MECHANISM DESIGN

JOHN P DICKERSON & MARINA KNITTEL

Lecture #19 – 04/04/2022

CMSC498T
Mondays & Wednesdays
2:00pm – 3:15pm

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

ANNOUNCEMENTS

Short project proposals:

- Were due to John before Spring Break
- Still working through them, will send out comments this week.

TURN-BASED STOCHASTIC GAMES (TBSGS)

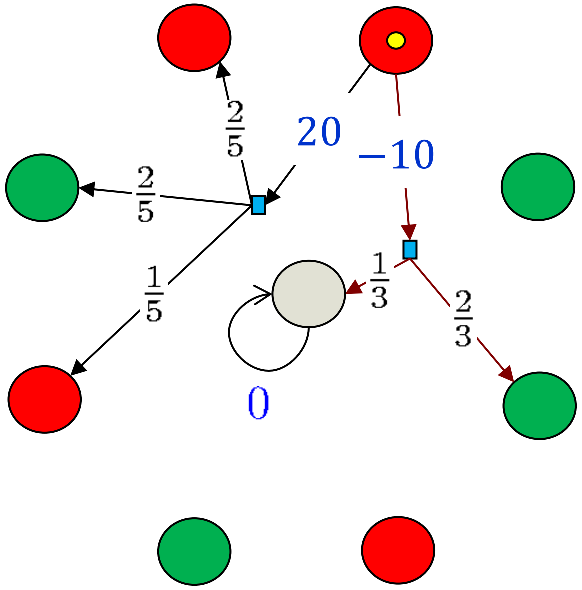
(Possibly) Infinite duration games played by *two* players, min and max, on a *finite* weighted directed graph.

Vertices of the graph divided among min, max and rand.

Edges emanating from min and max vertices, also called *actions*, have *costs* (or *payoff*)

Edges emanating from rand vertices have *probabilities* that sum up to 1.

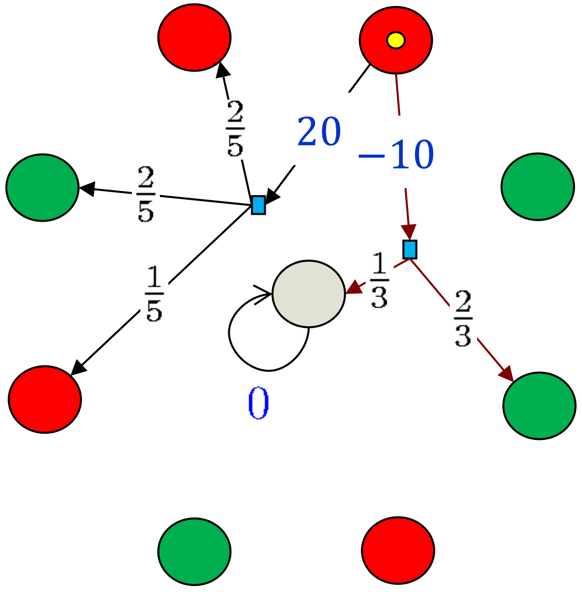
There might be a *sink*, aka, a state with actions that lead only back to itself



TURN-BASED STOCHASTIC GAMES (TBSGS)

Game is played as follows:

- **A token is placed on an initial vertex.**
- **If the token is in a min/max vertex, then min/max chooses an action.**
- **If the token is in a rand vertex, a random choice is made.**
- **If the token reaches a sink, the game ends.**
- **The result is an (infinite) sequence of costs (or rewards):** $c_0, c_1, c_2, \ldots$



TURN-BASED STOCHASTIC GAMES (TBSGS)

Objective function?

Total cost – finite horizon

$$\min/\max \ \mathbb{E}\left[\sum_{i=0}^{T} c_i\right]$$

Total cost – infinite horizon

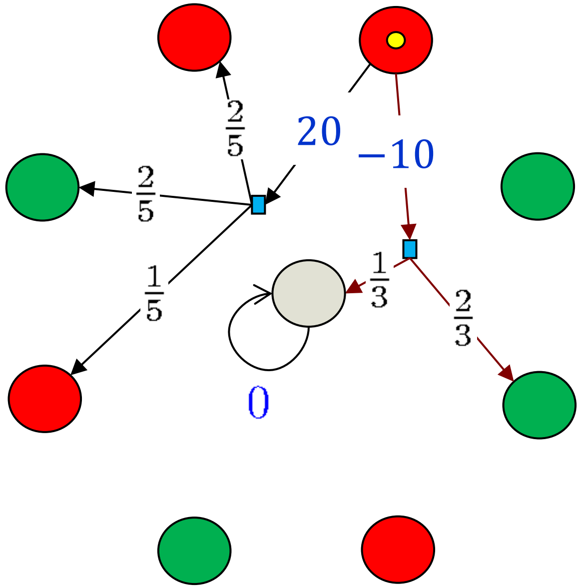
$$\min/\max \ \mathbb{E}\left[\sum_{i=0}^{\infty} c_i\right]$$

Discounted cost

$$\min/\max \ \mathbb{E}\left[\sum_{i=0}^{\infty} \lambda^i c_i\right]$$

Limiting ~~average~~ cost

$$\min/\max \ \mathbb{E}\left[\lim_{T\to\infty} \frac{1}{T} \sum_{i=0}^{T-1} c_i\right]$$



BACKGAMMON

Backgammon is a TBSG.

Author: Ptkfgs
Wikimedia Commons

A single cost/reward of $+1$ or -1 in the last move.

A huge number of vertices/states.

Stochasticity: dice rolls dictate legal states

Think back: can it be solved in polynomial time in the number of states ??????????

CHESS

Attribution: Bubba73
at English Wikipedia

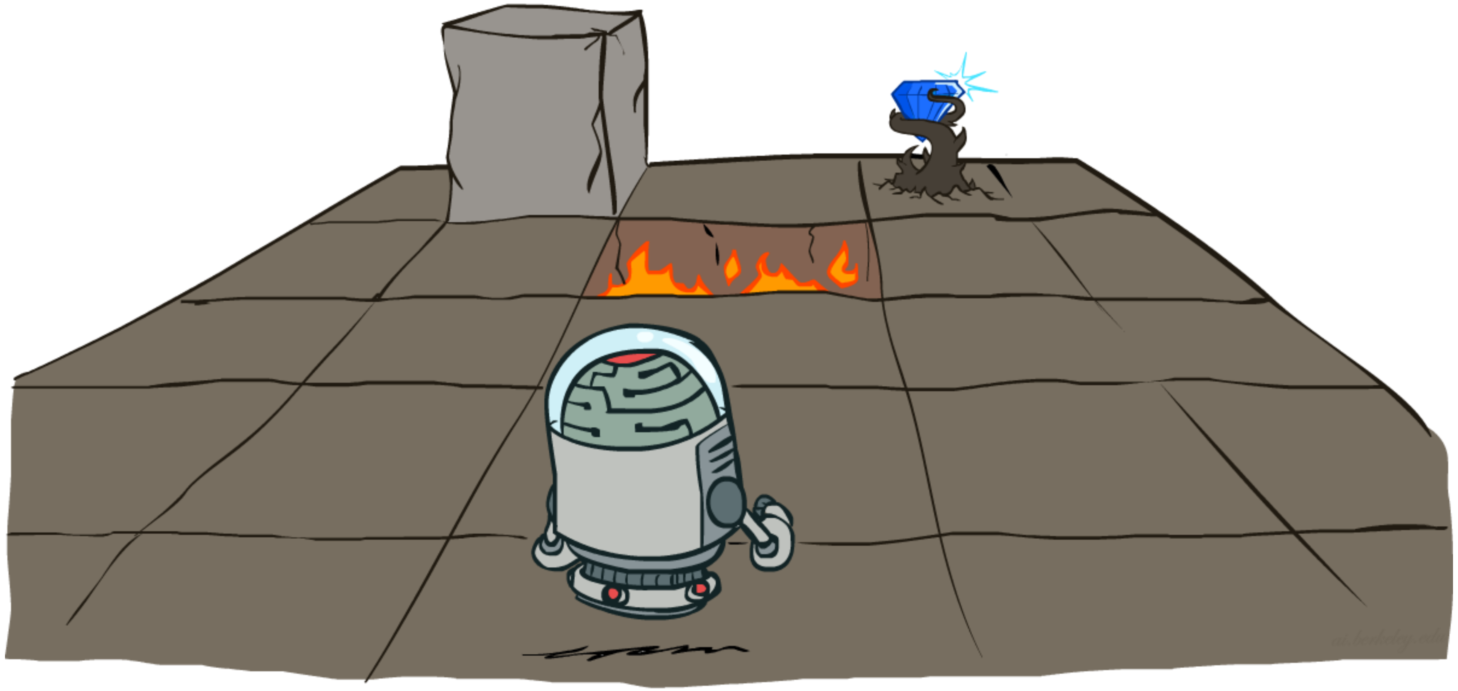
Chess is a deterministic TBSG.

A single cost/reward of -1, 0 or $+1$ in the last move.

Finite? "Threefold repetition of position"

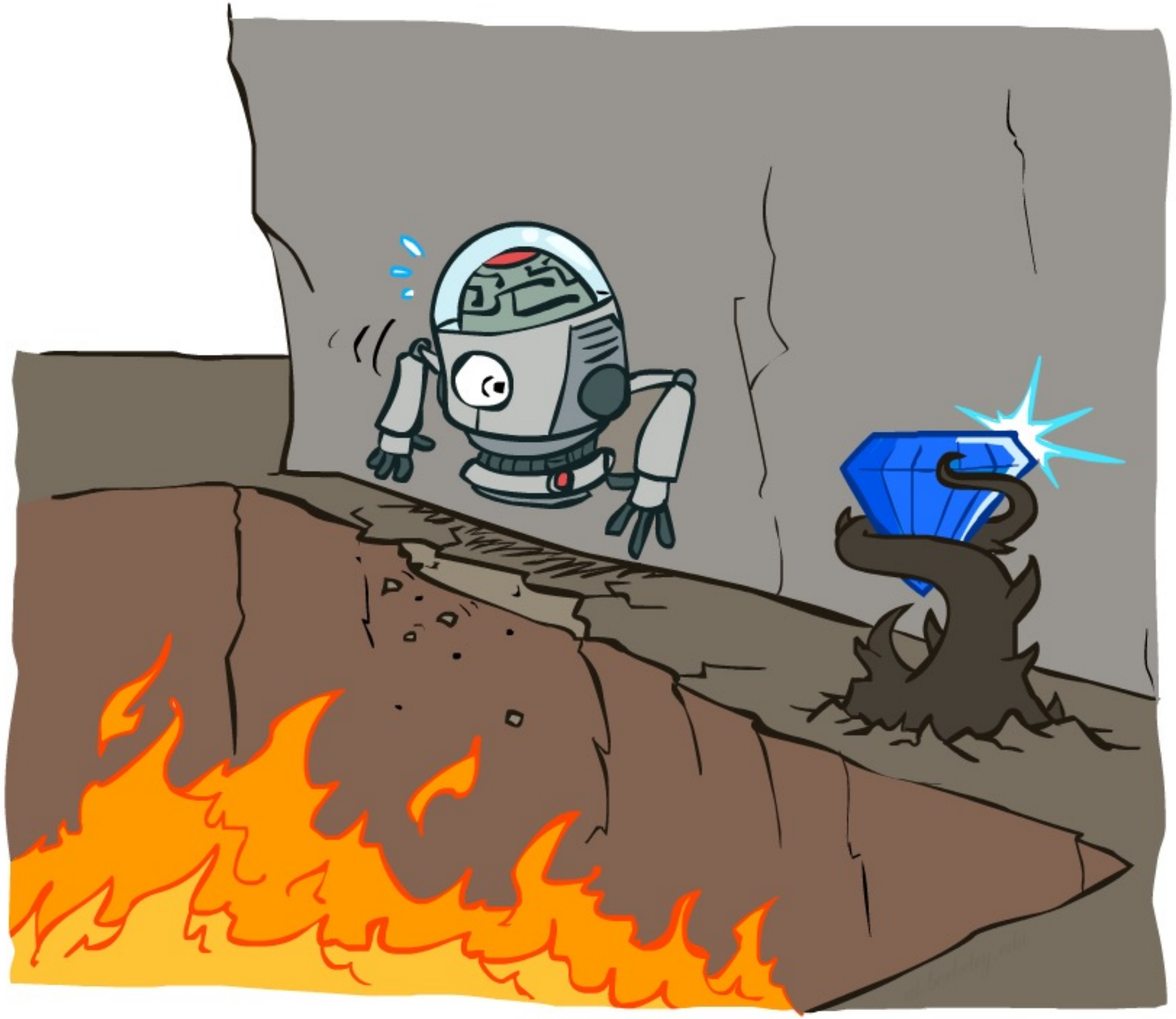
LET'S START EVEN EASIER …

ONE-PLAYER STOCHASTIC GAMES
AKA
MARKOV DECISION PROCESSES

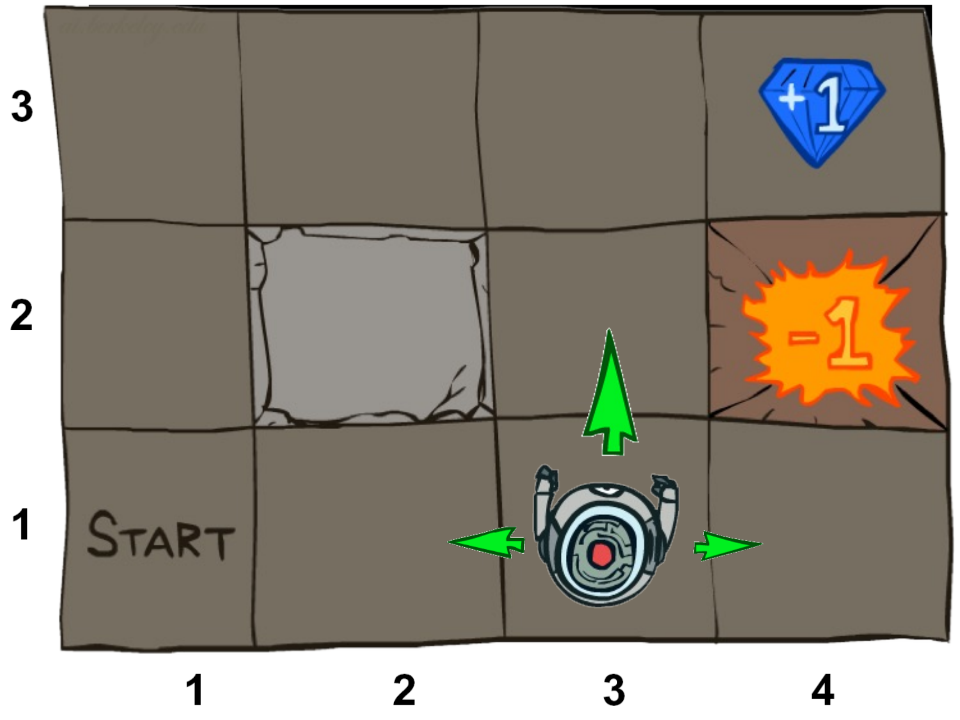


Slide credits: CMU AI and http://ai.berkeley.edu

NON-DETERMINISTIC SEARCH



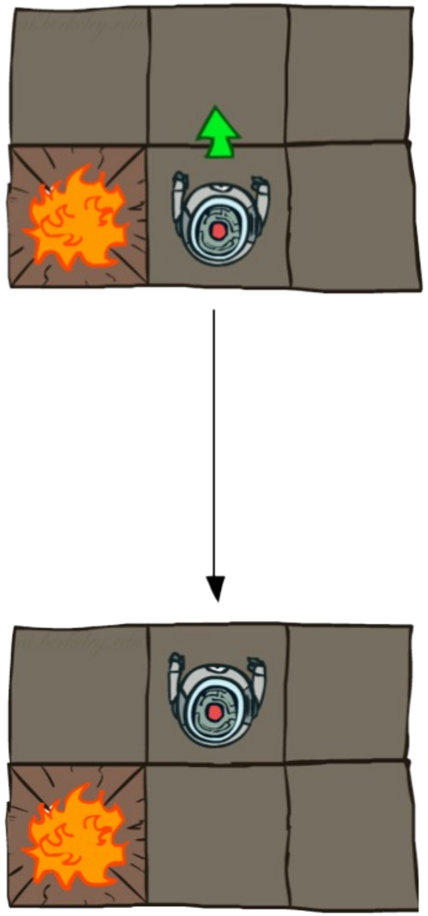
EXAMPLE: GRID WORLD



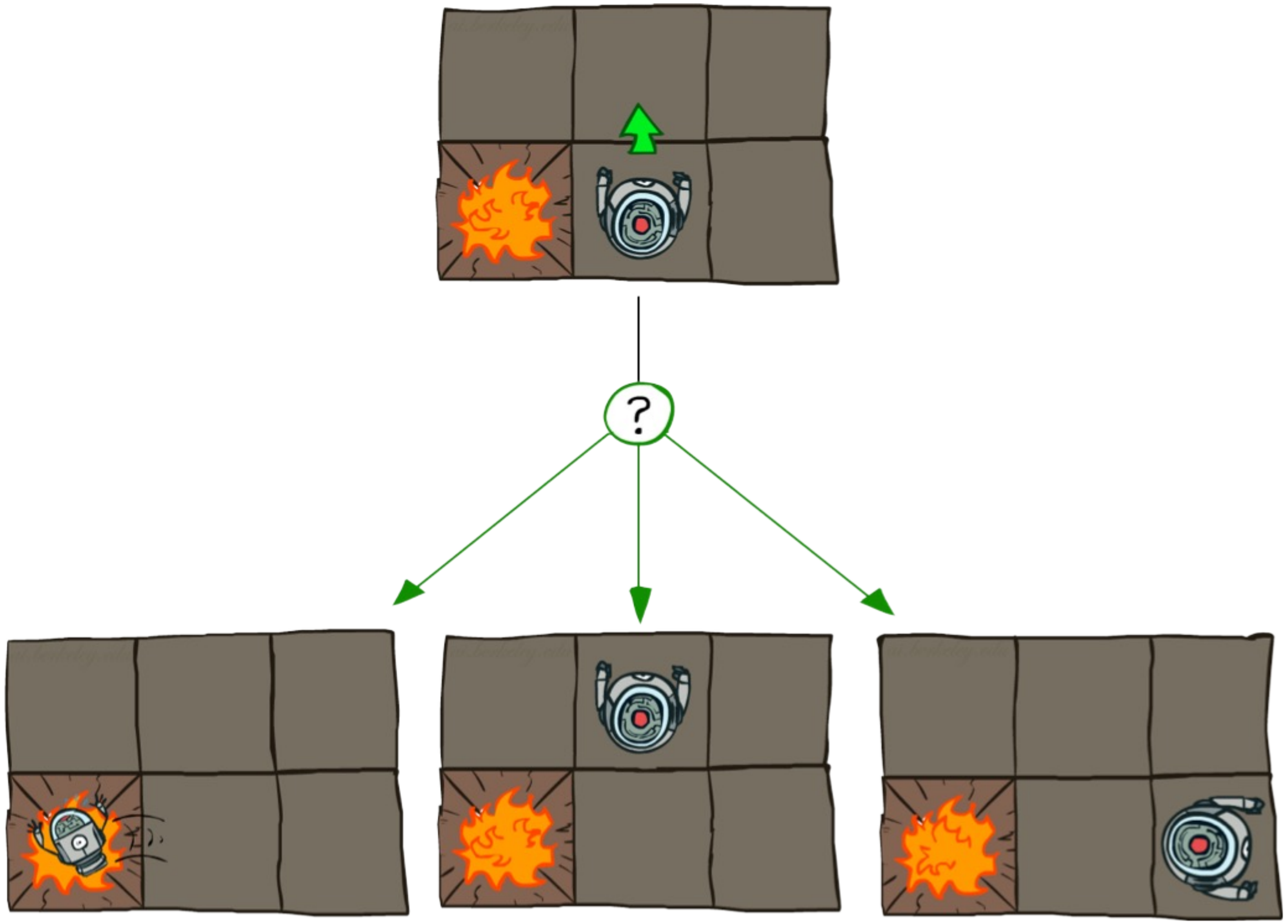
- A maze-like problem
  - The agent lives in a grid
  - Walls block the agent's path
- Noisy movement: actions do not always go as planned
  - If agent takes action North
    - 80% of the time: Get to the cell on the North (if there is no wall there)
    - 10%: West; 10%: East
  - If path after roll dice blocked by wall, stays put
- The agent receives rewards each time step
  - "Living" reward (can be negative)
  - Additional reward at pit or target (good or bad) and will exit the grid world afterward
- Goal: maximize sum of rewards

GRID WORLD ACTIONS

Deterministic Grid World



Stochastic Grid World

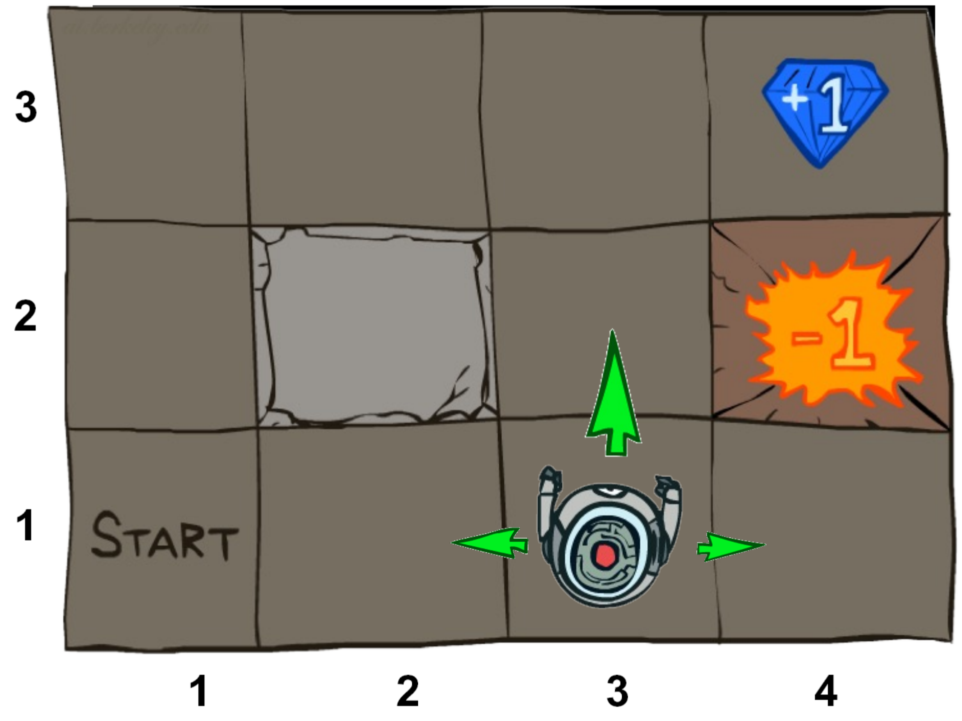


MARKOV DECISION PROCESS (MDP)

An MDP is defined by a tuple (S,A,T,R):

- S: a set of states
- A: a set of actions
- T: a transition function
  - T(s, a, s') where s ∈ S, a ∈ A, s' ∈ S is P(s'| s, a)
- R: a reward function
  - R(s, a, s') is reward at this time step
  - Sometimes just R(s) or R(s')

- Sometimes also have
  - γ: discount factor (introduced later)
  - μ: distribution of initial state (or just start state s_0)
  - Terminal states: processes end after reaching these states



The Grid World problem as an MDP

$\boxed{R(s_{4,2}, exit, s_{virtual\_terminal})=-1}$

R($s_{4,2}$)=-1, no virtual terminal state

How to define the terminal states & reward function for the Grid World problem?

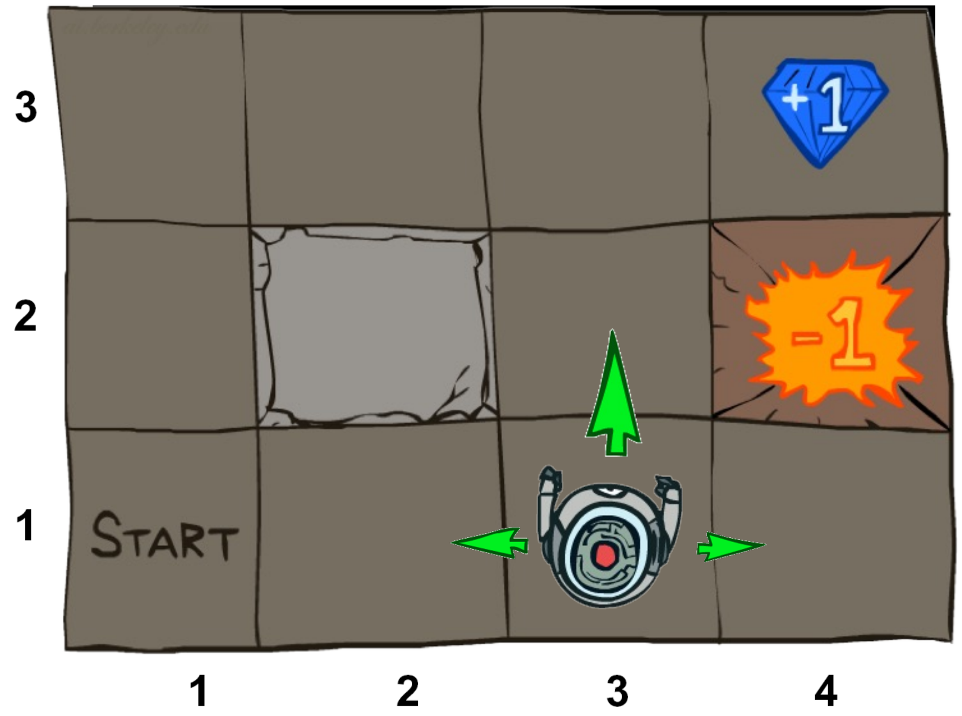
MARKOV DECISION PROCESS (MDP)

An MDP is defined by a tuple (S,A,T,R)

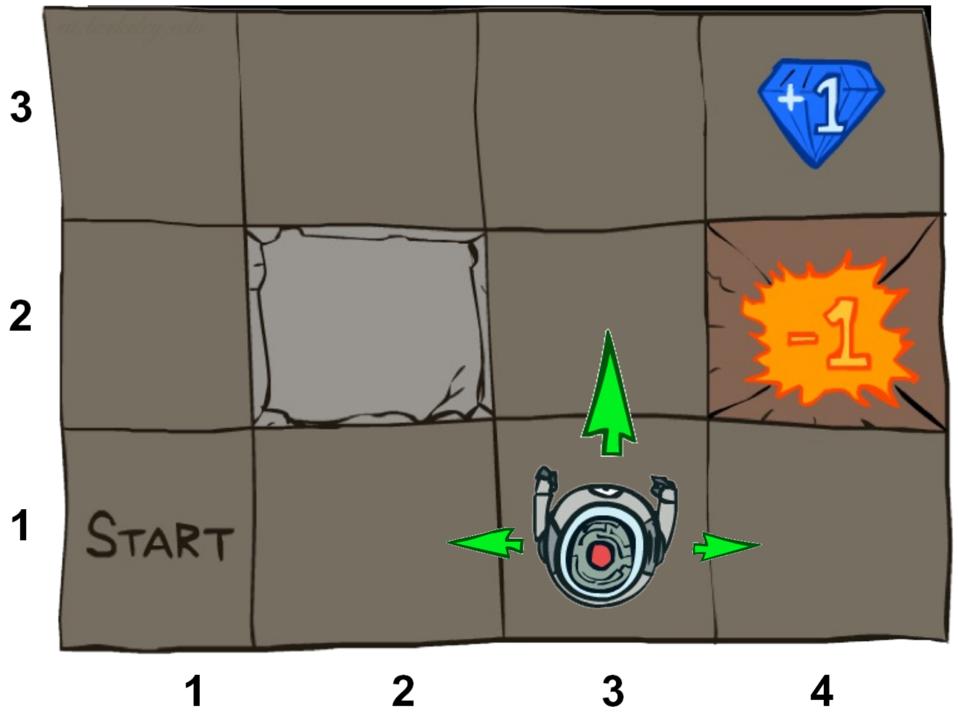
Why is it called Markov Decision Process?

Decision:

Process:



MARKOV DECISION PROCESS (MDP)



An MDP is defined by a tuple (S,A,T,R)

Why is it called Markov Decision Process?

Decision:

Agent decides what action to take at each time step

Process:

The system (environment + agent) is changing over time

WHAT IS "MARKOVIAN" ABOUT MDPS?

Markov property: Conditional on the present state, the future and the past are independent

With respect to MDPs, it means outcome of an action depend only on current state

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

$$=$$

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

Andrey Markov
(1856-1922)
Russian
mathematician

POLICIES

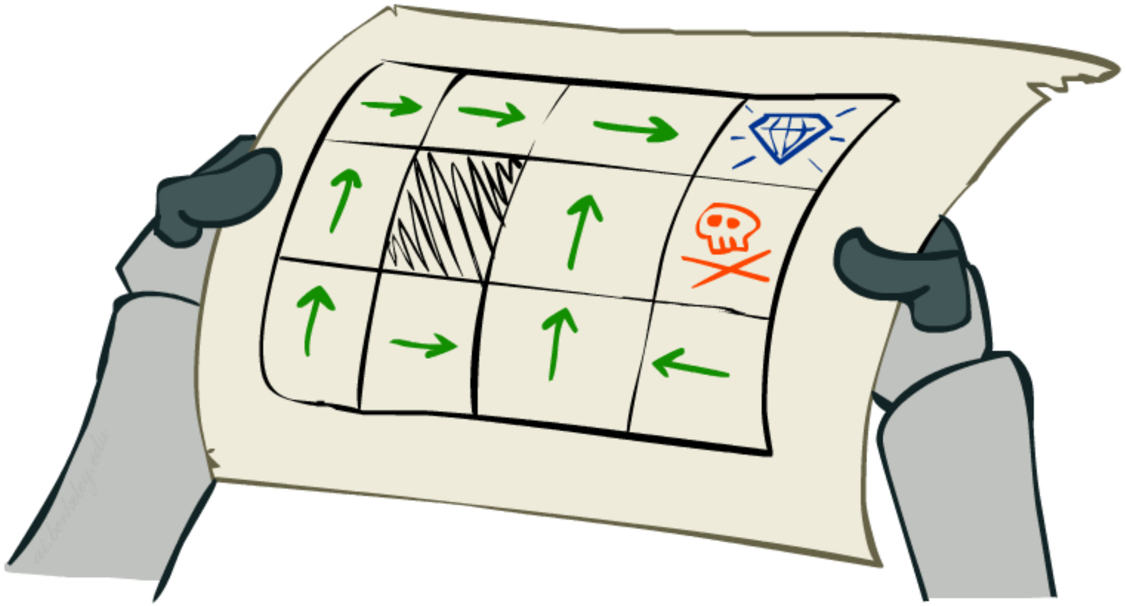
In deterministic single-agent search problems, w
sequence of actions, from start to a goal

For MDPs, we focus on policies

- Policy = map of states to actions
- π(s) gives an action for state s

We want an optimal policy π^*: S → A

- An optimal policy is one that maximizes expected utility if followed



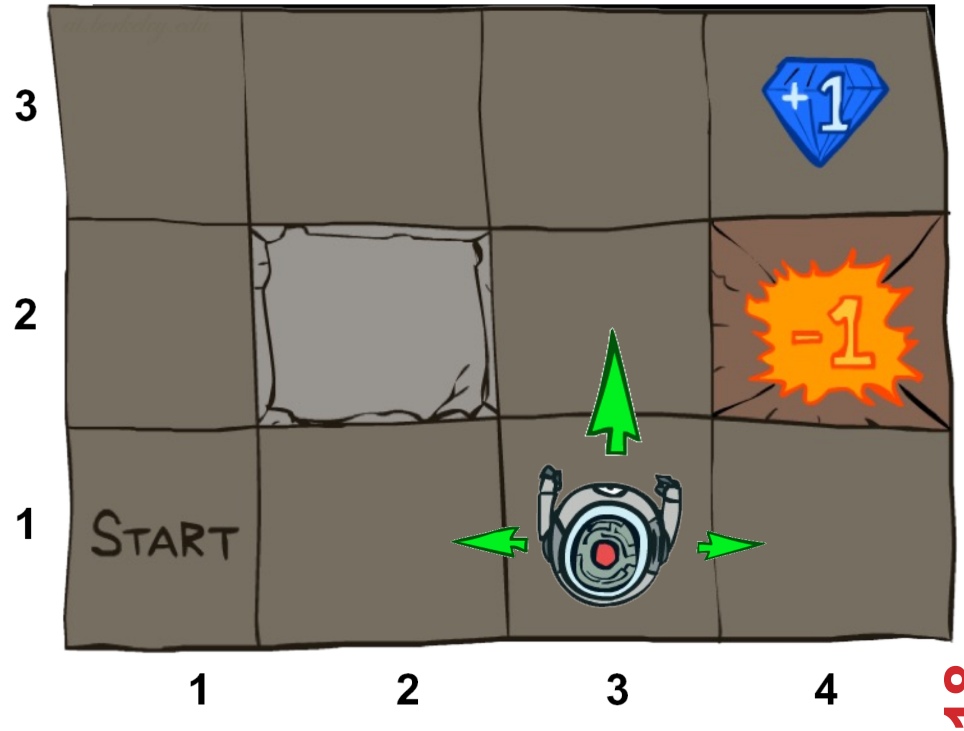
POLICIES

Recall: An MDP is defined S,A,T,R

Keep S,A,T fixed, optimal policy may vary given different R

What is the optimal policy if R(s,a,s')=-1000 for all states other than pit and target?

What is the optimal policy if R(s,a,s')=0 for all states other than pit and target, and reward=1000 and -1000 at pit and target respectively?

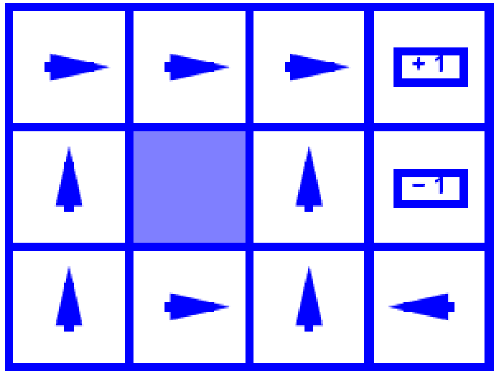


DISCUSSION POINT!

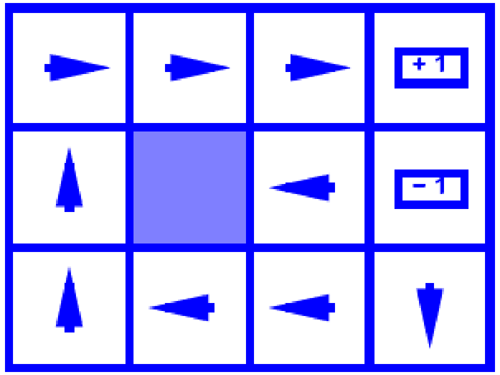
{A, B, C, D} are optimal policies for one of each of the following "reward for living" scenarios: {-0.01, -0.03, -0.04, -2.0}. Which policy maps to which reward setting?

I. {B, A, C, D}
II. {B, C, A, D}
III. {C, B, A, D}
IV. {D, A, C, B}

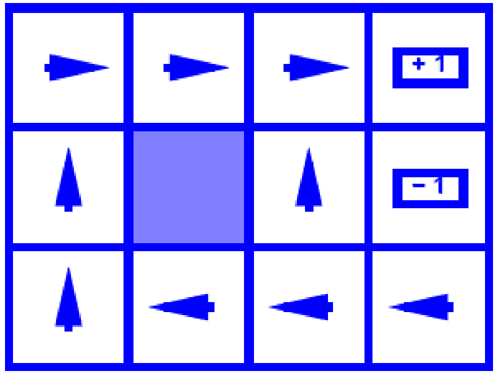
A)



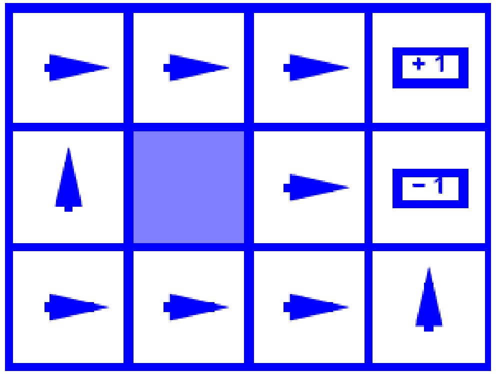
B)



C)



D)

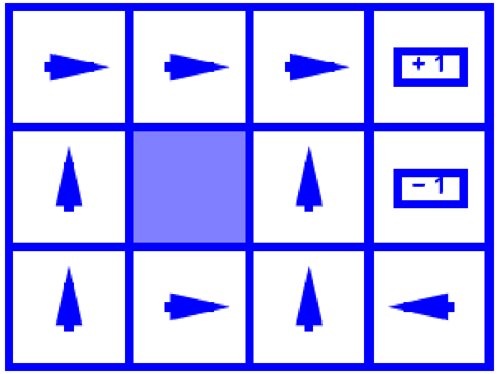


DISCUSSION POINT!

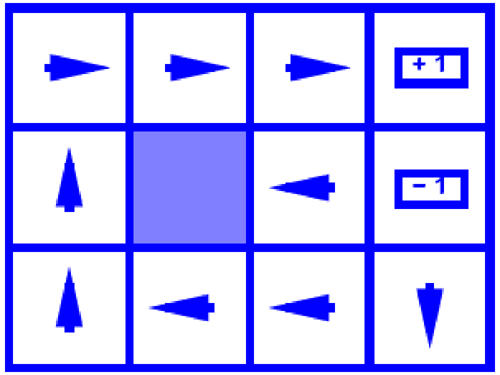
{A, B, C, D} are optimal policies for one of each of the following "reward for living" scenarios: {-0.01, -0.03, -0.04, -2.0}.
Which policy maps to which reward setting?

I. {B, A, C, D}
II. {B, C, A, D}
III. {C, B, A, D}
IV. {D, A, C, B}

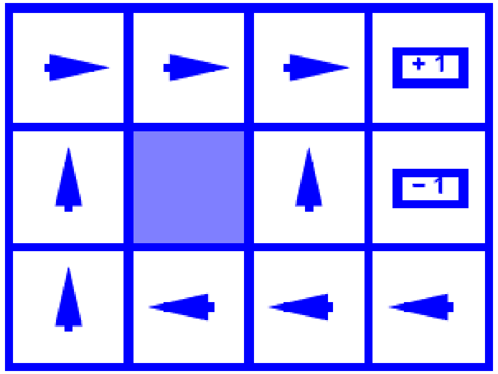
A)



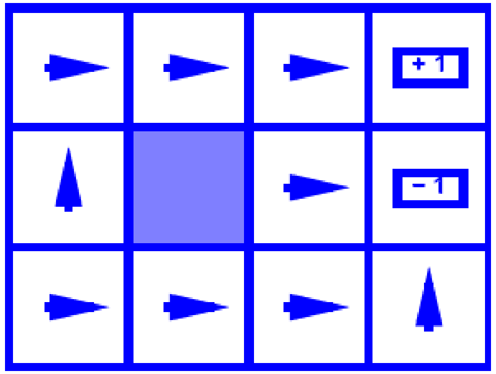
B)



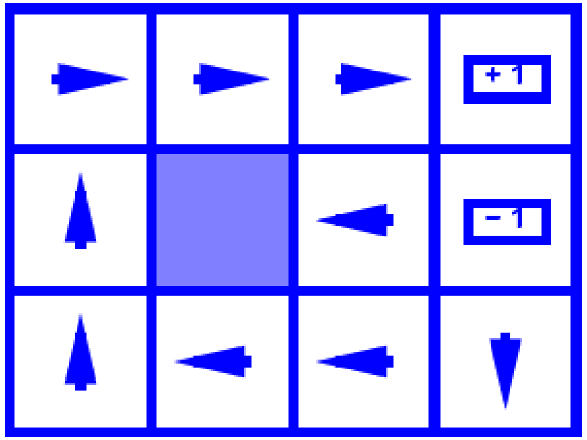
C)



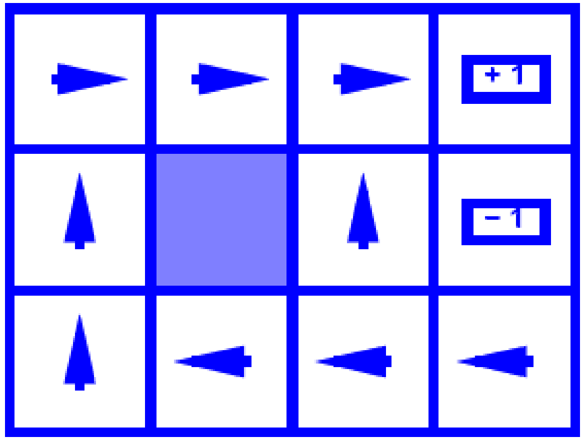
D)



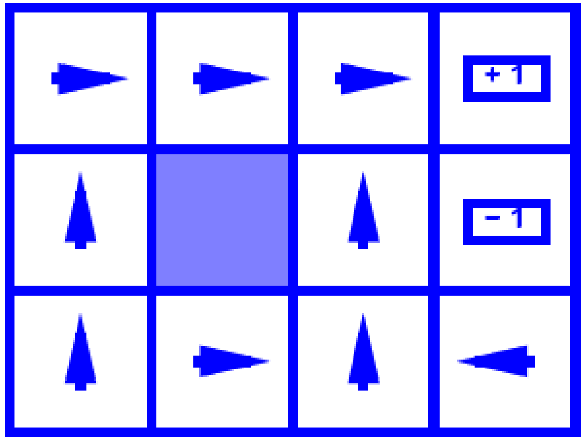
DISCUSSION POINT! POLICIES



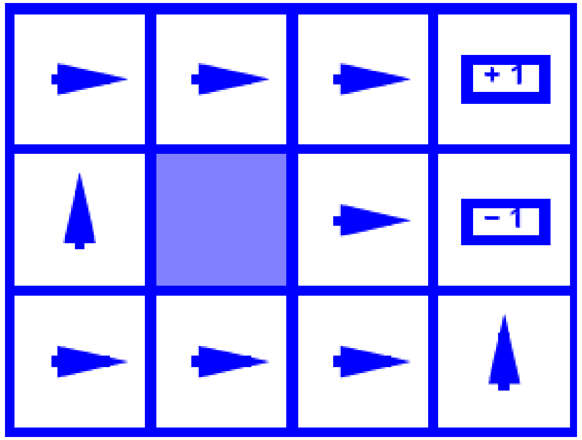
R(s) = -0.01



R(s) = -0.03

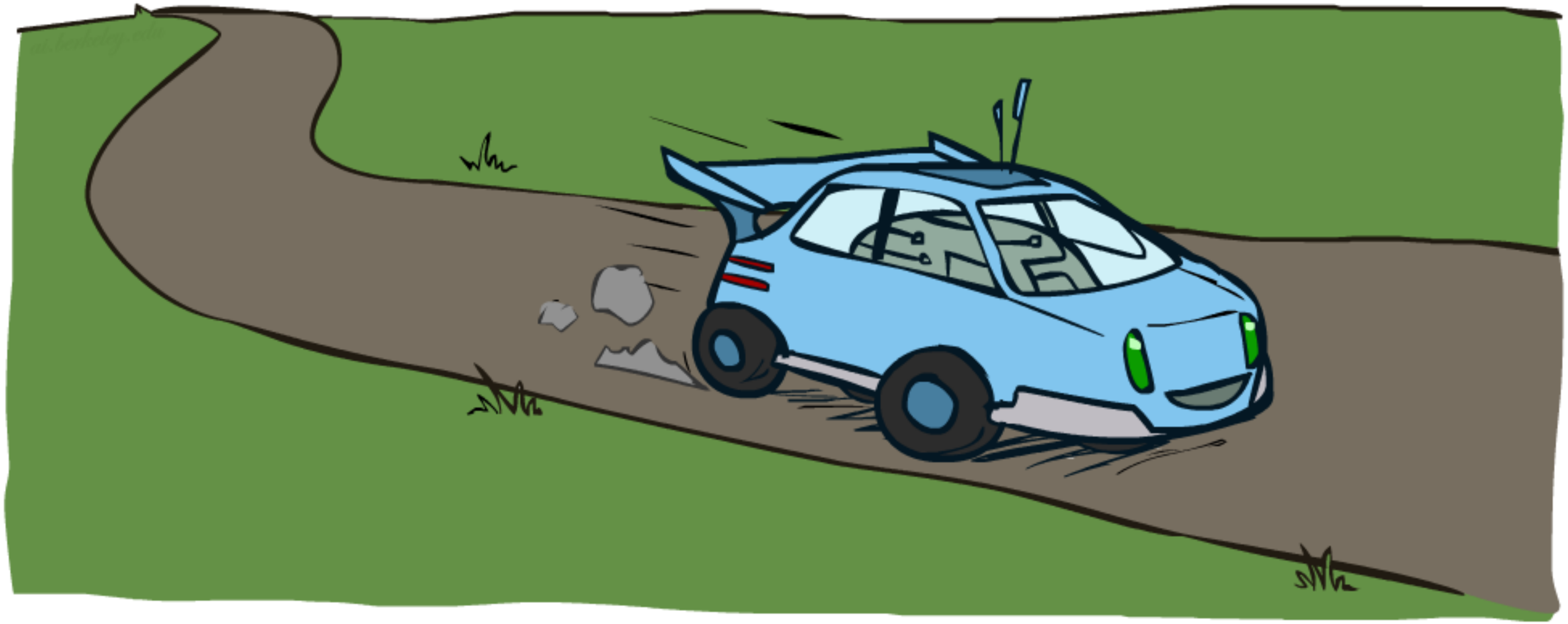


R(s) = -0.4



R(s) = -2.0

EXAMPLE: RACING



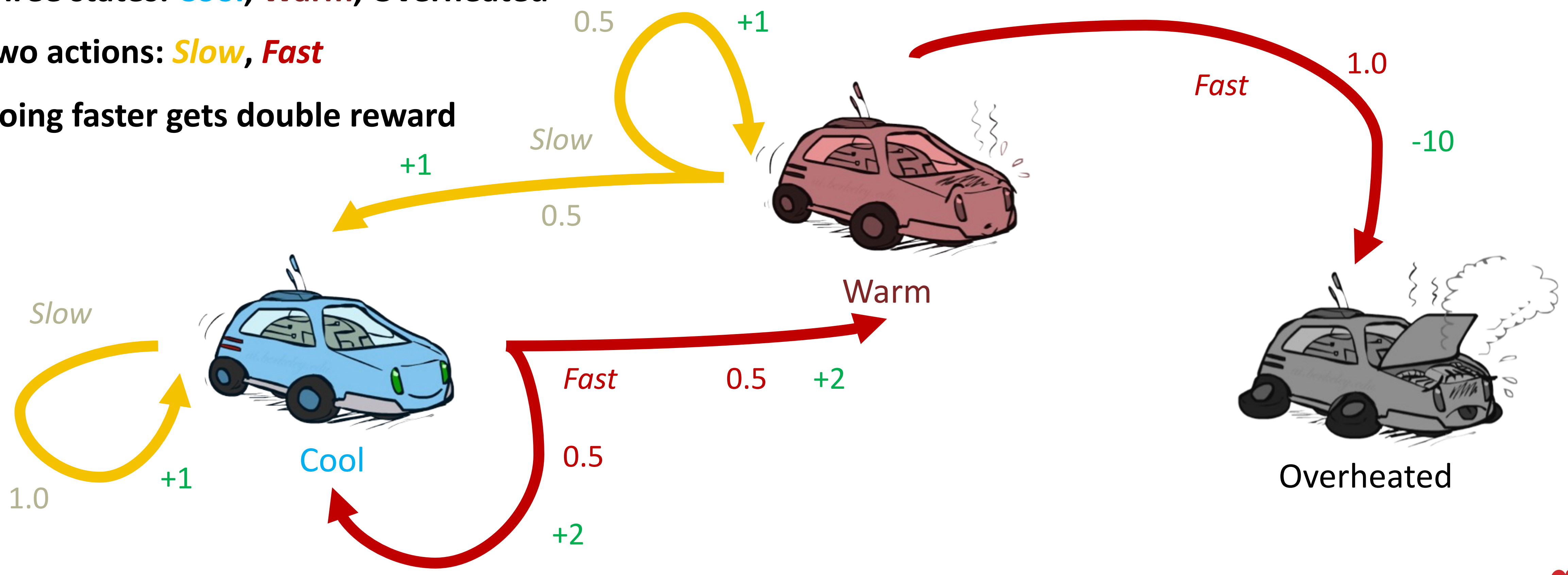
EXAMPLE: RACING

A robot car wants to travel far, quickly

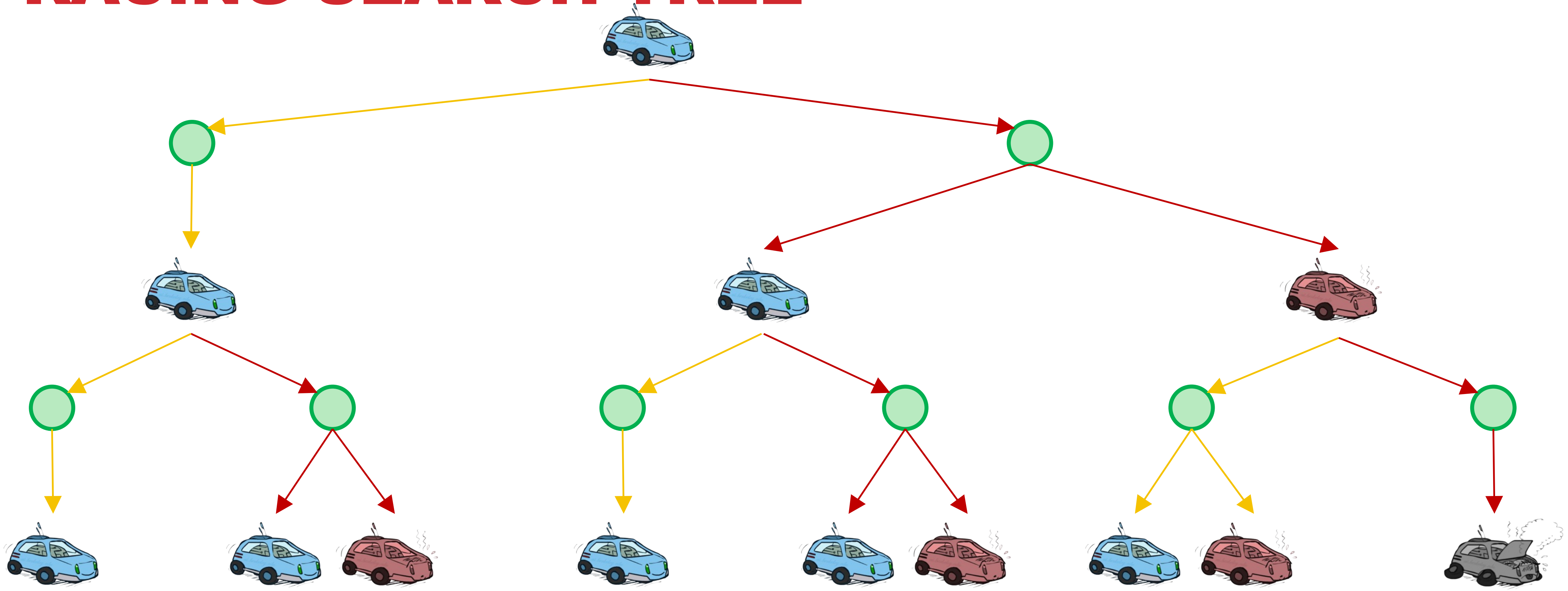
Three states: Cool, Warm, Overheated

Two actions: *Slow*, ***Fast***

Going faster gets double reward

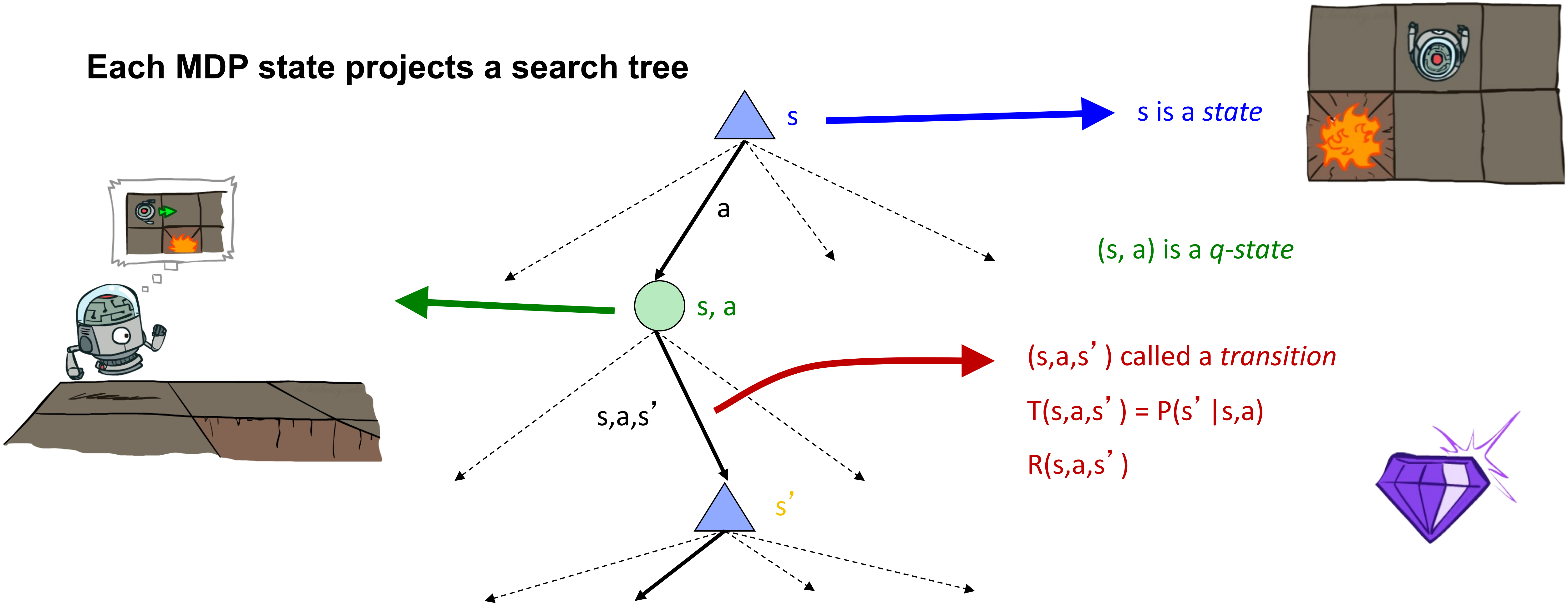


RACING SEARCH TREE

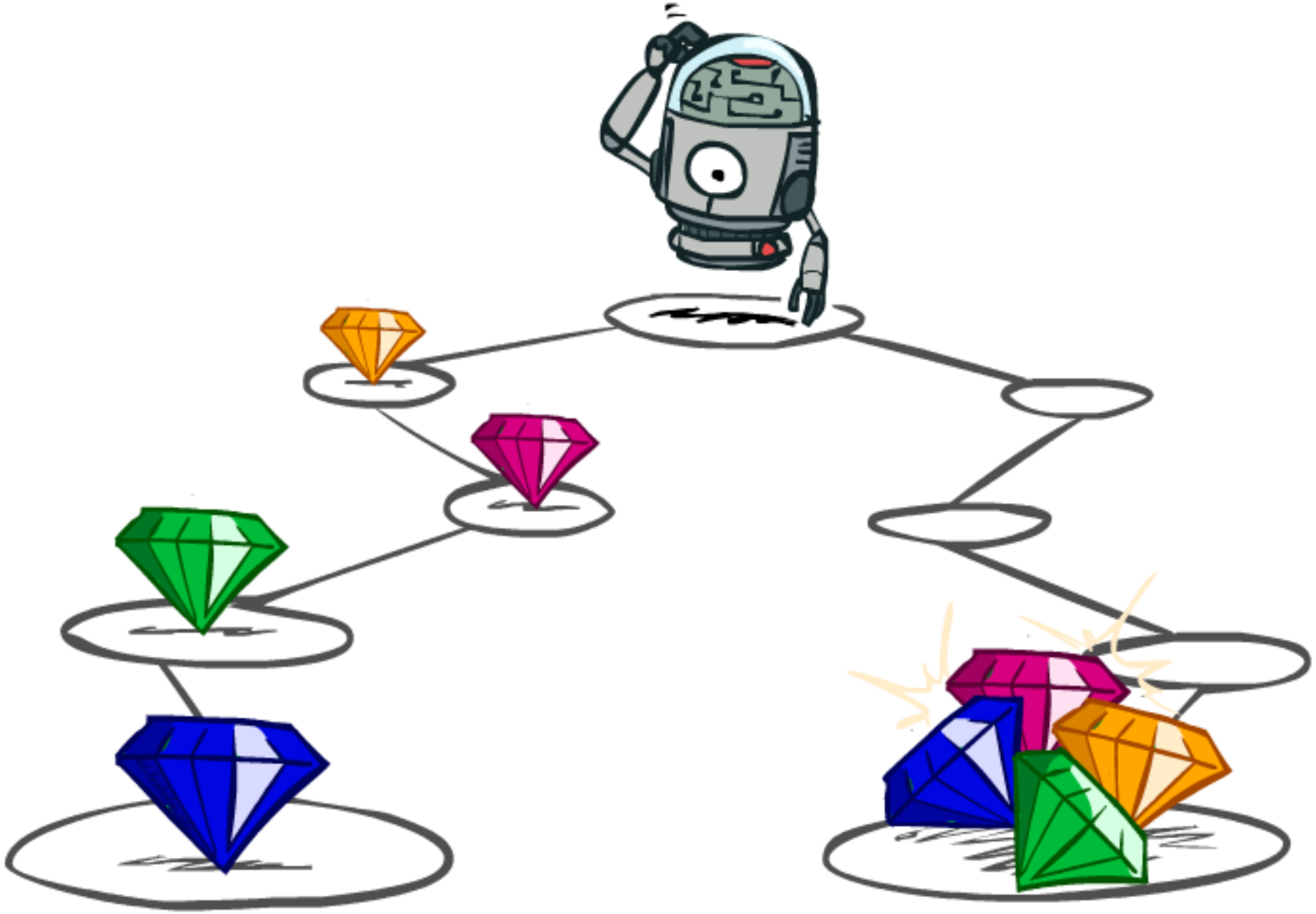


MDP SEARCH TREES

Each MDP state projects a search tree



UTILITIES OF SEQUENCES

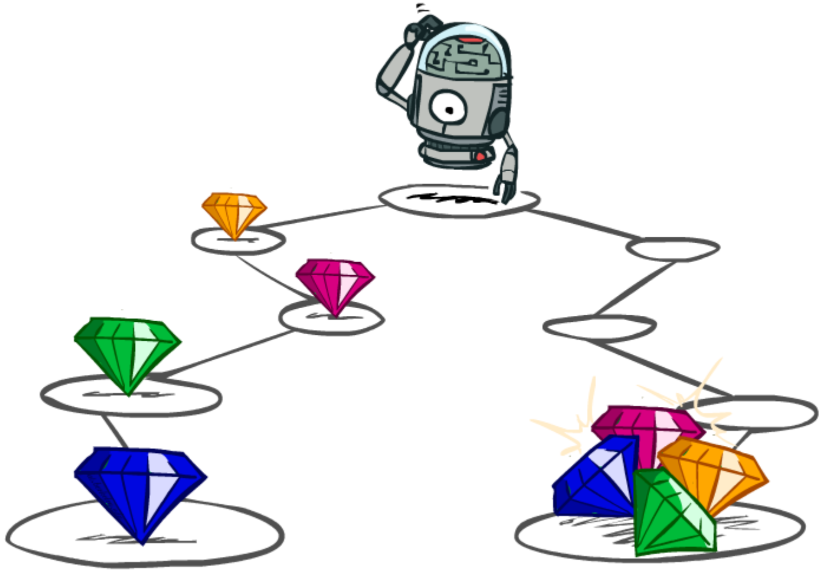


UTILITIES OF SEQUENCES

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: utility of rewards decay exponentially

1

Worth Now

γ

Worth Next Step

γ^2

Worth In Two Steps

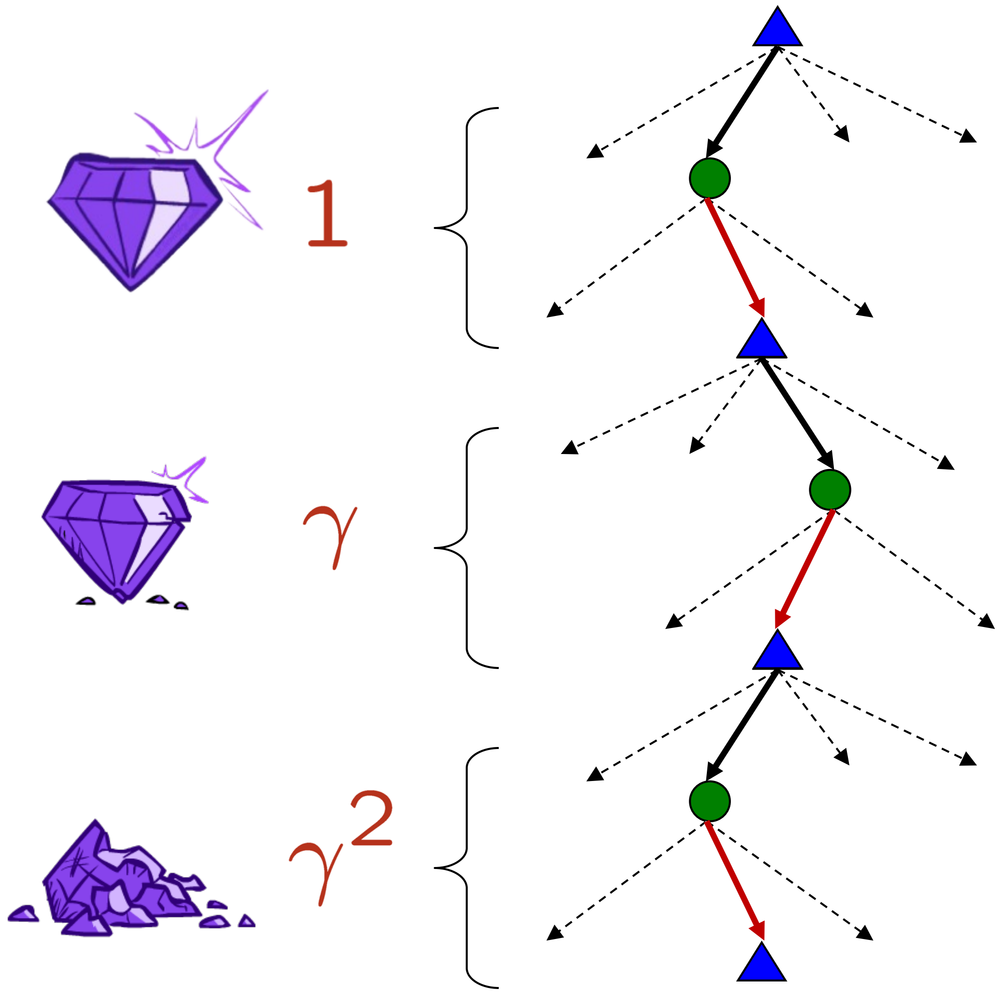
DISCOUNTING

How to discount?

- Each time we descend a level, we multiply in the discount once

Why discount?

- Sooner rewards probably do have higher utility than later rewards
- Also helps our algorithms converge



DISCUSSION POINT!

What is the value of U([2,4,8]) with $\gamma = 0.5$????????????????

(U(·) is the total utility of a reward sequence.)

- 3
- 6
- 7
- 14

Bonus: What is the value of U([8,4,2]) with $\boldsymbol{\gamma = 0.5}$?????????????

DISCUSSION POINT!

What is the value of U([2,4,8]) with $\gamma = 0.5$????????????????

(U(·) is the total utility of a reward sequence.)

- 3
- 6
- 7
- 14

$$\gamma^0 \times 2 + \gamma^1 \times 4 + \gamma^2 \times 8 = 2 + 0.5 \times 4 + 0.5 \times 0.5 \times 8 = 2 + 2 + 2 = 6$$

Bonus: What is the value of U([8,4,2]) with $\boldsymbol{\gamma = 0.5}$?????????????

$$\gamma^0 \times 8 + \gamma^1 \times 4 + \gamma^2 \times 2 = 8 + 0.5 \times 4 + 0.5 \times 0.5 \times 2 = 8 + 2 + 0.5 = 10.5$$

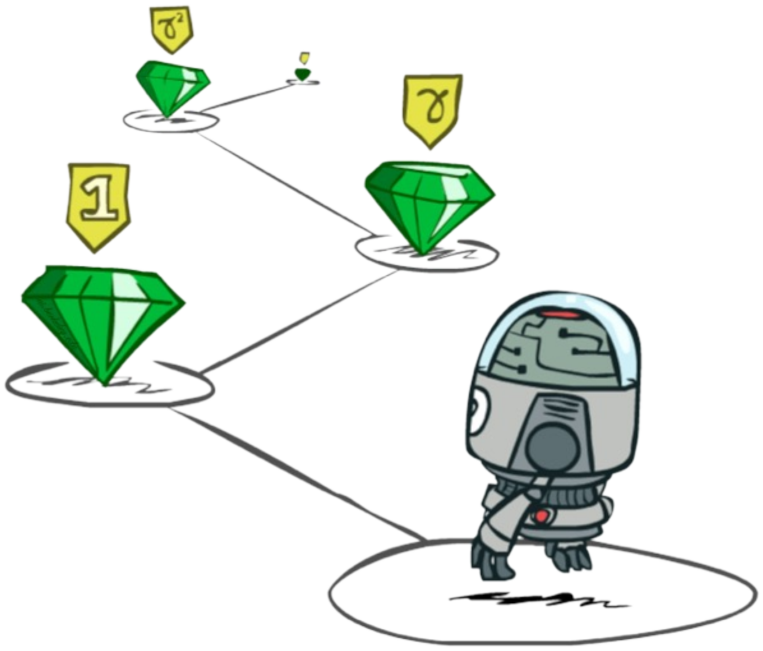
STATIONARY PREFERENCES

Theorem: if we assume stationary preferences:

$$[a_1, a_2, \ldots] \succ [b_1, b_2, \ldots]$$

$$\Updownarrow$$

$$[r, a_1, a_2, \ldots] \succ [r, b_1, b_2, \ldots]$$

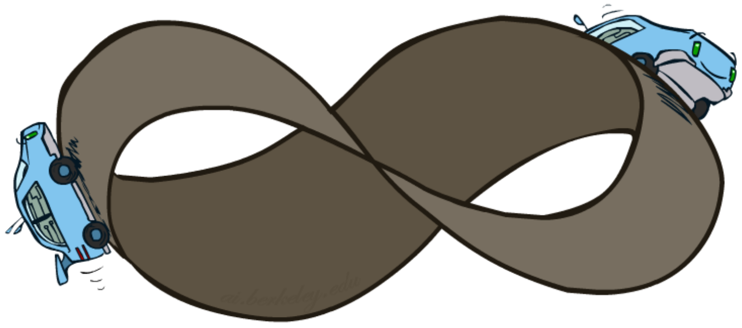


Then: there are only two ways to define utilities

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

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 (π depends on time left)

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

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

  - Smaller γ means smaller "horizon" – shorter term focus

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

OPTIMAL POLICY WITH DISCOUNTING

Given

10				1
a	b	c	d	e

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

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

10				1

OPTIMAL POLICY WITH DISCOUNTING

Given

10				1
a	b	c	d	e

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

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

10	←	←	←	1

OPTIMAL POLICY WITH DISCOUNTING

Given

10				1
a	b	c	d	e

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

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

10	←	←	←	1

For $\gamma = 0.1$, what is the optimal policy?

10				1

OPTIMAL POLICY WITH DISCOUNTING

Given

10				1
a	b	c	d	e

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

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

10	←	←	←	1

For $\gamma = 0.1$, what is the optimal policy?

10	←	←	→	1

OPTIMAL POLICY WITH DISCOUNTING

Given

10				1
a	b	c	d	e

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

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

10	←	←	←	1

For $\gamma = 0.1$, what is the optimal policy?

10	←	←	→	1

For which γ are West and East equally good when in state d?

OPTIMAL POLICY WITH DISCOUNTING

Given

a	b	c	d	e
10				1

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

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

10	←	←	←	1

For $\gamma = 0.1$, what is the optimal policy?

10	←	←	→	1

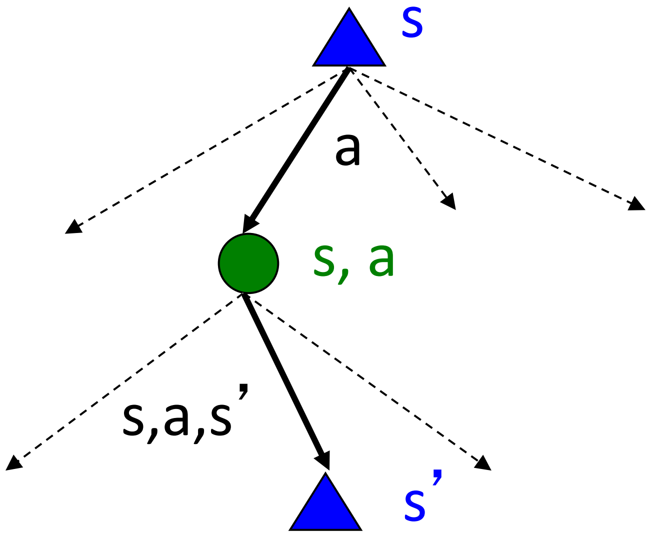
For which γ are West and East equally good when in state d?

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

MDP QUANTITIES (SO FAR!)

Markov decision processes:

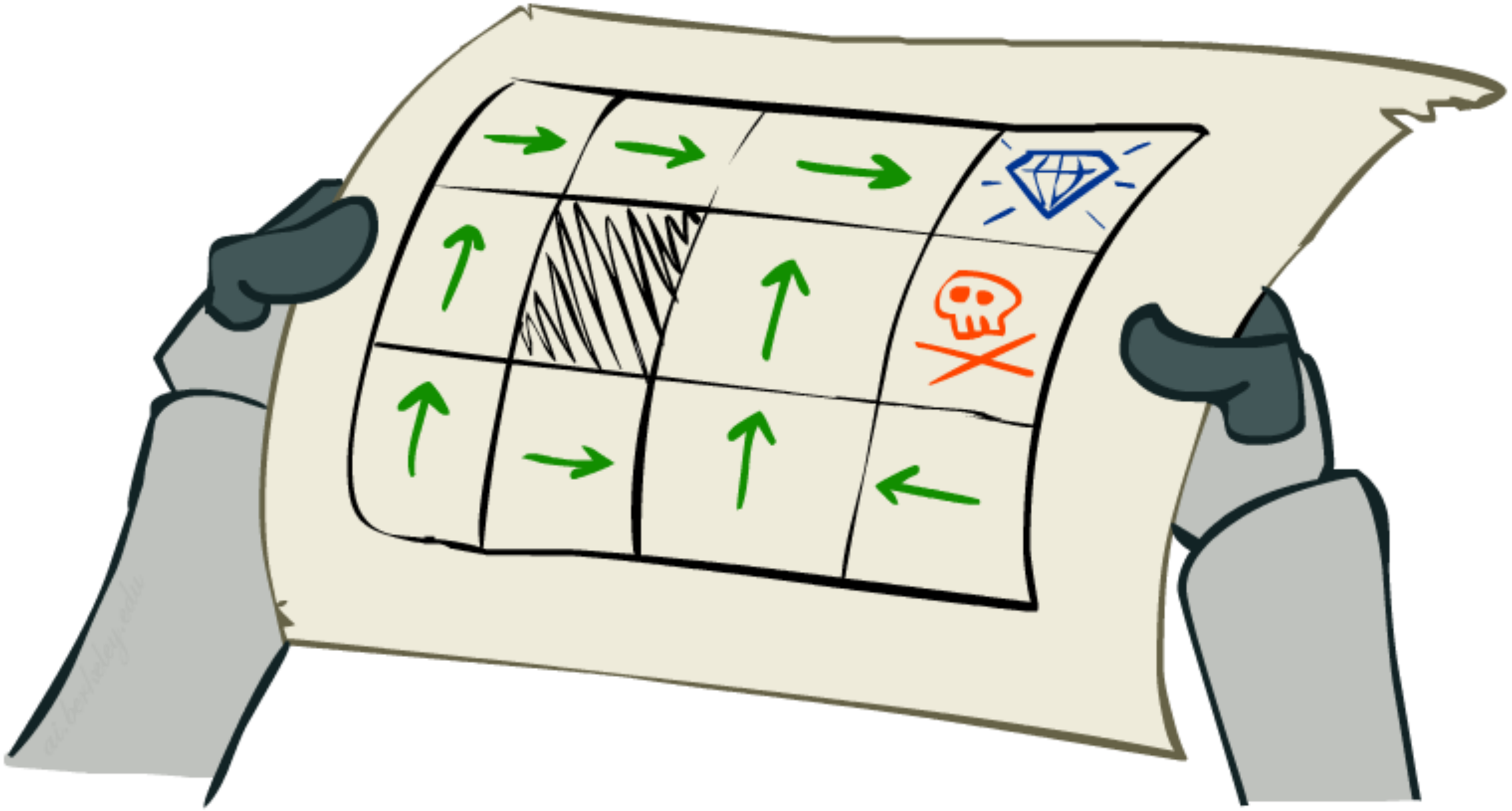
- States S
- Actions A
- Transitions P(s'|s,a) (or T(s,a,s'))
- Rewards R(s,a,s') (and discount γ)
- Start state s0



MDP quantities so far:

- Policy = map of states to actions
- Utility = sum of (discounted) rewards

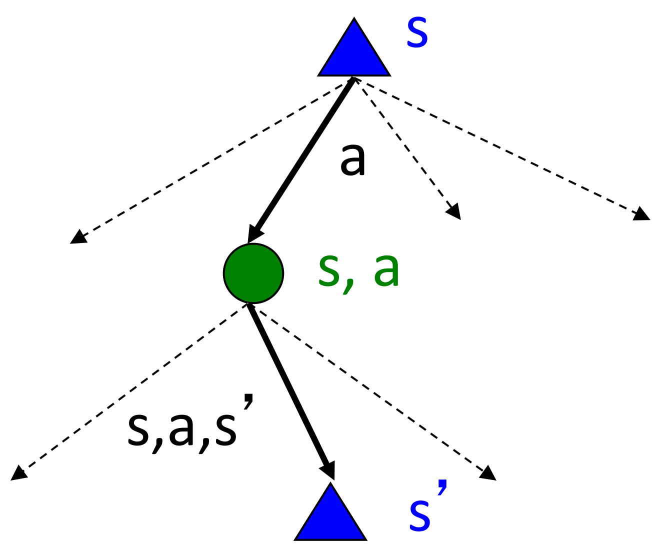
SOLVING MDPS



MDP QUANTITIES

Markov decision processes:

- States S
- Actions A
- Transitions P(s'|s,a) (or T(s,a,s'))
- Rewards R(s,a,s') (and discount γ)
- Start state s_0



MDP quantities:

- Policy = map of states to actions
- Utility = sum of (discounted) rewards
- (State) Value = expected utility starting from a state (max node)
- Q-Value = expected utility starting from a state-action pair, i.e., q-state (chance node)

MDP OPTIMAL QUANTITIES

The optimal policy:

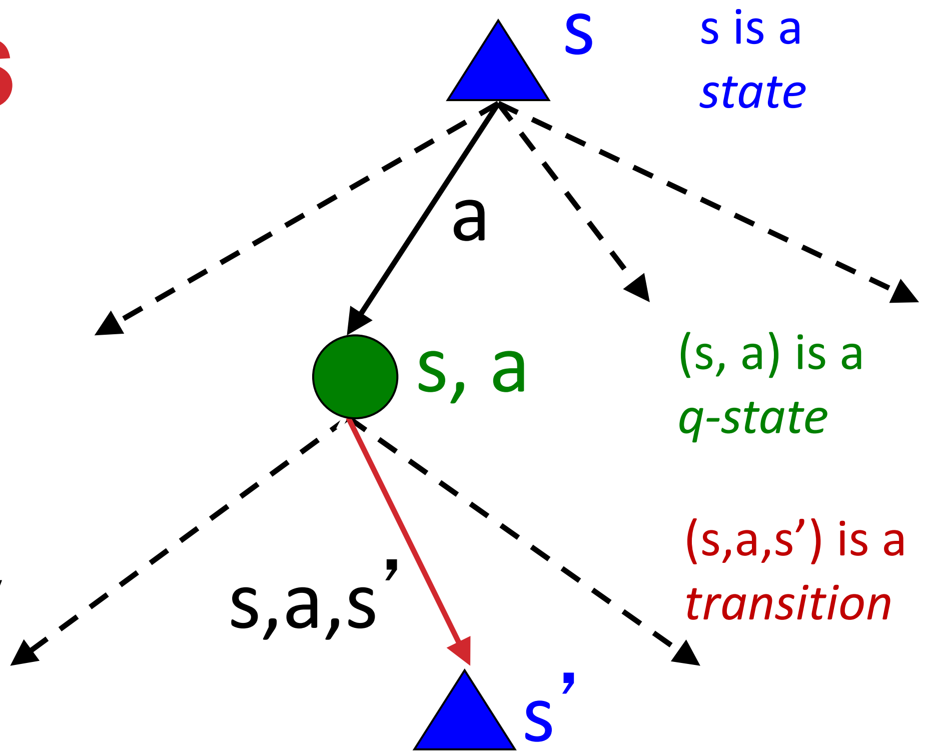
- $\pi^*(s)$ = optimal action from state s

The (true) value (or utility) of a state s:

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

The (true) value (or utility) of a q-state (s,a):

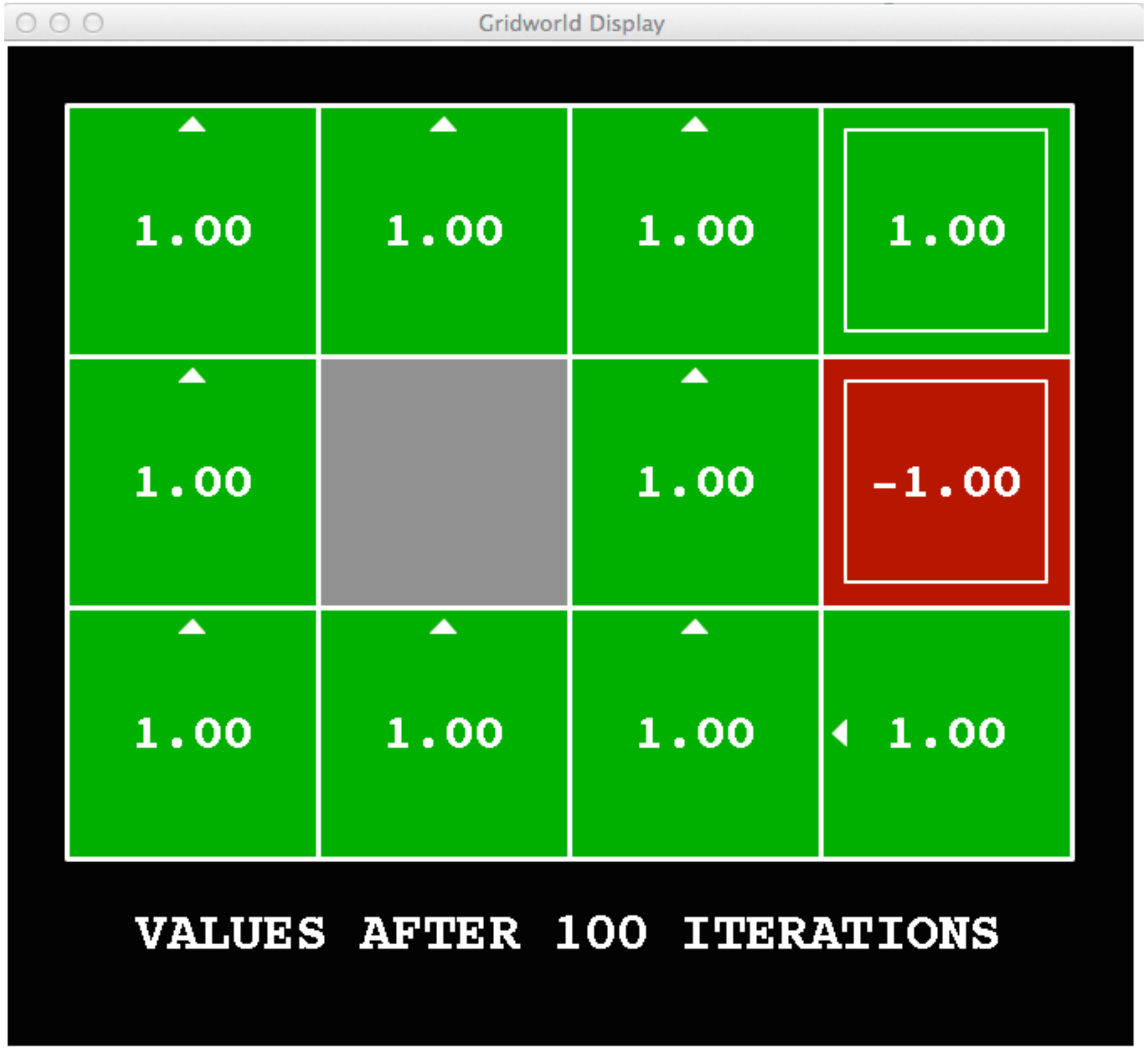
- $Q^*(s,a)$ = expected utility starting out having taken action a from state s and (thereafter) acting optimally



Solve MDP: Find π^*, V^* and/or Q^*

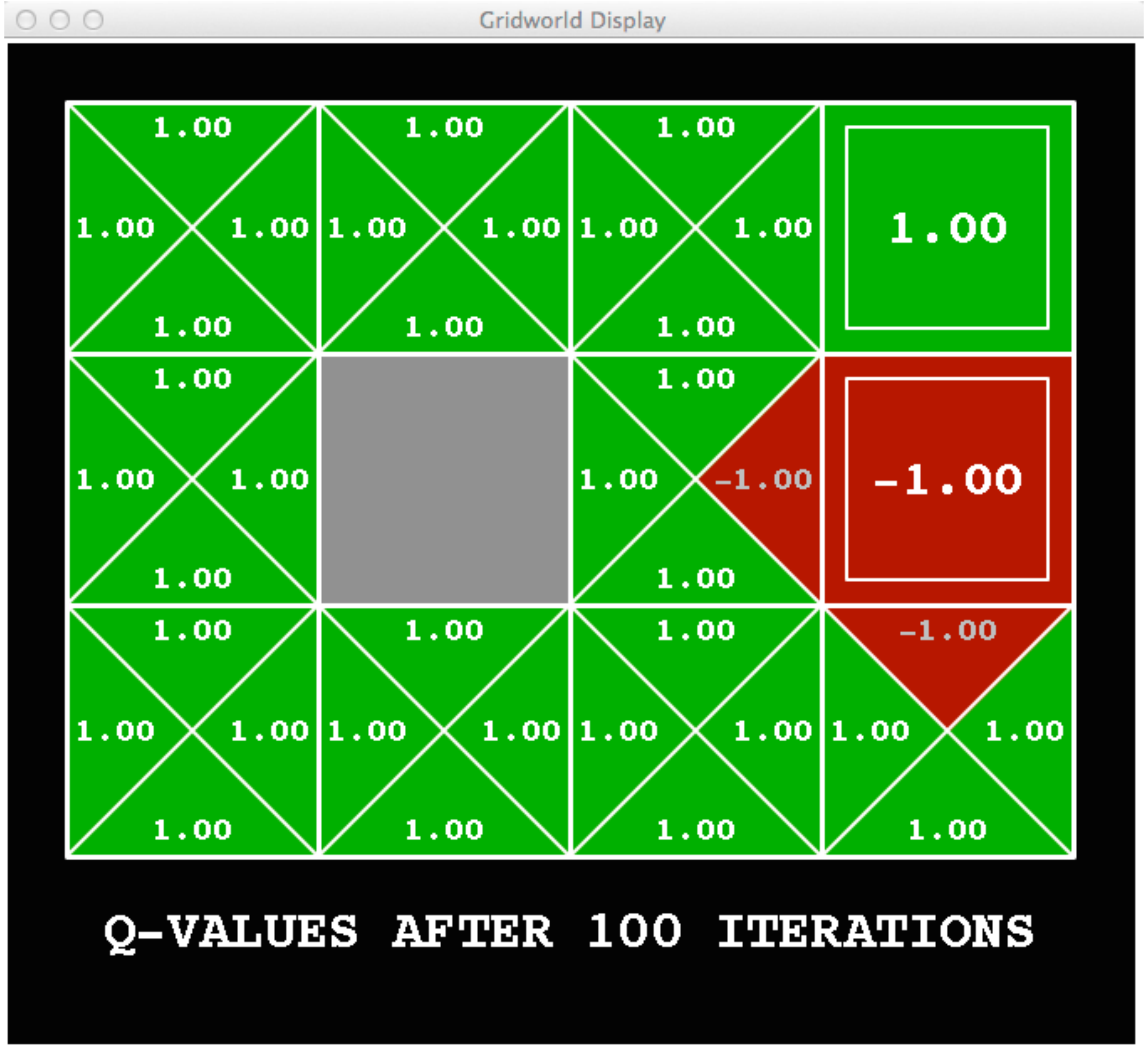
EXAMPLES WHERE WE ASSUME WE HAVE V AND Q VALUES …

GRIDWORLD V VALUES



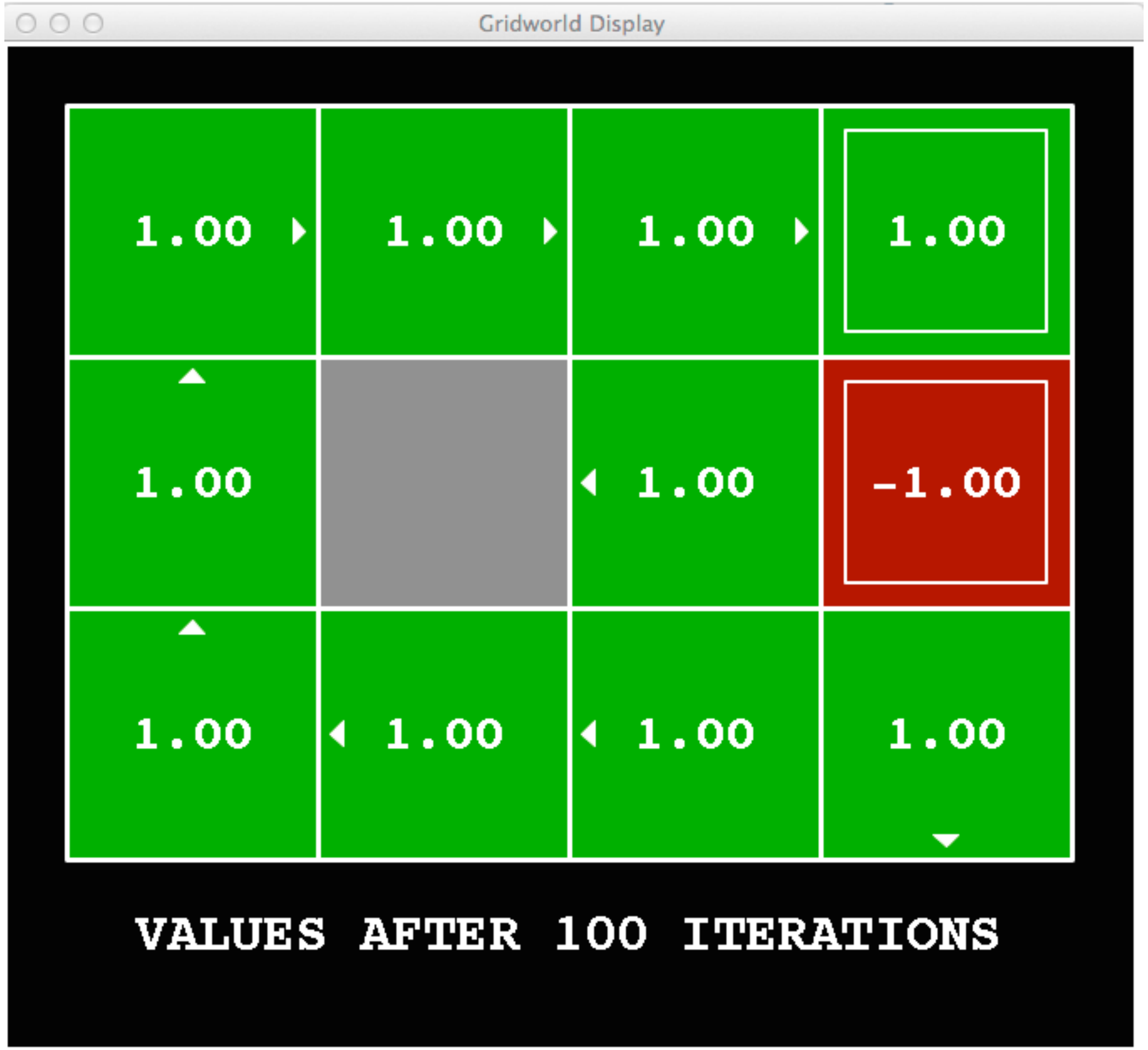
Noise = 0
Discount = 1
Living reward = 0

GRIDWORLD Q VALUES



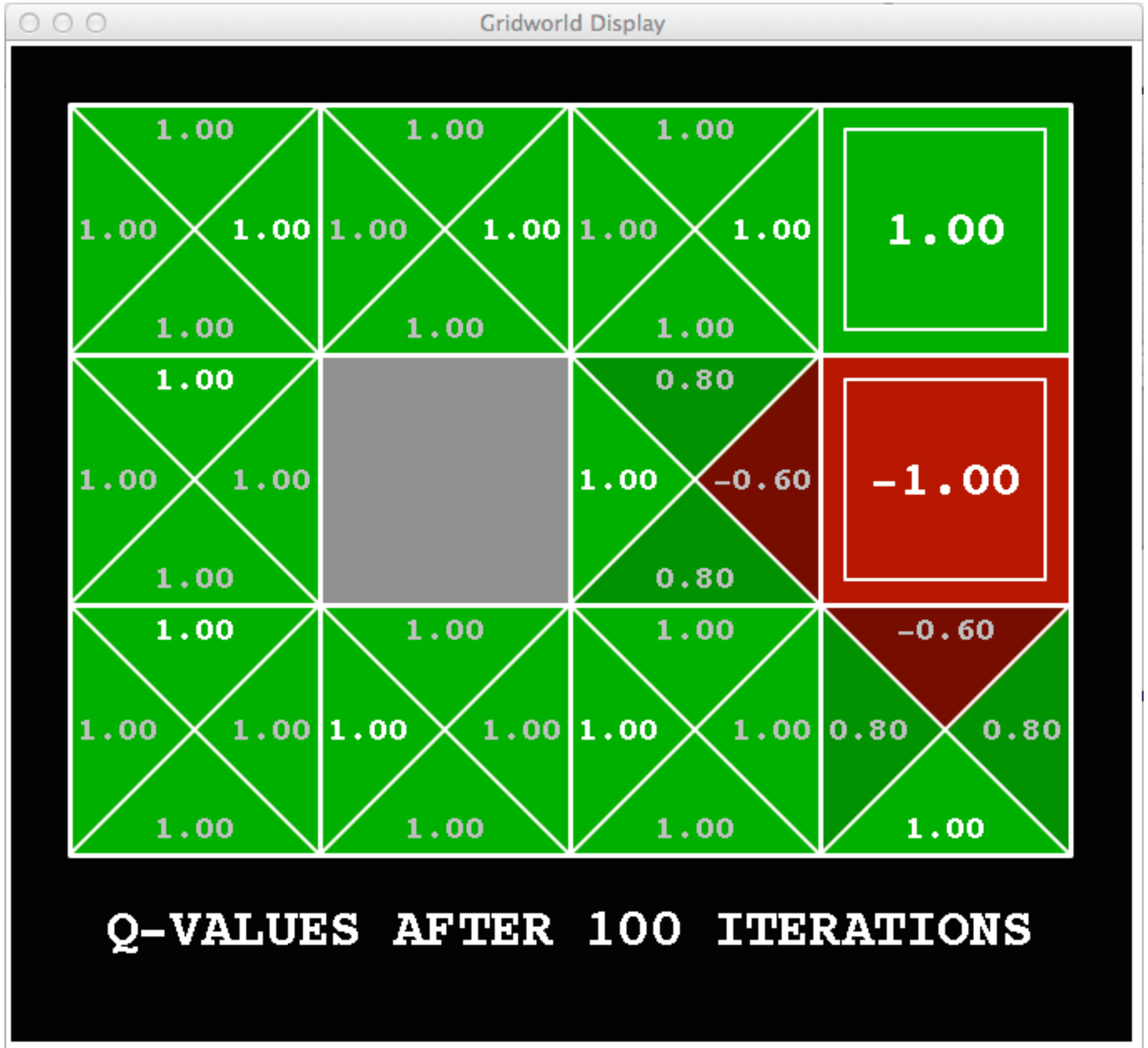
Noise = 0
Discount = 1
Living reward = 0

GRIDWORLD V VALUES



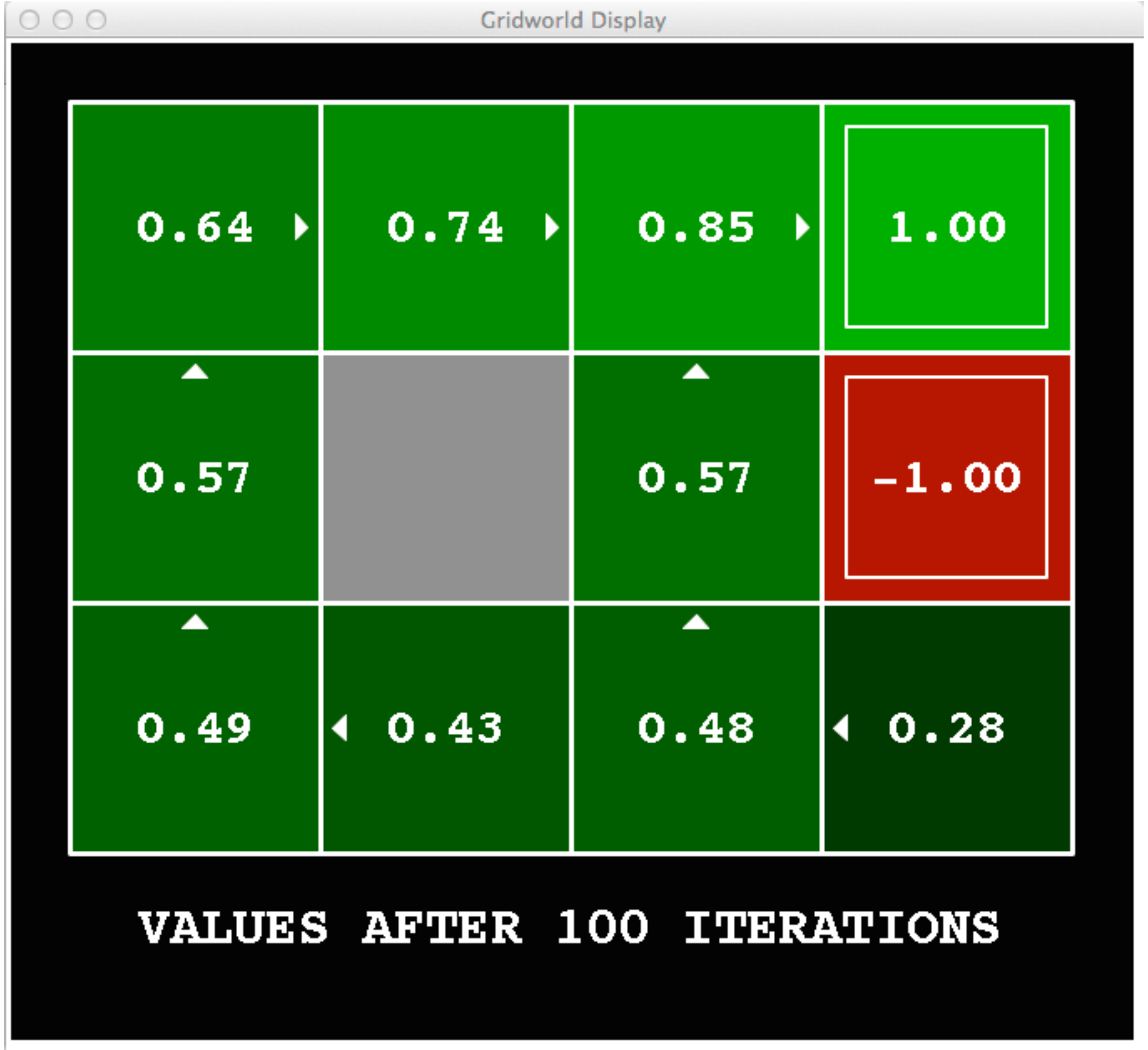
Noise = 0.2
Discount = 1
Living reward = 0

GRIDWORLD Q VALUES



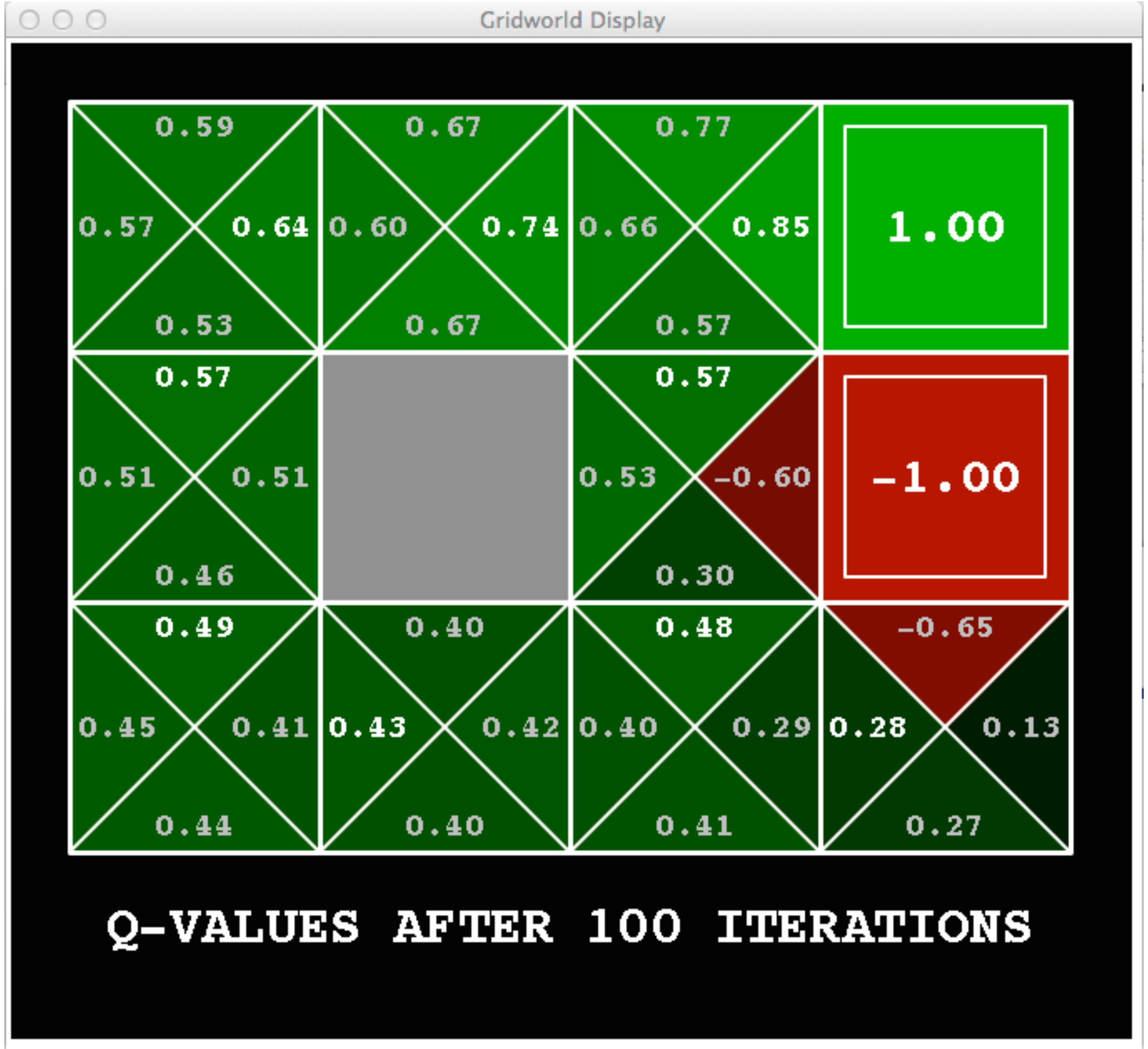
Noise = 0.2
Discount = 1
Living reward = 0

GRIDWORLD V VALUES



Noise = 0.2
Discount = 0.9
Living reward = 0

GRIDWORLD Q VALUES

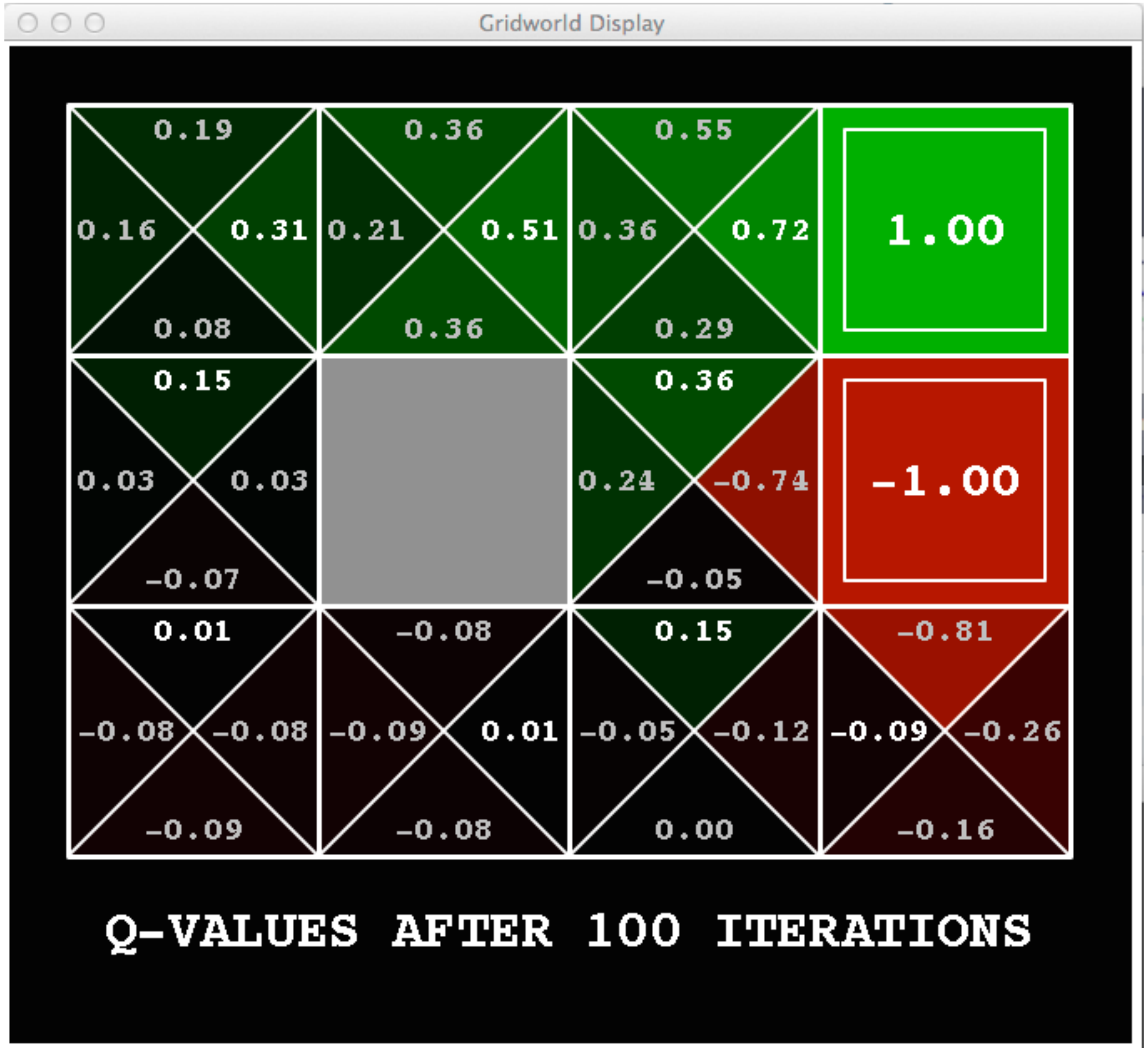


Noise = 0.2
Discount = 0.9
Living reward = 0

GRIDWORLD V VALUES

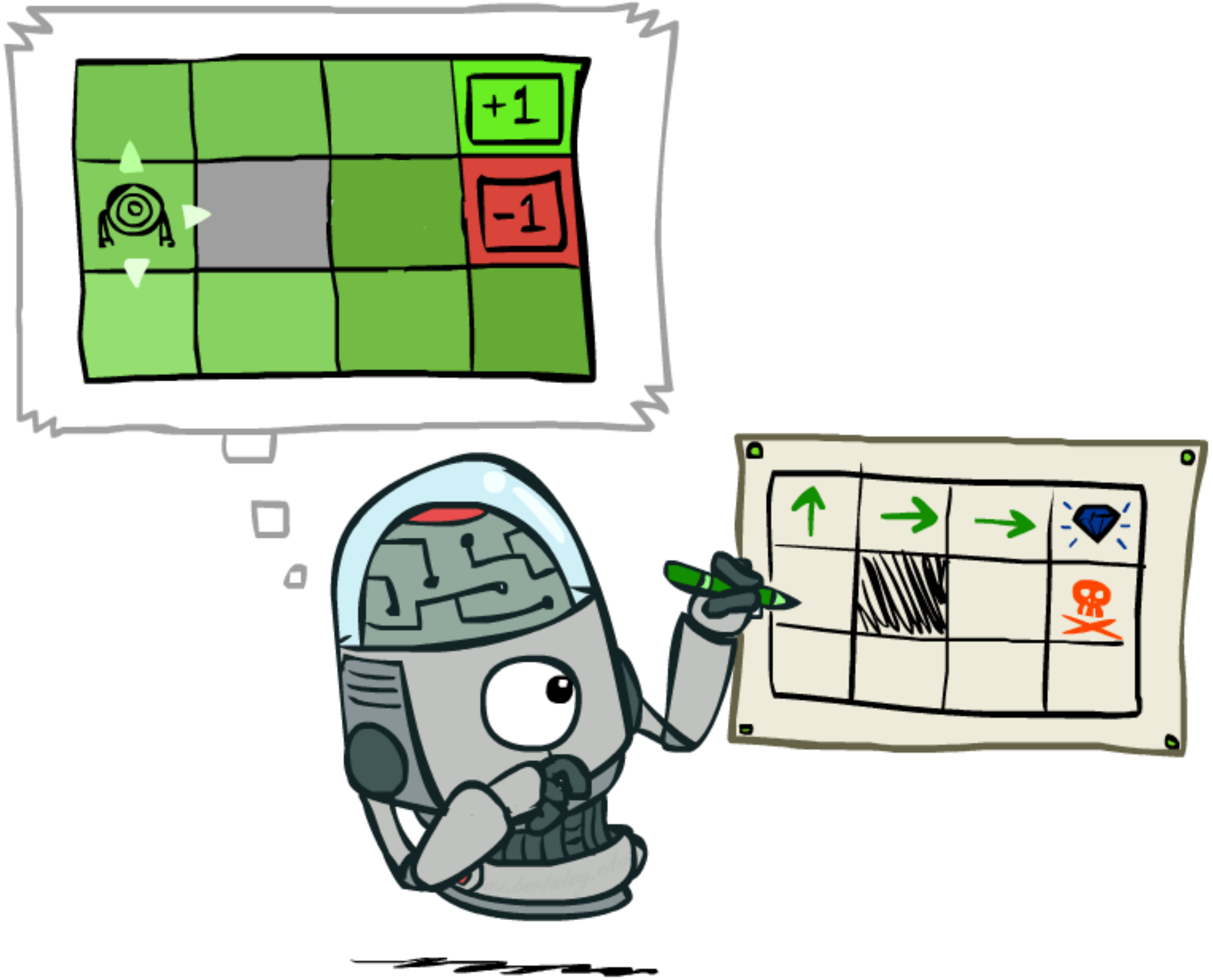
Noise = 0.2
Discount = 0.9
Living reward = -0.1

GRIDWORLD Q VALUES



Noise = 0.2
Discount = 0.9
Living reward = -0.1

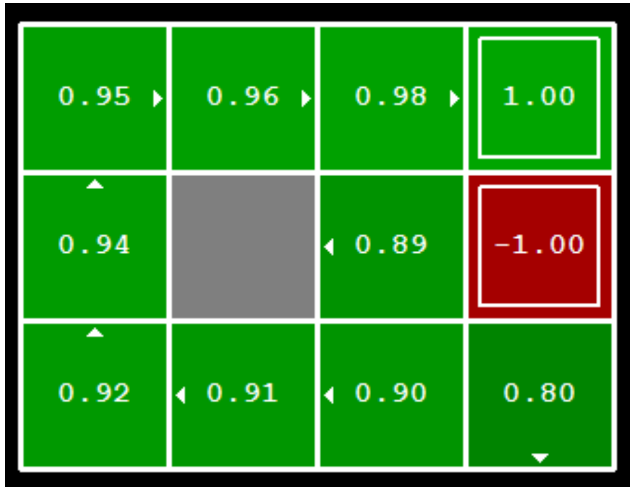
COMPUTING OPTIMAL POLICY FROM VALUES



COMPUTING OPTIMAL POLICY FROM VALUES

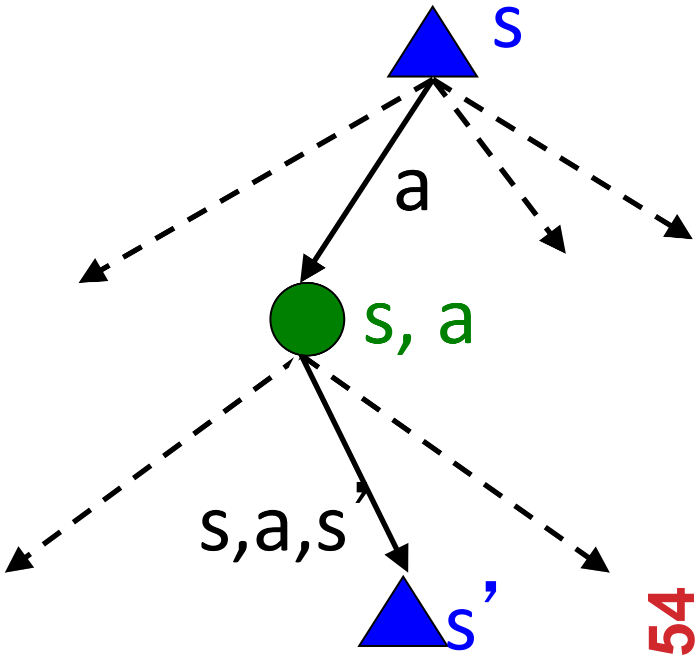
Let's imagine we have the optimal values V*(s)

How should we act?



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

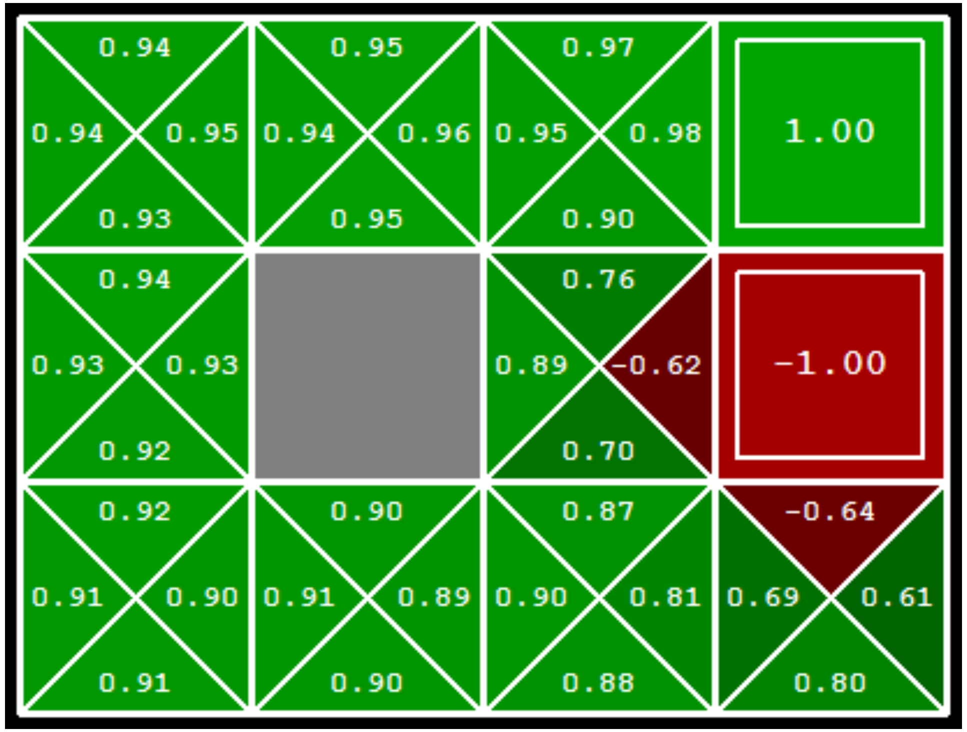
$$\pi^*(s) = \arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$



Sometimes this is called policy extraction, since it gets the policy implied by the values

COMPUTING OPTIMAL POLICY FROM Q-VALUES

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



How should we act?

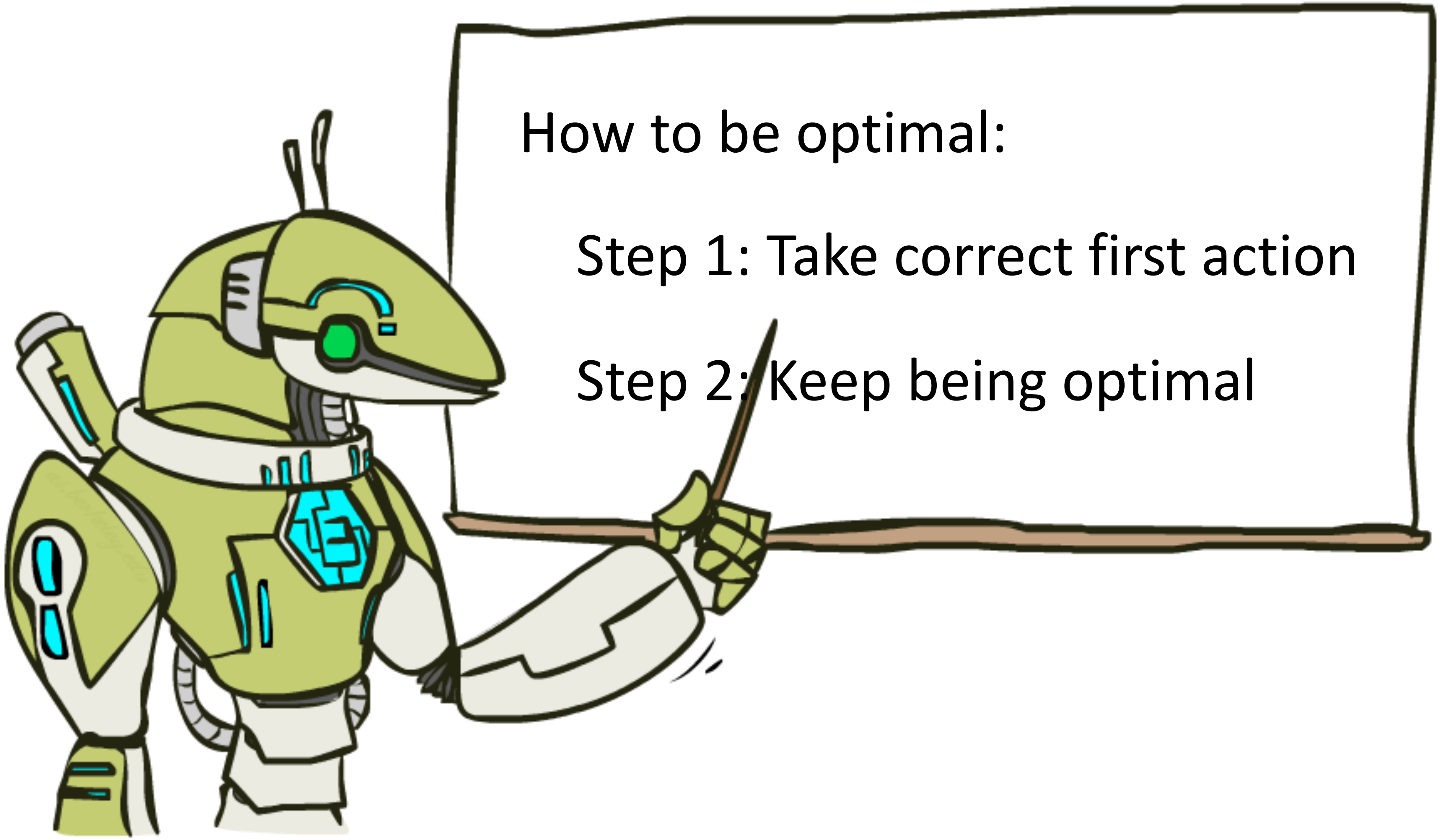
- Completely trivial to decide!

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

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

So, how do we compute these state values and q-values?

THE BELLMAN EQUATIONS



BELLMAN EQUATIONS: THE VALUE OF A STATE

Fundamental operation: compute the (expectimax) value of a state

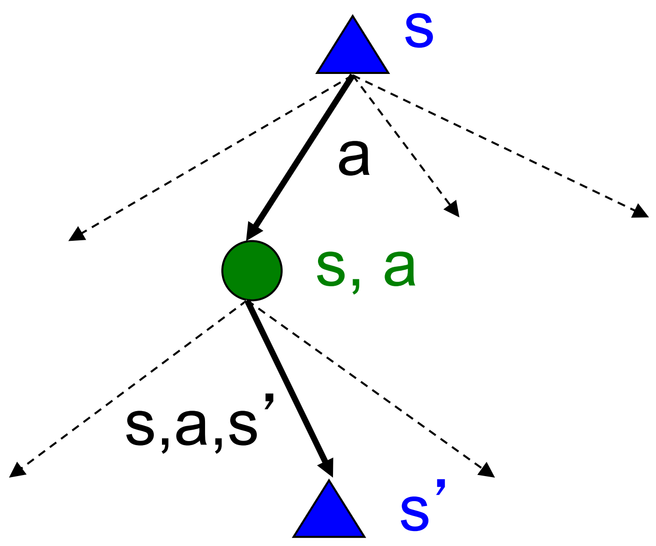
- Expected utility under optimal action
- Average sum of (discounted) rewards

Recursive definition of value:

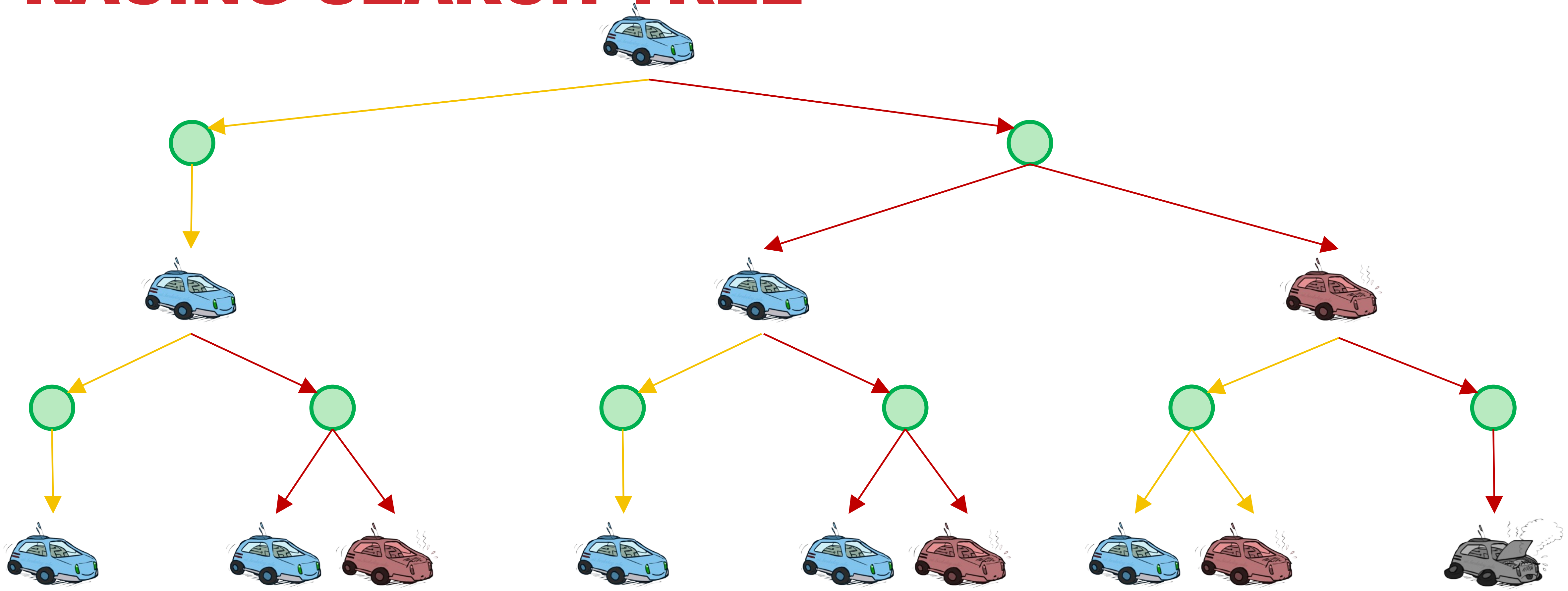
$$V^*(s) = \max_a Q^*(s,a)$$

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

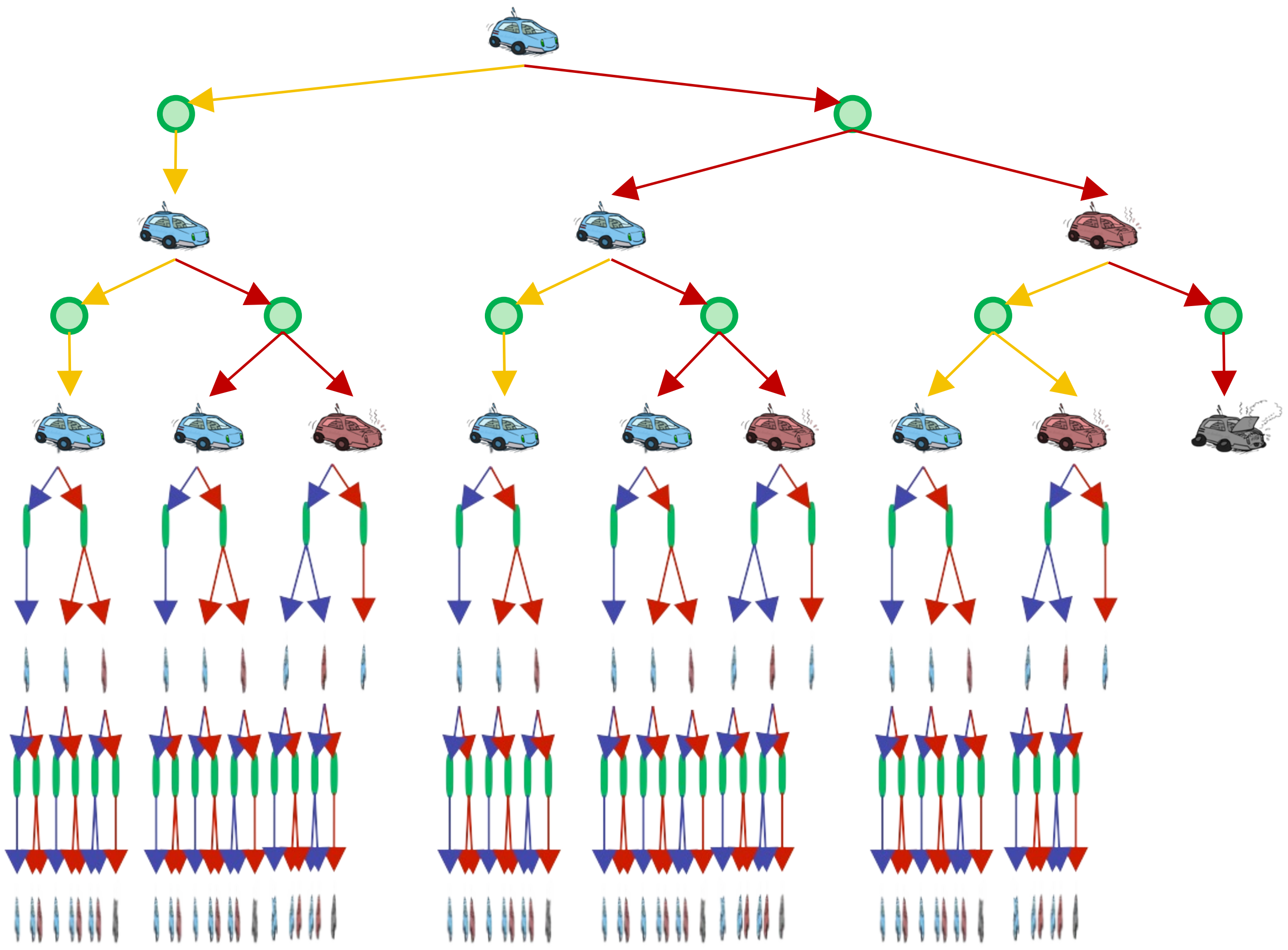
$$V^*(s) = \max_a \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V^*(s')\right]$$



RACING SEARCH TREE



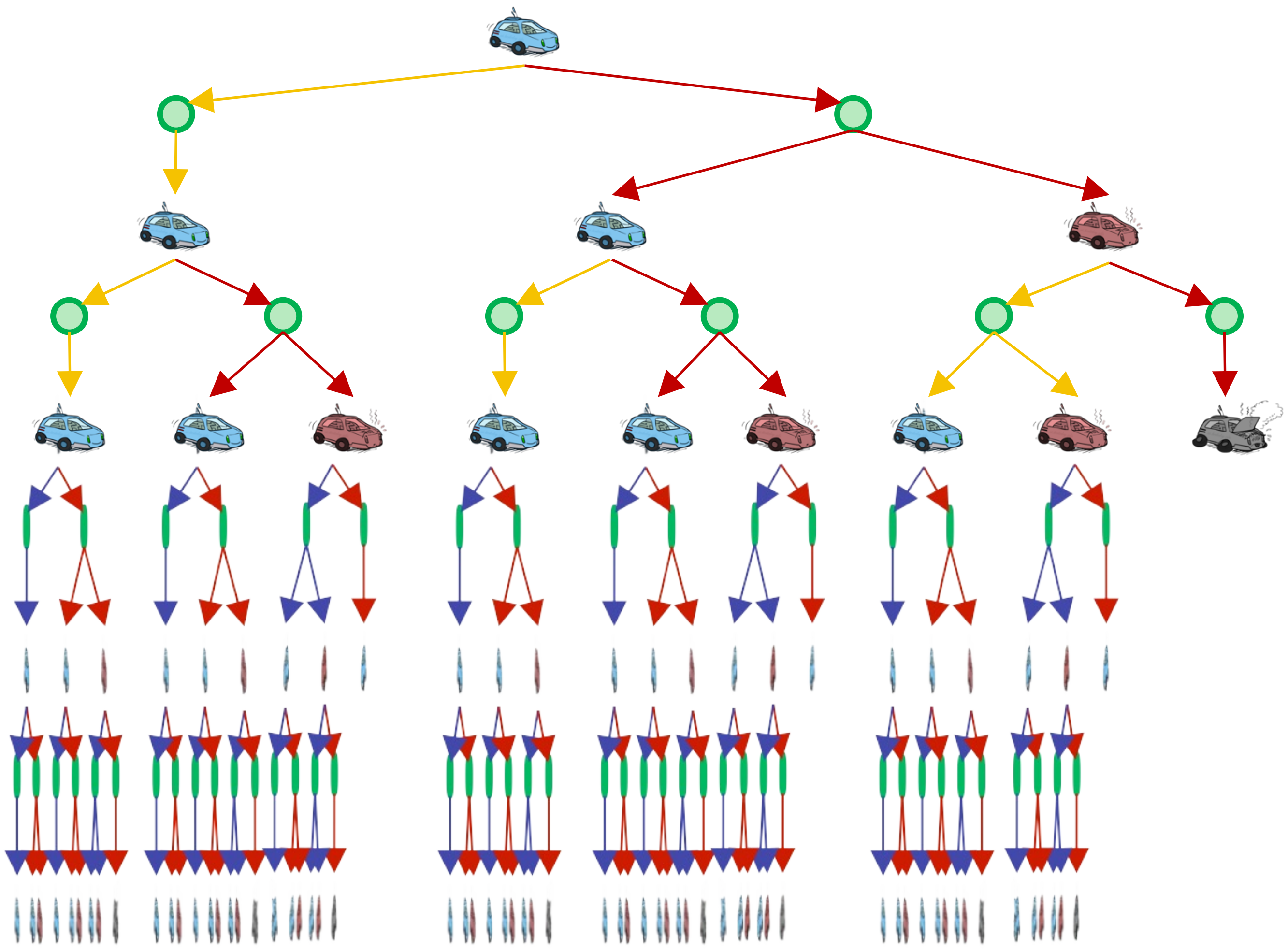
RACING SEARCH TREE



Enumerate all paths, determine value of each path, choose path with highest value (aka expectimax)…?

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

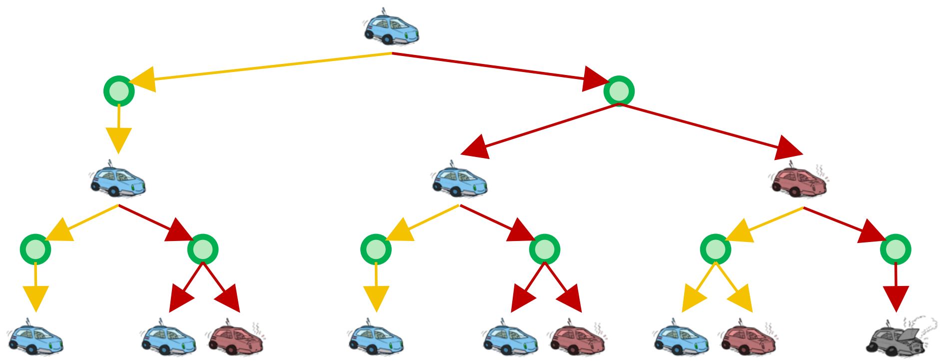
- Idea: Do a depth-limited computation, but with increasing depths until change is small
- Note: deep parts of the tree eventually don't matter if $\gamma < 1$

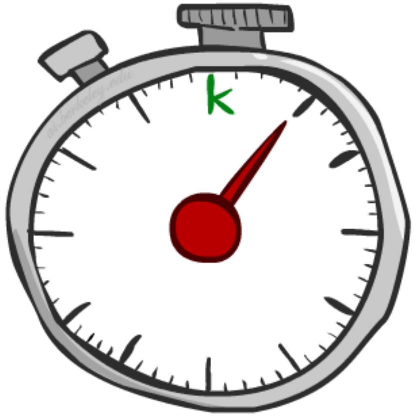
TIME-LIMITED VALUES

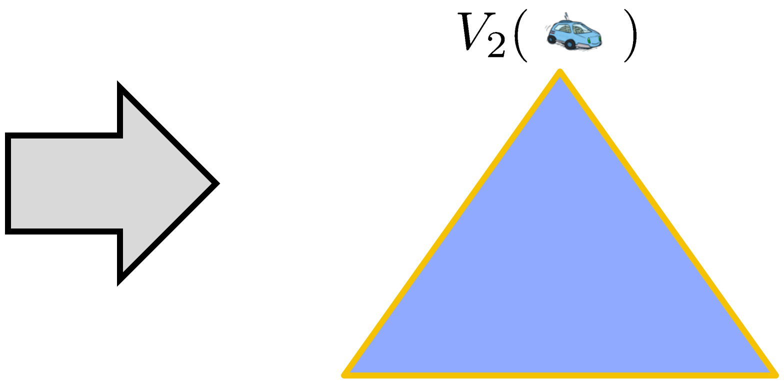
Key idea: time-limited values

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

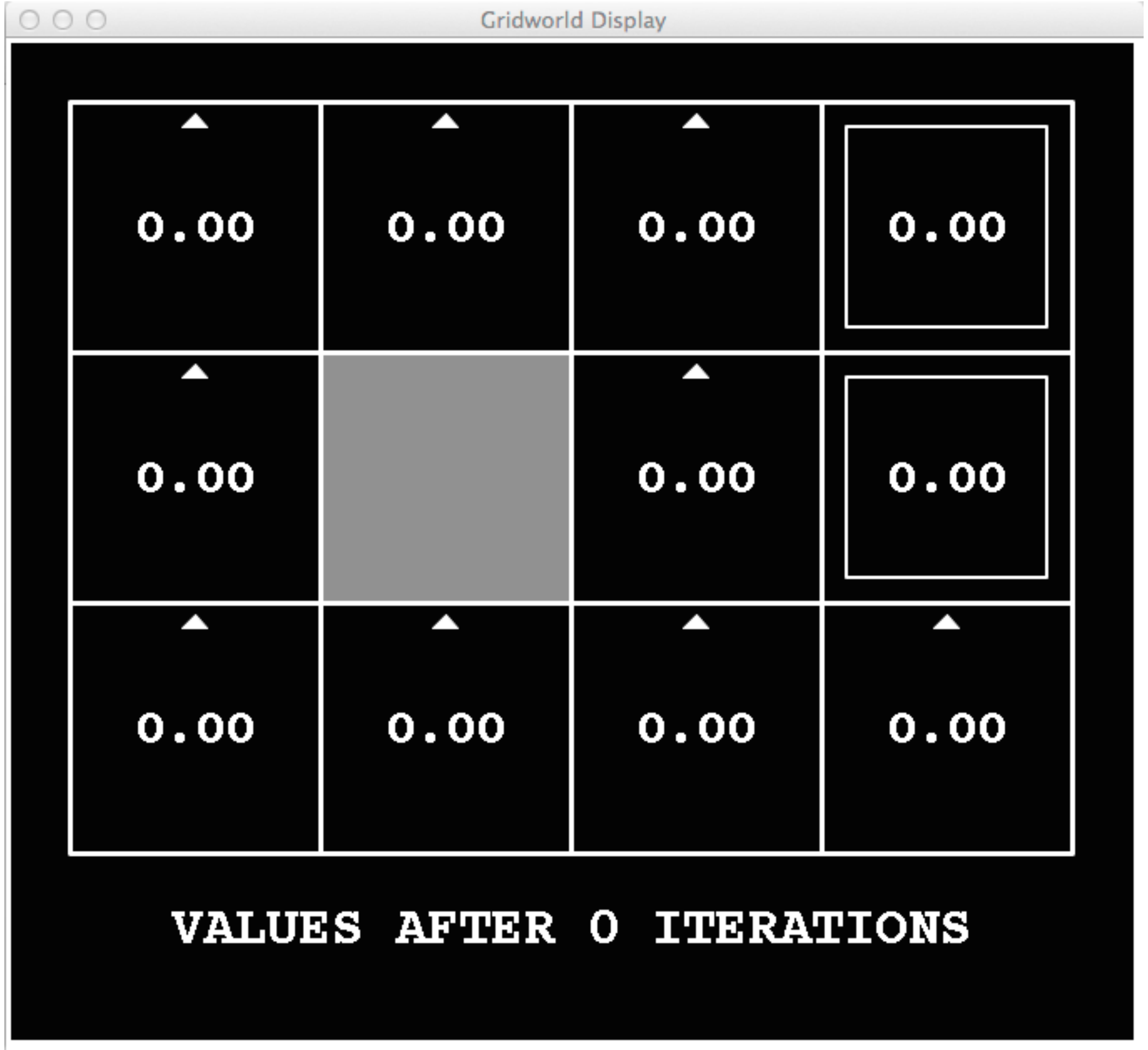






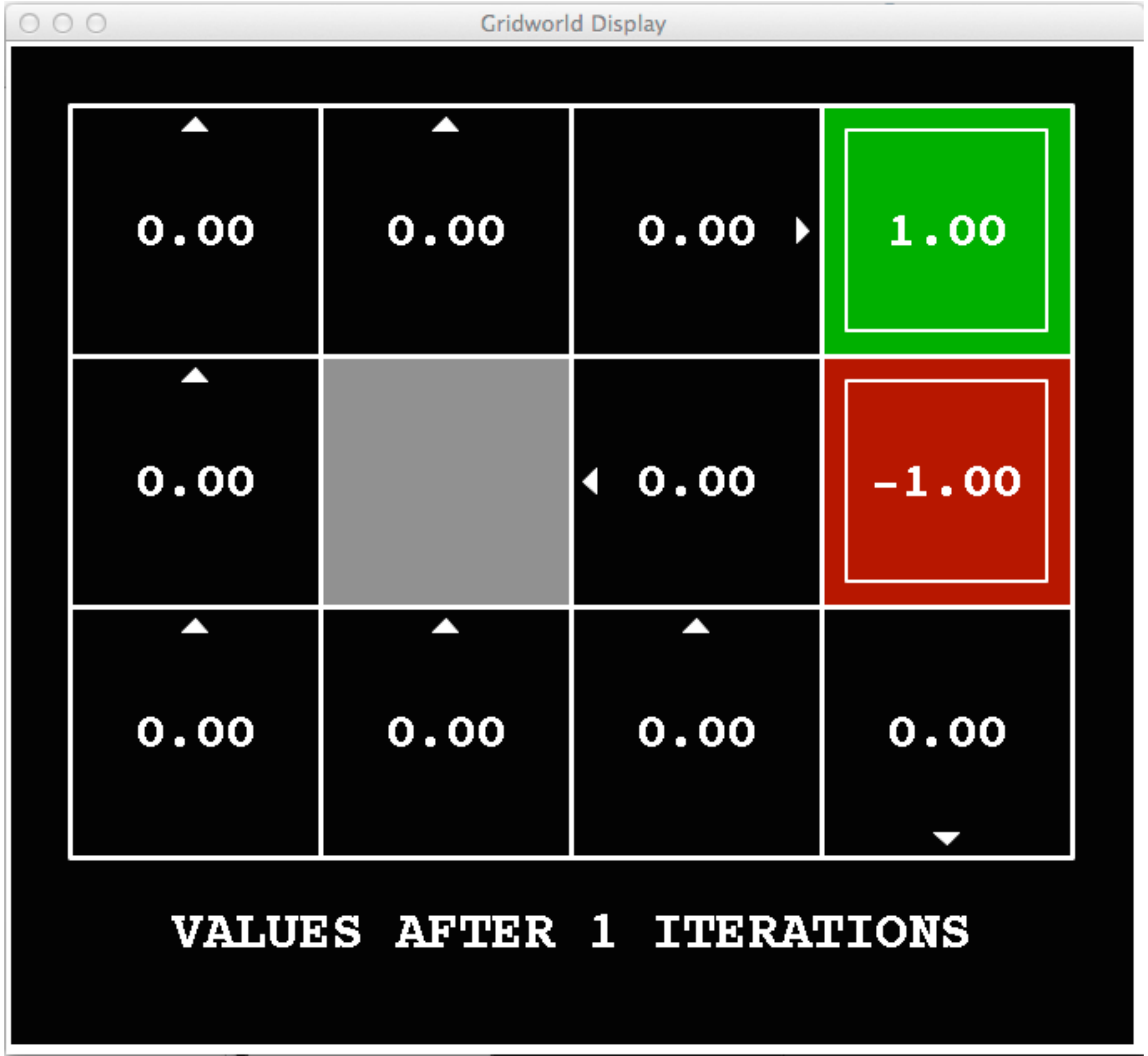
(Also watch the actions change as we go along.)

K=0



Noise = 0.2
Discount = 0.9
Living reward = 0

K=1



Noise = 0.2
Discount = 0.9
Living reward = 0

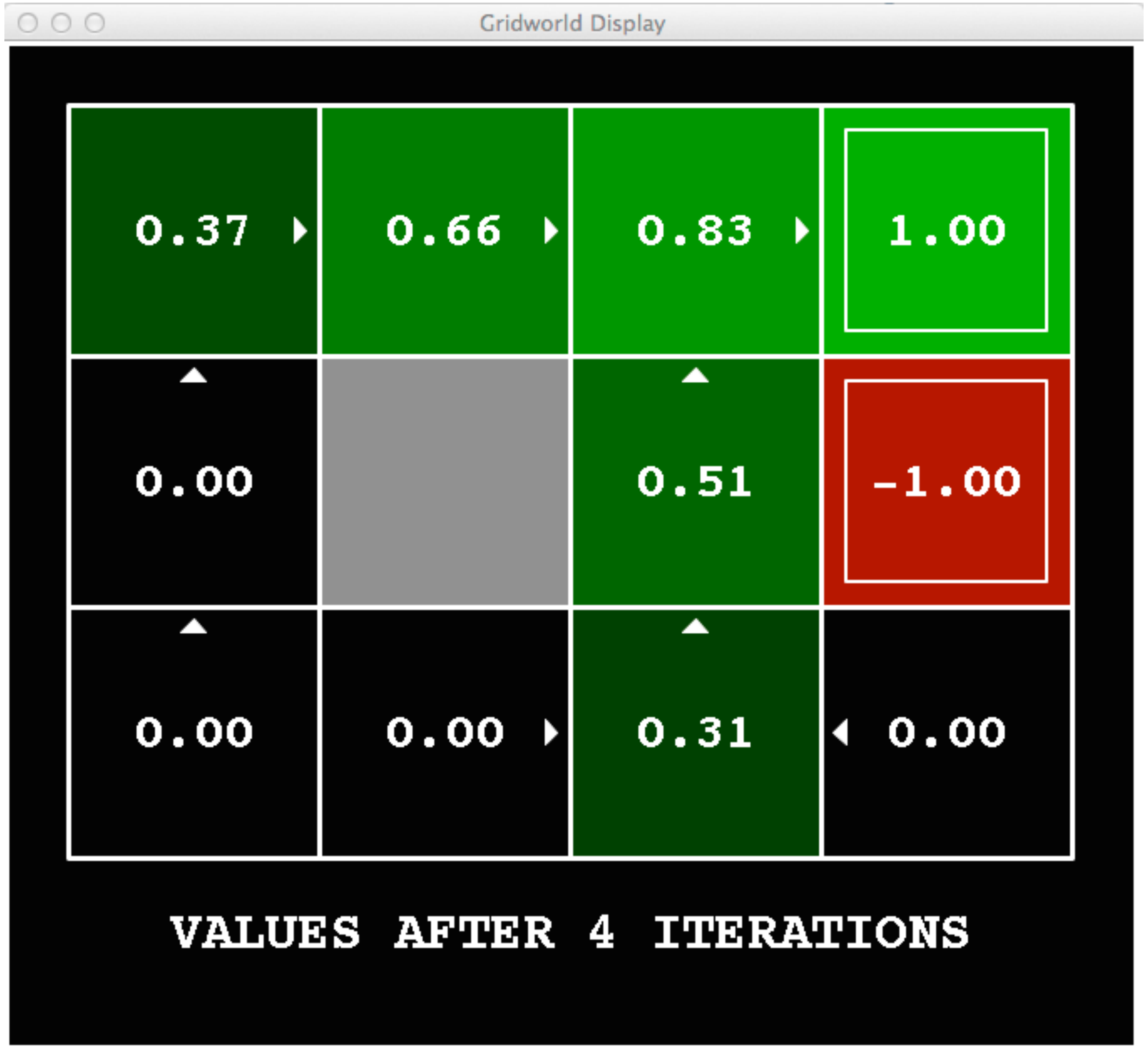
K=2

Noise = 0.2
Discount = 0.9
Living reward = 0

K=3

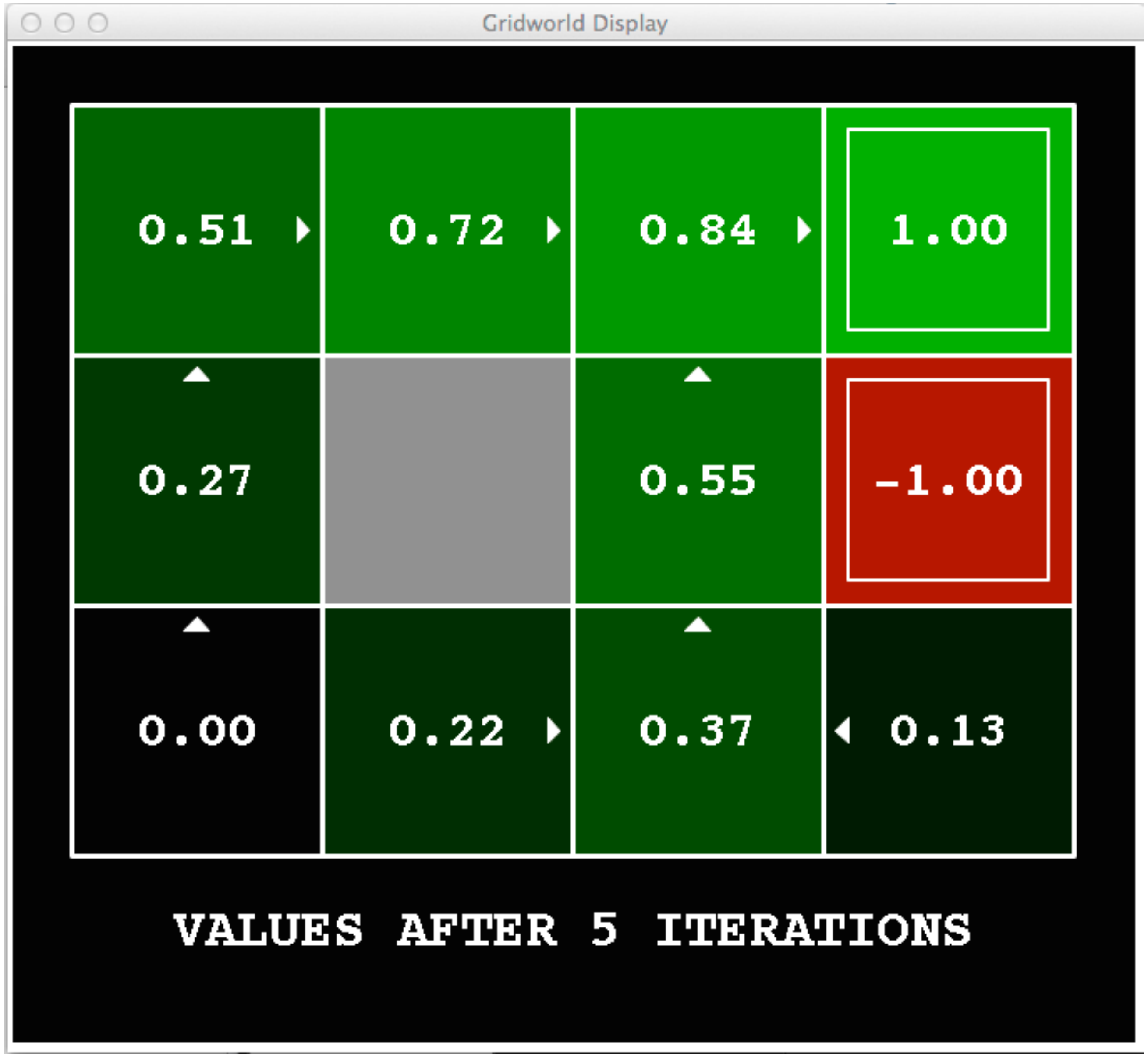
Noise = 0.2
Discount = 0.9
Living reward = 0

K=4



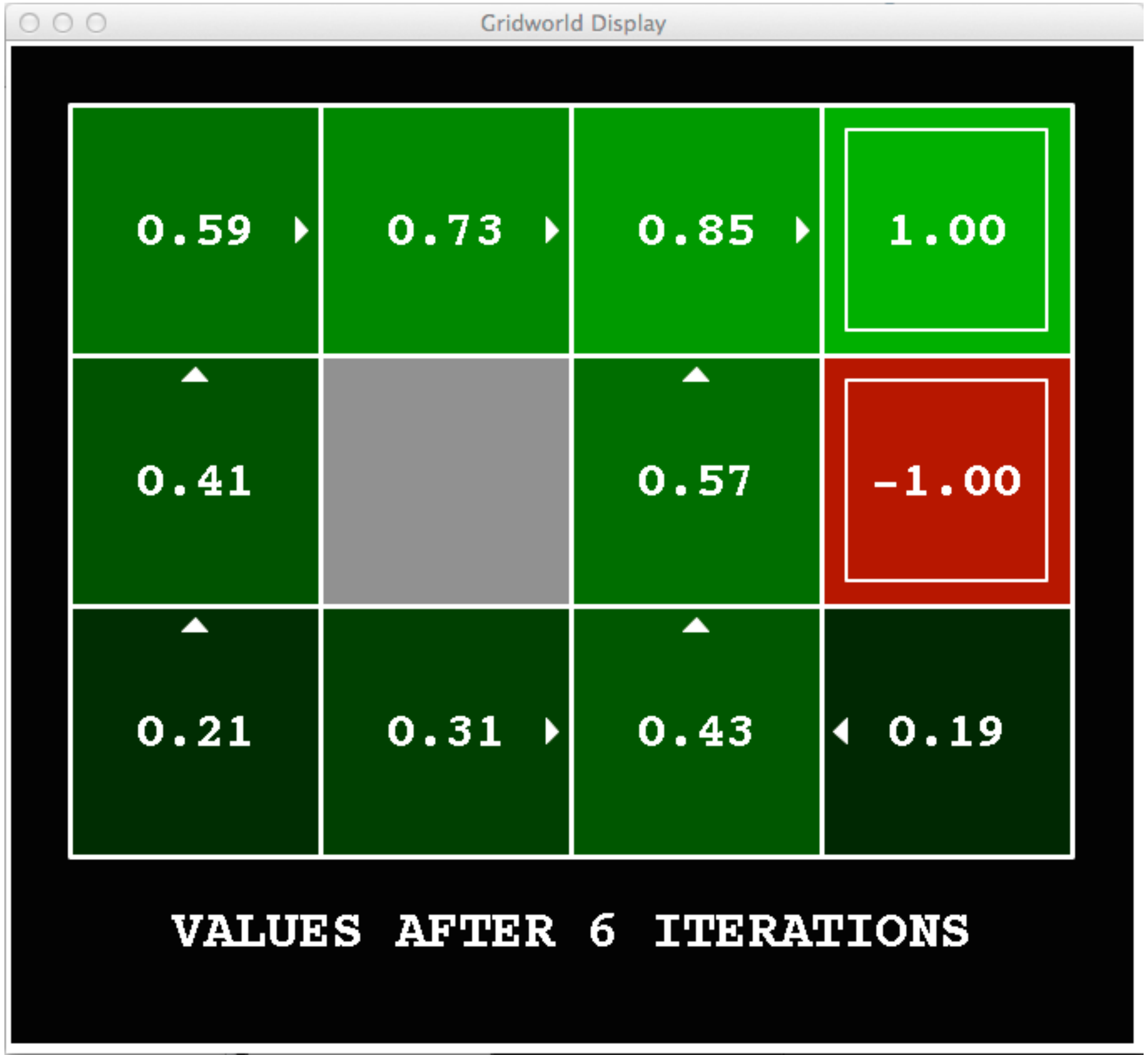
Noise = 0.2
Discount = 0.9
Living reward = 0

K=5



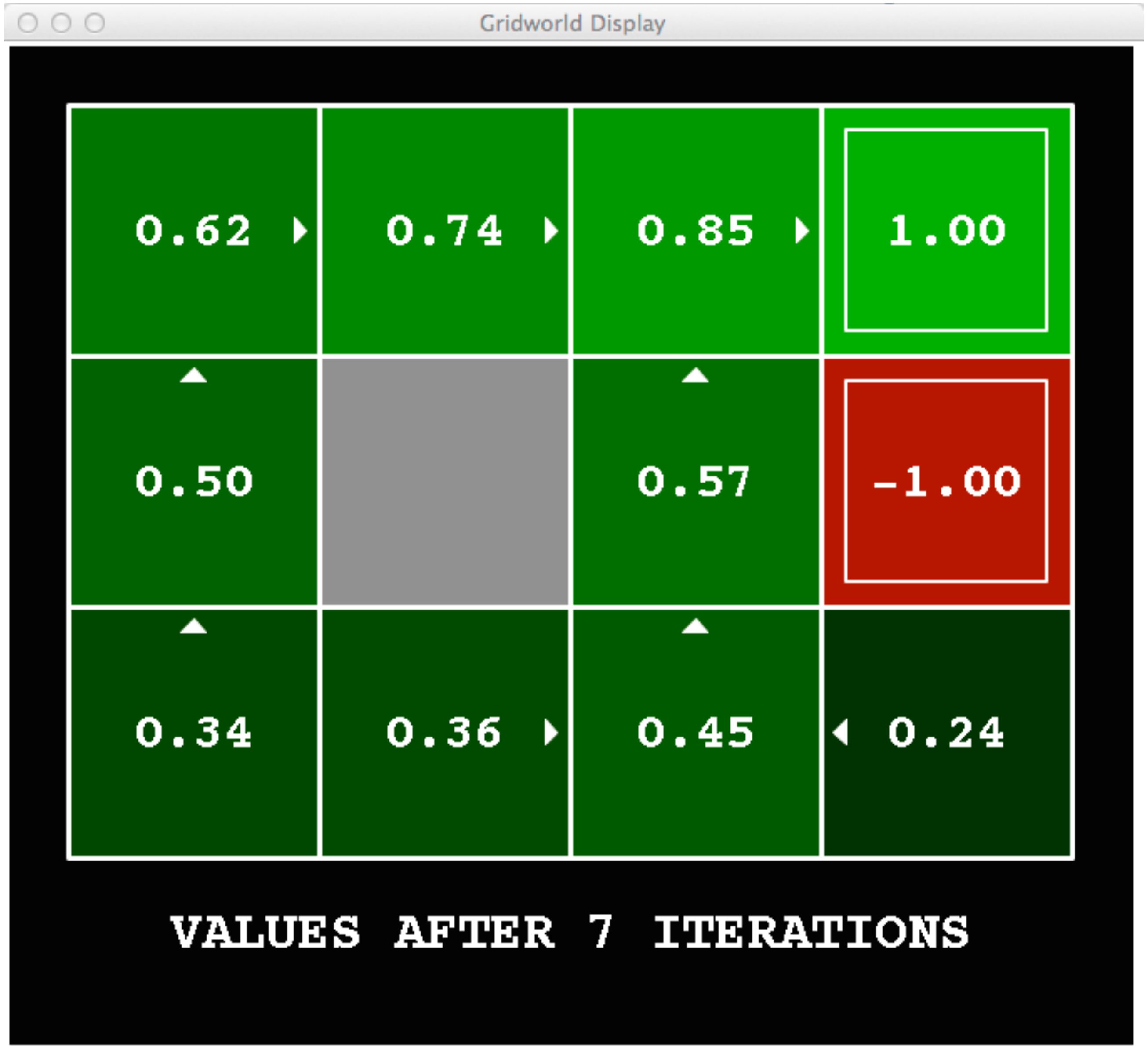
Noise = 0.2
Discount = 0.9
Living reward = 0

K=6



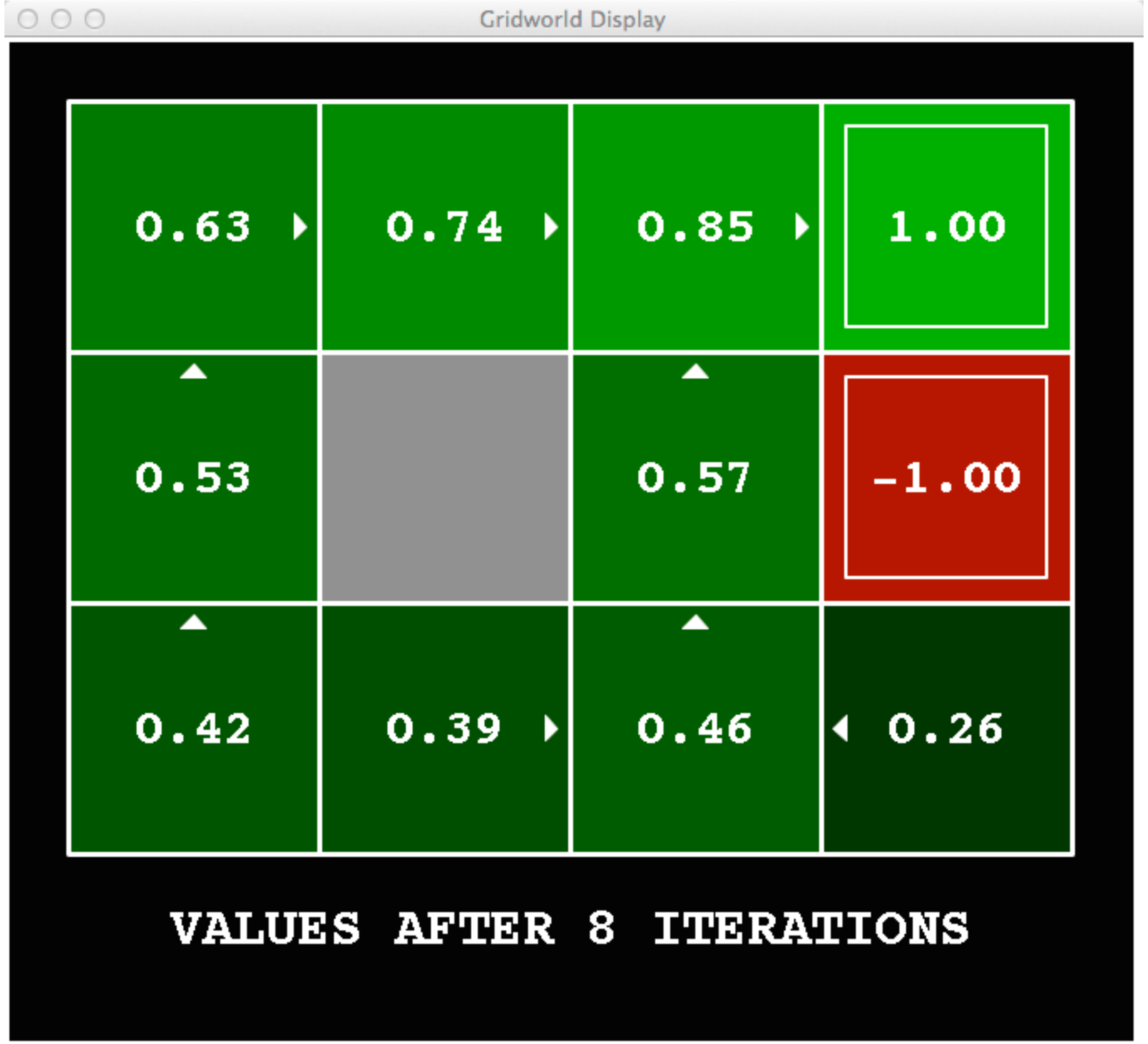
Noise = 0.2
Discount = 0.9
Living reward = 0

K=7



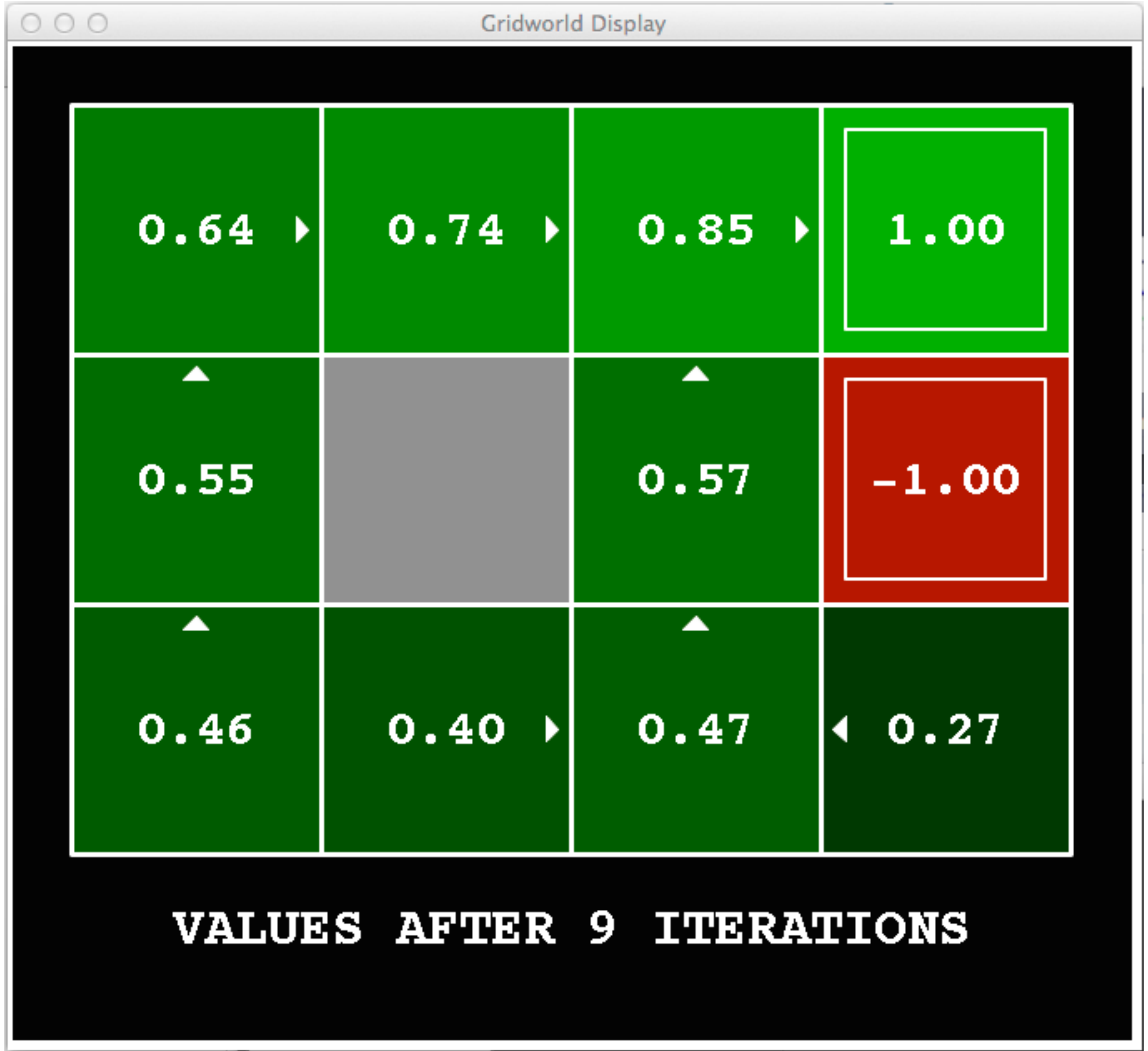
Noise = 0.2
Discount = 0.9
Living reward = 0

K=8



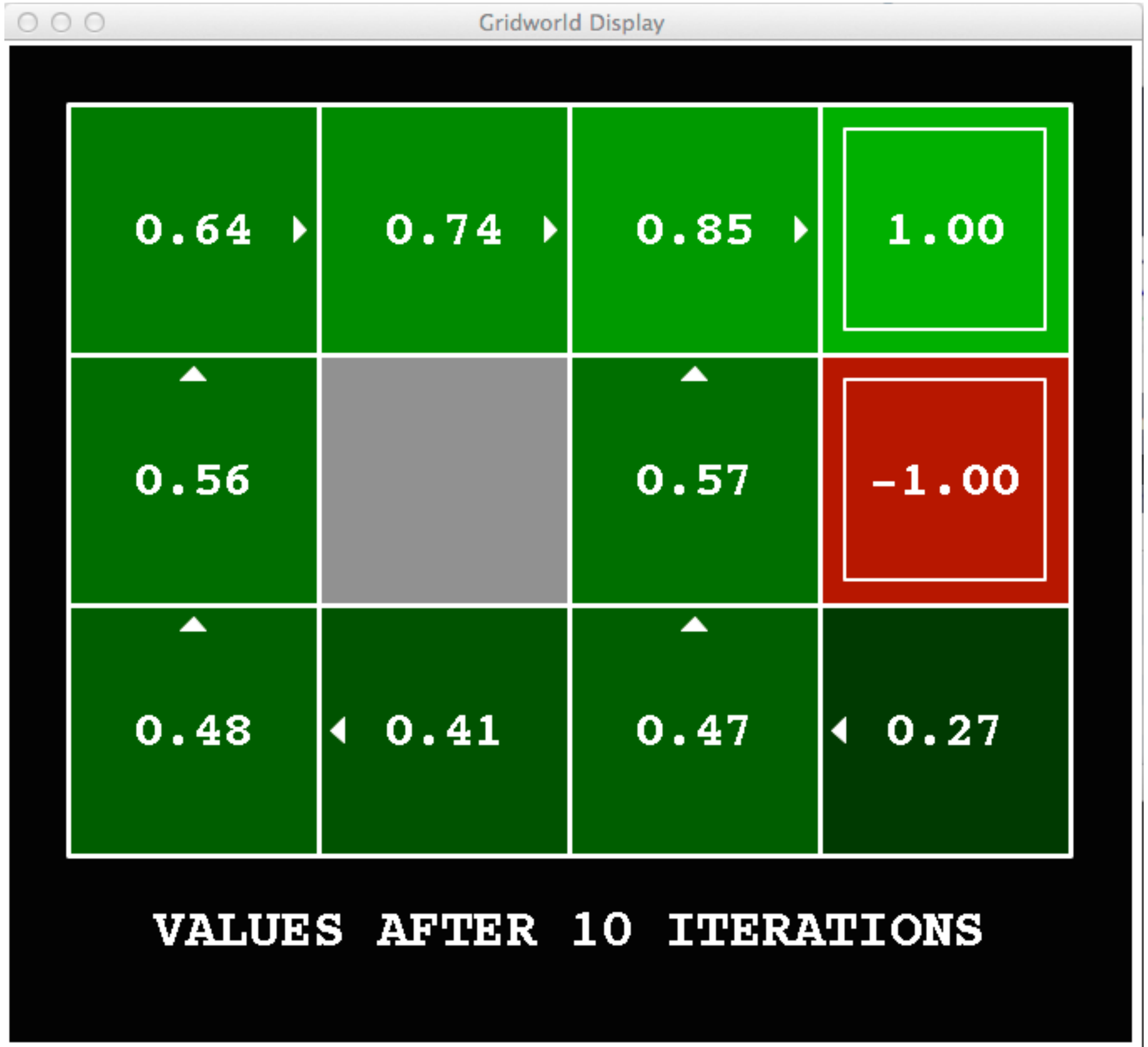
Noise = 0.2
Discount = 0.9
Living reward = 0

K=9



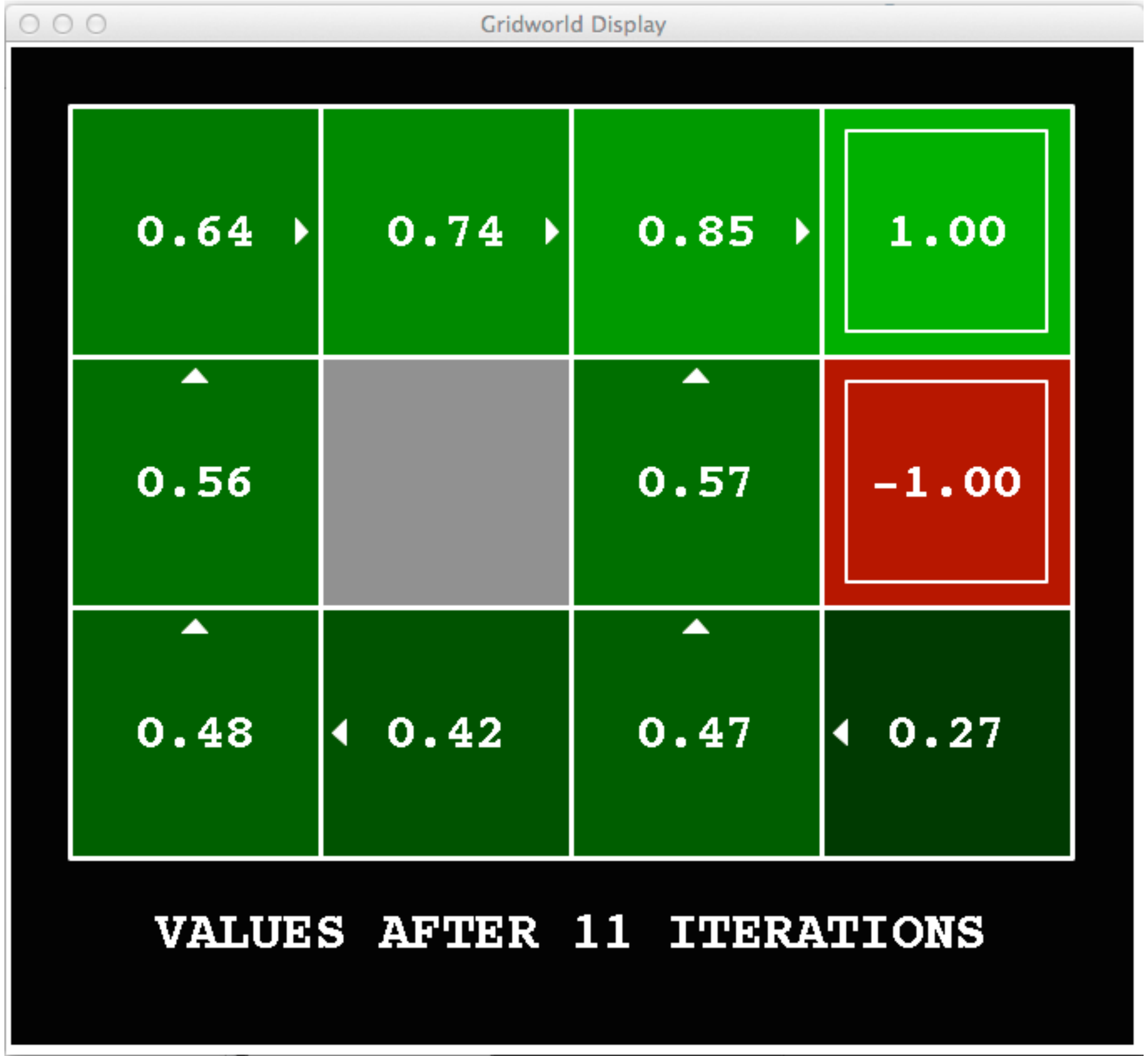
Noise = 0.2
Discount = 0.9
Living reward = 0

K=10



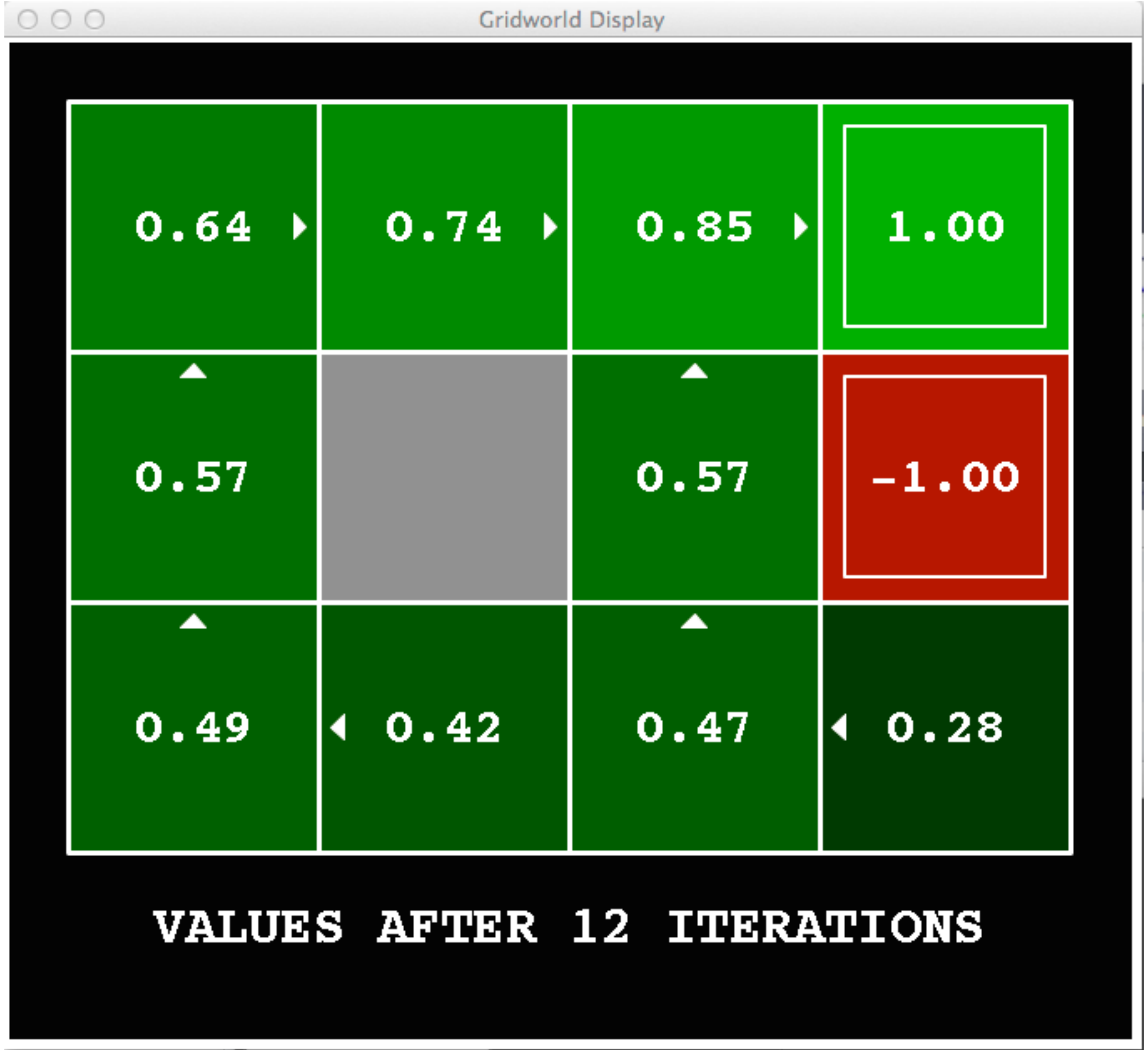
Noise = 0.2
Discount = 0.9
Living reward = 0

K=11



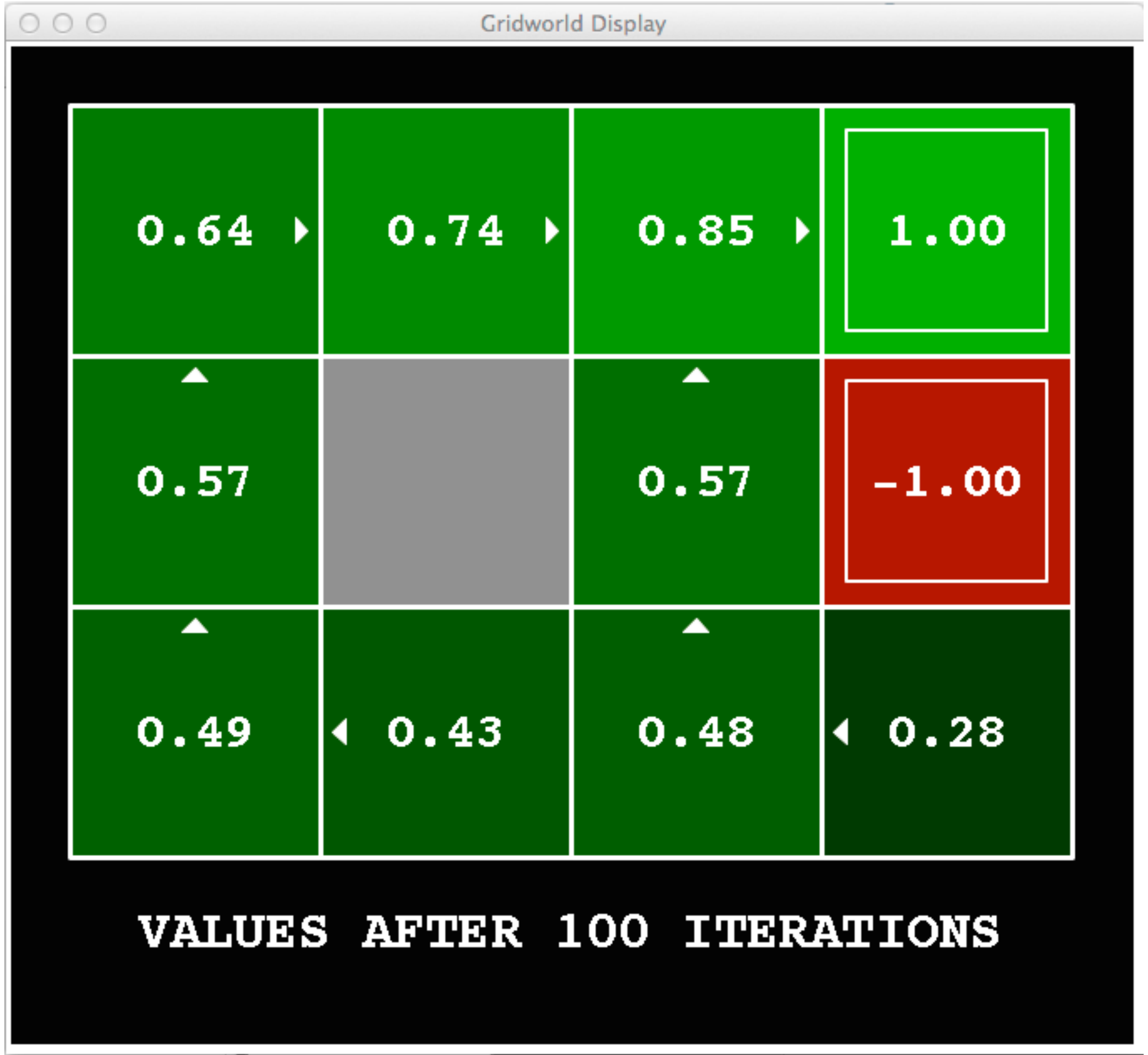
Noise = 0.2
Discount = 0.9
Living reward = 0

K=12



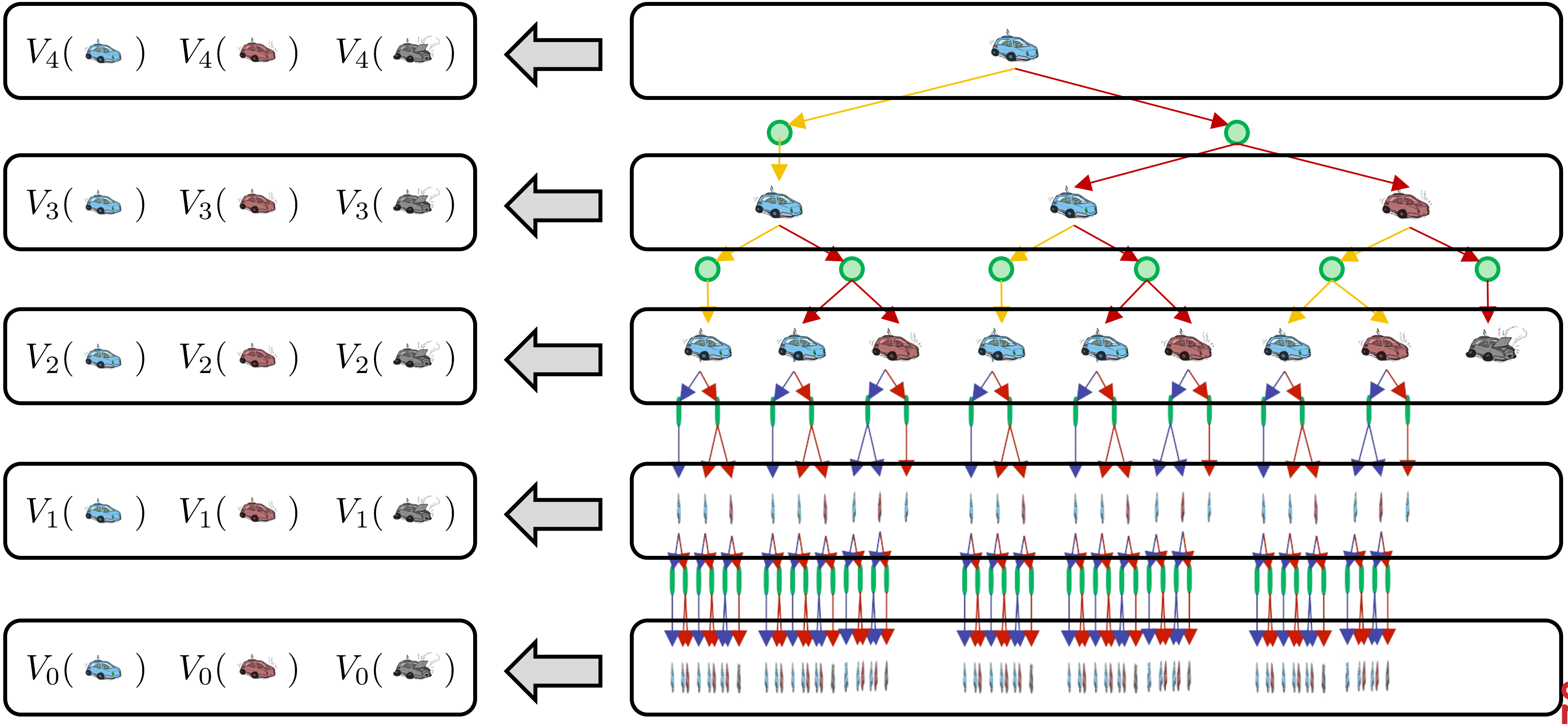
Noise = 0.2
Discount = 0.9
Living reward = 0

K=100

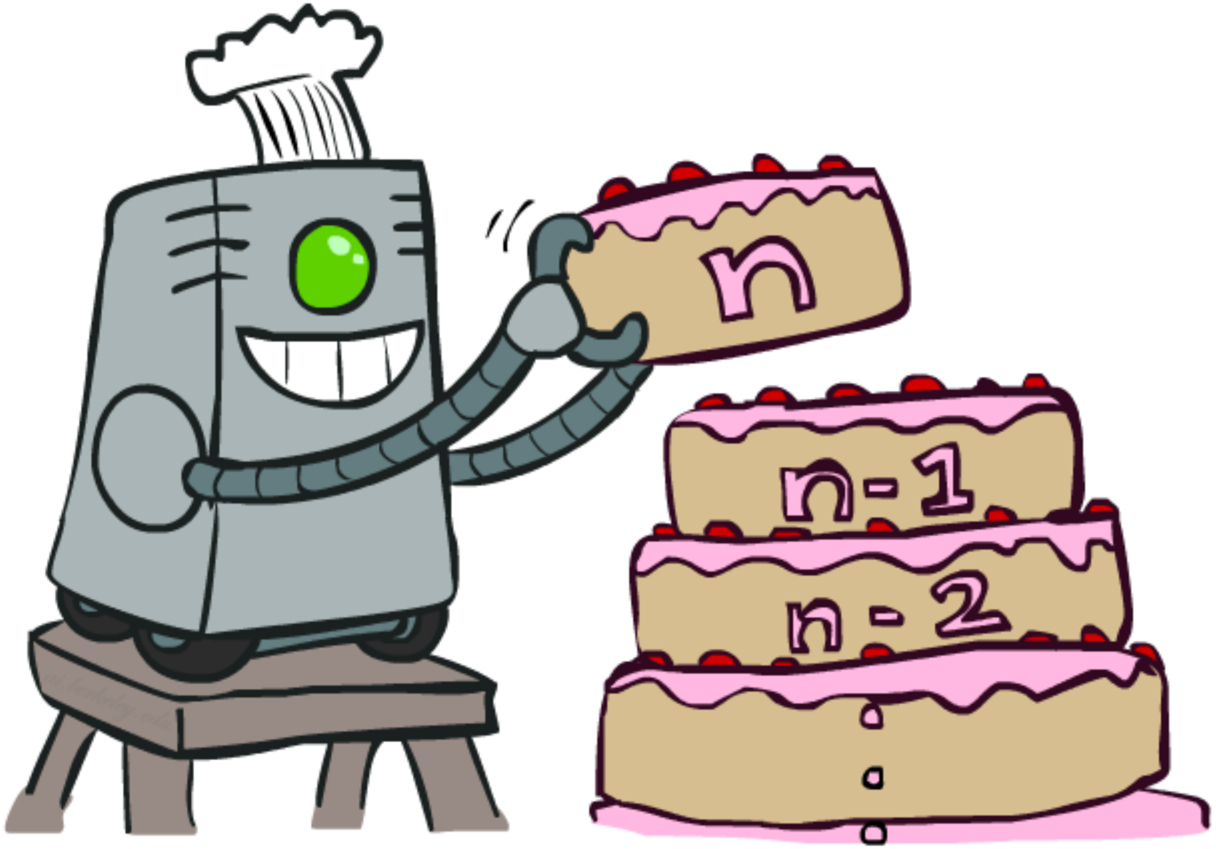


Noise = 0.2
Discount = 0.9
Living reward = 0

COMPUTING TIME-LIMITED VALUES



VALUE ITERATION

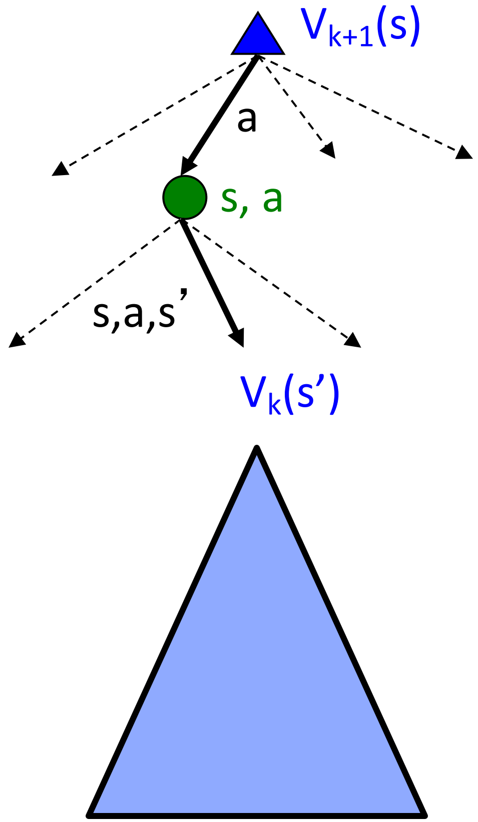


VALUE ITERATION

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: ????????????????

- $O(S^2A)$

Theorem: will converge to unique optimal values

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

VALUE ITERATION

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

function VALUE-ITERATION(MDP=(S,A,T,R,γ), *threshold*) returns a state value function

```
for s in S
    V_0(s) ← 0
k ← 0
repeat
    δ ← 0
    for s in S
        V_{k+1}(s) ← −∞
        for a in A
            v ← 0
            for s' in S
                v ← v + T(s,a,s')(R(s,a,s') + γV_k(s'))
            V_{k+1}(s) ← max{V_{k+1}(s), v}
        δ ← max{δ, |V_{k+1}(s) − V_k(s)|}
    k ← k + 1
until δ < threshold
return V_{k−1}
```

Do we really need to store the value of V_k for each k ??????

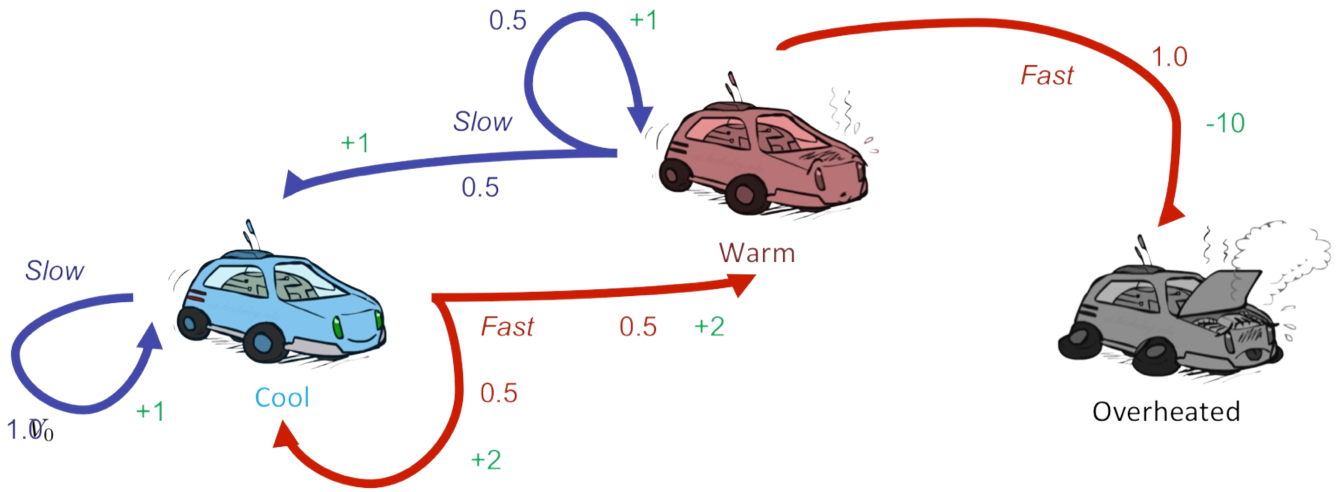
No. Use $V = V_{last}$ and $V' = V_{current}$

Does $V_{k+1}(s) \geq V_k(s)$ always hold ????????????????

No. If $T(s,a,s') = 1$ and $R(s,a,s') < 0$, then $V_1(s) = R(s,a,s') < 0$

EXAMPLE: VALUE ITERATION

V_2	3.5	2.5	0
V_1	2	1	0
	0	0	0



Assume no discount!

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

BELLMAN EQUATION VS VALUE ITERATION VS BELLMAN UPDATE

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 by applying Bellman update repeatedly

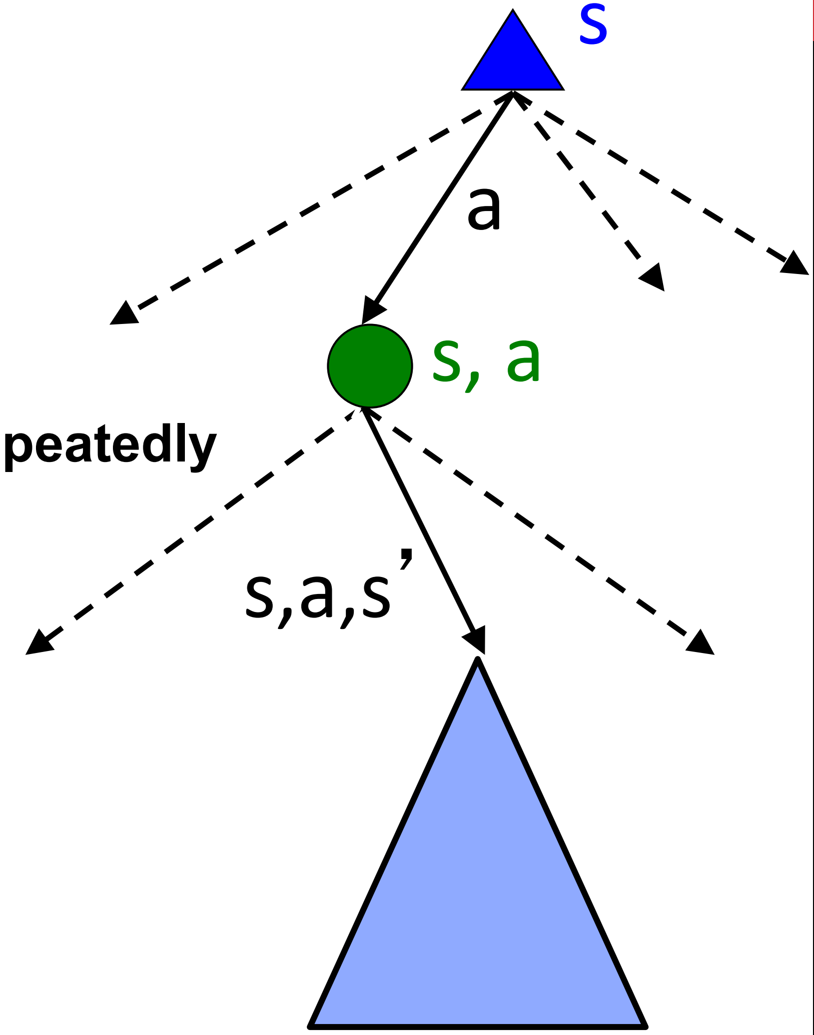
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s')\right]$$

Value iteration is a method for solving Bellman Equation

V_k **vectors are also interpretable as time-limited values**

Value iteration finds the fixed point of the function

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



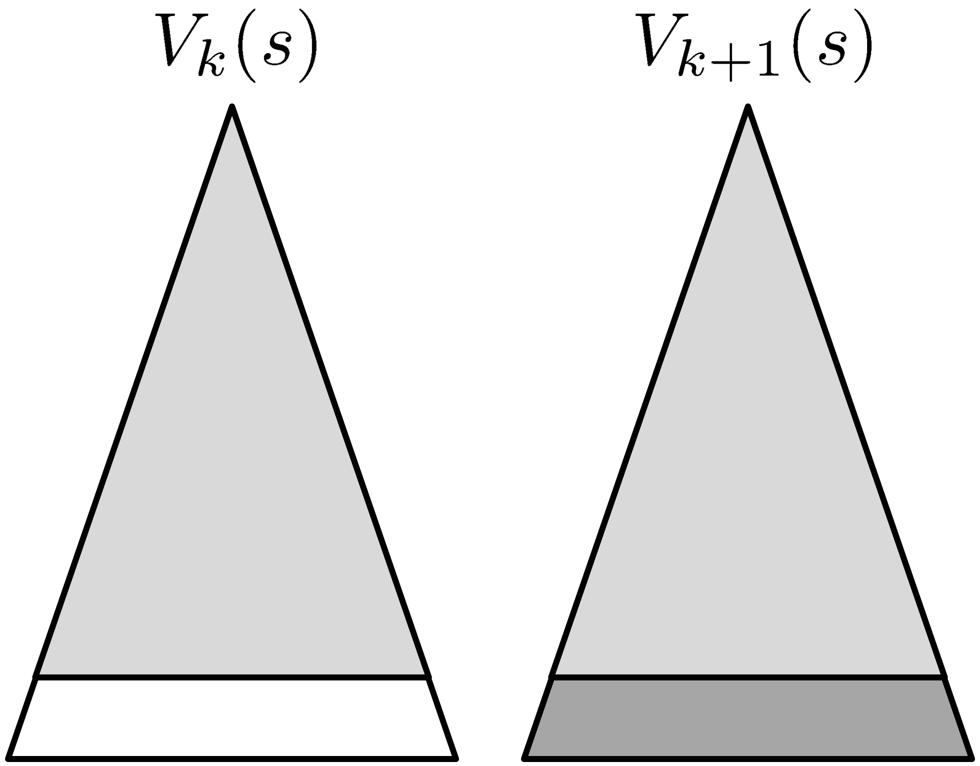
CONVERGENCE

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



OTHER WAYS TO SOLVE BELLMAN EQUATION?

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

Treat $V^*(s)$ as variables

Solve Bellman Equation through Linear Programming

Basic idea ????????

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

$\Downarrow$ (|A| constraints, one per action a)

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

SOLVING BELLMAN EQUATIONS USING LINEAR PROGRAMMING

Full linear program:

$$\min_{V^*} \sum_s V^*(s)$$

$$\text{s.t. } V^*(s) \geq \textstyle\sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')] \qquad \forall s, a$$

Why is this right ??????????

Assume not: after LP, suppose there exists a state s with strictly higher value:

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

(That is, minimizing the sum of values of states didn't bind to equality.)

Then we can find a better (i.e., lower) solution with only V(s) changed to make this an equality

- **All constraints for the other states are valid because their RHS only goes down! ><**

COOL THINGS ABOUT THE LP SOLUTION FOR BELLMAN EQUATIONS

1. Proof from previous slide holds if we optimize any linear function of V(s)! E.g.,

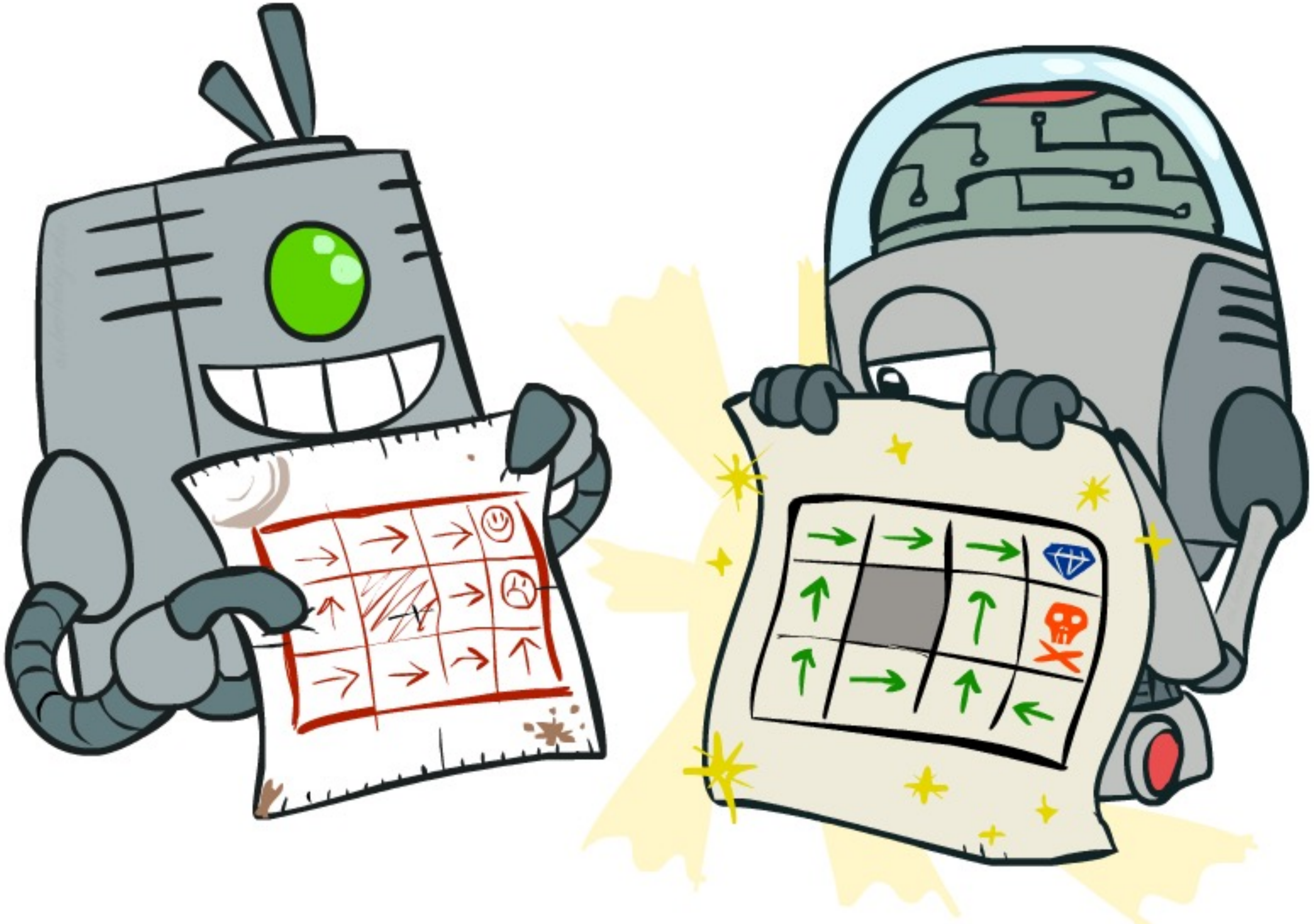
$$\min_{V^*} \sum_s p(s) V^*(s)$$

Would still find optimal policy, but the objective would represent the expected cumulative reward when the initial state is drawn as p(s).

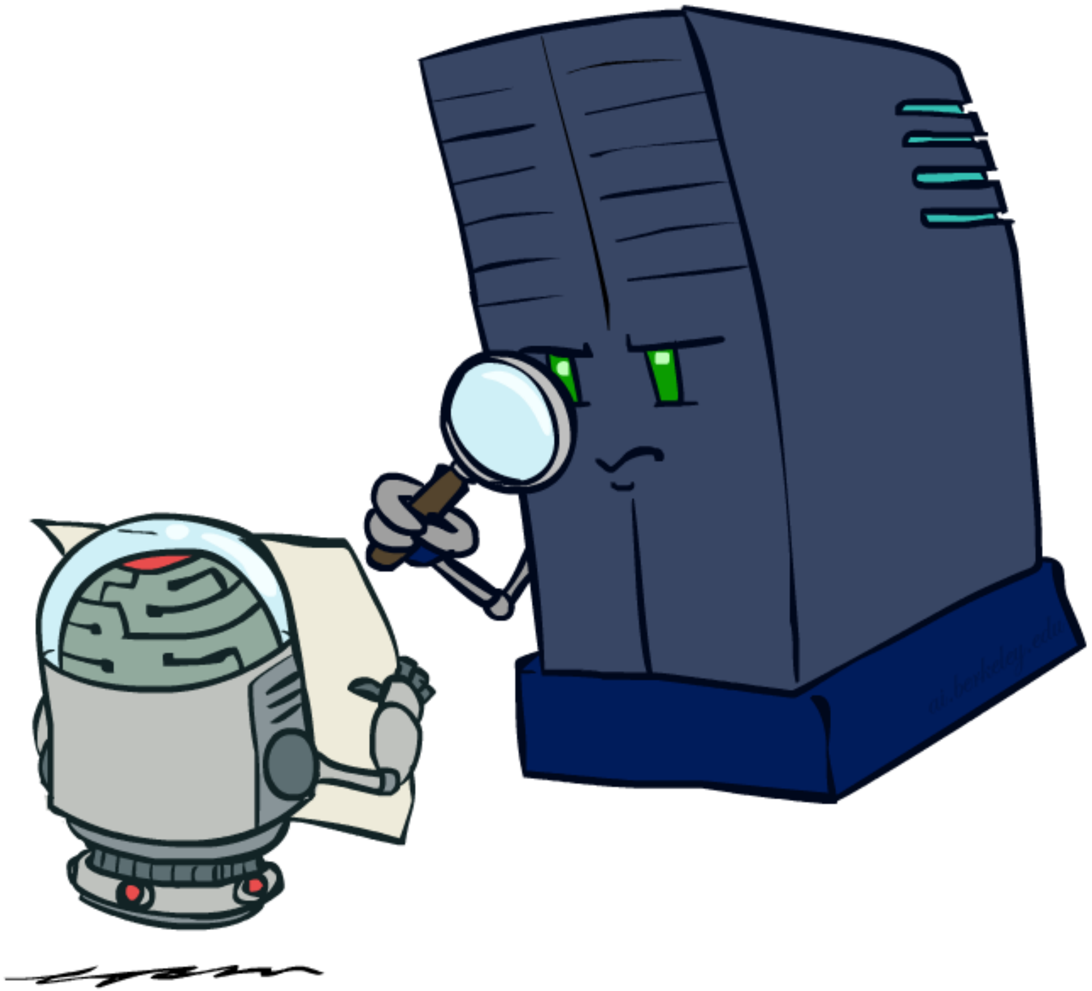
2. A “canonical” form of the dual simplex method is equivalent to policy iteration

3. Not the fastest method (specialized policy iteration methods often are), but can give nice bounds for e.g. state abstractions in large MDP solving

POLICY ITERATION FOR SOLVING MDPS

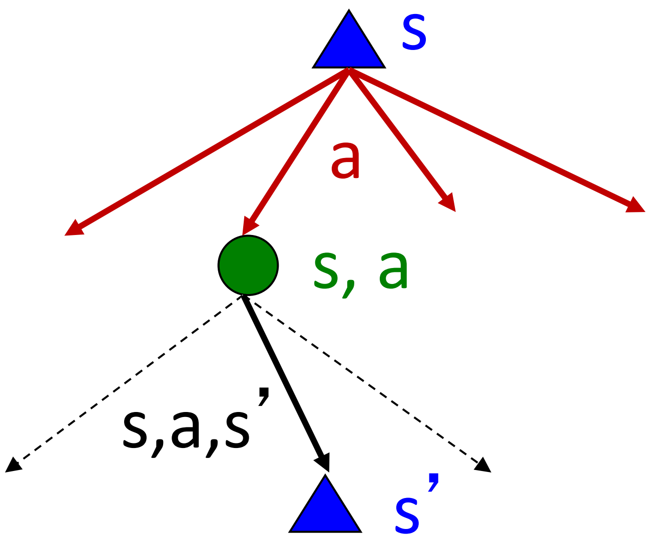


POLICY EVALUATION

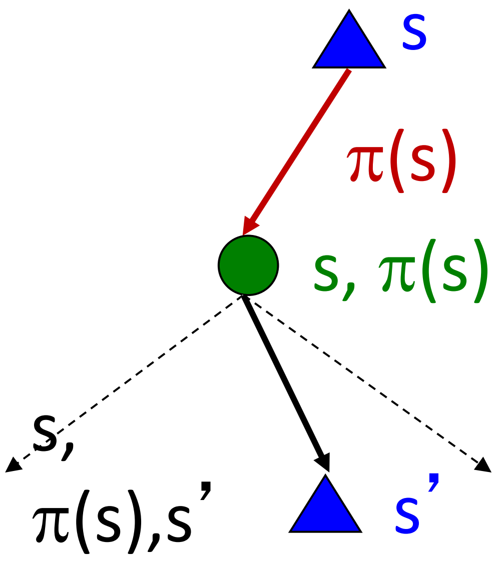


FIXED POLICIES

Do the optimal action



Do what π says to do

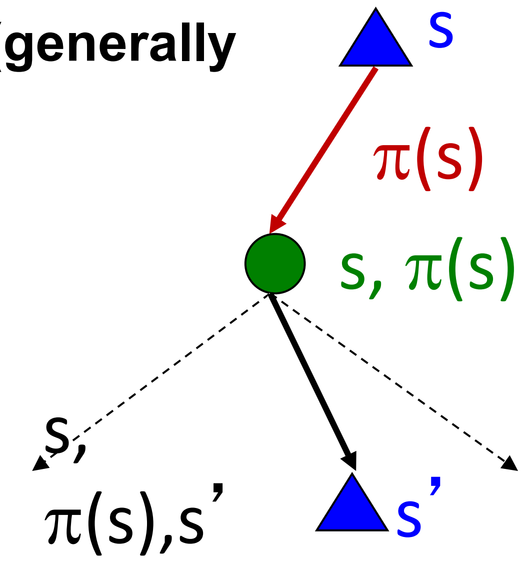


Expectimax trees max over all actions to compute the optimal values

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



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):

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

COMPARE

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

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

RECALL: MDP OPTIMAL QUANTITIES

A policy π: map of states to actions

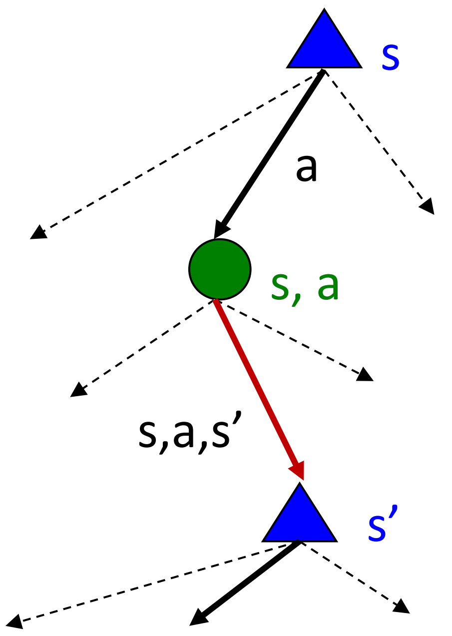
- The optimal policy π*: π*(s) = optimal action from state s

Value function of a policy $V^{\pi}(s)$: expected utility starting in s and acting according to π

- Optimal value function V*: V*(s) = $V^{\pi^*}(s)$

Q function of a policy $Q^{\pi}(s)$: expected utility starting out having taken action a from state s and (thereafter) acting according to π

- Optimal Q function Q* : Q*(s,a) = $Q^{\pi^*}(s)$



s is a *state*

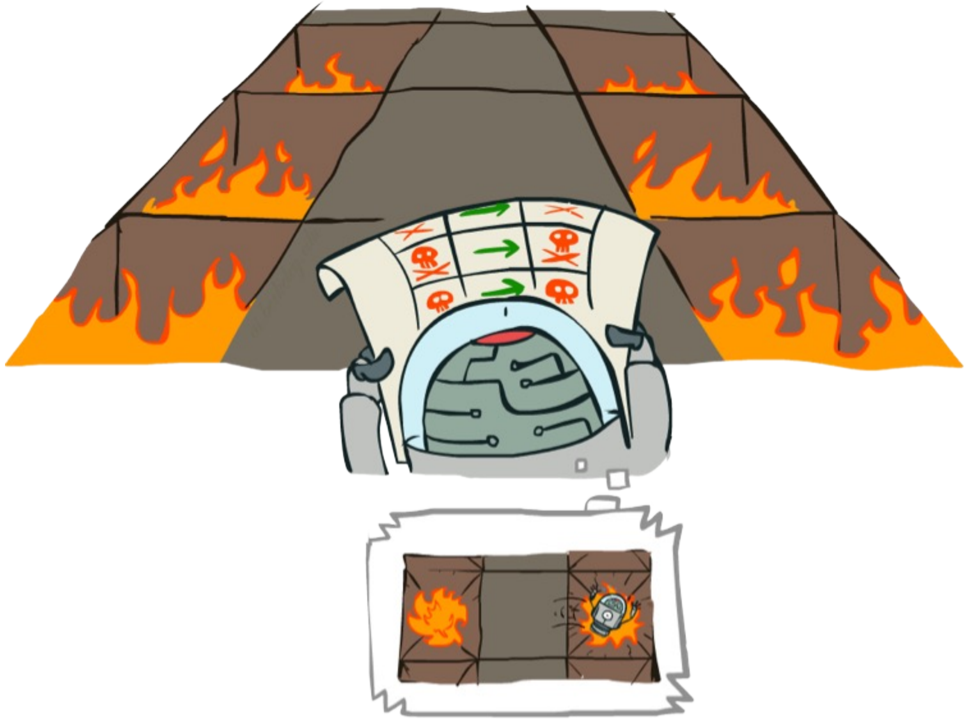
(s, a) is a *state-action pair*

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

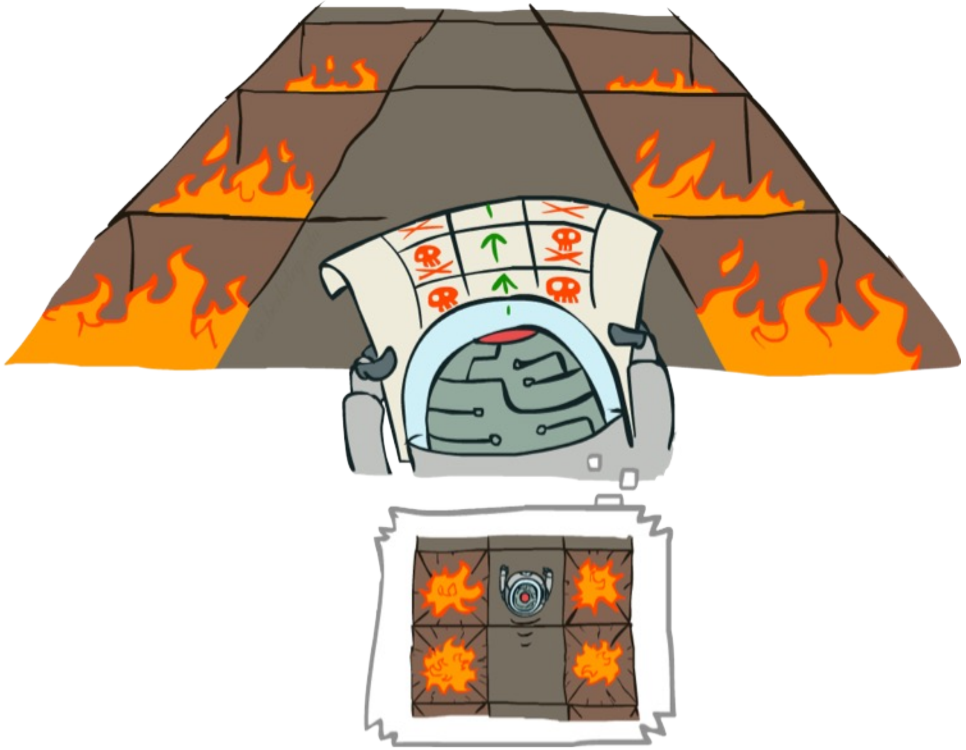
Solve MDP: Find π^*, V^* and/or Q^*

EXAMPLE: POLICY EVALUATION

Always Go Right



Always Go Forward



EXAMPLE: POLICY EVALUATION

Always Go Right

Always Go Forward

POLICY EVALUATION

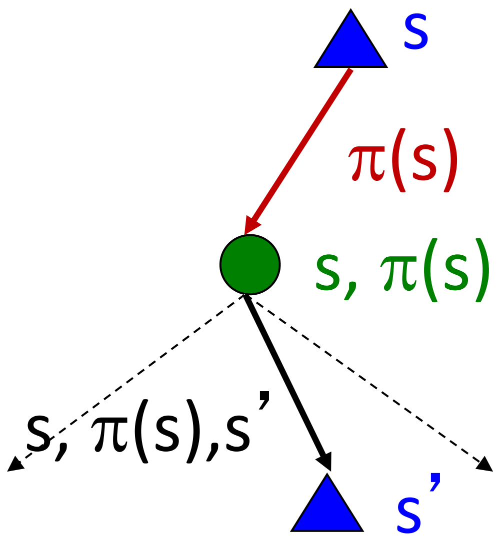
How do we calculate the V's for a fixed policy π?

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

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

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

O(|S²|) time per iteration



POLICY EVALUATION

Idea 2: Bellman Equation w.r.t. a given policy π defines a linear system

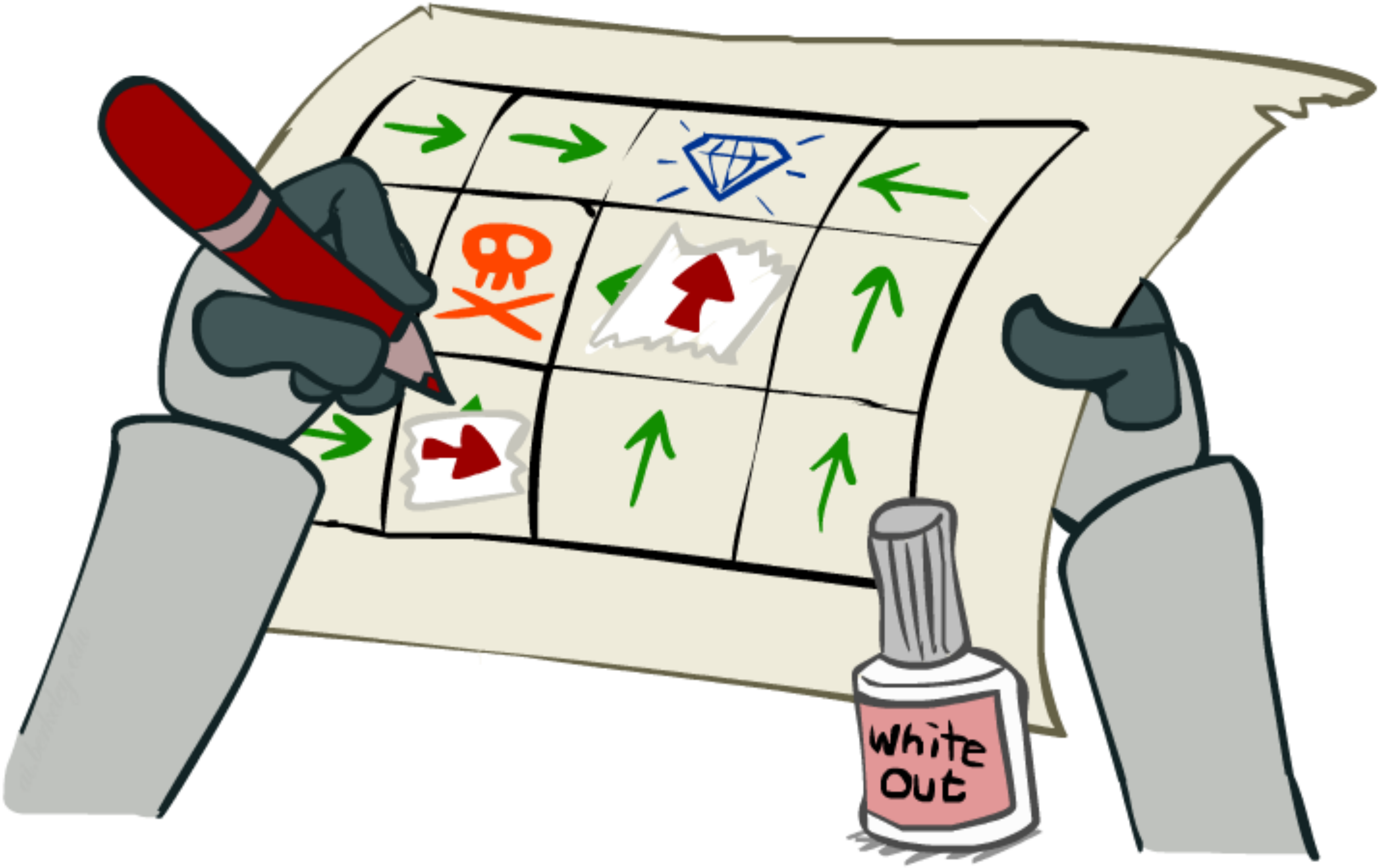
- Solve with your favorite linear system solver!

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

|S| variables – each state has a unique value under a policy π

|S| constraints – one equality per state to compute that state's value

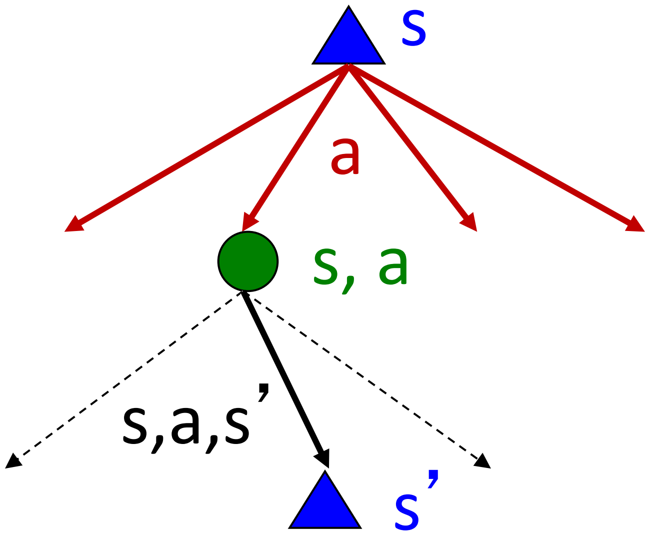
POLICY ITERATION



PROBLEMS WITH VALUE ITERATION

Value iteration repeats the Bellman updates:

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



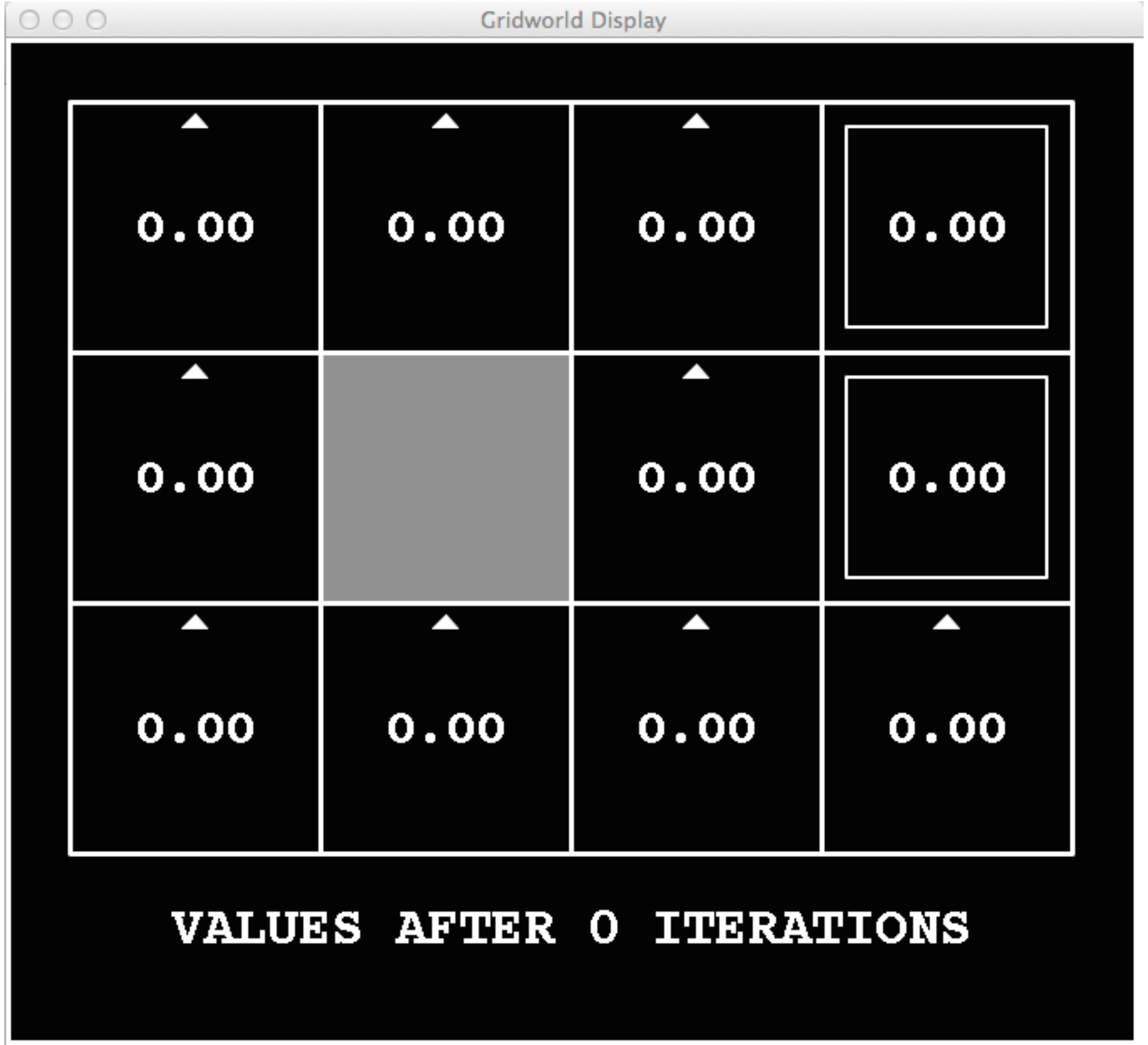
Problem 1: It's slow – O($|S|^2|A|$) per iteration

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

Problem 3: The policy often converges long before the values

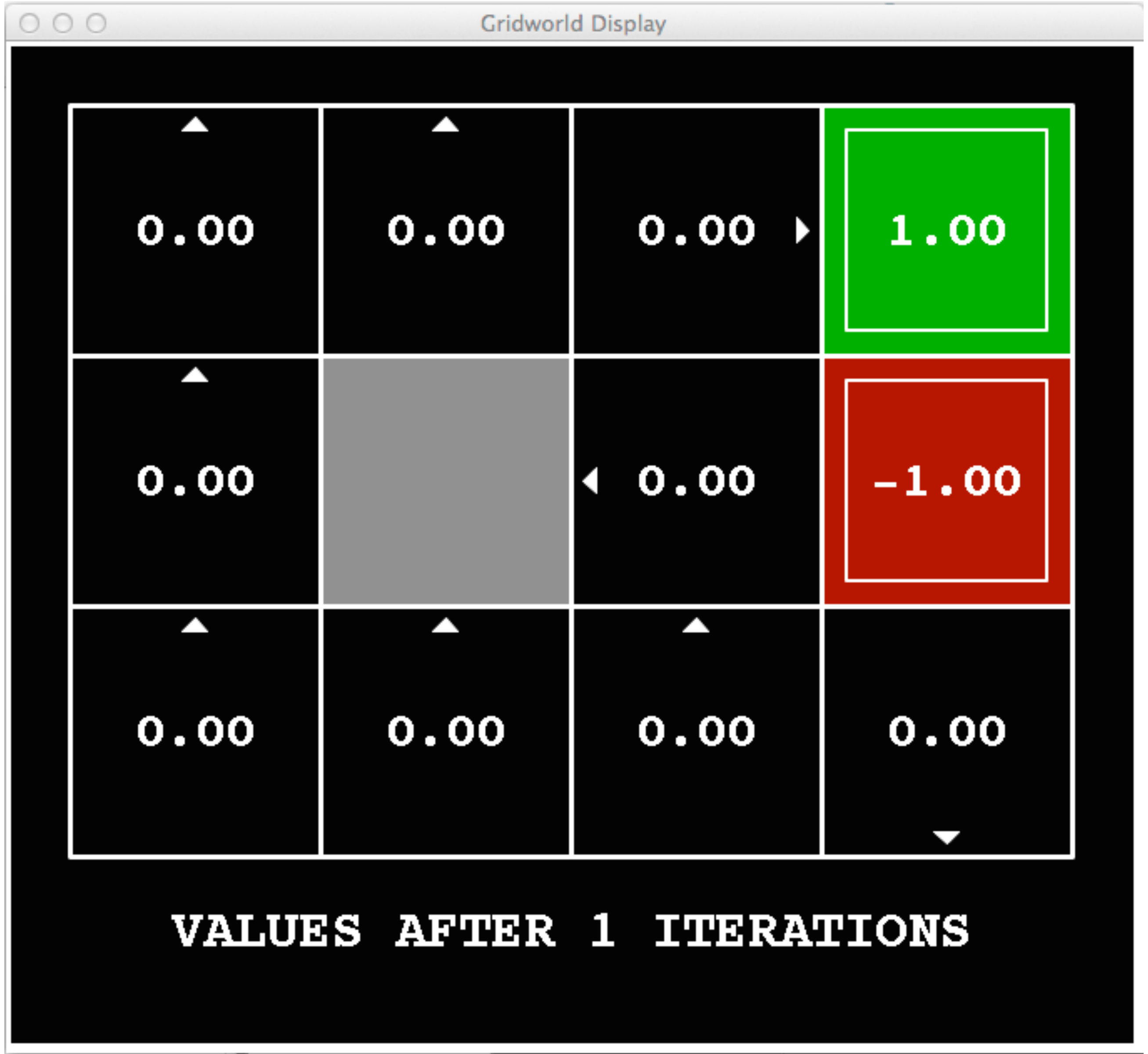
Compare: values of states versus policy (i.e., “arrows in the boxes”) over many iterations

K=0



Noise = 0.2
Discount = 0.9
Living reward = 0

K=1



Noise = 0.2
Discount = 0.9
Living reward = 0

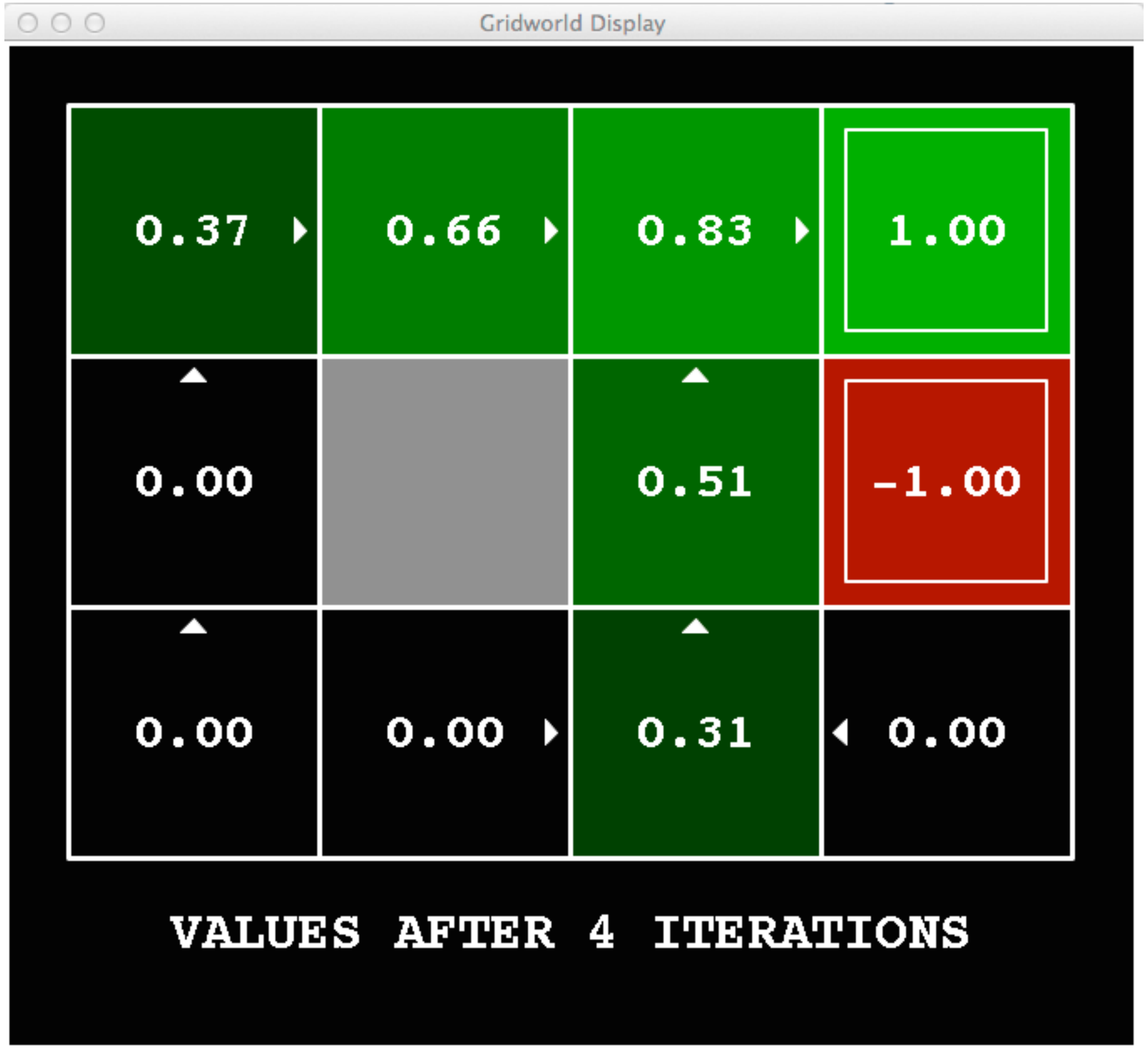
K=2

Noise = 0.2
Discount = 0.9
Living reward = 0

K=3

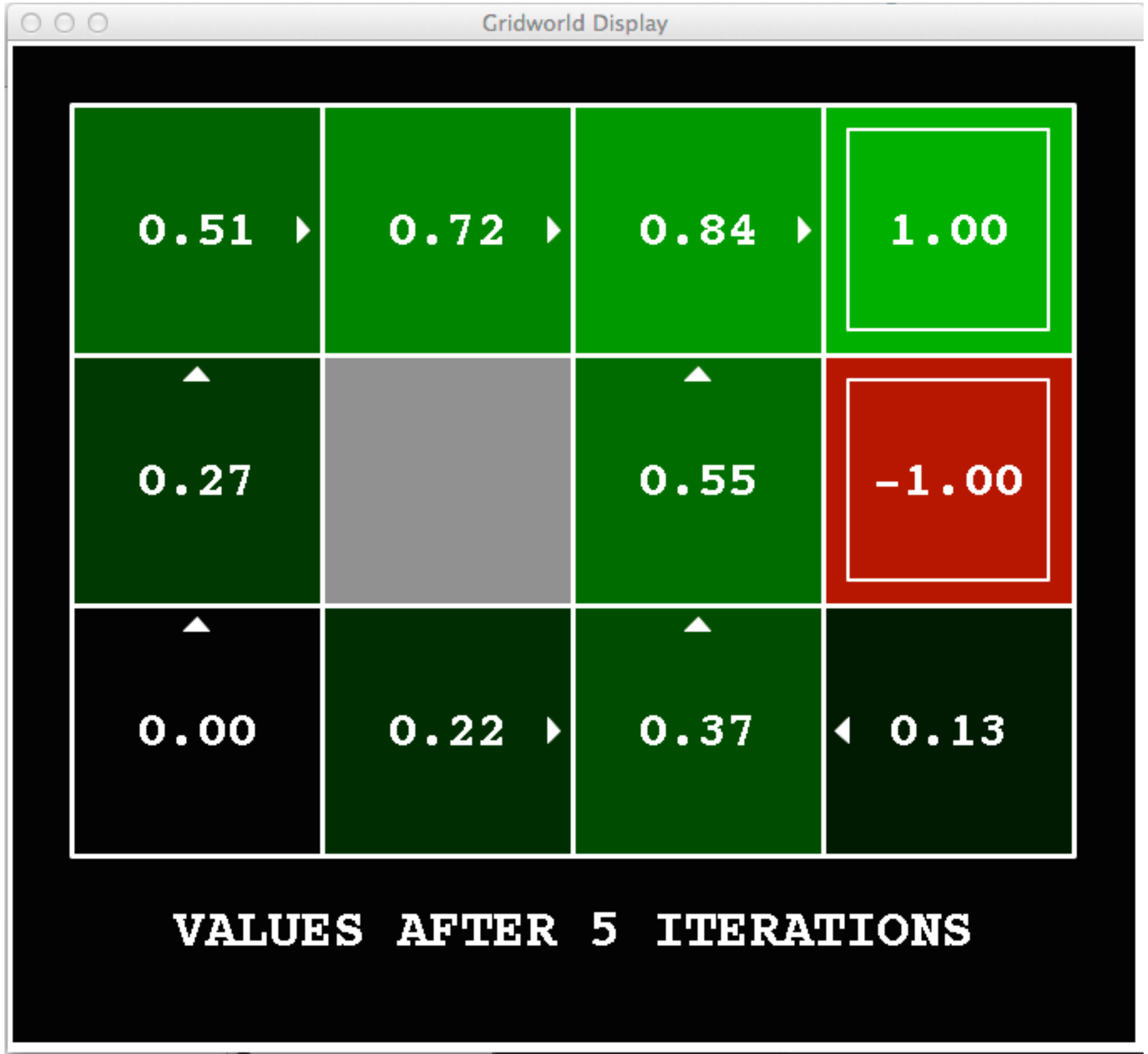
Noise = 0.2
Discount = 0.9
Living reward = 0

K=4



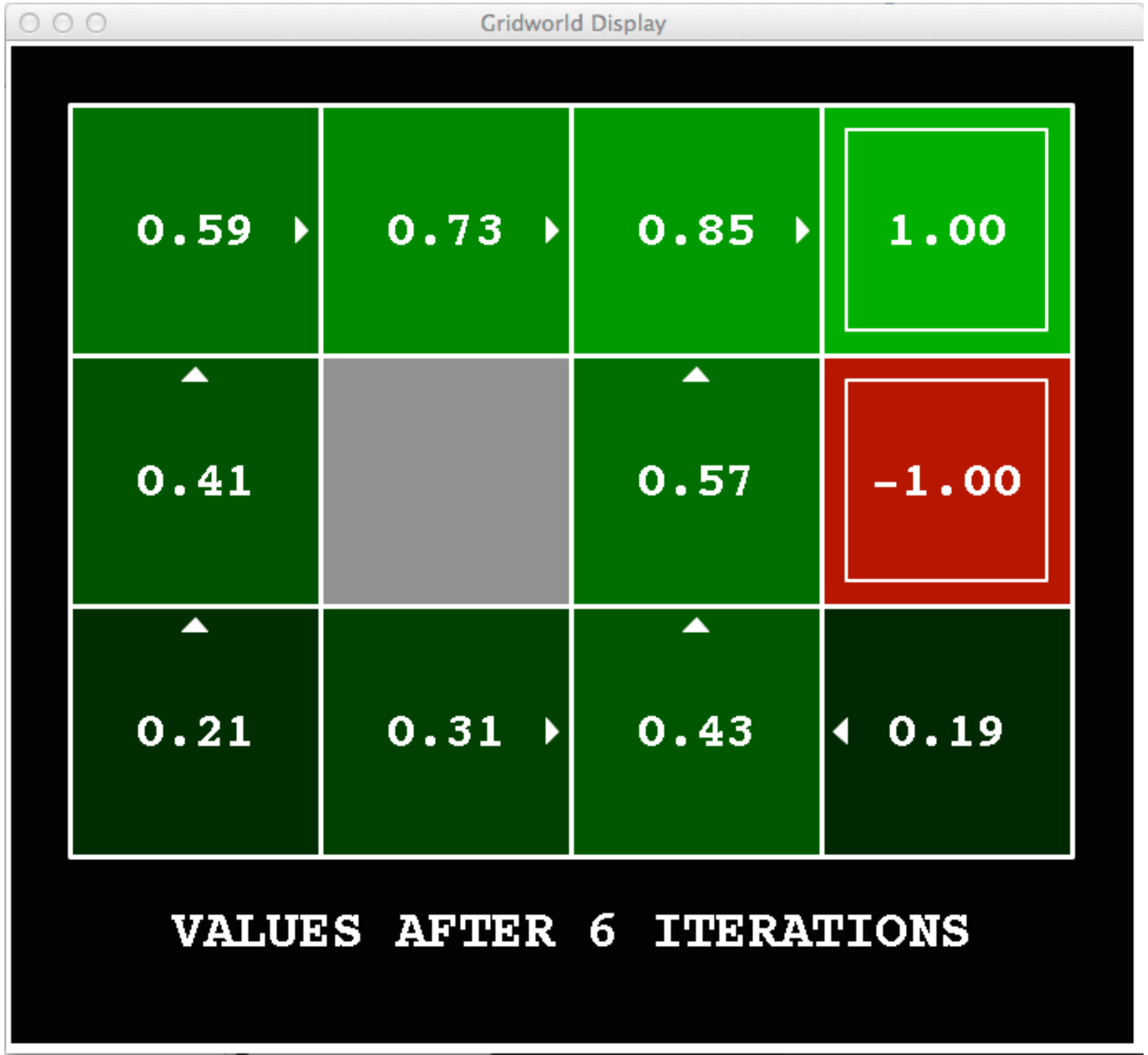
Noise = 0.2
Discount = 0.9
Living reward = 0

K=5



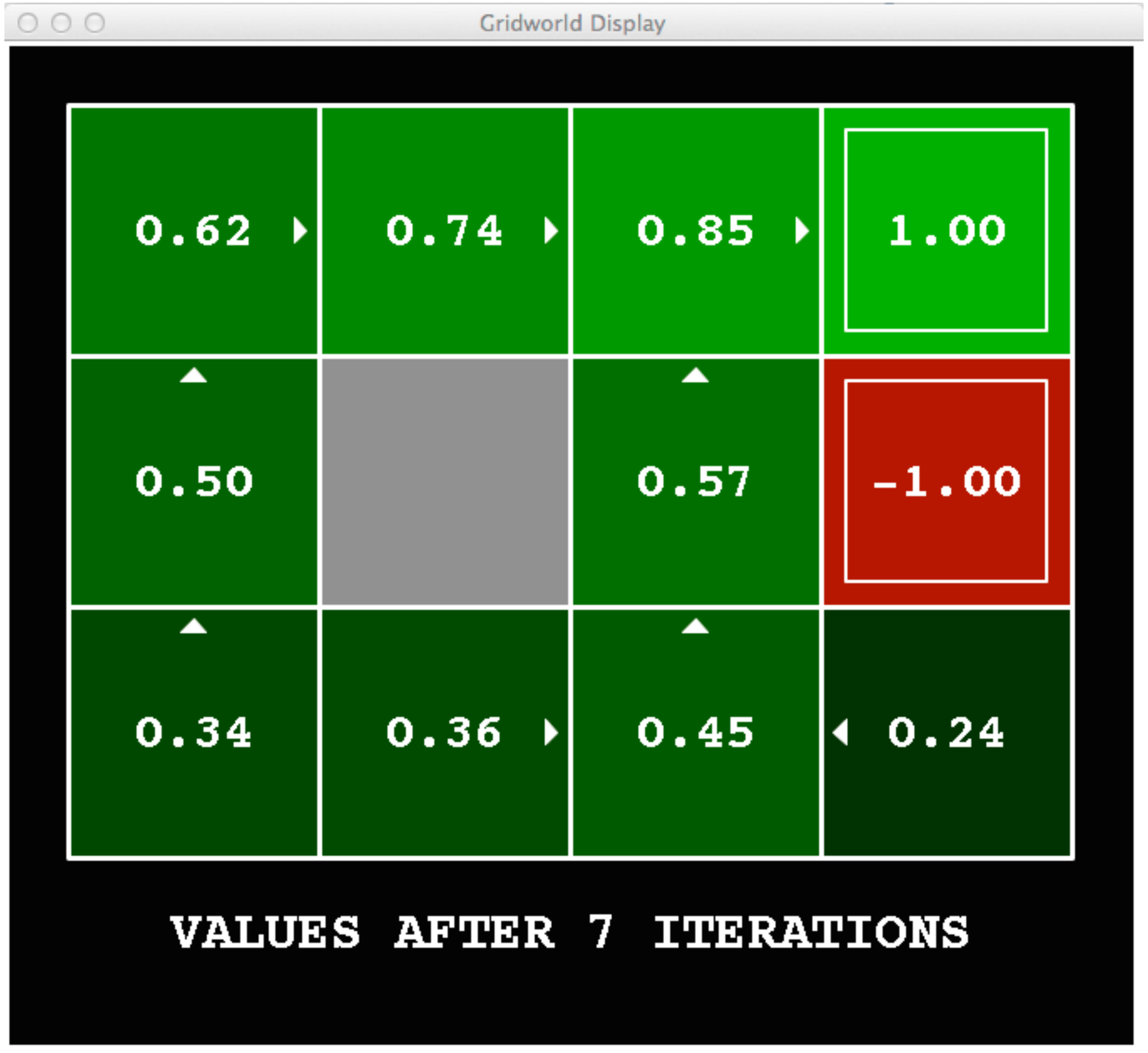
Noise = 0.2
Discount = 0.9
Living reward = 0

K=6



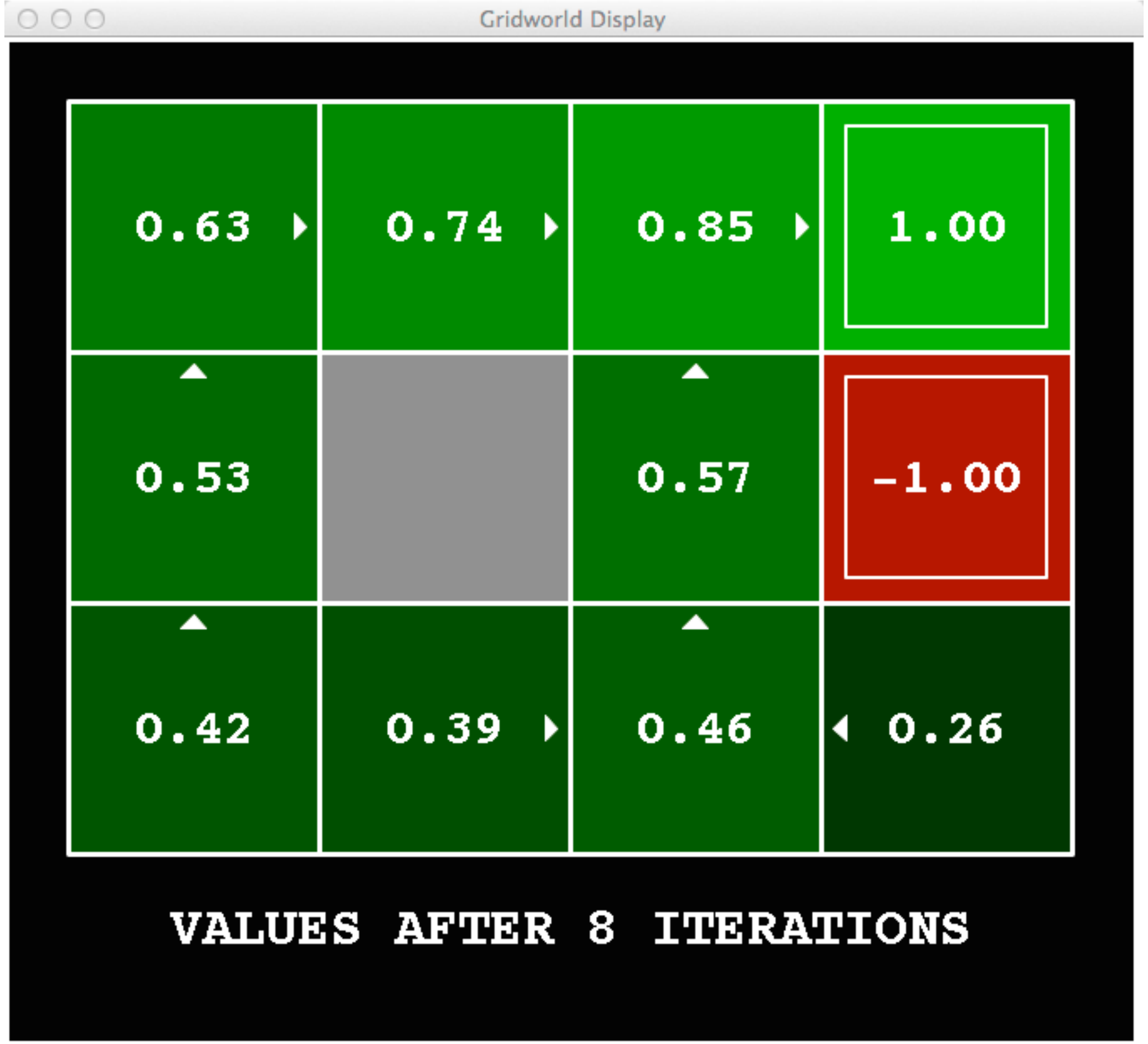
Noise = 0.2
Discount = 0.9
Living reward = 0

K=7



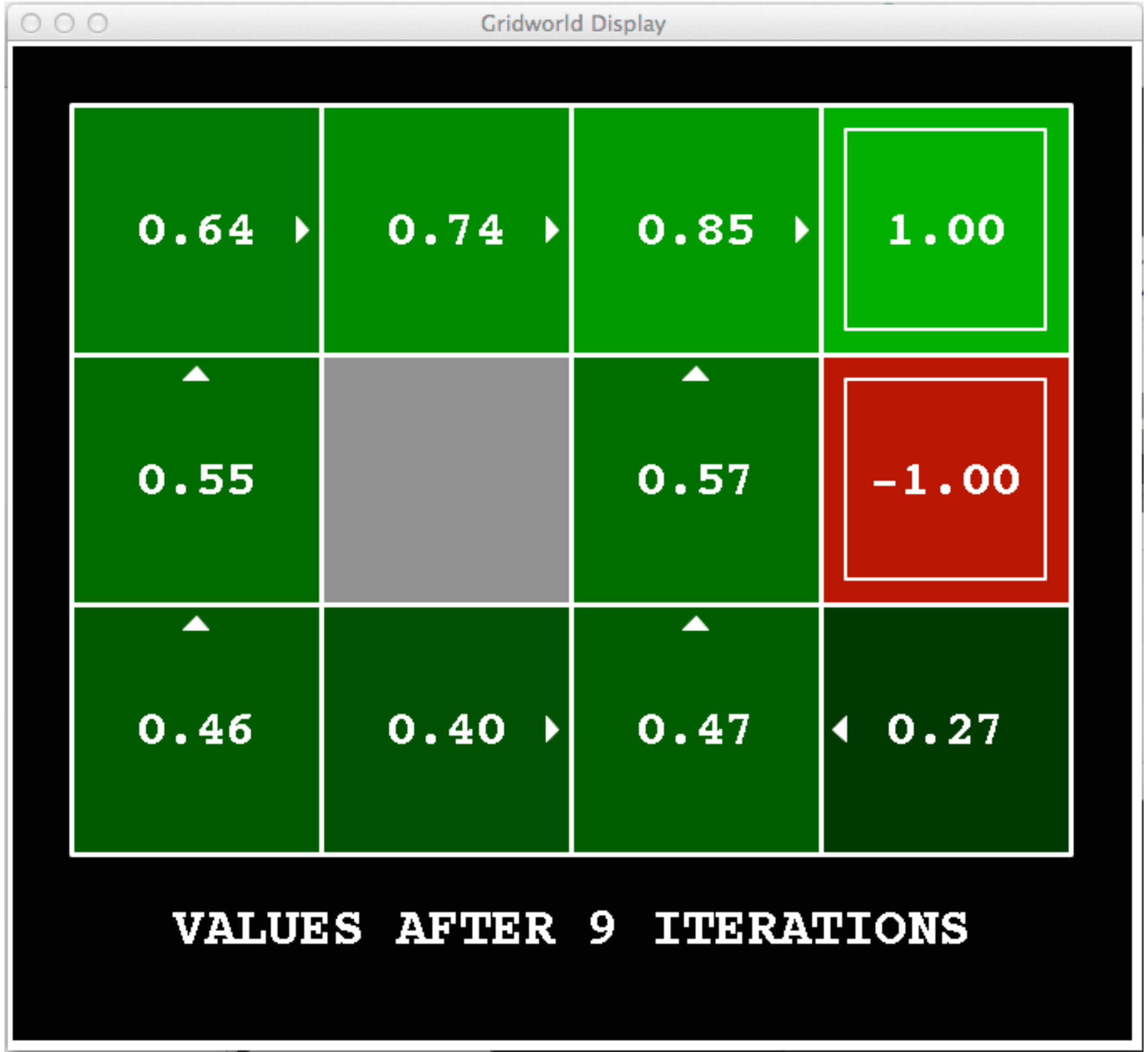
Noise = 0.2
Discount = 0.9
Living reward = 0

K=8



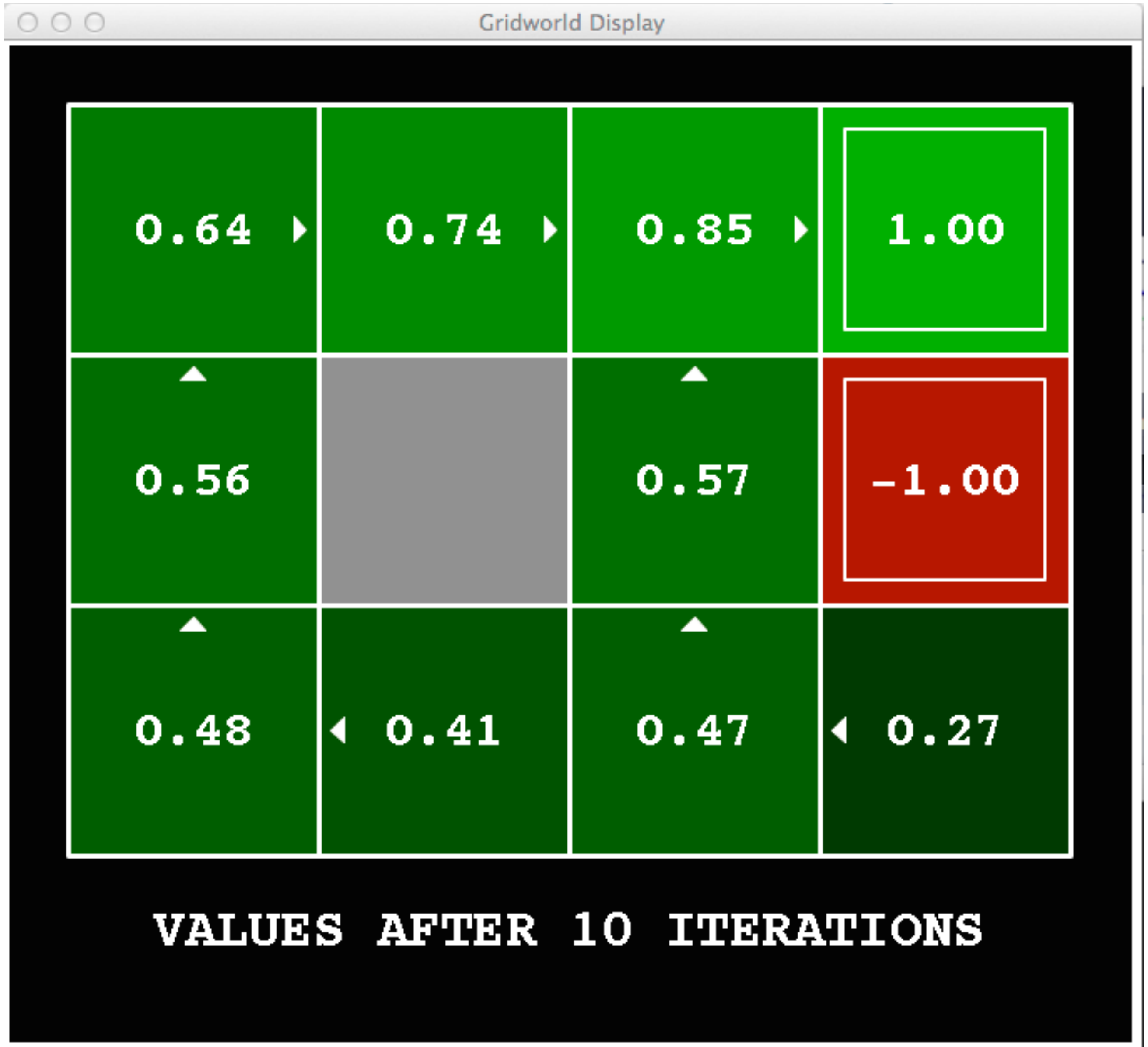
Noise = 0.2
Discount = 0.9
Living reward = 0

K=9



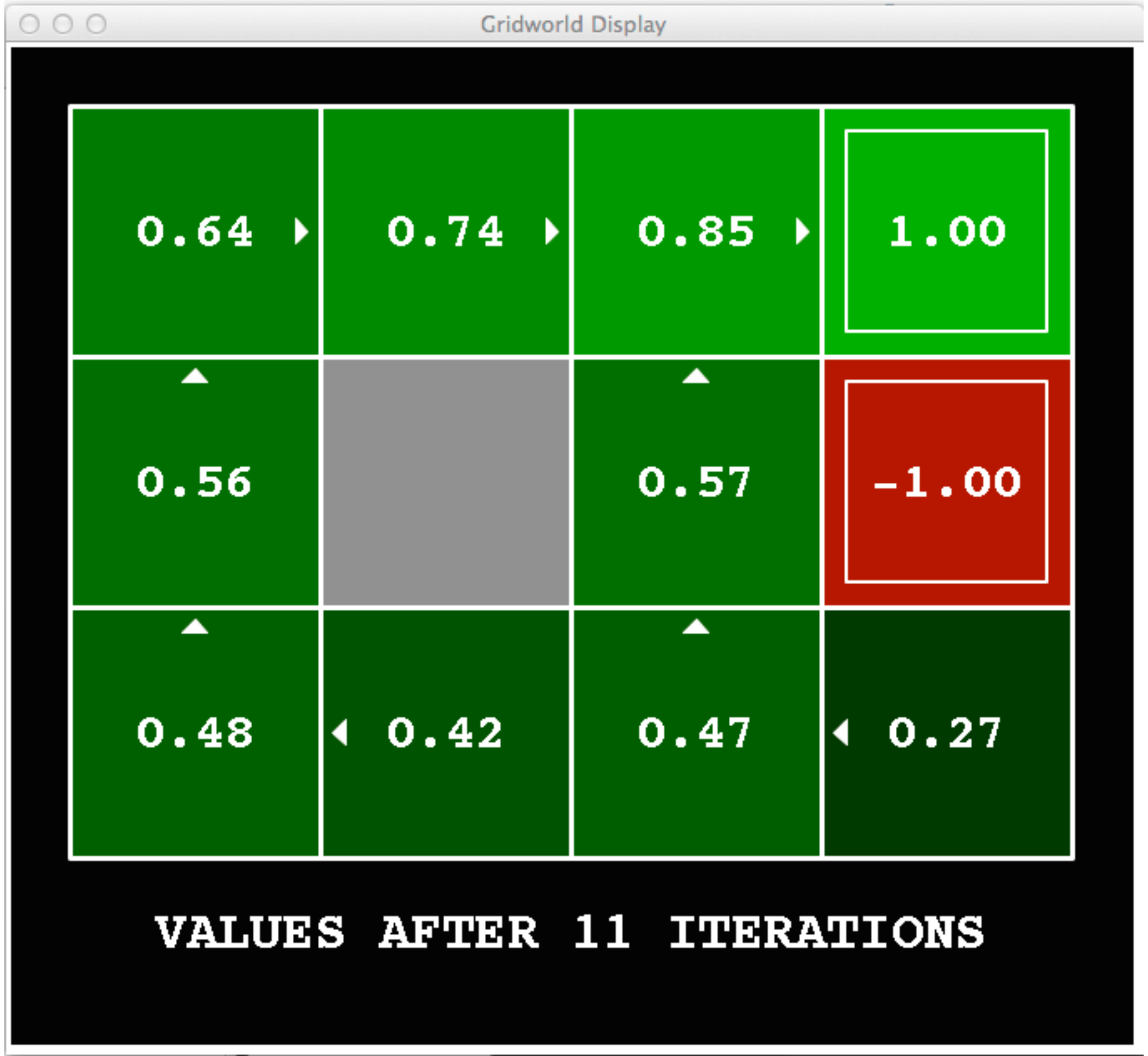
Noise = 0.2
Discount = 0.9
Living reward = 0

K=10



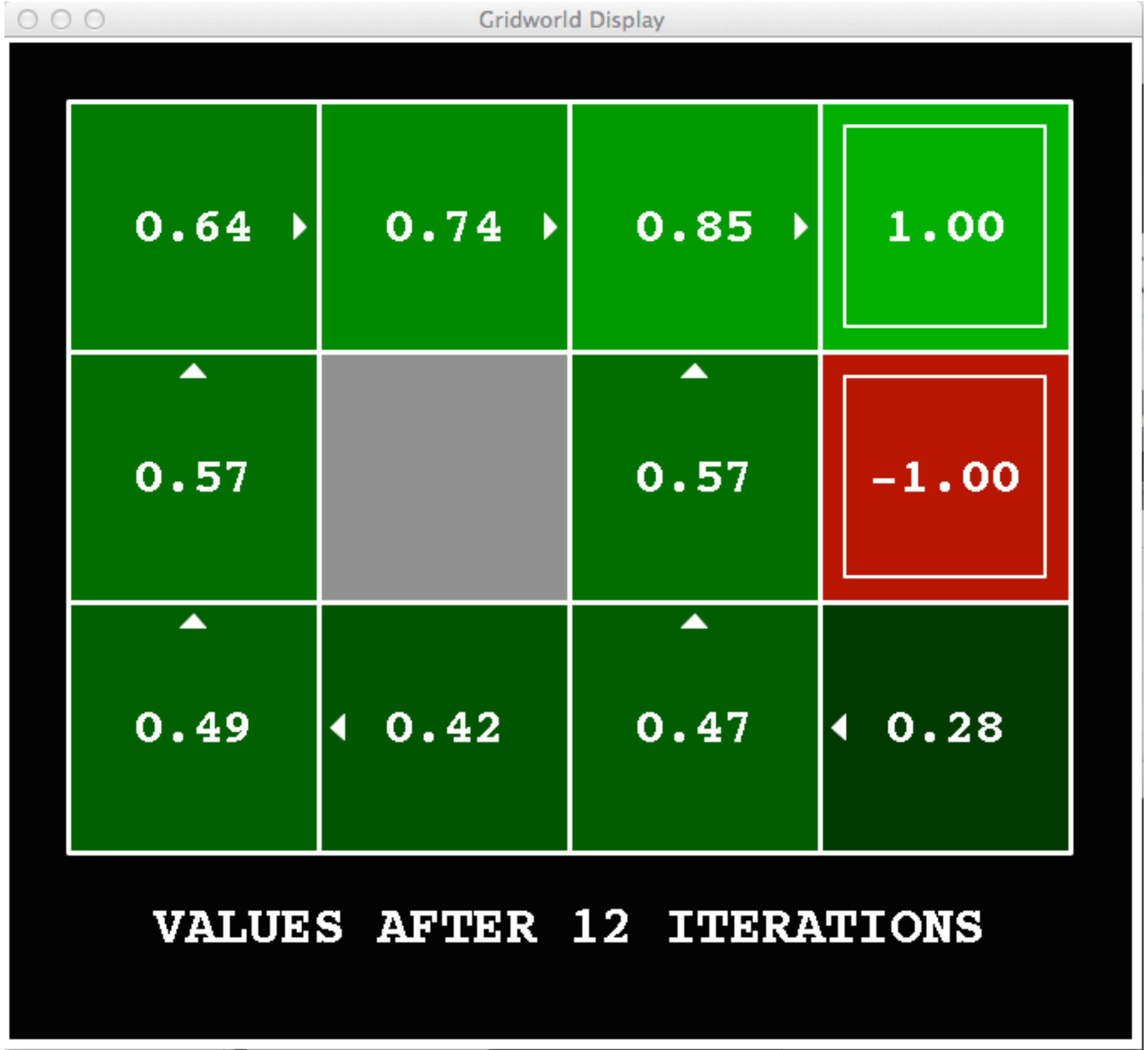
Noise = 0.2
Discount = 0.9
Living reward = 0

K=11



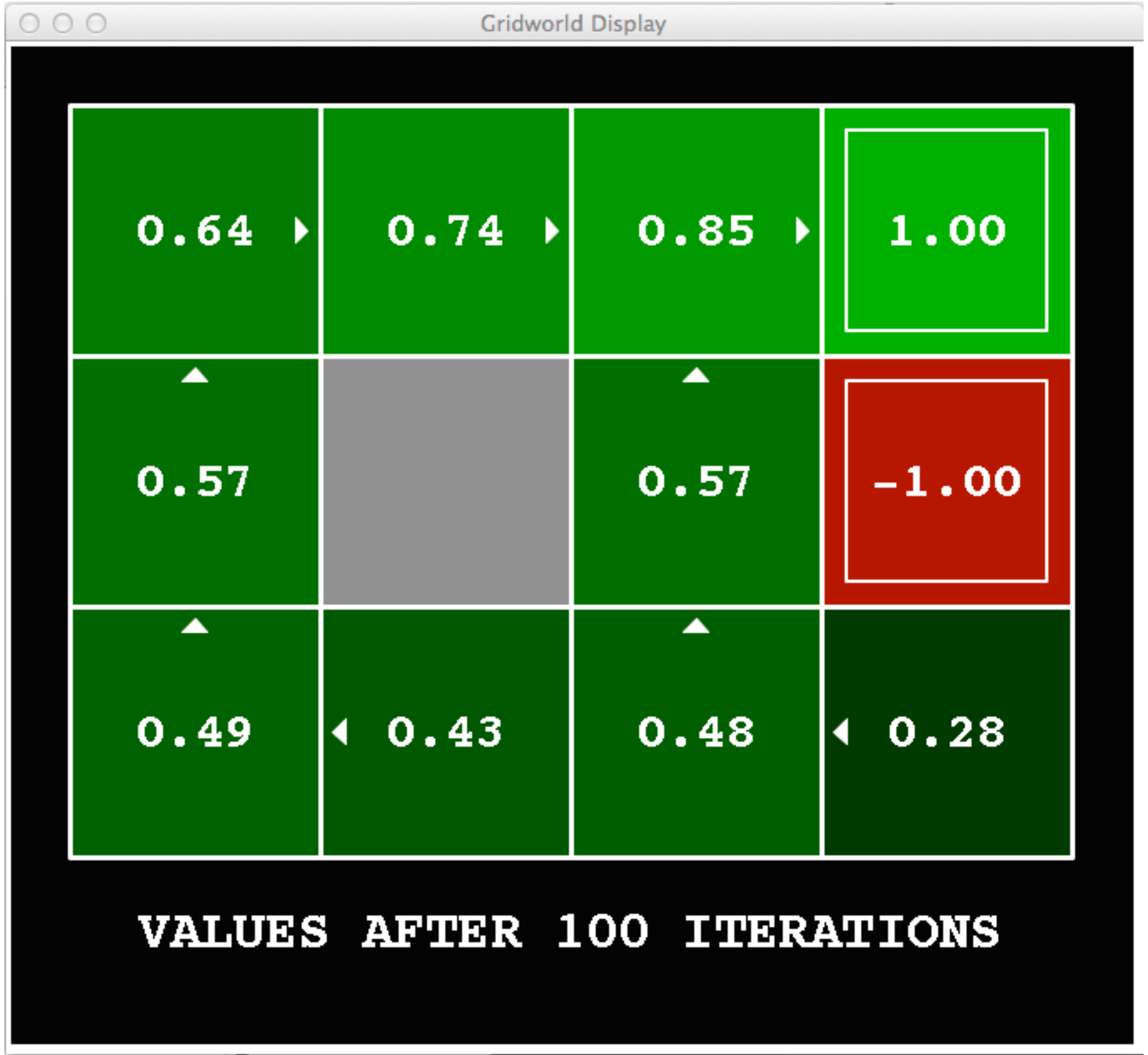
Noise = 0.2
Discount = 0.9
Living reward = 0

K=12



Noise = 0.2
Discount = 0.9
Living reward = 0

K=100



Noise = 0.2
Discount = 0.9
Living reward = 0

POLICY ITERATION

Alternative approach for optimal values:

- Step 1: Policy evaluation: calculate utilities for some fixed policy (may not be optimal!) 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

Policy Evaluation: For fixed current policy π, find values w.r.t. the policy

- Iterate until values converge:

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

Policy Improvement: For fixed values, get a better policy with one-step look-ahead:

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

Similar to how you derive optimal policy π^* given optimal value V^*

COMPARISON OF "VI" AND "PI"

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….

- Turn values into a policy: use one-step lookahead
- Compute optimal values: use value iteration or policy iteration
- Compute values for a particular policy: use policy evaluation

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

FINAL SLIDE: MDP NOTATION

Standard expectimax: $V(s) = \max_a \sum_{s'} P(s'|s,a)V(s')$

Bellman equations: $V(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')]$

Value iteration: $V_{k+1}(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V_k(s')], \quad \forall\, s$

Q-iteration: $Q_{k+1}(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')], \quad \forall\, s,a$

Policy extraction: $\pi_V(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')], \quad \forall\, s$

Policy evaluation: $V_{k+1}^{\pi}(s) = \sum_{s'} P(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma V_k^{\pi}(s')], \quad \forall\, s$

Policy improvement: $\pi_{new}(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^{\pi_{old}}(s')], \quad \forall\, s$