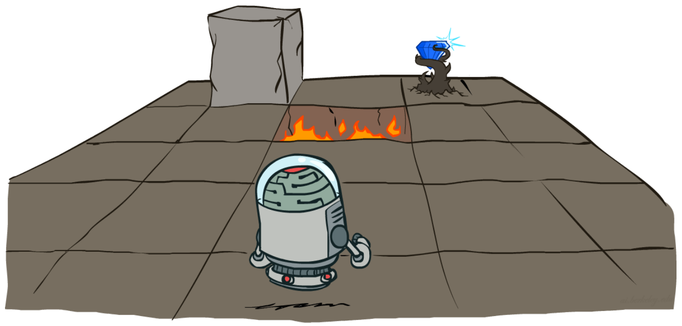
# CSE 473: Introduction to Artificial Intelligence

Hanna Hajishirzi

Markov Decision Processes



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

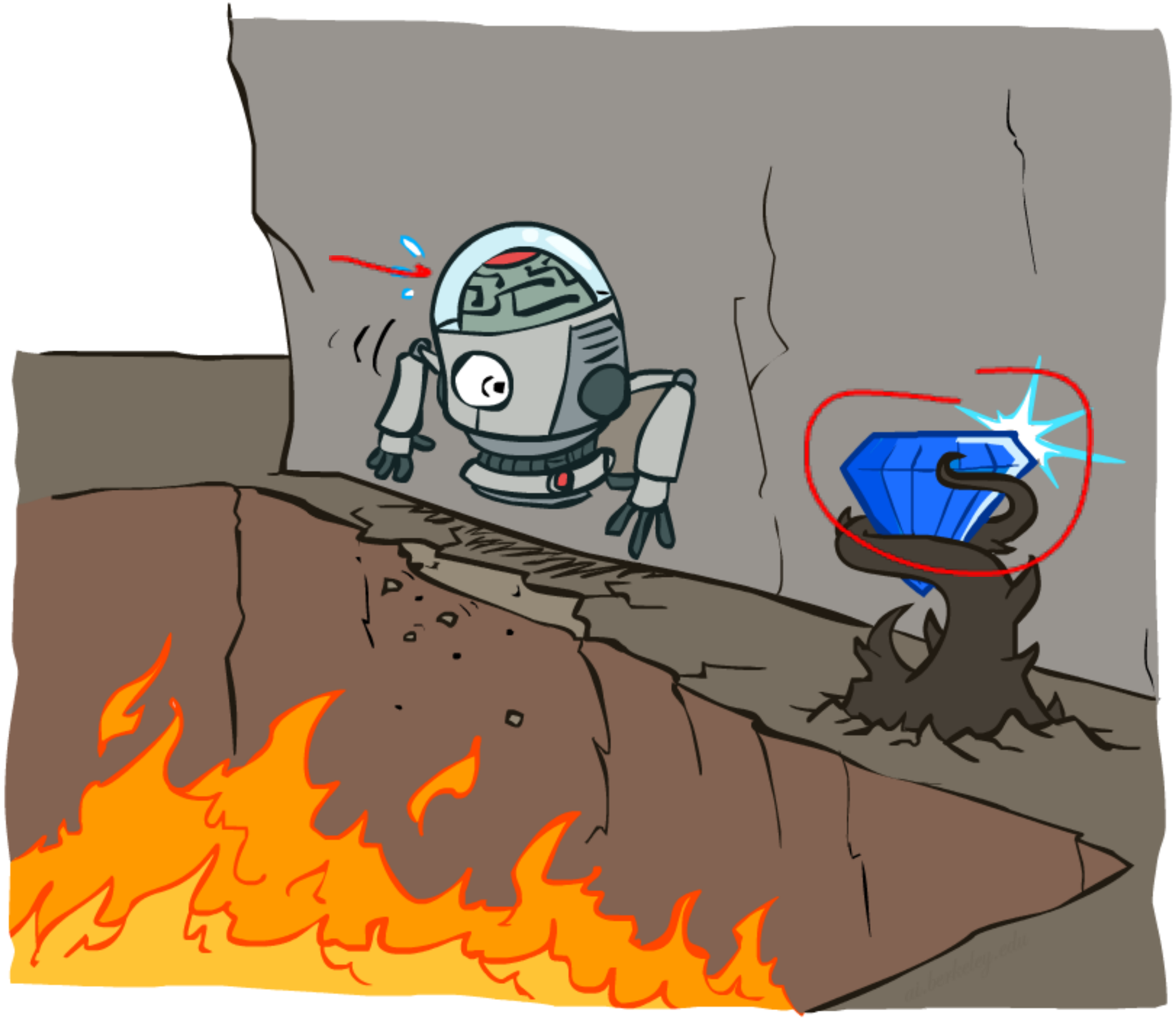
# Announcements

- HW1, PS1 grades are released.
- HW2 is released → DUE: Friday, 11:59pm
- PS2 -> DUE: next Wednesday

# Review and Outline

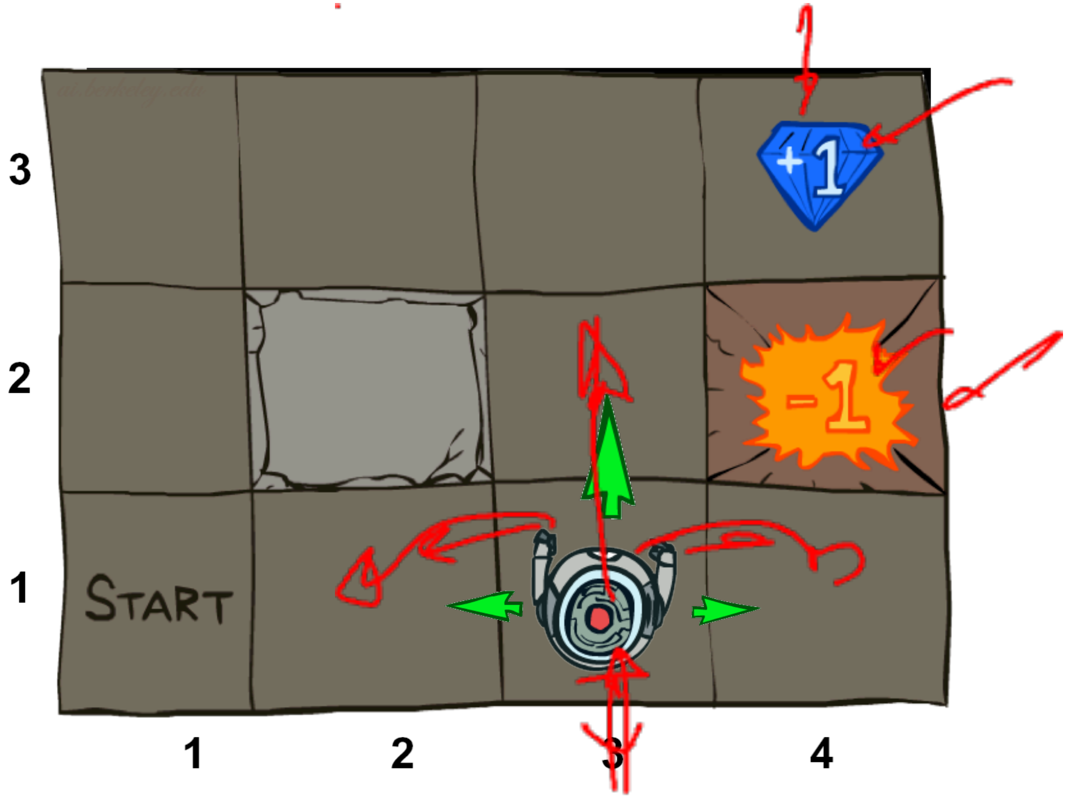
- Adversarial Games
  - Minimax search
  - α-β search
  - Evaluation functions
  - Multi-player, non-0-sum
- Stochastic Games
  - Expectimax
  - Markov Decision Processes
  - Reinforcement Learning

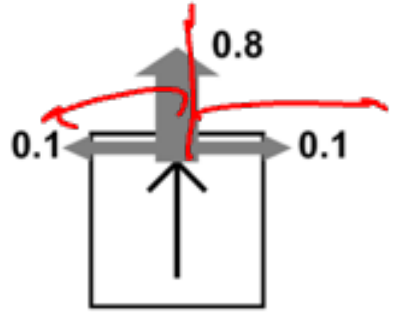
# Non-Deterministic Search



# Example: Grid World

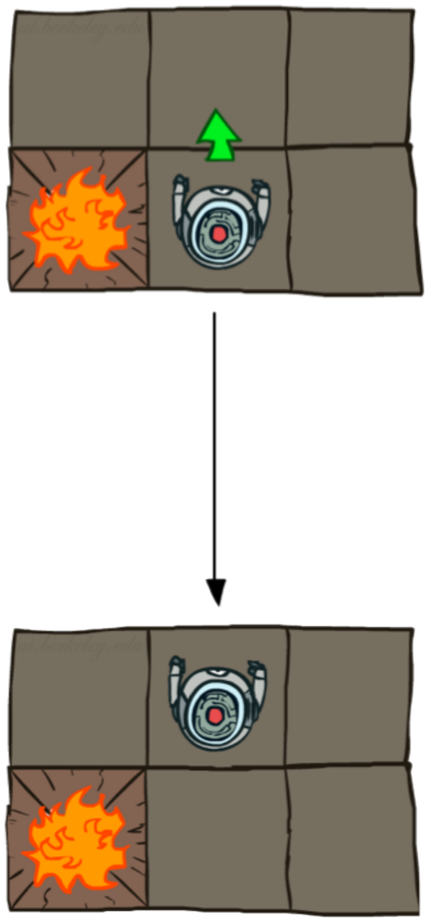
- A maze-like problem
  - The agent lives in a grid
  - Walls block the agent's path
- Noisy movement: actions do not always go as planned
  - 80% of the time, the action North takes the agent North
    (if there is no wall there)
  - 10% of the time, North takes the agent West; 10% East
  - If there 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)



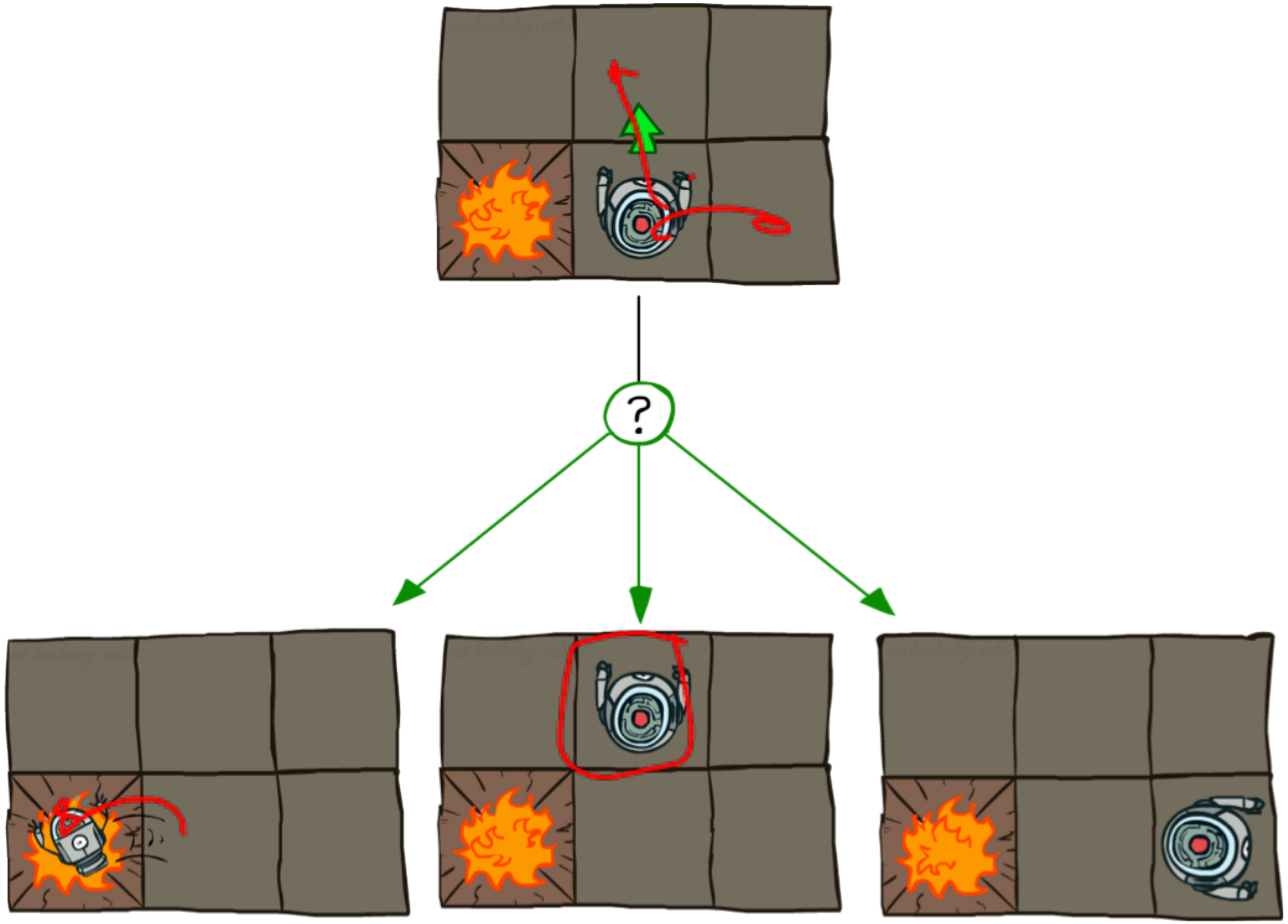



# Grid World Actions

Deterministic Grid World



Stochastic Grid World



# Markov Decision Processes

- An MDP is defined by:
  - A set of states $s \in S$
  - A set of actions $a \in A$ N, E, S, W
  - A transition function ~~T(s, a, s')~~
    - Probability that a from s leads to s', i.e., ~~P(s' | s, a)~~ $P(s' | s, a)$
    - Also called the model or the dynamics

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

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

$\ldots$

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

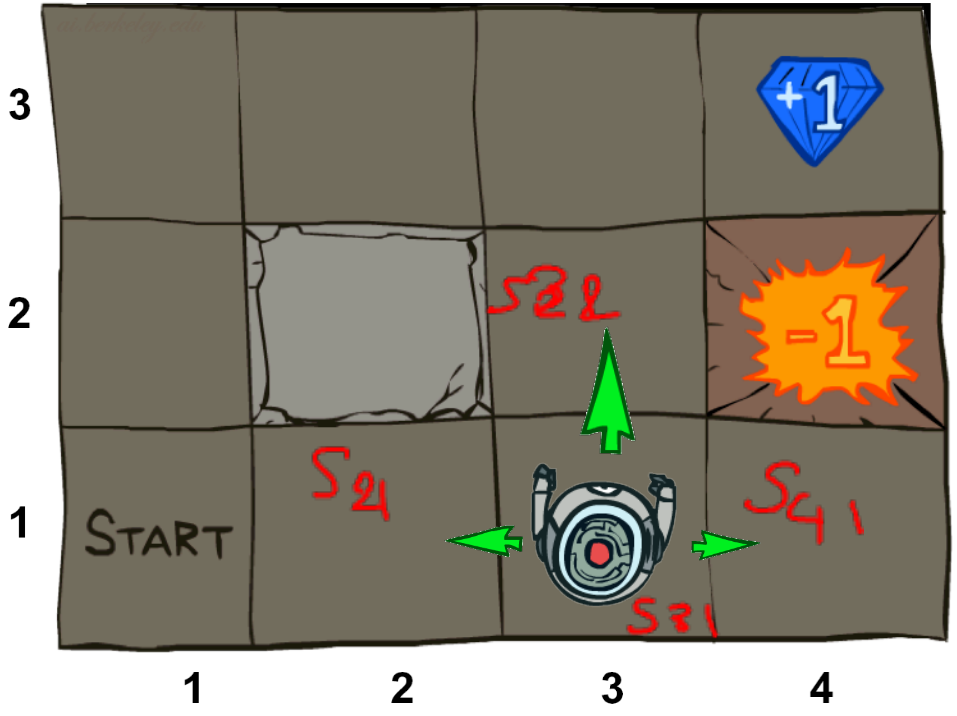
$\ldots$

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

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

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

$\ldots$

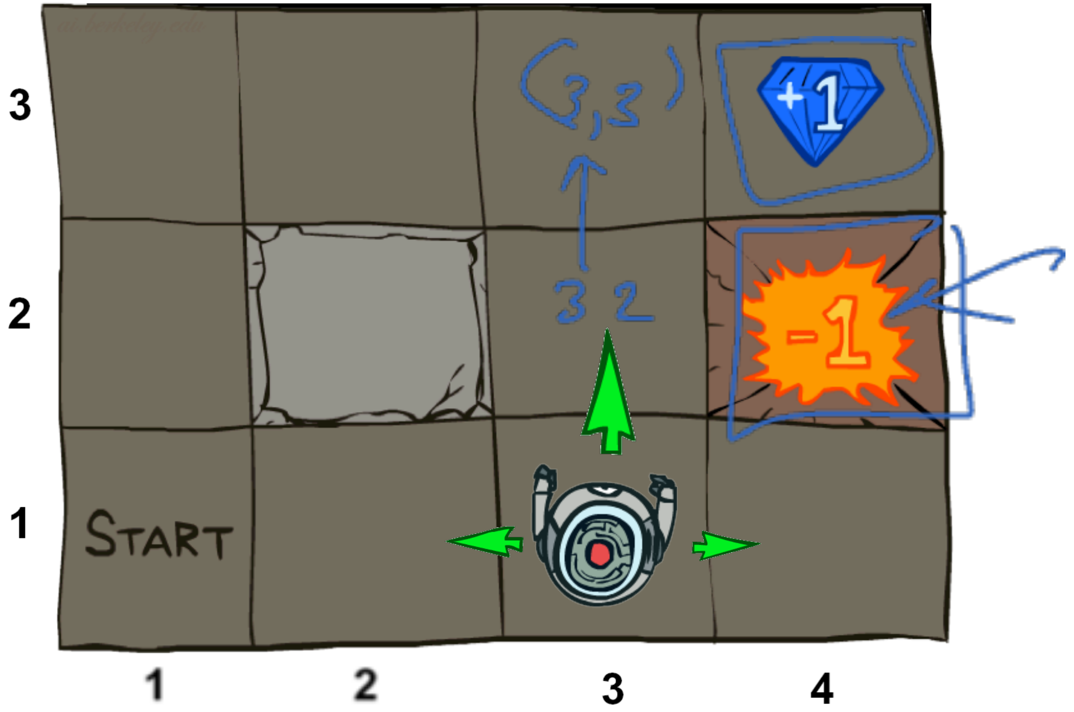



*T is a Big Table!*
*11 X 4 x 11 = 484 entries*

**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., $P(s' \mid s, a)$
    - Also called the model or the dynamics
  - A reward function $R(s, a, s')$ R(s, a |
    - Sometimes just $R(s)$ or $R(s')$




...

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

...

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

...

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

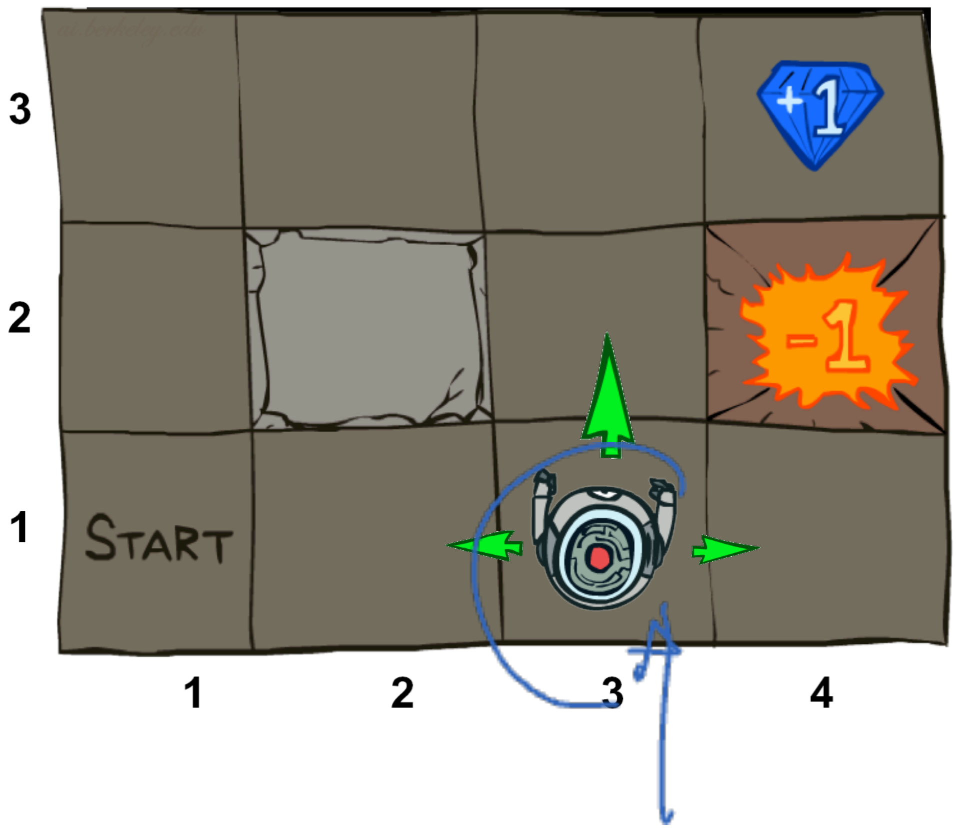
***Cost of breathing***

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 ∈ 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?

Past Future

- “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

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

$$=$$

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

$S_{t+1}$ $s_t$ $S_{t-1}$ $S_0$

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

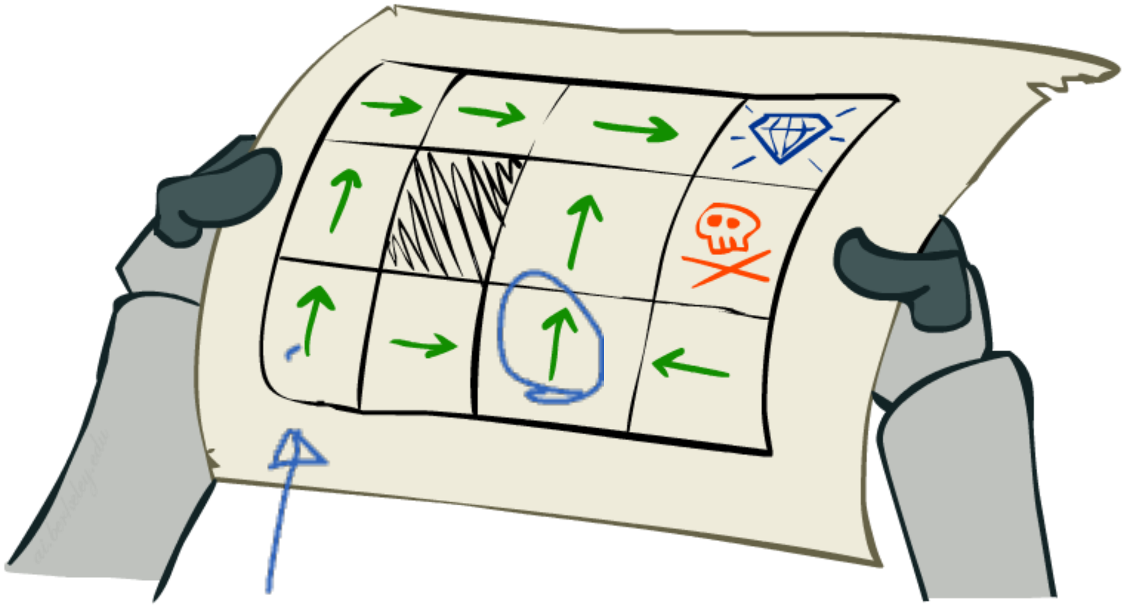
Andrey Markov
(1856-1922)

# Policies

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

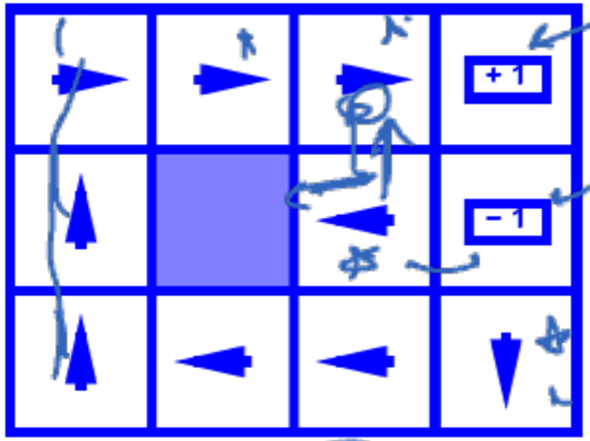
- For MDPs, we want an optimal

  policy $\pi^*: S \rightarrow A$
  - A policy $\pi$ gives an action for each state
  - An optimal policy is one that maximizes expected utility if followed
  - An explicit policy defines a reflex agent

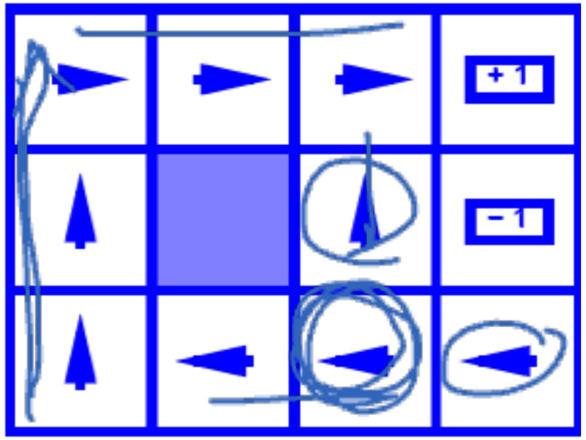


Optimal policy when R(s, a, s') = -0.4 for all non-terminals s

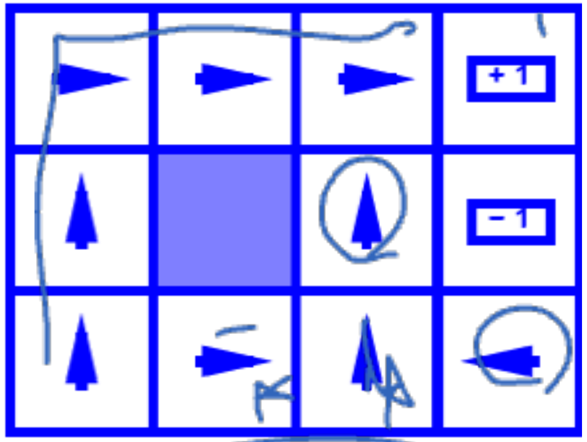
# Optimal Policies




R(s) = -0.01

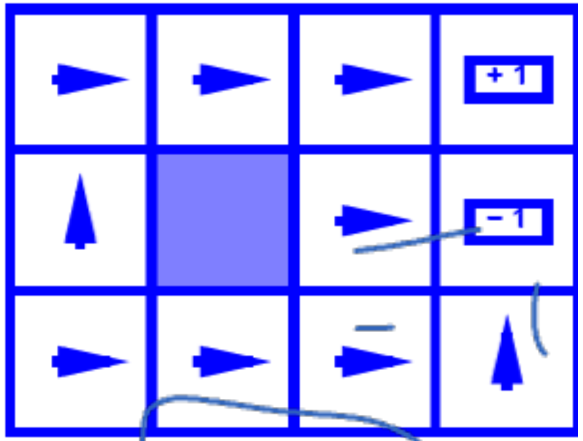



R(s) = -0.03



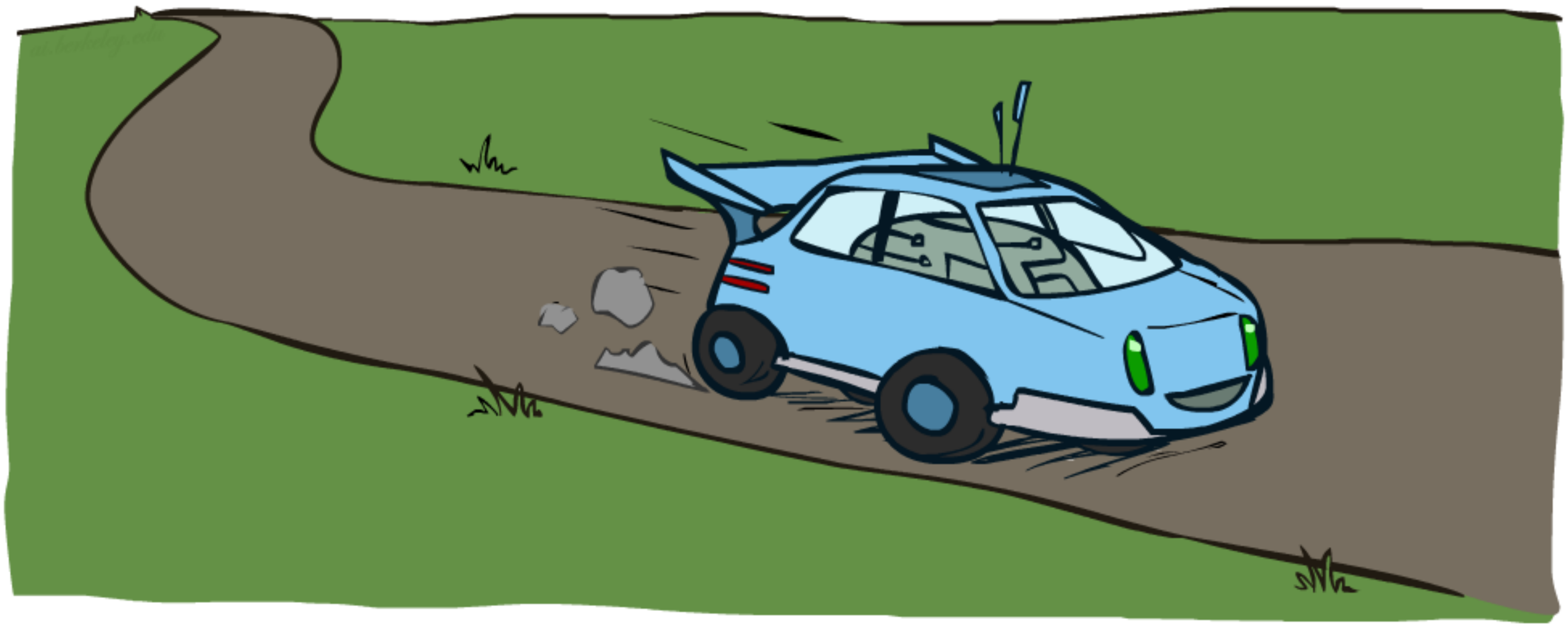

R(s) = -0.4

2




R(s) = -2.0

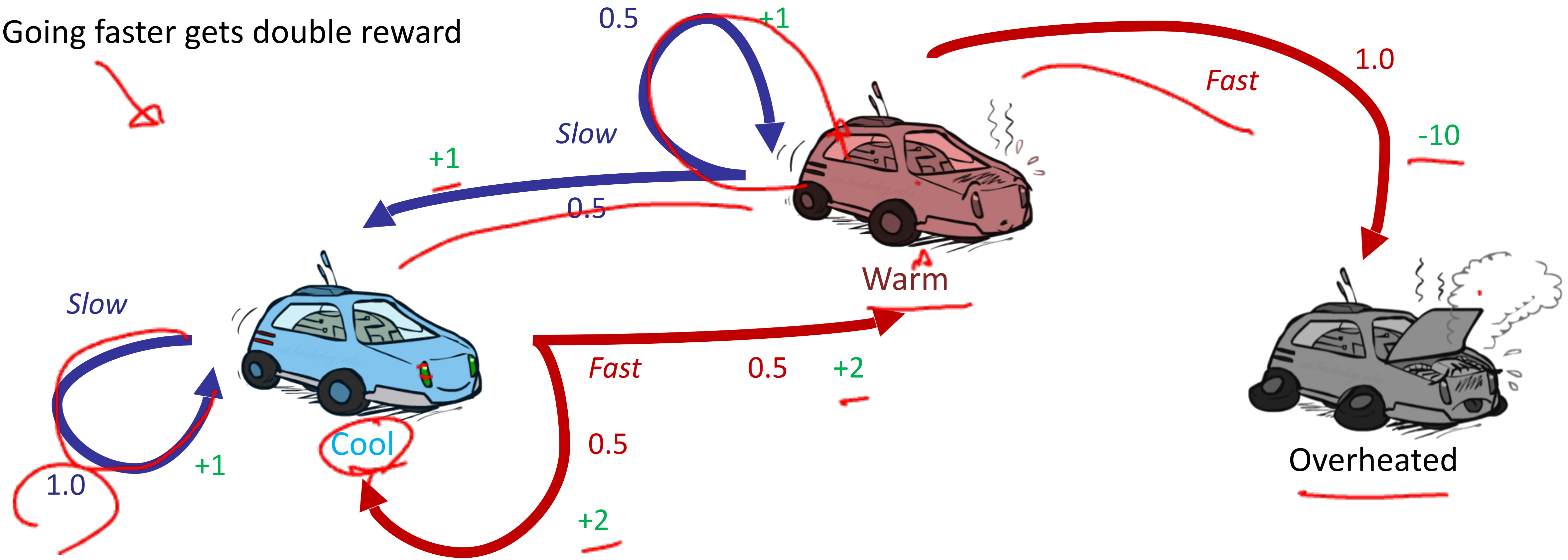
# Example: Racing



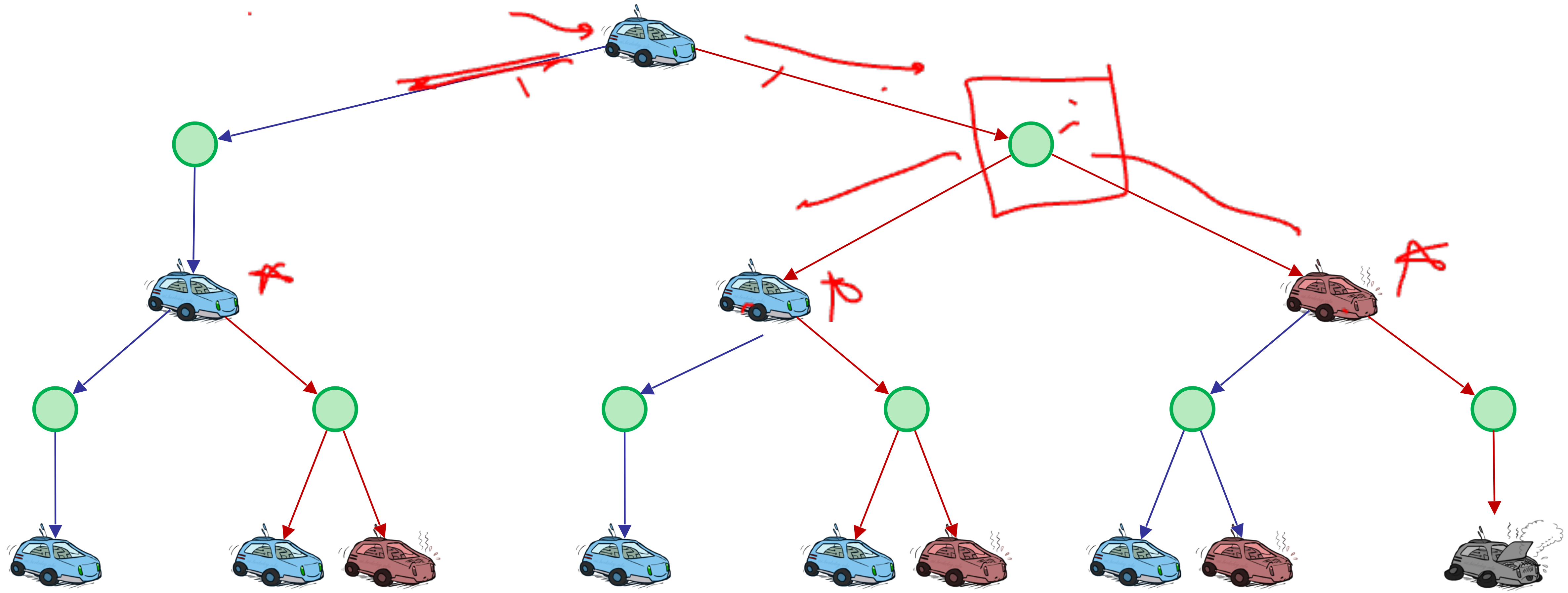
# Example: Racing

Cool

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

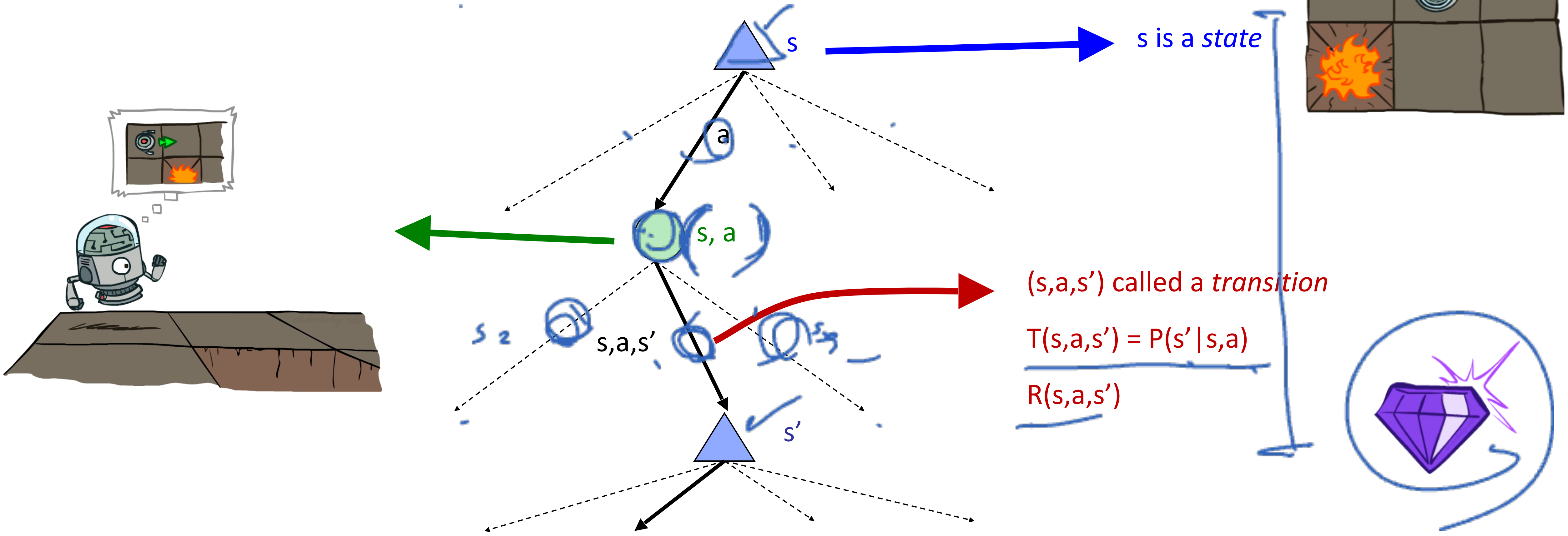


# Racing Search Tree

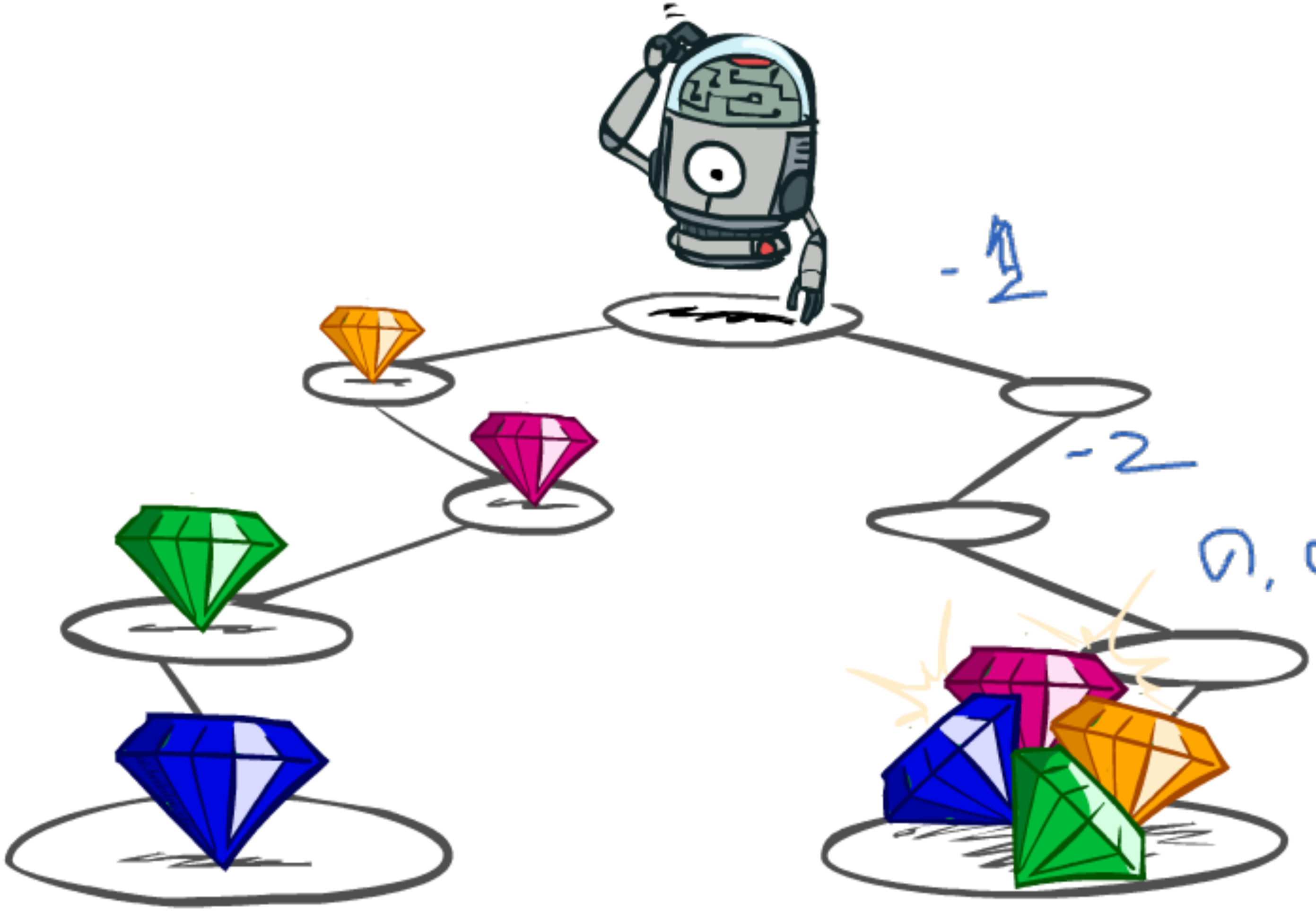


# MDP Search Trees

- Each MDP state projects an expectimax-like search tree



# Utilities of Sequences



# Utilities of Sequences

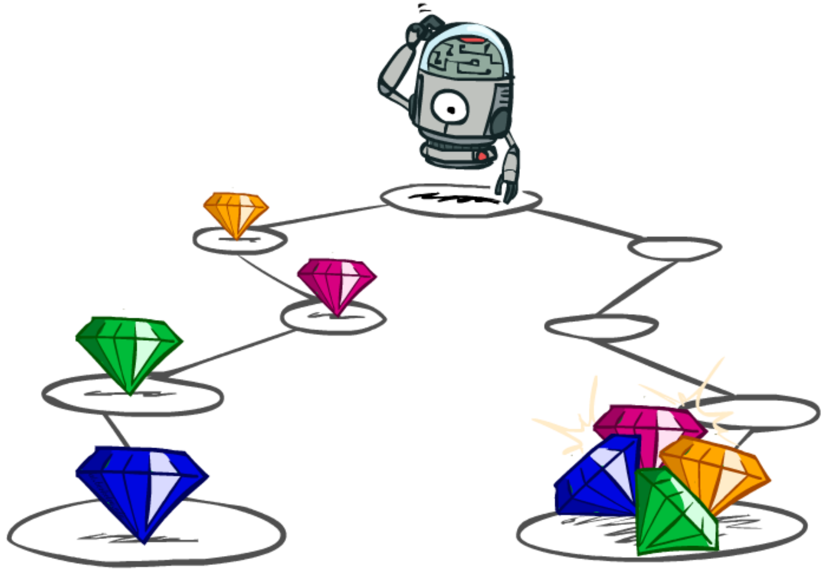
- What preferences should an agent have over reward sequences?

left    right    [-0.1, -0.1, 1]

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

left    right

- 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

$0 < \gamma \leq 1$

$\gamma^2$

Worth In Two Steps

$\gamma^2$

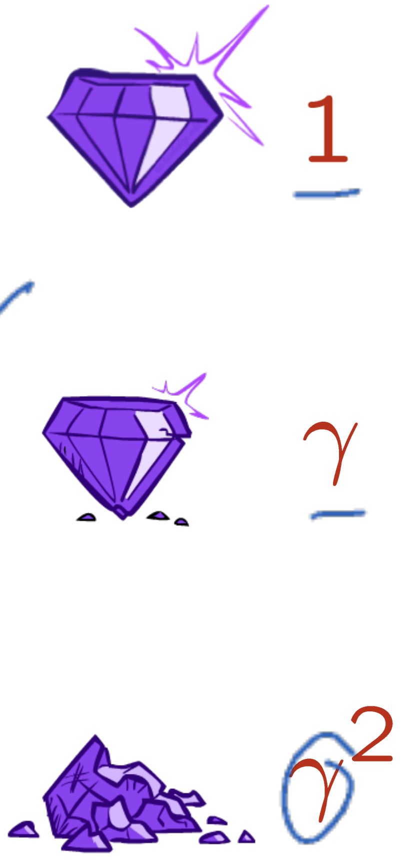
# Discounting

$\underbrace{\gamma^{n} \times r}$

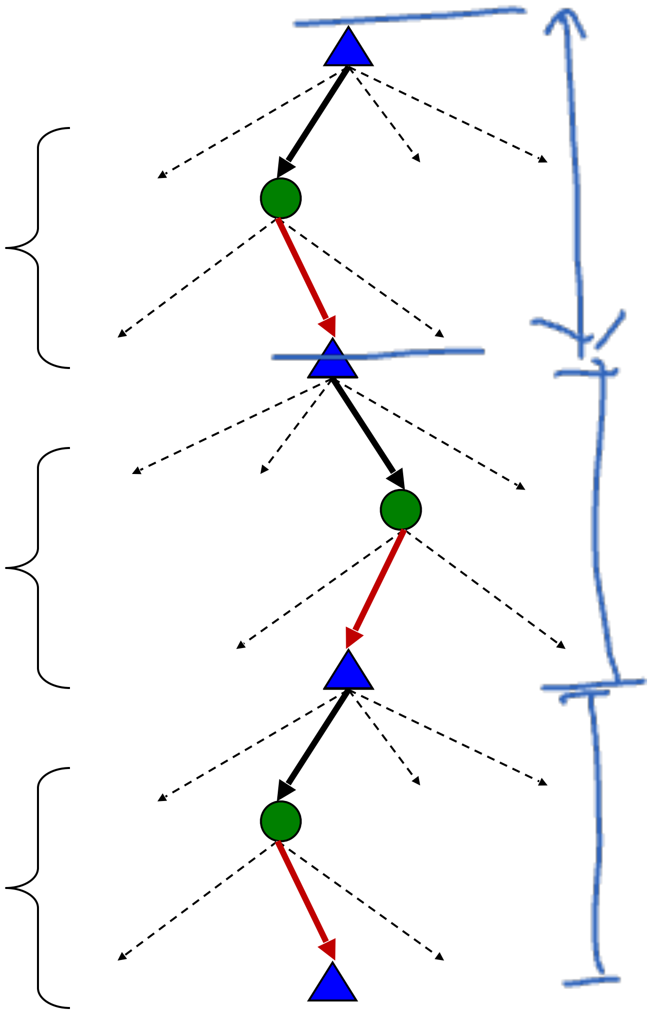
- How to discount?
  - Each time we descend a level, we multiply in the discount once
- Why discount?
  - Think of it as a gamma chance of ending the process at every step
  - Also helps our algorithms converge
- Example: discount of 0.5
  - U([1,2,3]) = 1*1 + 0.5*2 + 0.25*3
  - U([1,2,3]) < U([3,2,1])

$(0.5)^2 \times 3$

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




$\gamma^n$



# 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 |
|---|---|---|---|---|

$10 \times \gamma^3$

$10 \times \gamma^3$ $0 + \gamma * 1$

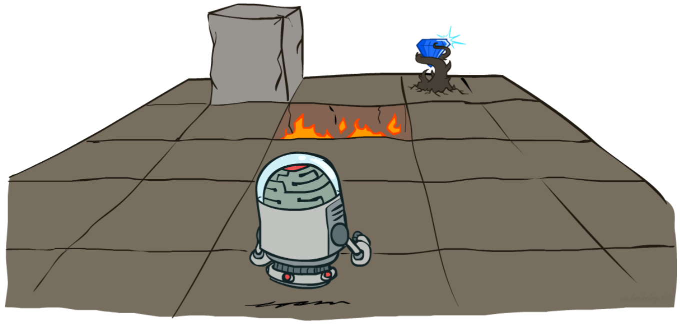
- Quiz 3: For which $\gamma$ are West and East equally good when in state d?

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

# CSE 473: Introduction to Artificial Intelligence

Hanna Hajishirzi

Markov Decision Processes

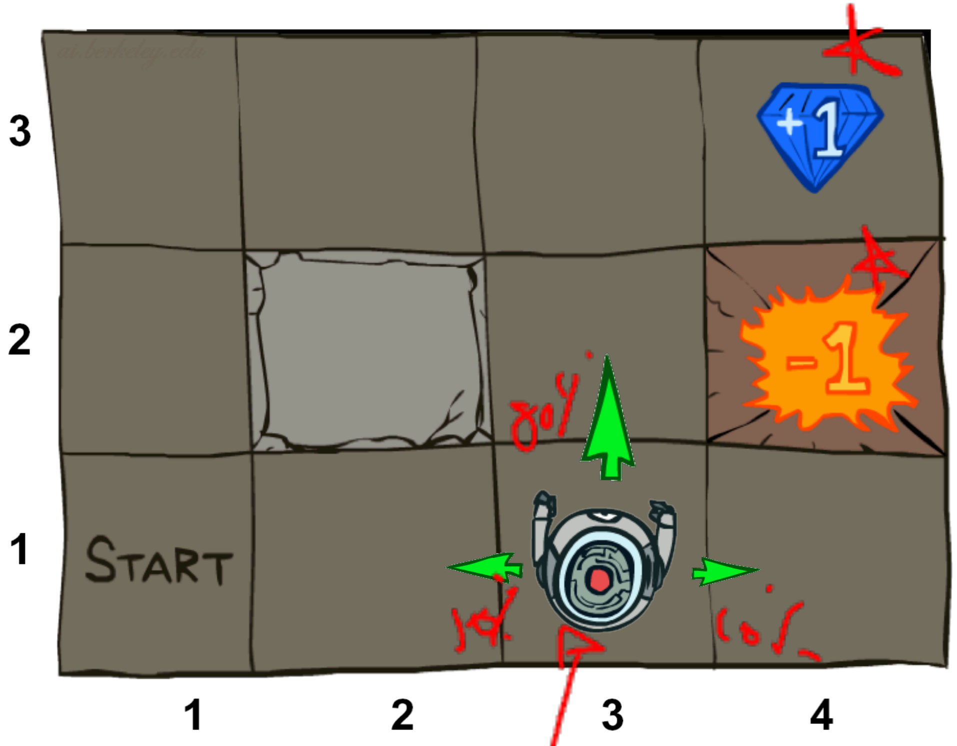


slides adapted from
Dan Klein, Pieter Abbeel ai.berkeley.edu
And Dan Weld, Luke Zettlemoyer

# 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., $P(s' \mid s, a)$
    - 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

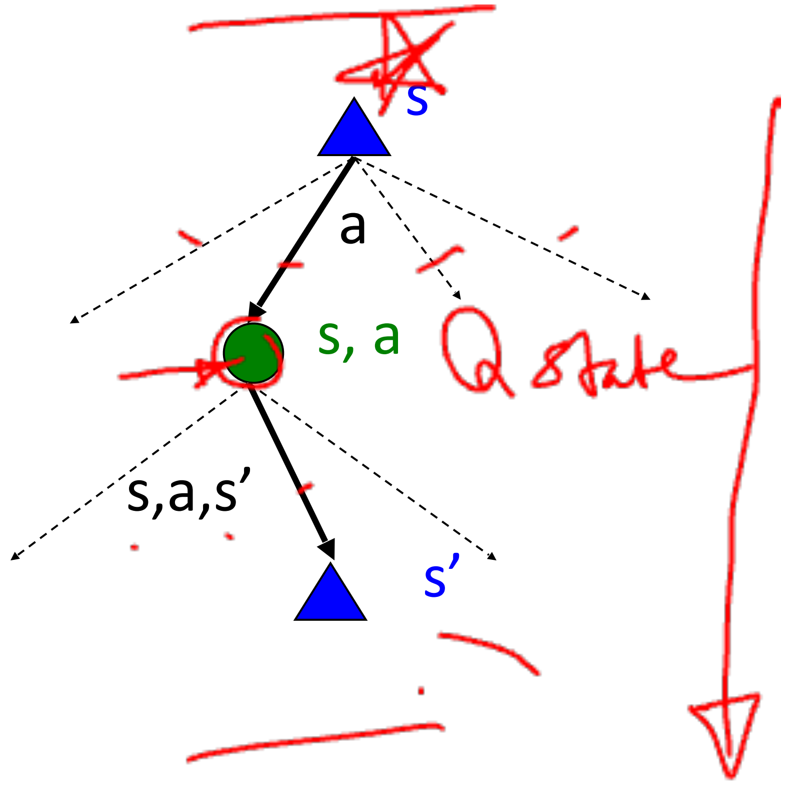



start

goal

# Recap: Defining MDPs

- Markov decision processes:
  - Set of states S
  - Start state $s_0$
  - Set of actions A
  - Transitions $P(s' \mid s,a)$ (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

$[1, 2, 3]$ $\qquad 1 + \gamma \times 2 + \gamma^2 \times 3$

# Infinite Utilities?!

S a

[1, 2, 3, . . .

+ + +

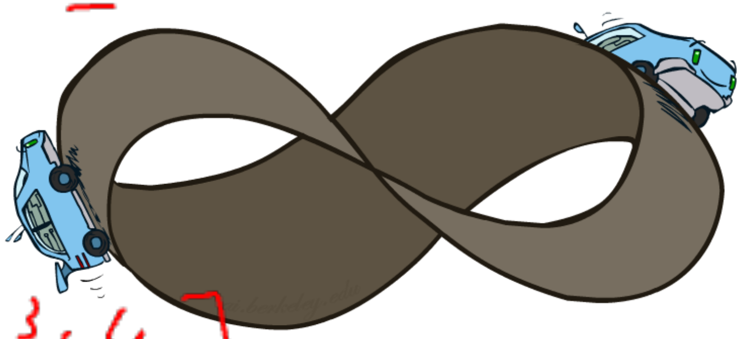
- Problem: What if the game lasts forever? Do we get infinite rewards?

- 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 $0 < \gamma < 1$

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

    - Sm
  - Absorbing state: guarantee that for every policy, a terminal state will eventually be reached (like "overheated" for racing)

$\gamma$

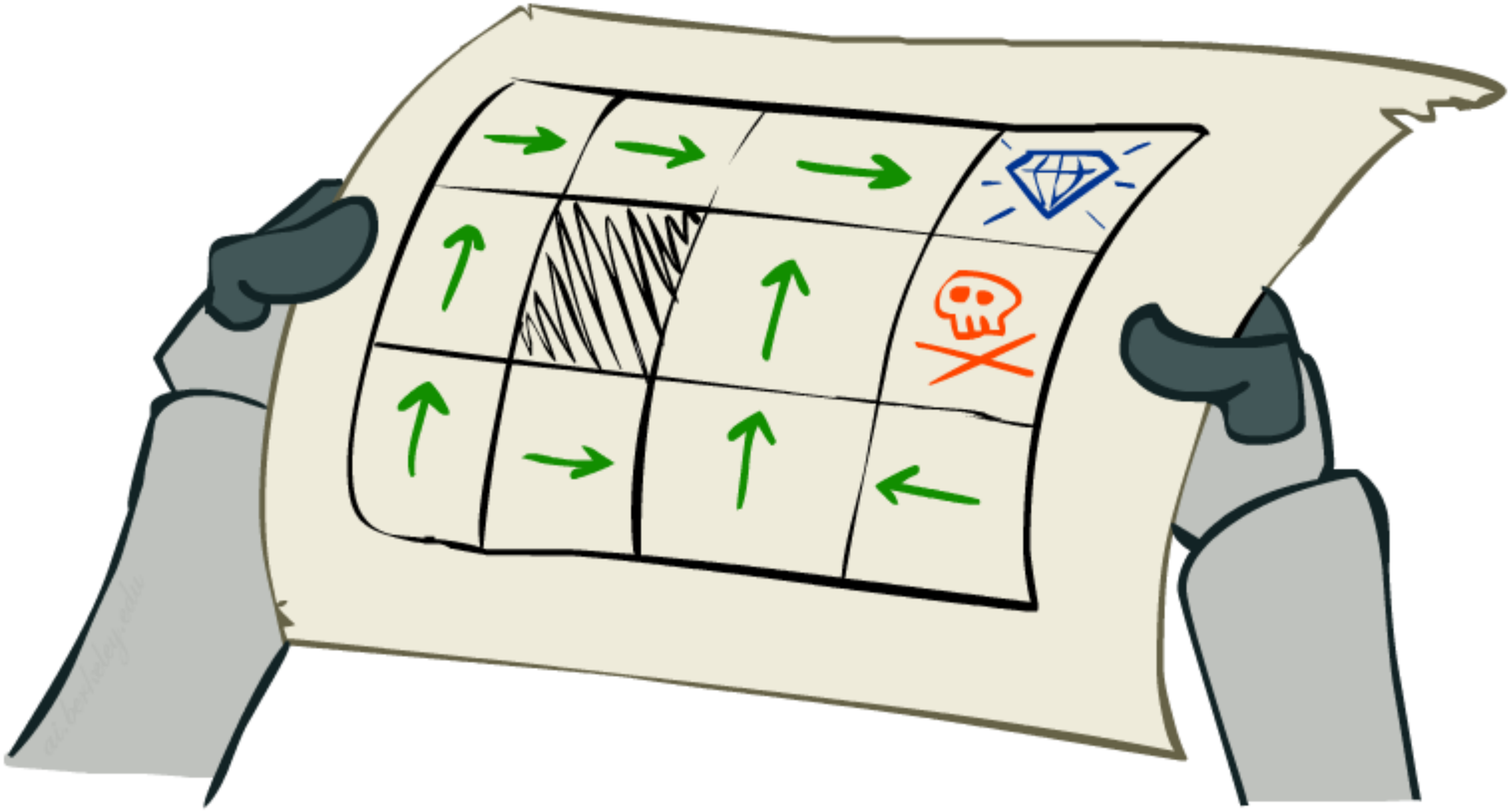


[1, 2, 3, 4]

$r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots$

$R_{max}[1+\gamma+\gamma^2+\cdots$

# Solving MDPs

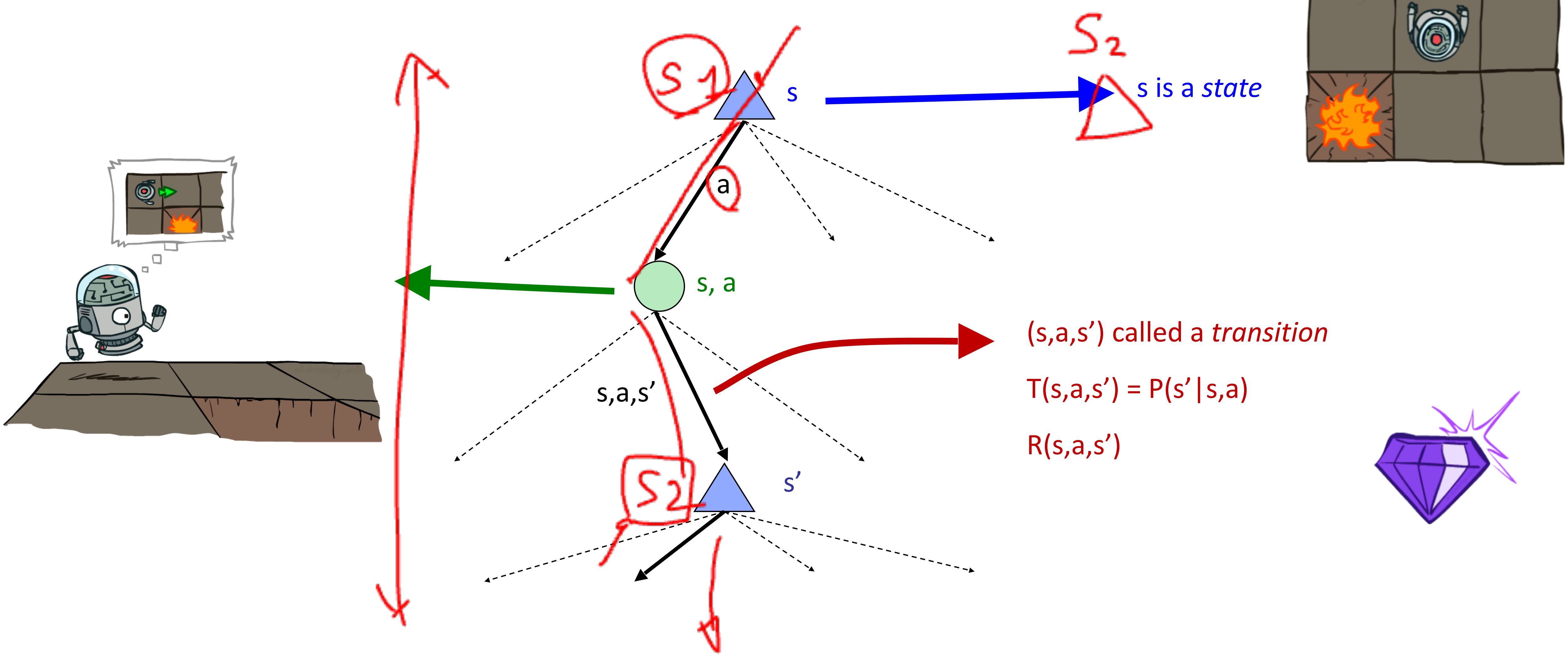


# MDP Search Trees

all states ?

S1 S2 S3

- Each MDP state projects an expectimax-like search tree



# Optimal Quantities

- The value (utility) of a state s:

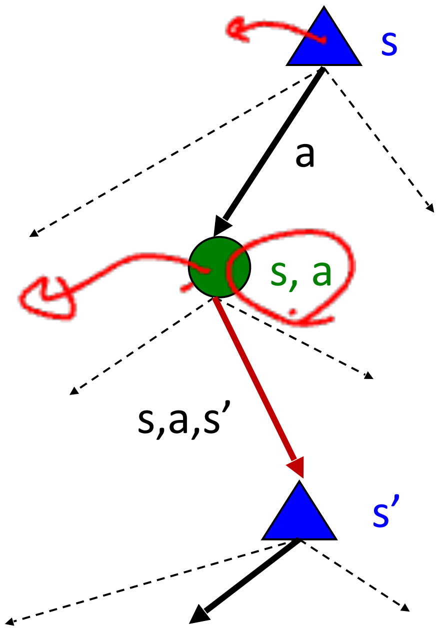
  $V^*(s)$ = expected utility starting in s and acting optimally

- 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



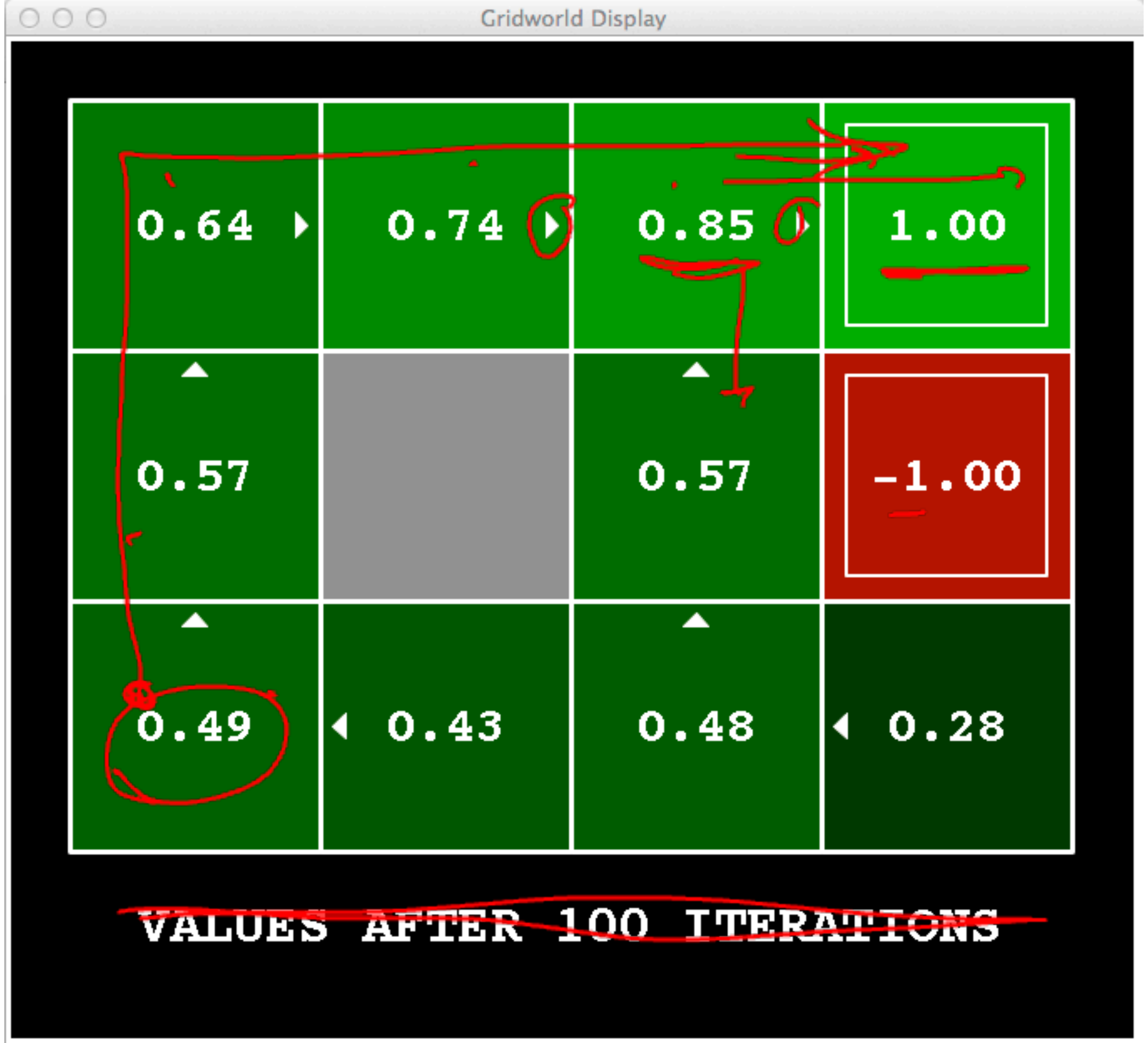

s is a *state*

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

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

# Snapshot of Demo – Gridworld V Values

$V^*$




0.8

.1

.1

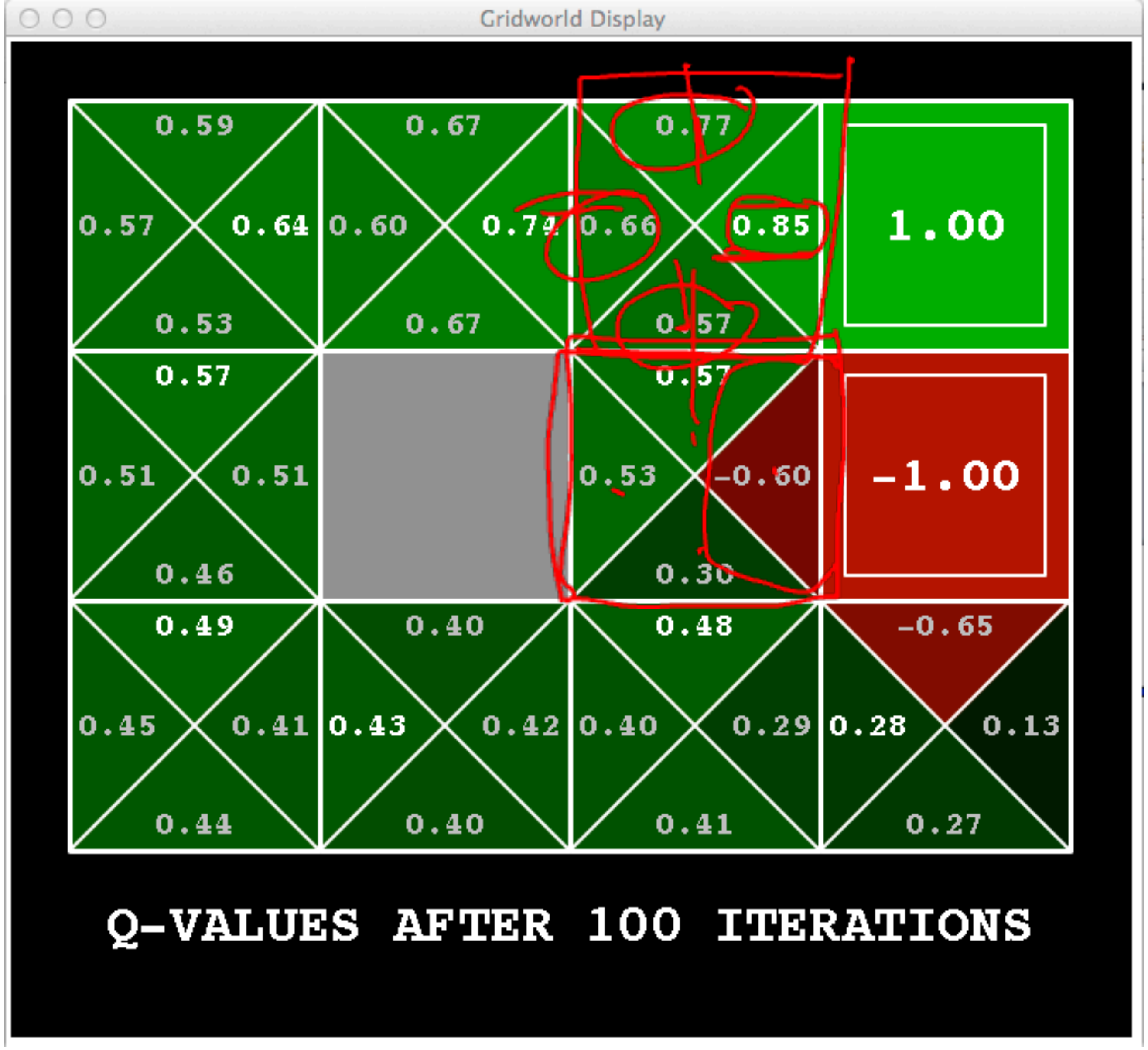
Noise = 0.2

Discount = 0.9 $\gamma$

Living reward = 0

# Snapshot of Demo – Gridworld Q Values

$Q^*(s,a)$



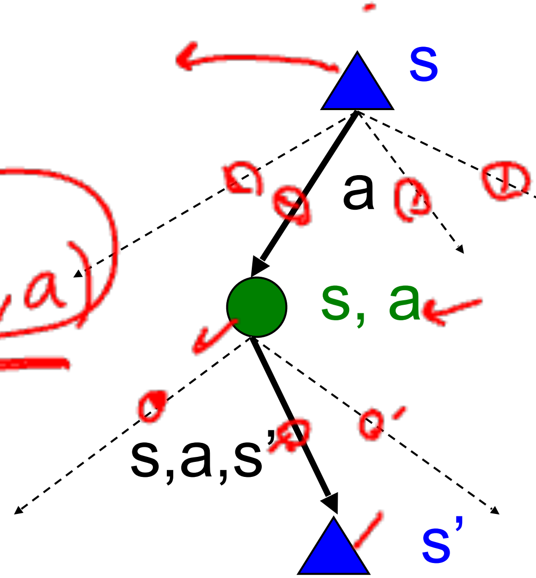

Noise = 0.2
Discount = 0.9
Living reward = 0

# Values of States (Bellman Equations)

$V^*$, $Q^*$ values $\pi^*$ action

- 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!

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




- Recursive definition of value:

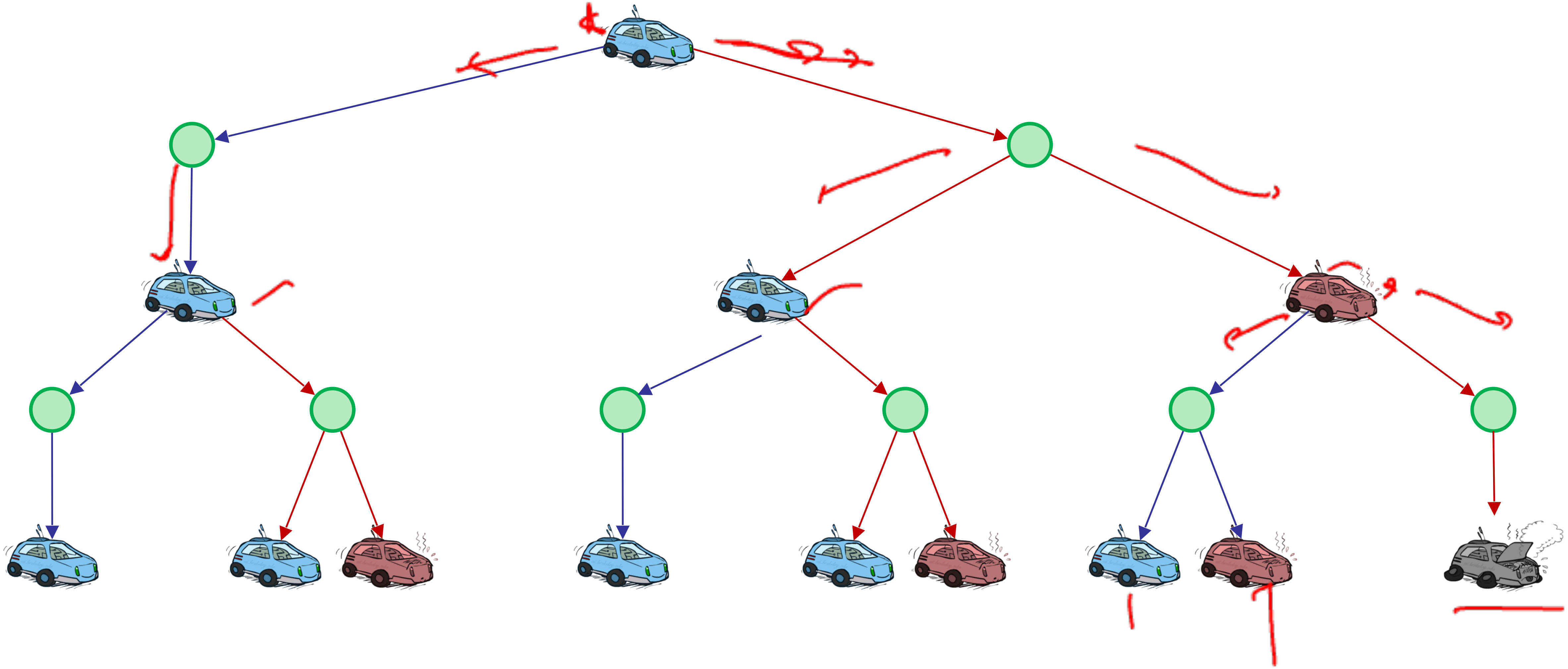
$$V^*(s) = \max_a Q^*(s,a)$$

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

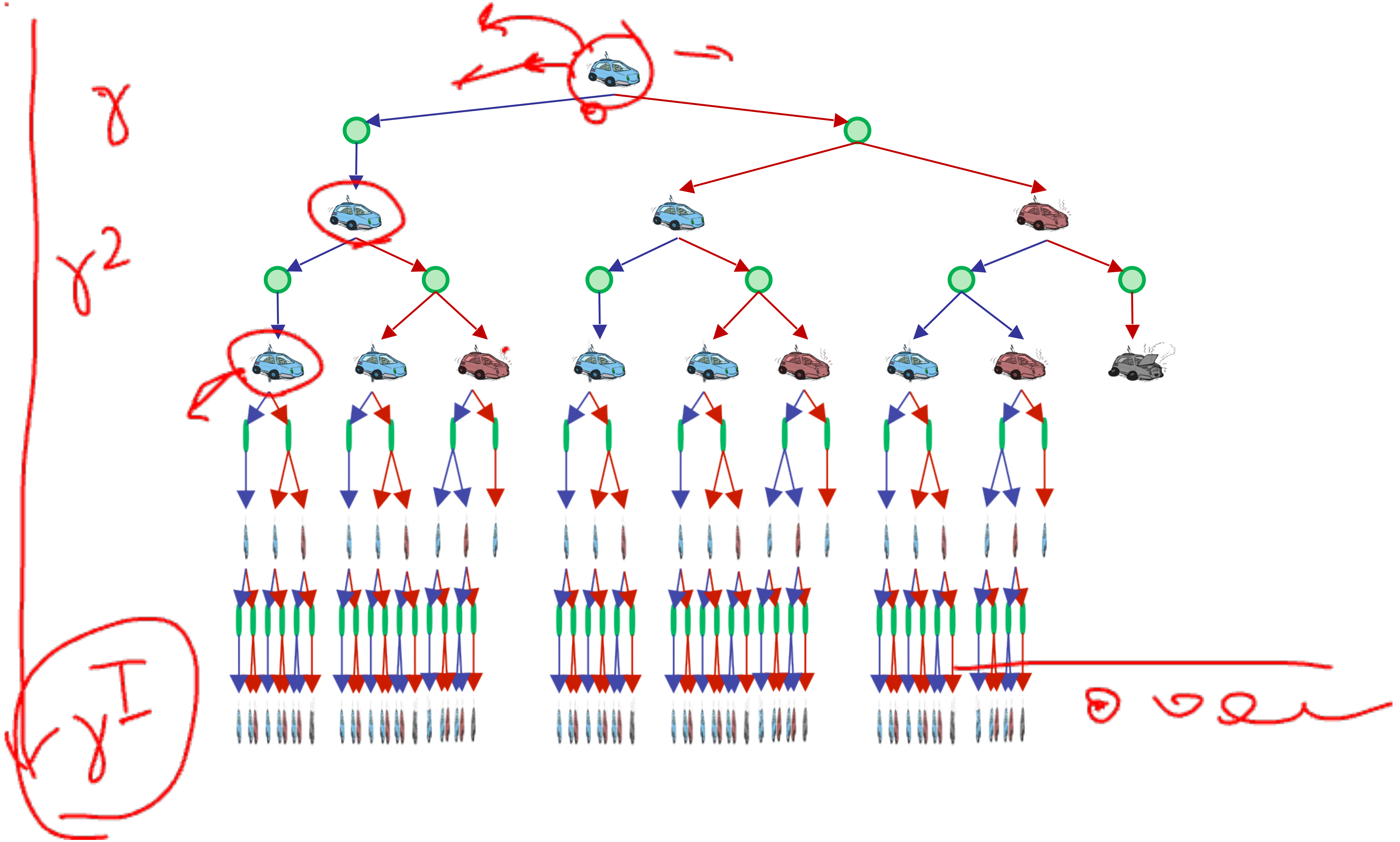
$$V^*(s) = \max_a \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V^*(s')\right]$$

# Racing Search Tree

Cool Warm overh

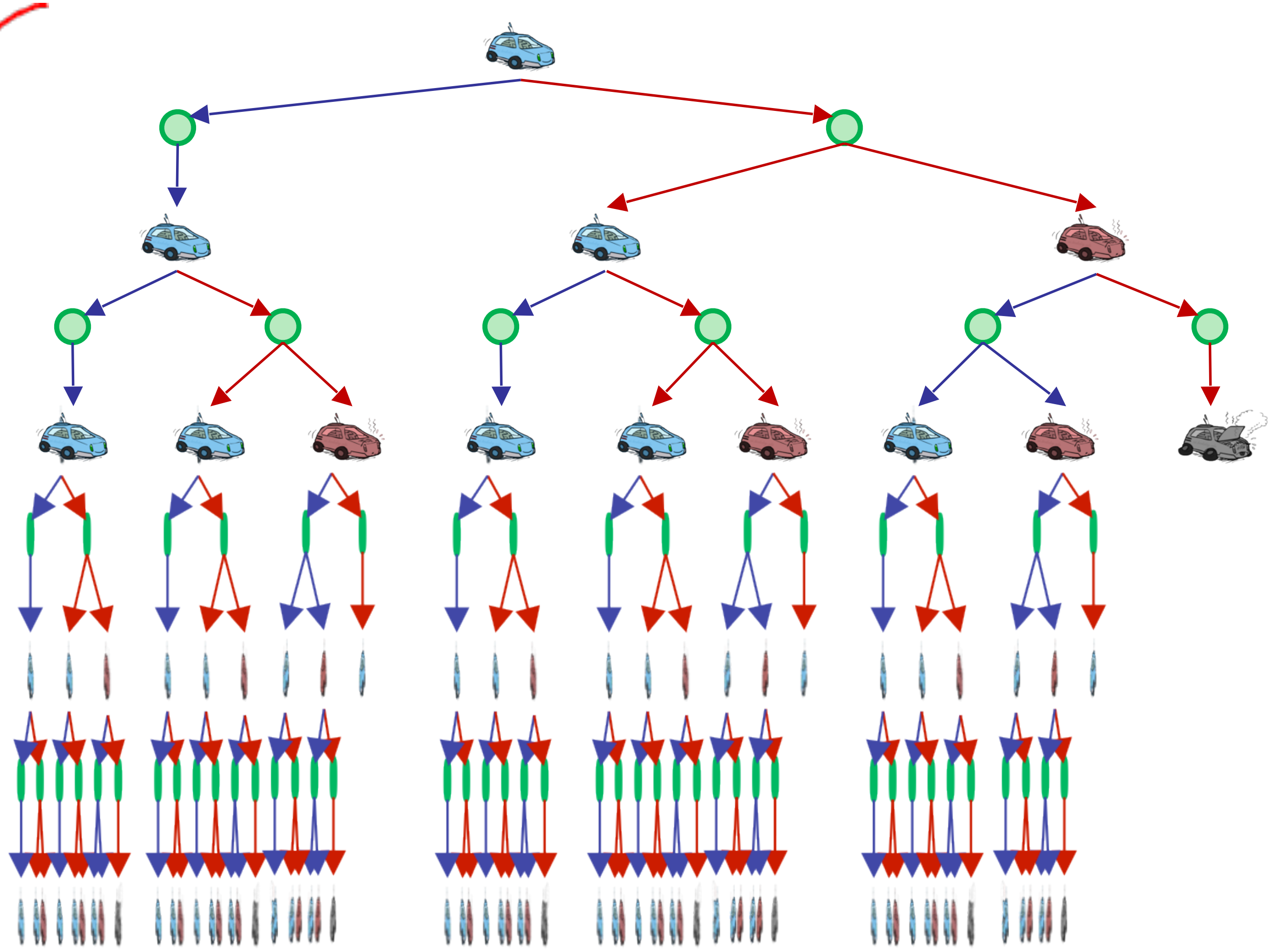


# Racing Search Tree



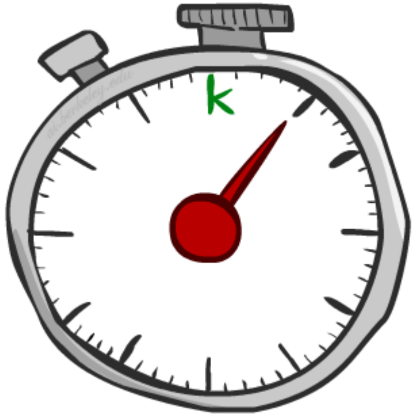
# Racing Search Tree

- 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 eventually don't matter if $\gamma < 1$

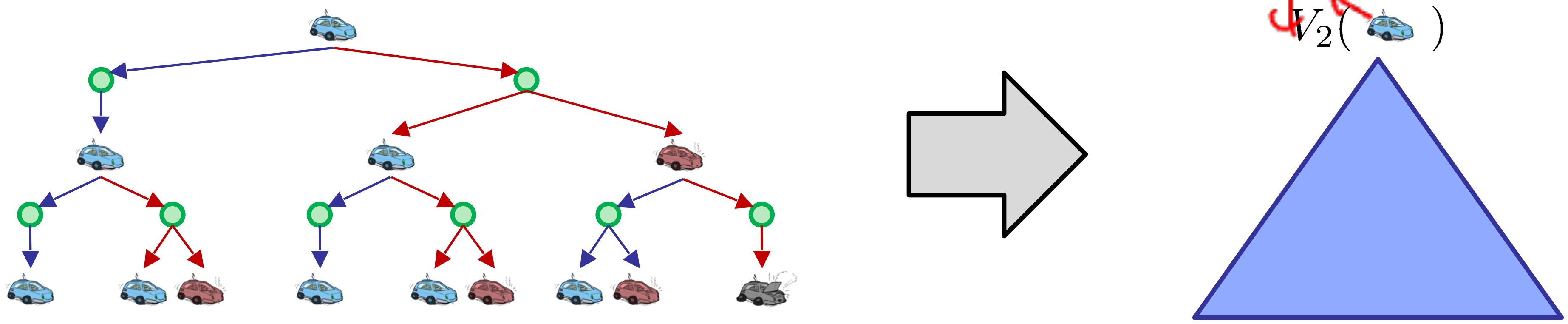


# Time-Limited Values

- 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

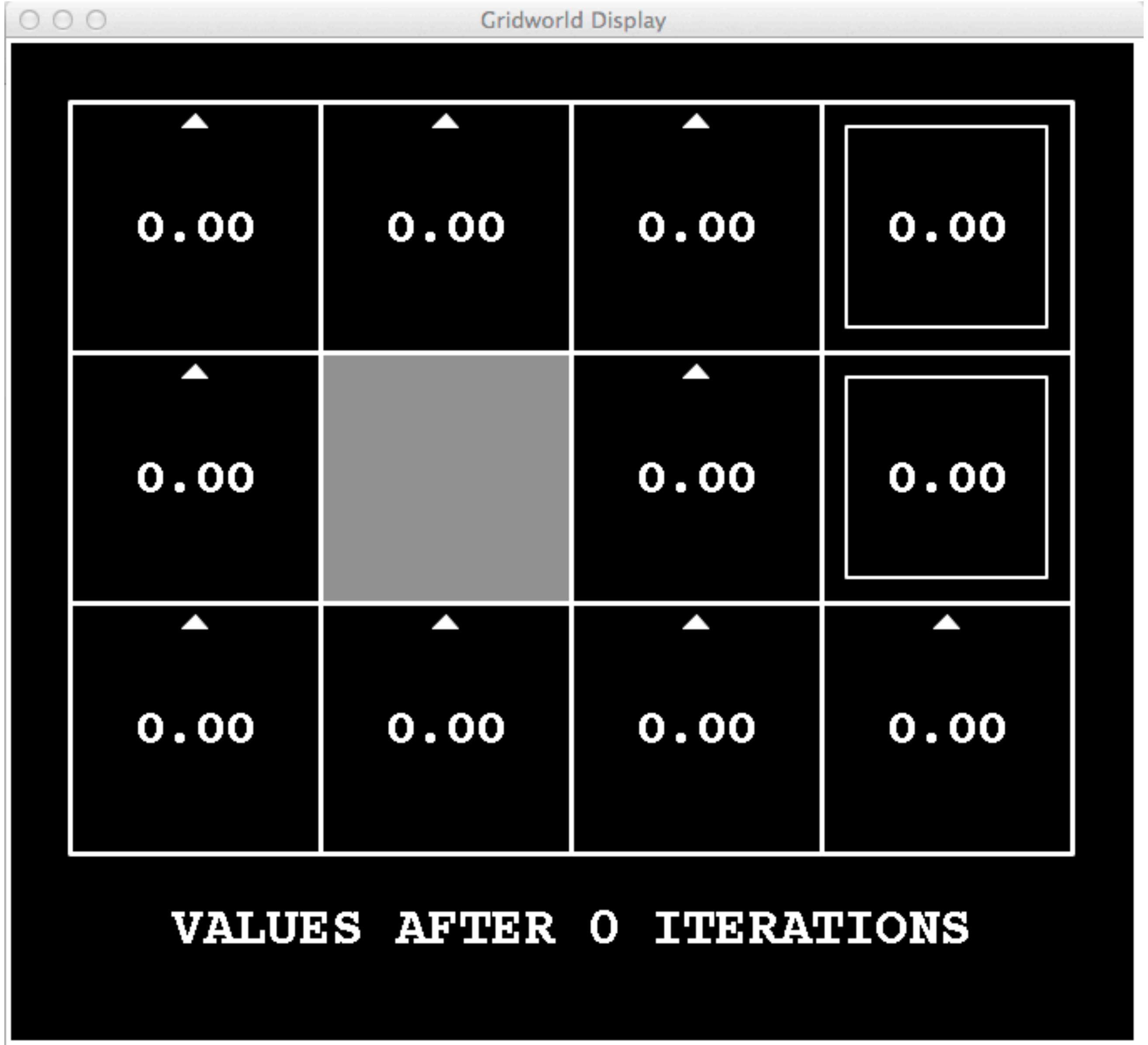



$V_2$




[Demo – time-limited values (L8D6)]

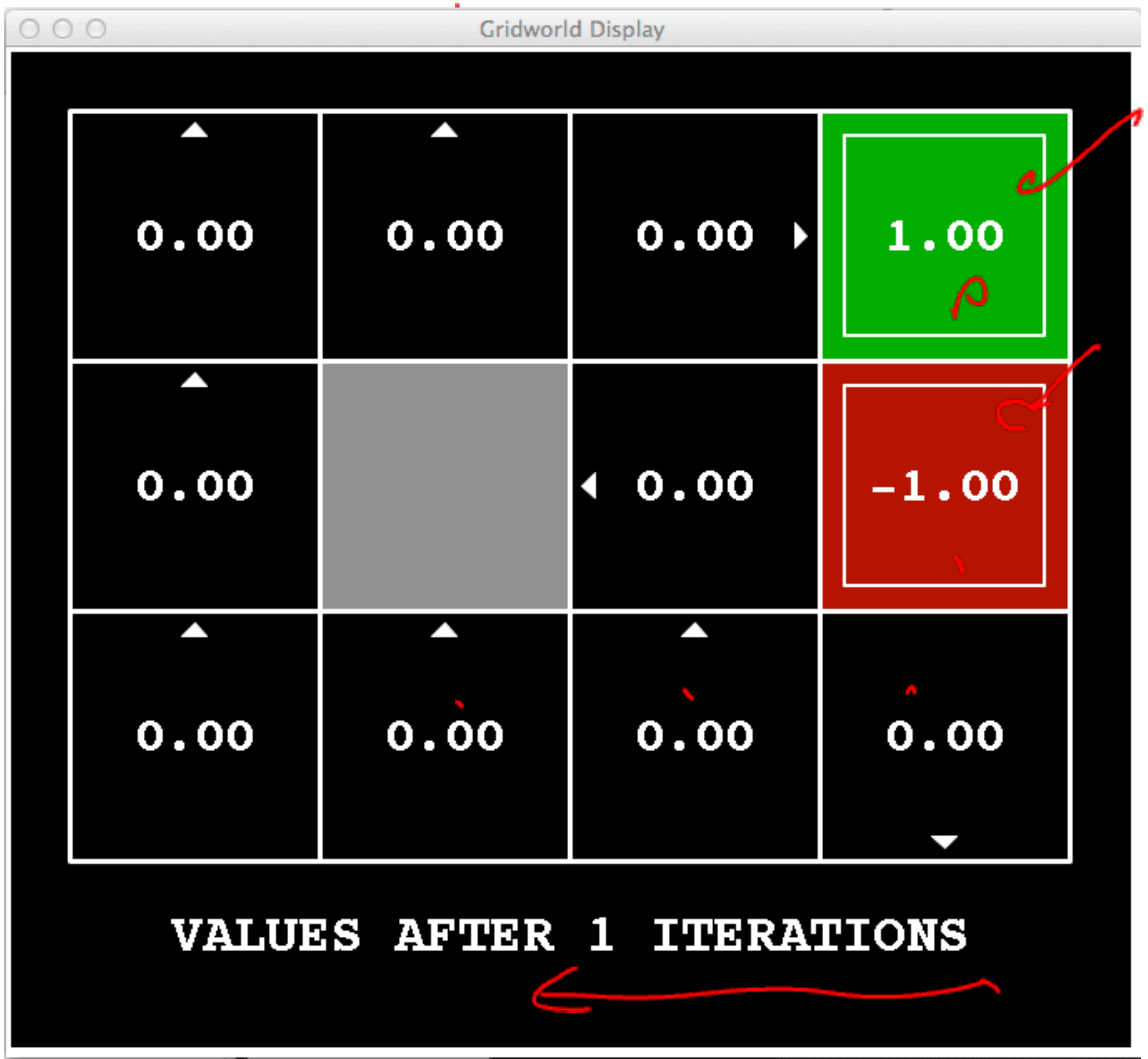
# k=0




$V_0$

Noise = 0.2
Discount = 0.9
Living reward = 0

# k=1

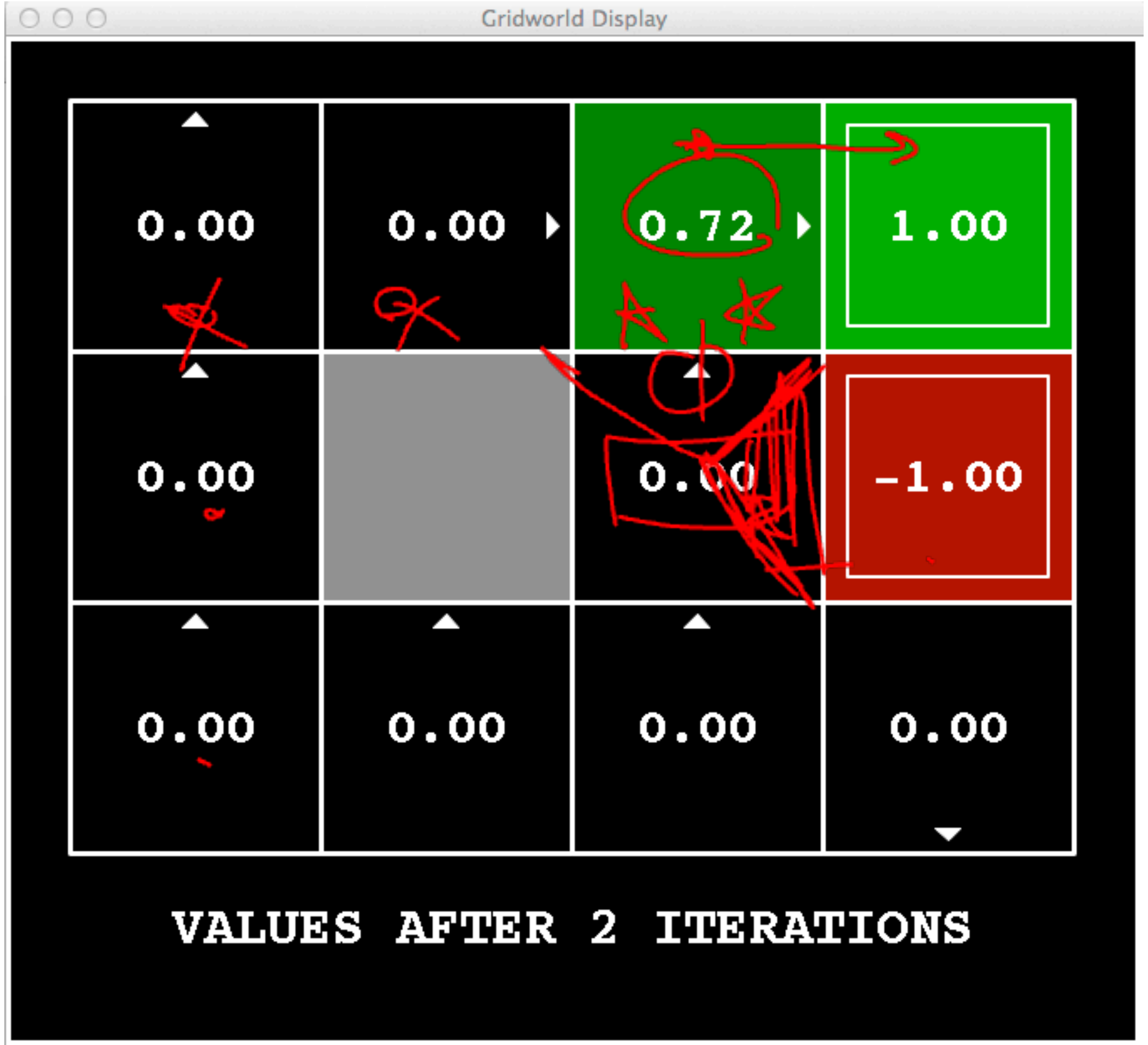



$V_1(s)$

Noise = 0.2
Discount = 0.9
Living reward = 0

# k=2

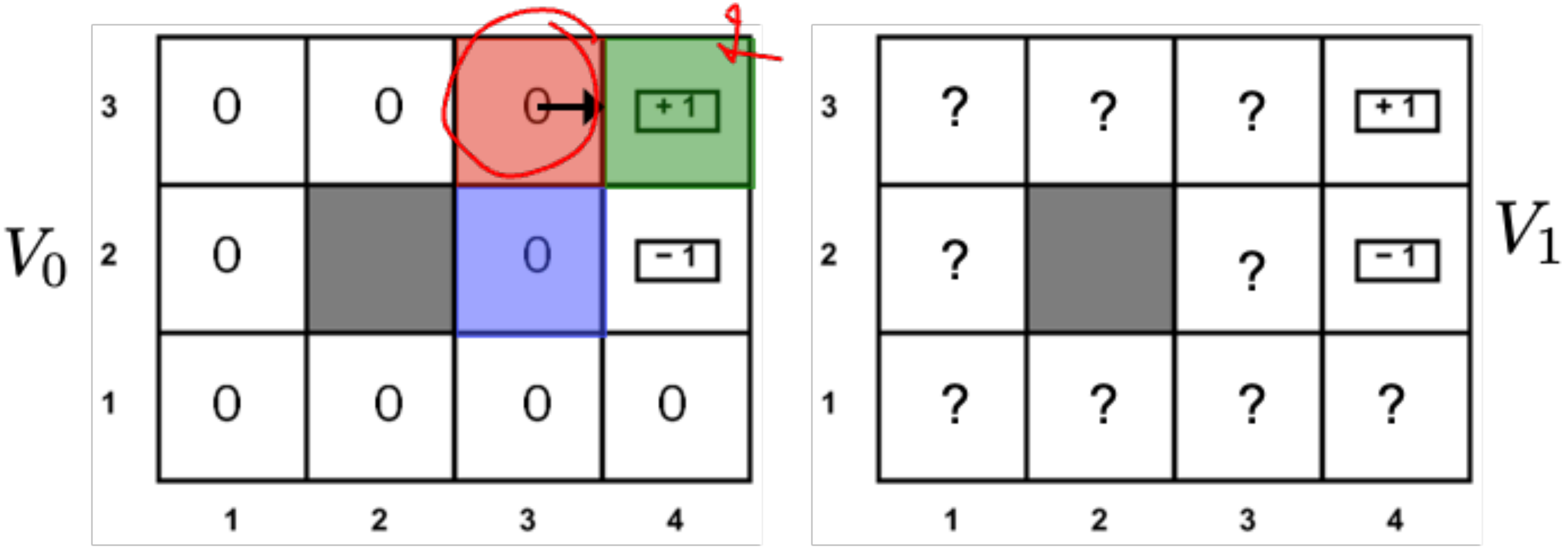
$V_2(S)$




Noise = 0.2
Discount = 0.9
Living reward = 0

# Bellman Updates

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




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

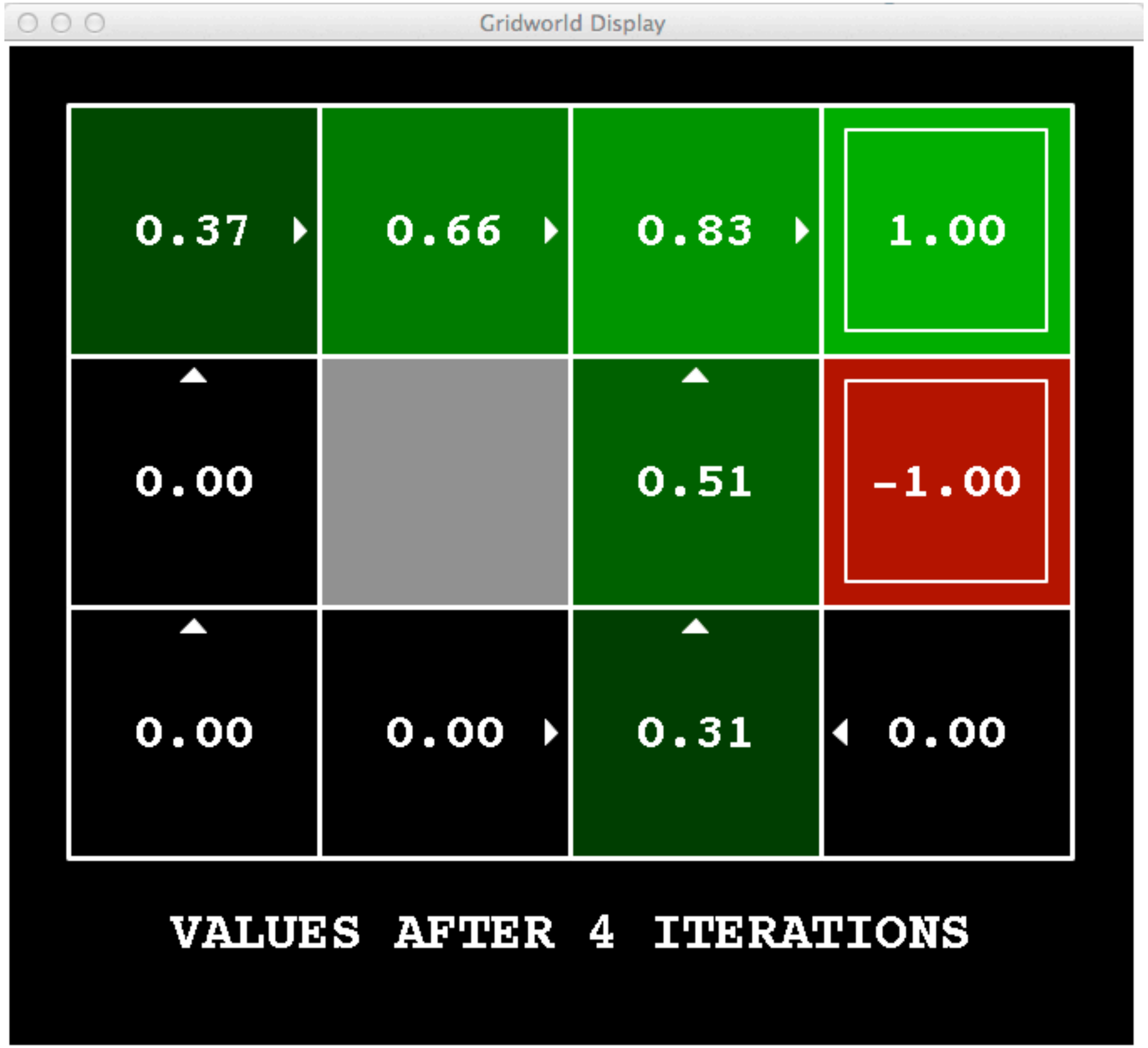
$$Q_1(\langle 3,3 \rangle, \text{right}) = \sum_{s'} T(\langle 3,3 \rangle, \text{right}, s') \left[ R(\langle 3,3 \rangle, \text{right}, s') + \gamma V_i(s') \right]$$

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

# k=3

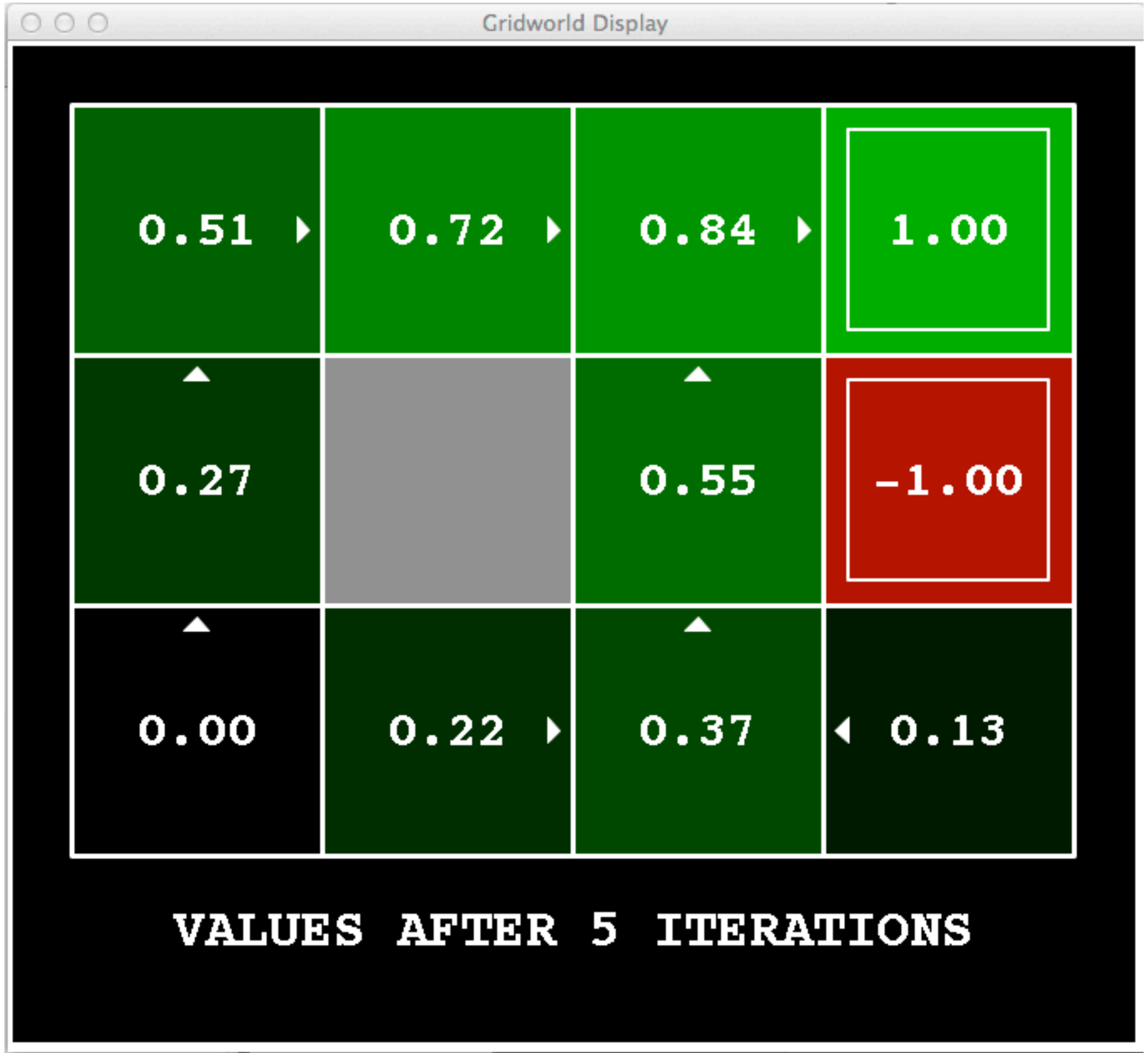

Noise = 0.2
Discount = 0.9
Living reward = 0

# k=4



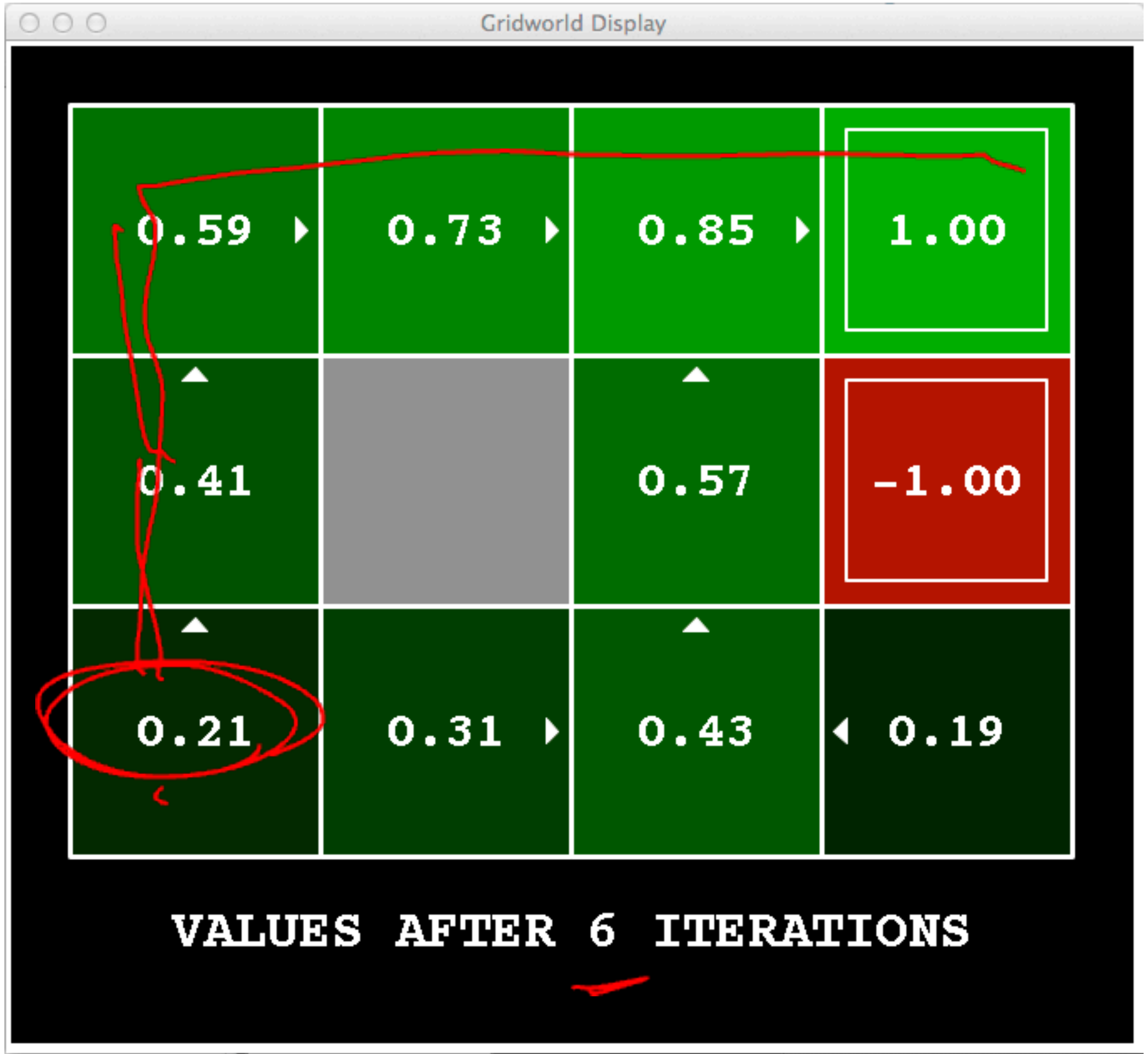

Noise = 0.2
Discount = 0.9
Living reward = 0

# k=5



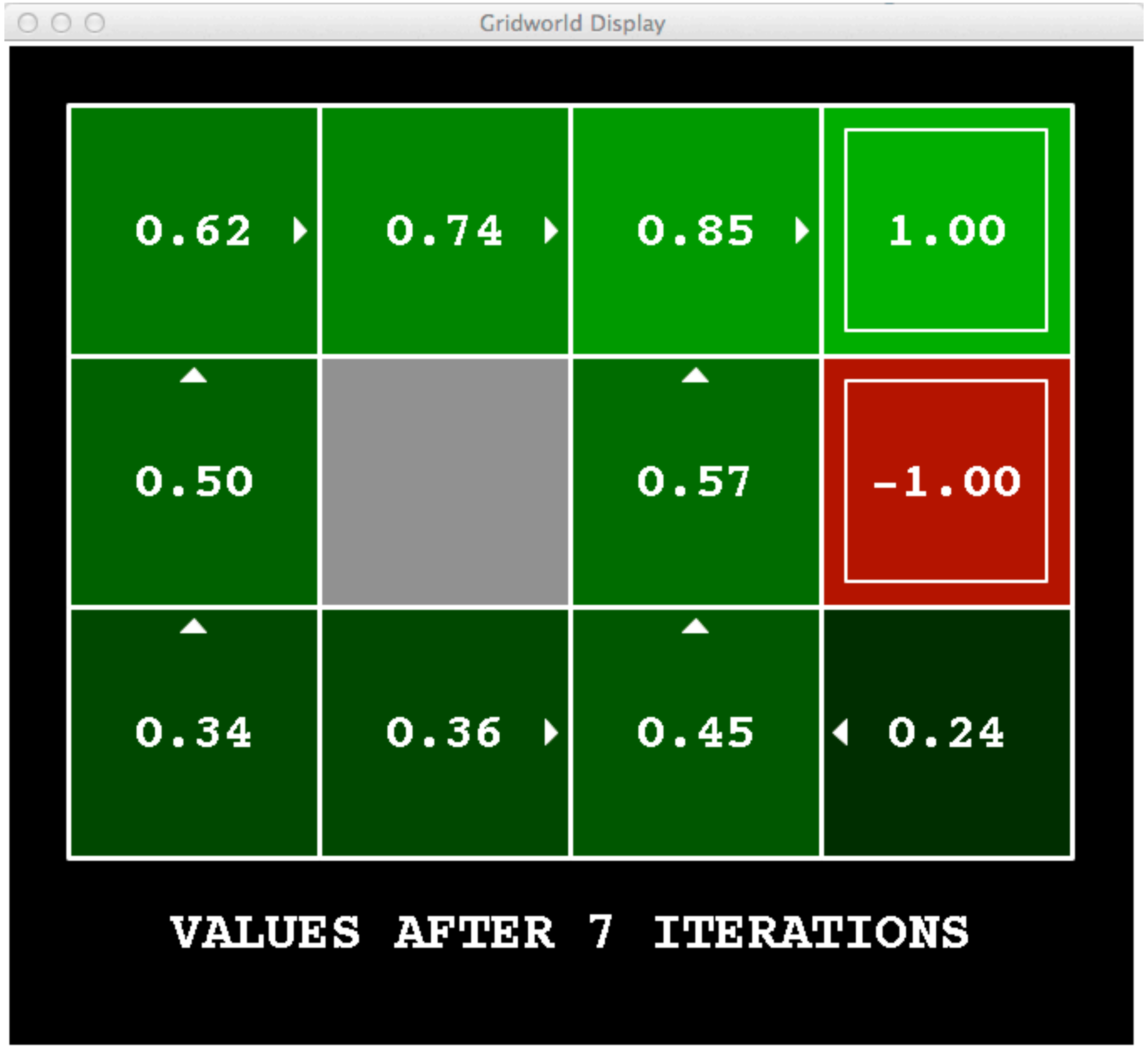

Noise = 0.2
Discount = 0.9
Living reward = 0

# k=6



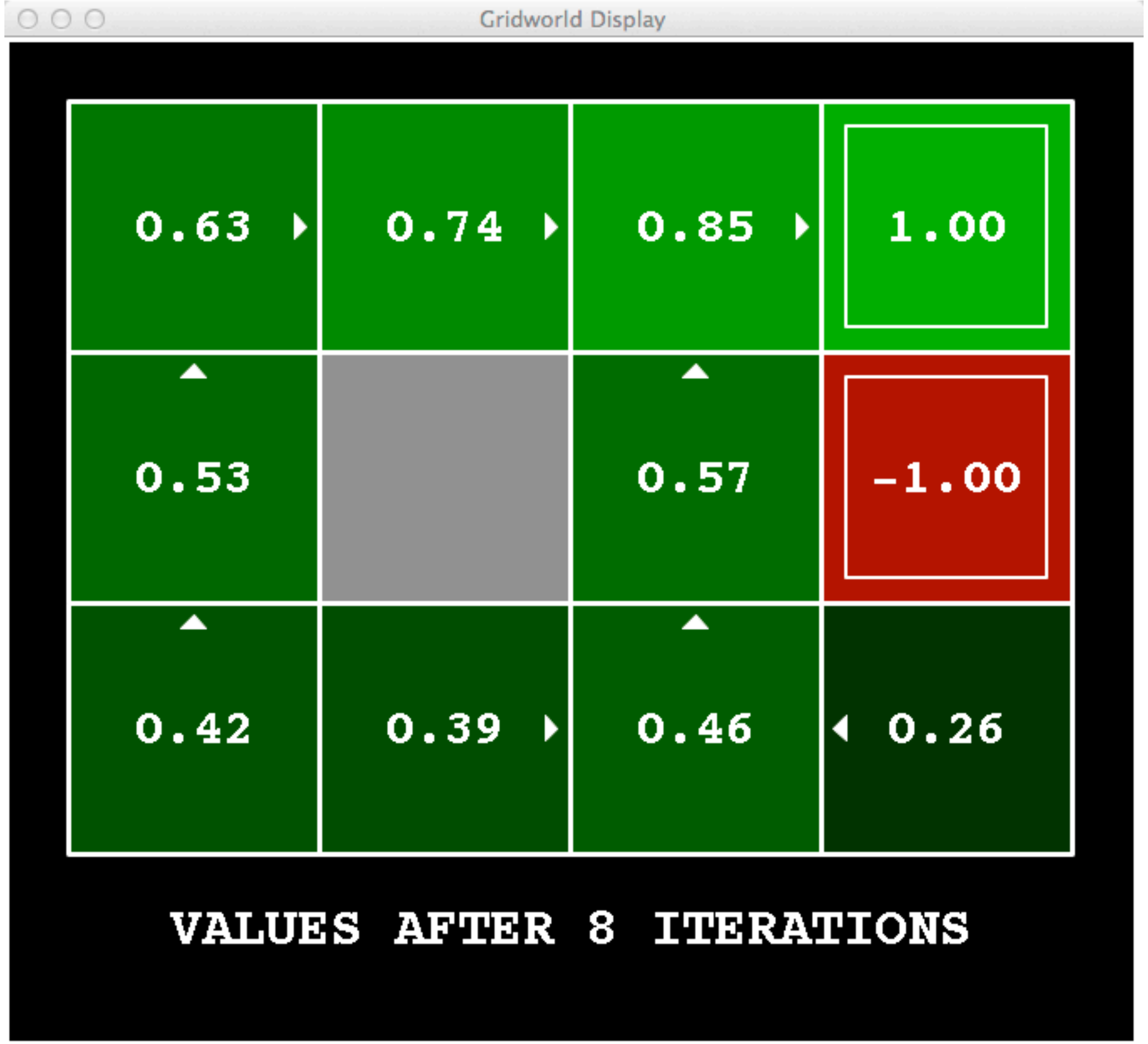

Noise = 0.2
Discount = 0.9
Living reward = 0

# k=7



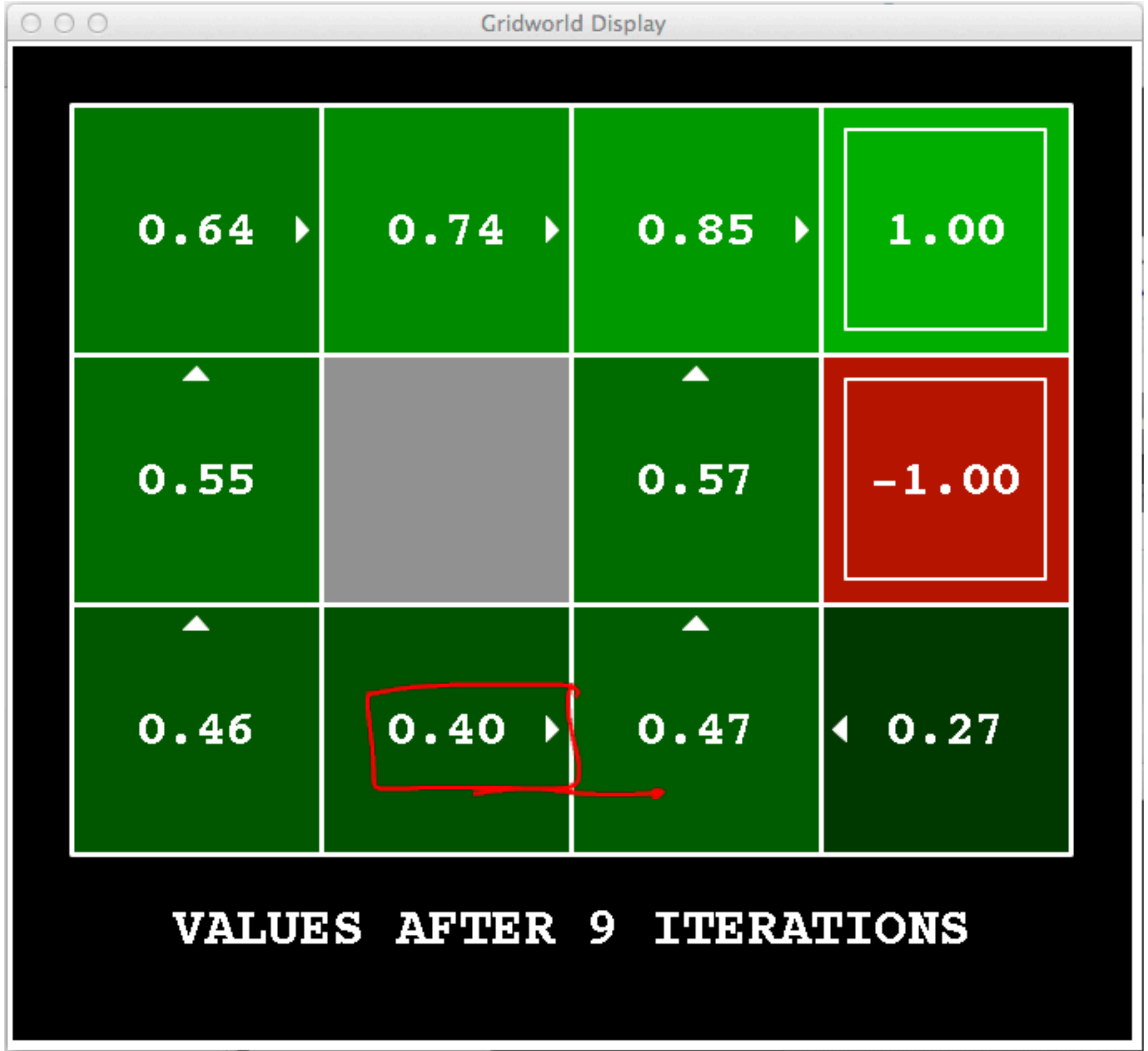

Noise = 0.2
Discount = 0.9
Living reward = 0

# k=8



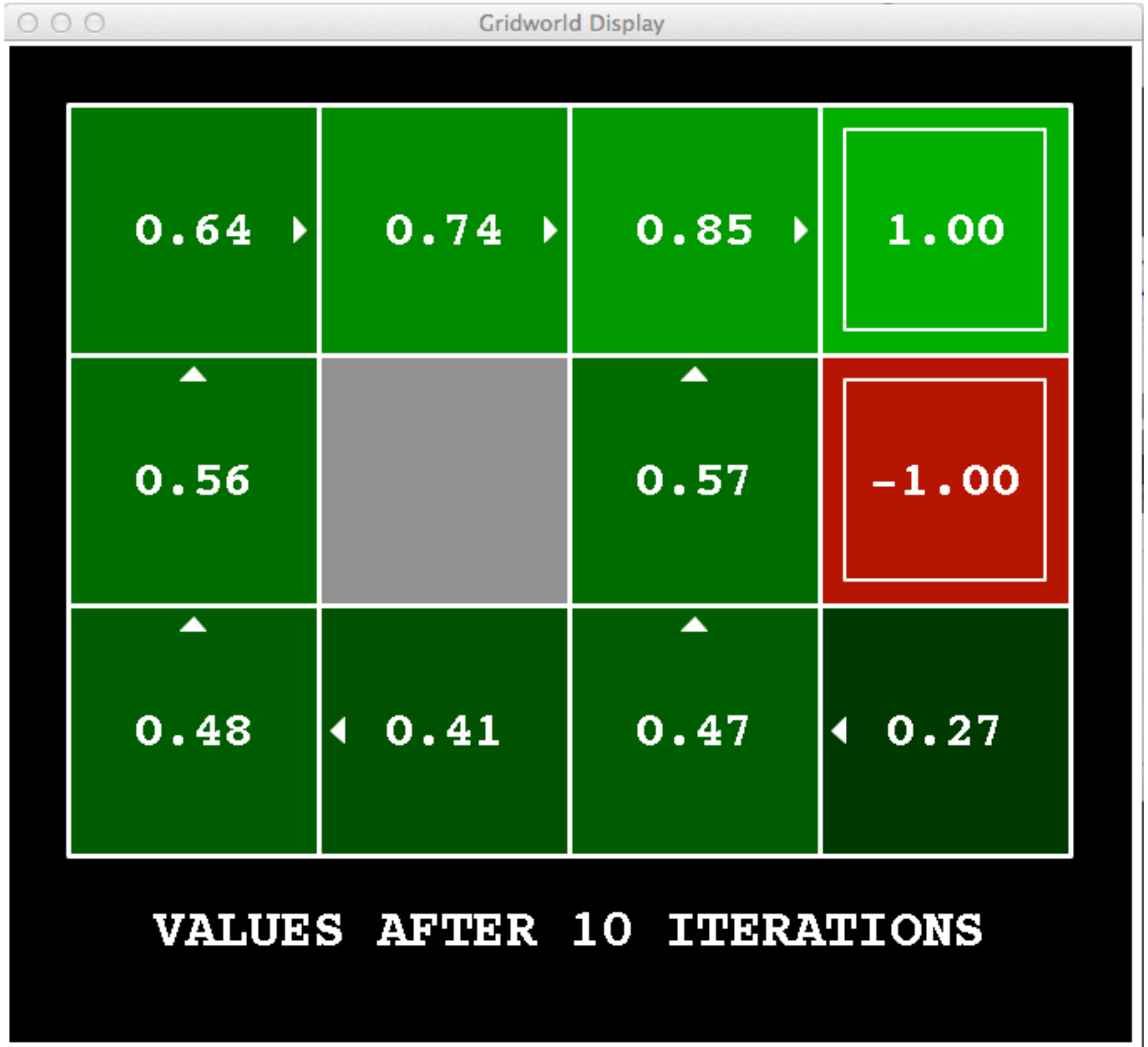

Noise = 0.2
Discount = 0.9
Living reward = 0

# k=9



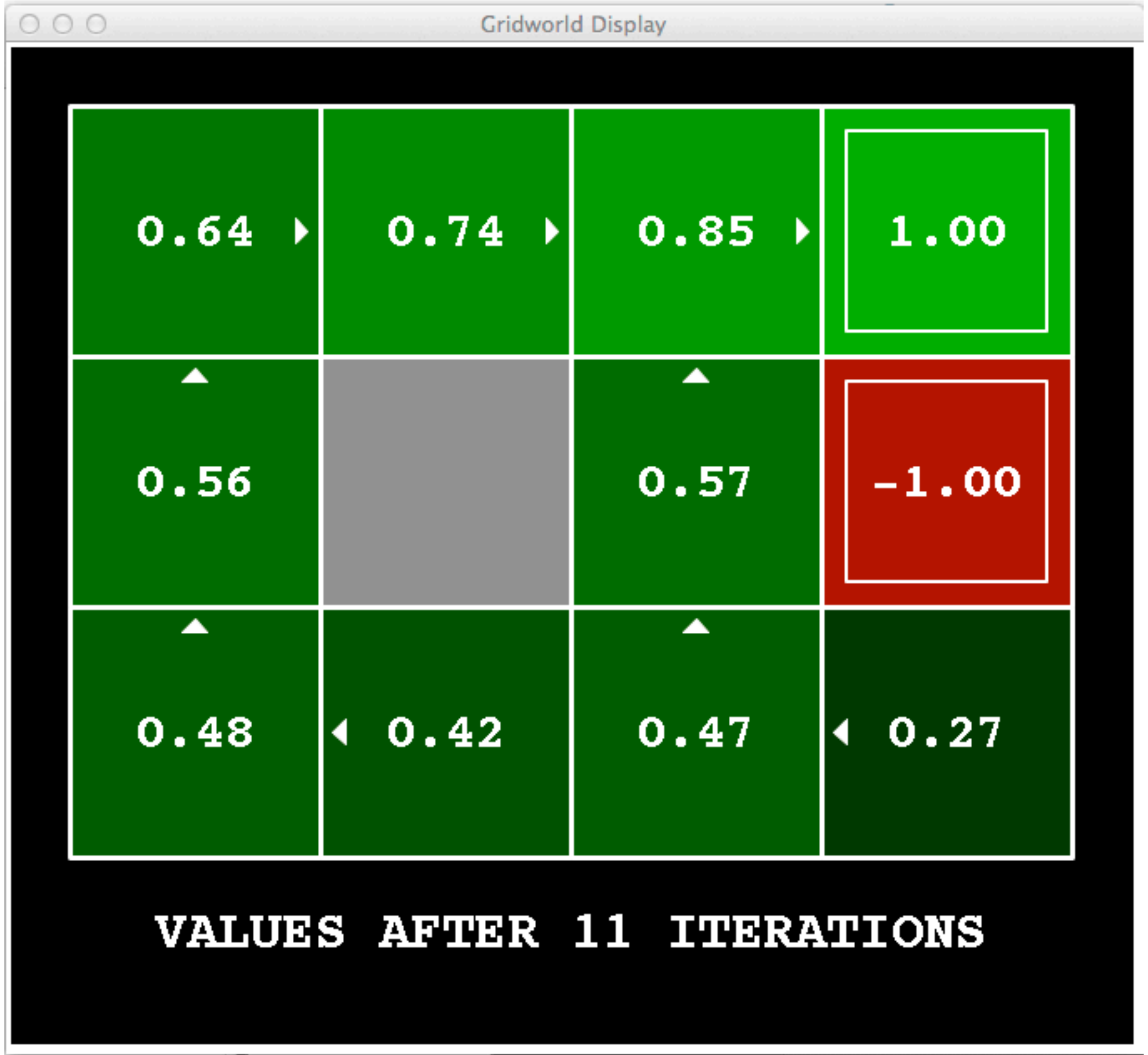

Noise = 0.2
Discount = 0.9
Living reward = 0

# k=10



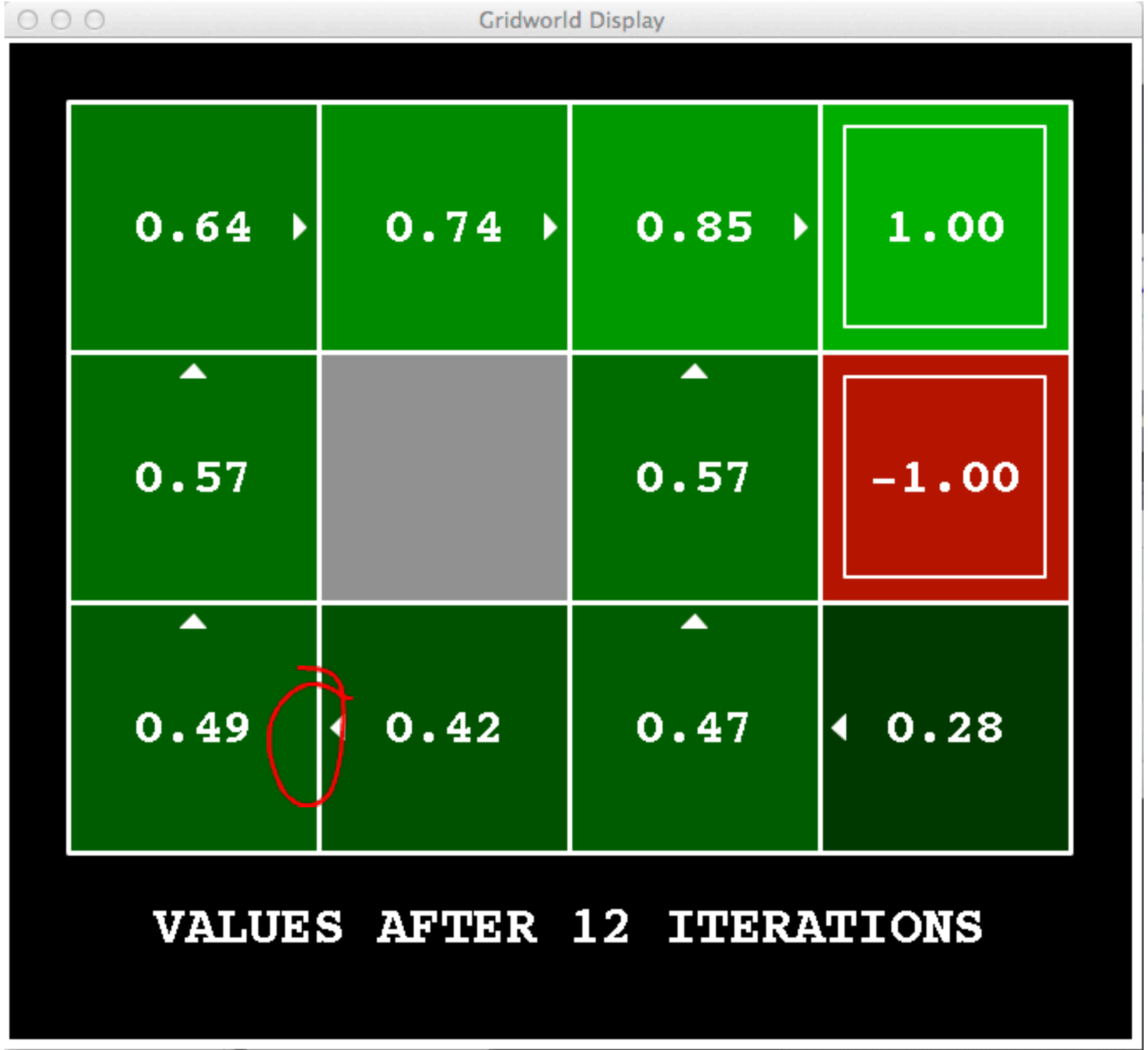

Noise = 0.2
Discount = 0.9
Living reward = 0

# k=11



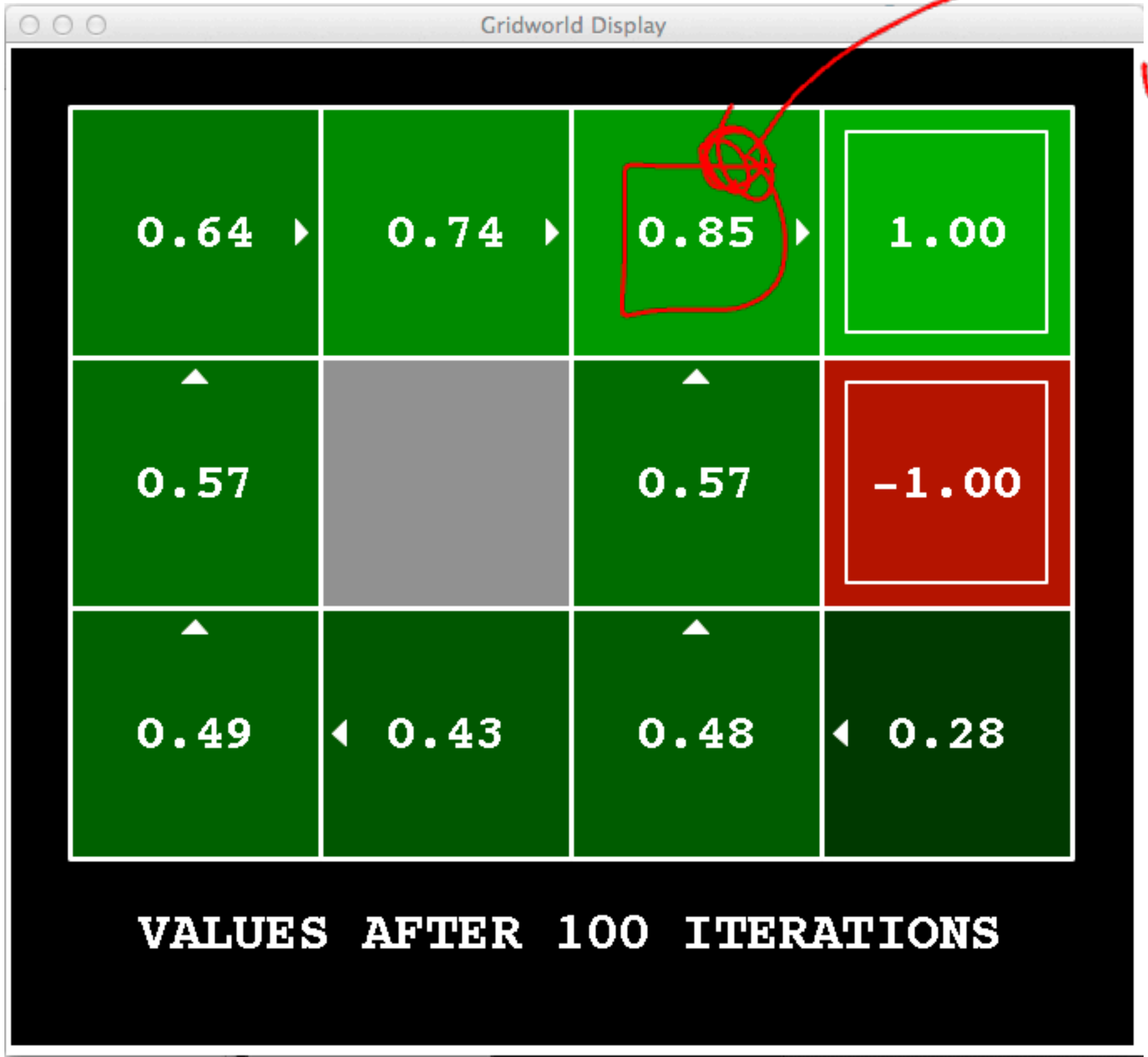

Noise = 0.2
Discount = 0.9
Living reward = 0

# k=12




Noise = 0.2
Discount = 0.9
Living reward = 0

# k=100




$V^*$

$Q^*(s, \text{right})$

$Q^*(s, \text{left})$

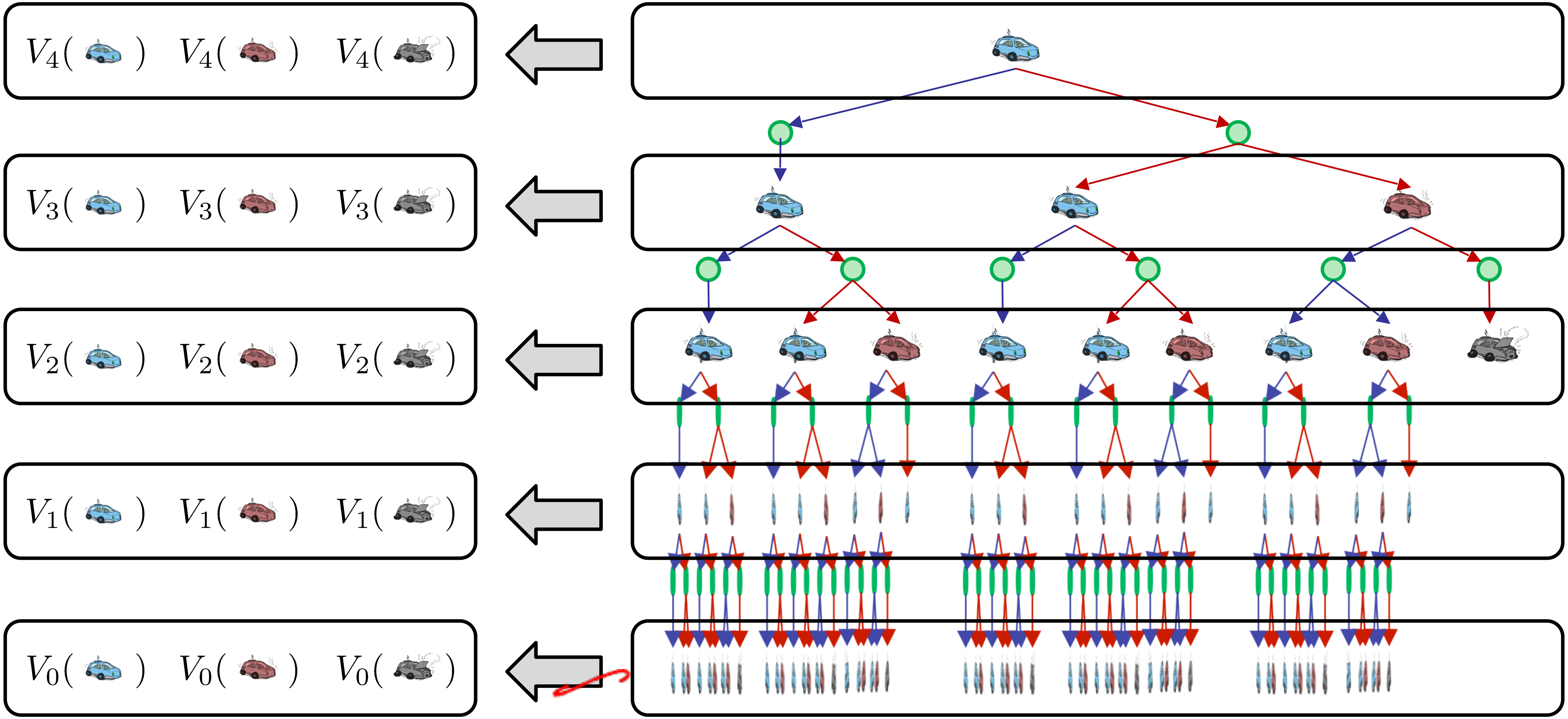
$Q(s, b$

Noise = 0.2

Discount = 0.9

Living reward = 0

# Computing Time-Limited Values

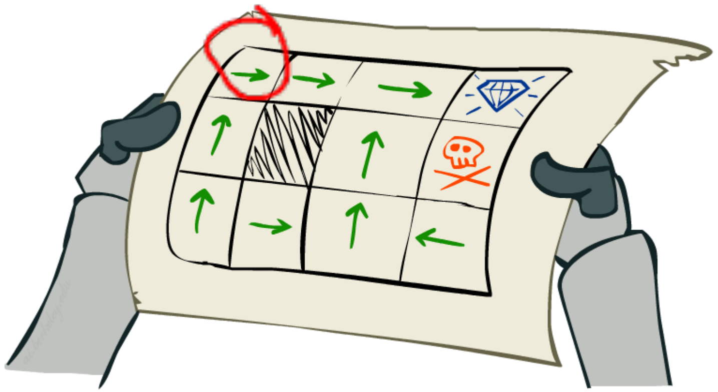


# Recap: MDPs

- 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
  - Solving MDPs: Finding the best policy or mapping of actions to states

$T(s,a,s')$

$R(s,a,s')$



# Optimal Quantities

- The value (utility) of a state s:

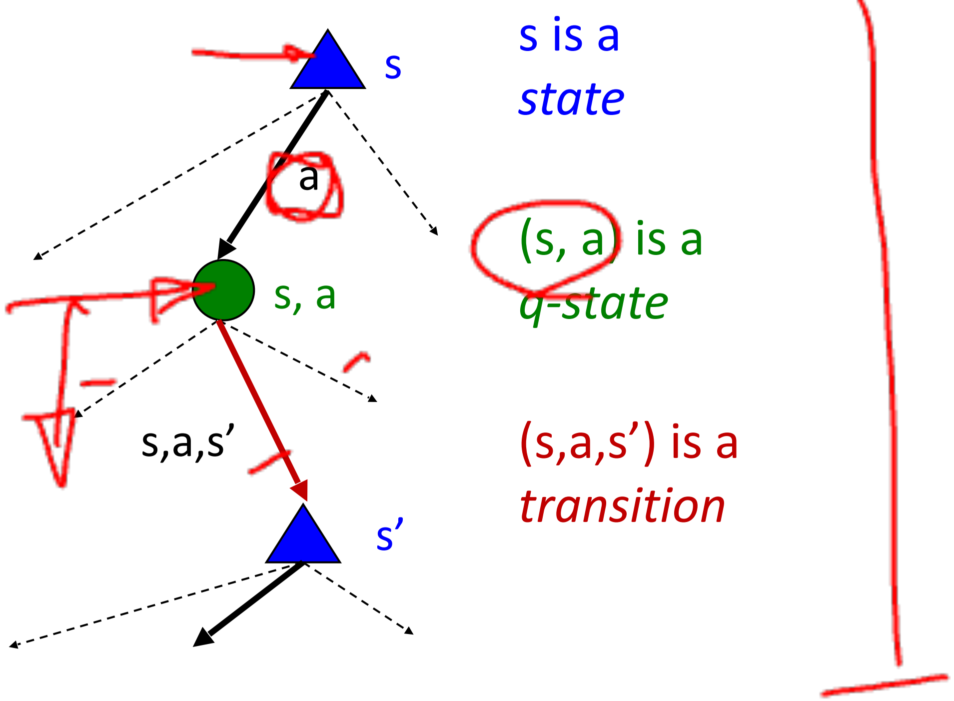
  $V^*(s)$ = expected utility starting in s and acting optimally

- 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

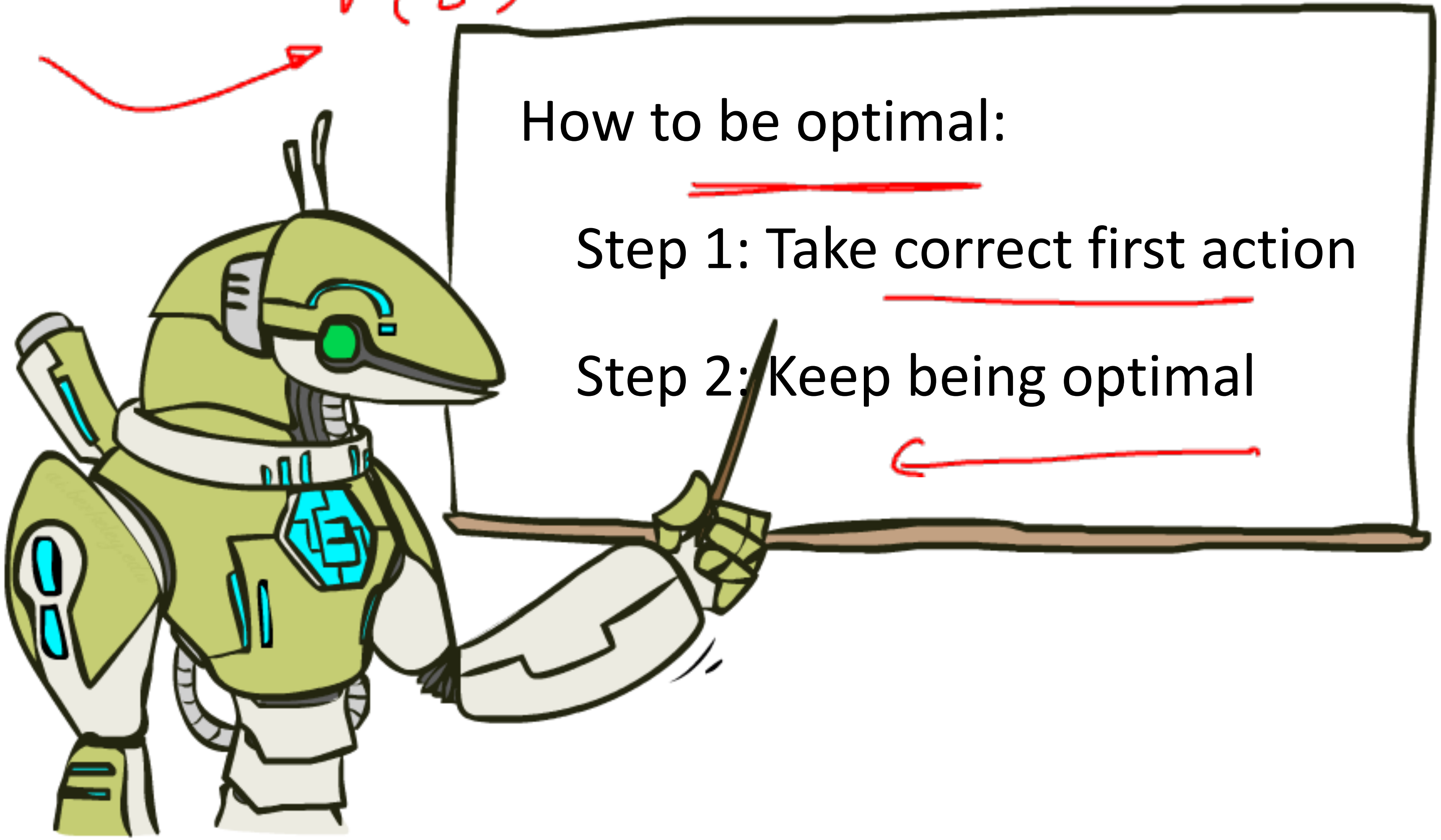
- The optimal policy:

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



# The Bellman Equations

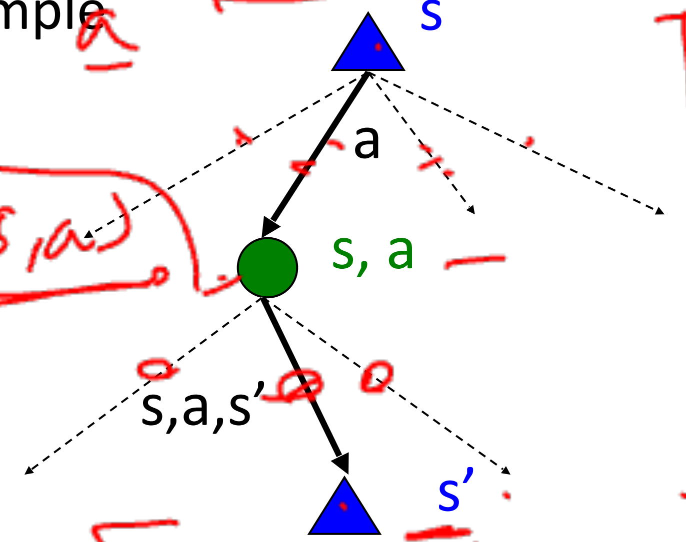
$V(s)$ $V(s')$



# The Bellman Equations

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

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




$Q^*(s,a)$

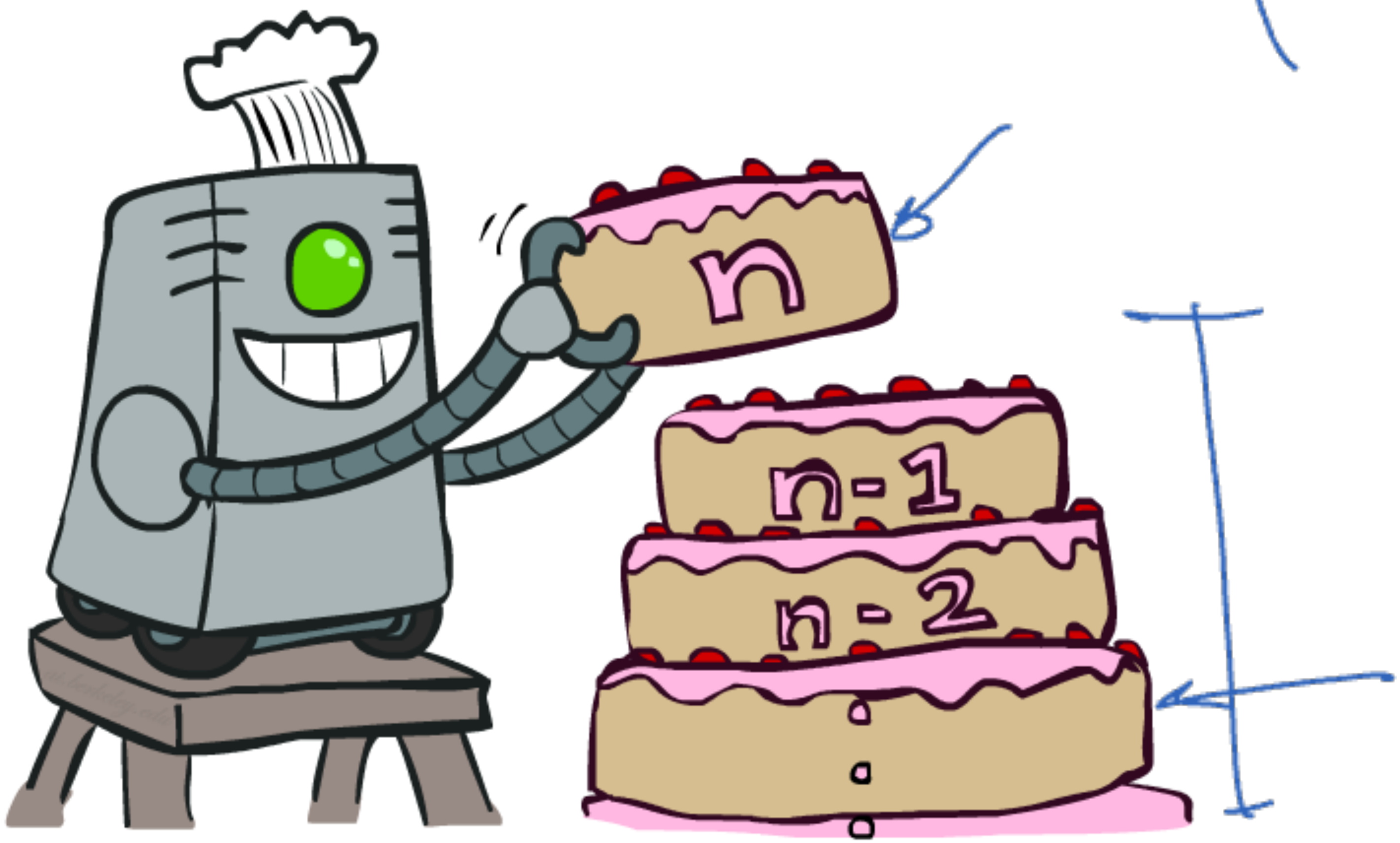
$$V^*(s) = \max_a Q^*(s,a)$$

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

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

- Thes $V^*(s) = \max_a \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V^*(s')\right]$
  valu... in a way, we'll use over and over

# Solving MDPs
# Approach 1: Value Iteration



$V_0$ : 0 steps ahead

# Value Iteration

$V^*$

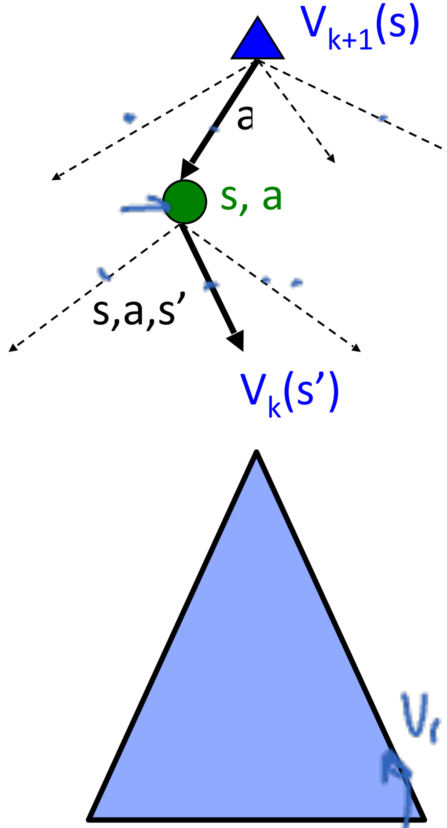
- Start with $V_0(s) = 0$: no time steps left means an expected reward sum of zero
- Given vector of $V_k(s)$ values, do one ply of expectimax from each state:

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

- Repeat until convergence
- Complexity of each iteration: $O(S^2A)$

$|A| \times S^2$

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




$V_0$

# Example: Value Iteration

$V_2$

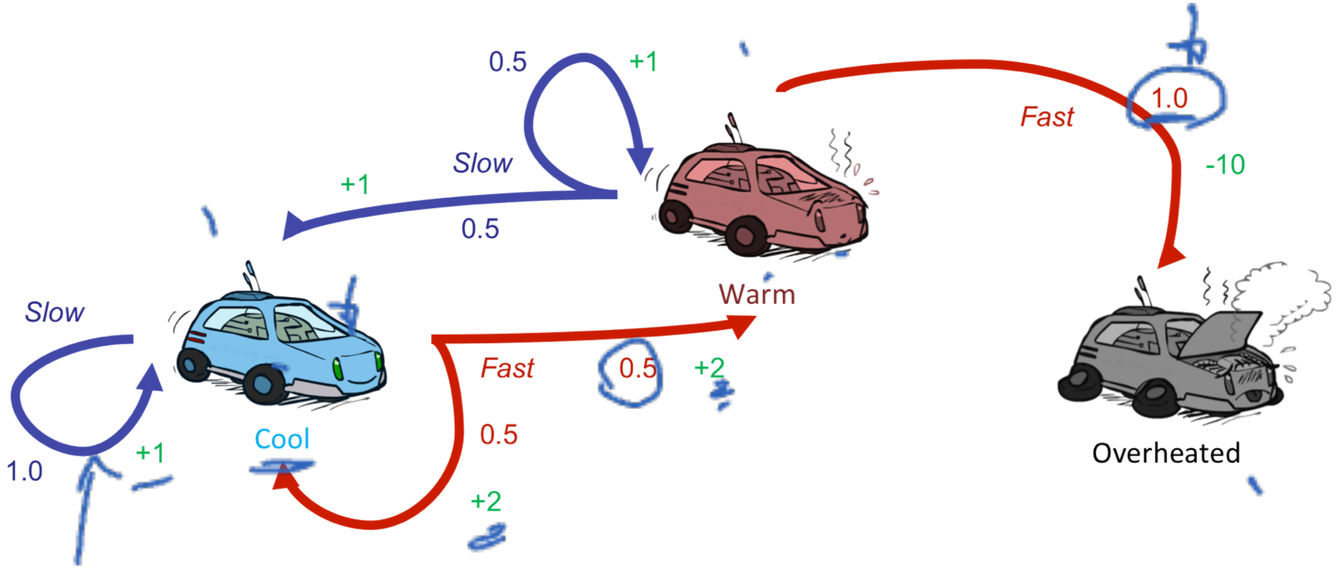
$V_1$

S: 1

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

$V_0$

| | | |
|---|---|---|
| 0 | 0 | 0 |




*Assume no discount!*

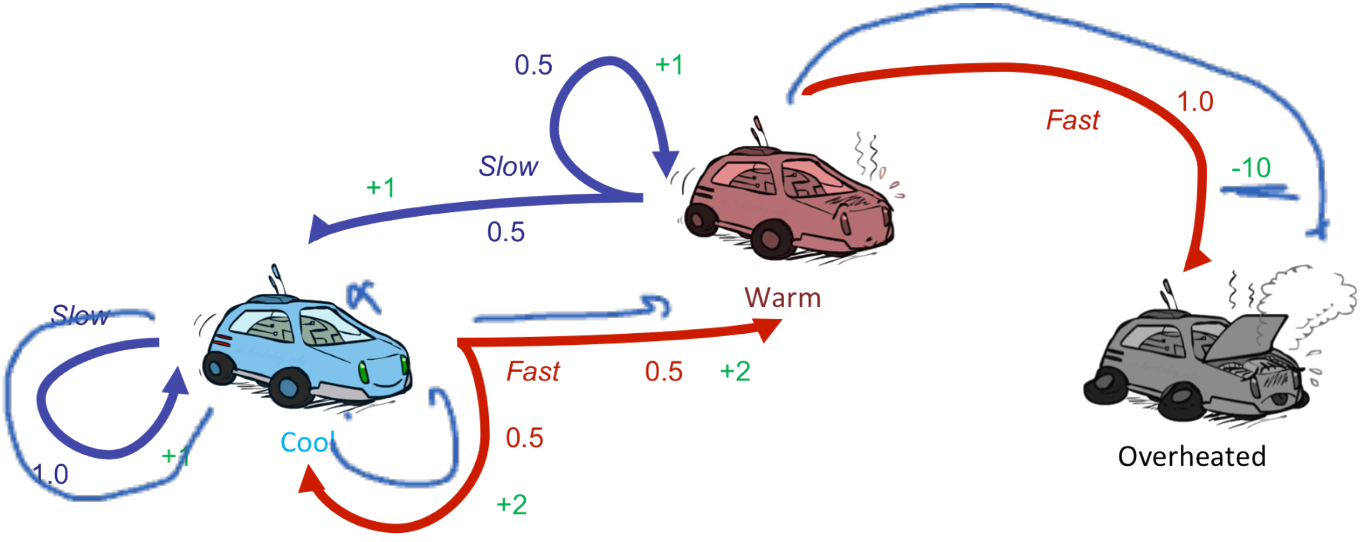
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[ R(s,a,s') + \gamma V_k(s') \right]$$

$V_1 \to$ F: $0.5[+2+0] + 0.5[+2+0] = 2$

S: 1

# 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!*

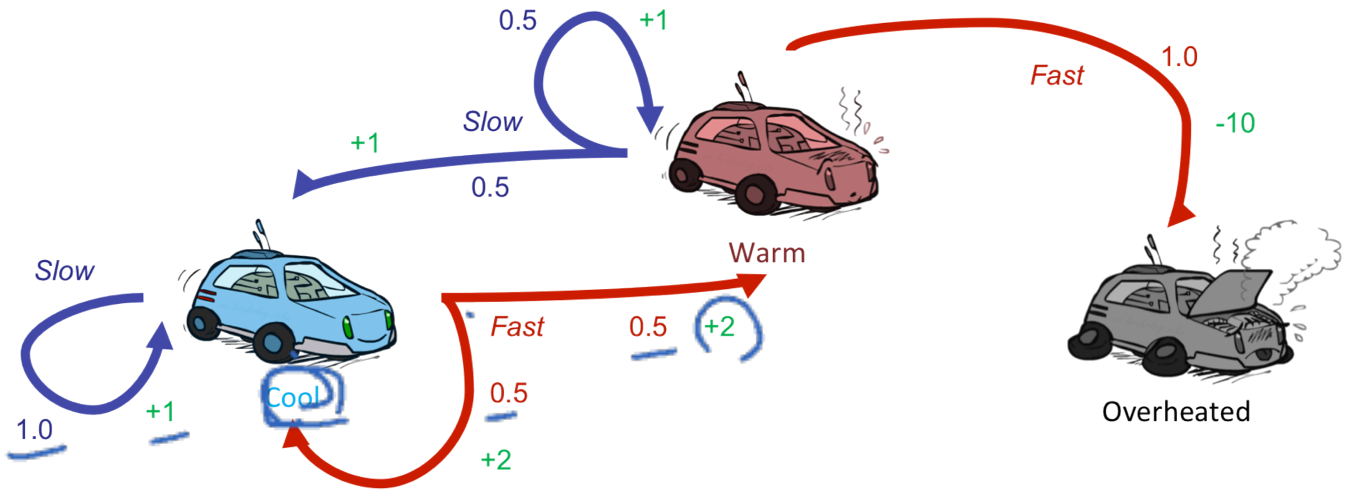
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V_k(s')\right]$$

F  -10  +.

S

# Example: Value Iteration

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




*Assume no discount!*

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

S: $1[+1 + 1 \times 2] = 3$

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

# Example: Value Iteration

$V_2$

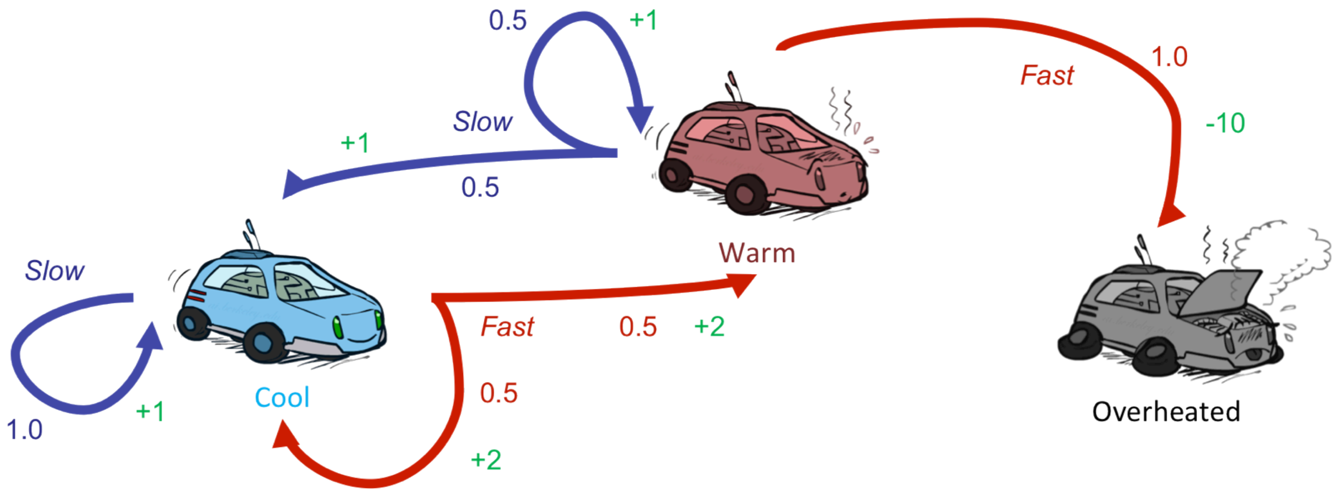
S: 1+2=3
F: .5*(2+2)+.5*(2+1)=3.5

$V_1$

| | | |
|---|---|---|
| 2 | 1 | 0 |

$V_0$

| | | |
|---|---|---|
| 0 | 0 | 0 |

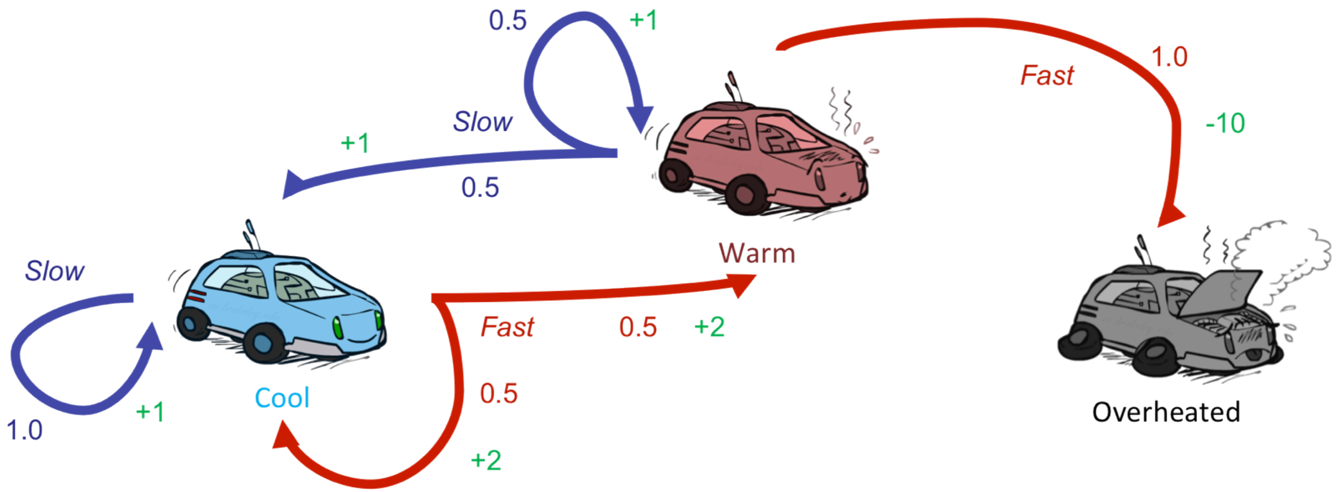



*Assume no discount!*

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

# Example: Value Iteration

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




*Assume no discount!*

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

# Value Iteration

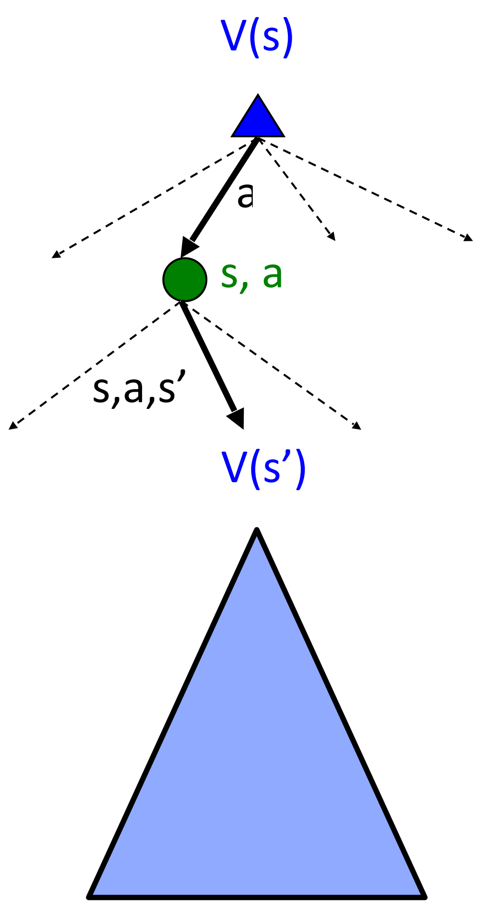
- Bellman equations characterize the optimal values:

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

- 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



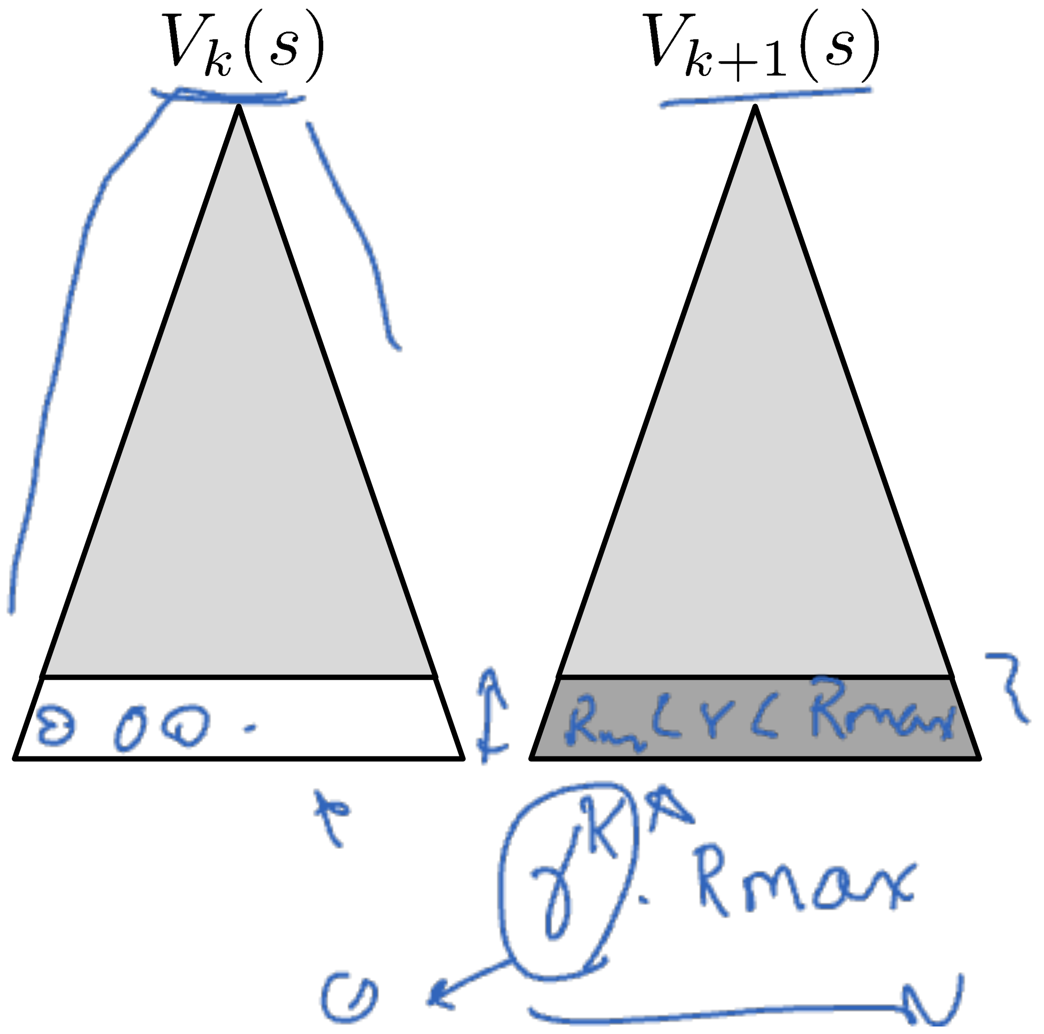
# Convergence*

$\epsilon$

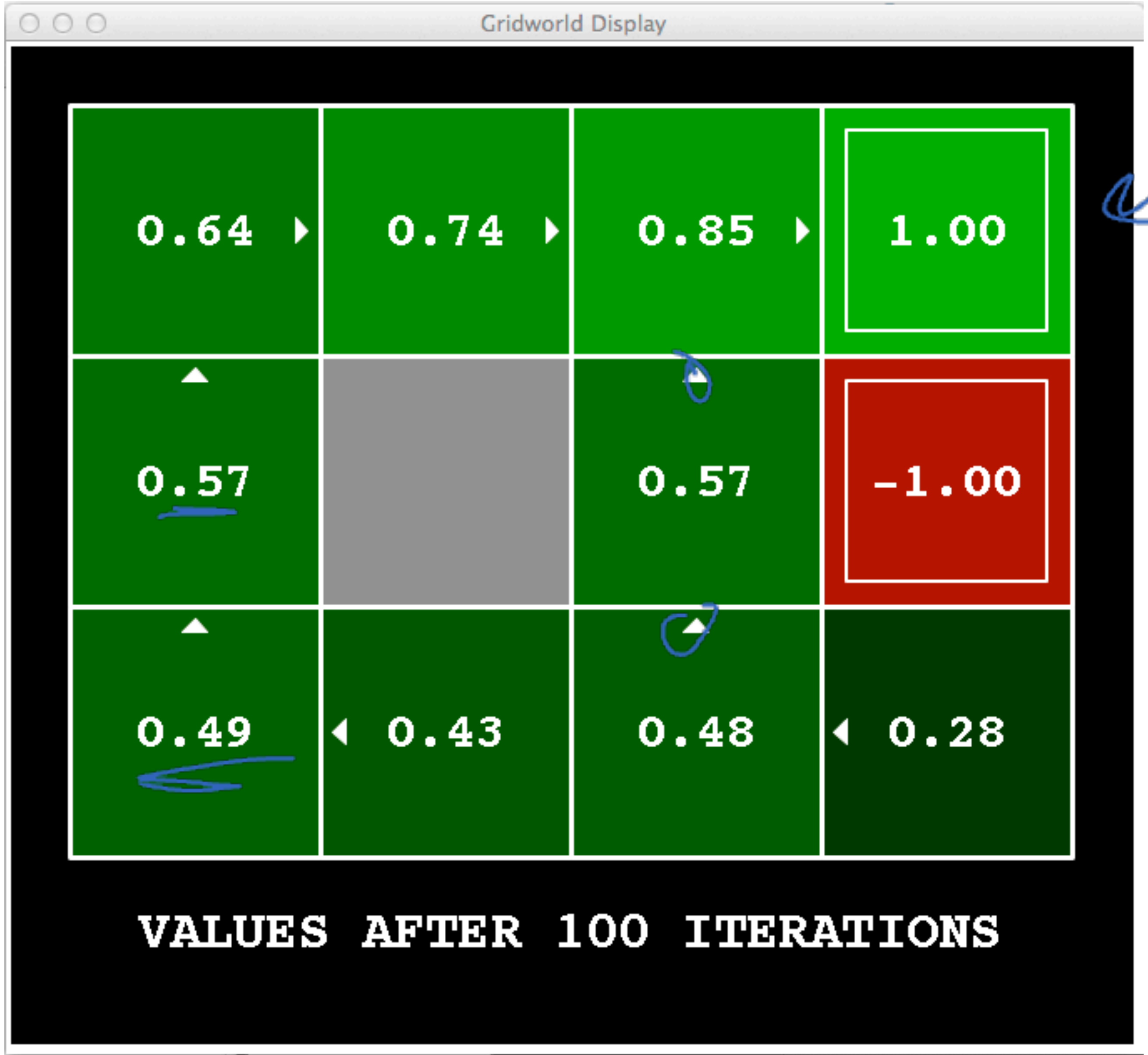
- How do we know the $V_k$ vectors are going to converge?

- 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 $\gamma$
  - 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



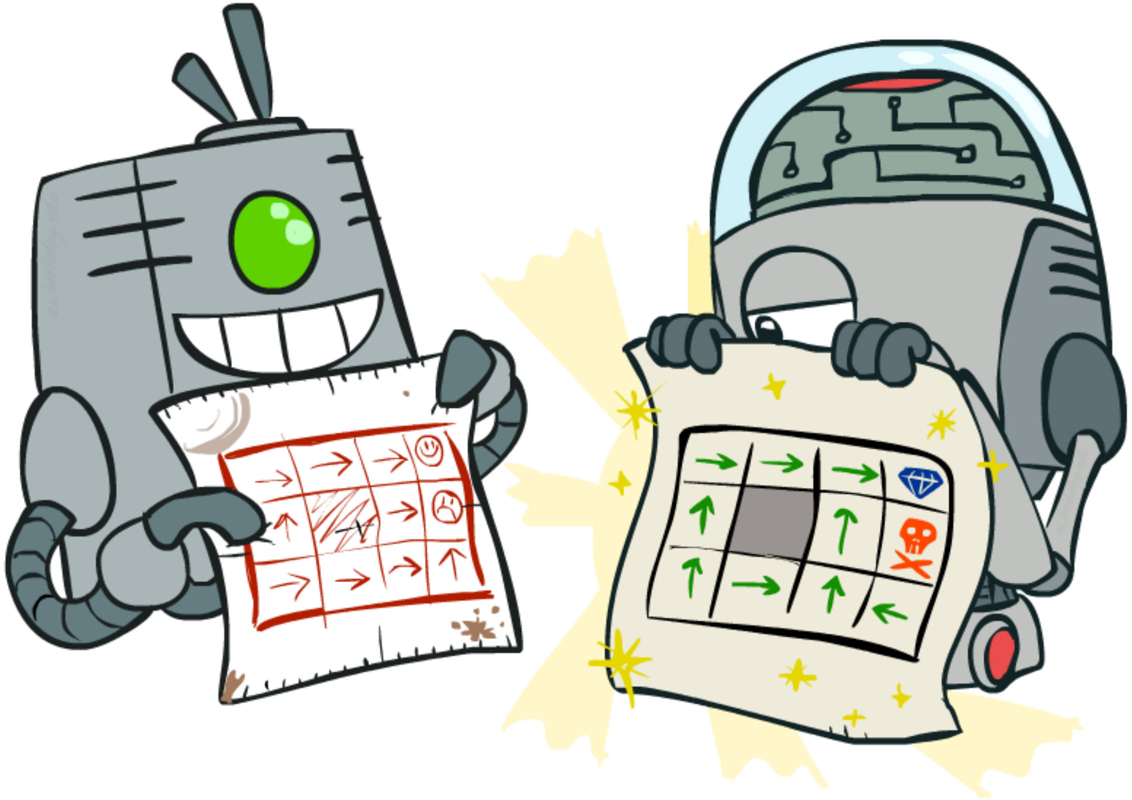
# Outcome of Value Iteration?



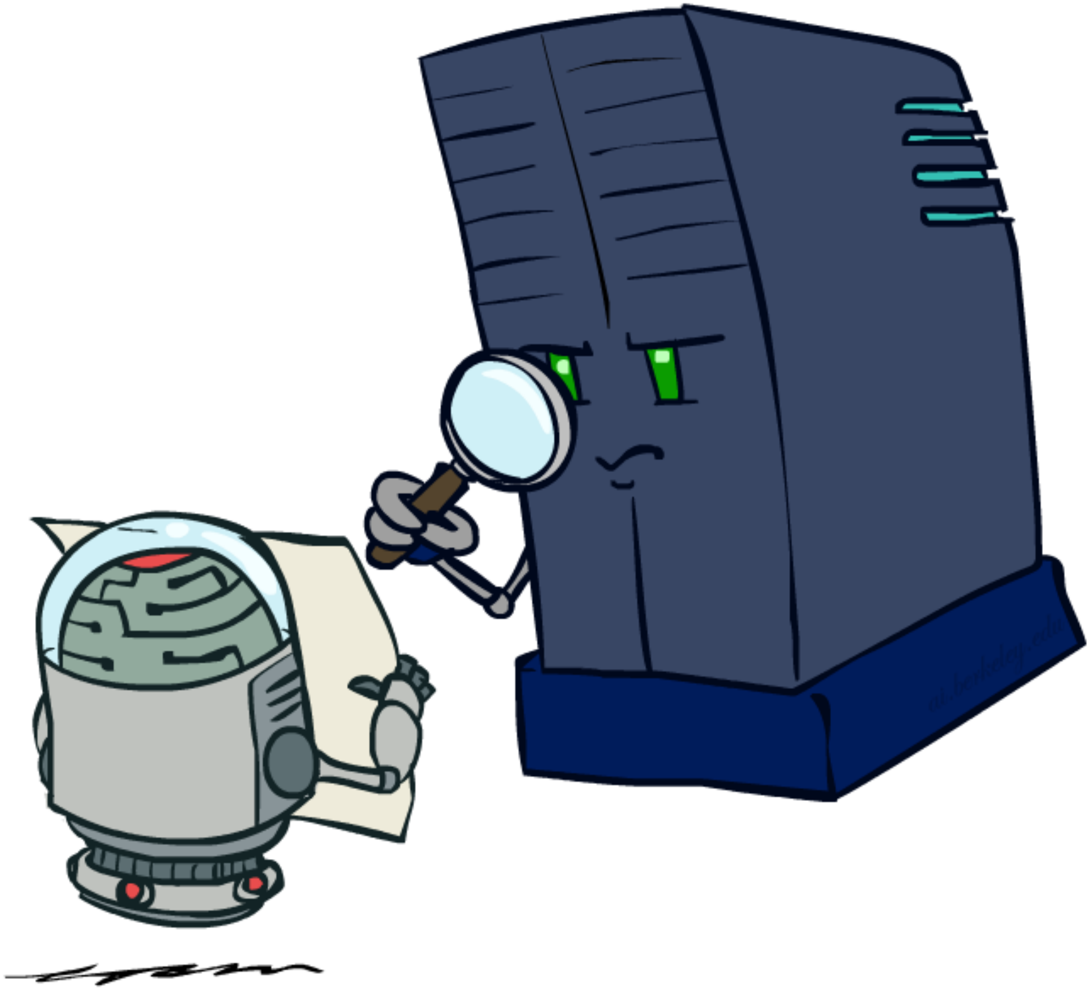

Noise = 0.2
Discount = 0.9
Living reward = 0

# Policy Methods

- Compute Values for Policies
- Compute Actions from Values
- Directly search for policies

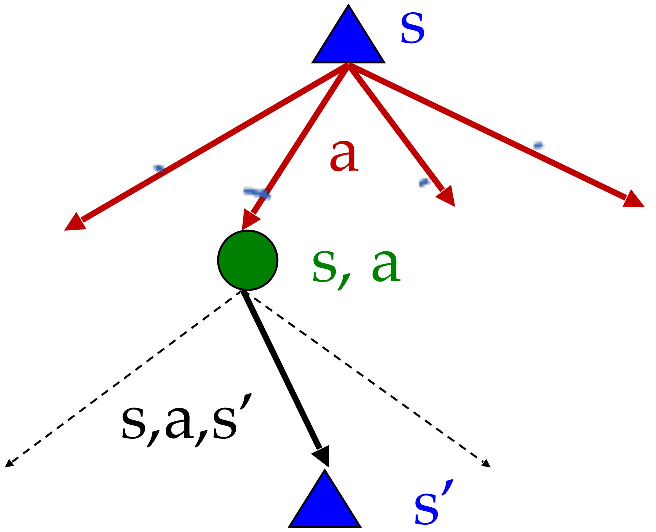


# Policy Evaluation

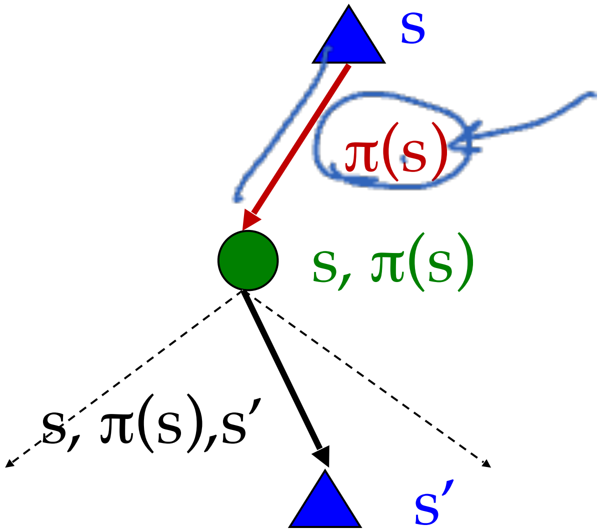


# Fixed Policies

Do the optimal action




Do what π says to do




- Expectimax trees max over all actions to compute the optimal values
- If we fixed some policy π(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

$V^*$ optimal

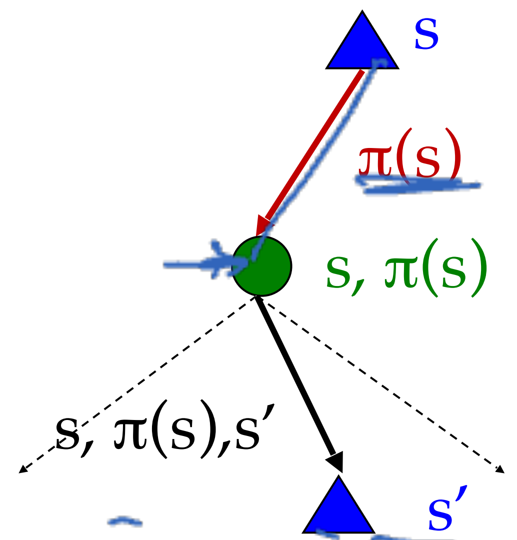
- Define the utility of a state s, under a fixed policy π:

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

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

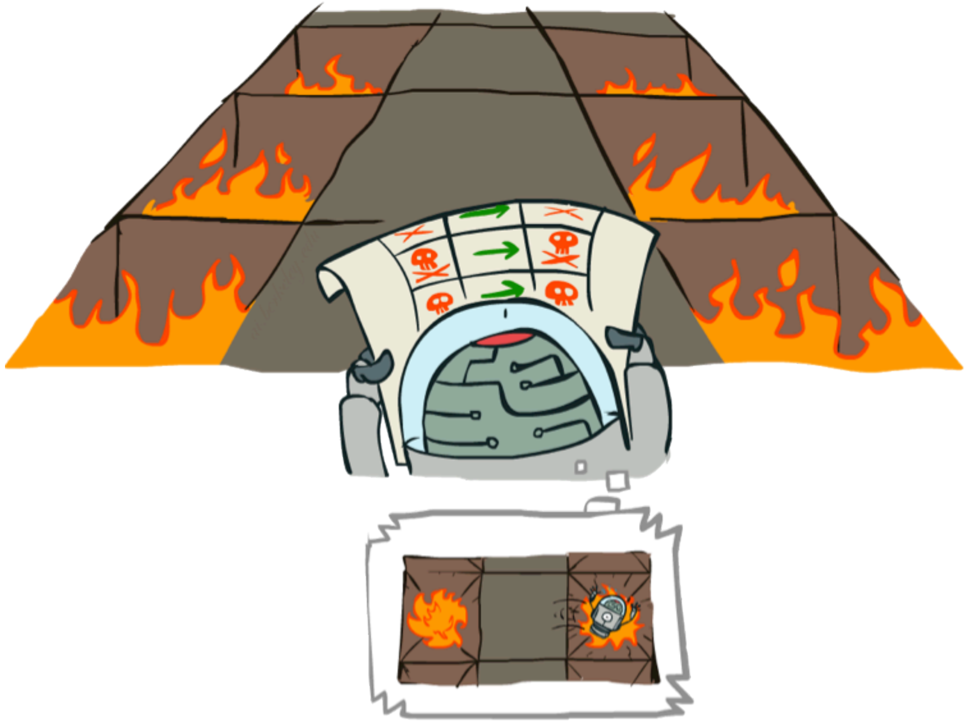
$$V^\pi(s) = \sum_{s'} T(s,\pi(s),s')\left[R(s,\pi(s),s') + \gamma V^\pi(s')\right]$$

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

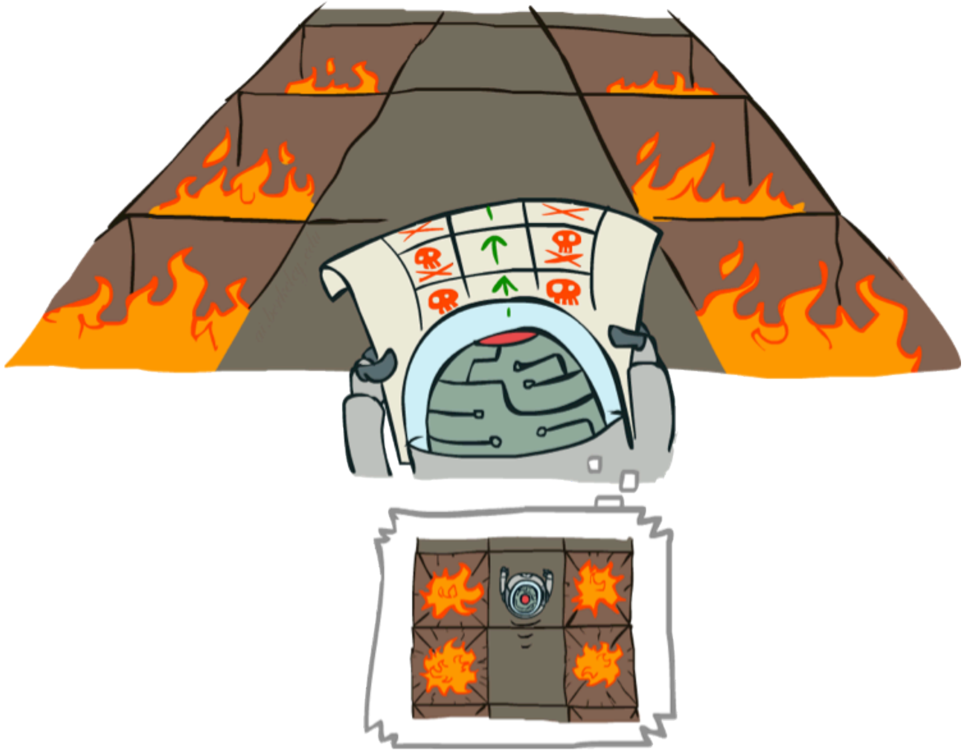


# Example: Policy Evaluation

Always Go Right

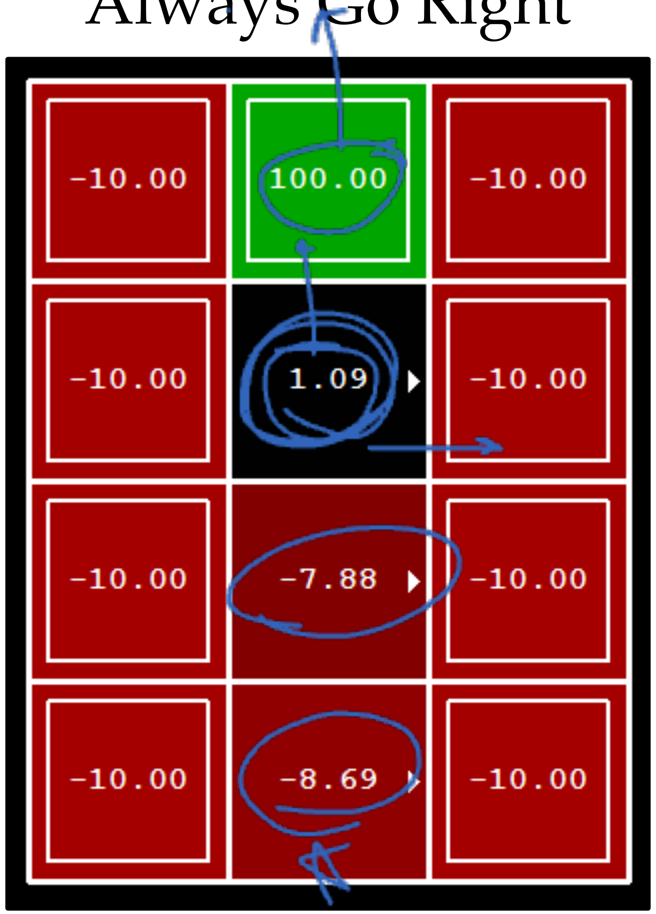


Always Go Forward

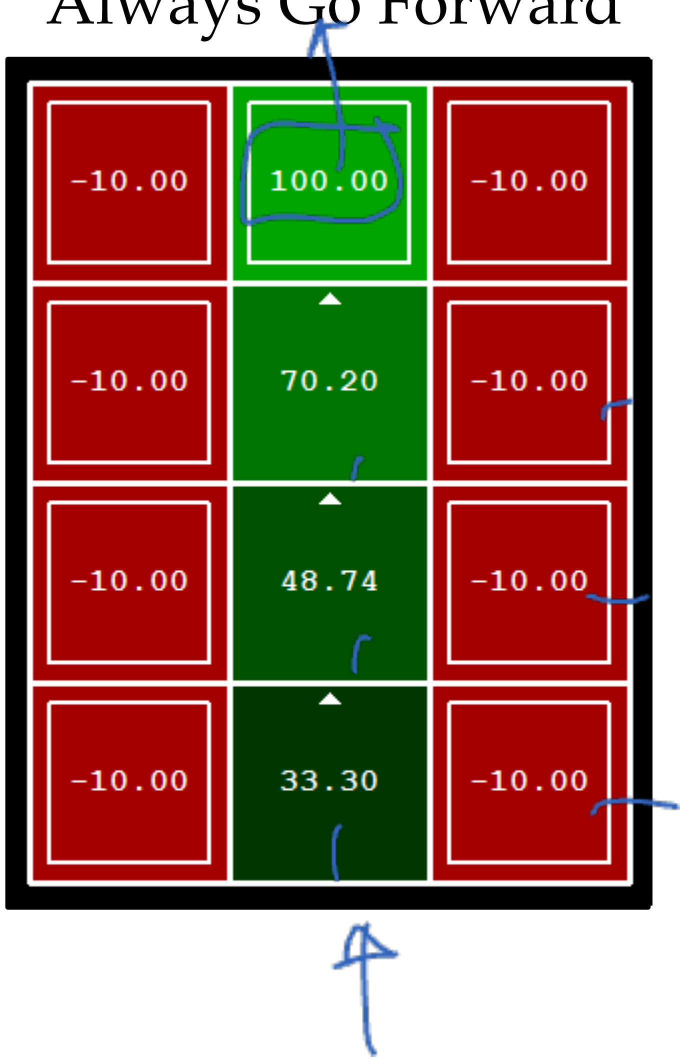


# Example: Policy Evaluation

Always Go Right




Always Go Forward



optmil policy & values

Value Iteration

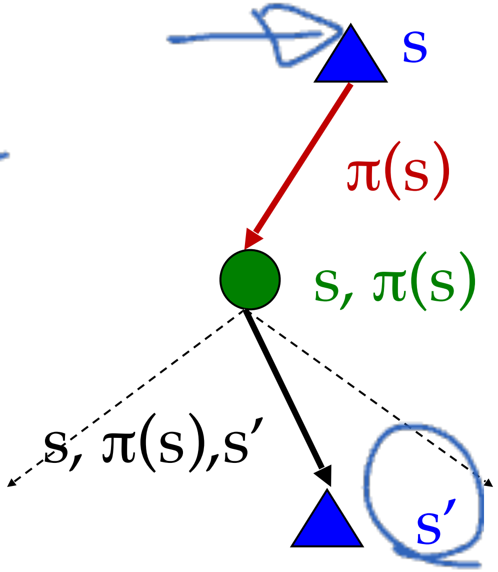
# Policy Evaluation

s'

- How do we calculate the V's for a fixed policy π?
- Idea 1: Turn recursive Bellman equations into updates (like value iteration)

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

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




S1, S, S3

- Efficiency: O(S²) per iteration
- Idea 2: Without the maxes, the Bellman equations are just a linear system
  - Solve with Matlab (or your favorite linear system solver)

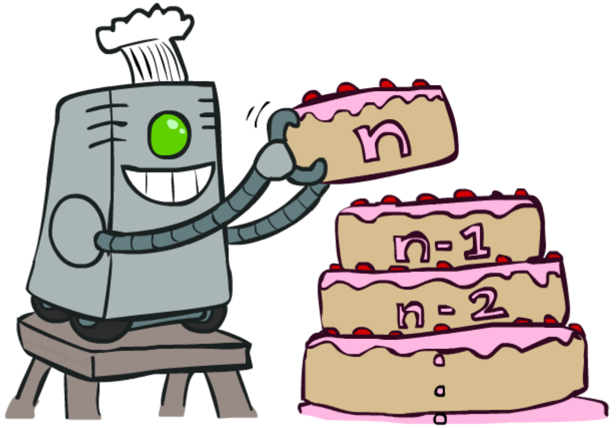
$V^\pi(S_1) =$

$V(S_2)$

$V^\pi(S_3)$

# Solving MDPs

- Finding the best policy → mapping of actions to states
- So far, we have talked about two methods
  - Policy evaluation: computes the value of ~~a **fixed** policy~~
  - ~~Value iteration~~: computes ~~the **optimal** values~~ of states



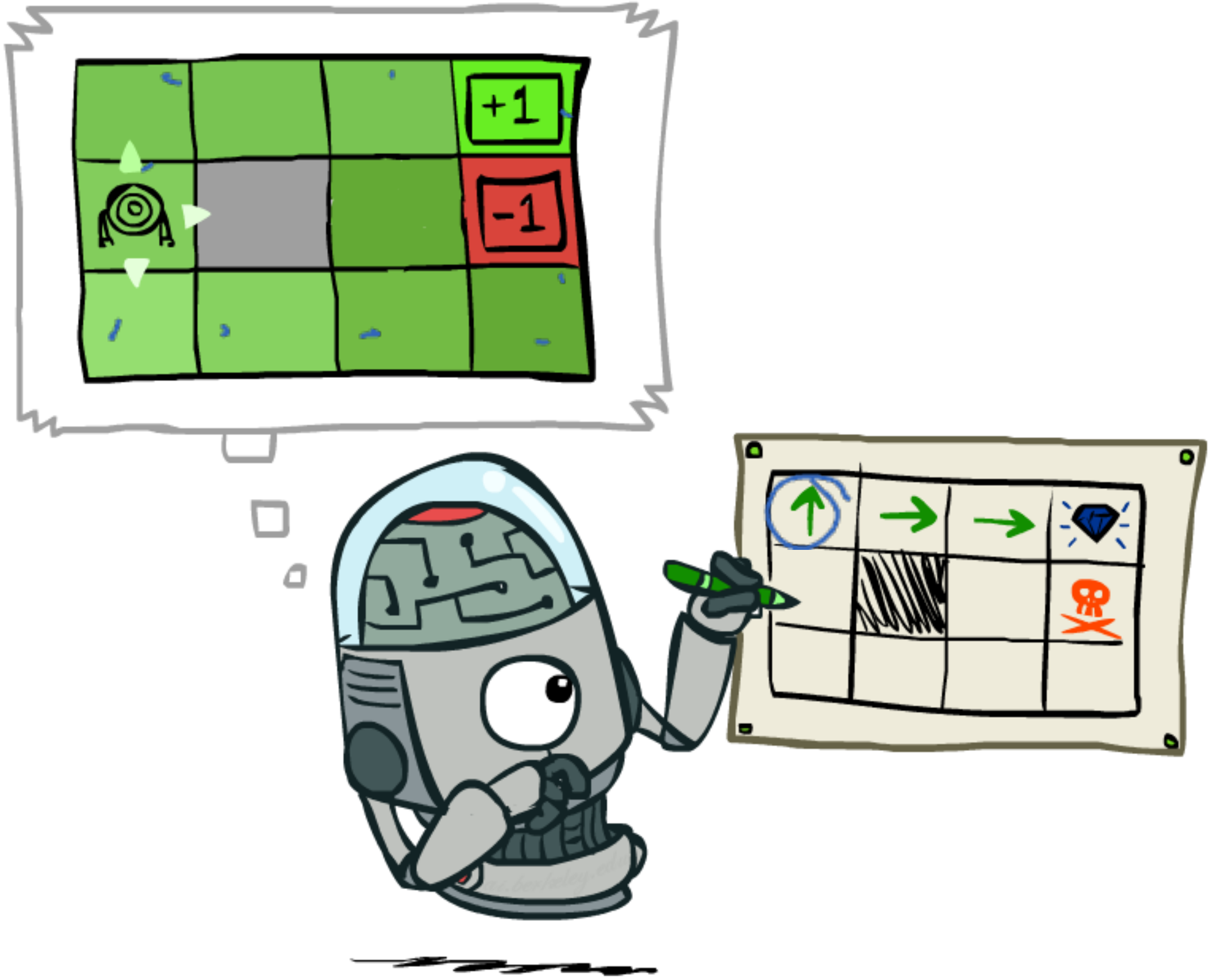
# Let's think…

values

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

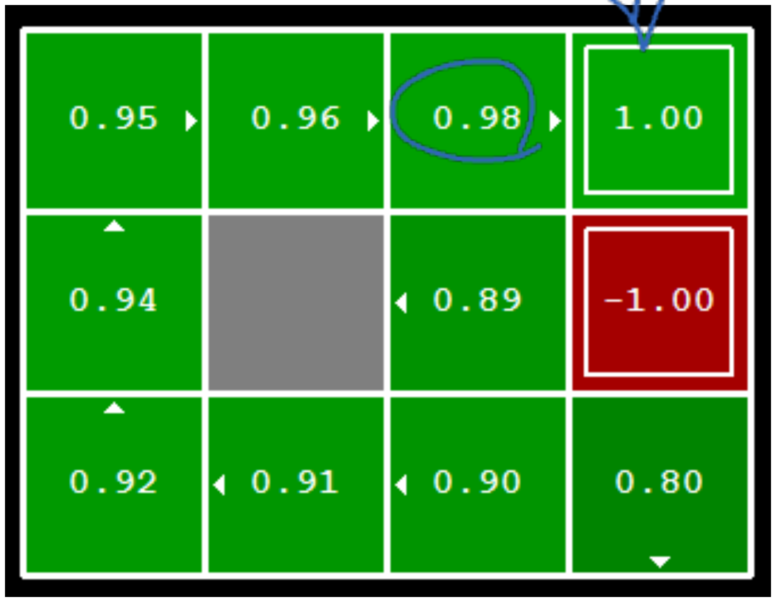
action

# Policy Extraction



# Computing Actions from Values

$V^* \to a^*$




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

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

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

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

$V^*(s)$


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

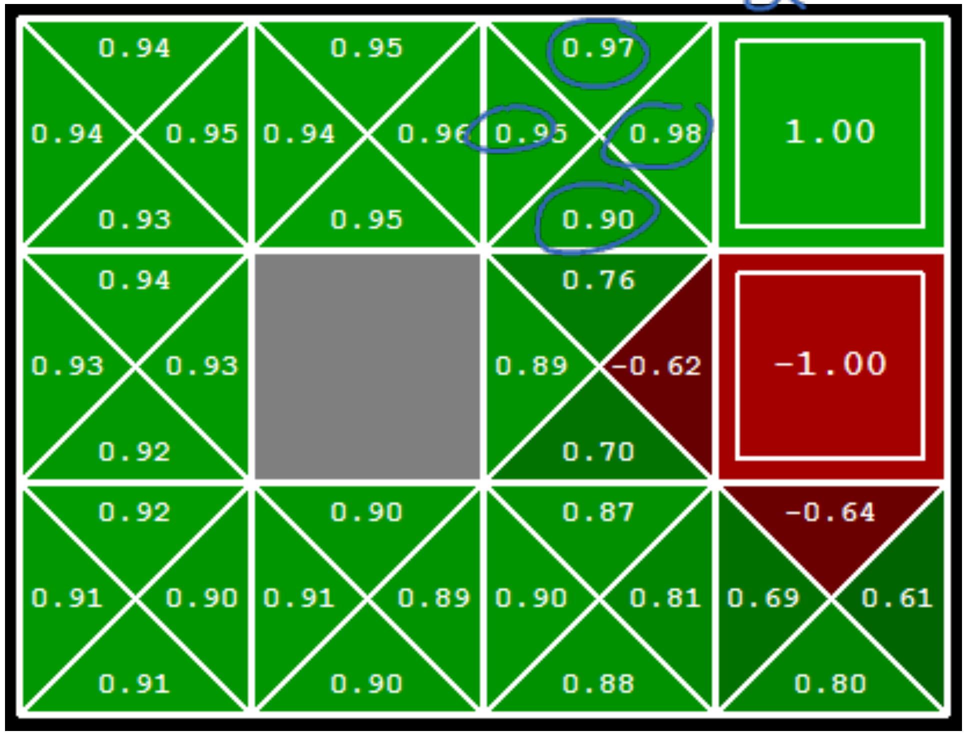
# Computing Actions from Q-Values

V Q

max {1, 2, 3}

$\operatorname{argmax}_a \{f_1(a), f_2, \ldots\}$

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




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

$$\pi^*(s) = \underbrace{\arg\max_a}\ \boxed{Q^*(s,a)}$$

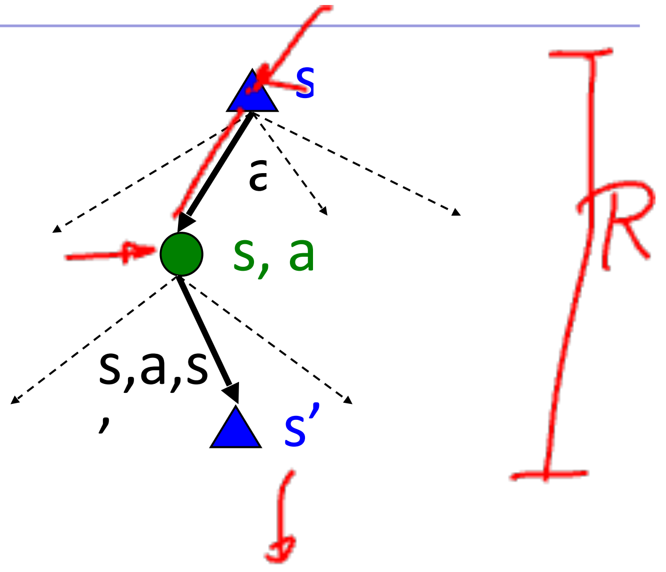
- Important lesson: actions are easier to select from q-values than values!

V → actions

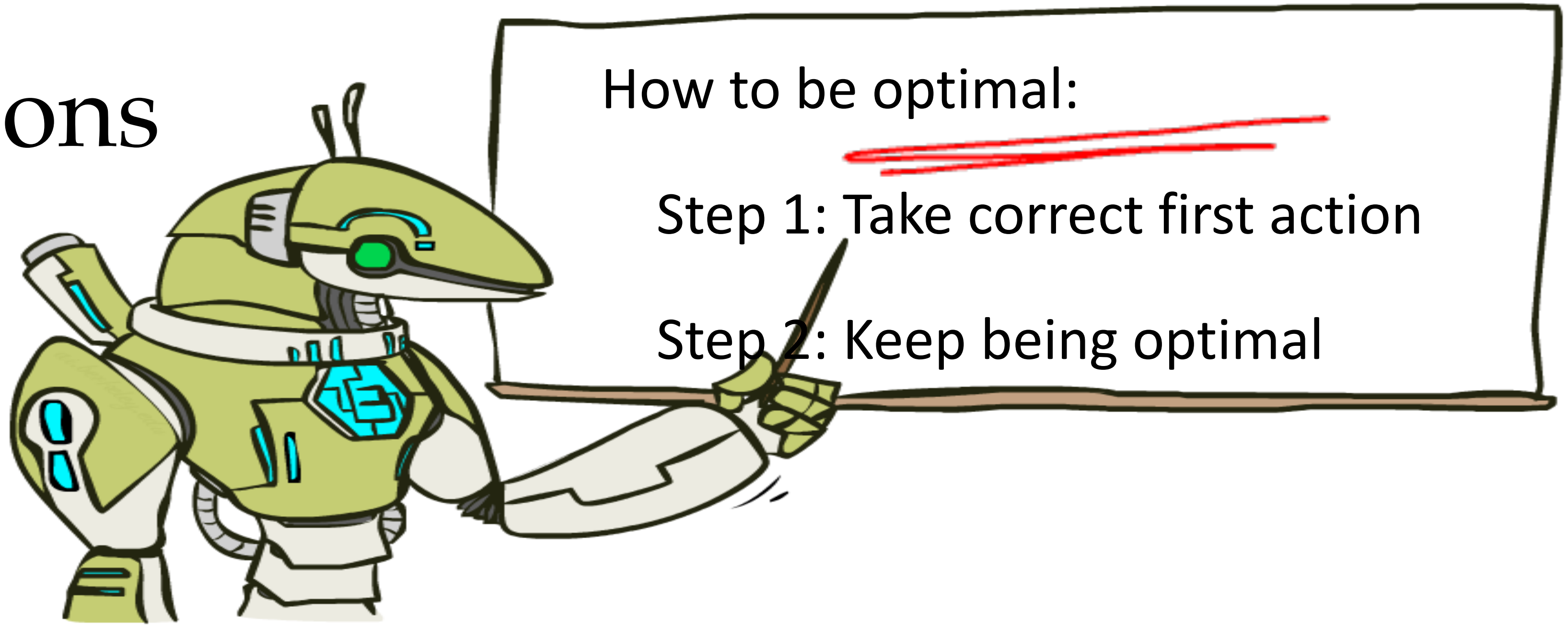
Q → ach

# Recap: MDPs and Bellman Updates

- 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 $\gamma$)

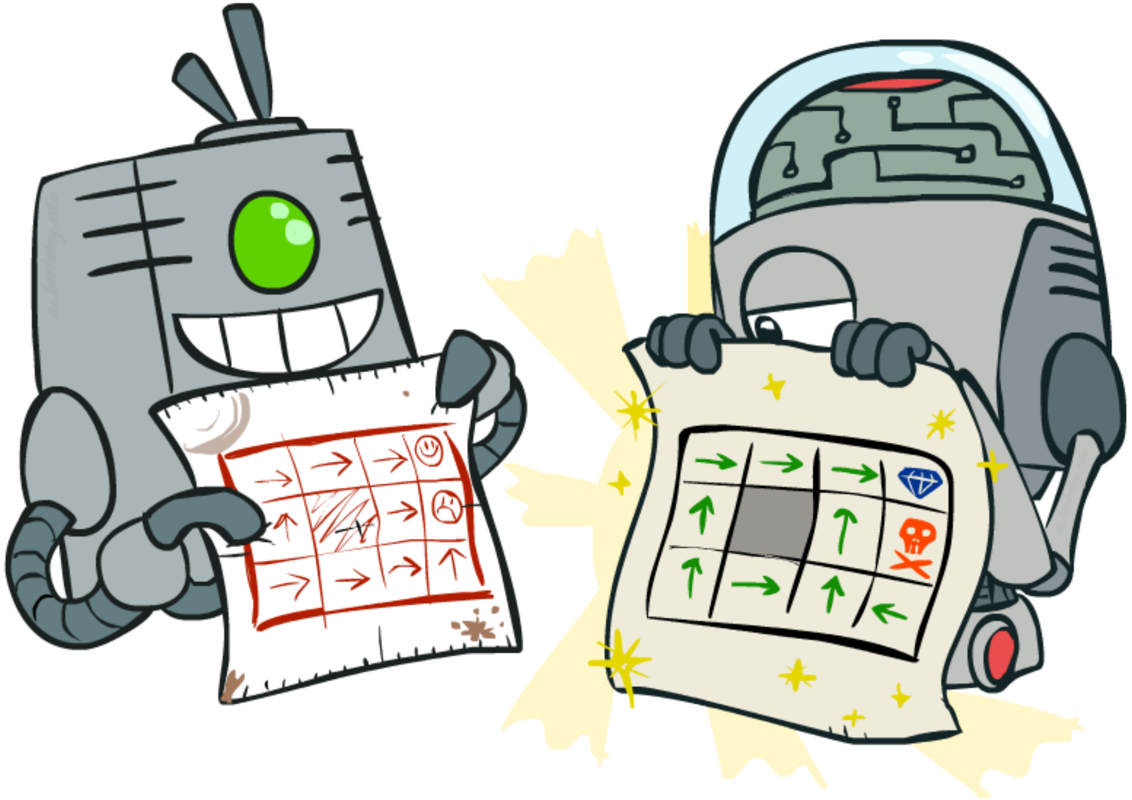



## Bellman Equations



# Recap: Computations

- Compute Values for Policies
  - Optimal policy: Value Iteration
  - Fixed policy: Policy evaluation
- Compute Actions from Values
  - Policy extraction
- Directly search for policies

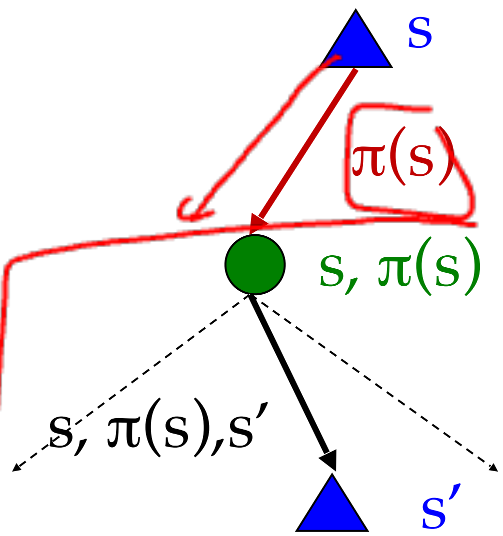


# Recap: Computing values for policies

- Compute values for a fixed policy: policy evaluation

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

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



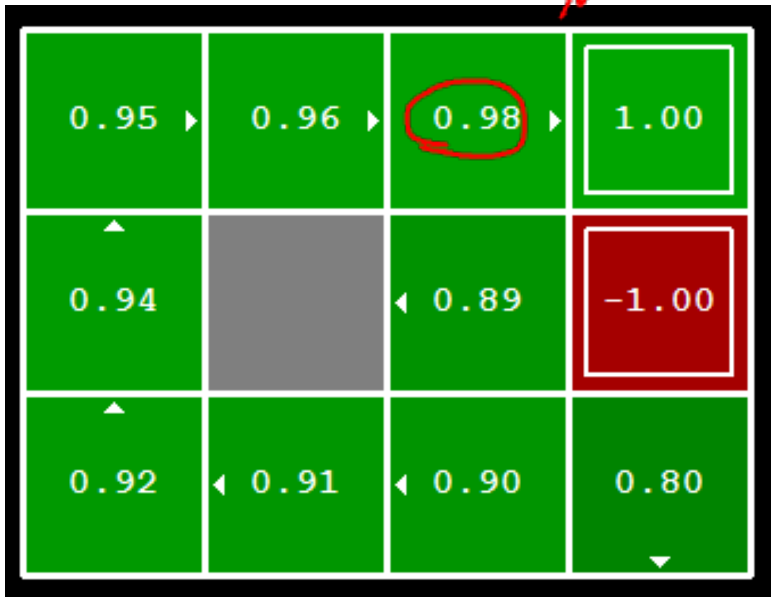

- Compute values for optimal policy: Value Iteration

$$V_0(s) = 0$$

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

# Computing Actions from Values

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



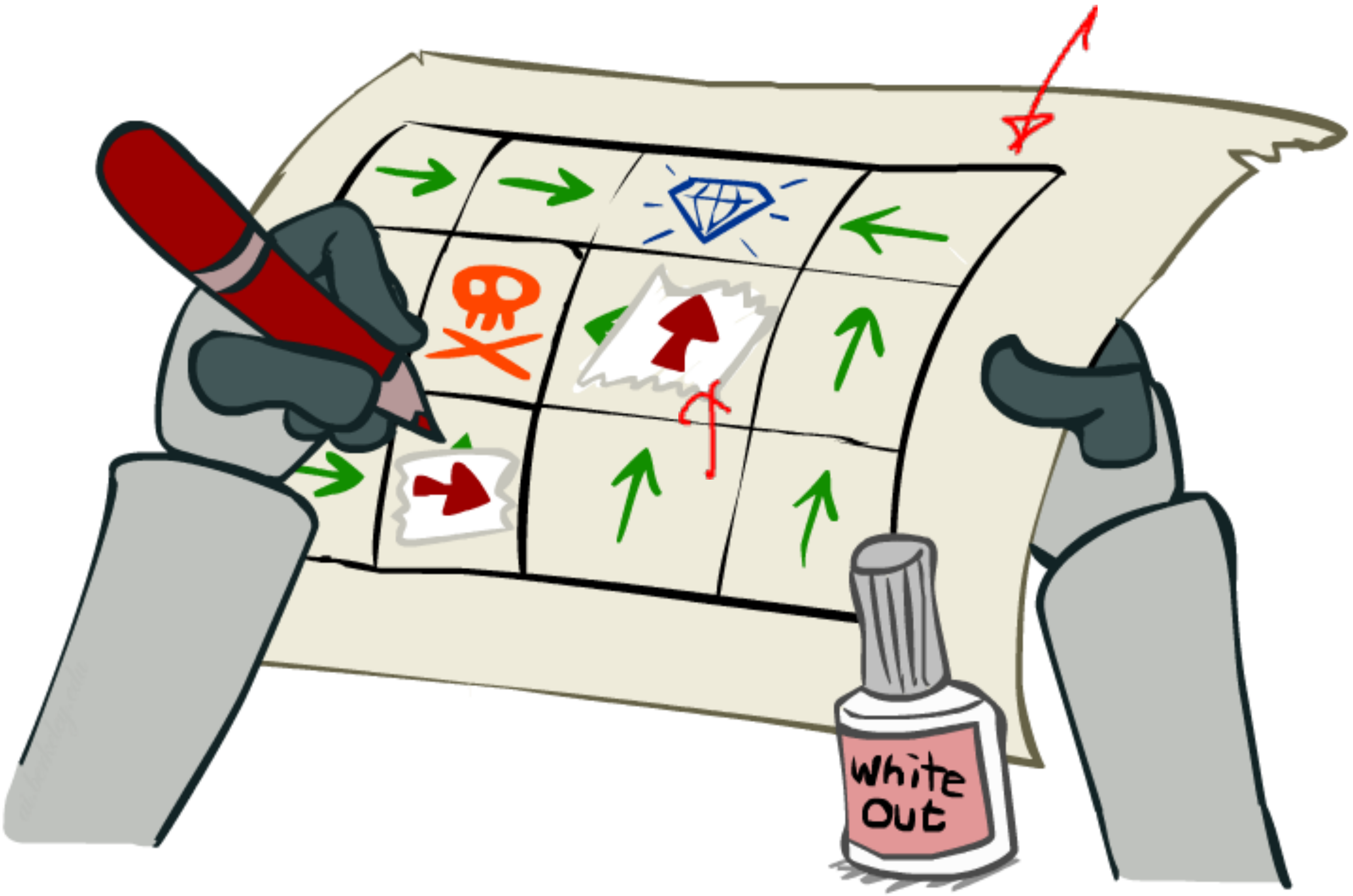

- How should we act?
  - It's not obvious!

- 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

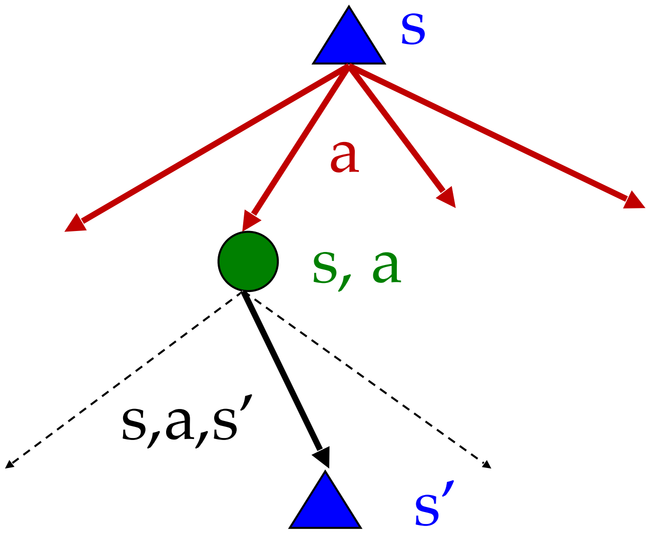
# Policy Iteration



# Problems with Value Iteration

- Value iteration repeats the Bellman updates:

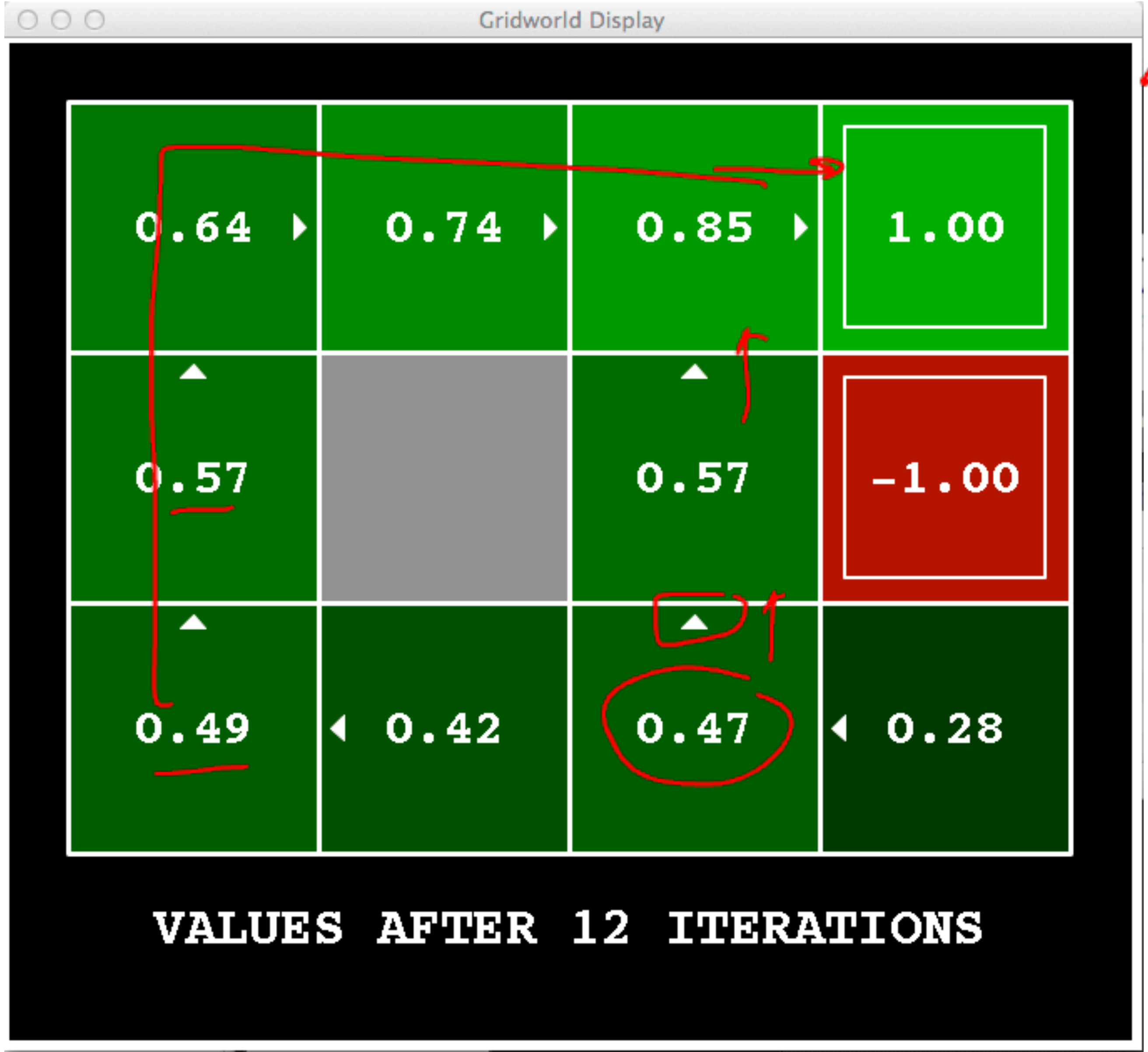
$$V_{k+1}(s) \leftarrow \max_{a} \sum_{s'} T(s,a,s') \left[ R(s,a,s') + \gamma V_k(s') \right]$$




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

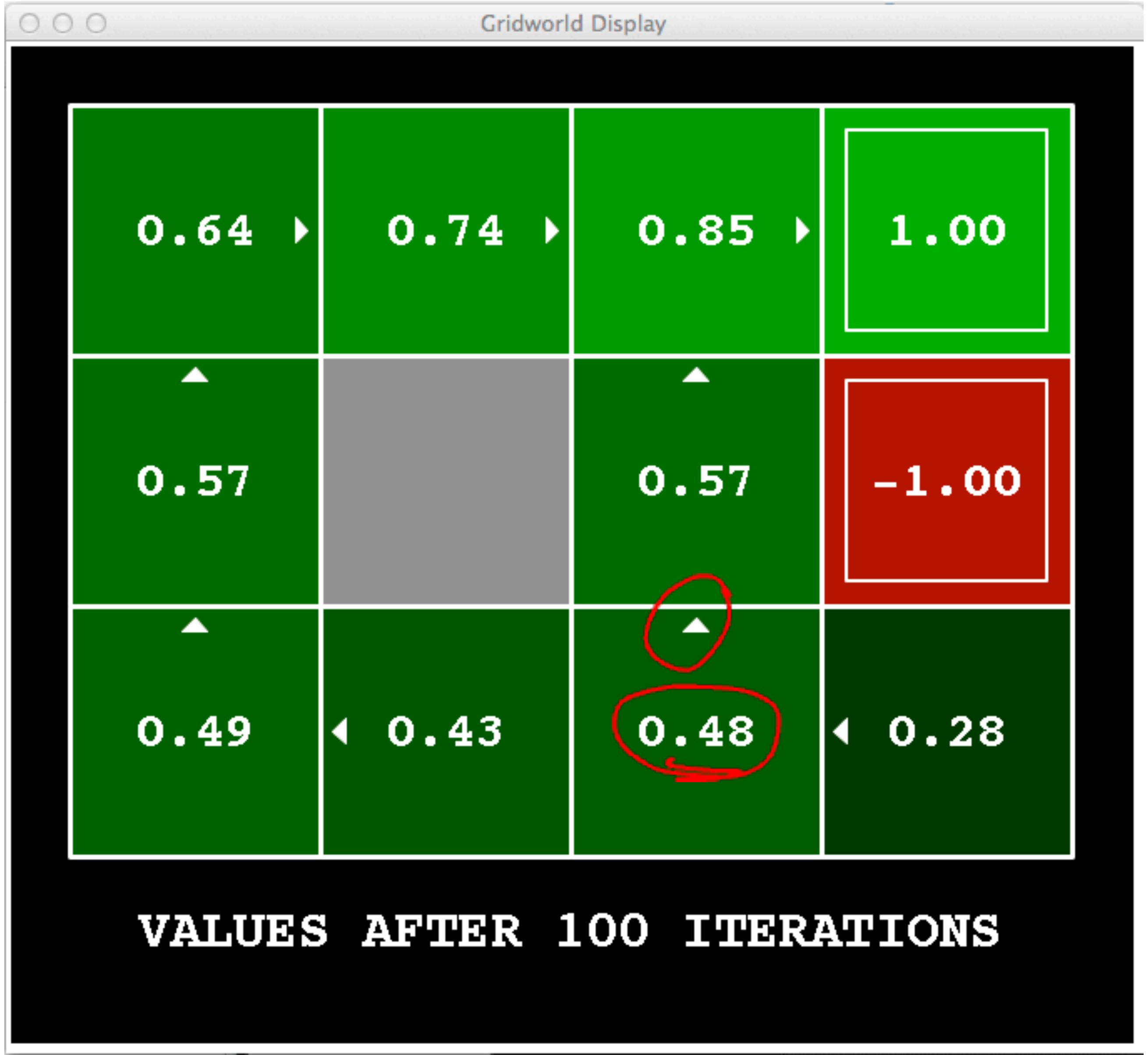
- Problem 2: The "max" at each state rarely changes, policy often converges long before the values

# k=12




Noise = 0.2
Discount = 0.9
Living reward = 0

# k=100

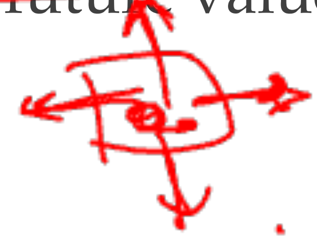



Noise = 0.2
Discount = 0.9
Living reward = 0

# Policy Iteration

fixed policy → $\boxed{\text{P. eval}}$

- 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~~ $\pi_{i+1}$
  - Repeat ~~steps until policy converges~~



- This is policy iteration
  - It's still optimal!
  - Can con~~verge (much) faste~~r under some conditions

iter i,

Step: $\pi_i($ $)$

Step2 : update it

# Policy Iteration

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

  $i+1$ $\pi_i$

  - 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

policy evaluation

- 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)

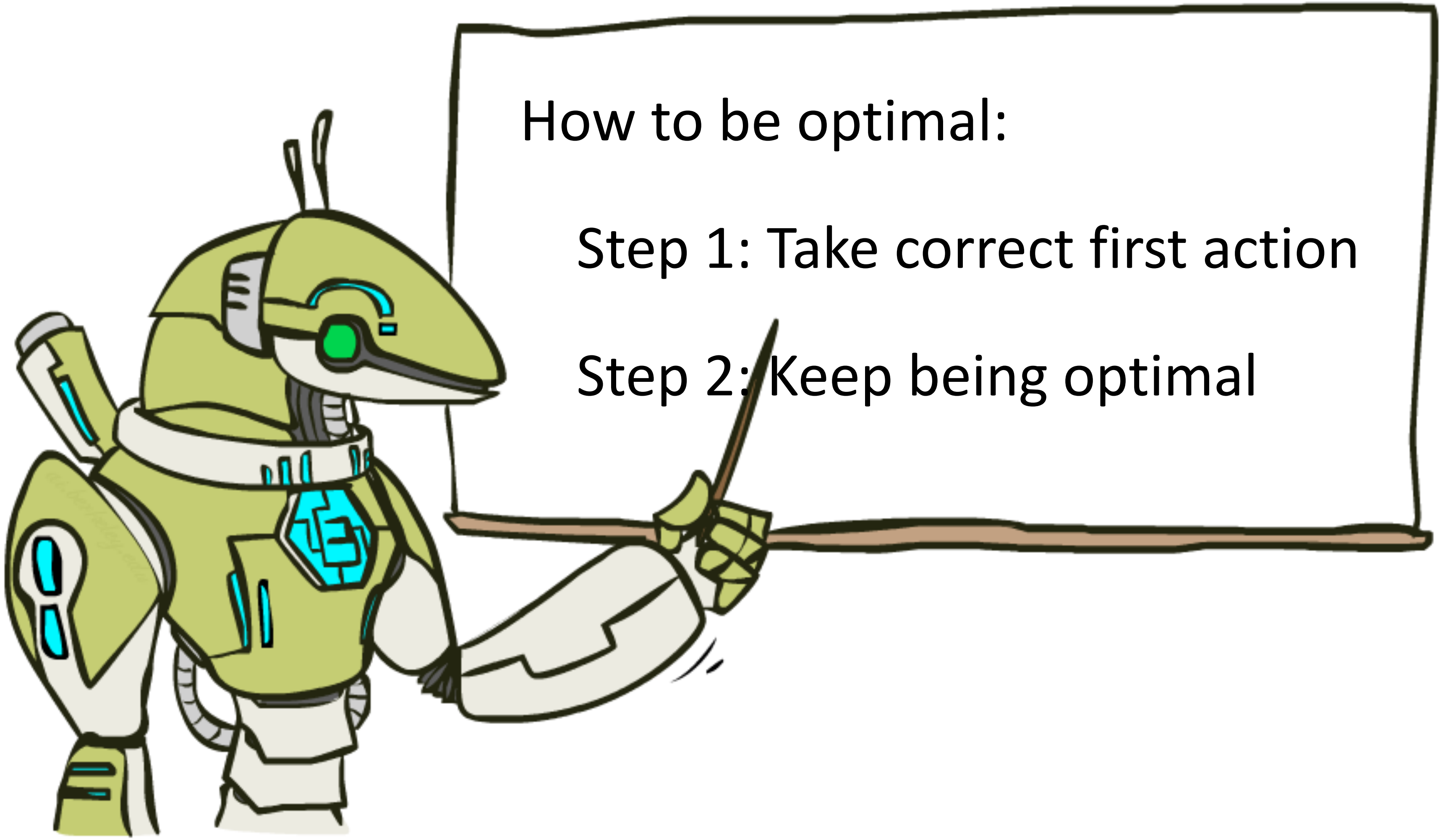
action value

- 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

input/output

$\pi(s) \rightarrow a$

# The Bellman Equations



# Next Topic: Reinforcement Learning!