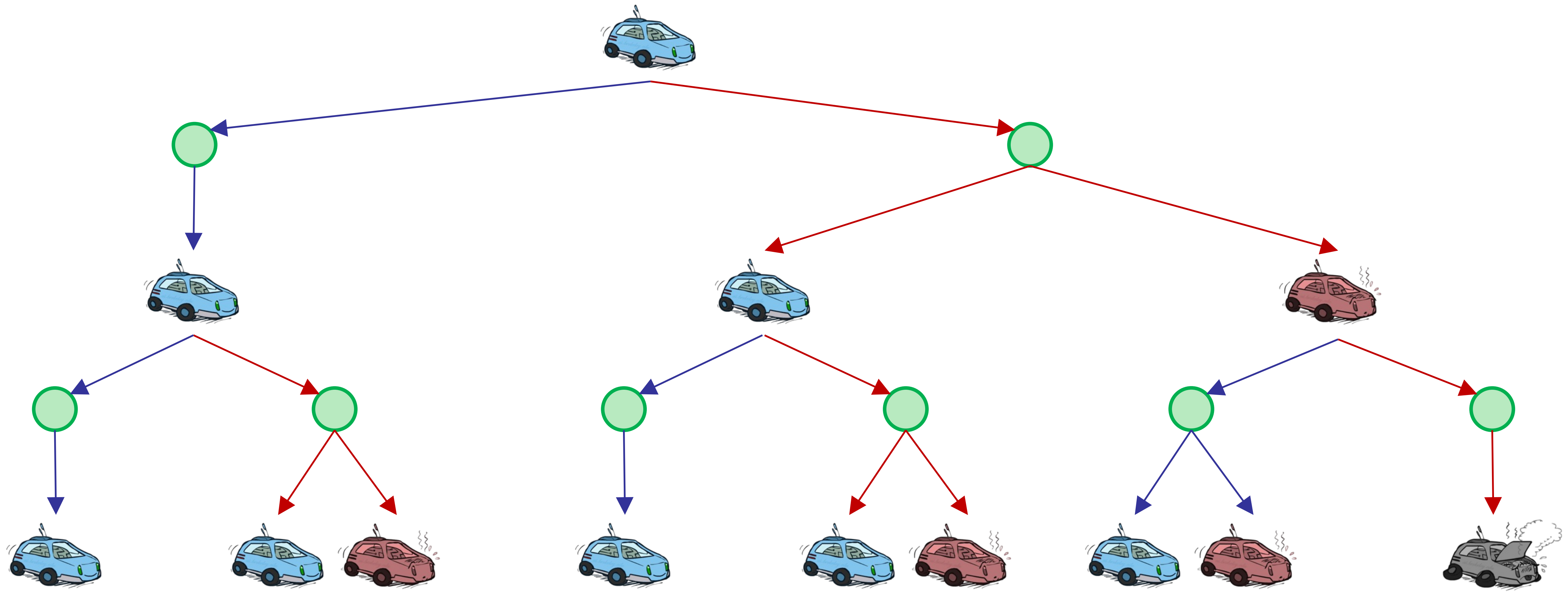


# Example: Racing

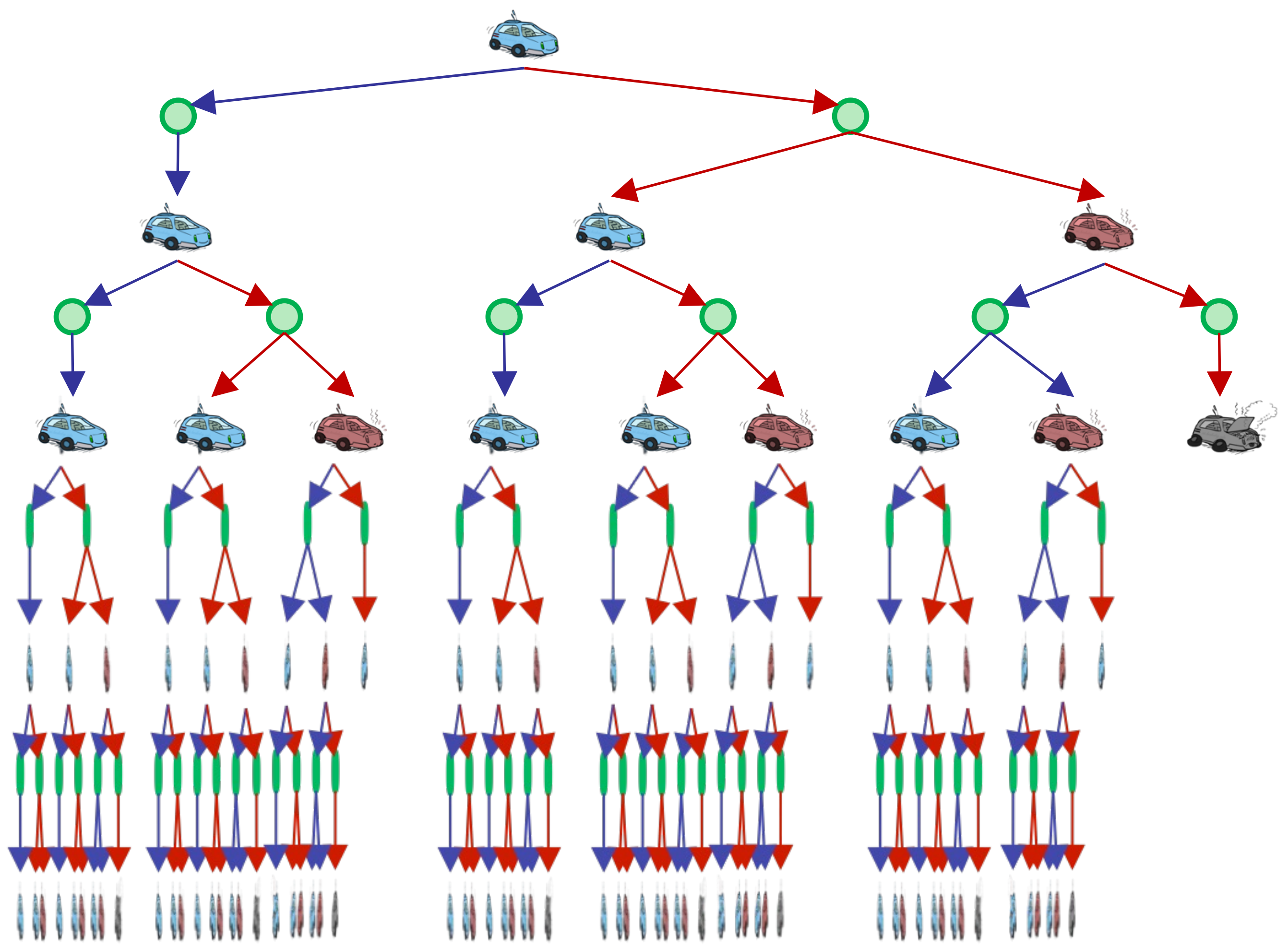
| s | a | s' | T(s,a,s') | R(s,a,s') |
|---|---|---|---|---|
|  | Slow |  | 1.0 | +1 |
|  | Fast |  | 0.5 | +2 |
|  | Fast |  | 0.5 | +2 |
|  | Slow |  | 0.5 | +1 |
|  | Slow |  | 0.5 | +1 |
|  | Fast |  | 1.0 | –10 |
|  | (end) |  | 1.0 | 0 |



# Racing Search Tree

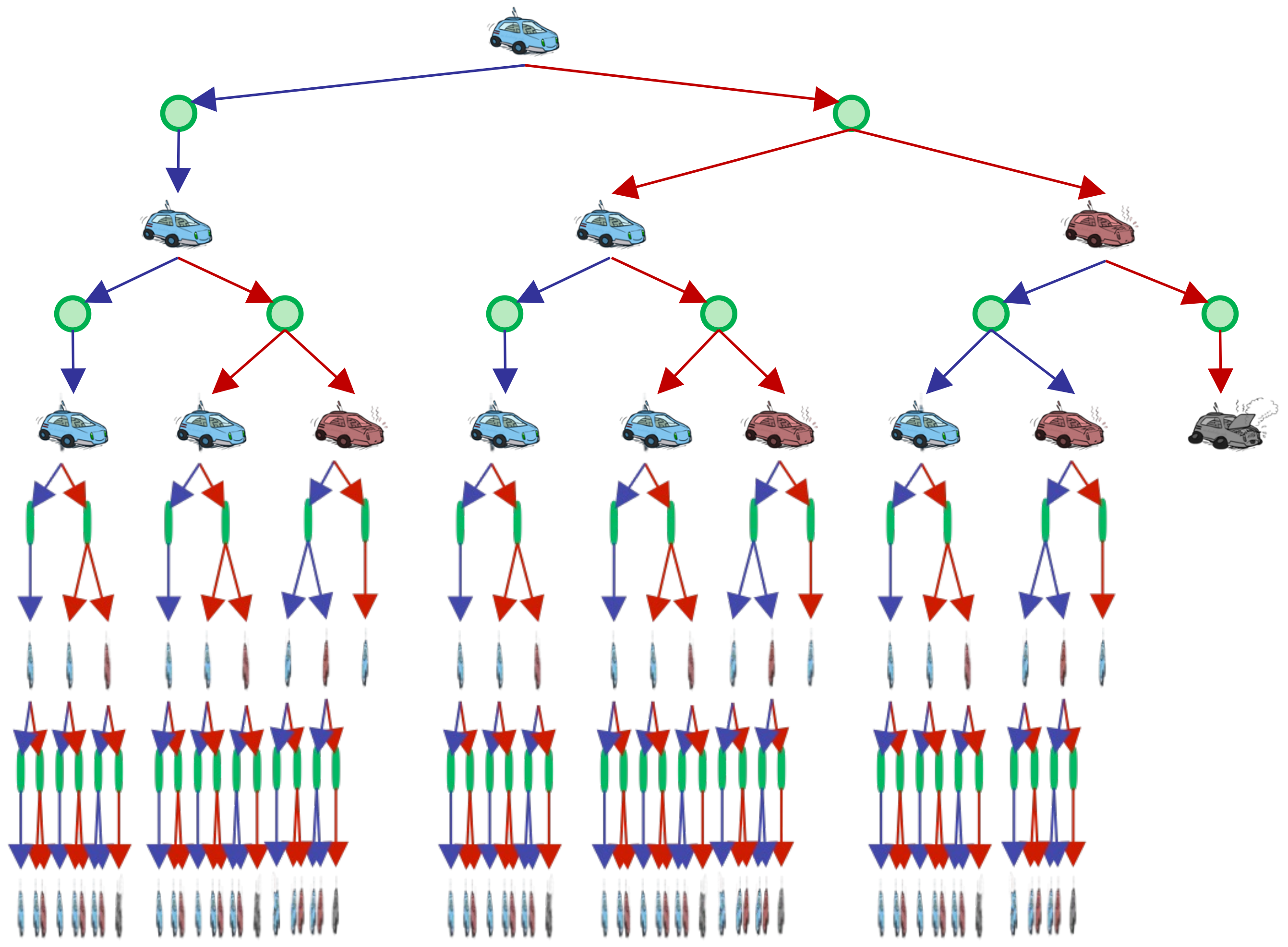


# Racing Search Tree



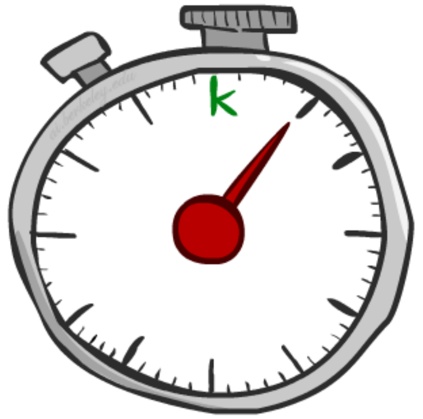
# Racing Search Tree

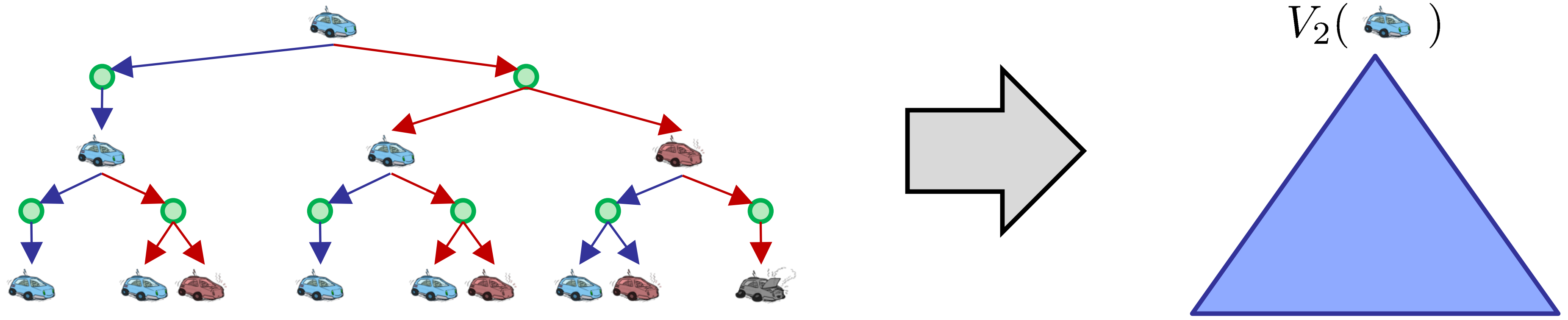
- We're doing way too much work with expectimax!
- Problem: States are repeated
  - Idea: Only compute needed quantities once
- Problem: Tree goes on forever
  - Idea: Do a depth-limited computation, but with increasing depths until change is small
  - Note: deep parts of the tree eventually don't matter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



# Time-Limited Values

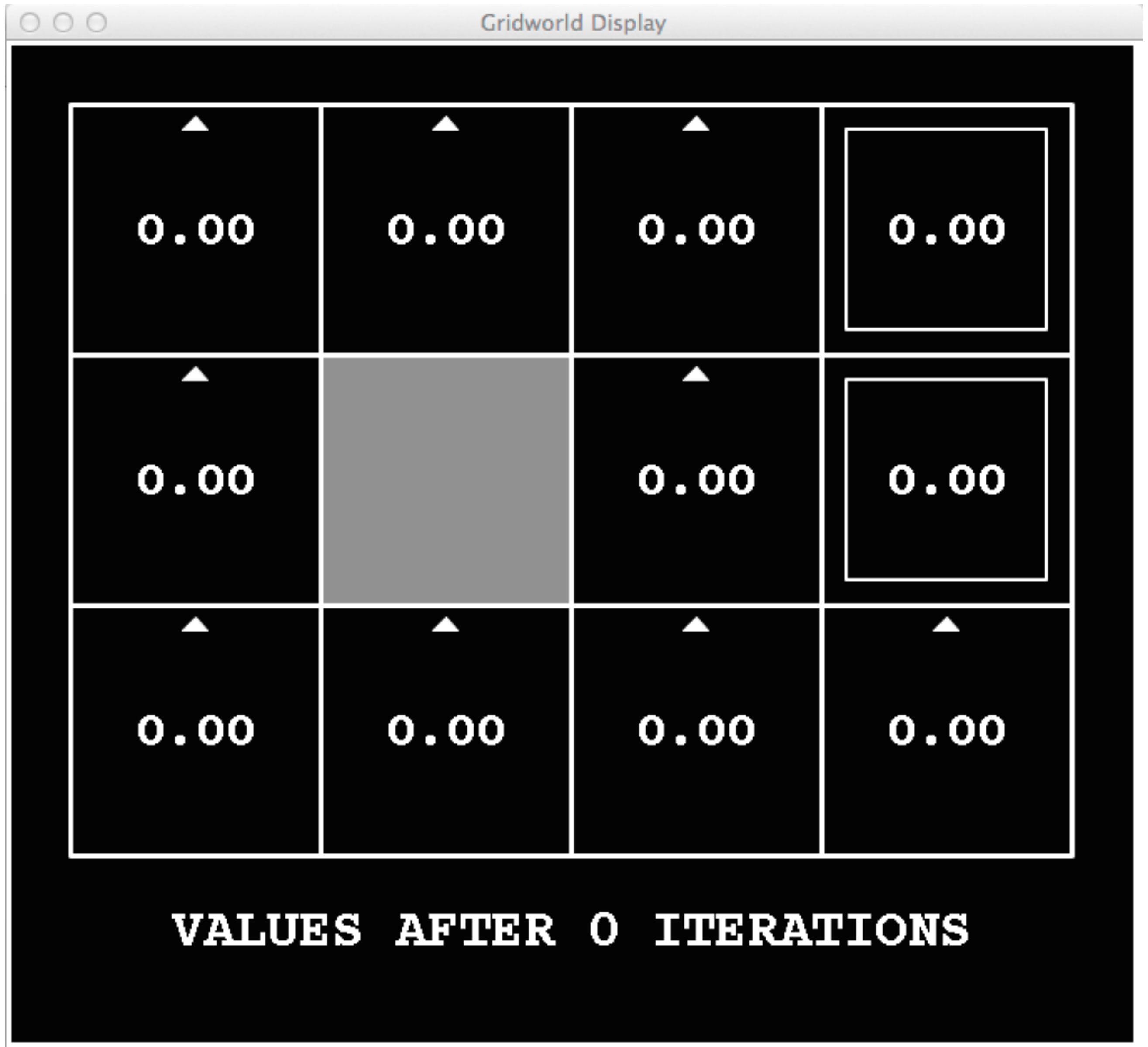
- Key idea: time-limited values
- Define $V_k(s)$ to be the optimal value of s if the game ends in k more time steps
  - Equivalently, it's what a depth-k expectimax would give from s





[Demo – time-limited values (L8D4)]

## k=0

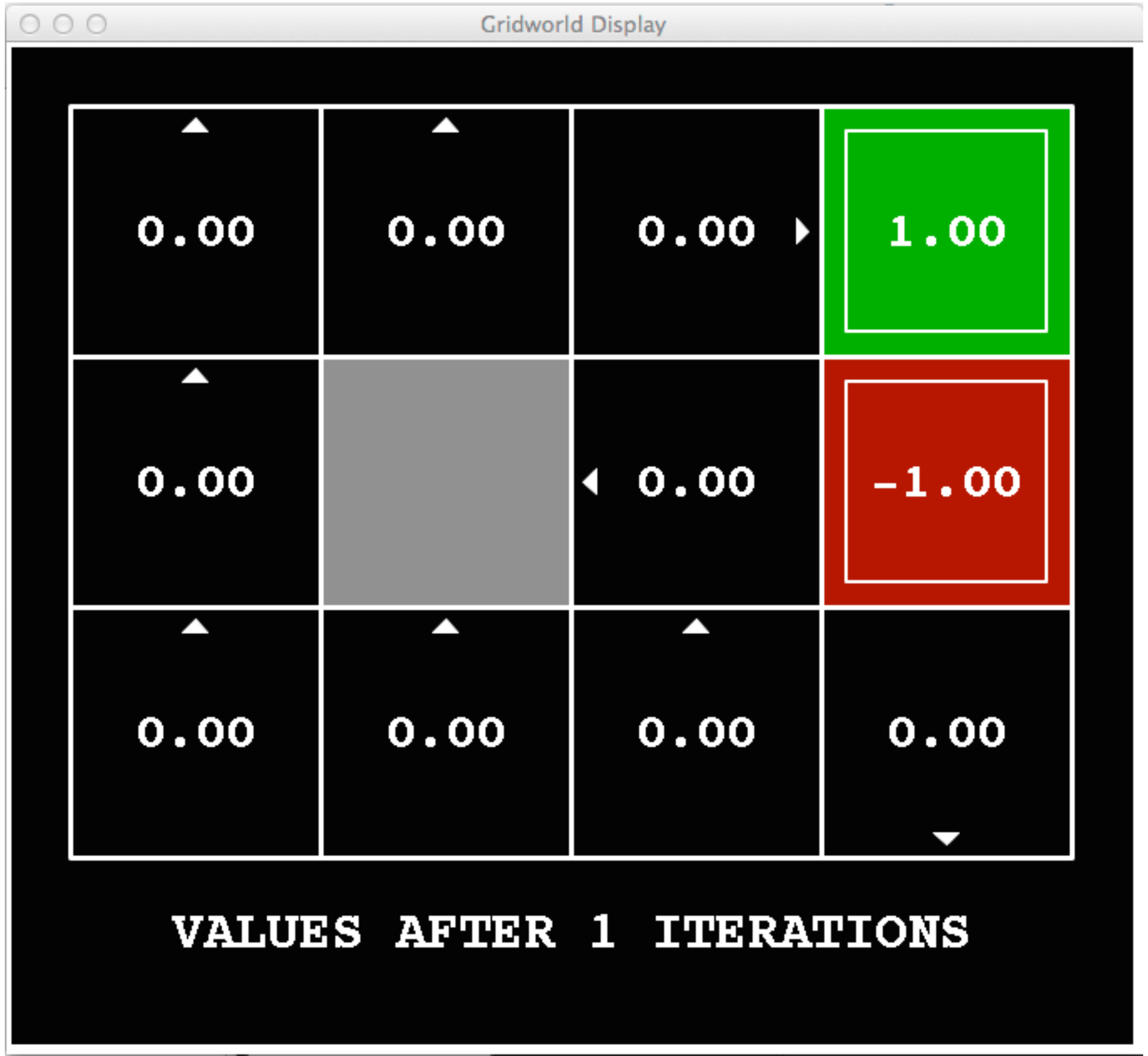


| | | | |
|---|---|---|---|
| 0.00 | 0.00 | 0.00 | 0.00 |
| 0.00 | | 0.00 | 0.00 |
| 0.00 | 0.00 | 0.00 | 0.00 |

VALUES AFTER 0 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

## k=1

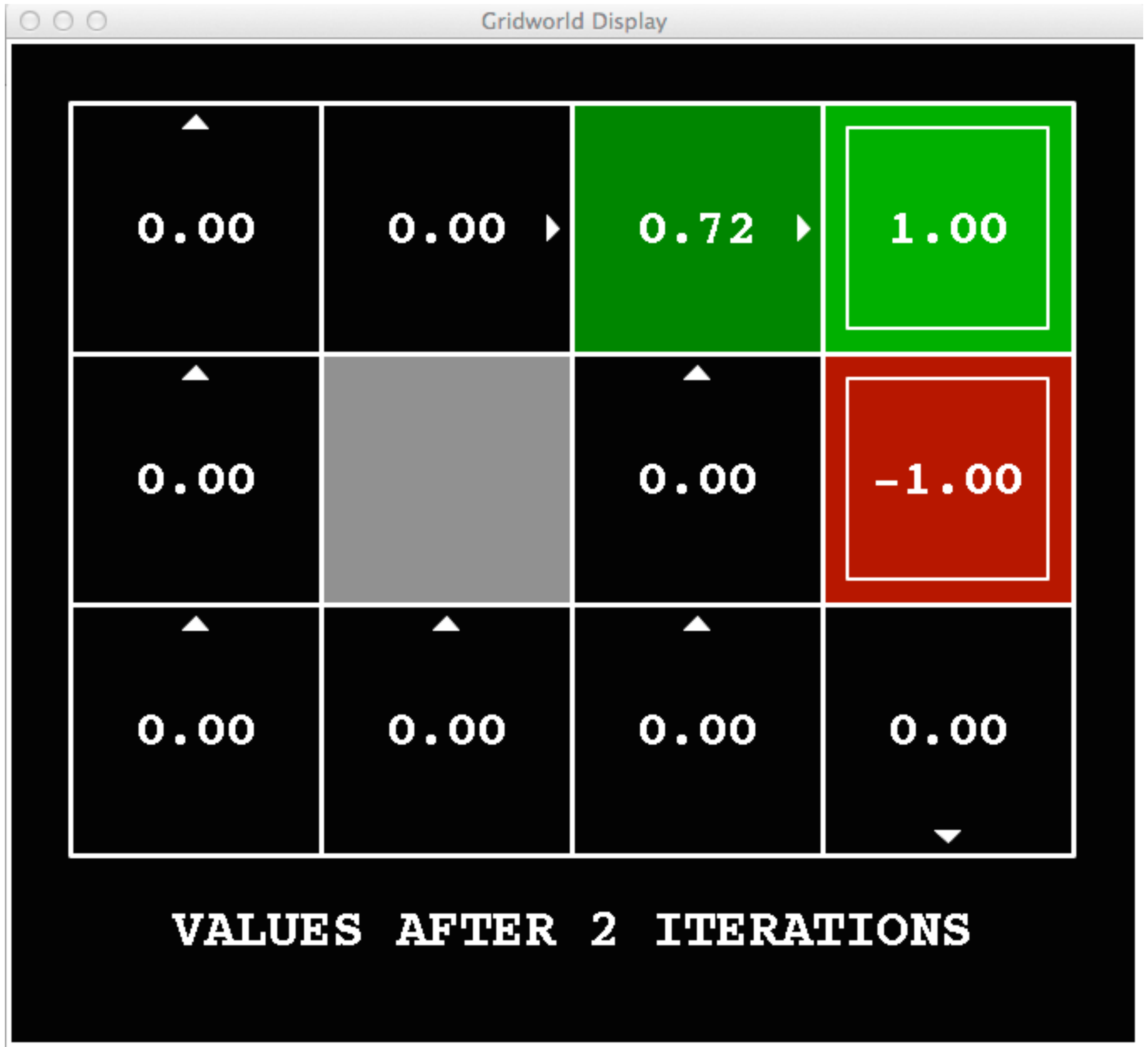


| | | | |
|---|---|---|---|
| 0.00 | 0.00 | 0.00 | 1.00 |
| 0.00 | | 0.00 | -1.00 |
| 0.00 | 0.00 | 0.00 | 0.00 |

VALUES AFTER 1 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

# k=2

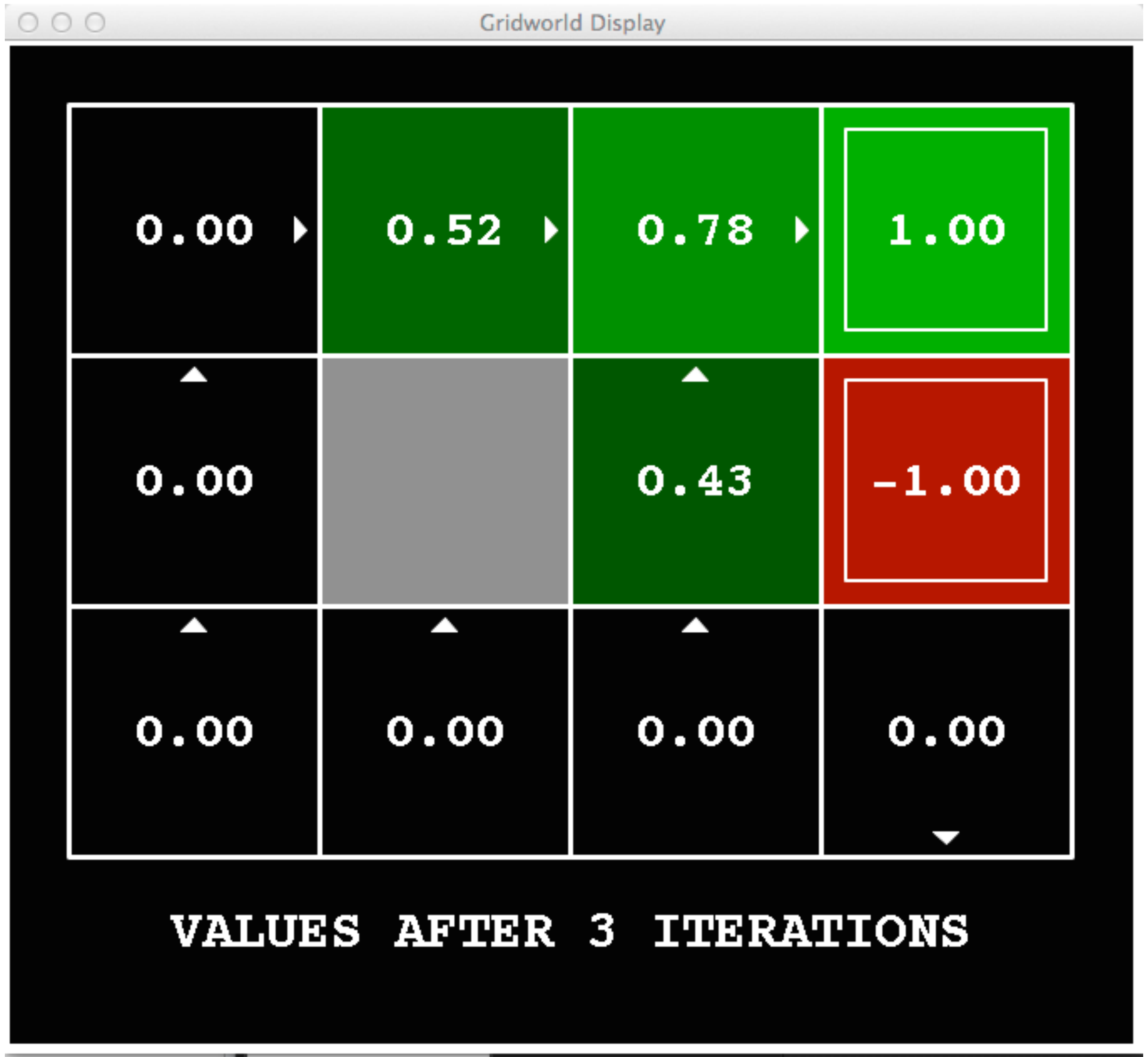


| | | | |
|---|---|---|---|
| 0.00 ▲ | 0.00 ▶ | 0.72 ▶ | 1.00 |
| 0.00 ▲ | | 0.00 ▲ | -1.00 |
| 0.00 ▲ | 0.00 ▲ | 0.00 ▲ | 0.00 ▼ |

VALUES AFTER 2 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

# k=3

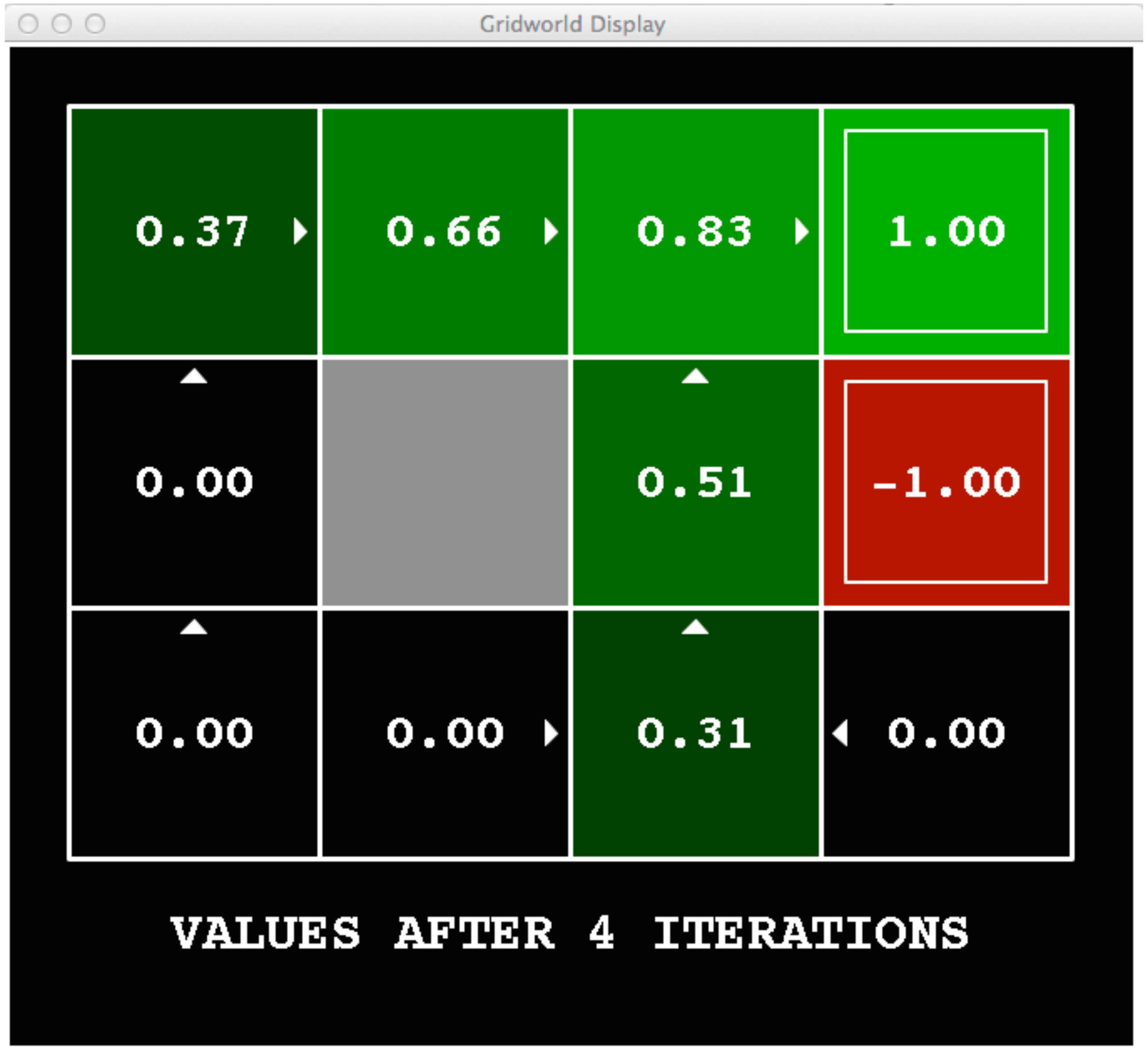


|  |  |  |  |
|---|---|---|---|
| 0.00 ▶ | 0.52 ▶ | 0.78 ▶ | 1.00 |
| ▲ 0.00 |  | ▲ 0.43 | -1.00 |
| ▲ 0.00 | ▲ 0.00 | ▲ 0.00 | 0.00 ▼ |

VALUES AFTER 3 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

# k=4

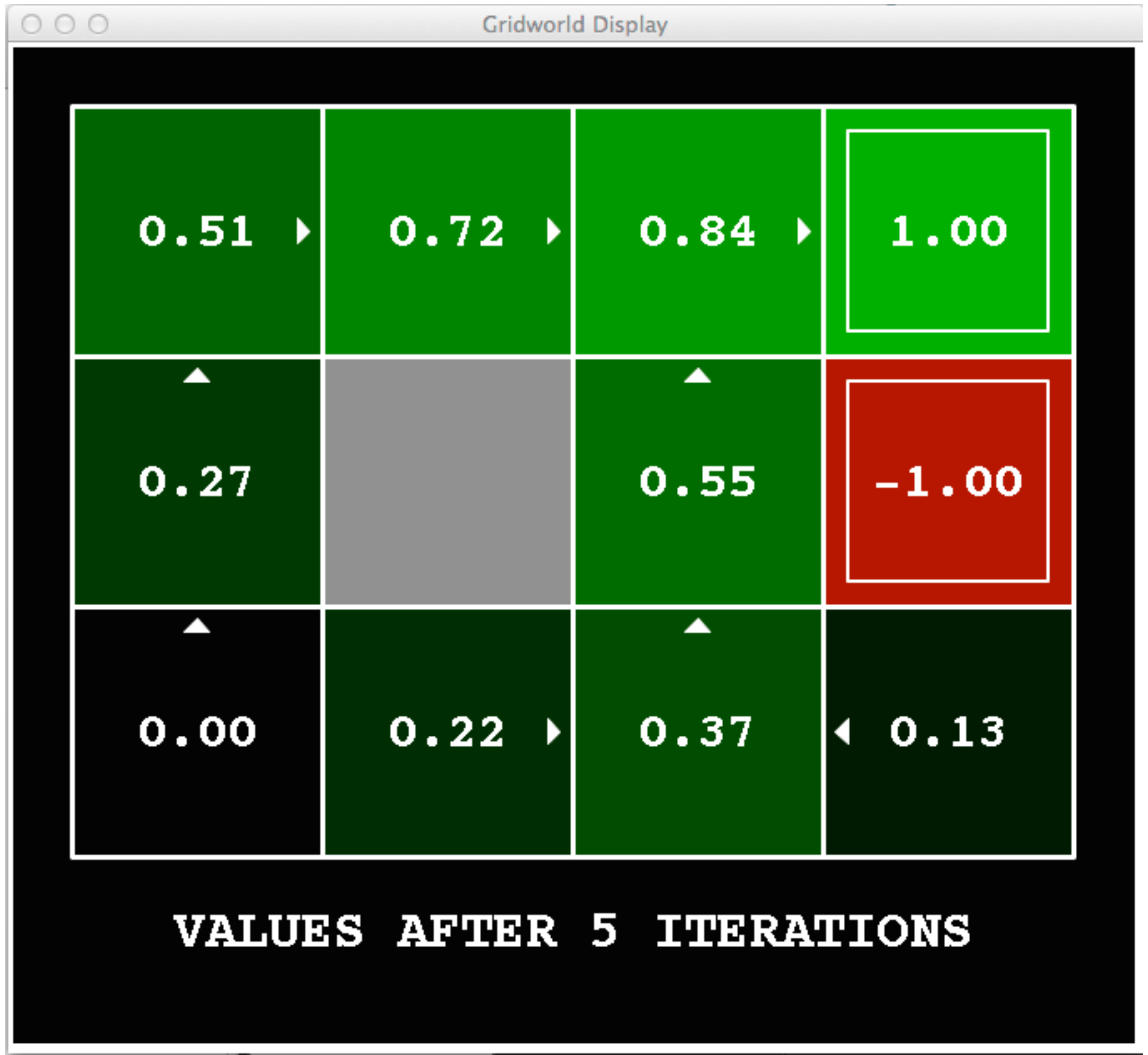


| | | | |
|---|---|---|---|
| 0.37 ▸ | 0.66 ▸ | 0.83 ▸ | 1.00 |
| ▴ 0.00 | | ▴ 0.51 | -1.00 |
| ▴ 0.00 | 0.00 ▸ | ▴ 0.31 | ◂ 0.00 |

VALUES AFTER 4 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

# k=5

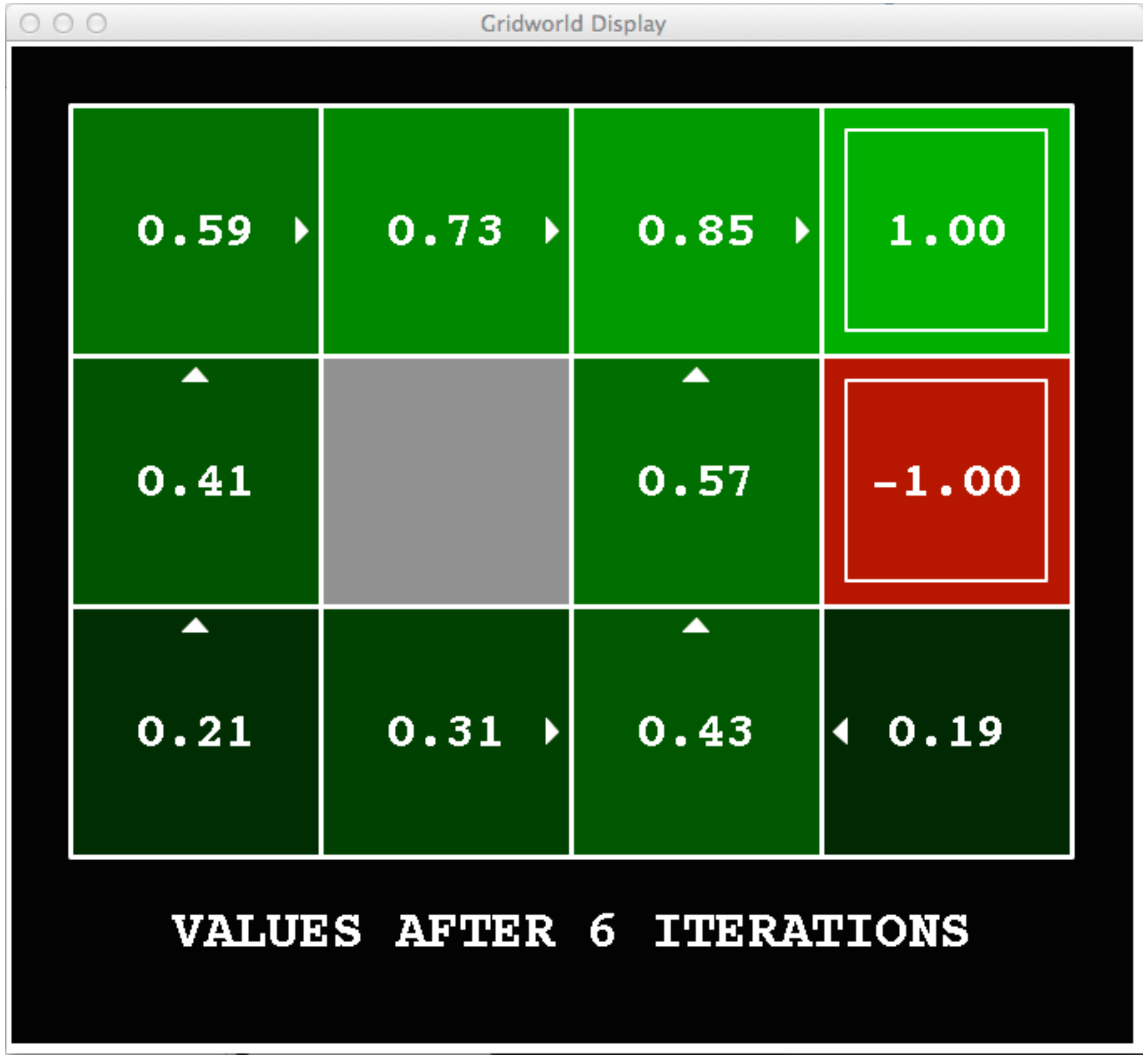


| | | | |
|---|---|---|---|
| 0.51 ▸ | 0.72 ▸ | 0.84 ▸ | 1.00 |
| ▴ 0.27 | | ▴ 0.55 | -1.00 |
| ▴ 0.00 | 0.22 ▸ | ▴ 0.37 | ◂ 0.13 |

VALUES AFTER 5 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

# k=6

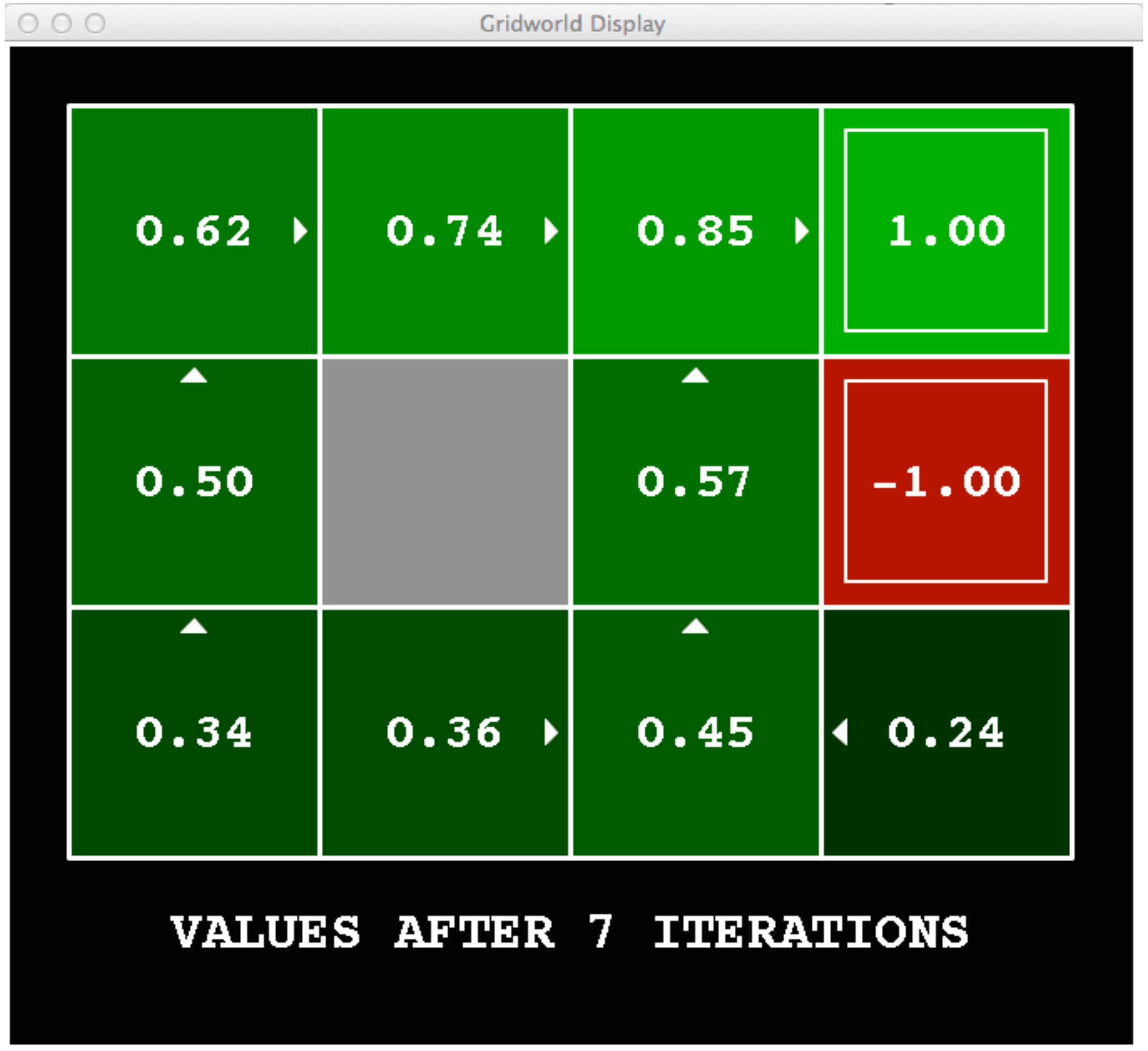


| | | | |
|---|---|---|---|
| 0.59 ▶ | 0.73 ▶ | 0.85 ▶ | 1.00 |
| ▲ 0.41 | | ▲ 0.57 | -1.00 |
| ▲ 0.21 | 0.31 ▶ | ▲ 0.43 | ◀ 0.19 |

VALUES AFTER 6 ITERATIONS

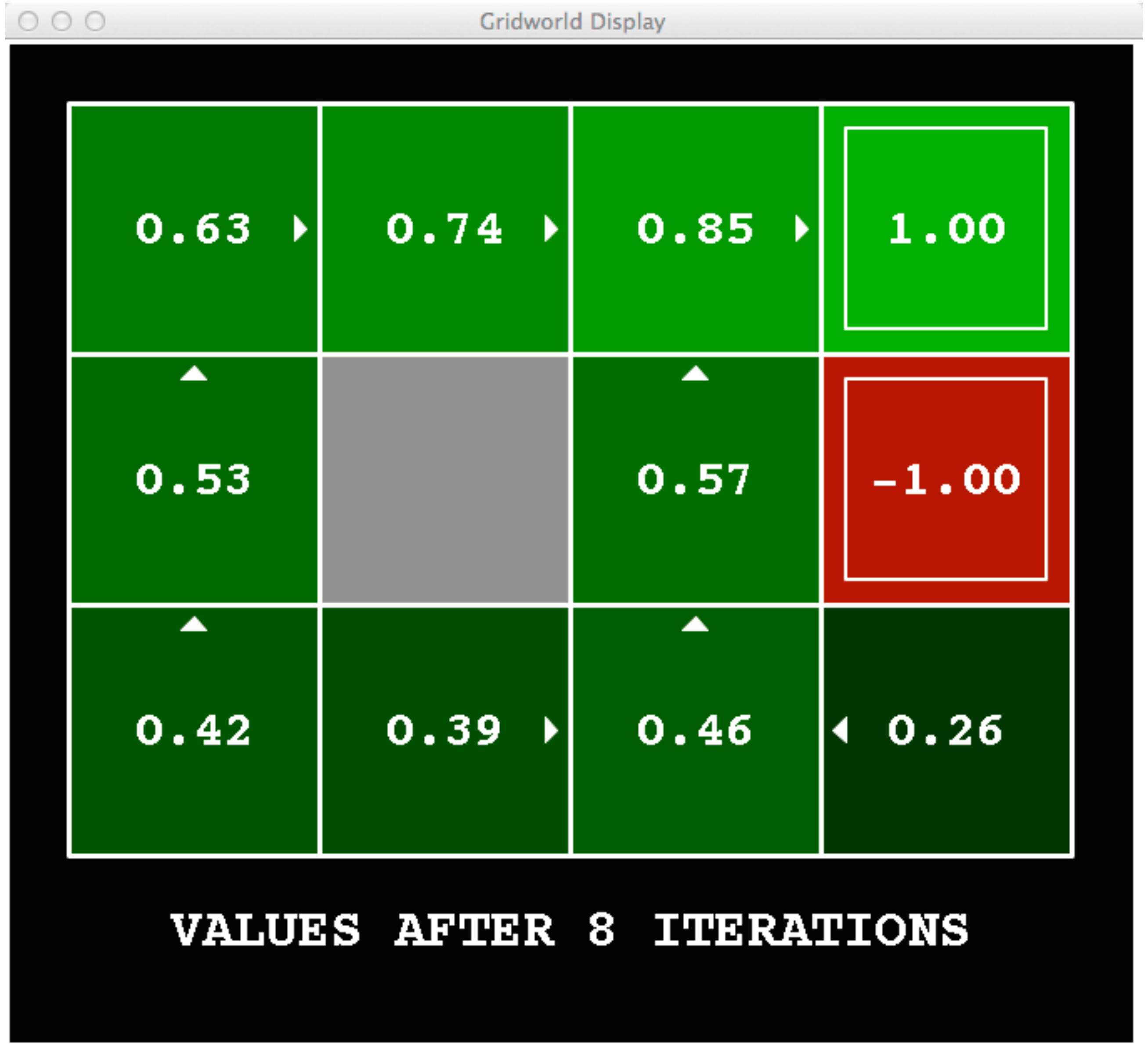
Noise = 0.2
Discount = 0.9
Living reward = 0

# k=7



Noise = 0.2
Discount = 0.9
Living reward = 0

# k=8

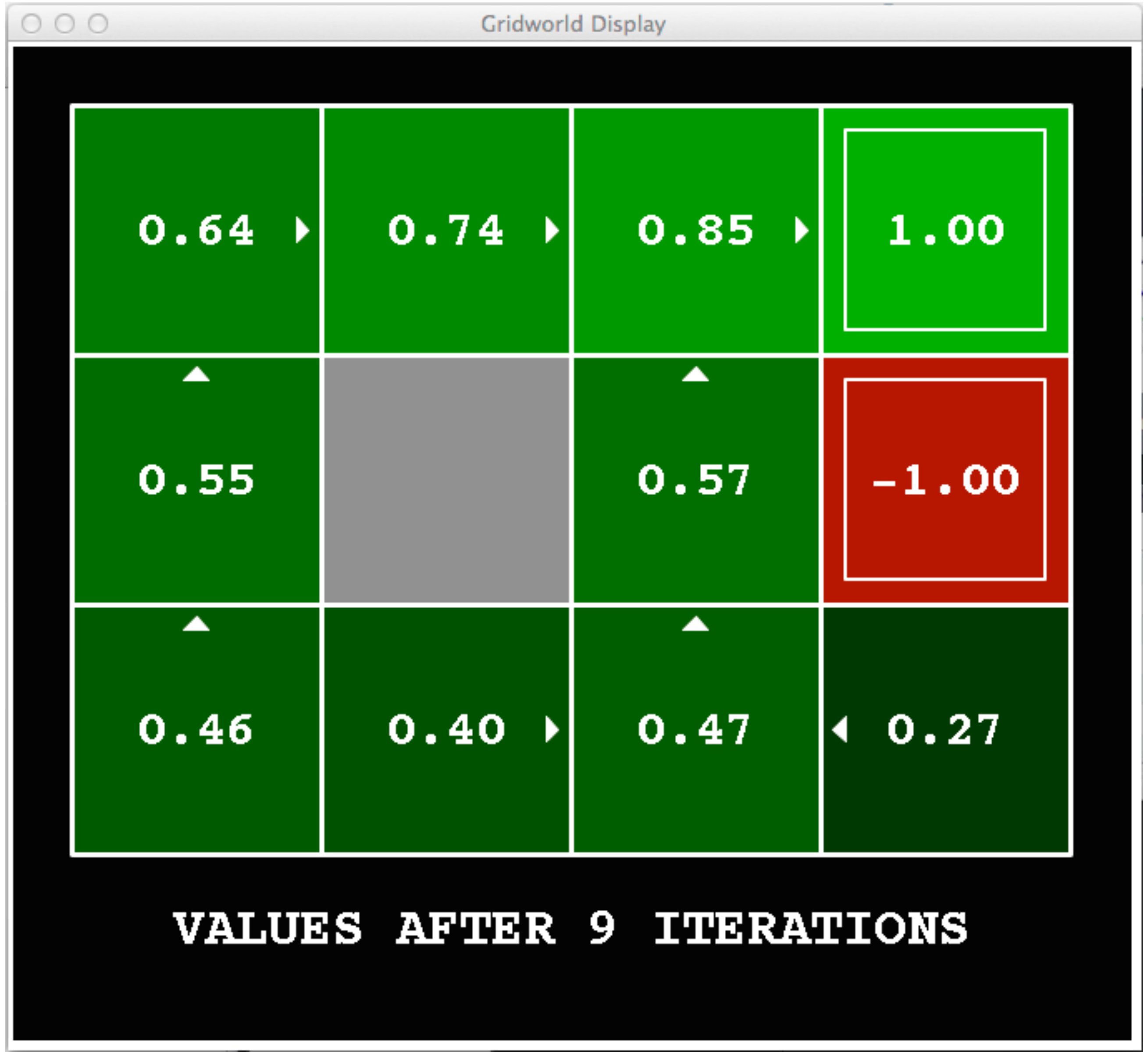


| | | | |
|---|---|---|---|
| 0.63 ▶ | 0.74 ▶ | 0.85 ▶ | 1.00 |
| ▲ 0.53 | | ▲ 0.57 | -1.00 |
| ▲ 0.42 | 0.39 ▶ | ▲ 0.46 | ◀ 0.26 |

VALUES AFTER 8 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

# k=9

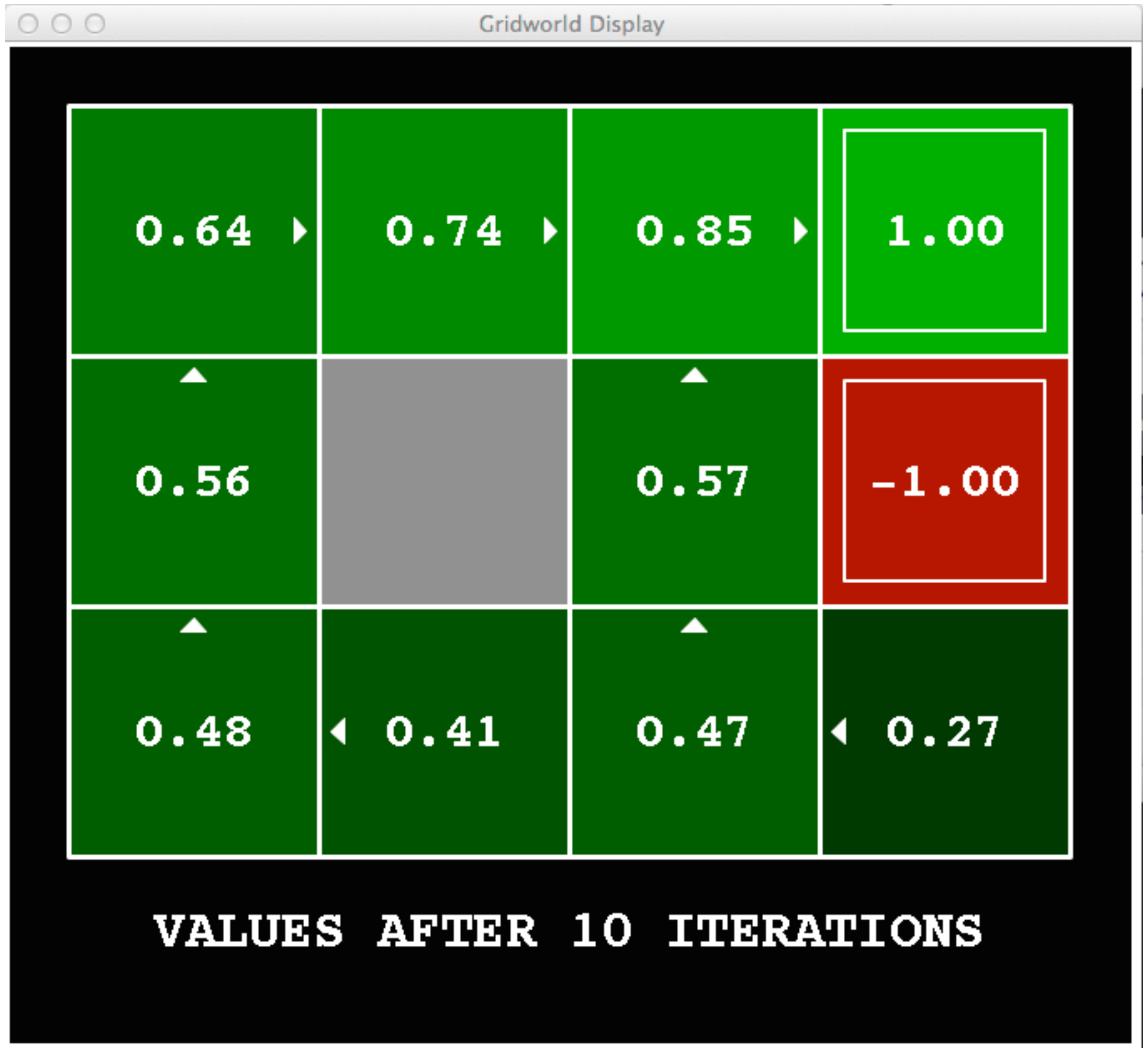


| | | | |
|---|---|---|---|
| 0.64 ▶ | 0.74 ▶ | 0.85 ▶ | 1.00 |
| ▲ 0.55 | | ▲ 0.57 | -1.00 |
| ▲ 0.46 | 0.40 ▶ | ▲ 0.47 | ◀ 0.27 |

VALUES AFTER 9 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

# k=10

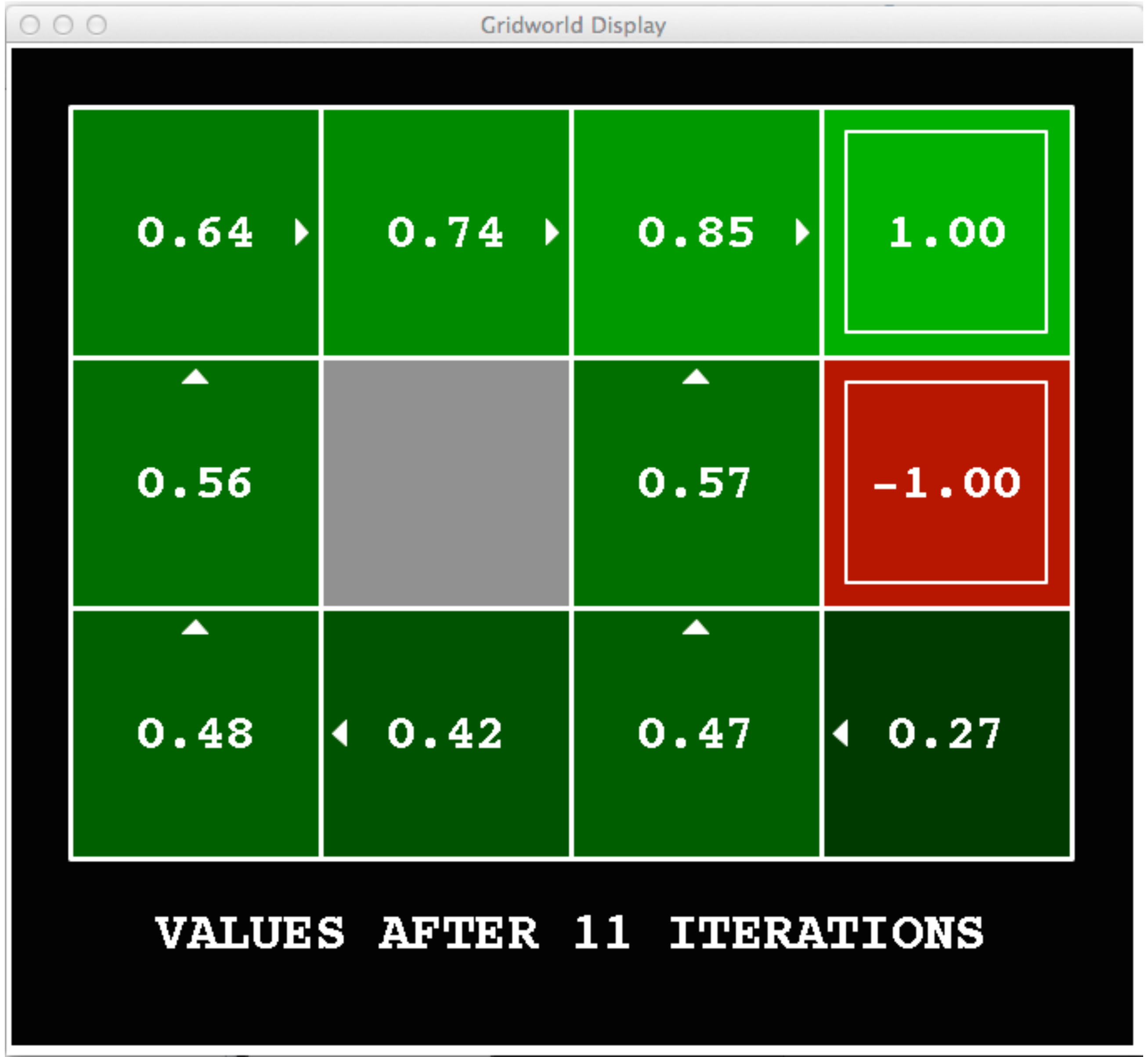


| | | | |
|---|---|---|---|
| 0.64 ▶ | 0.74 ▶ | 0.85 ▶ | 1.00 |
| ▲ 0.56 | | ▲ 0.57 | -1.00 |
| ▲ 0.48 | ◀ 0.41 | ▲ 0.47 | ◀ 0.27 |

VALUES AFTER 10 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

# k=11

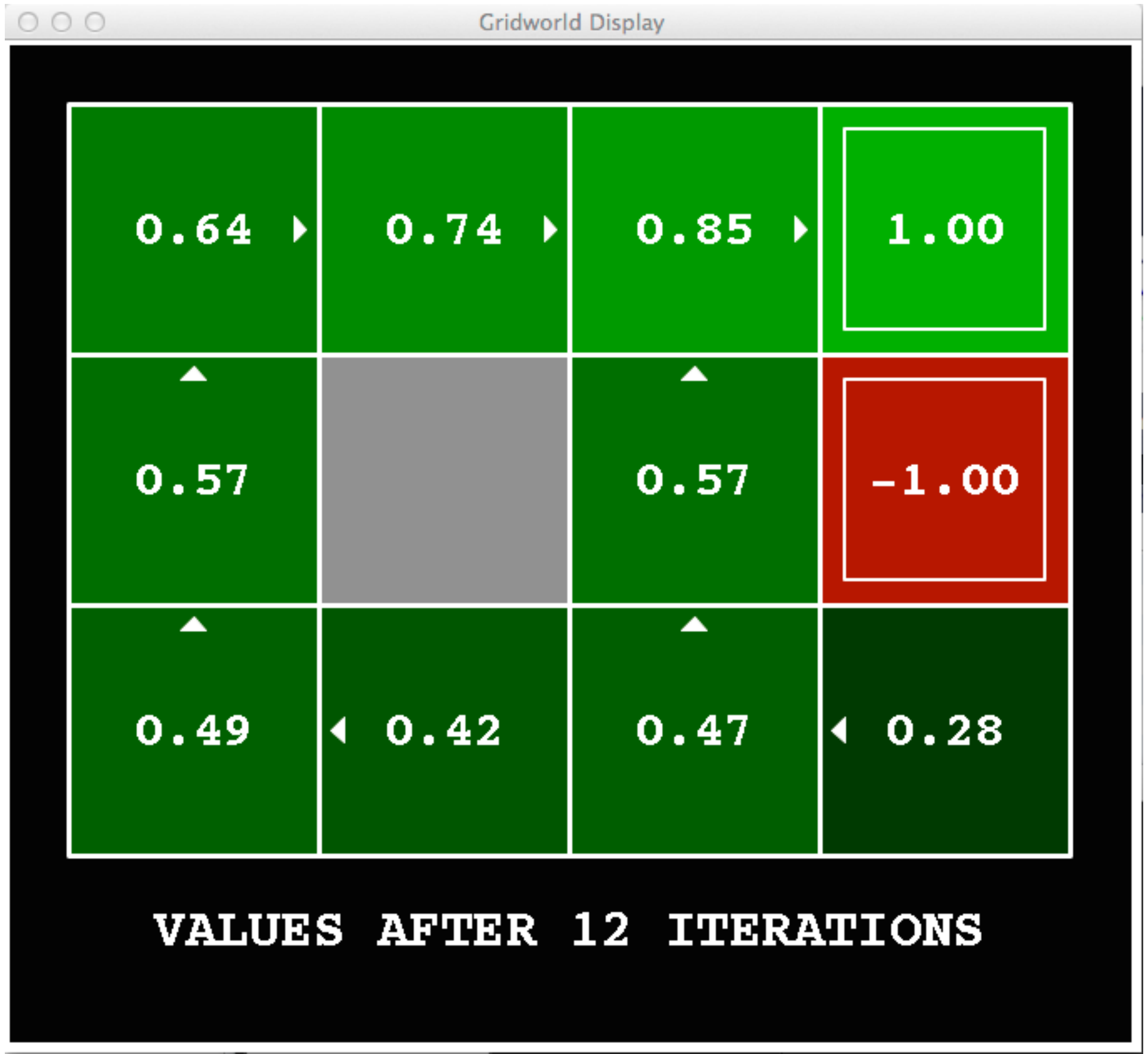


| | | | |
|---|---|---|---|
| 0.64 ▶ | 0.74 ▶ | 0.85 ▶ | 1.00 |
| ▲ 0.56 | | ▲ 0.57 | -1.00 |
| ▲ 0.48 | ◀ 0.42 | ▲ 0.47 | ◀ 0.27 |

VALUES AFTER 11 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

# k=12

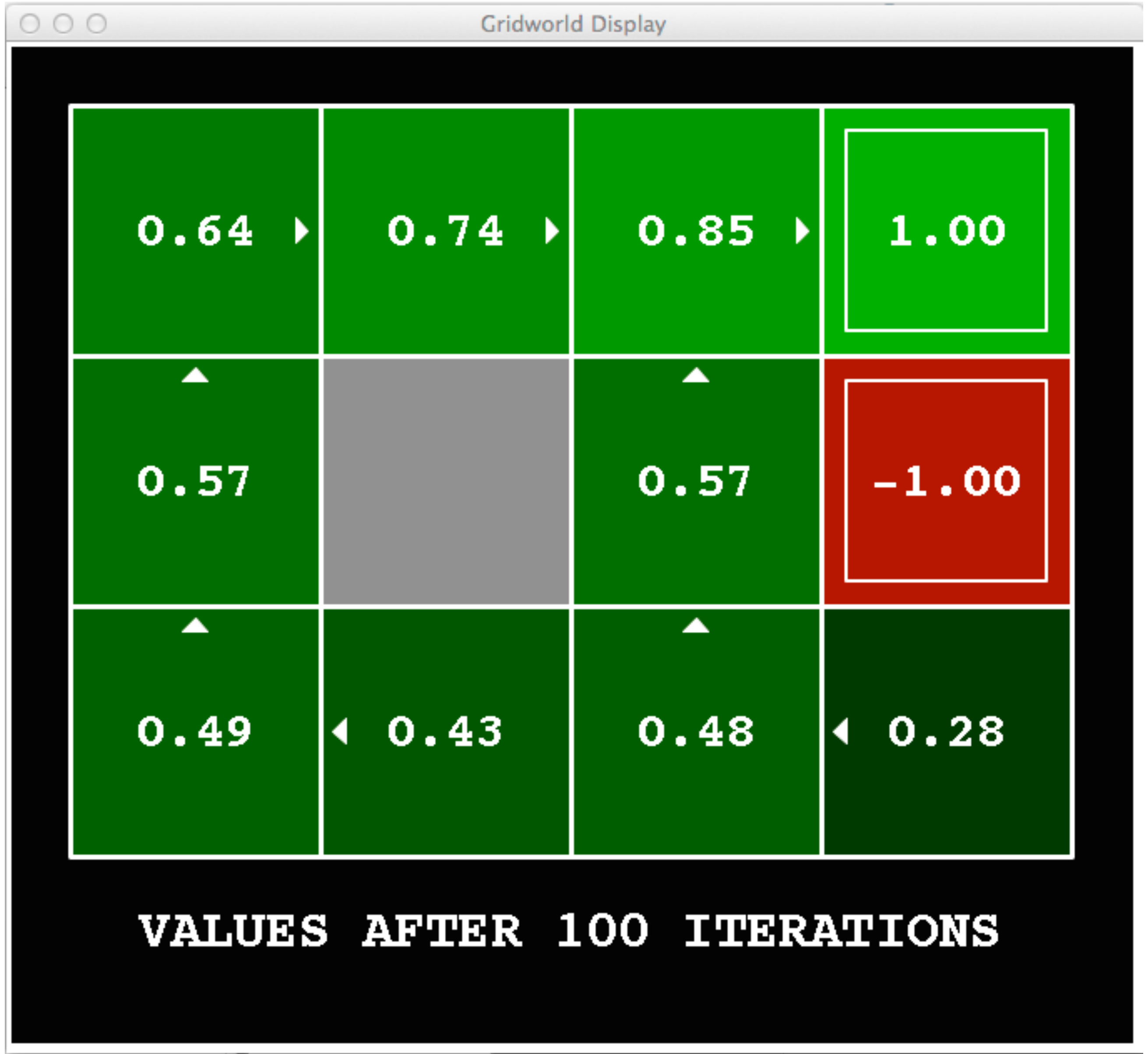


| | | | |
|---|---|---|---|
| 0.64 ▶ | 0.74 ▶ | 0.85 ▶ | 1.00 |
| ▲ 0.57 | | ▲ 0.57 | -1.00 |
| ▲ 0.49 | ◀ 0.42 | ▲ 0.47 | ◀ 0.28 |

VALUES AFTER 12 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

# k=100

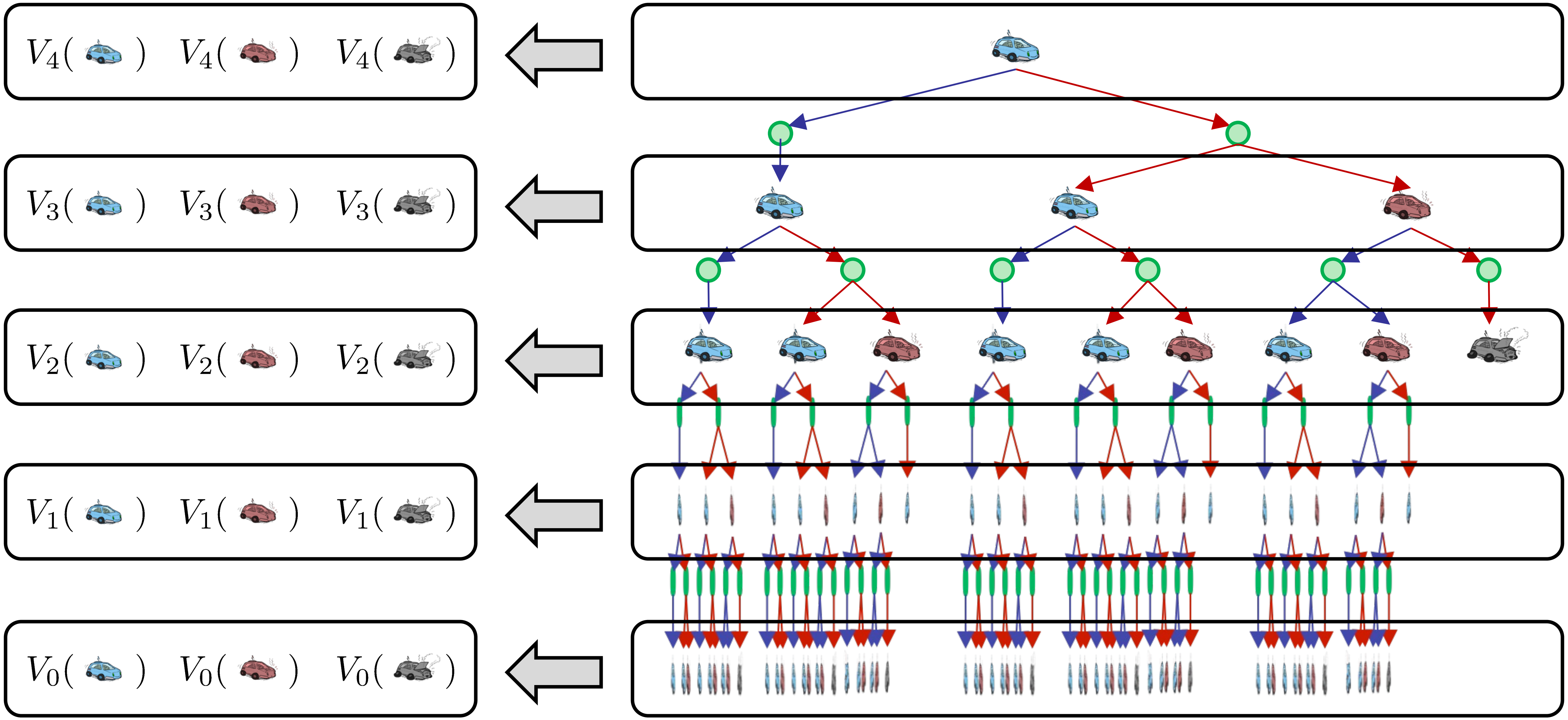


| | | | |
|---|---|---|---|
| 0.64 ▸ | 0.74 ▸ | 0.85 ▸ | 1.00 |
| ▴ 0.57 | | ▴ 0.57 | -1.00 |
| ▴ 0.49 | ◂ 0.43 | ▴ 0.48 | ◂ 0.28 |

VALUES AFTER 100 ITERATIONS

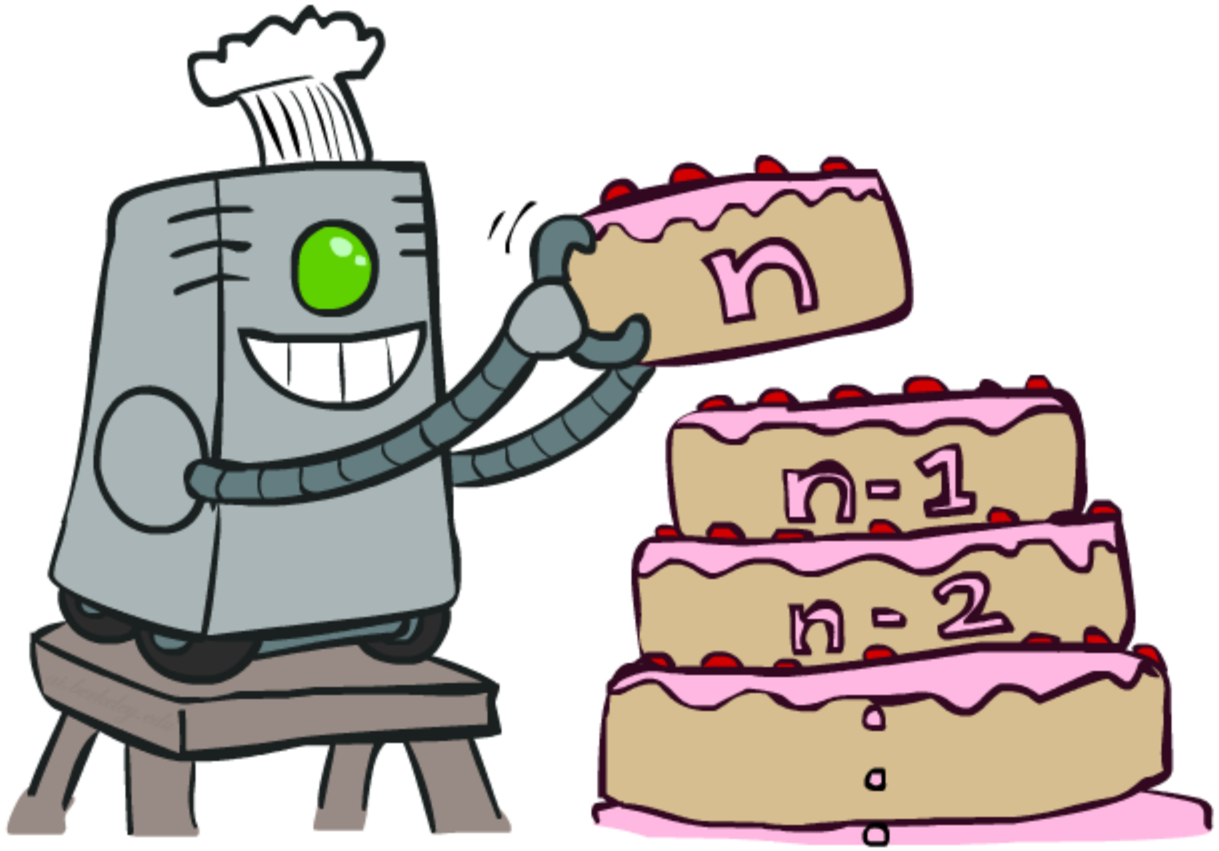
Noise = 0.2
Discount = 0.9
Living reward = 0

# Computing Time-Limited Values



$V_4(\;)$ $V_4(\;)$ $V_4(\;)$
$V_3(\;)$ $V_3(\;)$ $V_3(\;)$
$V_2(\;)$ $V_2(\;)$ $V_2(\;)$
$V_1(\;)$ $V_1(\;)$ $V_1(\;)$
$V_0(\;)$ $V_0(\;)$ $V_0(\;)$

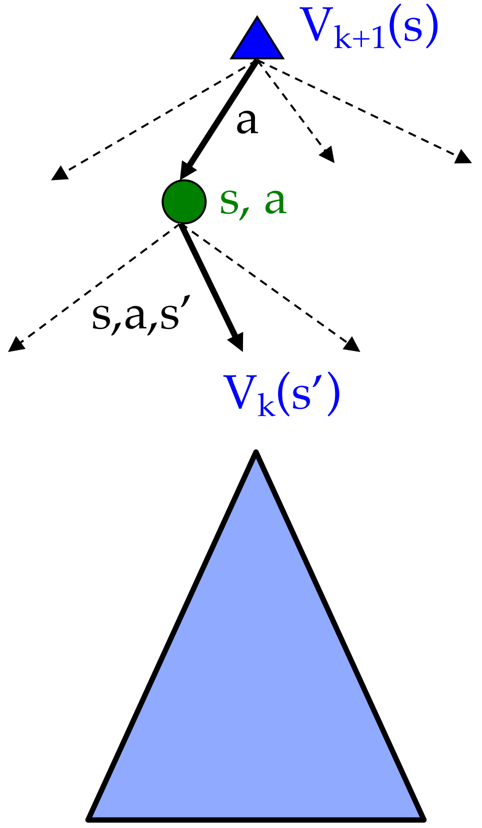
# Value Iteration



# 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, which yields V*
- 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

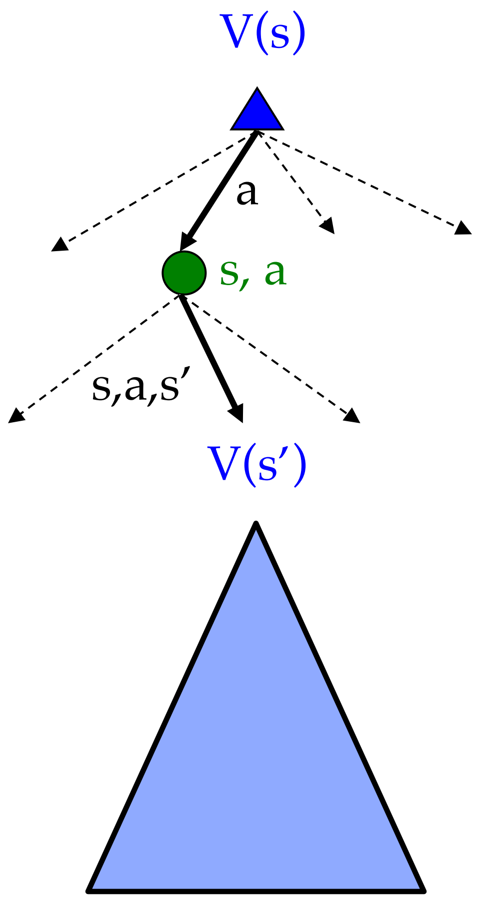
## Value Iteration

- Bellman equations characterize the optimal values:

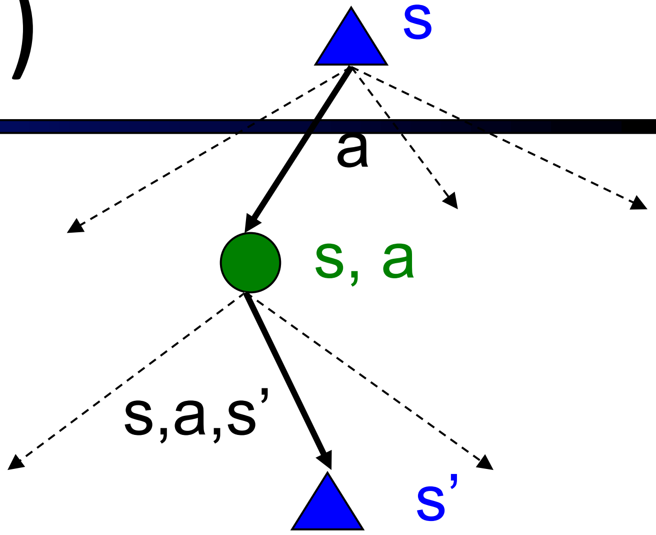
$$V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s')\right]$$

- 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 (again ☺ )



- Init:

$$\forall s: \quad V(s) = 0$$

- Iterate:

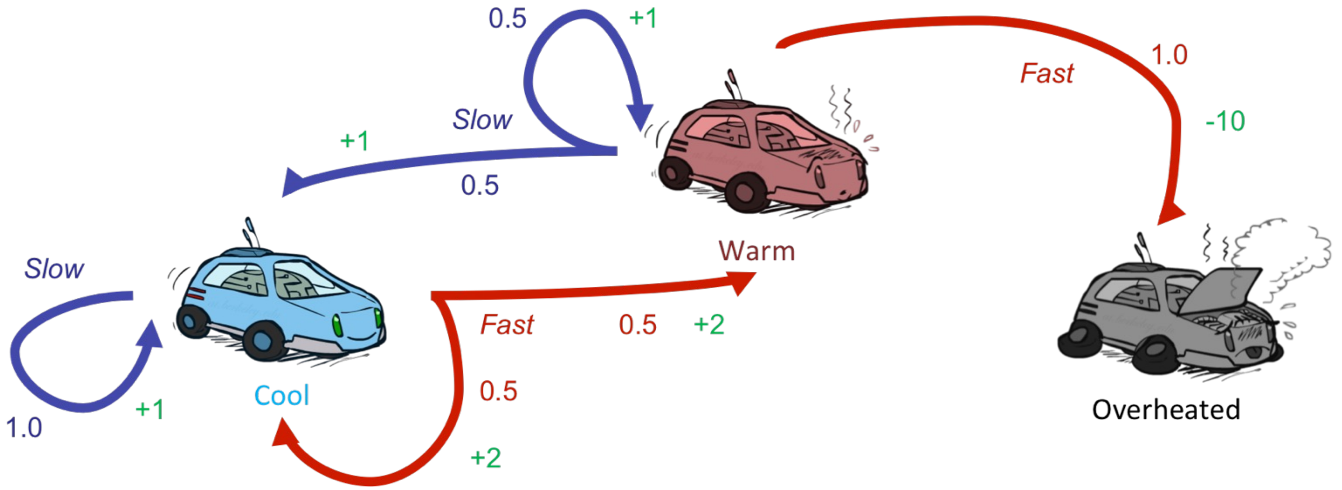
$$\forall s: \quad V_{new}(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V(s')]$$

$$V = V_{new}$$

Note: can even directly assign to *V(s)*, which will not compute the sequence of $V_k$ but will still converge to $V^*$

# Example: Value Iteration

|  | | | |
|---|---|---|---|
| $V_2$ |  |  |  |
| $V_1$ | S: 1<br>F: .5*2+.5*2=2 |  |  |
| $V_0$ | 0 | 0 | 0 |

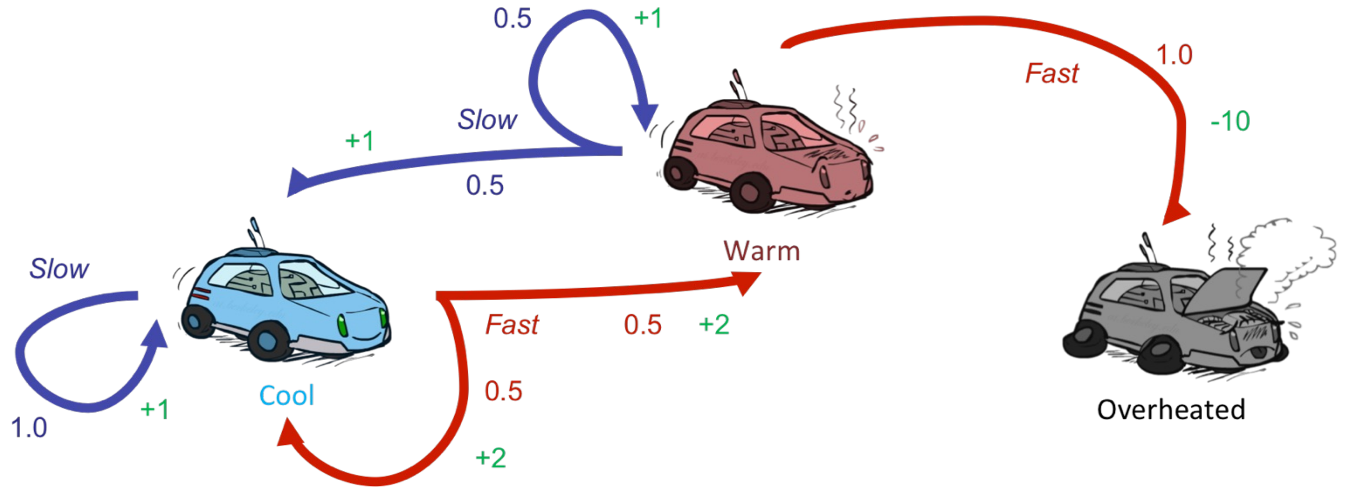


*Assume no discount!*

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

# Example: Value Iteration

| | | | |
|---|---|---|---|
| $V_2$ | | | |
| $V_1$ | 2 | S: .5*1+.5*1=1<br>F: -10 | |
| $V_0$ | 0 | 0 | 0 |

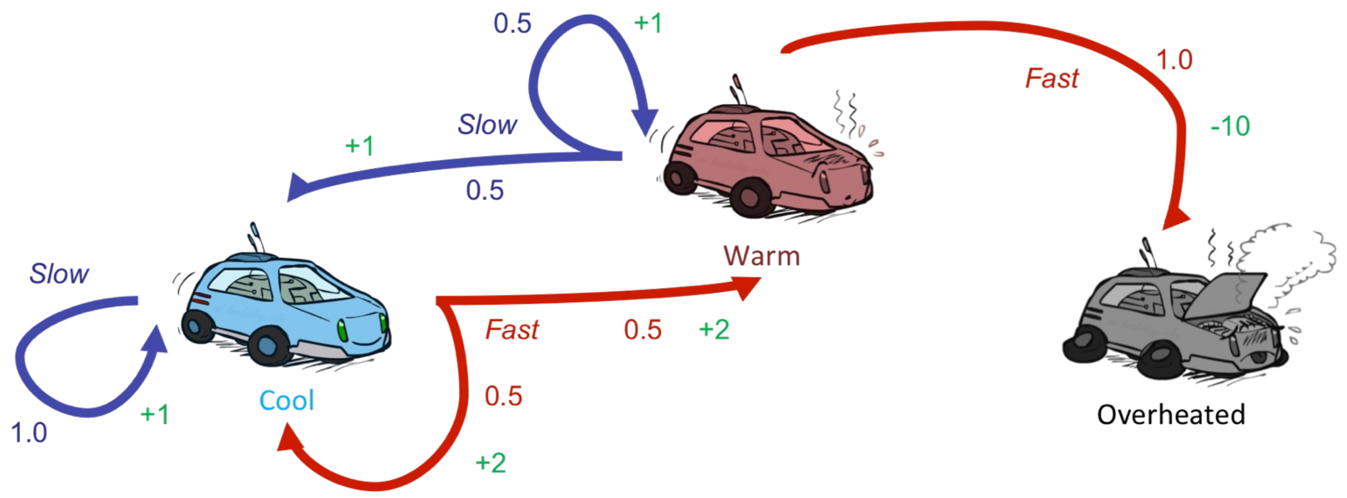


*Assume no discount!*

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

# Example: Value Iteration

| | | | |
|---|---|---|---|
| $V_2$ | | | |
| $V_1$ | 2 | 1 | 0 |
| $V_0$ | 0 | 0 | 0 |

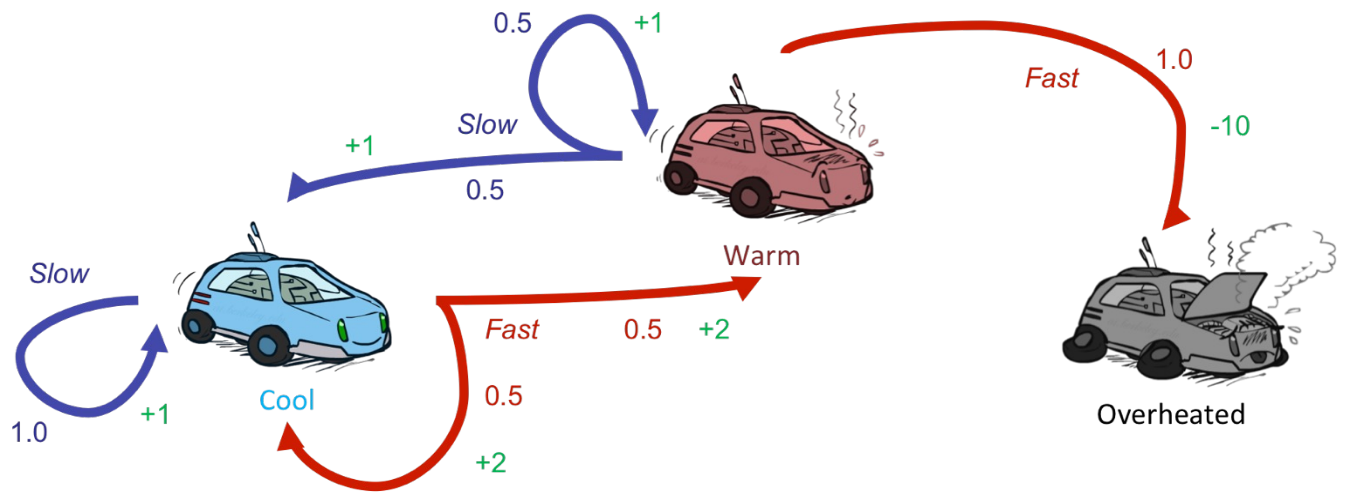


*Assume no discount!*

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

# Example: Value Iteration

| | | | |
|---|---|---|---|
| $V_2$ | S: 1+2=3<br>F: .5*(2+2)+.5*(2+1)=3.5 | | |
| $V_1$ | 2 | 1 | 0 |
| $V_0$ | 0 | 0 | 0 |

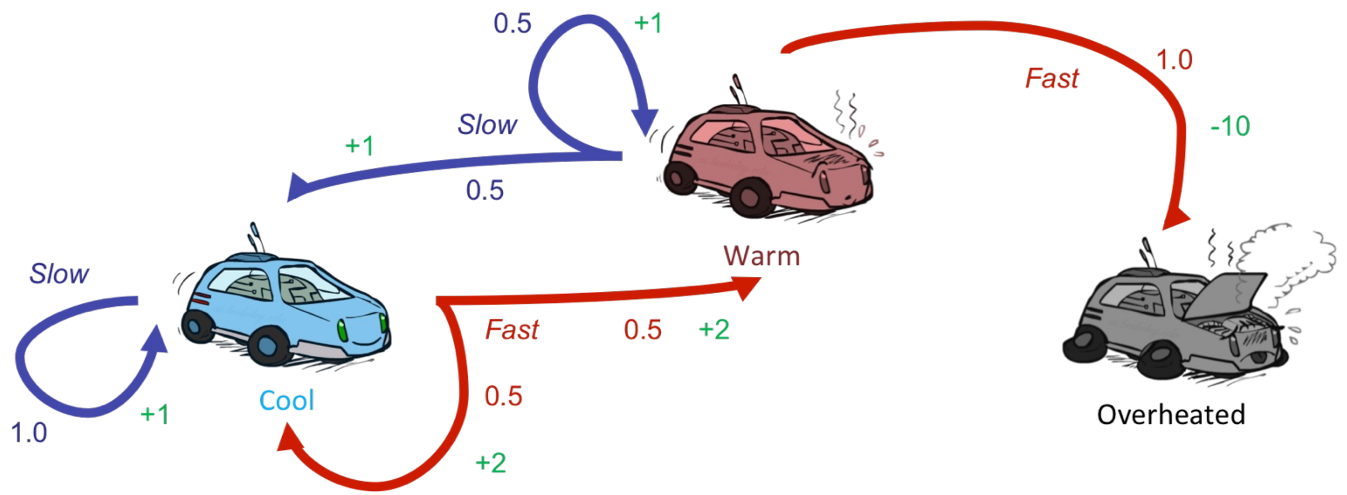


*Assume no discount!*

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

# Example: Value Iteration

| | | | |
|---|---|---|---|
| $V_2$ | 3.5 | 2.5 | 0 |
| $V_1$ | 2 | 1 | 0 |
| $V_0$ | 0 | 0 | 0 |



*Assume no discount!*

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

# Convergence*

- How do we know the $V_k$ vectors are going to converge? (assuming $0 < \gamma < 1$)
- Proof 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 $\gamma^k$ that far out
  - So $V_k$ and $V_{k+1}$ are at most $\gamma^k$ max|R| different
  - So as k increases, the values converge