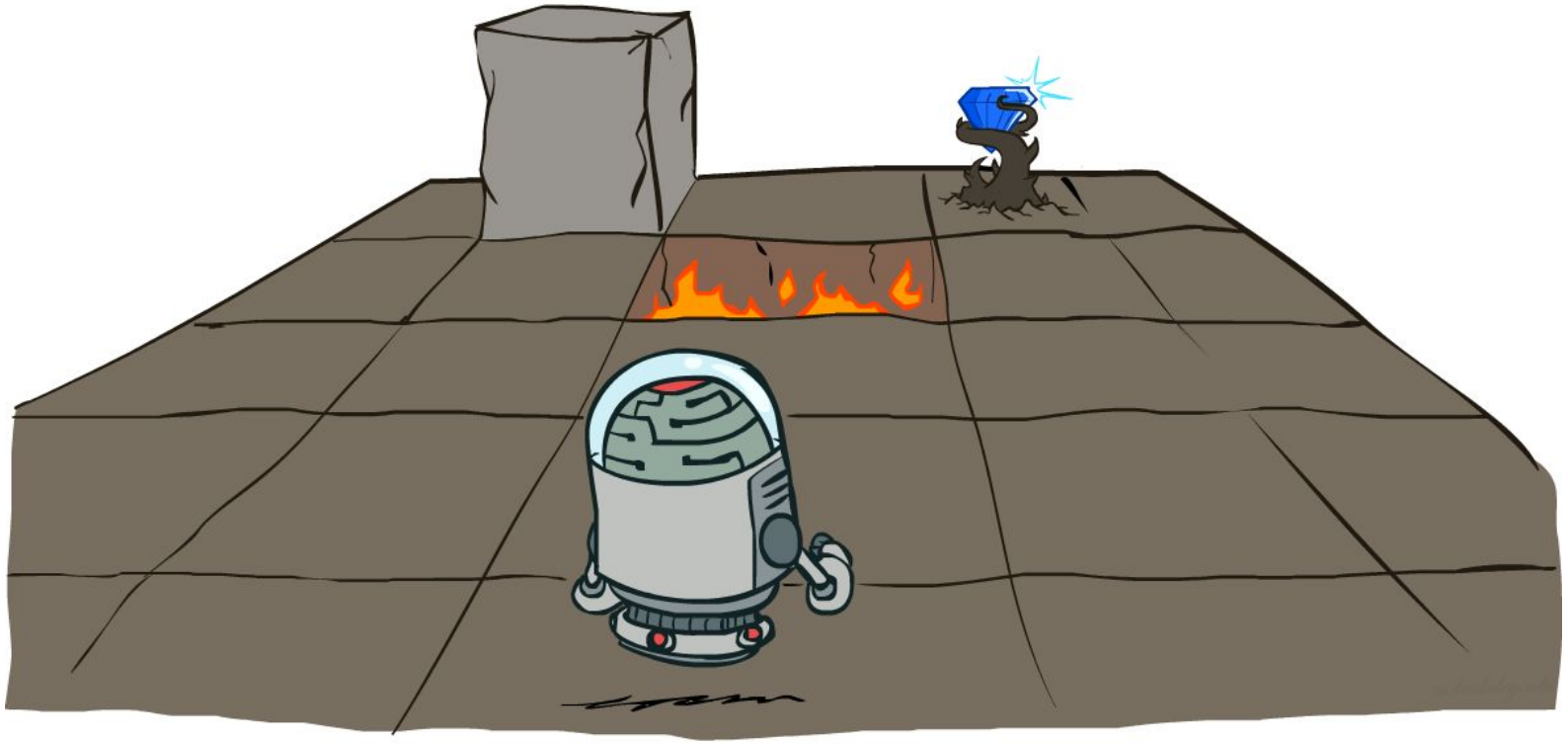
# Markov Decision Processes



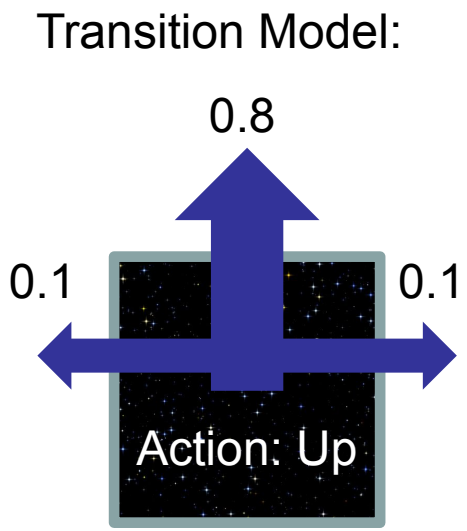
Slides courtesy of Dan Klein and Pieter Abbeel

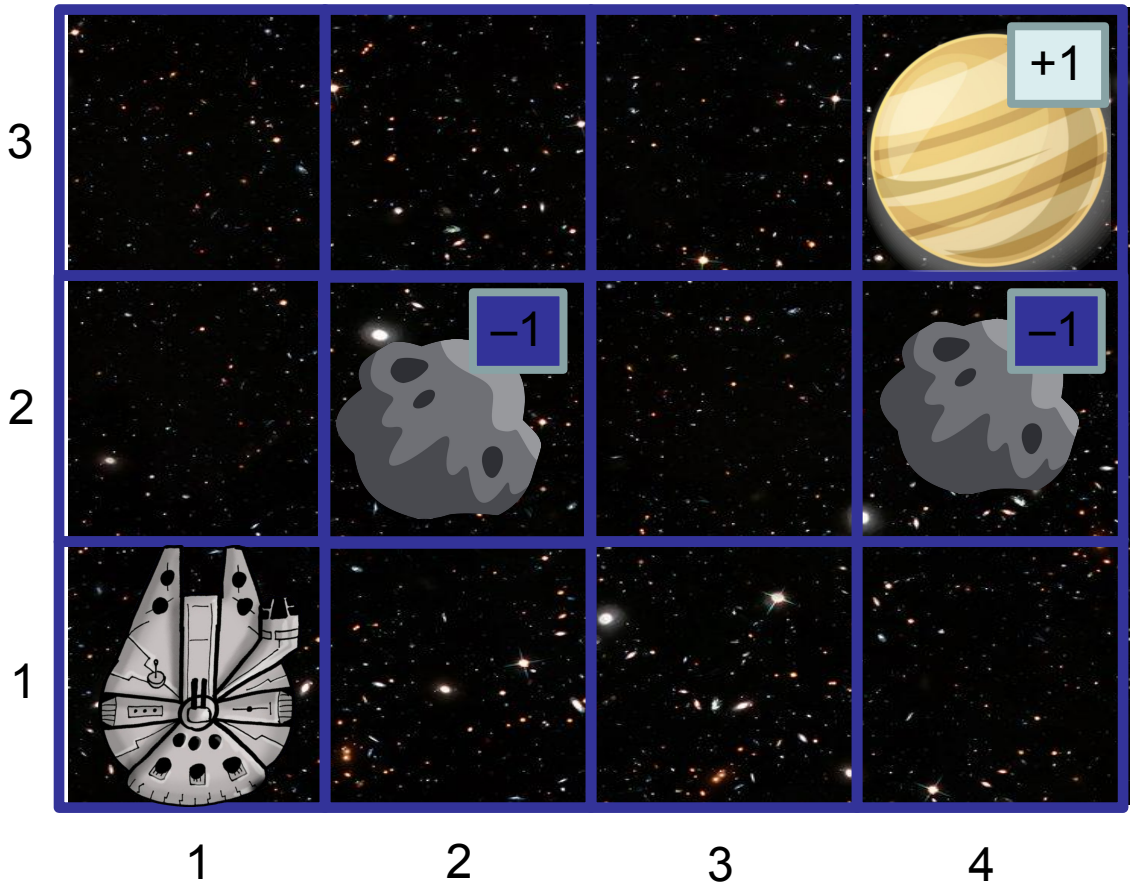
University of California, Berkeley

[These slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkeley.edu.]

# Stochastic Search Problems

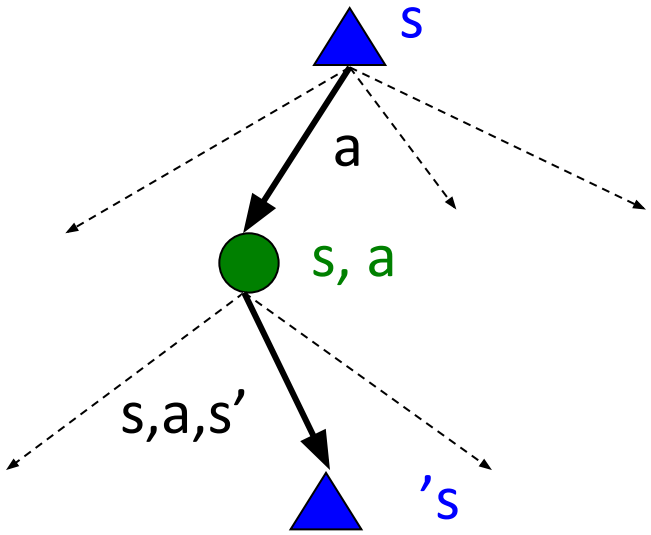
- Instead of dealing with situations where the environment deterministic, MDPs deal with **stochastic** environments.



# Defining MDPs

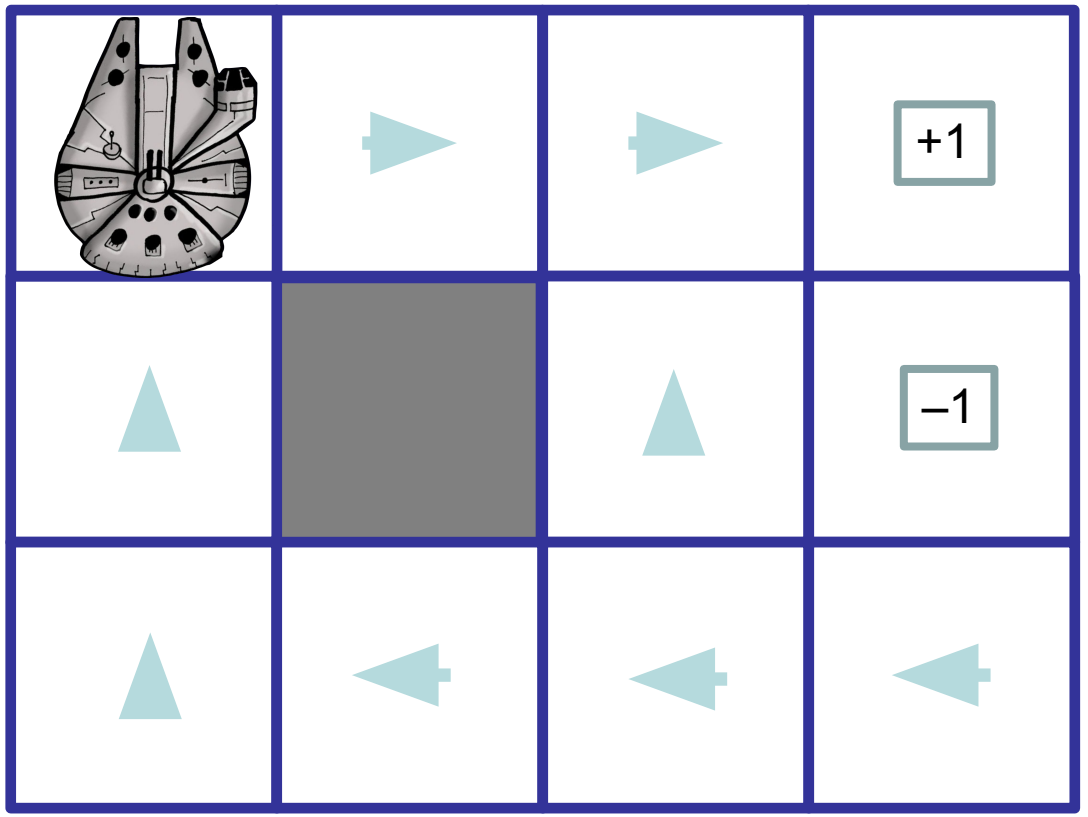
- Markov decision processes:
  - Set of states S
  - Start state $s_0$
  - Set of actions A
  - Transitions P(s’|s,a) (or T(s,a,s’))
  - Rewards R(s,a,s’) (and discount γ)




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

# Solution == Policy

- In search problems a solution was **a plan**: a sequence of action that corresponded to the shortest path from the start to a goal.
- Because of the non-determinism in MDPs we cannot simply give a sequence of actions.
- Instead, the solution to an MDP is a **policy.** A policy maps from a state onto the action to take if the agent is in that state.
    - $\pi(s) = a$

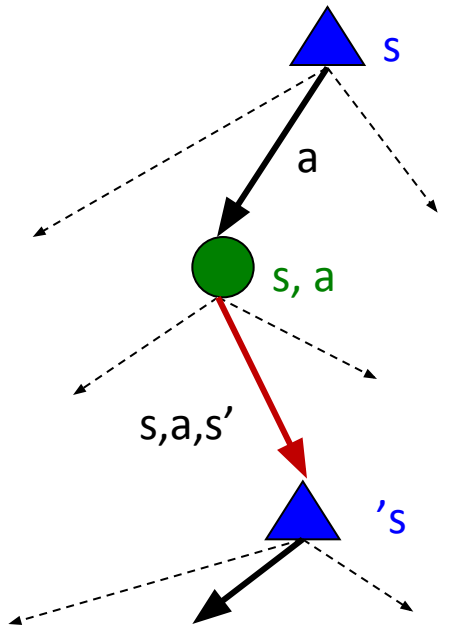


# Optimal Quantities

- 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

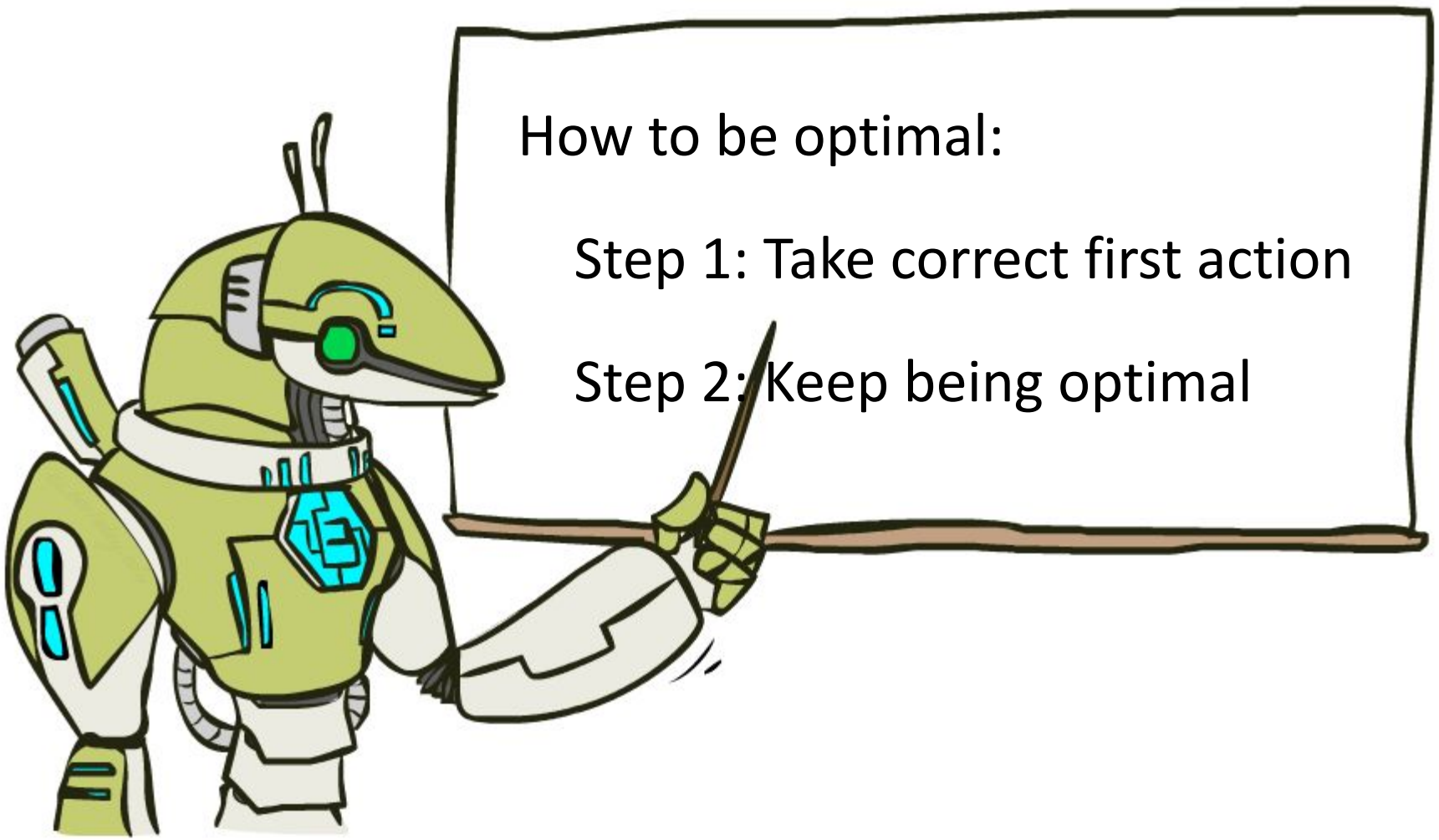



s is a *state*

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

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

# The Bellman Equations



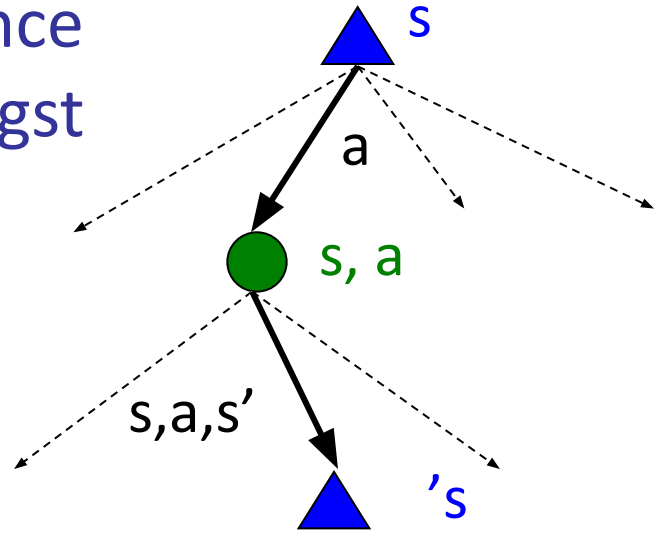
# The Bellman Equations

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

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

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

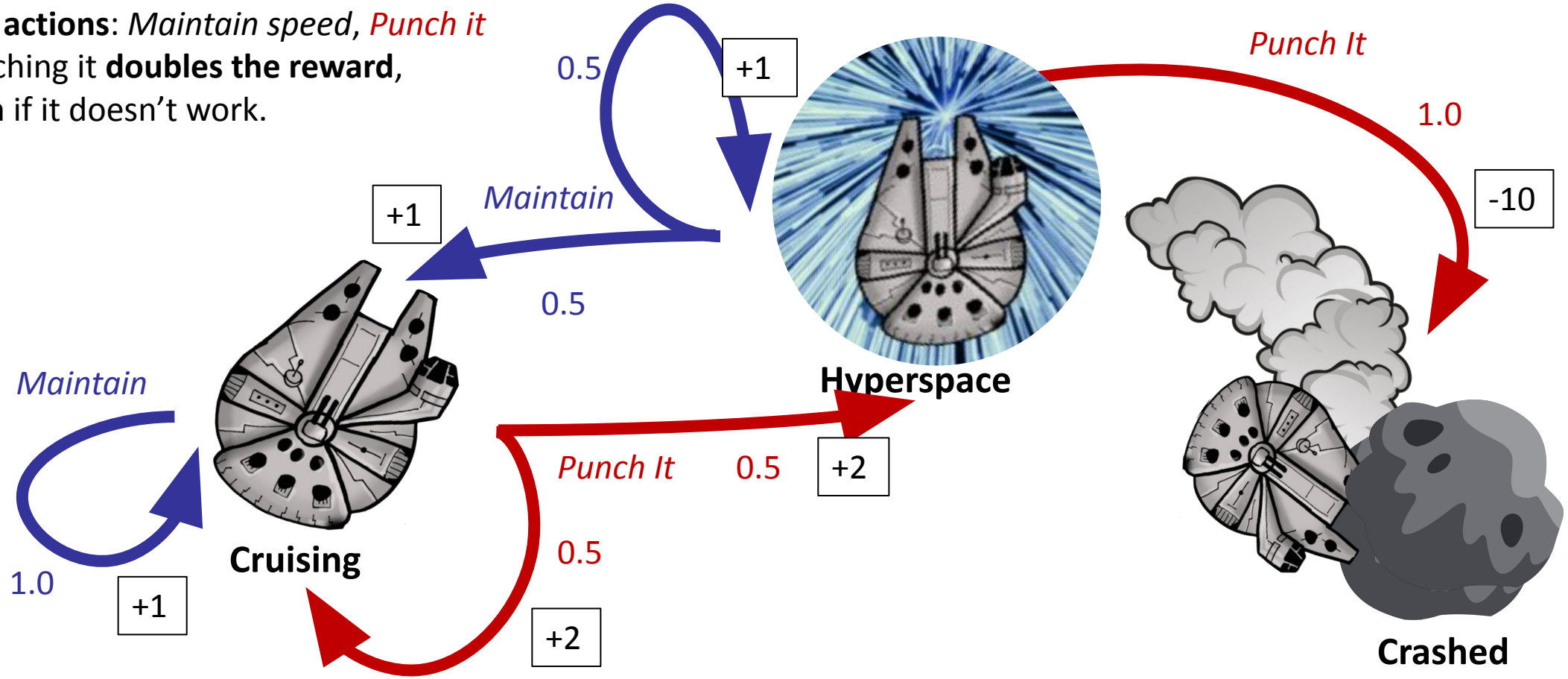
$$V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[ R(s,a,s') + \gamma V^*(s') \right]$$




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

# Example Hyperdrive MDP

The Millennium Falcon needs to travel far far away, quickly
Three **states**: Cruising, Hyperspace, Crashed
Two **actions**: *Maintain speed*, *Punch it*
Punching it **doubles the reward**,
even if it doesn't work.

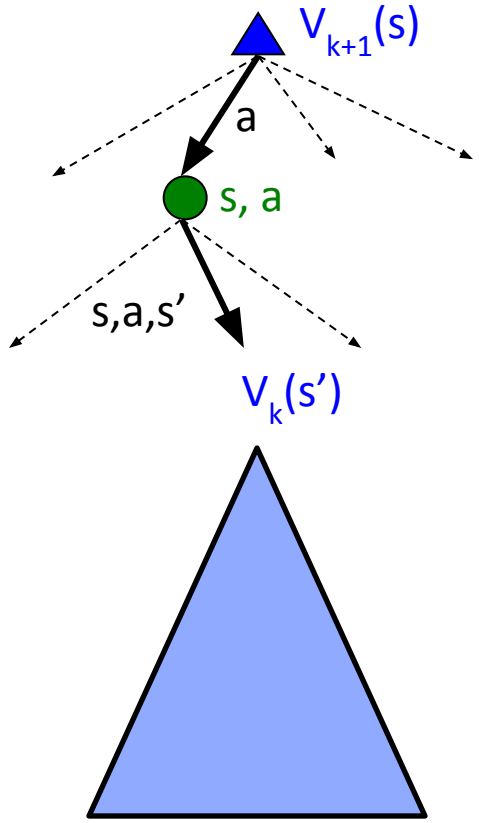


# Value Iteration

- 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:

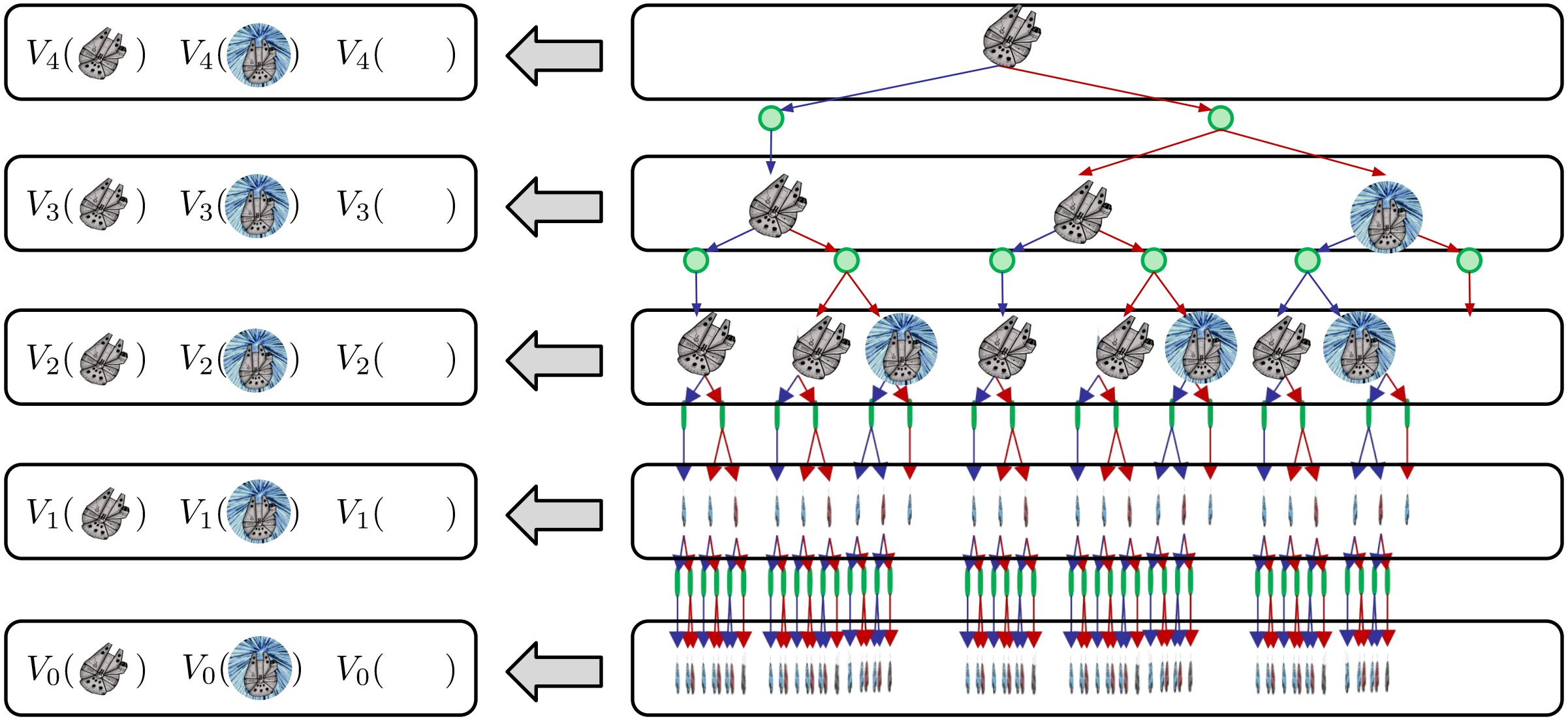
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[ R(s,a,s') + \gamma V_k(s') \right]$$

- 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

# Computing Time-Limited Values

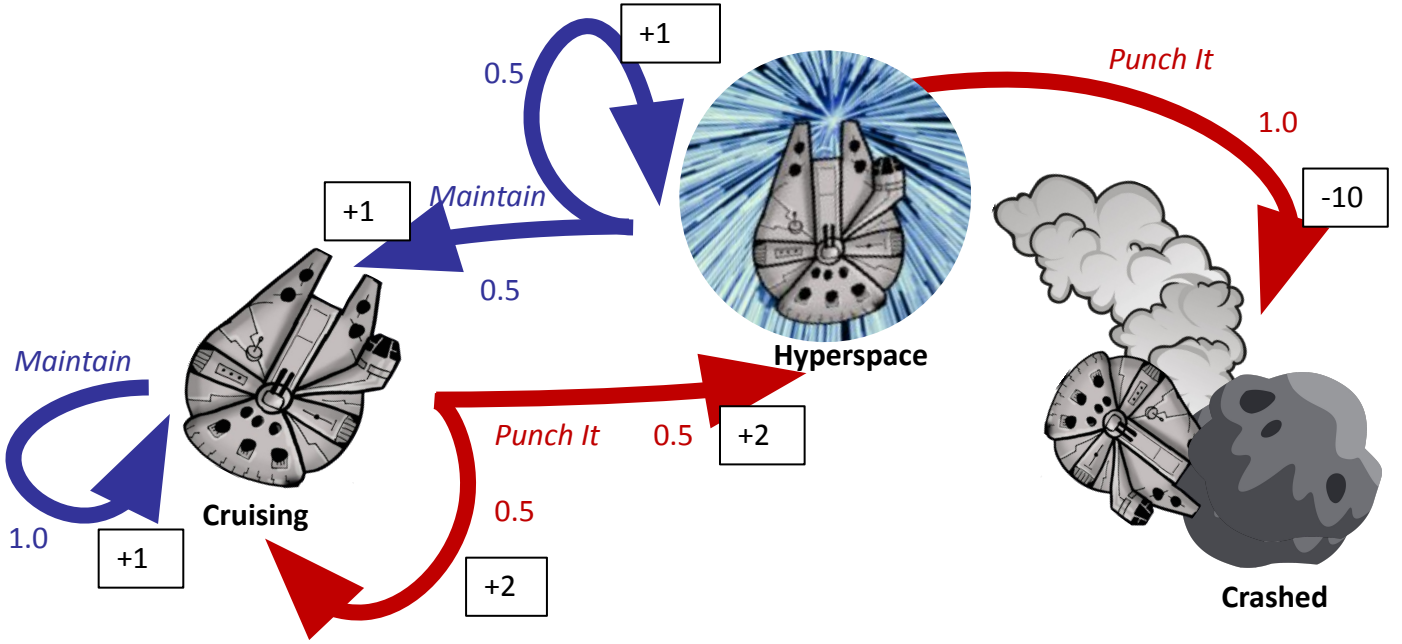


# Example: Value Iteration

$V_2$

$V_1$

$V_0$

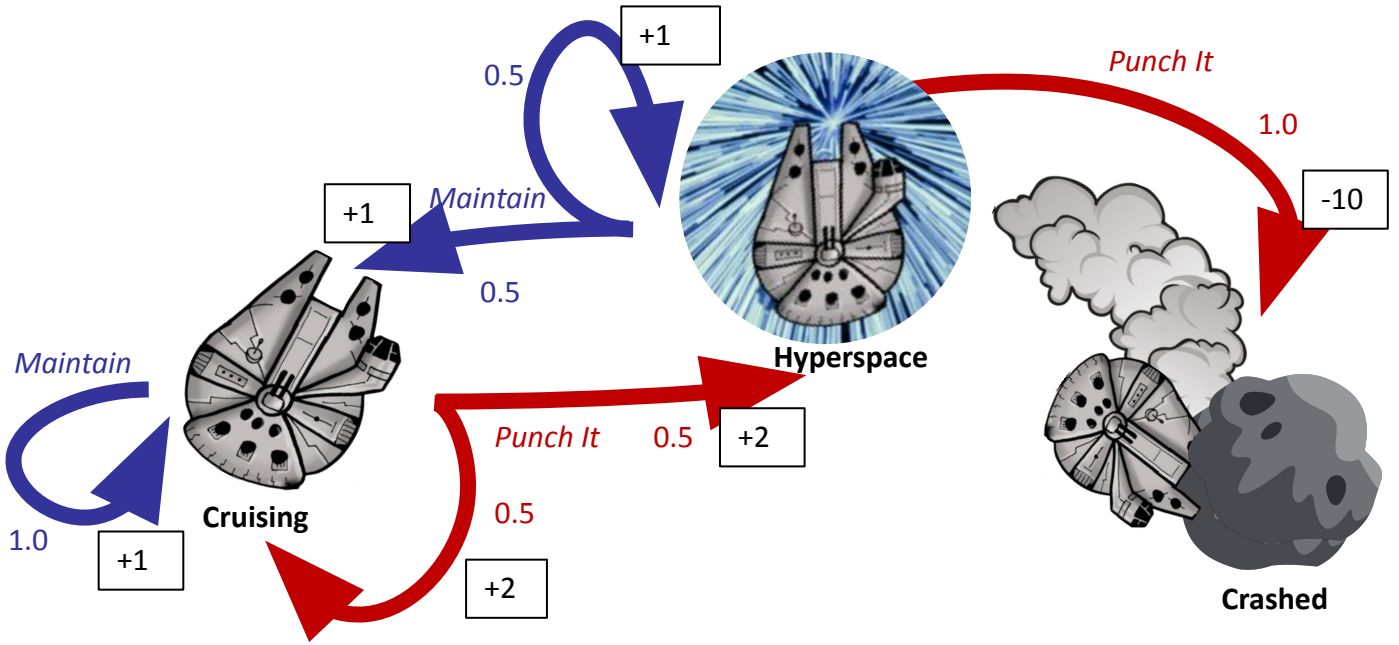



*Assume no discount!*

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

# Example: Value Iteration

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



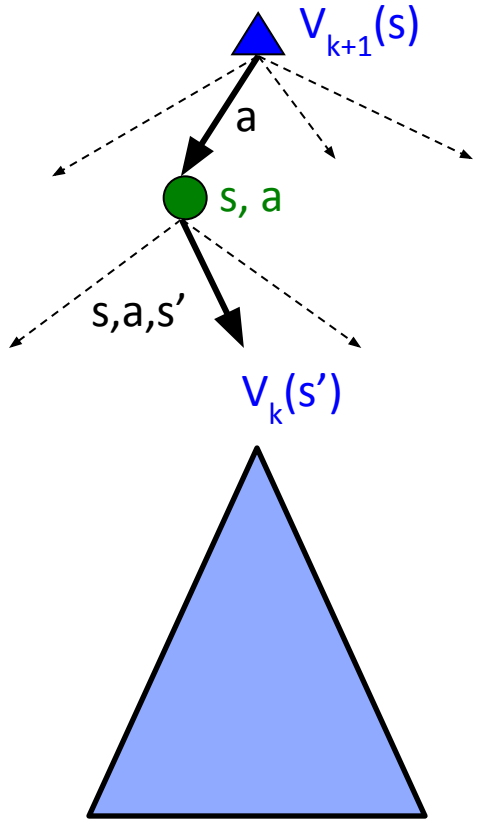

*Assume no discount!*

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

# Value Iteration

- 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:

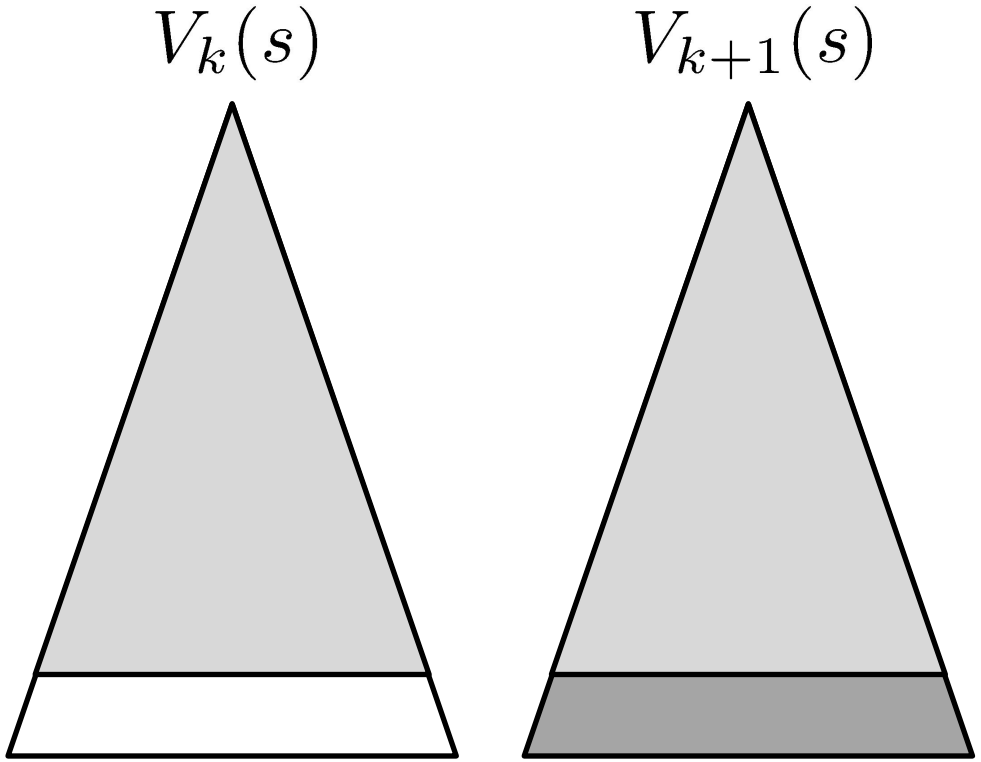
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[ R(s,a,s') + \gamma V_k(s') \right]$$



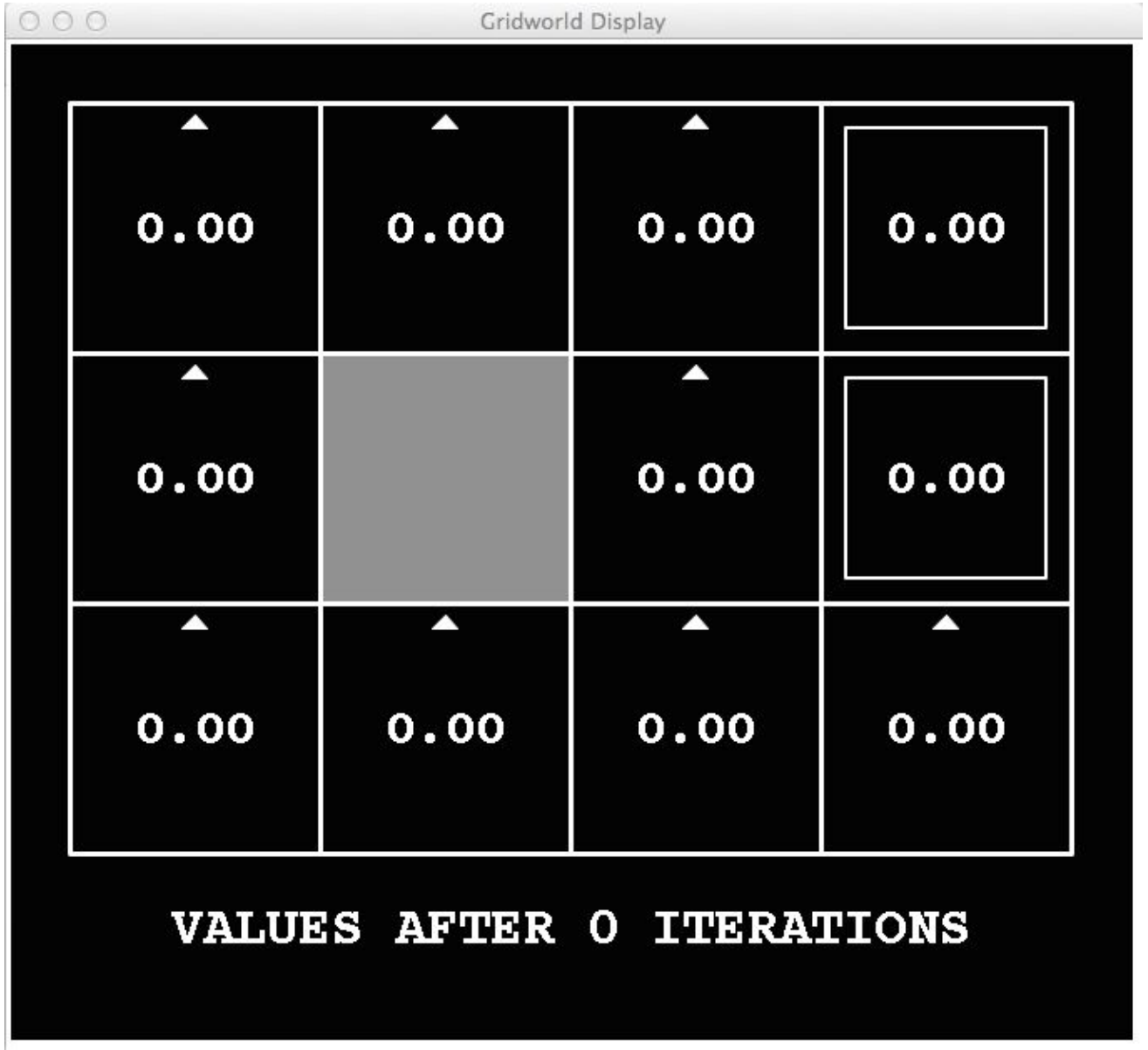

- Repeat until convergence
- Complexity of each iteration: $O(S^2A)$
- Theorem: will converge to unique optimal values
  - Basic idea: approximations get refined towards optimal values
  - Policy may converge long before values do

# 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 $\gamma^k$ 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

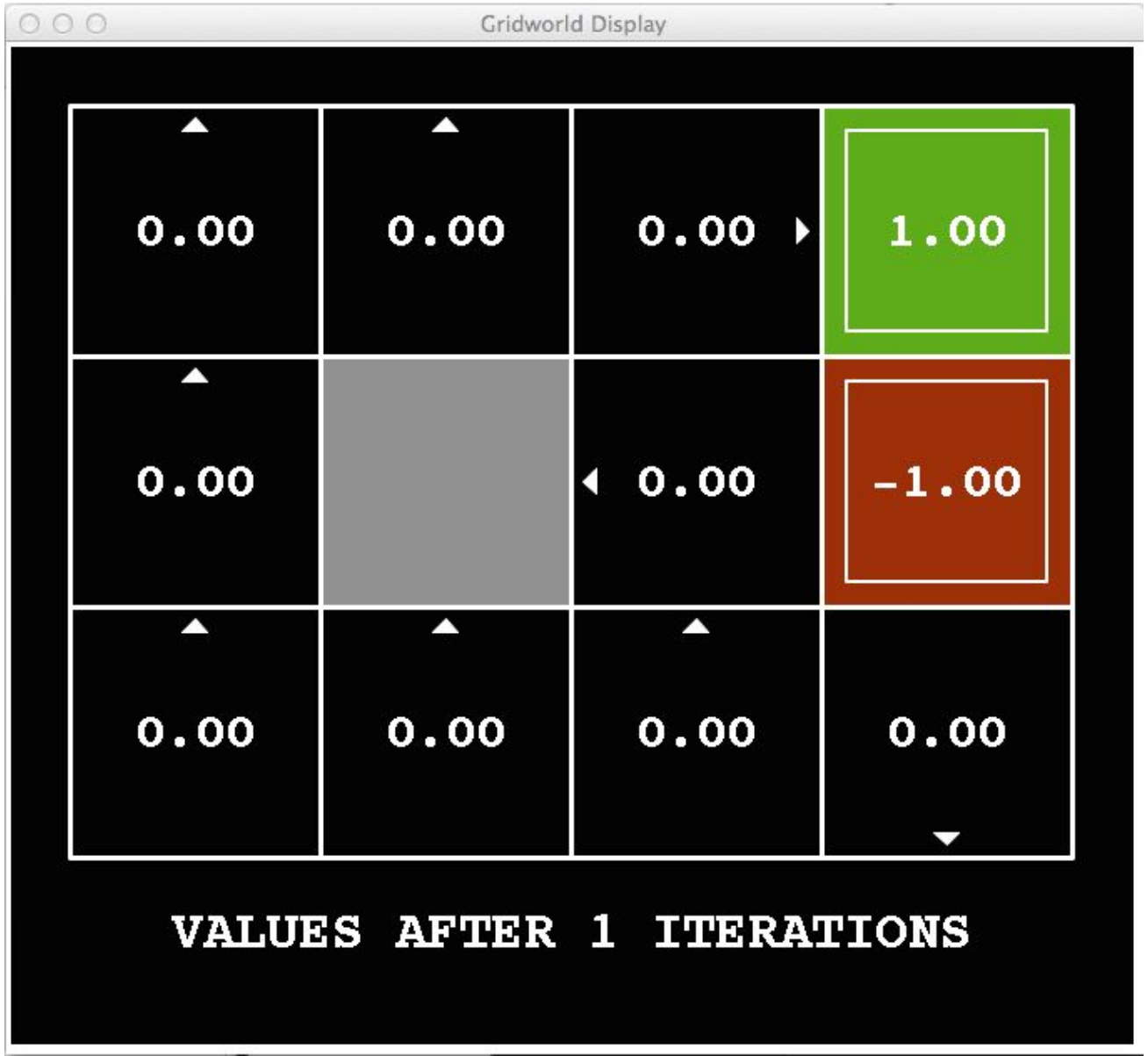


# k=0




Noise = 0.2
Discount = 0.9
Living reward = 0

# k=1




Noise = 0.2
Discount = 0.9
Living reward = 0

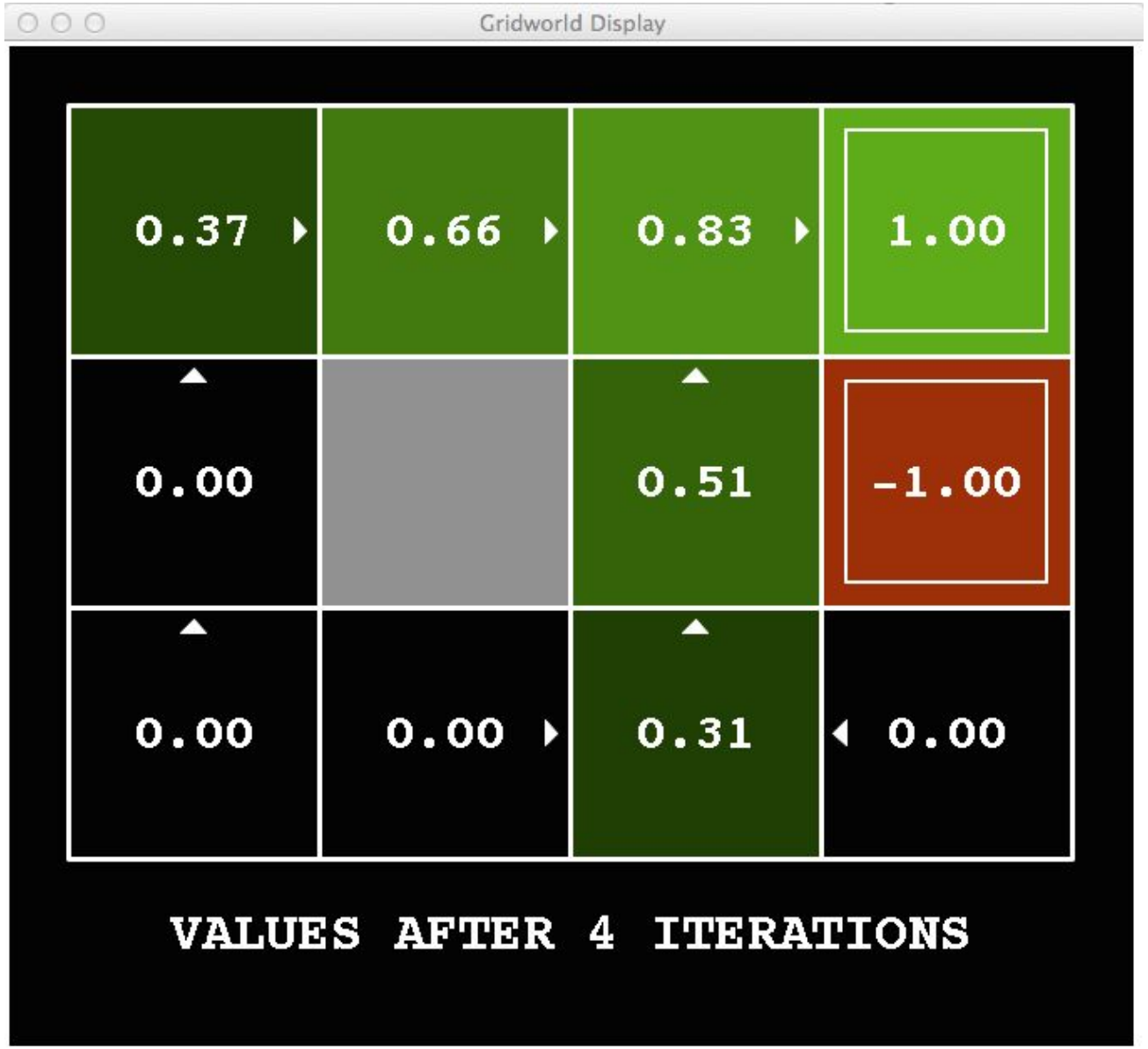
# k=2

Noise = 0.2
Discount = 0.9
Living reward = 0

# k=3

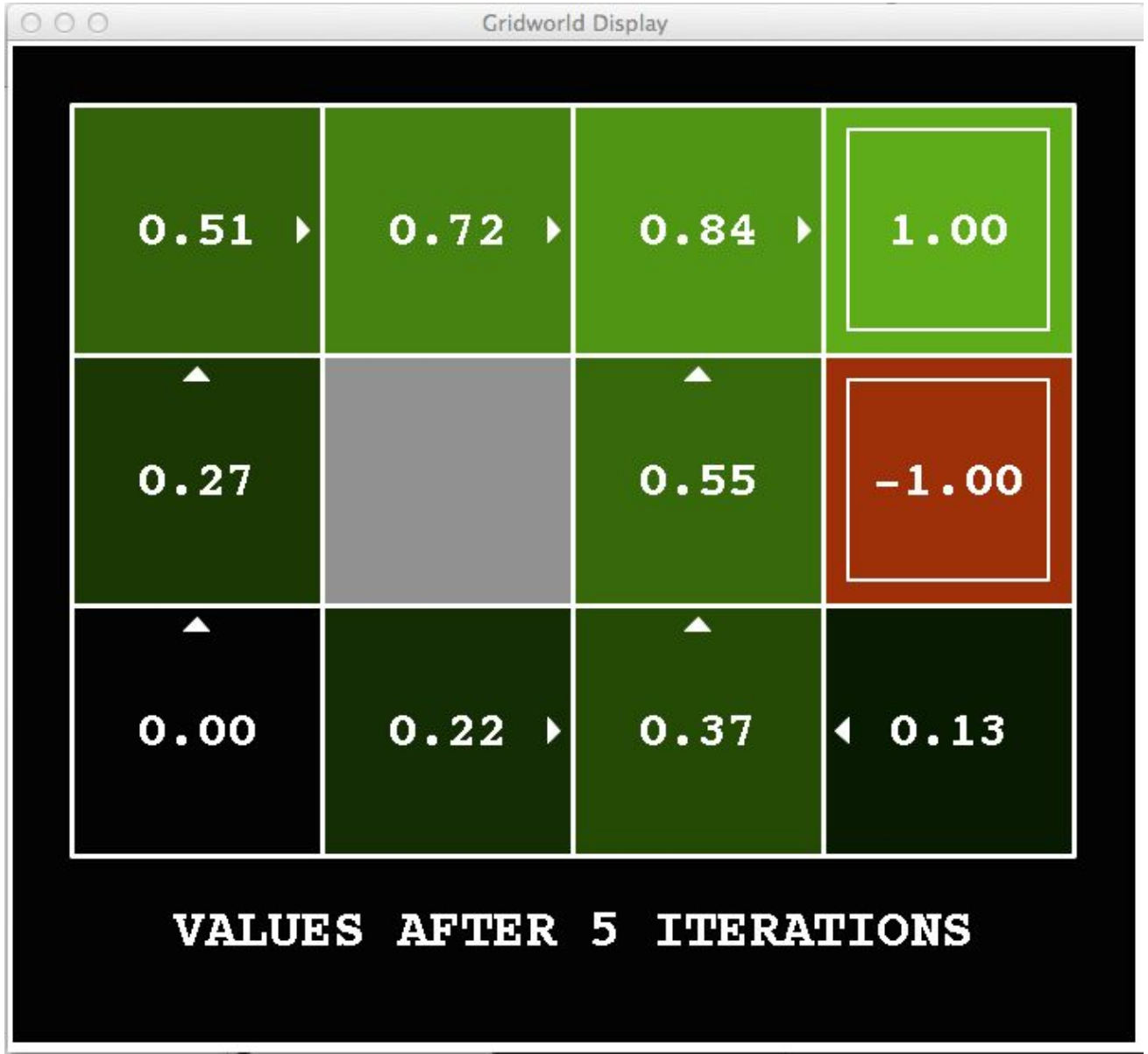
Noise = 0.2
Discount = 0.9
Living reward = 0

# k=4



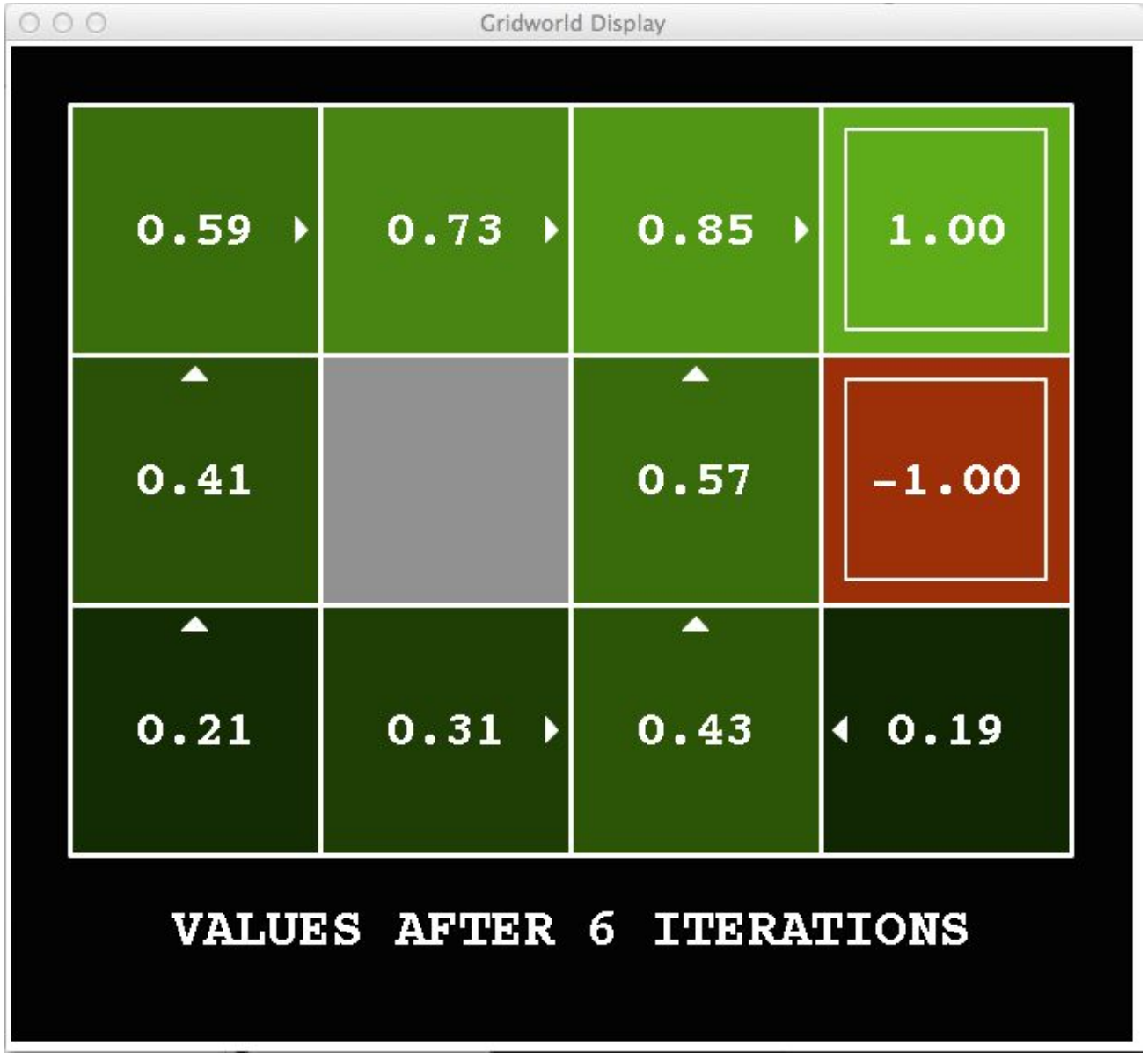

Noise = 0.2
Discount = 0.9
Living reward = 0

# k=5



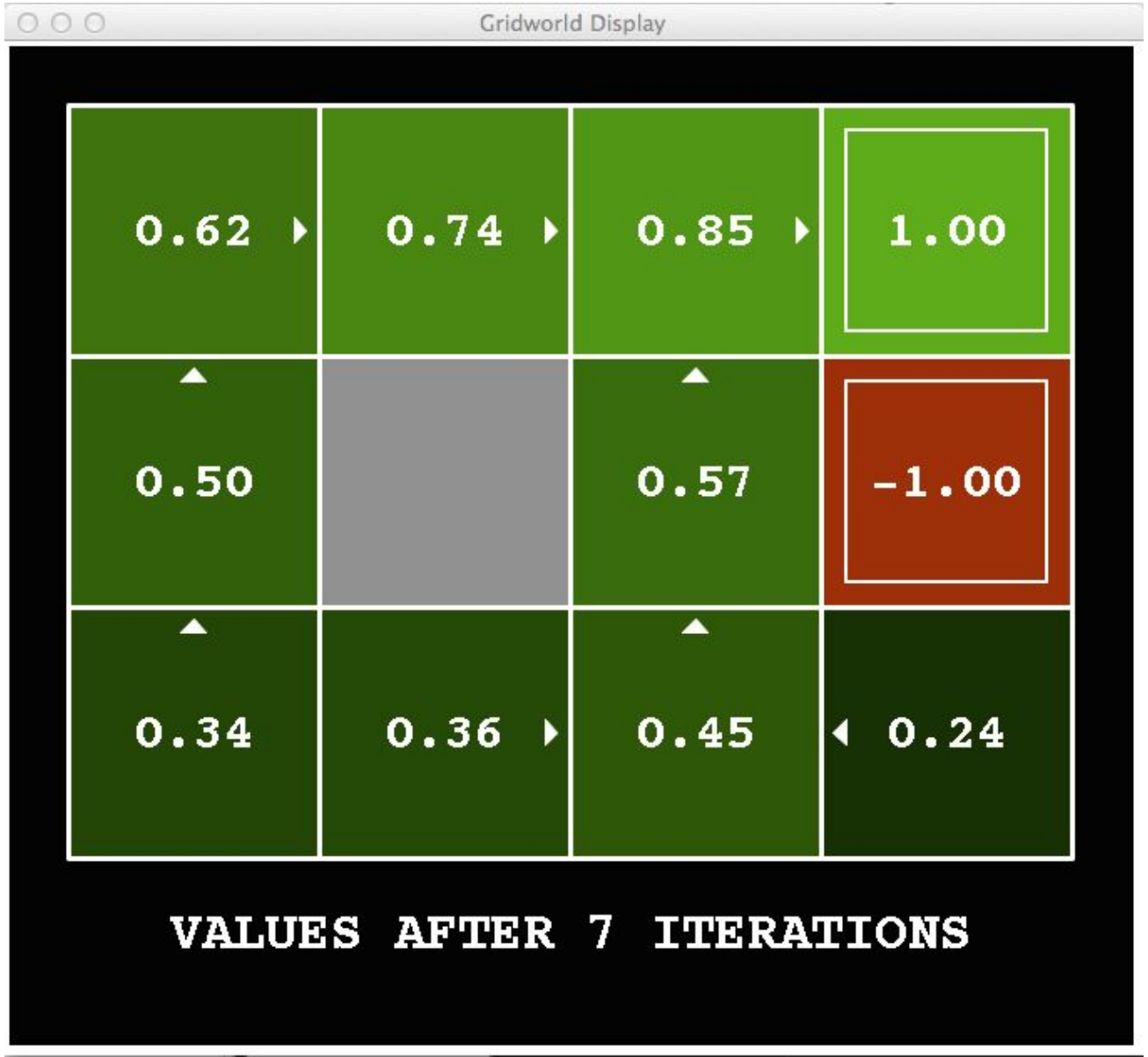

Noise = 0.2
Discount = 0.9
Living reward = 0

# k=6



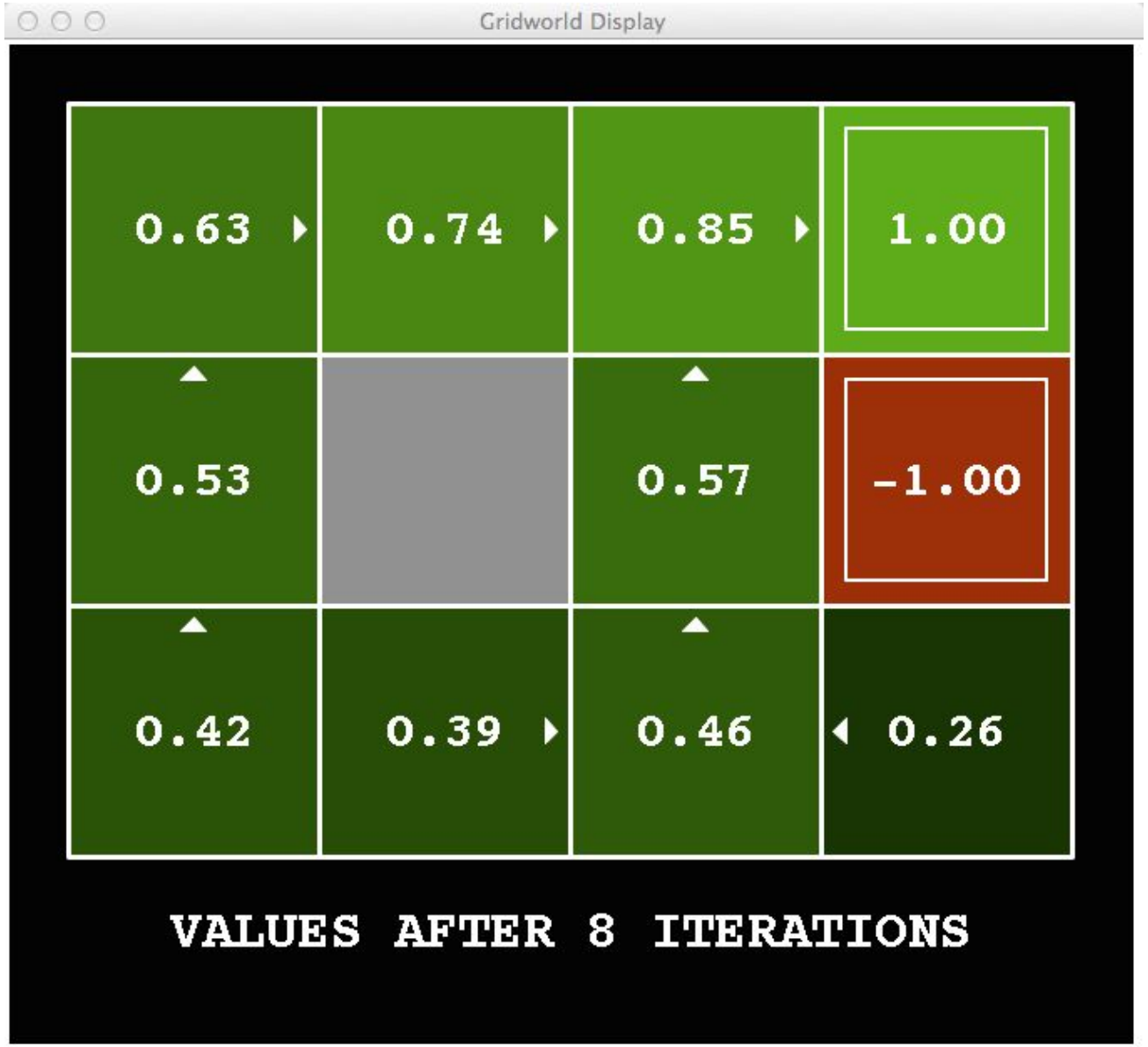

Noise = 0.2
Discount = 0.9
Living reward = 0

# k=7



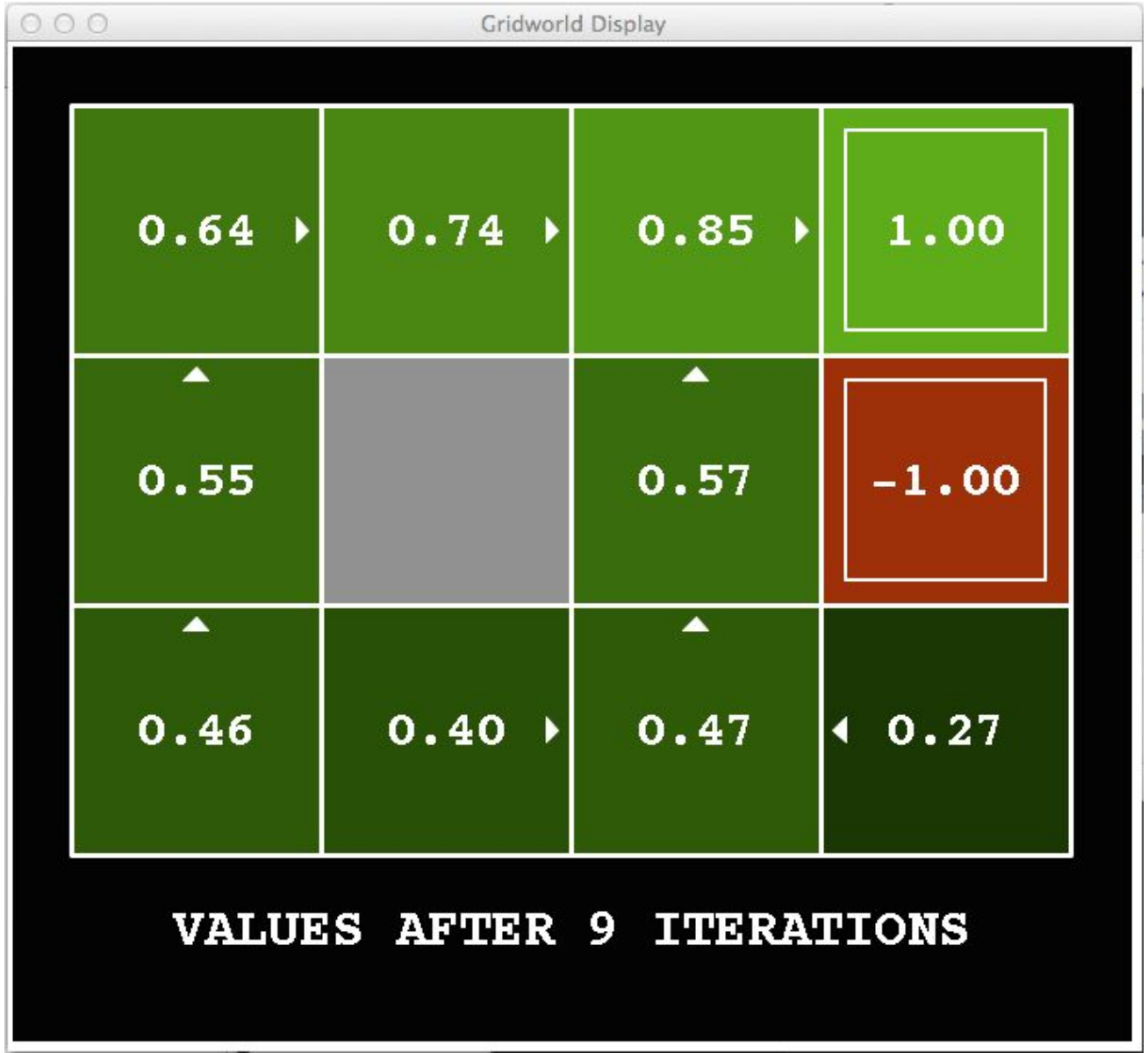

Noise = 0.2
Discount = 0.9
Living reward = 0

# k=8



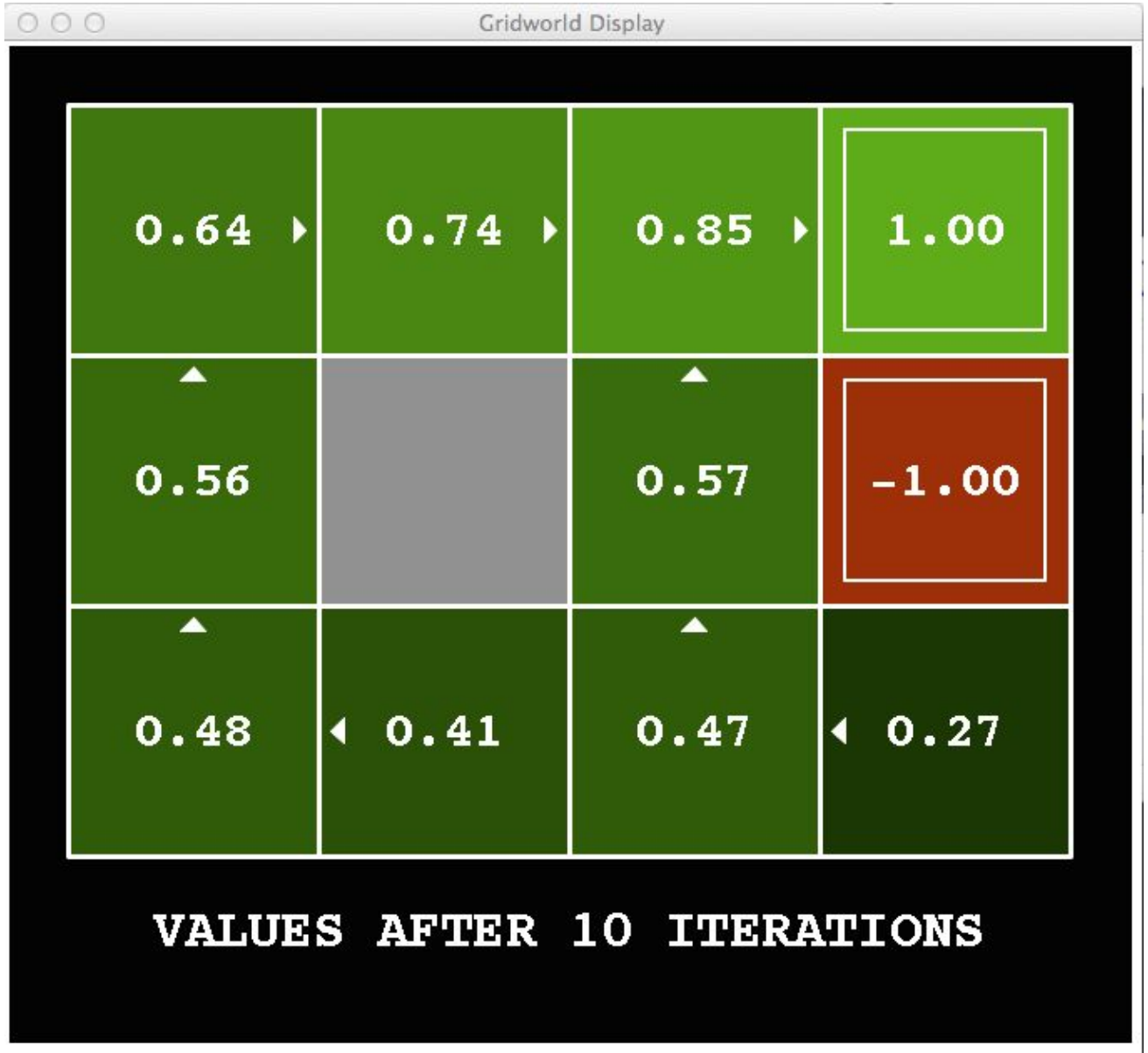

Noise = 0.2
Discount = 0.9
Living reward = 0

# k=9



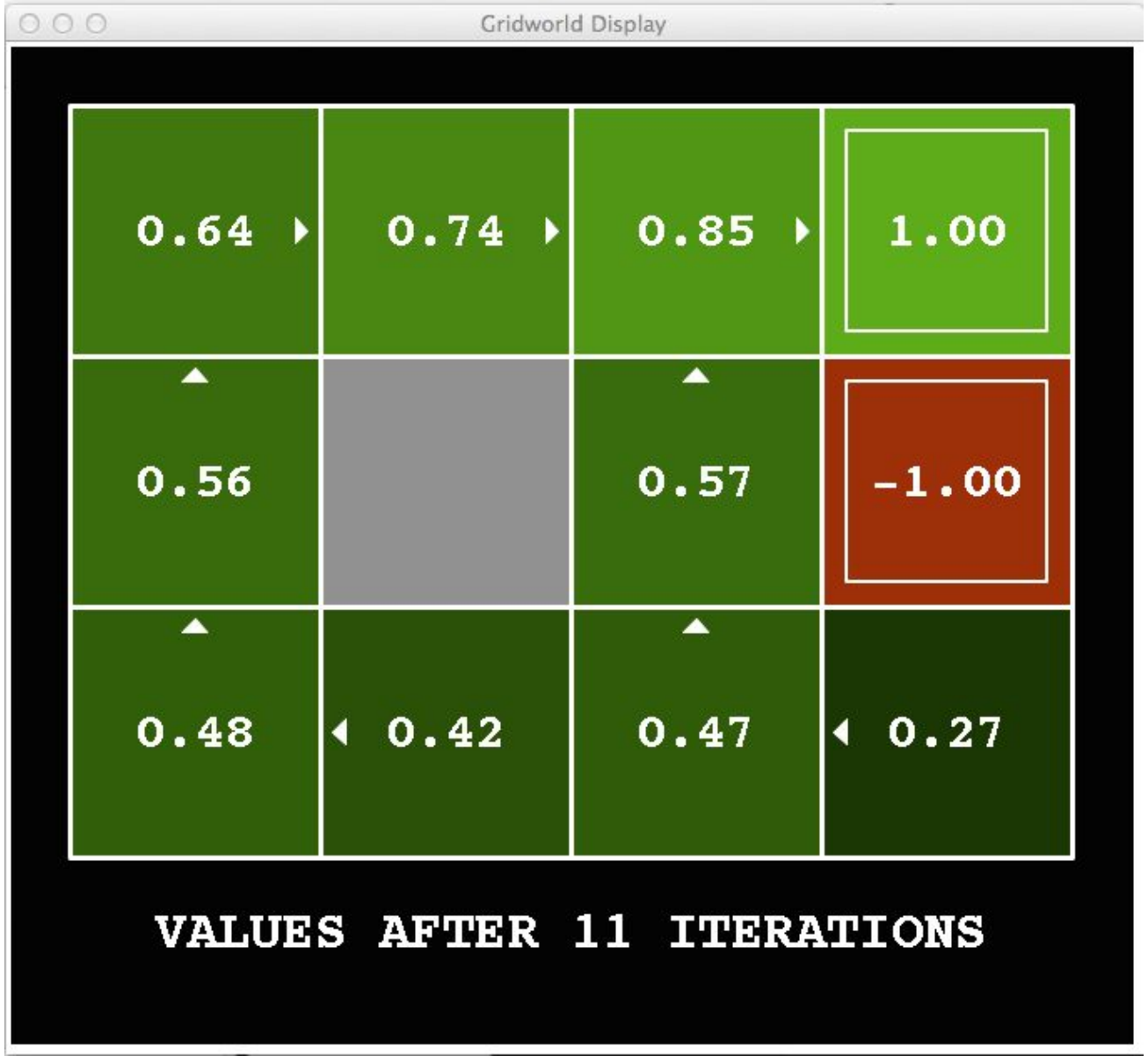

Noise = 0.2
Discount = 0.9
Living reward = 0

# k=10



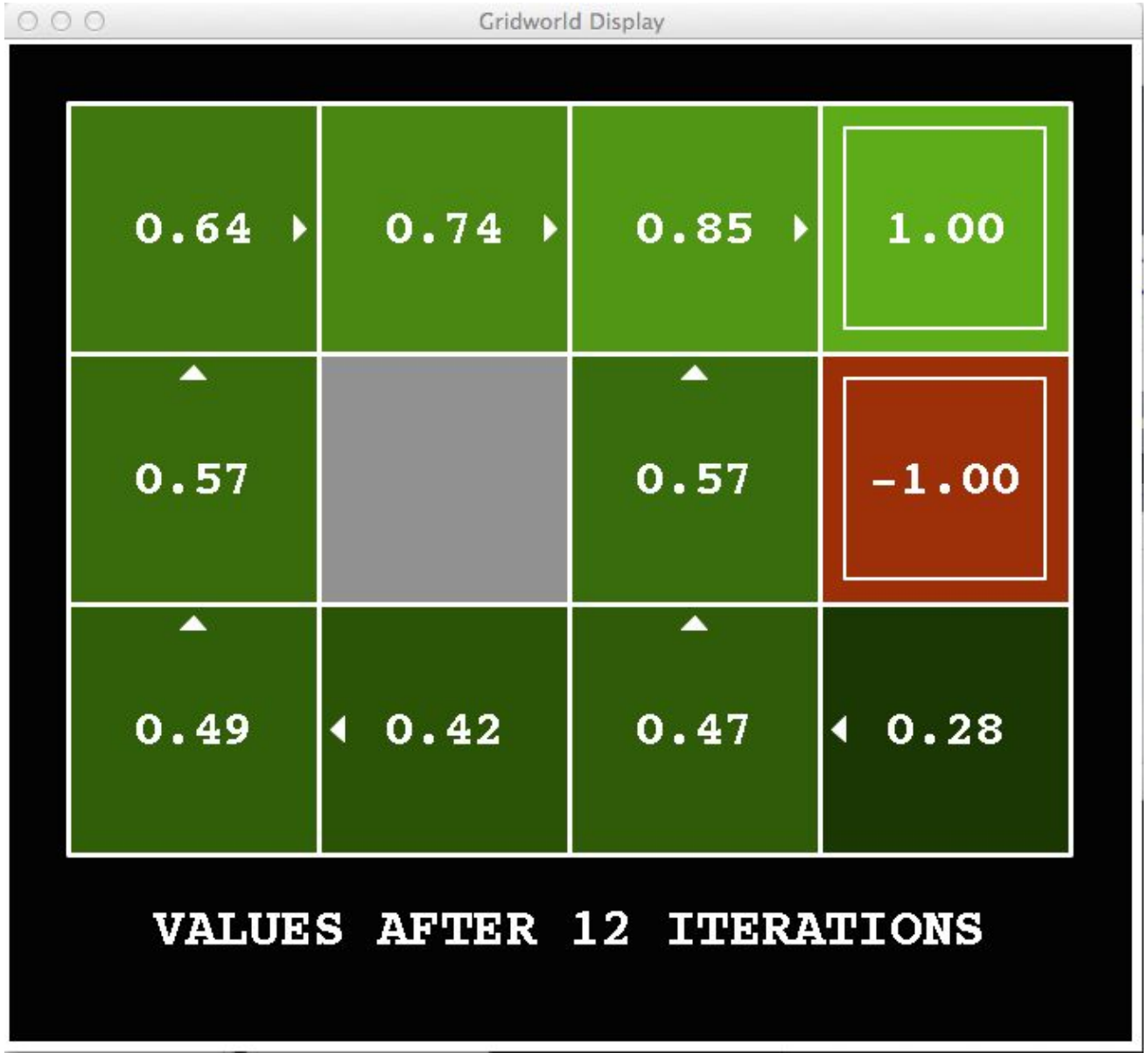

Noise = 0.2
Discount = 0.9
Living reward = 0

# k=11



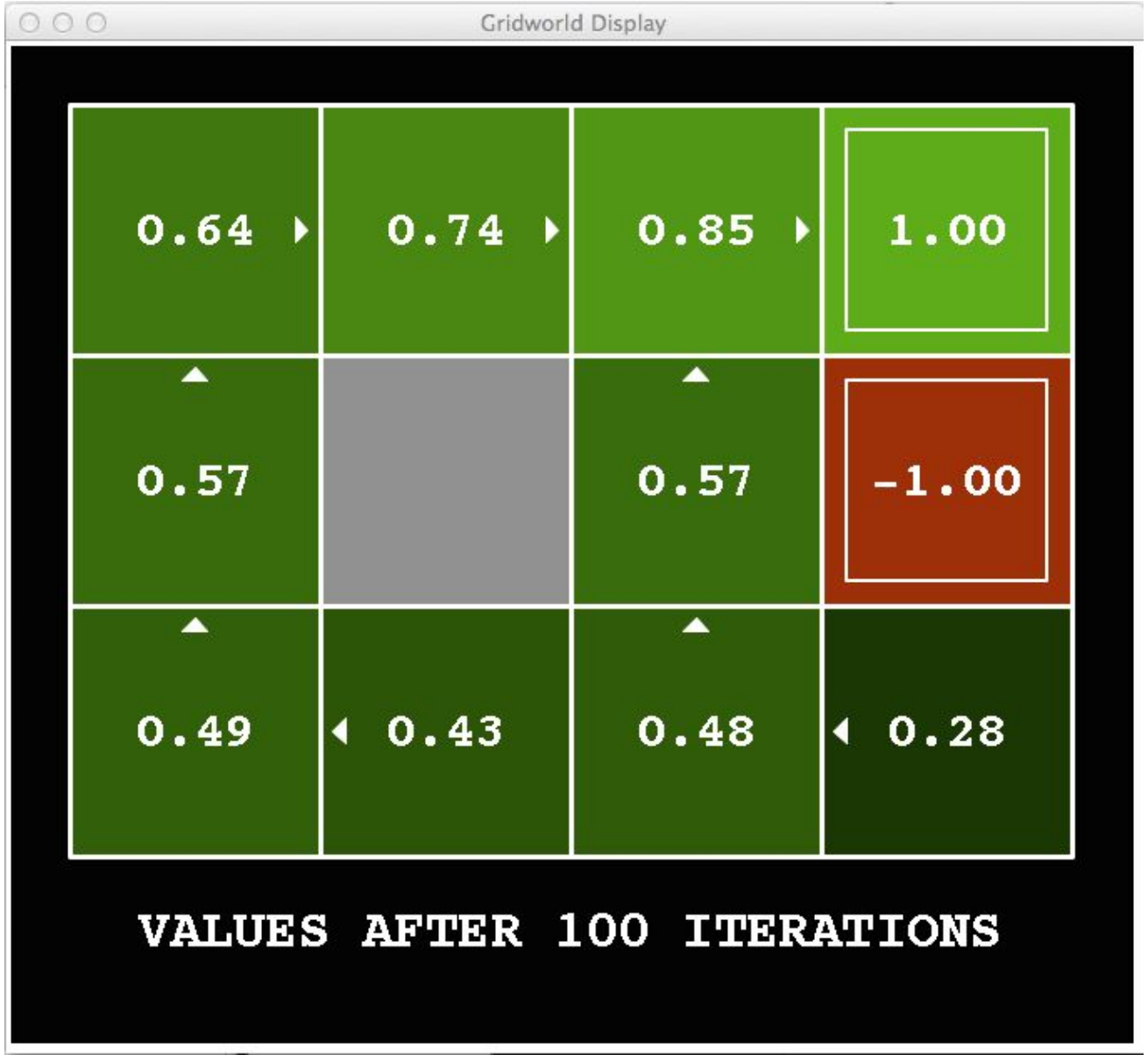

Noise = 0.2
Discount = 0.9
Living reward = 0

# k=12



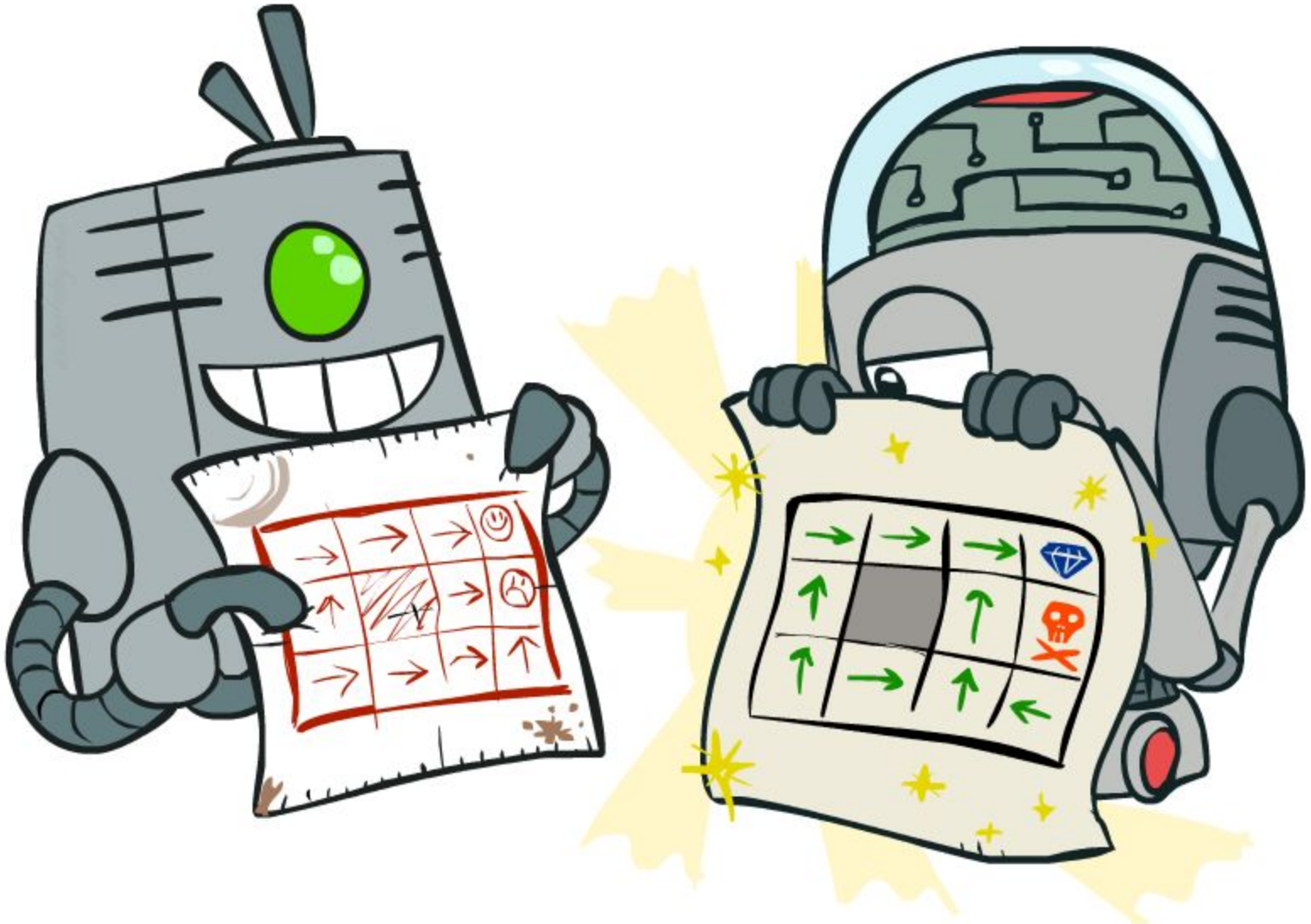

Noise = 0.2
Discount = 0.9
Living reward = 0

# k=100

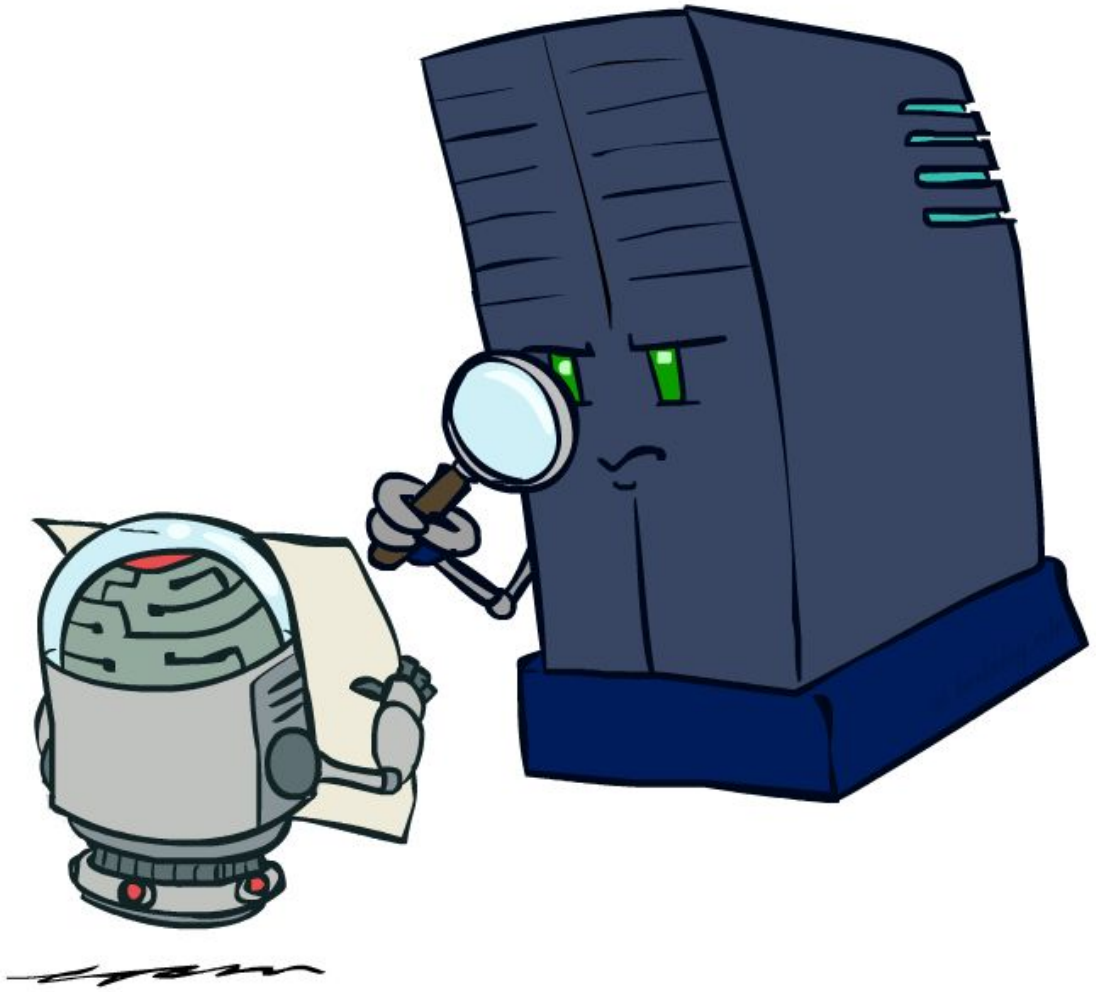



Noise = 0.2
Discount = 0.9
Living reward = 0

# Policy Methods

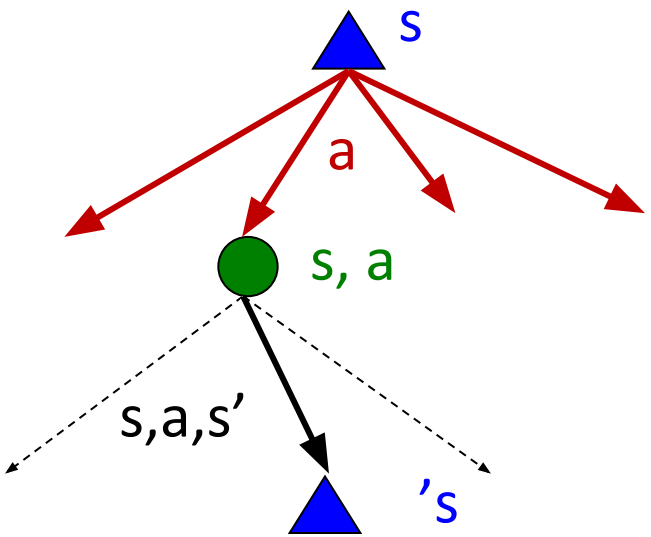


# Policy Evaluation

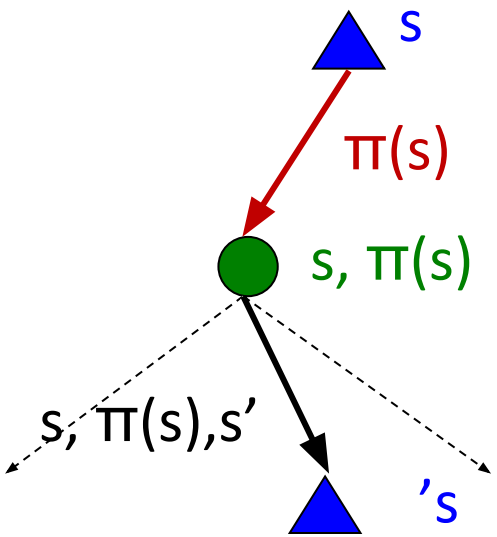


# Fixed Policies

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

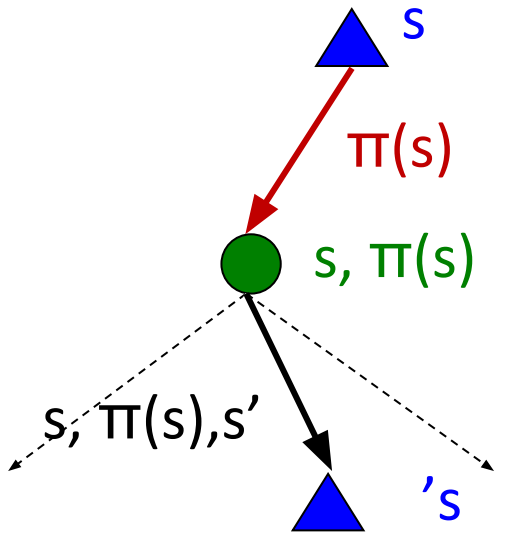
# Utilities for a Fixed Policy

- 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

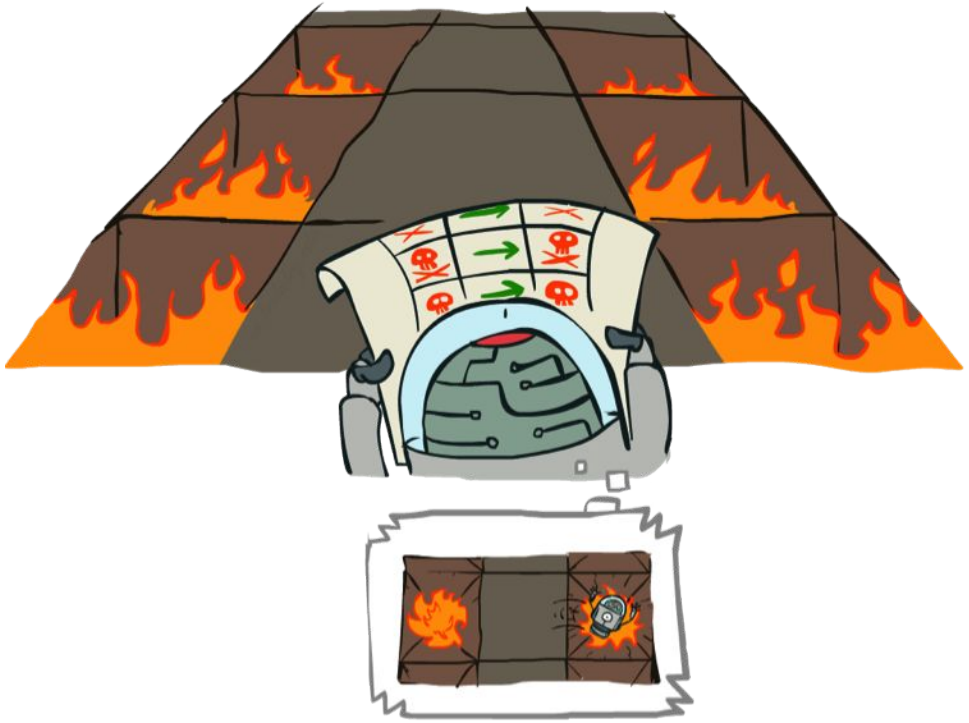
- Recursive relation (one-step look-ahead / Bellman equation):



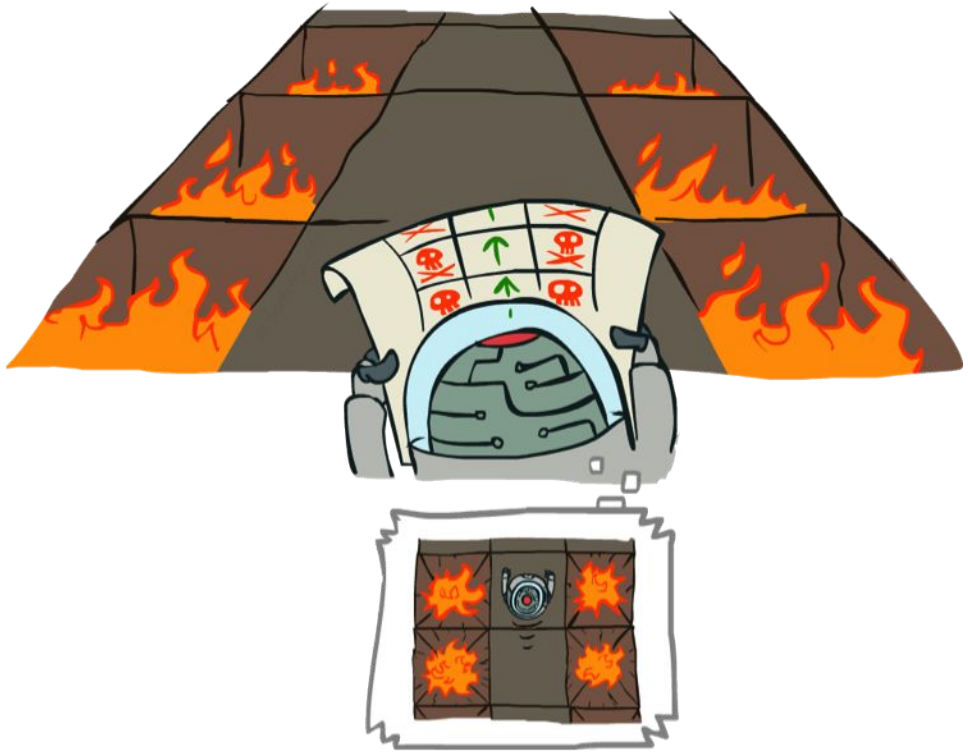

$$V^{\pi}(s) = \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^{\pi}(s')]$$

# Example: Policy Evaluation

Always Go Right



Always Go Forward



# Example: Policy Evaluation

Always Go Right

Always Go Forward

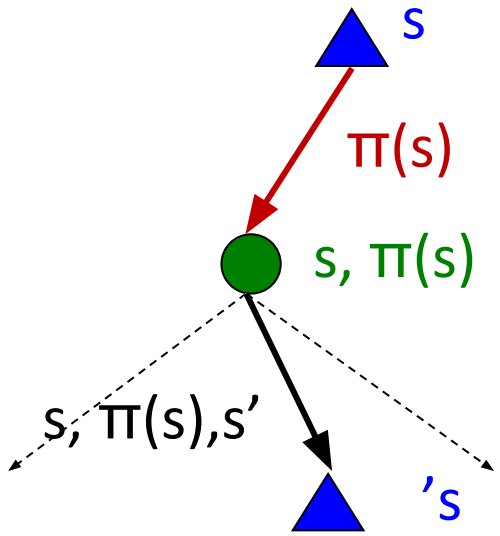
# Policy Evaluation

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

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

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

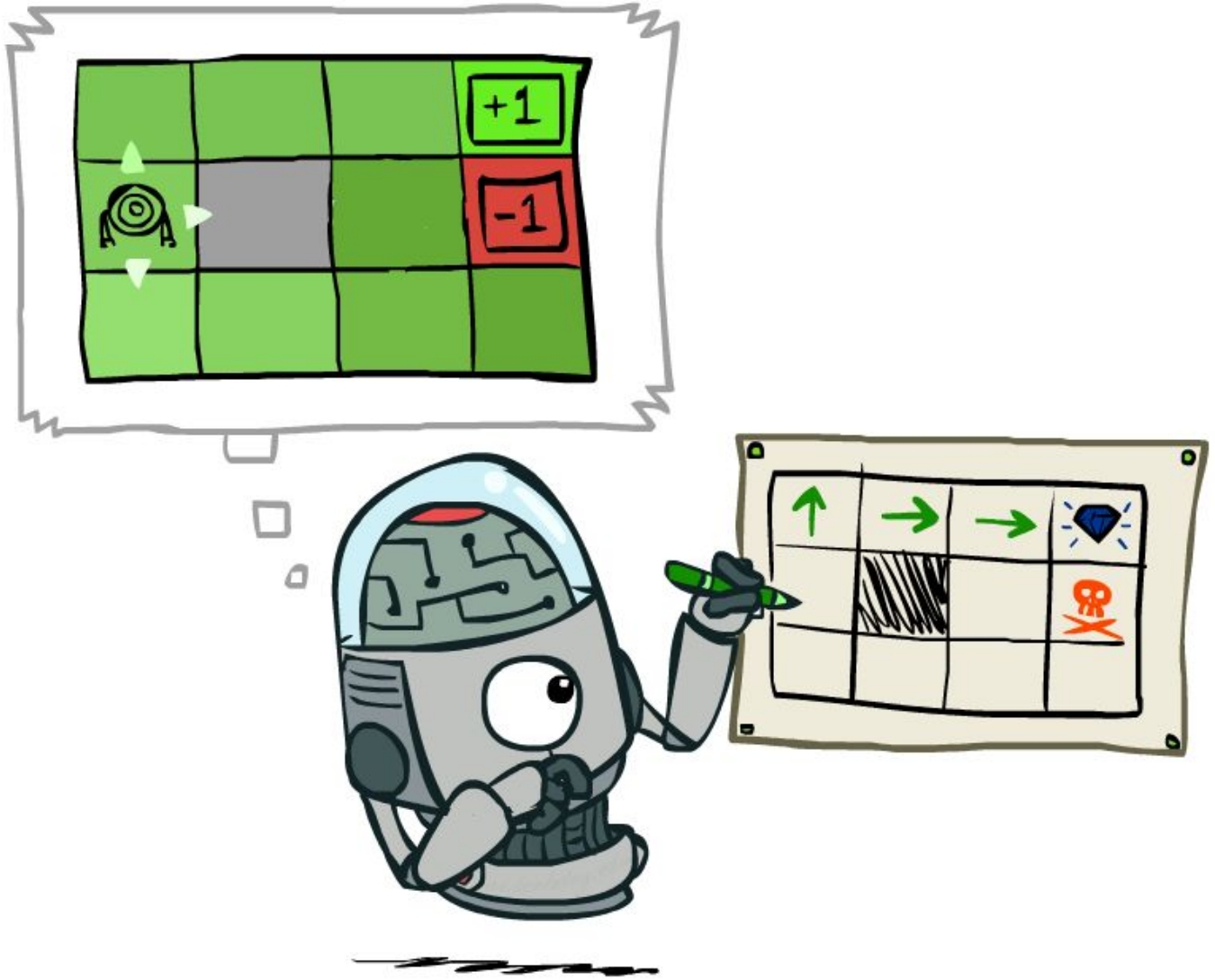
$$V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V_k^\pi(s')]$$




- Efficiency: $O(S^2)$ per iteration

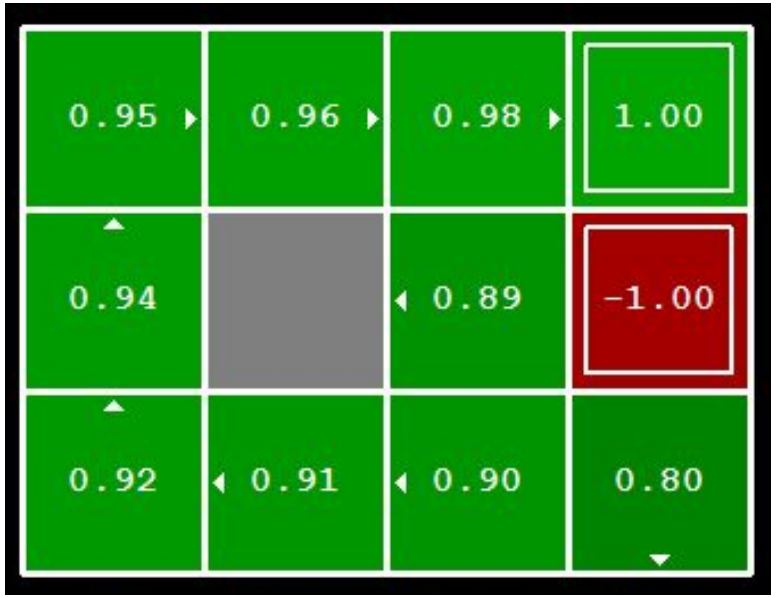
- Idea 2: Without the maxes, the Bellman equations are just a linear system
  - Solve with Matlab (or your favorite linear system solver)

# Policy Extraction



# 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)

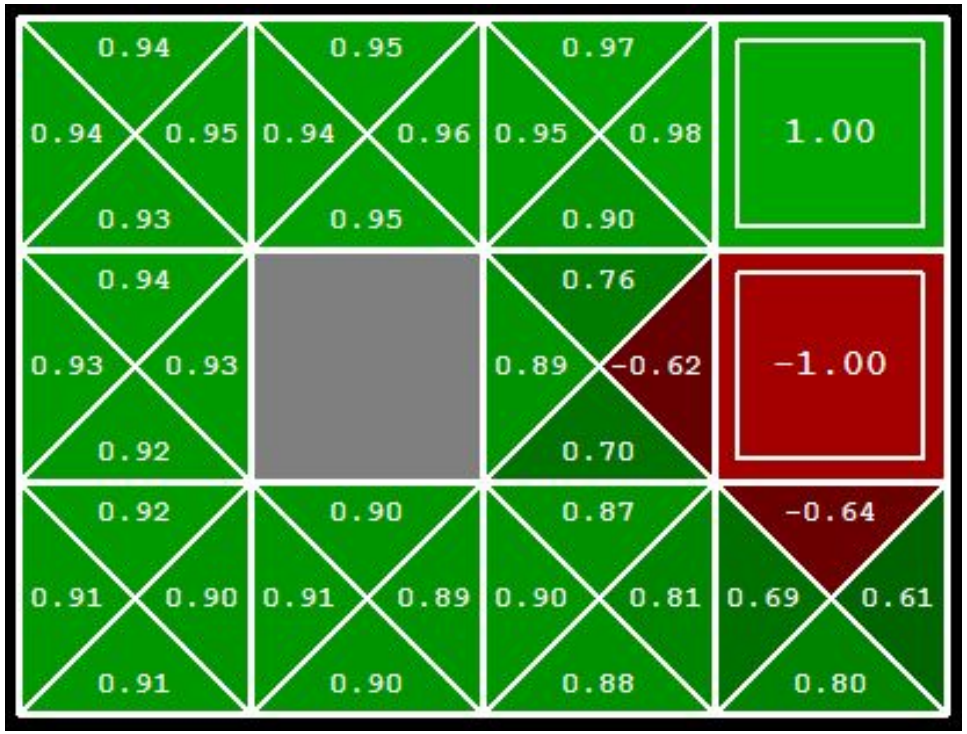



$$\pi^*(s) = \arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

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

# Computing Actions from Q-Values

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

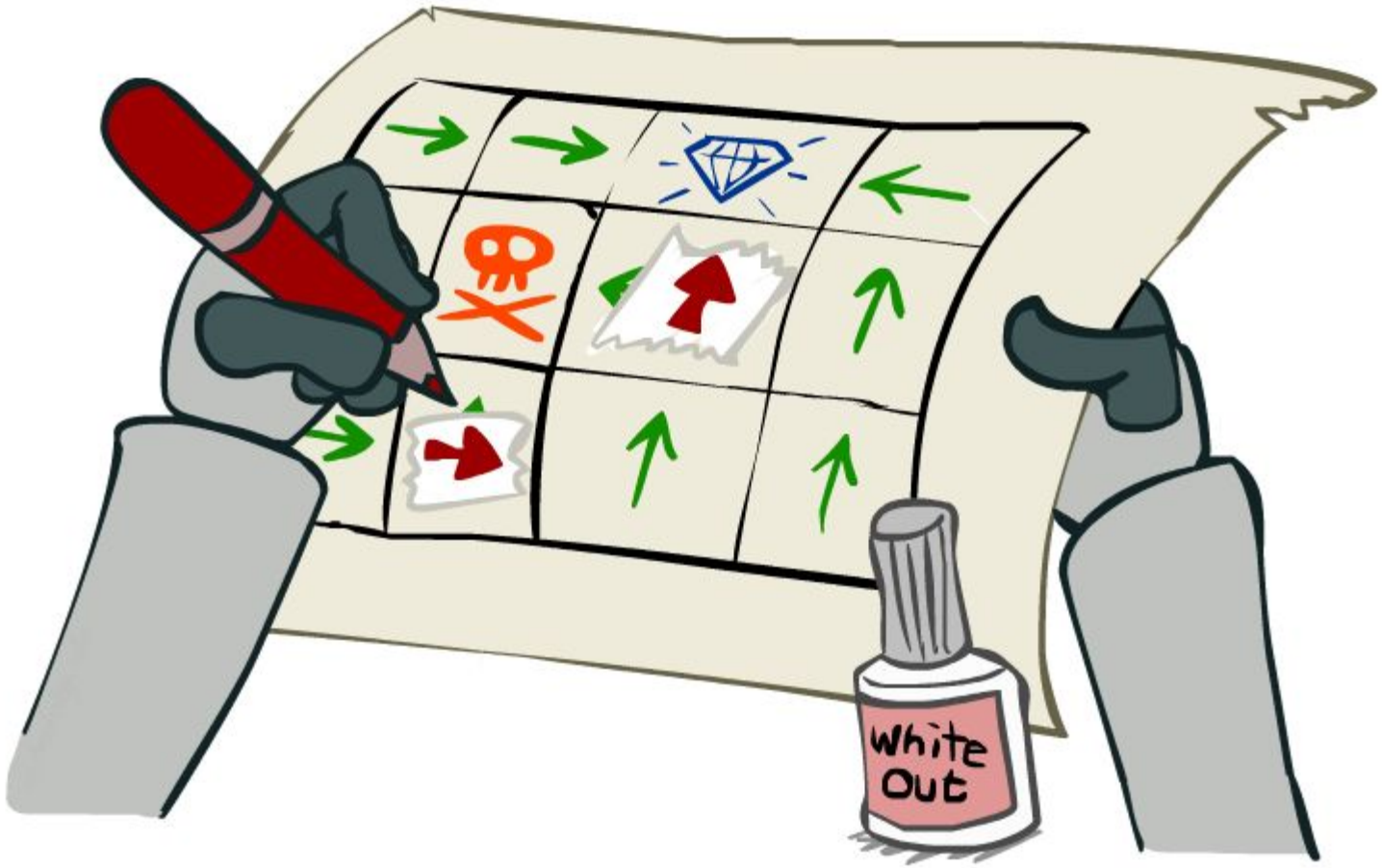



- How should we act?
  - Completely trivial to decide!

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

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

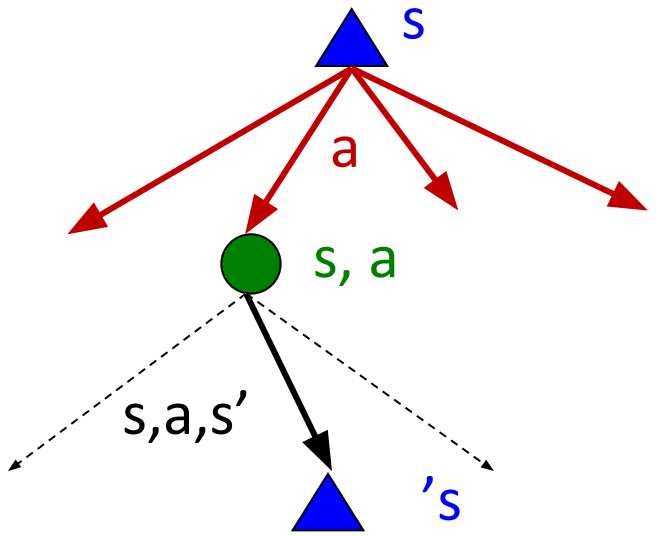
# Policy Iteration



# Problems with Value Iteration

- Value iteration repeats the Bellman updates:

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




- Problem 1: It's slow – $O(S^2A)$ per iteration
- Problem 2: The "max" at each state rarely changes
- Problem 3: The policy often converges long before the values

# Policy Iteration

- Alternative approach for optimal values:
  - Step 1: Policy evaluation: calculate utilities for some fixed policy (not optimal utilities!) until convergence
  - Step 2: Policy improvement: update policy using one-step look-ahead with resulting converged (but not optimal!) utilities as future values
  - Repeat steps until policy converges

- This is policy iteration
  - It's still optimal!
  - Can converge (much) faster under some conditions

# Policy Iteration

- Step 1 (Policy Evaluation): For fixed current policy π, find values with policy evaluation:
  - Iterate until values converge:

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

- Step 2 (Policy Improvement): For fixed values, get a better policy using policy extraction
  - One-step look-ahead:

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

# Comparison

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

# Summary: MDP Algorithms

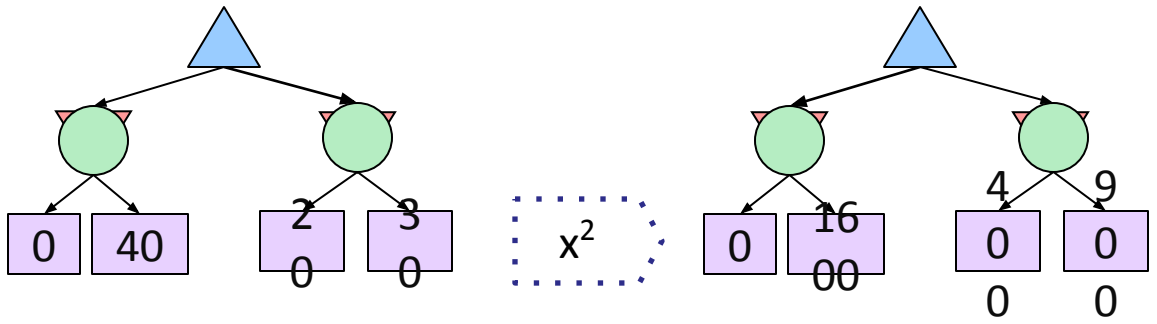
- So you want to….
  - Compute optimal values: use value iteration or policy iteration
  - Compute values for a particular policy: use policy evaluation
  - Turn your values into a policy: use policy extraction (one-step lookahead)

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

# Maximum Expected Utility

- Why should we average utilities? Why not minimax?
- Principle of maximum expected utility:
  - A rational agent should choose the action that maximizes its expected utility, given its knowledge
- Questions:
  - Where do utilities come from?
  - How do we know such utilities even exist?
  - How do we know that averaging even makes sense?
  - What if our behavior (preferences) can't be described by utilities?

# What Utilities to Use?

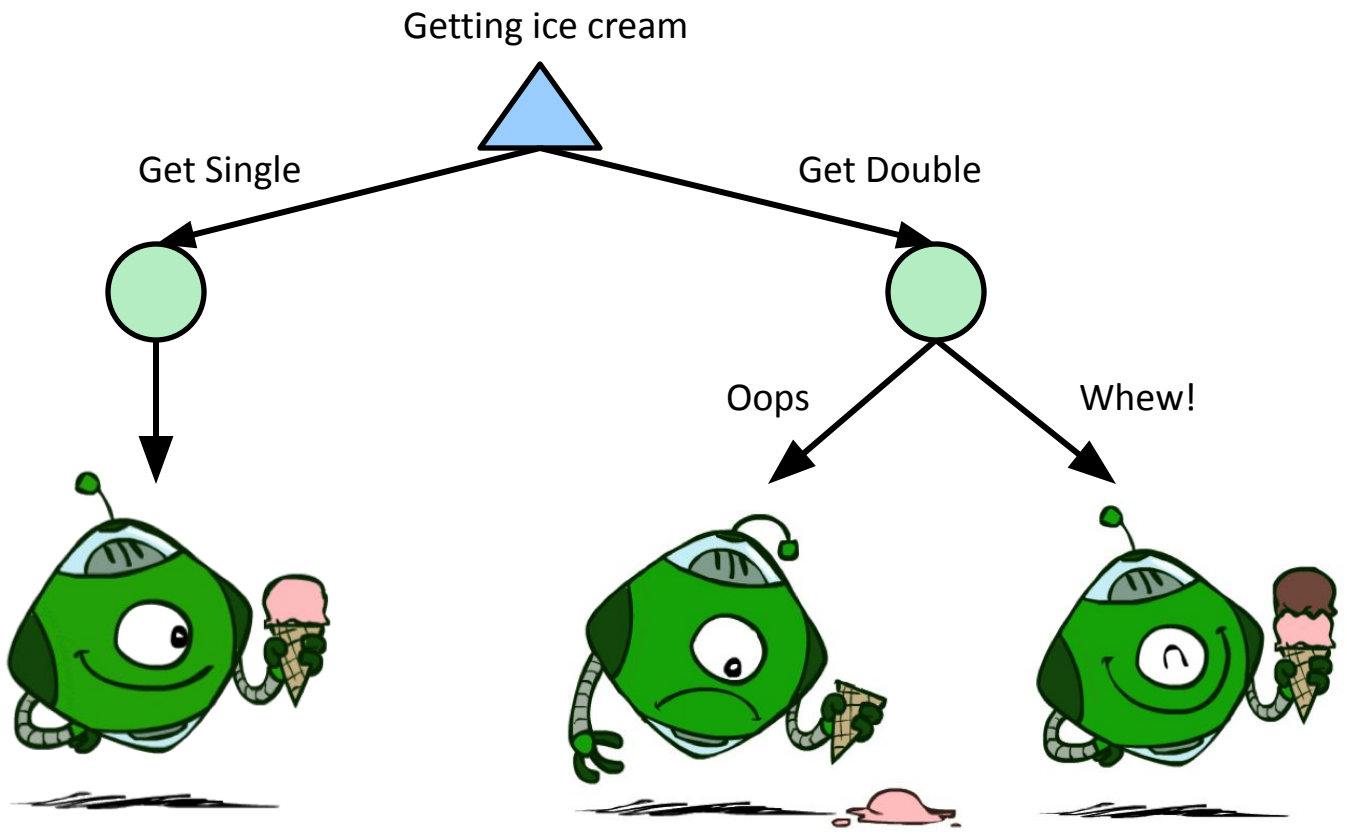



- For worst-case minimax reasoning, terminal function scale doesn't matter
  - We just want better states to have higher evaluations (get the ordering right)
  - We call this insensitivity to monotonic transformations
- For average-case expectimax reasoning, we need *magnitudes* to be meaningful

# Utilities

- Utilities are functions from outcomes (states of the world) to real numbers that describe an agent's preferences
- Where do utilities come from?
  - In a game, may be simple (+1/-1)
  - Utilities summarize the agent's goals
  - Theorem: any "rational" preferences can be summarized as a utility function
- We hard-wire utilities and let behaviors emerge
  - Why don't we let agents pick utilities?
  - Why don't we prescribe behaviors?

# Utilities: Uncertain Outcomes



# Preferences

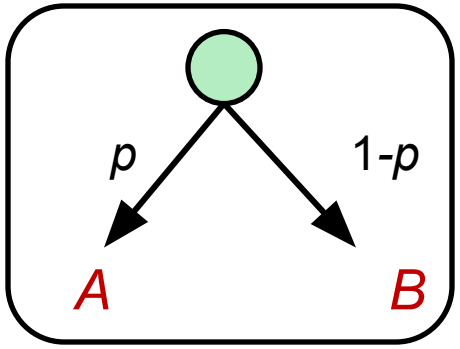
- An agent must have preferences among:
  - Prizes: $A$, $B$, etc.
  - Lotteries: situations with uncertain prizes

$$L = [p, A;\ \ (1-p), B]$$

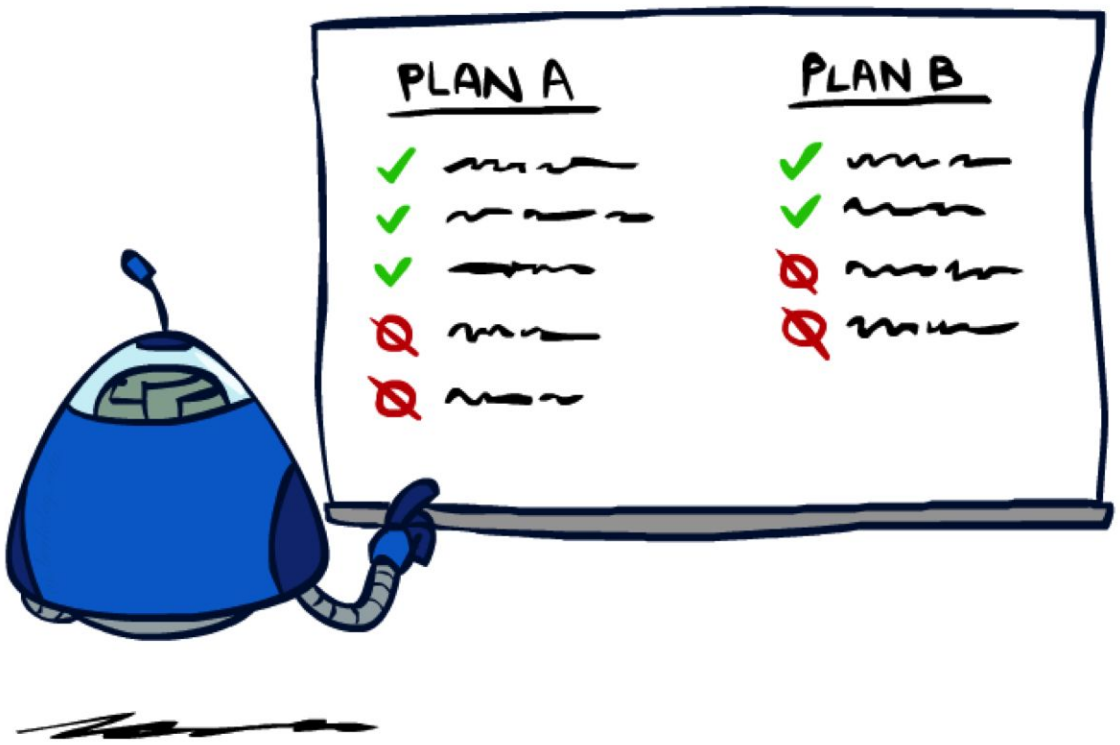
- Notation:
  - Preference: $A \succ B$
  - Indifference: $A \sim B$

A Prize

A Lottery



# Rationality

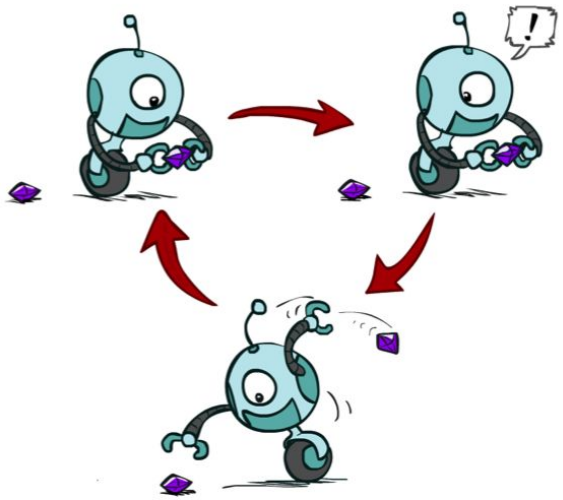


# Rational Preferences

- We want some constraints on preferences before we call them rational, such as:

Axiom of Transitivity: $(A \succ B) \wedge (B \succ C) \Rightarrow (A \succ C)$

- For example: an agent with intransitive preferences can be induced to give away all of its money
  - If B > C, then an agent with C would pay (say) 1 cent to get B
  - If A > B, then an agent with B would pay (say) 1 cent to get A
  - If C > A, then an agent with A would pay (say) 1 cent to get C



# Rational Preferences

The Axioms of Rationality

Orderability

$$(A \succ B) \vee (B \succ A) \vee (A \sim B)$$

Transitivity

$$(A \succ B) \wedge (B \succ C) \Rightarrow (A \succ C)$$

Continuity

$$A \succ B \succ C \Rightarrow \exists p\ [p, A;\ 1-p, C] \sim B$$

Substitutability

$$A \sim B \Rightarrow [p, A;\ 1-p, C] \sim [p, B; 1-p, C]$$

Monotonicity

$$A \succ B \Rightarrow$$
$$(p \geq q \Leftrightarrow [p, A;\ 1-p, B] \succeq [q, A;\ 1-q, B])$$

Theorem: Rational preferences imply behavior describable as maximization of expected utility

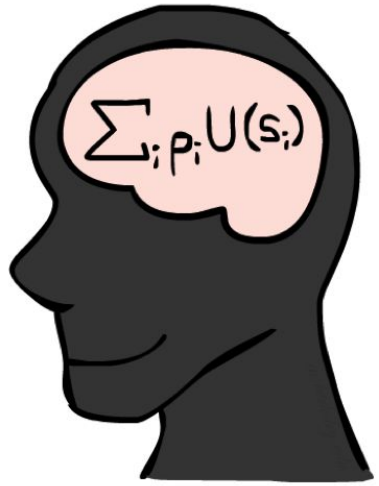
# MEU Principle

- Theorem [Ramsey, 1931; von Neumann & Morgenstern, 1944]
  - Given any preferences satisfying these constraints, there exists a real-valued function U such that:

$$U(A) \geq U(B) \;\Leftrightarrow\; A \succeq B$$

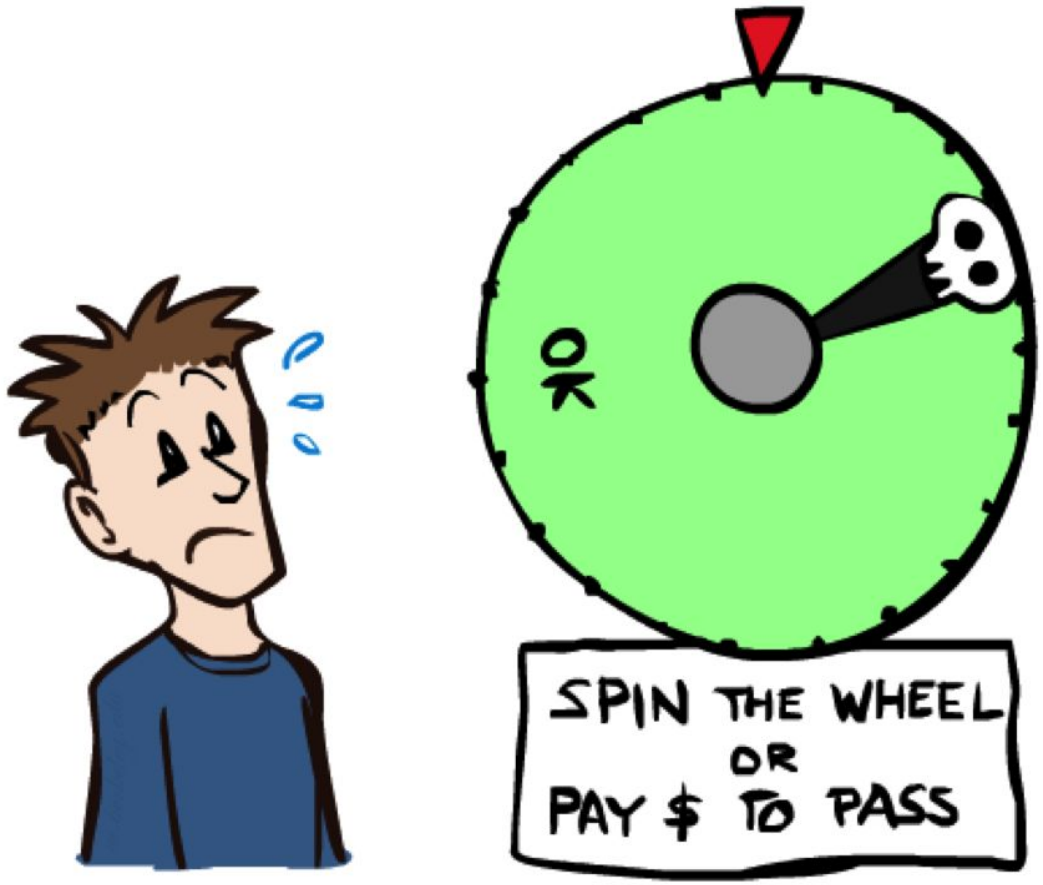
$$U([p_1, S_1;\; \ldots\; ;\; p_n, S_n]) = \sum_i p_i U(S_i)$$

  - I.e. values assigned by U preserve preferences of both prizes and lotteries!




- Maximum expected utility (MEU) principle:
  - Choose the action that maximizes expected utility
  - Note: an agent can be entirely rational (consistent with MEU) without ever representing or manipulating utilities and probabilities
  - E.g., a lookup table for perfect tic-tac-toe, a reflex vacuum cleaner

# Human Utilities



# Utility Scales

- Normalized utilities: $u_+ = 1.0$, $u_- = 0.0$
- Micromorts: one-millionth chance of death, useful for paying to reduce product risks, etc.
- QALYs: quality-adjusted life years, useful for medical decisions involving substantial risk
- Note: behavior is invariant under positive linear transformation

$$U'(x) = k_1 U(x) + k_2 \quad \text{where } k_1 > 0$$

- With deterministic prizes only (no lottery choices), only ordinal utility can be determined, i.e., total order on prizes

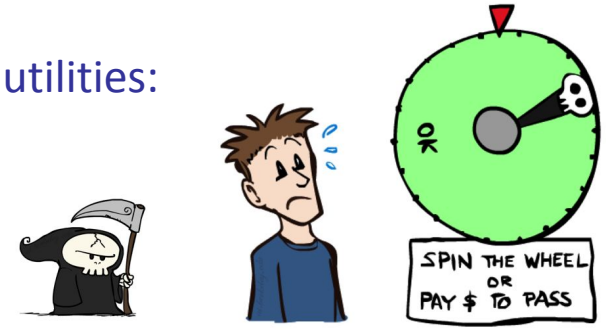
# Micromort examples

| Death from | Micromorts per exposure |
|---|---|
| Scuba diving | 5 per dive |
| Skydiving | 7 per jump |
| Base-jumping | 430 per jump |
| Climbing Mt. Everest | 38,000 per ascent |

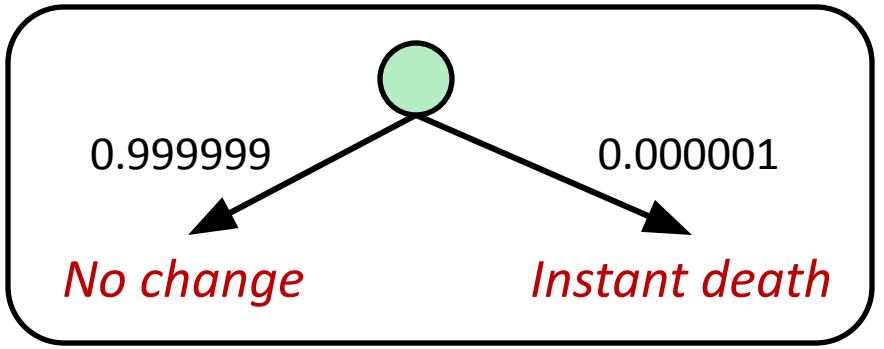
| 1 Micromort | |
|---|---|
| Train travel | 6000 miles |
| Jet | 1000 miles |
| Car | 230 miles |
| Walking | 17 miles |
| Bicycle | 10 miles |
| Motorbike | 6 miles |

# Human Utilities

- Utilities map states to real numbers. Which numbers?
- Standard approach to assessment (elicitation) of human utilities:
  - Compare a prize A to a standard lottery $L_p$ between
    - "best possible prize" $u_+$ with probability p
    - "worst possible catastrophe" $u_-$ with probability 1-p
  - Adjust lottery probability p until indifference: A ~ $L_p$
  - Resulting p is a utility in [0,1]

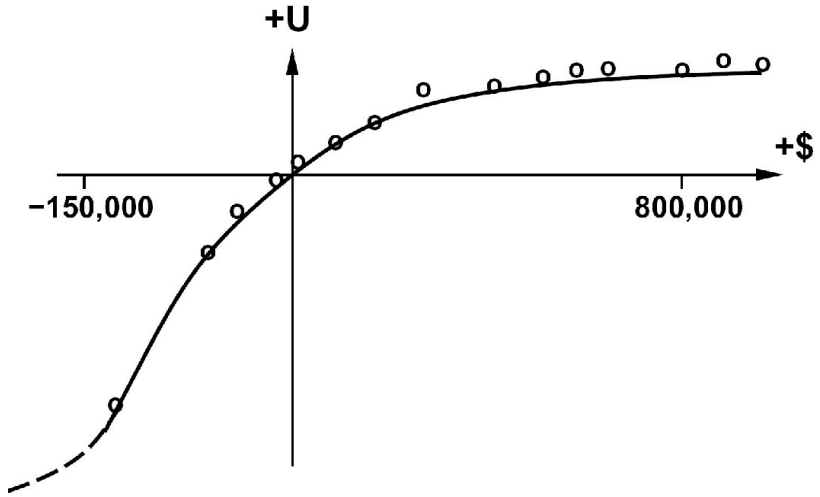



*Pay $30* ~



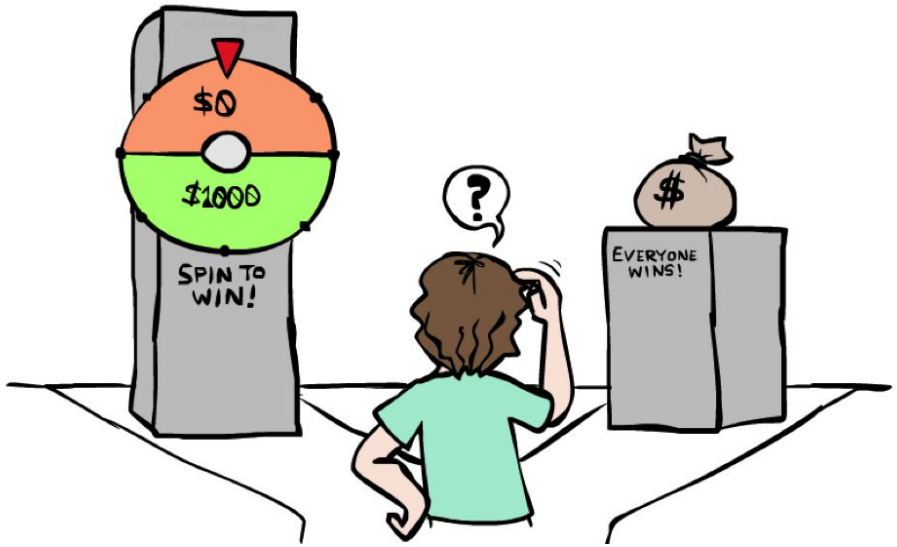
# Money

- Money does not behave as a utility function, but we can talk about the utility of having money (or being in debt)
- Given a lottery L = [p, $X; (1-p), $Y]
  - The expected monetary value EMV(L) is p*X + (1-p)*Y
  - U(L) = p*U($X) + (1-p)*U($Y)
  - Typically, U(L) < U( EMV(L) )
  - In this sense, people are risk-averse
  - When deep in debt, people are risk-prone



# Example: Insurance

- Consider the lottery [0.5, $1000;  0.5, $0]
  - What is its expected monetary value? ($500)
  - What is its certainty equivalent?
    - Monetary value acceptable in lieu of lottery
    - $400 for most people
  - Difference of $100 is the insurance premium
    - There's an insurance industry because people will pay to reduce their risk
    - If everyone were risk-neutral, no insurance needed!
  - It's win-win: you'd rather have the $400 and the insurance company would rather have the lottery (their utility curve is linear and they have many lotteries)



# Example: Human Rationality?

- Famous example of Allais (1953)
  - A: [0.8, $4k;  0.2, $0]
  - B: [1.0, $3k;  0.0, $0]
  - C: [0.2, $4k;  0.8, $0]
  - D: [0.25, $3k;  0.75, $0]
- Most people prefer B > A, C > D
- But if U($0) = 0, then
  - B > A ⇒ U($3k) > 0.8 U($4k)
  - C > D ⇒ 0.8 U($4k) > U($3k)