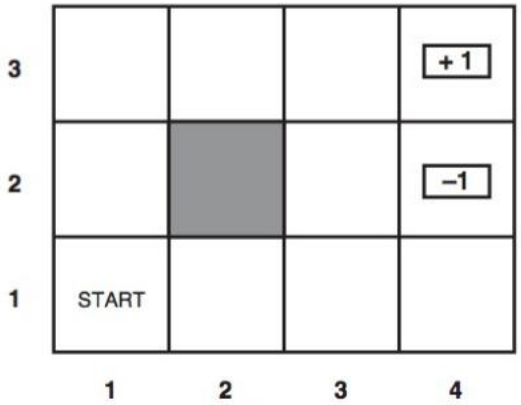
17 MAKING COMPLEX DECISIONS



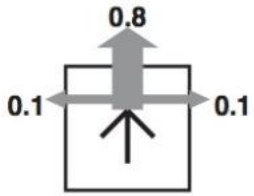


Figure 17.1 (a) A simple 4×3 environment that presents the agent with a sequential decision problem. (b) Illustration of the transition model of the environment: the "intended" outcome occurs with probability 0.8, but with probability 0.2 the agent moves at right angles to the intended direction. A collision with a wall results in no movement. The two terminal states have reward +1 and –1, respectively, and all other states have a reward of –0.04.

CS 188: Artificial Intelligence

Markov Decision Processes

Instructors: Dan Klein and Pieter Abbeel

University of California, Berkeley

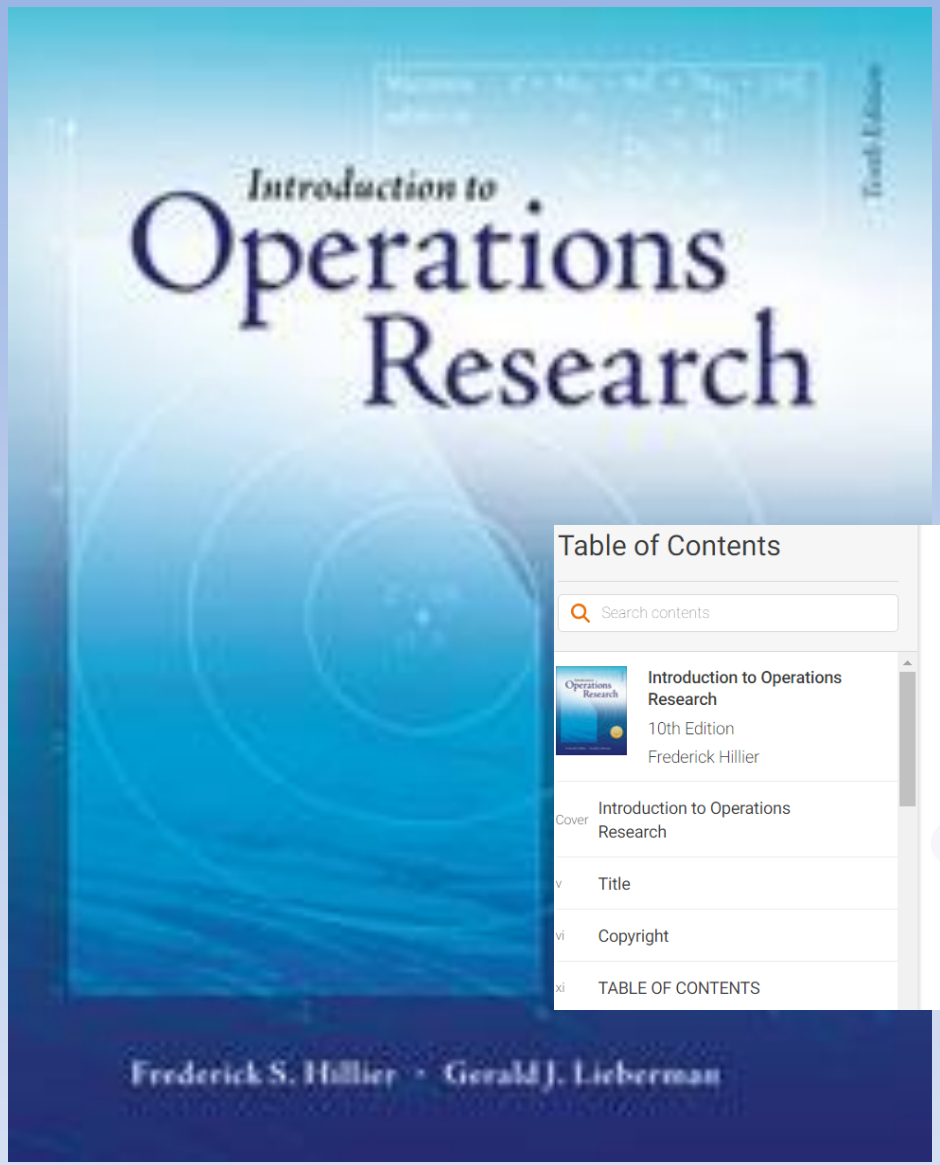


Table of Contents

Search contents

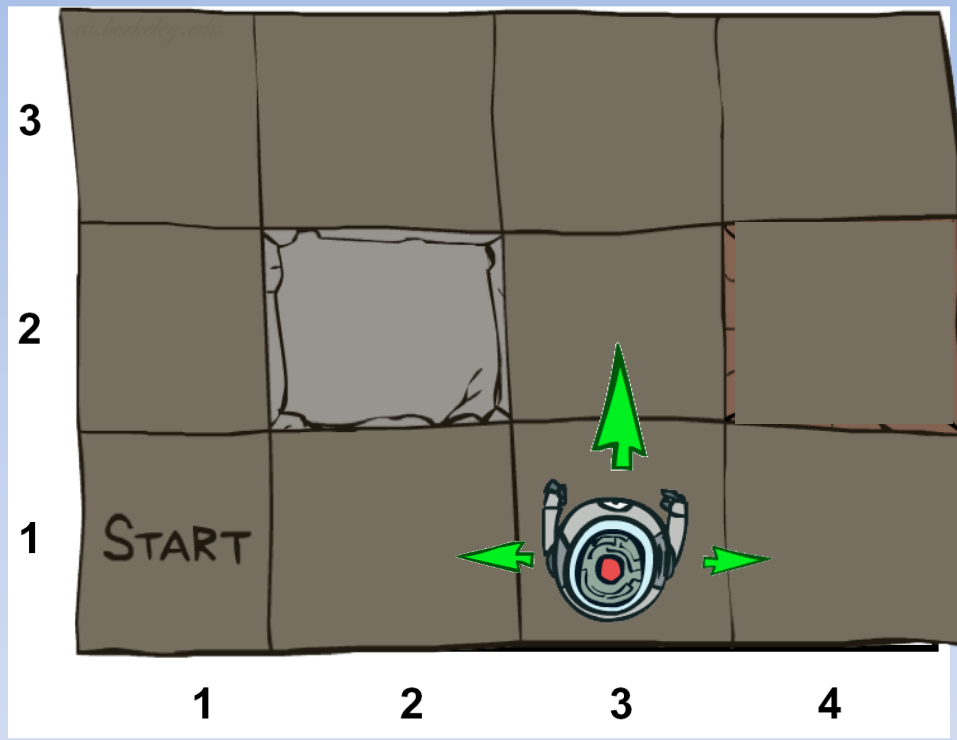
Introduction to Operations Research
10th Edition
Frederick Hillier

CHAPTER 19

Markov Decision Processes

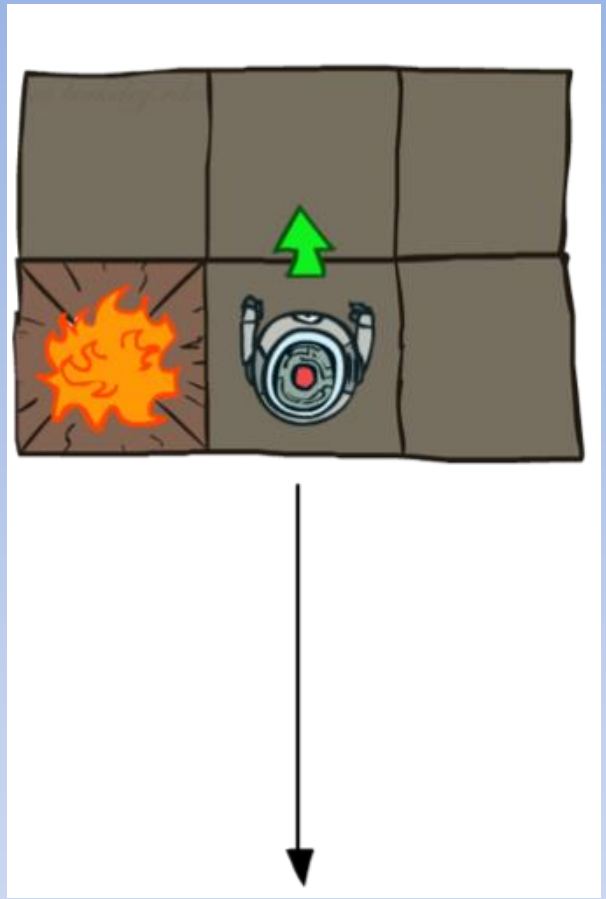
Grid World

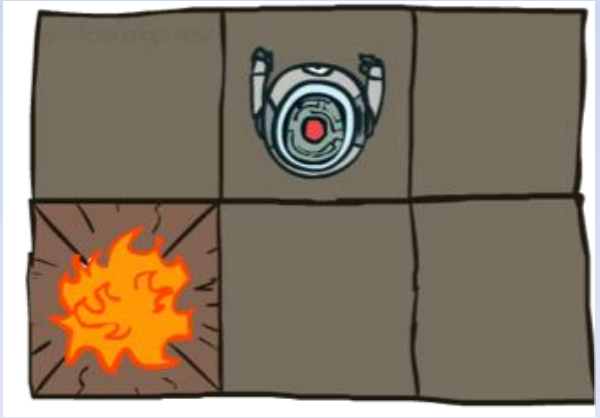
- Robot wants to find gems!
- Robot wants to avoid fire pits!
- Robots wheels slip:
  - Sometimes when trying to go forward, it actually goes left/right.



Deterministic Grid Word

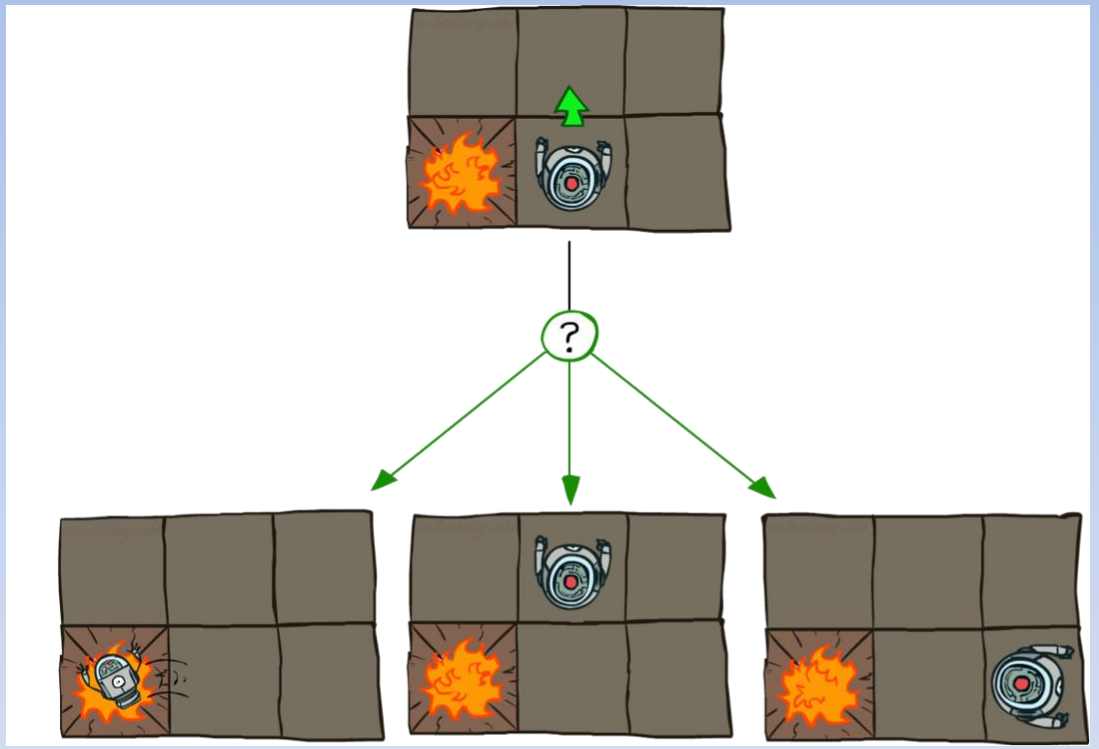
- Actions always produce same result.
- Solve w/ State-Space Search
- Solution is an action sequence





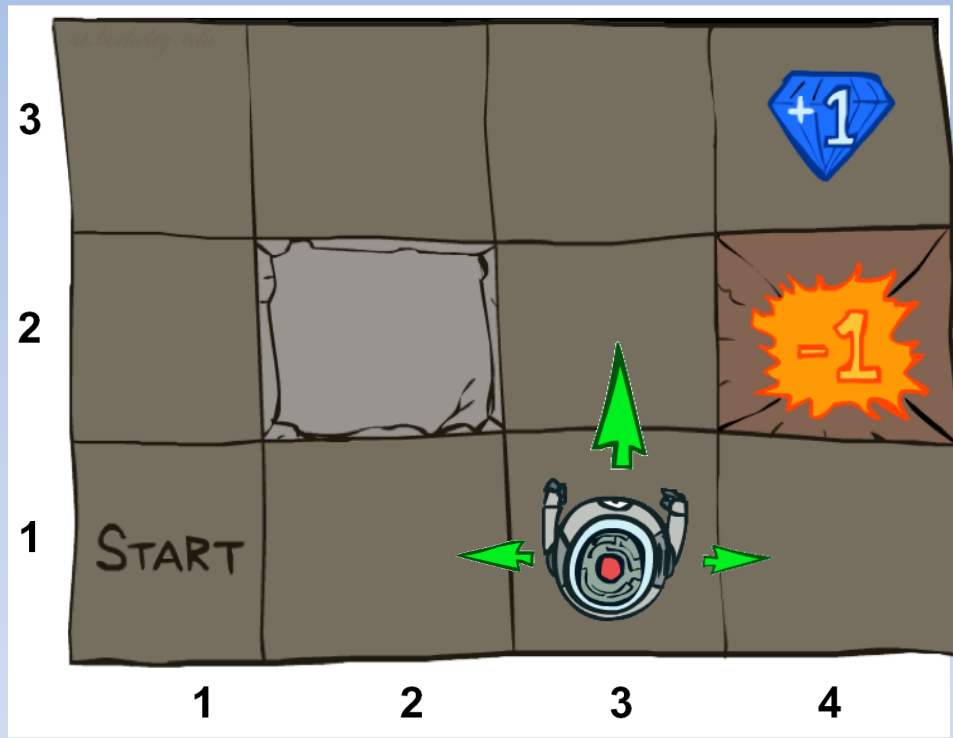
Stochastic Grid World

- Action results have uncertainty



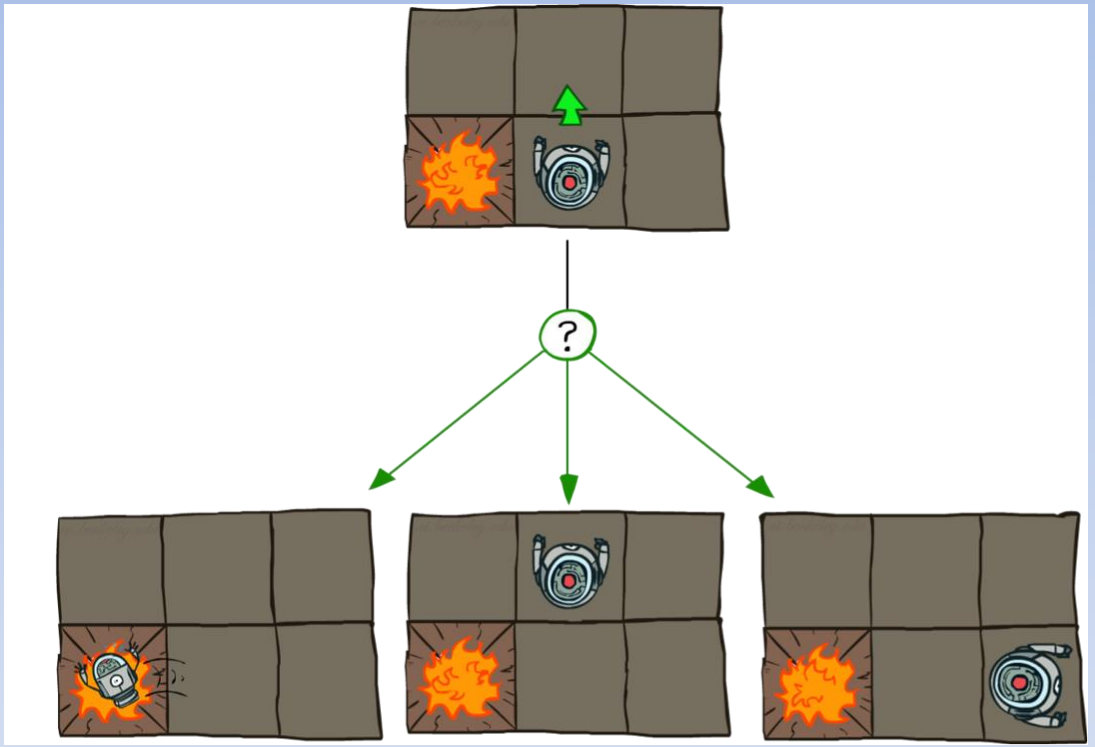
Grid World

- 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

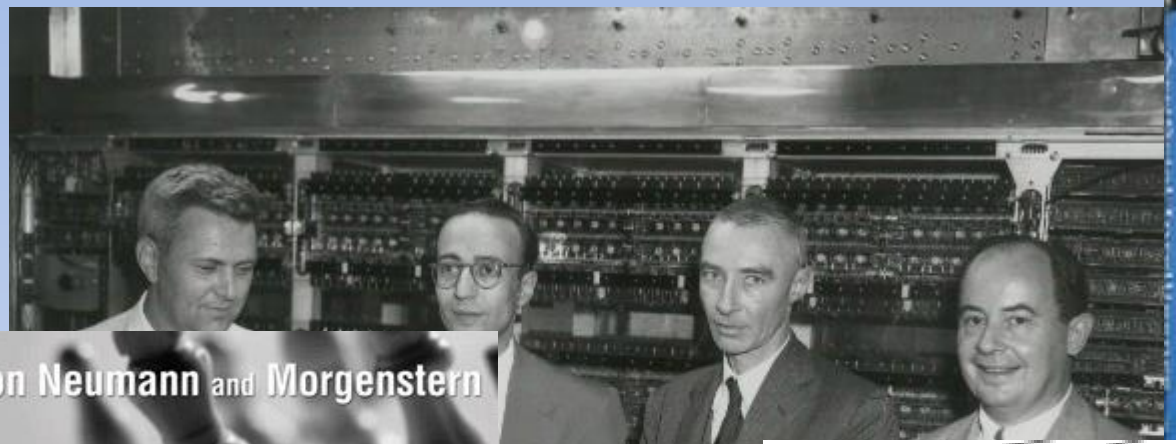


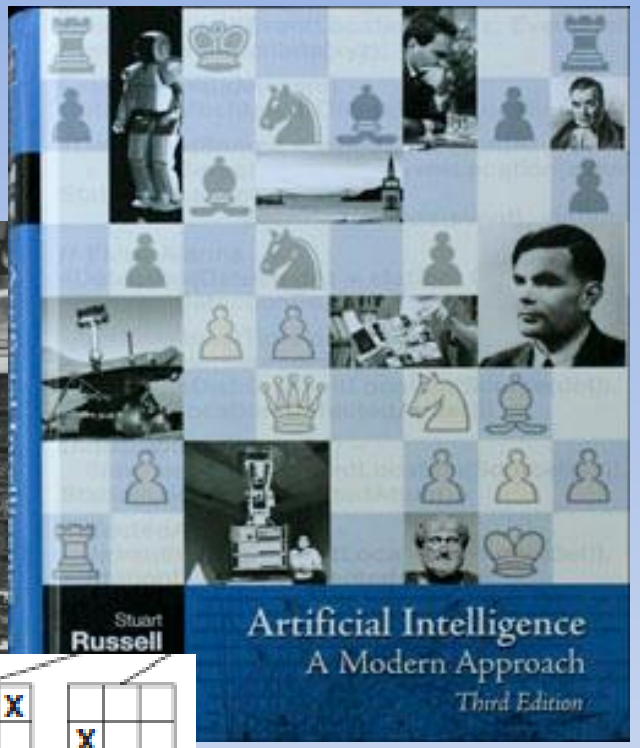
Dealing w/ Uncertainty

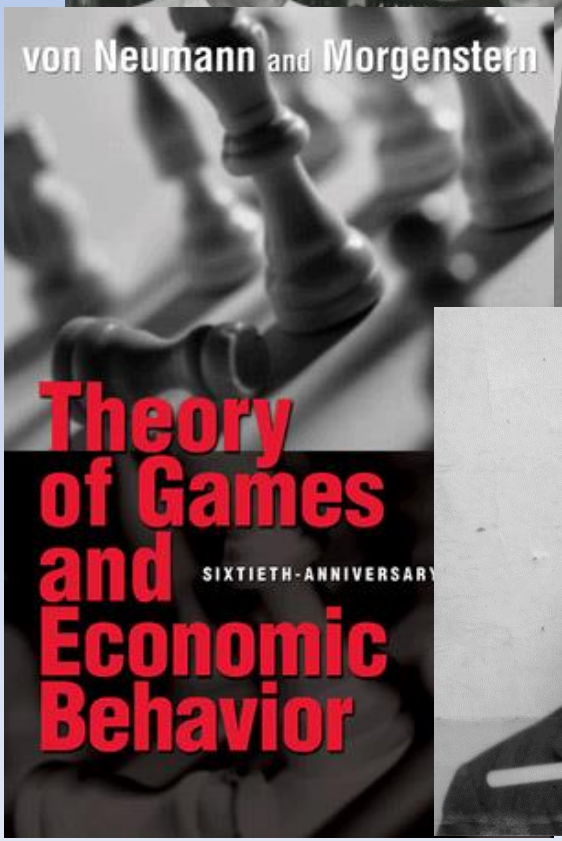
- Action results have uncertainty
- How have we handled this?

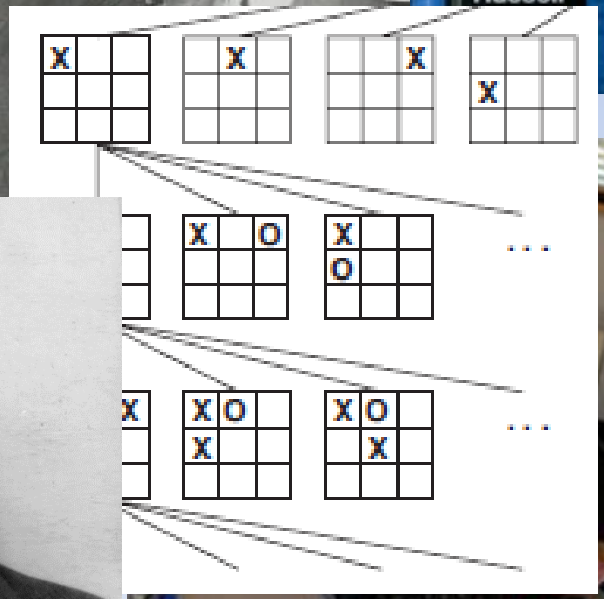


Adversarial Search / Min.I.maX





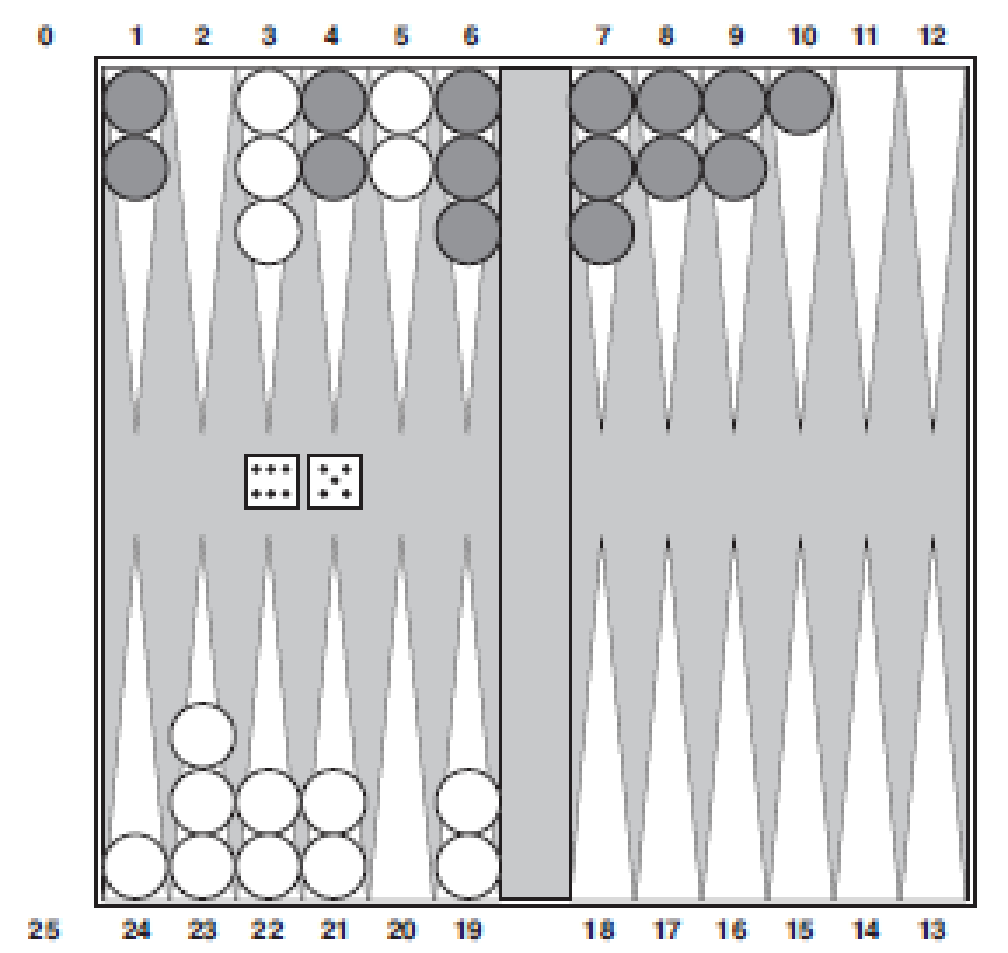




Games of Chance

- Many games include an element of chance
  - Rolling dice
  - Spinning wheel
  - Drawing a card from shuffled deck.
- Games of chance called **Stochastic Games**

Backgammon



(5-10,5-11), (5-11,19-24), (5-10,10-16), (5-11,11-16)

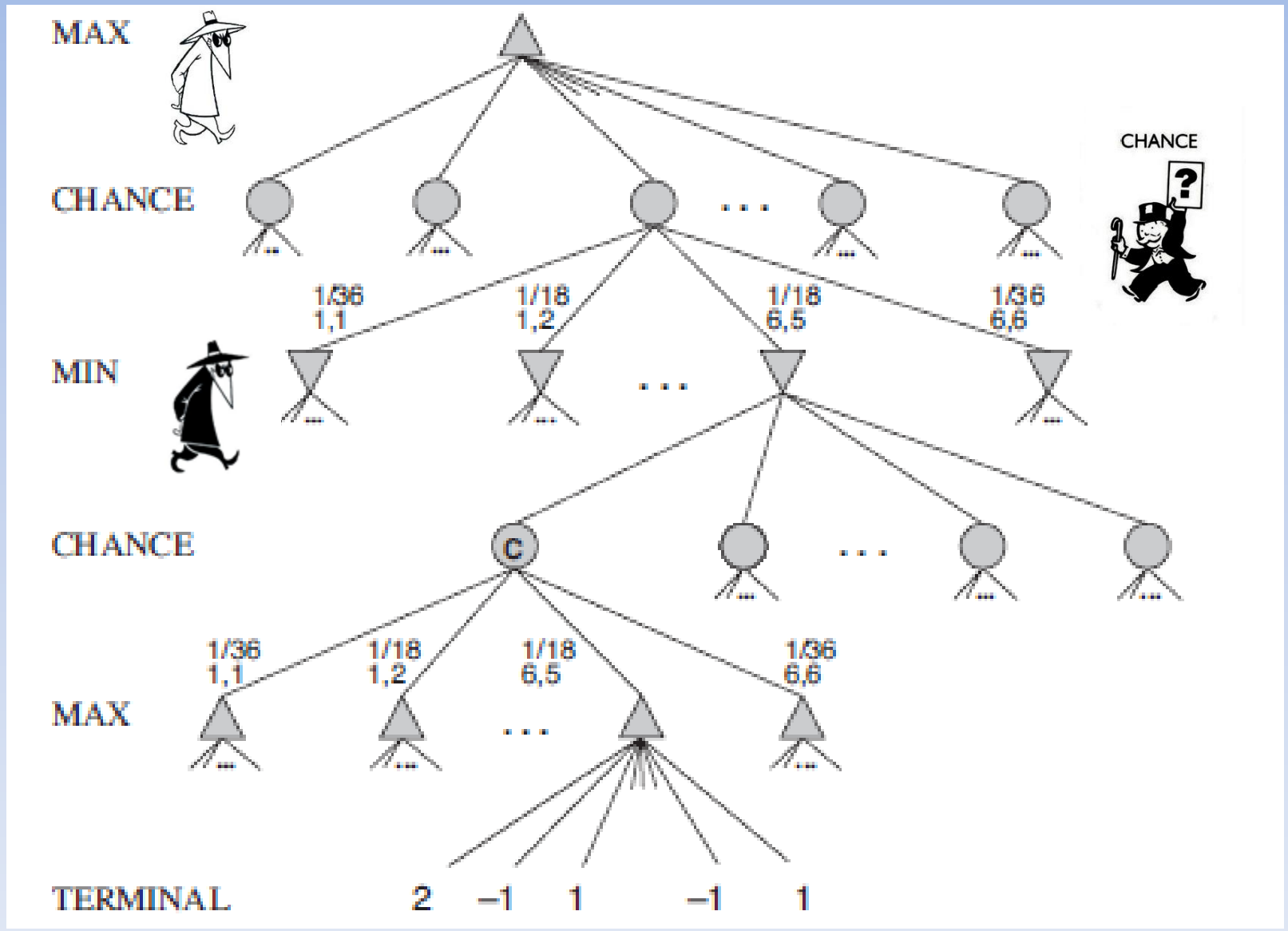
Min & Max w/ Chance

Max

Min

- We can extend Minimax to handle Chance.
- Chance introduced as new Player, along with Max & Min

Game Trees w/ Chance



Expectiminimax

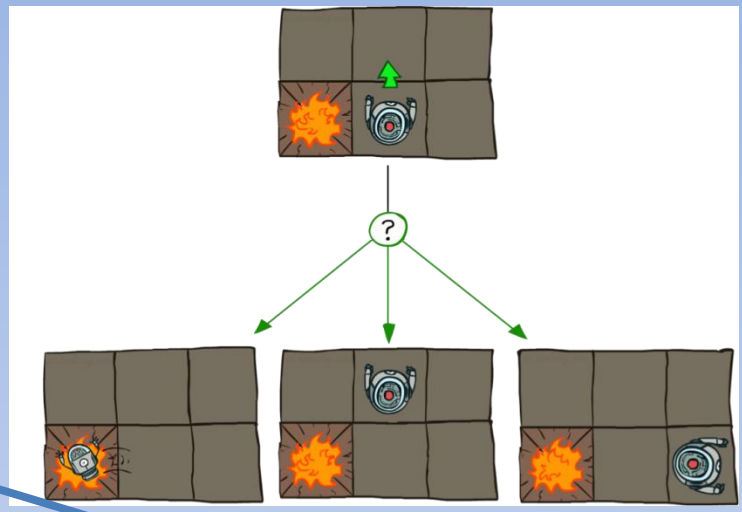
- Returns an *expected* minimax value
- Can be modified to return actions
- Can also be modified to do alpha-beta pruning.
  - More complicated and less effective than in deterministic games.

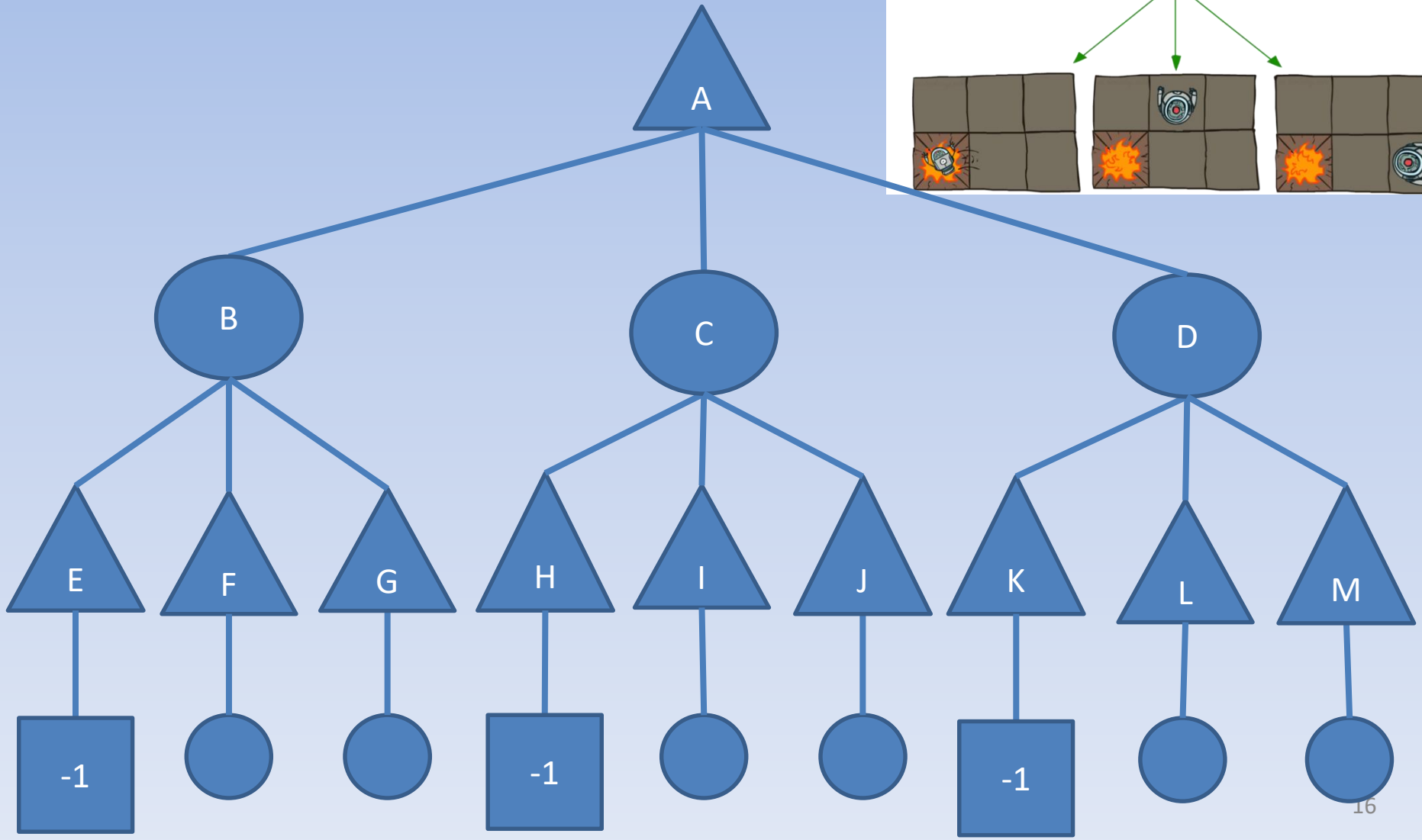
Expectiminimax

- Returns an *expected* minimax value
- Can be modified to return actions
- Can also be modified to do alpha-beta pruning.
  - More complicated and less effective than in deterministic games.

```
function ExpectiMinimax(s, d)
    if s is a terminal state then return Max's payoff at s
    else if d = 0 then return Eval(s)
    else if s is a "chance" node then
        return Σ_{t∈children(s)} P(t|s) ExpectiMinimax(t, d − 1)
    else if it is Max's move at s then
        return max{ExpectiMinimax(result(a, s), d − 1) : a is applicable to s}
    else return min{ExpectiMinimax(result(a, s), d − 1) : a is applicable to s}
```

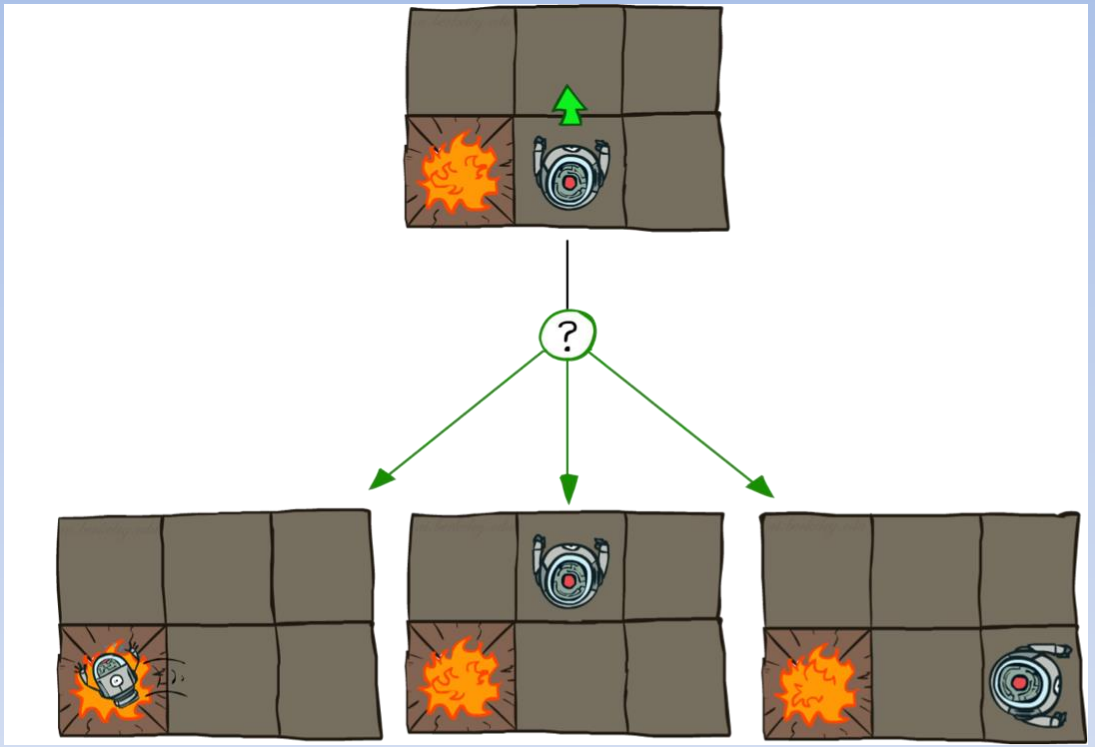
Stochastic Grid World





Stochastic Grid World w/ Expectiminimax

- From any state, calculate it's value/utility.
- From any state, choose best action.



What is Markov about MDPs?

Andrey Markov
(1856-1922)

- "Markov" generally means that given the present state, the future and the past are independent

- For Markov decision processes, "Markov" means action outcomes depend only on the current state

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

$$=$$

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

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

Andrey (Andrei) Andreyevich Markov

- Markov Assumption:
- The FUTURE is INDEPENDENT of the PAST given the PRESENT.

Born	14 June 1856 N.S. Ryazan, Russian Empire
Died	20 July 1922 (aged 66) Petrograd, Russian SFSR
Residence	Russia
Nationality	Russian
Fields	Mathematician
Institutions	St. Petersburg University
Alma mater	St. Petersburg University
Doctoral advisor	Pafnuty Chebyshev
Doctoral students	Abram Besicovitch Nikolai Günther Veniamin Kagan Jacob Tamarkin J. V. Uspensky Georgy Voronoy
Known for	Markov chains; Markov processes; stochastic

Markov Assumption: The FUTURE is INDEPENDENT of the PAST given the PRESENT.

- Definition: We say that a dynamic system over the template variables X satisfies the Markov assumption if, for all $t \geq 0$,
  - $(X^{(t+1)} \perp X^{(0:(t-1))} \mid X^{(t)})$
- Call These:
  - MARKOVIAN SYSTEMS

Markov Decision Process

MARKOV DECISION PROCESS

To sum up: a sequential decision problem for a fully observable, stochastic environment with a Markovian transition model and additive rewards is called a **Markov decision process**, or **MDP**, and consists of a set of states (with an initial state s_0); a set $\text{ACTIONS}(s)$ of actions in each state; a transition model $P(s' \mid s, a)$; and a reward function $R(s)$.[1]

- What does a solution look like?

Solutions

- In deterministic single-agent search problems, we wanted an optimal plan, or sequence of actions, from start to a goal
- This will not work with MDP!
- Given a plan $[a_1, a_2, a_3, \cdots, a_n]$
  - After first action a_1 may end up in any of states $\{s_1, \cdots, s_m\}$
  - Probability of resulting state is P(s' | s_0, a_1)

- What can we do????

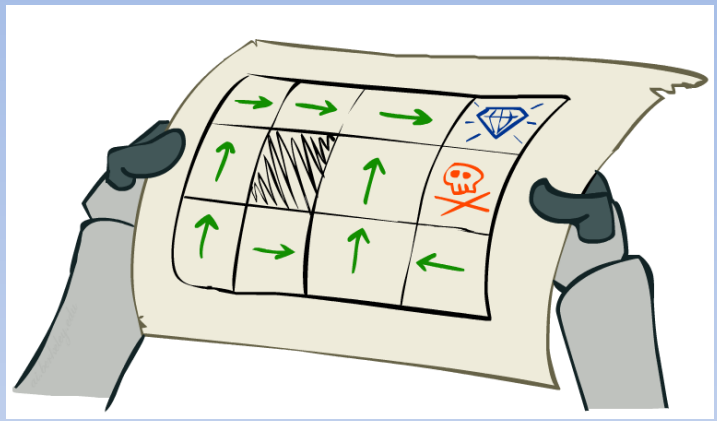
Policies

- With MDP recognize that we never know for sure when we might end up in any particular state.
- Need to know what to do not just from the start state, but from any state!!!
  - Sounds costly.
  - Need generalization (later).

Policies

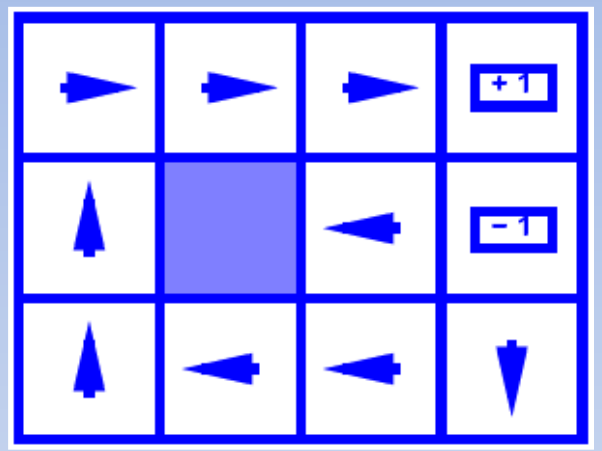
- For MDPs, we want an optimal policy
  - $\pi^*: S \rightarrow A$
  - A policy π gives an action for each state
  - An optimal policy is one that maximizes expected utility if followed
  - An explicit policy defines a reflex agent

- Expectimax didn't compute entire policies
  - It computed the value for a single state only

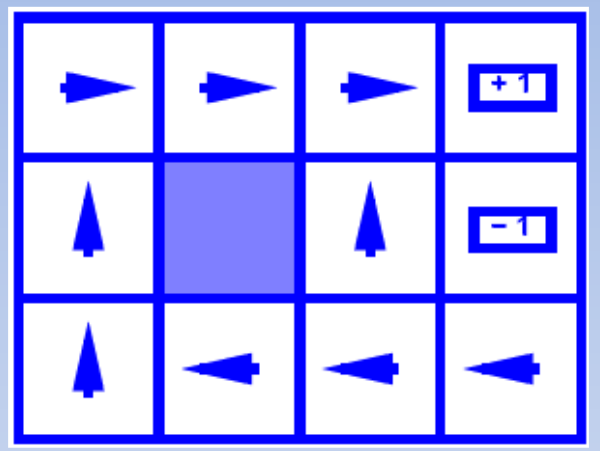


Optimal policy when R(s, a, s') = -0.03 for all non-terminals s

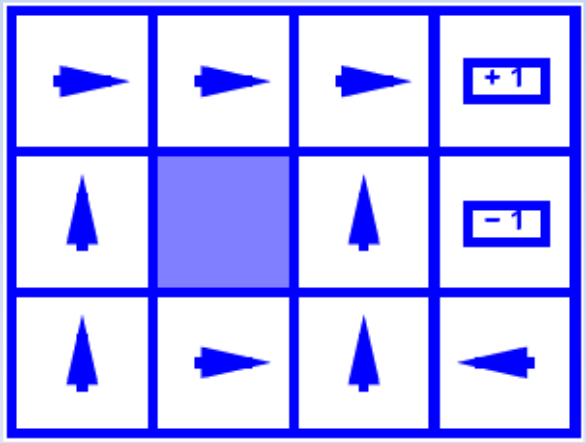
Optimal Policies



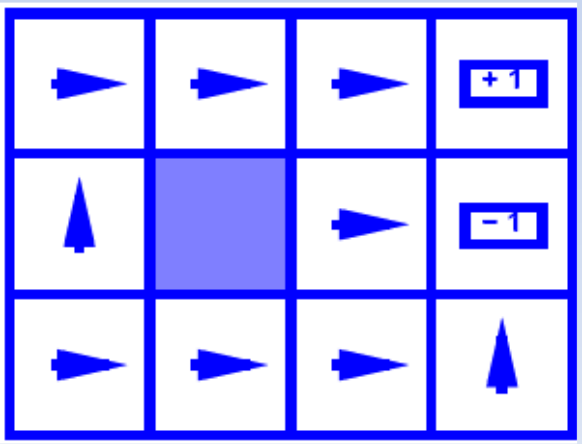
R(s) = -0.01



R(s) = -0.03

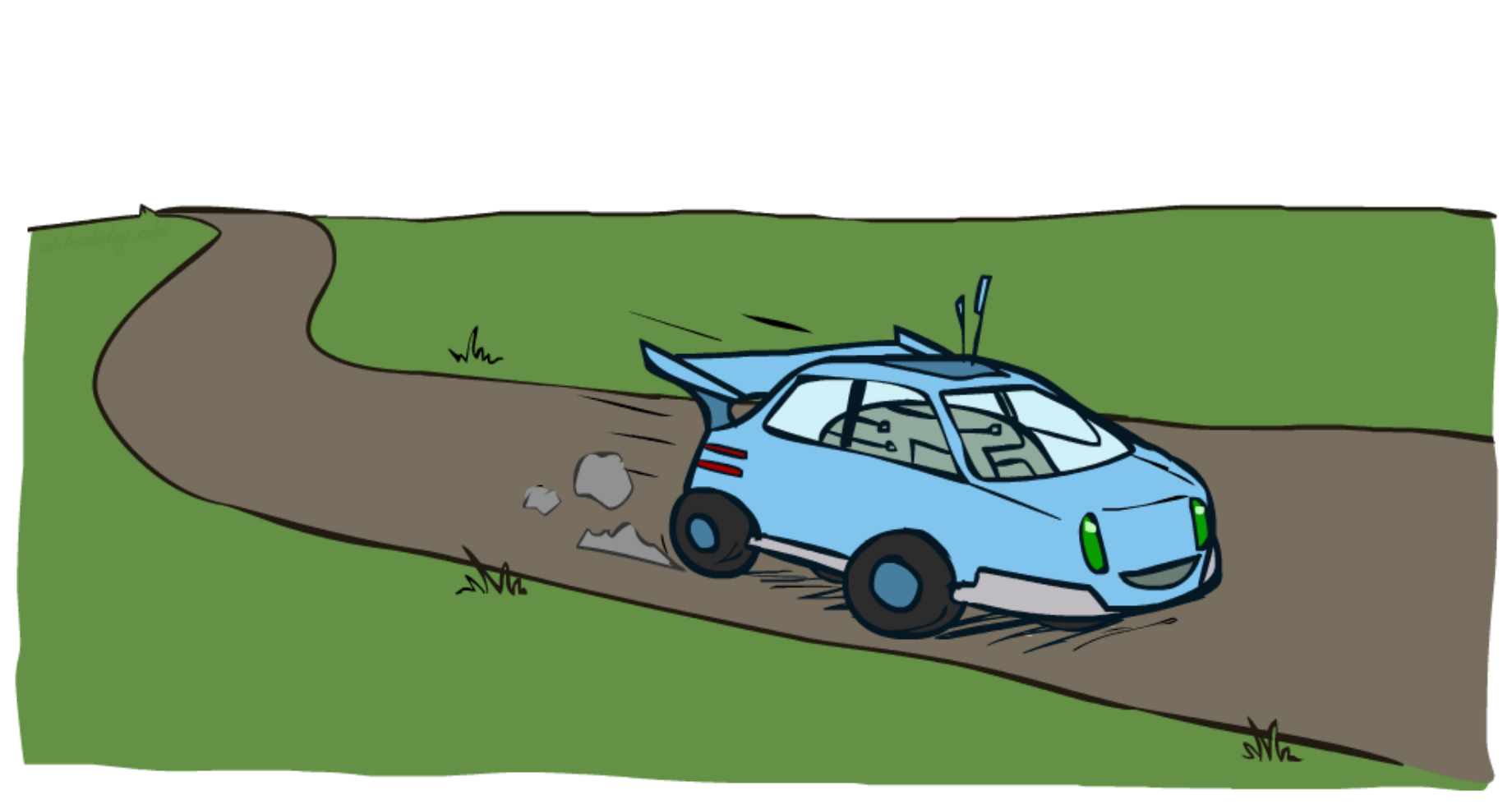


R(s) = -0.4

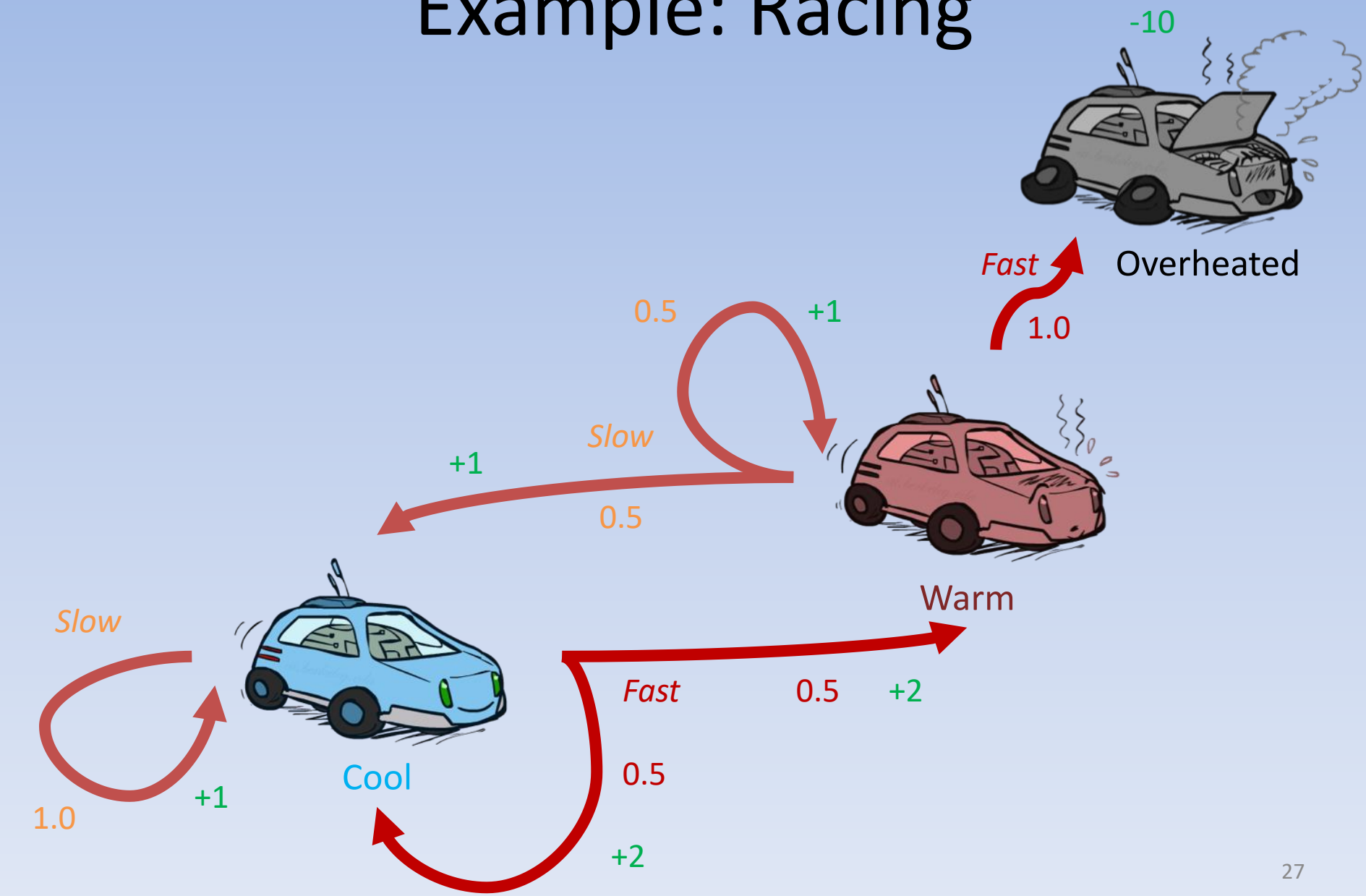


R(s) = -2.0

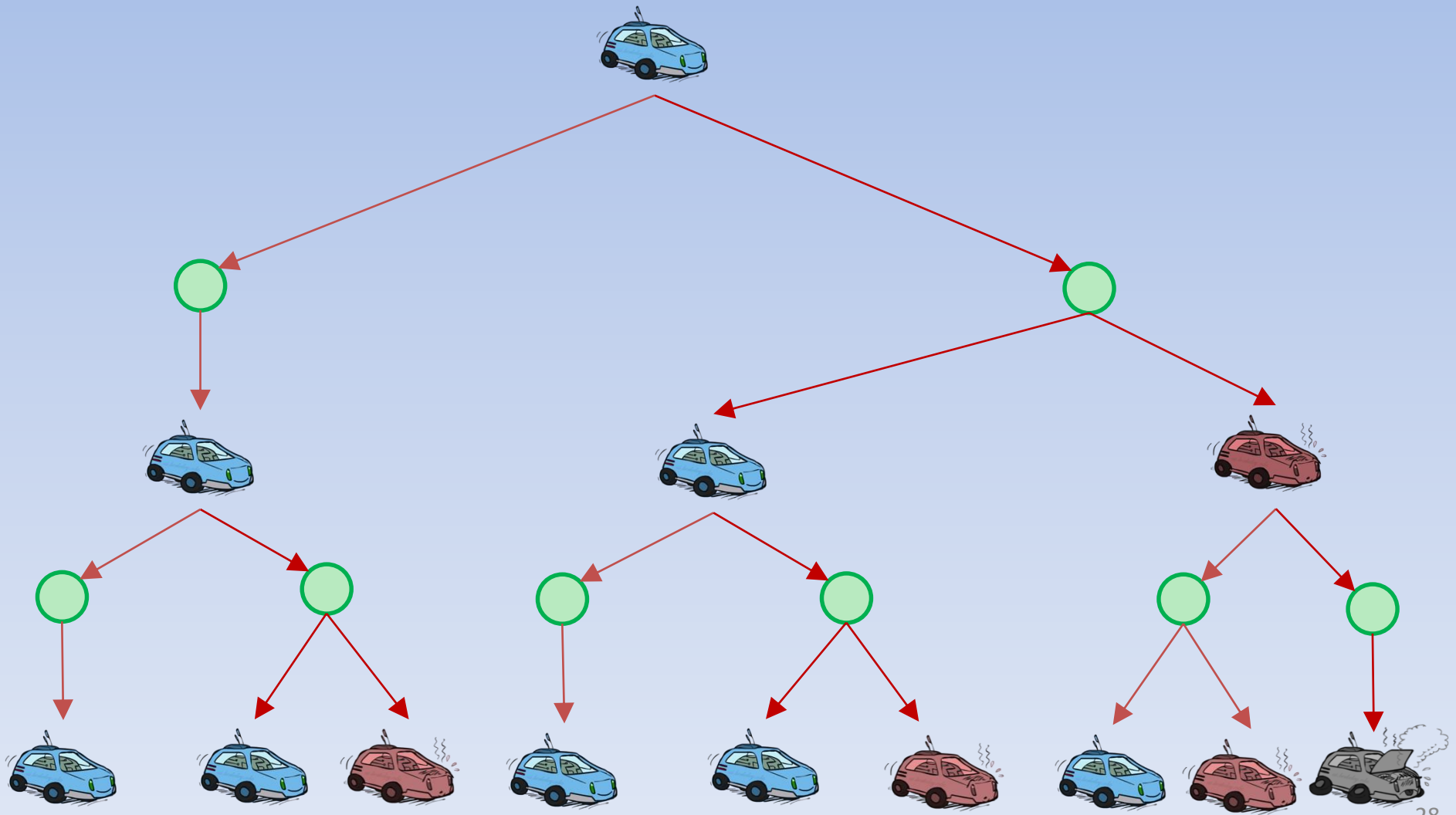
Example: Racing



Example: Racing

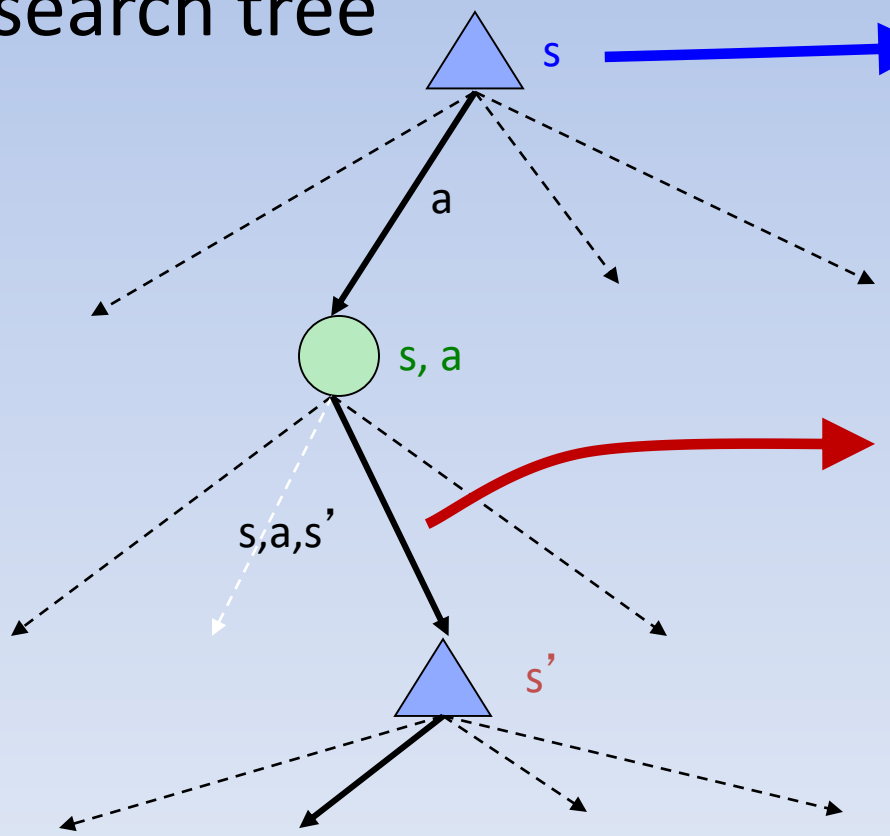


Example: Racing



MDP Search Trees

- Each MDP state projects an expectimax-like search tree



s is a *state*

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

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

R(s,a,s')

Rewards

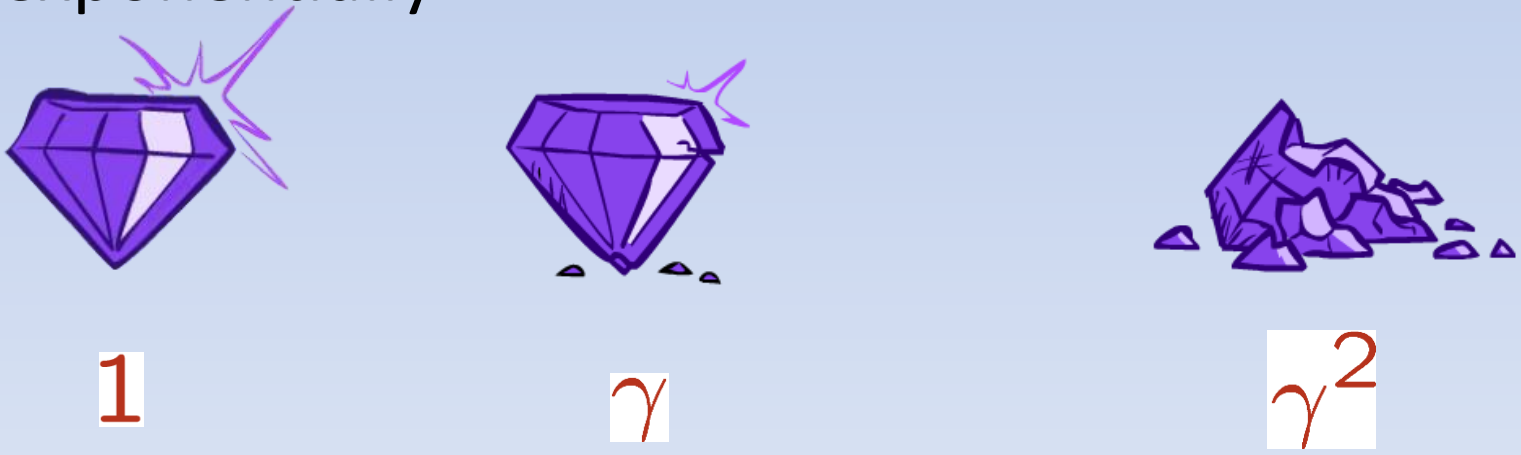
- R(s) or R(s,a,s')
- Expectiminimax only worried about terminal states.
- How should we evaluate the sequence of rewards??

Utilities of Sequences

- What preferences should an agent have over reward sequences?
- More or less? [1, 2, 2] or [2, 3, 4]
- Now or later? [0, 0, 1] or [1, 0, 0]

Discounting

- It's reasonable to maximize the sum of rewards
- It's also reasonable to prefer rewards now to rewards later
- One solution: values of rewards decay exponentially



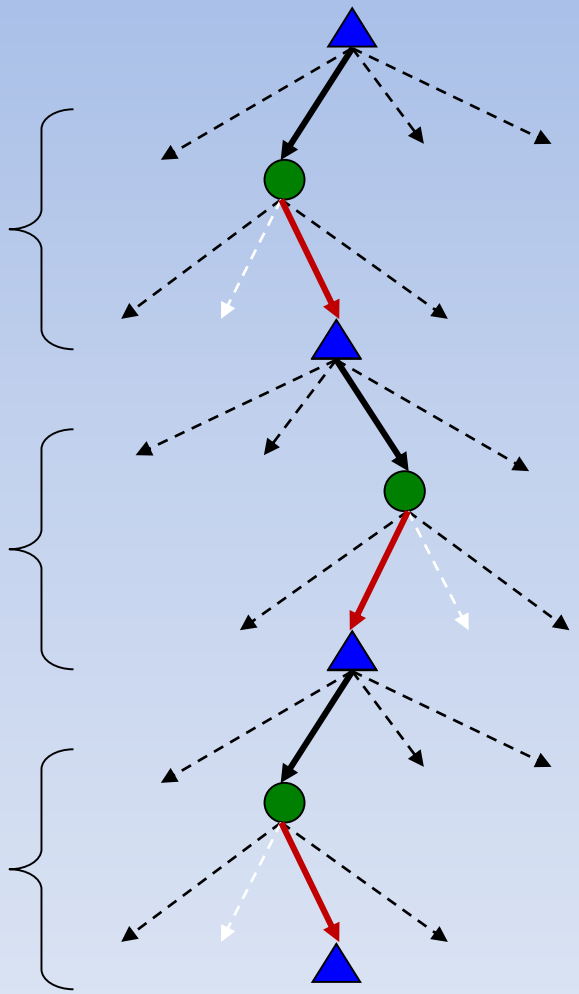
Discounting

- 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
- Example: discount of 0.5
  - U([1,2,3]) = 1*1 + 0.5*2 + 0.25*3
  - U([1,2,3]) < U([3,2,1])

1

γ

γ^2



Stationary Preferences

- Theorem: if we assume stationary preferences:

$$[a_1, a_2, \cdots] \succ [b_1, b_2, \cdots]$$
$$\Updownarrow$$
$$[r, a_1, a_2, \cdots] \succ [r, b_1, b_2, \cdots]$$

- Then: there are only two ways to define utilities
  - Additive utility:

$$\mathrm{U}([r_0, r_1, r_2, \cdots]) = r_0 + r_1 + r_2 + \cdots$$

  - Discounted utility:

$$\mathrm{U}([r_0, r_1, r_2, \cdots]) = r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots$$

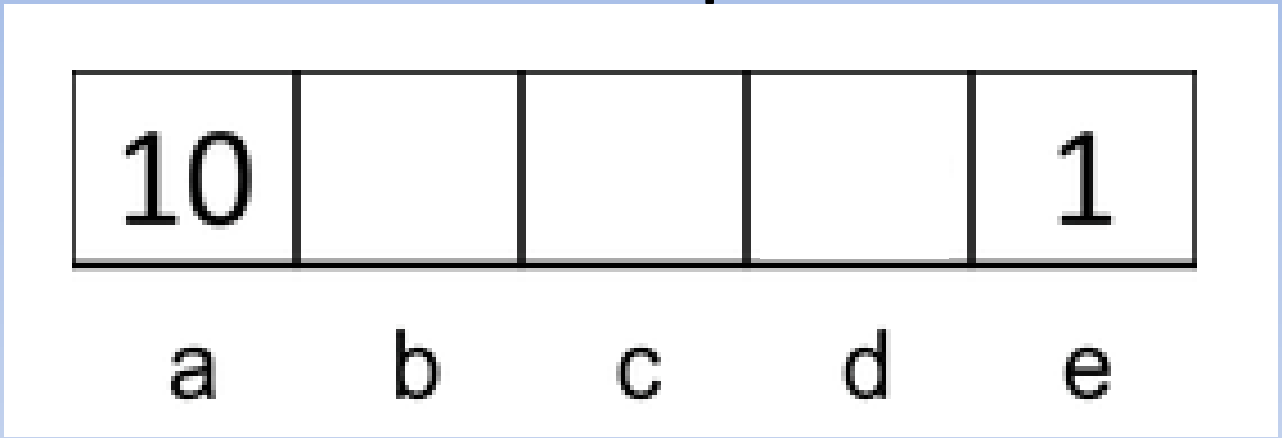
Infinite Utilities?!

- 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 $0 < \gamma < 1$

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

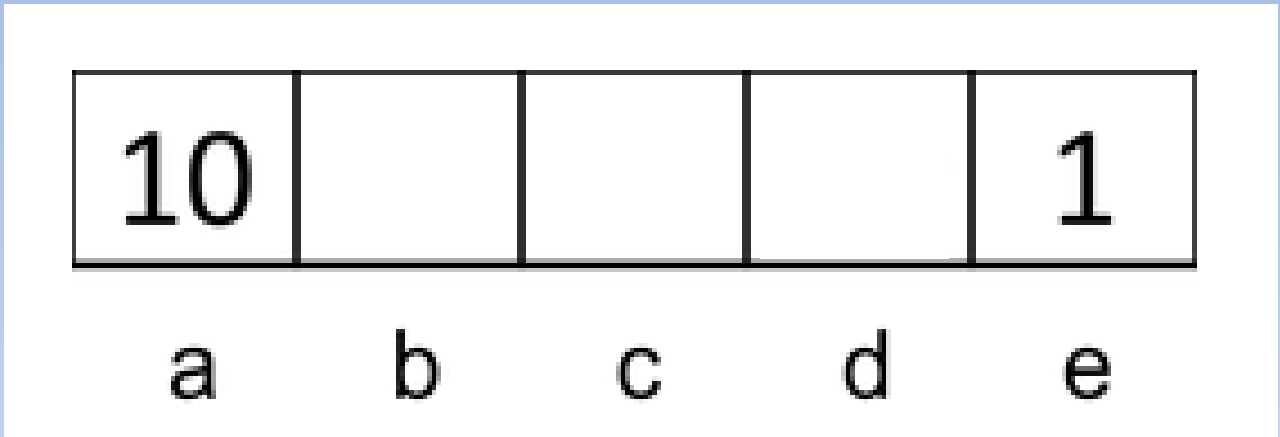
    - Smaller γ means smaller "horizon" – shorter term focus
  - Absorbing state: guarantee that for every policy, a terminal state will eventually be reached (like "overheated" for racing)

Discounting Example



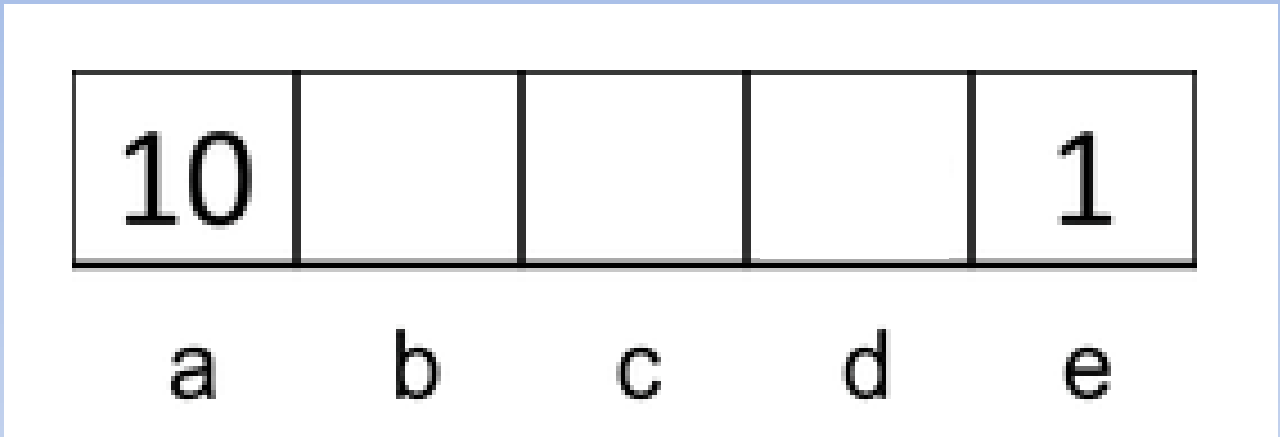
- Our World
- 5 State = {a, b, c, d, e}
- 3 Actions = {Left, Right, Exit}
  - Exit available only in a & e.
  - Exit from a yields reward of 10
  - Exit from e yields reward of 1
- Transitions are deterministic

Question 1



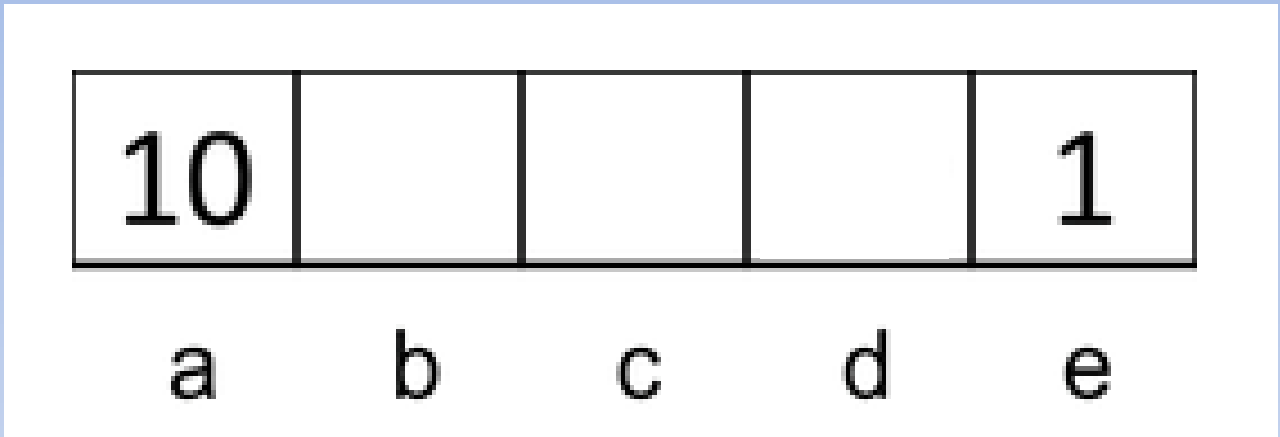
- For $\gamma = 1$, what is the optimal policy?

Question 1



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

Question 1

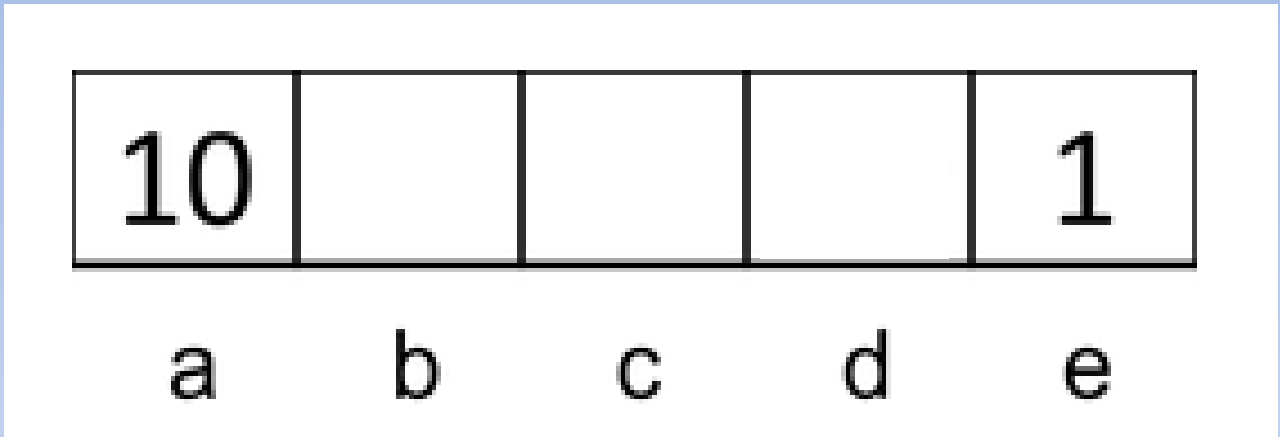


- For which γ, are West and East equally good when in state d?

Stochastic Element

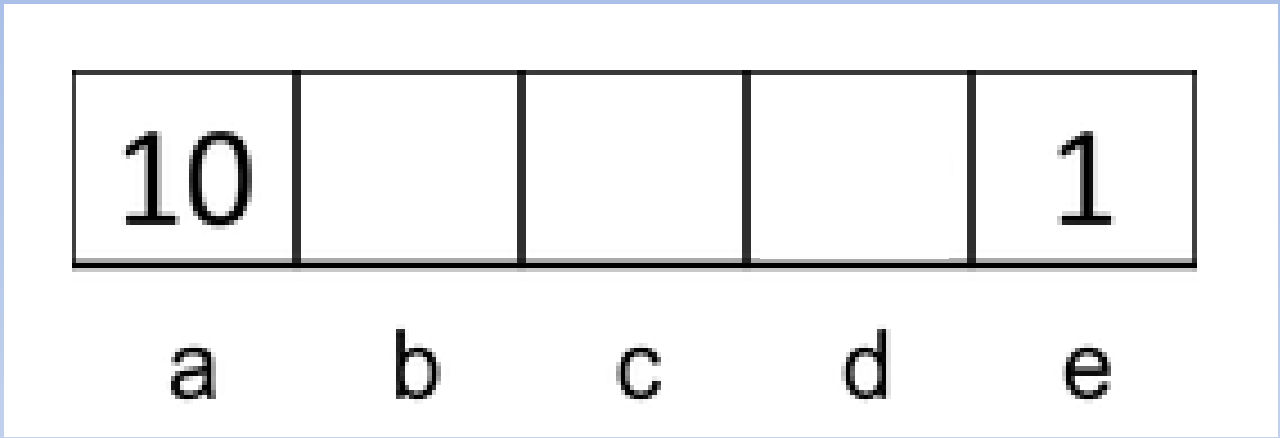
- Not lets add in a stochastic element.
- Actions are sometimes not successful.

Discounting
+ Stochastic Element



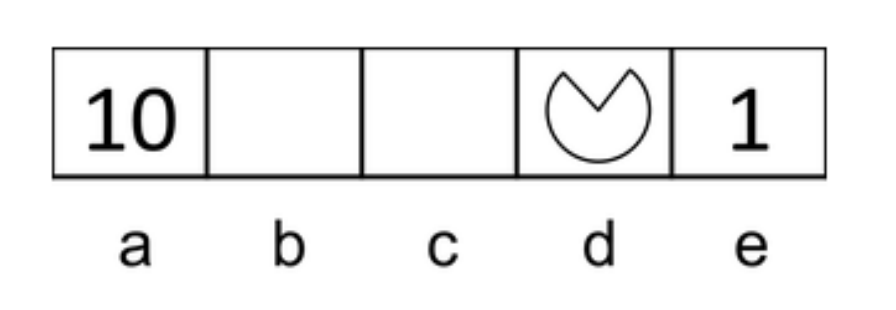
- Our World:
  - 5 State = {a, b, c, d, e}
  - 3 Actions = {Left, Right, Exit}
- **Transitions are no longer deterministic**
- Left and Right are successful 80% of the time.
  - When not success, the agent stays in place.
- Exit is successful 100% of the time

Discounting
+ Stochastic Element



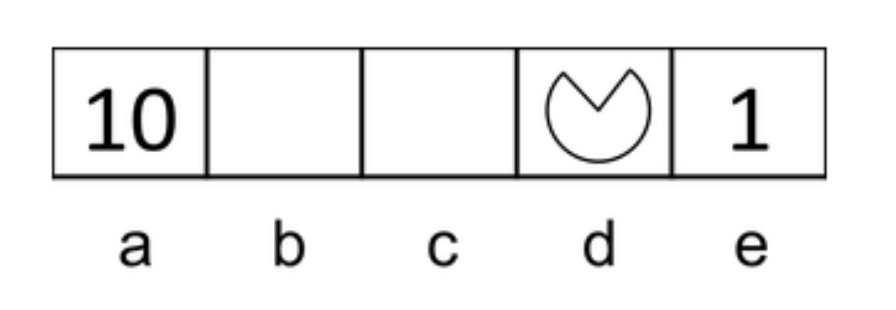
- Left and Right are successful 80% of the time.
  - When not success, the agent stays in place.
- Exit is successful 100% of the time
- Develop transition model:
  - T(d, Right, e) = P(e | d, Right) = ?

Agent Action?



- Our agent is in state d.
- Given
- Given the transition model just developed and $\gamma = 1$, what should the agent do?

Agent Action?



- Our agent is in state d.
- Given
- Given the transition model just developed and $\gamma = 0.1$, what should the agent do?

Solving MDP's

- Problem Review: 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)
    - sometimes R(s)

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

Define Some Optimal Quantities

- The value (utility) of a state s:

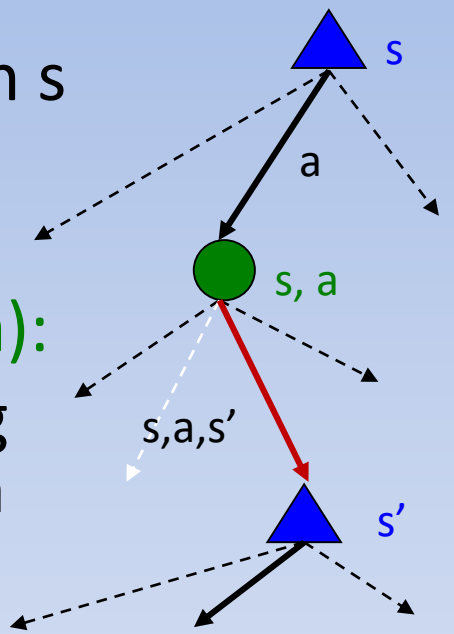
  $V^*(s)$ = expected utility starting in s and acting optimally

- 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



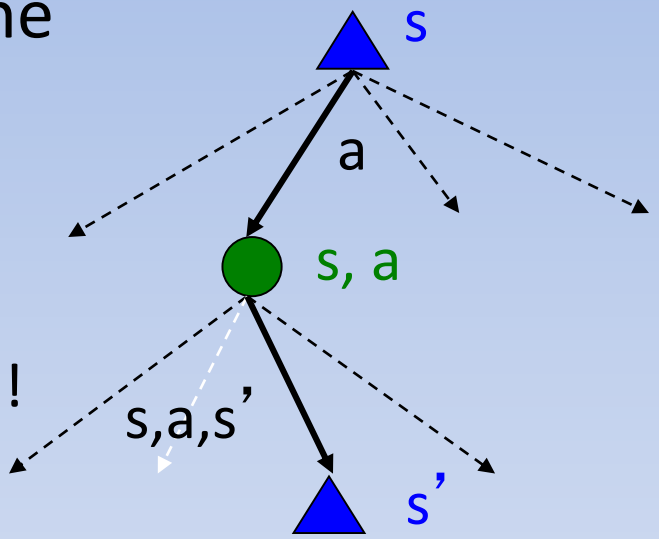
s is a *state*

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

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

Values of States

- 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!
- Can we define the utility/value in a closed form (recursively).



Bellman Equation

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a)U(s')$$

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)U(s') \; . \qquad (17.5)$$

BELLMAN EQUATION

This is called the **Bellman equation**, after Richard Bellman (1957). The utilities of the states—defined by Equation (17.2) as the expected utility of subsequent state sequences—are solutions of the set of Bellman equations. In fact, they are the *unique* solutions, as we show in Section 17.2.3.

Closer Look

$$\sum_{s'} P(s'|s,a)U(s')$$

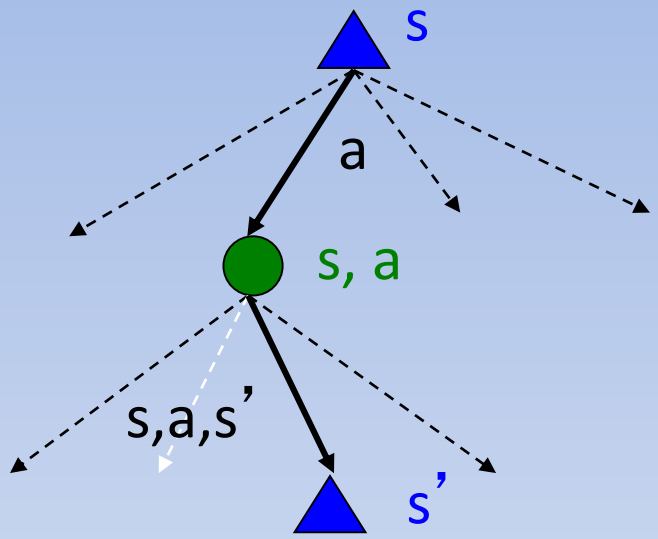
- For every possible state (s') weight its utility U(s') by the probability of moving to that state when taking action a in state s.
- The sum is the Expected Utility of action a in state s.
- We want to take the action with the highest expected utility.

Bellman Equation

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a)U(s')$$

- The expected utility of the best action is then discounted by γ and added to the utility of the current state.
- Total utility is sum of:
  - Utility of current state.
  - Discounted expected utility of the best action for this state.

Bellman Equation



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

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

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

- Alternative with R(s, a, s')
- T(s, a, s') = P(s' | s, a)
- Rewards are given when action is taken.

Richard Bellman

- Beginning 1949 Bellman worked for many years at RAND corporation and it was during this time that he developed dynamic programming.

RAND CORPORATION

OBJECTIVE ANALYSIS. EFFECTIVE SOLUTIONS.

About | Support RAND | Press Room | Events

RESEARCH | LATEST INSIGHTS | POLICY EXPERTS | CAPABILITIES | GRADUATE SCHOOL

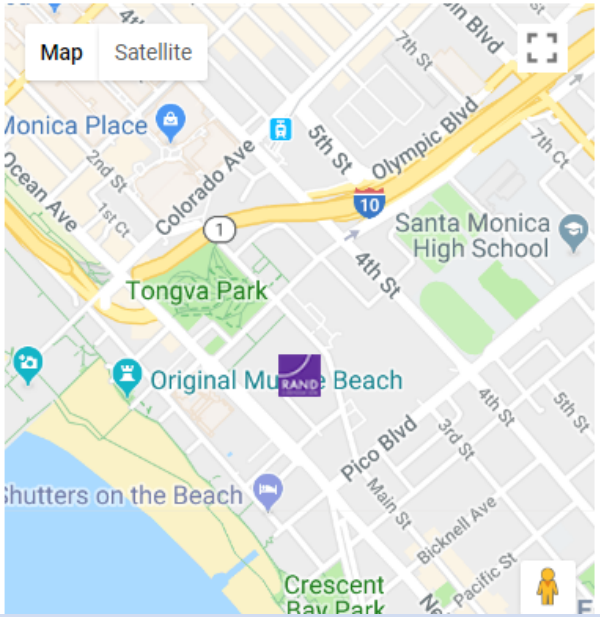
RAND Locations: Santa Monica Office, Headquarters Campus

Santa Monica Office, Headquarters Campus
Pardee RAND Graduate School

1776 Main Street
Santa Monica, California 90401-3208
Tel: (310) 393-0411
Fax: (310) 393-4818

Getting There

RAND's headquarters campus is located in Santa Monica off the I-10 West, not far from the Los Angeles International Airport. The building fronts on Main Street in Santa Monica.



- RAND Corporation

Richard Bellman

Biography MathSciNet

Ph.D. Princeton University 1947

Dissertation: *On the Boundedness of Solutions of Non-Linear Differential and Difference Equations*

Advisor: Solomon Lefschetz

Students:
Click here to see the students listed in chronological order.

Name	School	Year	Descendants
Angel, Edward	University of Southern California	1968	2
Casti, John	University of Southern California	1970	
Cooke, Kenneth	Stanford University	1952	5
Esogbue, Augustine	University of Southern California	1968	2
Ford-Livene, Carlos	University of Southern California	1972	
Hu, Lorinda	University of Southern California	1974	
Kadekodi, Gopal	University of Southern California	1971	
Shoemaker, Christine	University of Southern California	1971	101

Solomon Lefschetz

Biography MathSciNet

Ph.D. Clark University 1911

Dissertation: *On the Existence of Loci with Given Singularities*

Advisor: William Edward Story

Students:
Click here to see the students ordered by family name.

Name	School	Year	Descendants
Smith, Paul	Princeton University	1926	111
Flexner, William	Princeton University	1930	1
Tucker, Albert	Princeton University	1932	1602
Walker, Robert	Princeton University	1934	
Steenrod, Norman	Princeton University	1936	2173
Wallman, Henry	Princeton University	1937	290
Wylie, Shaun	Princeton University	1937	1270
Dowker, Clifford	Princeton University	1938	9
Fox, Ralph	Princeton University	1939	614
Tukey, John	Princeton University	1939	1475
Begle, Edward	Princeton University	1940	286
Stone, Arthur	Princeton University	1941	20
Truesdell, III, Clifford	Princeton University	1944	53
Bellman, Richard	Princeton University	1947	118
Diliberto, Stephen	Princeton University	1947	360

Solomon Lefschetz

Born	3 September 1884 Moscow, Russian Empire
Died	5 October 1972 (aged 88) Princeton, New Jersey, U.S.
nality	American
	Algebraic topology
	niversity of Nebraska ity of Kansas versity
Alma mater	Cla

Albert Tucker

Albert William Tucker

Biography MathSciNet

Ph.D. Princeton University 1932 

Dissertation: *An Abstract Approach to Manifolds*

Advisor: Solomon Lefschetz

Students:
Click here to see the students ordered by family name.

Name	School	Year	Descendants
Johnson, Lloyd	Princeton University	1941	
Gale, David	Princeton University	1949	69
Nash, Jr., John	Princeton University	1950	1
Shapley, Lloyd	Princeton University	1953	17
Isbell, John	Princeton University	1954	4
Minsky, Marvin	Princeton University	1954	1291
Whittlesey, Emmet	Princeton University	1957	
Balinski, Michel	Princeton University	1959	129
Singleton, Robert	Princeton University	1962	
Parsons, Torrence	Princeton University	1966	78
Maurer, Stephen	Princeton University	1972	1
Stein, Marjorie	Princeton University	1972	

Marvin Minsky (AI)

John Nash

Albert W. Tucker

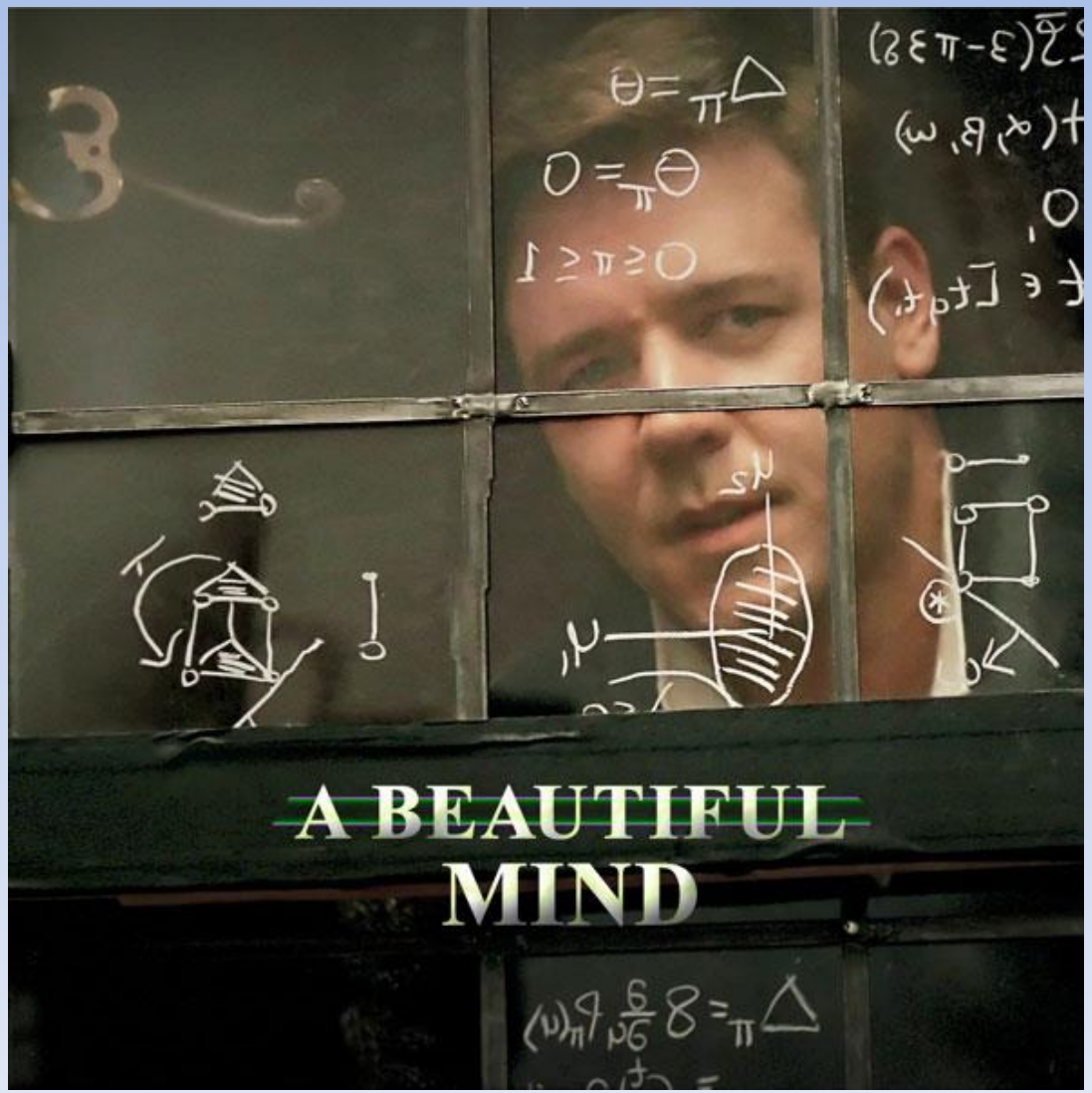
Albert William Tucker

Born	28 November 1905 Oshawa, Ontario, Canada
Died	25 January 1995 (aged 89) Hightstown, New Jersey, U.S.
Residence	U.S.
Nationality	Canadian American
Fields	Mathematician: Combinatorial topology Optimization
Institutions	Princeton University

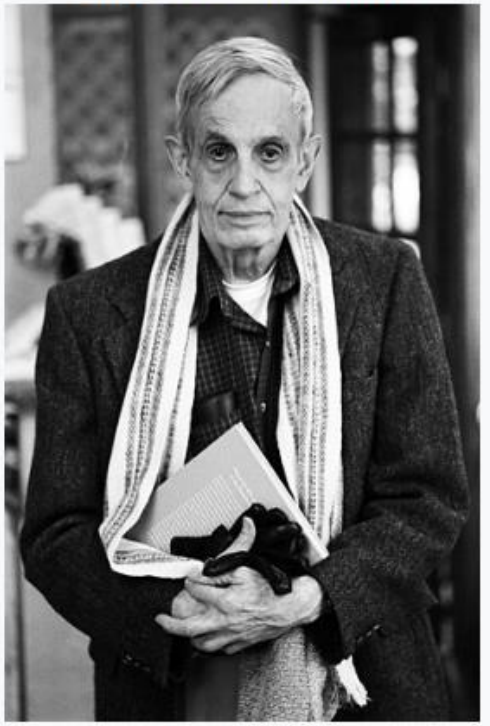
John Nash & RAND

- In 1950 Nash received his doctorate from Princeton with a thesis entitled Non-cooperative Games.
- In the summer of 1950 he worked for the RAND Corporation where his work on game theory made him a leading expert on the Cold War conflict which dominated RAND's work.
- John Nash worked there from time to time over the next few years as the Corporation tried to apply game theory to military and diplomatic strategy.

John Forbes Nash Jr.

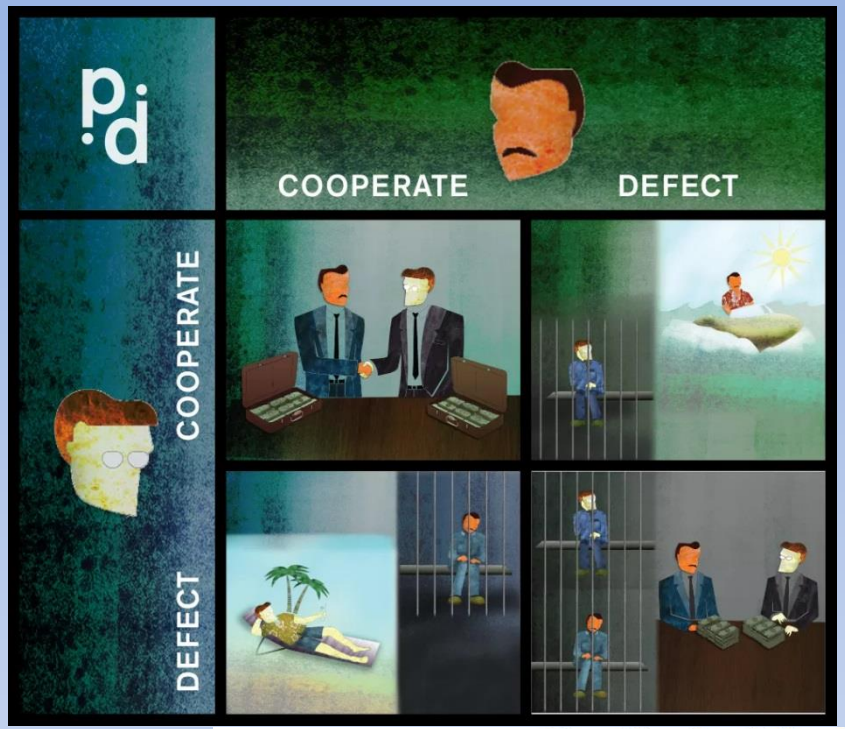


John Forbes Nash Jr.



Nash in 2006

Born	June 13, 1928 Bluefield, West Virginia, U.S.
Died	May 23, 2015 (aged 86) Monroe Township, Middlesex County, New Jersey, U.S.
Citizenship	United States
Fields	Mathematics Economics
Institutions	Massachusetts Institute of Technology Princeton University
Alma mater	Carnegie Institute of Technology



Prisoner's Dilemma

PRISONER'S DILEMMA

Consider the following story: Two alleged burglars, Alice and Bob, are caught red-handed near the scene of a burglary and are interrogated separately. A prosecutor offers each a deal: if you testify against your partner as the leader of a burglary ring, you'll go free for being the cooperative one, while your partner will serve 10 years in prison. However, if you both testify against each other, you'll both get 5 years. Alice and Bob also know that if both refuse to testify they will serve only 1 year each for the lesser charge of possessing stolen property. Now Alice and Bob face the so-called **prisoner's dilemma**: should they testify or refuse? Being rational agents, Alice and Bob each want to maximize their own expected utility. Let's assume that Alice is callously unconcerned about her partner's fate, so her utility decreases in proportion to the number of years she will spend in prison, regardless of what happens to Bob. Bob feels exactly the same way. To help reach a rational decision, they both construct the following payoff matrix:

	Alice:testify	*Alice:refuse*
Bob:testify	$A = -5, B = -5$	$A = -10, B = 0$
Bob:refuse	$A = 0, B = -10$	$A = -1, B = -1$

Nash Equilibrium

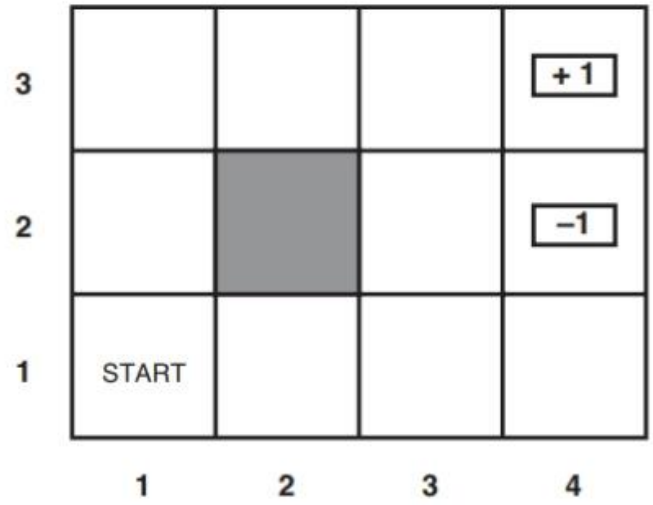
strategy. An equilibrium is essentially a **local optimum** in the space of policies; it is the top of a peak that slopes downward along every dimension, where a dimension corresponds to a player's strategy choices.

NASH EQUILIBRIUM

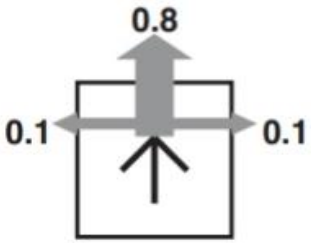
The mathematician John Nash (1928–) proved that *every game has at least one equilibrium*. The general concept of equilibrium is now called **Nash equilibrium** in his honor. Clearly, a dominant strategy equilibrium is a Nash equilibrium (Exercise 17.16), but some games have Nash equilibria but no dominant strategies.

The *dilemma* in the prisoner's dilemma is that the equilibrium outcome is worse for both players than the outcome they would get if they both refused to testify. In other words, $(testify, testify)$ is Pareto dominated by the (-1, -1) outcome of $(refuse, refuse)$. Is there any way for Alice and Bob to arrive at the (-1, -1) outcome? It is certainly an *allowable* option for both of them to refuse to testify, but is is hard to see how rational agents can get there, given the definition of the game. Either player contemplating playing *refuse* will realize that he or she would do better by playing *testify*. That is the attractive power of an equilibrium point. Game theorists agree that being a Nash equilibrium is a necessary condition for being a solution—although they disagree whether it is a sufficient condition.

Grid World



(a)



(b)

Figure 17.1 (a) A simple 4×3 environment that presents the agent with a sequential decision problem. (b) Illustration of the transition model of the environment: the "intended" outcome occurs with probability 0.8, but with probability 0.2 the agent moves at right angles to the intended direction. A collision with a wall results in no movement. The two terminal states have reward +1 and –1, respectively, and all other states have a reward of –0.04.

Example

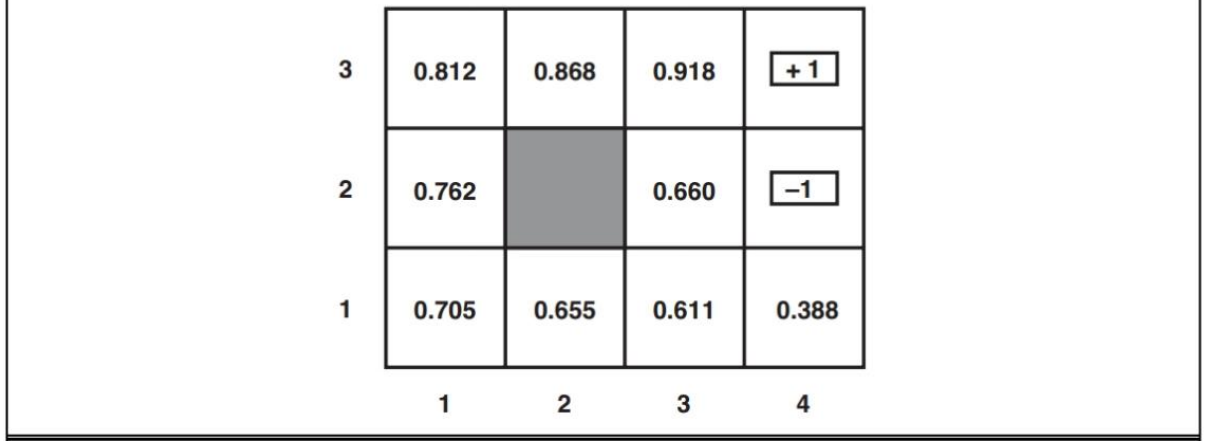


Figure 17.3 The utilities of the states in the 4×3 world, calculated with $\gamma = 1$ and $R(s) = -0.04$ for nonterminal states.

- Trying action up in state (1,1)
- P((1,2)|(1,1), up) = 0.8
- P((2,1)|(1,1), up) = 0.1
- P((1,1)|(1,1), up) = 0.1 , accidently going left

Q((1,1), up) = $0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1)$,

Example

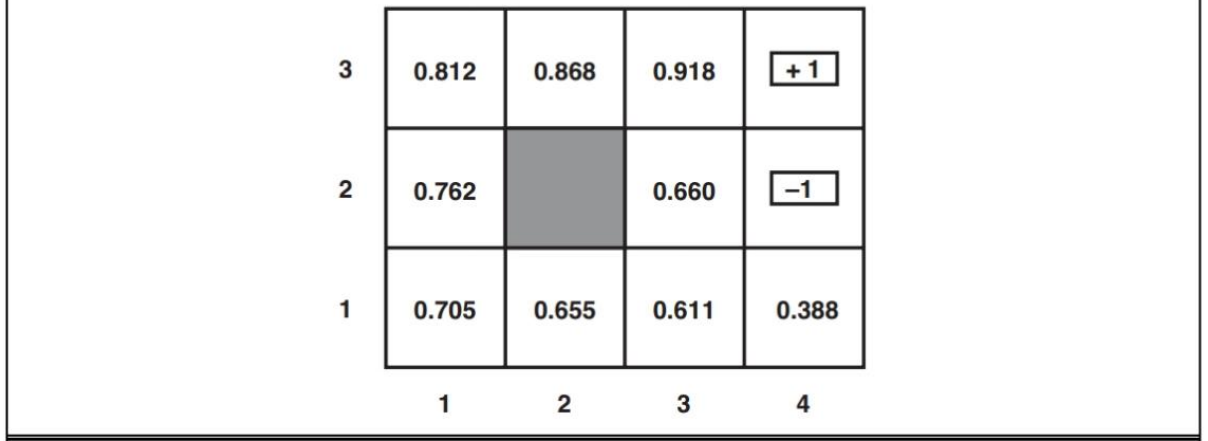


Figure 17.3 The utilities of the states in the 4×3 world, calculated with $\gamma = 1$ and $R(s) = -0.04$ for nonterminal states.

$$U(1,1) = -0.04 + \gamma \max[$$

(up) $0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1)$,

(left) $0.9U(1,1) + 0.1U(1,2)$,

(right) $0.9U(1,1) + 0.1U(2,1)$,

(down) $0.8U(2,1) + 0.1U(1,2) + 0.1U(1,1)$

]

Example

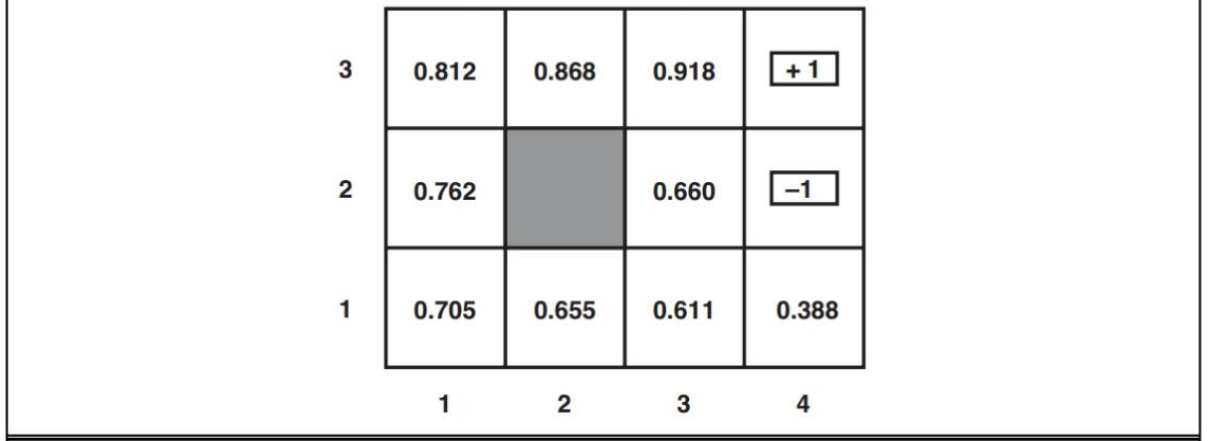


Figure 17.3 The utilities of the states in the 4×3 world, calculated with $\gamma = 1$ and $R(s) = -0.04$ for nonterminal states.

$$U(1,1) = -0.04 + \gamma \max[$$

(up)0.7456,

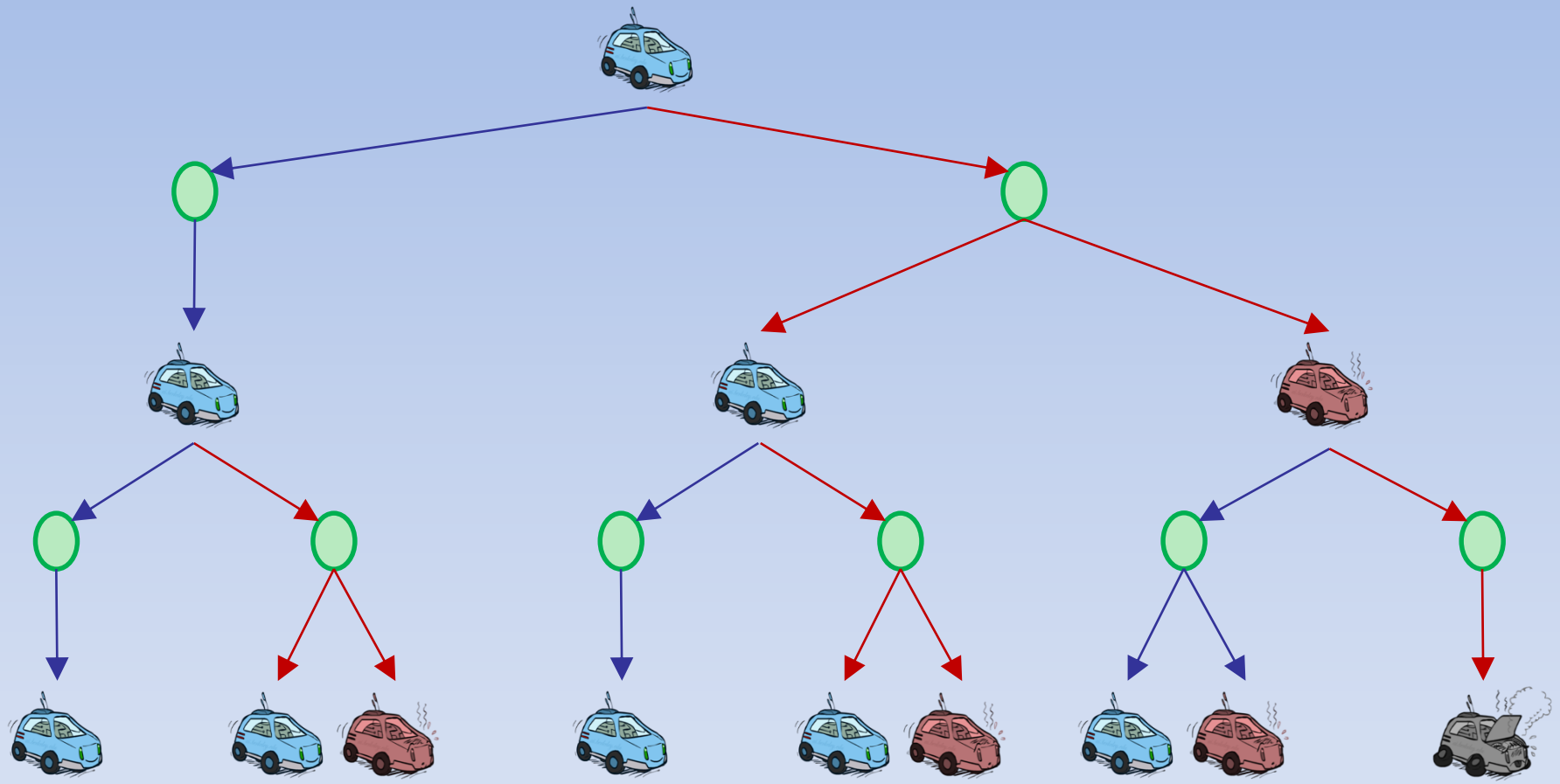
(left) 0.7107,

(right)0.7

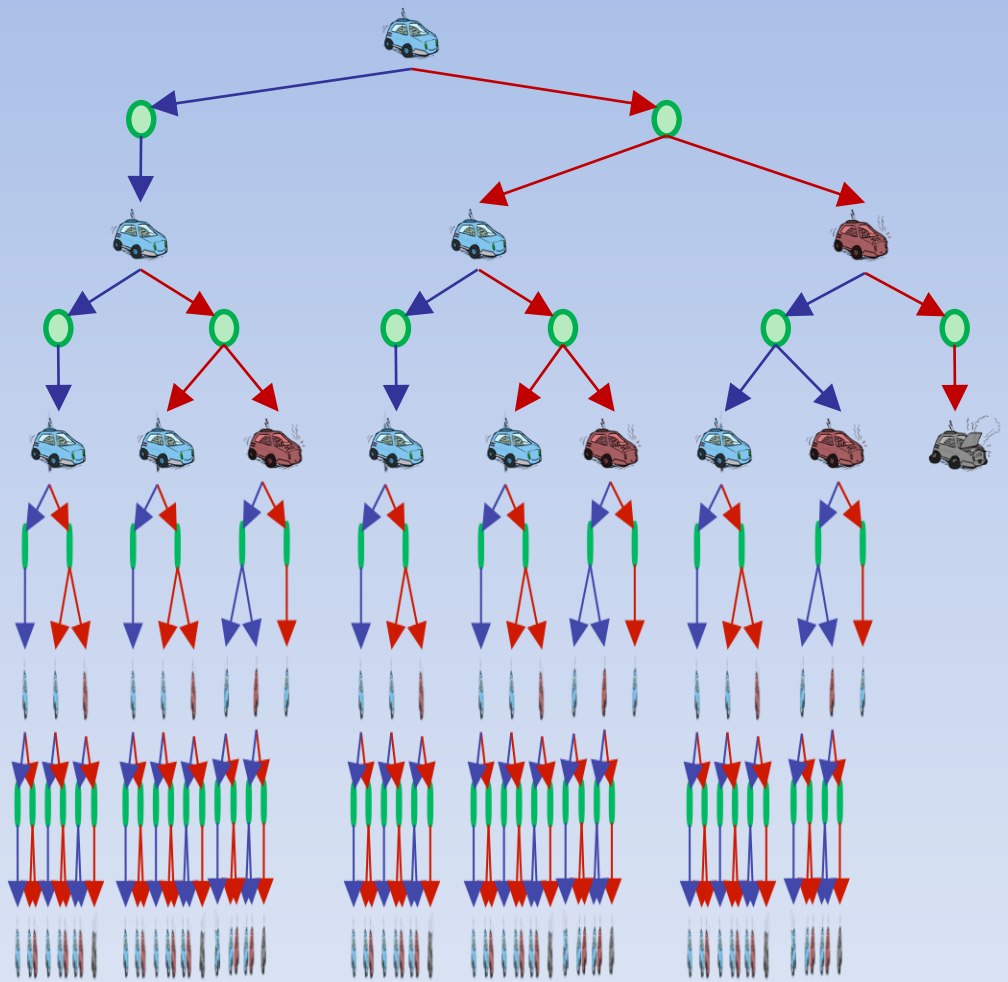
(down)0.6707

]

Racing Search Tree w/ Expectimax

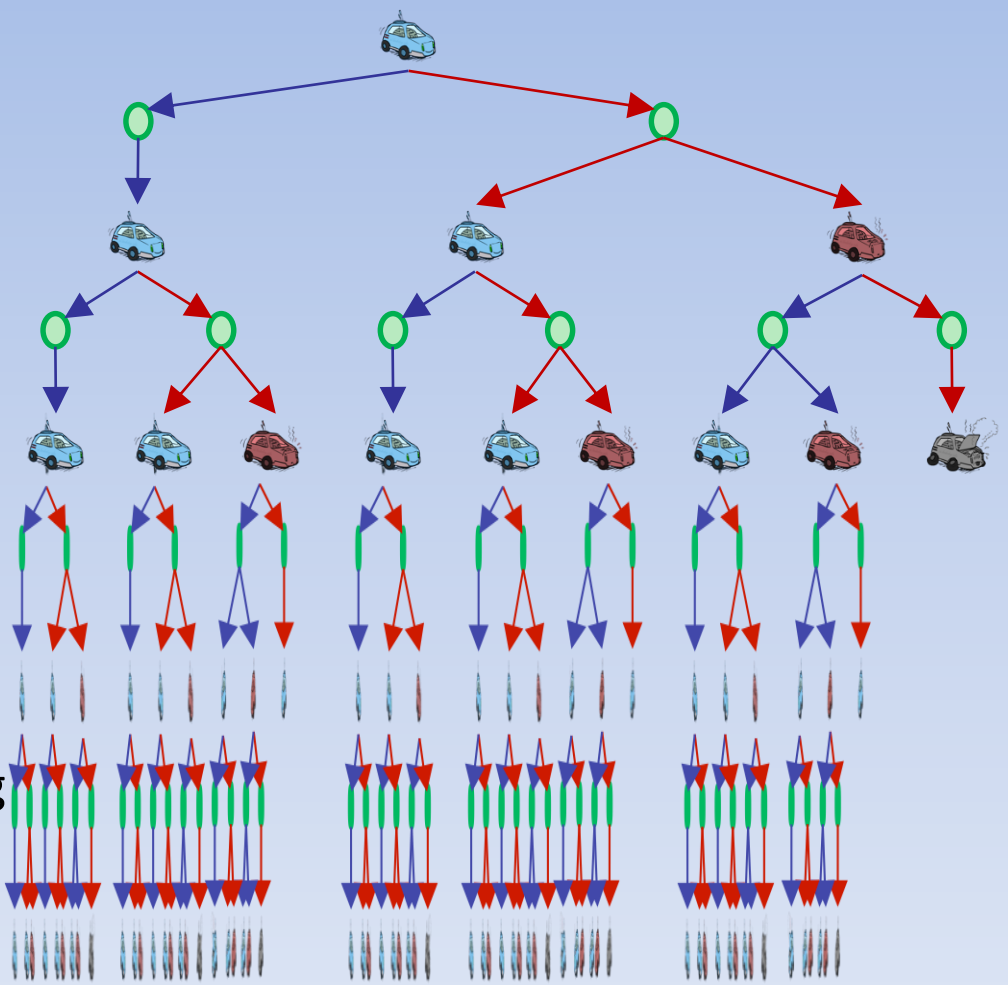


Racing Search Tree w/ Expectimax



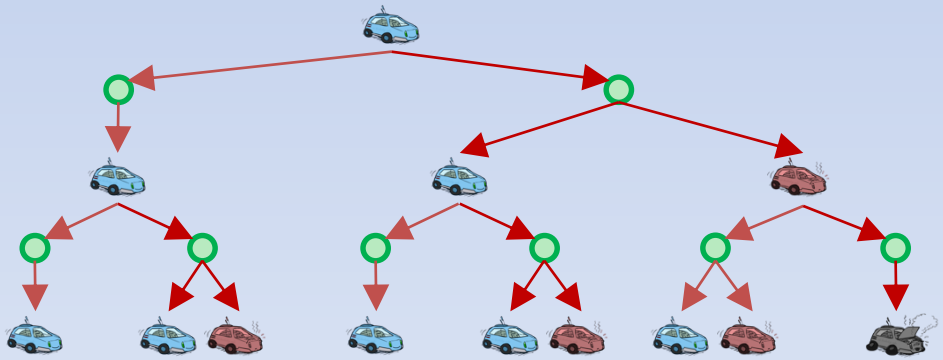
Racing Search Tree w/ Expectimax

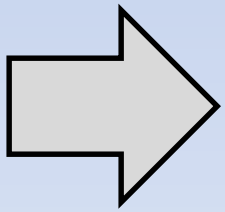
- We're doing way too much work with expectimax!
  - Like Graph search
  - Same States Repeated
- Problem: States are repeated
  - Idea: Only compute needed quantities once
  - Dynamic Programming
- Problem: Tree goes on forever
  - Idea: Do a depth-limited computation, but with increasing depths until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

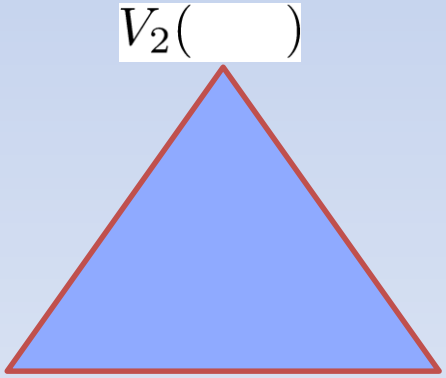


Time-Limited Values

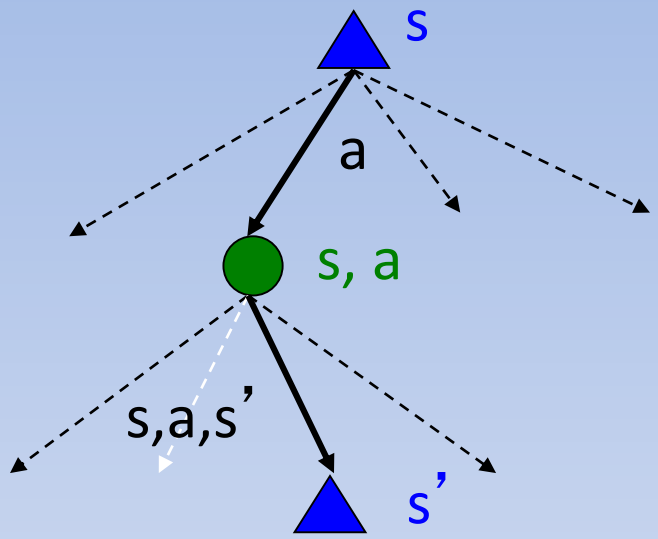
- Key idea: time-limited values
- Define $\mathbf{V_k(s)}$ to be the optimal value of s if the game ends in k more time steps
  - Equivalently, it's what a depth-k expectimax would give from s







Bellman Equation



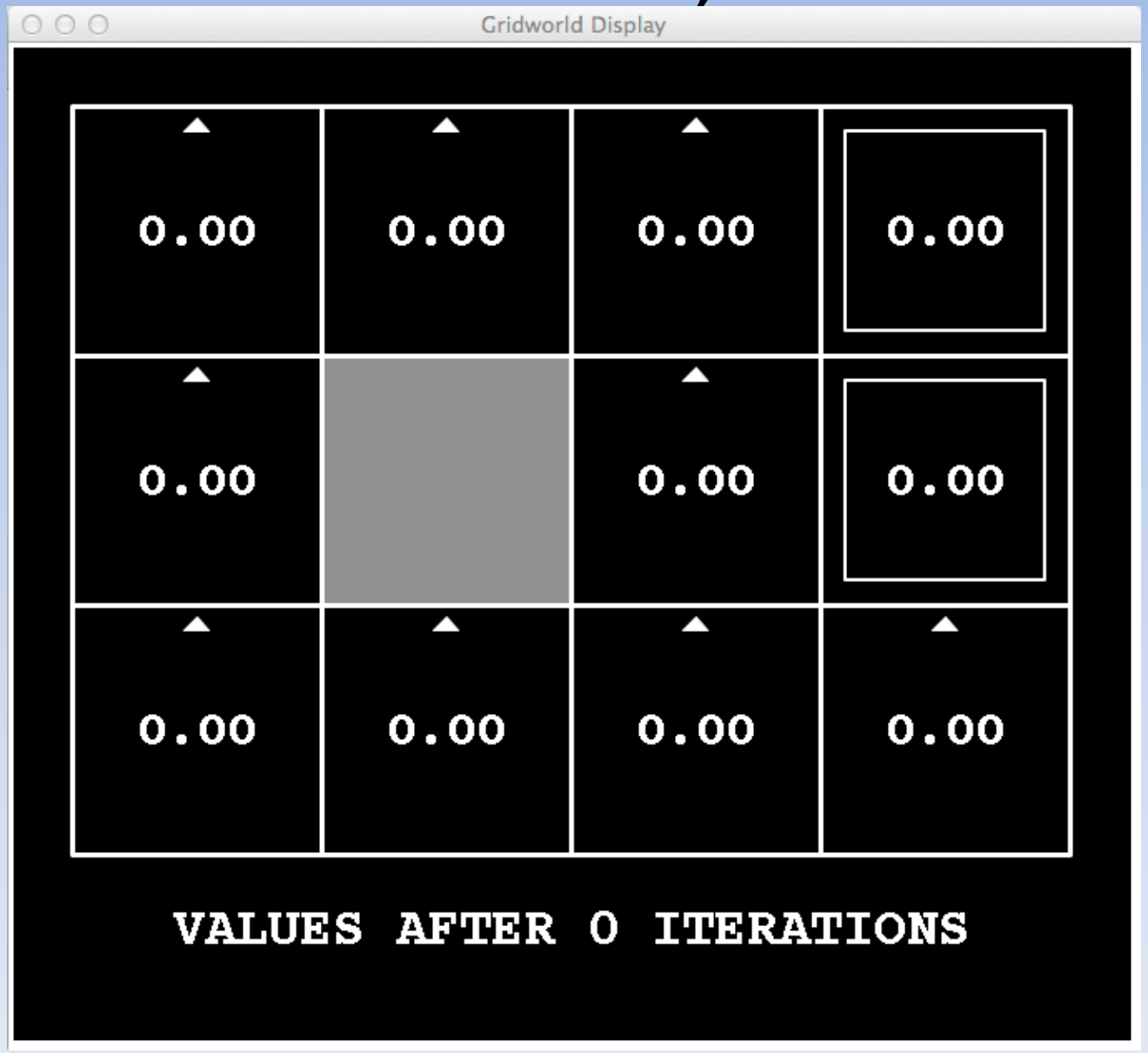
$$V^*(s) = \max_a Q^*(s,a)$$

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

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

- Alternative with R(s, a, s')
- T(s, a, s') = P(s' | s, a)
- Rewards are given when action is taken.

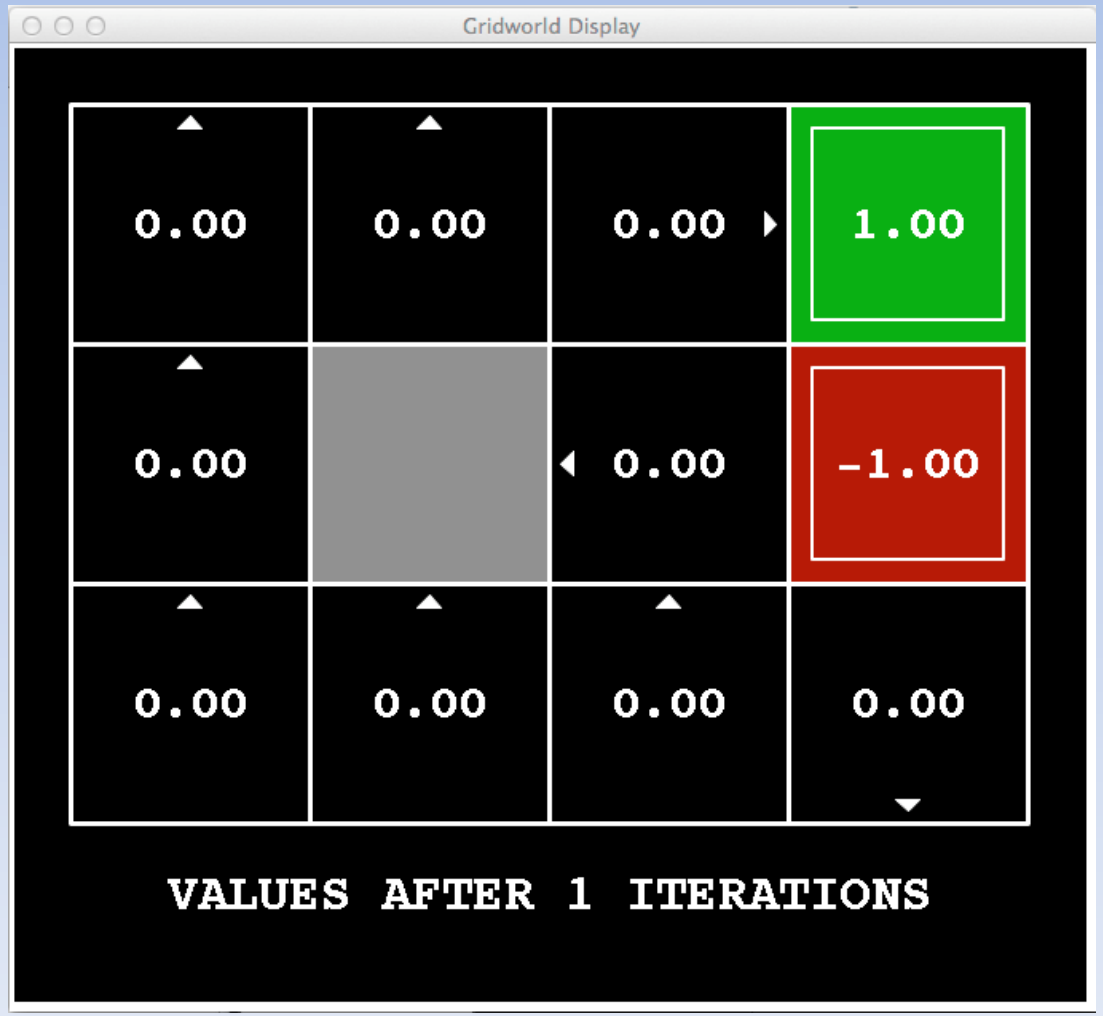
K=0, 0 moves



Discount = 0.9
Living reward = 0

K=1, 1 move

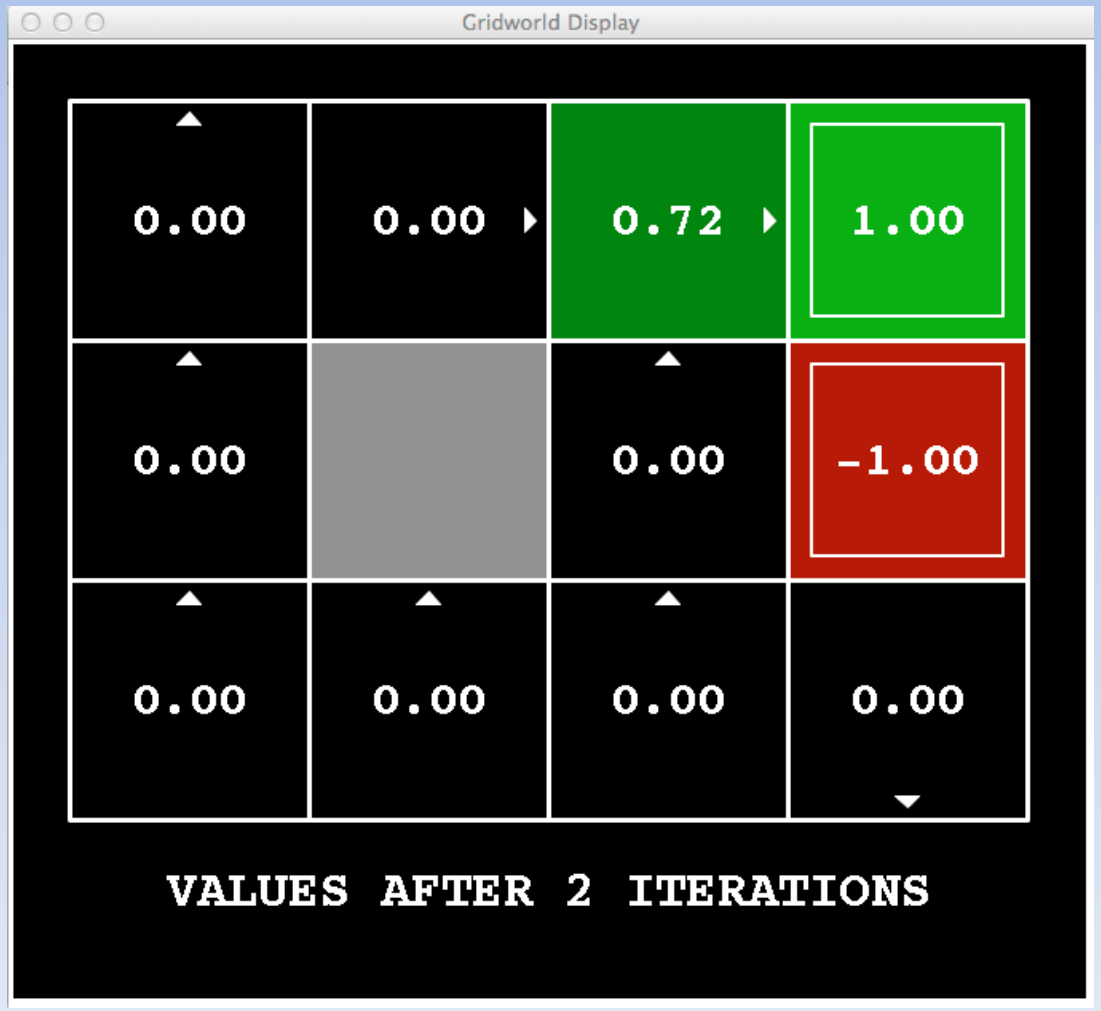
$$V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s') \right]$$



Discount = 0.9
Living reward = 0

K=2, 2 moves

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



Discount = 0.9
Living reward = 0

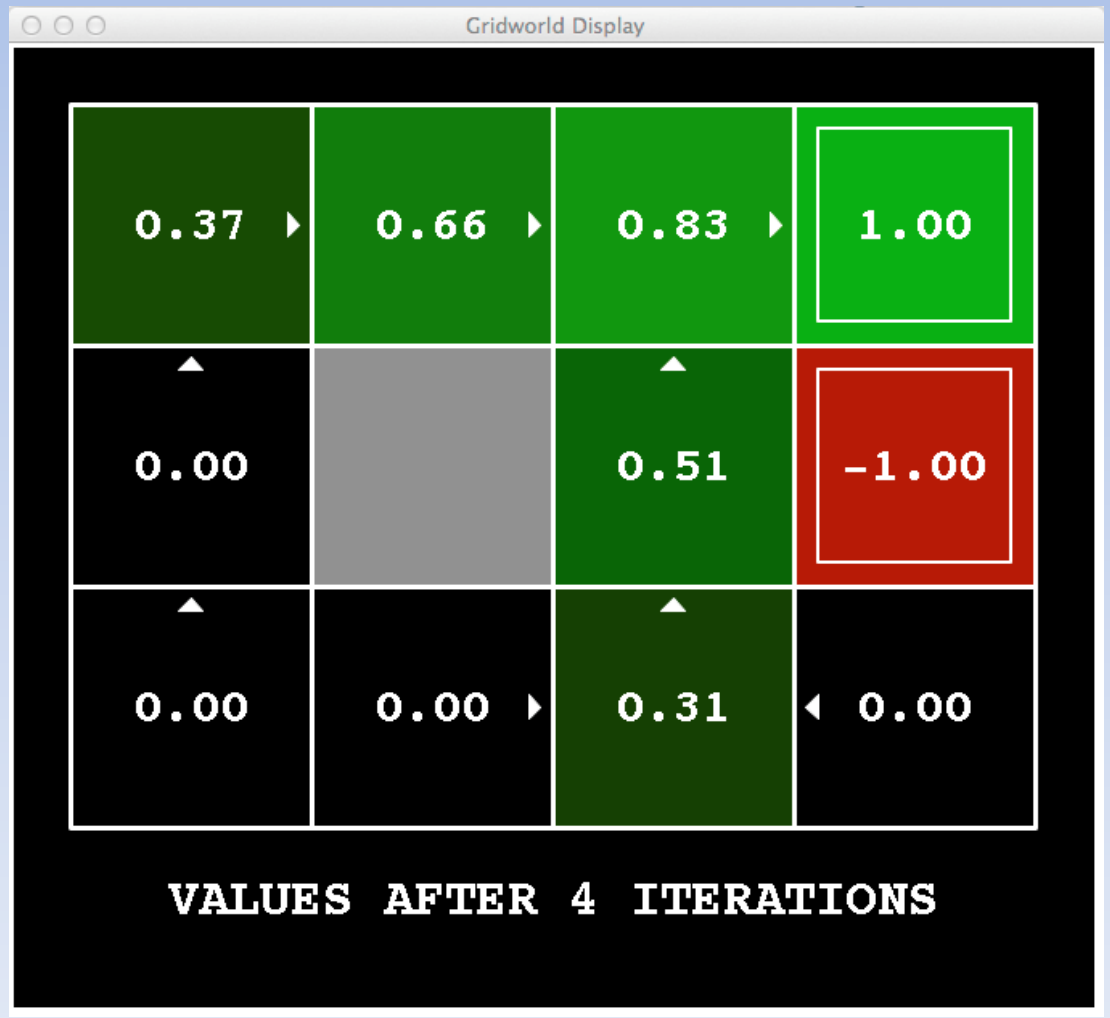
K=3

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

Discount = 0.9
Living reward = 0

K=4

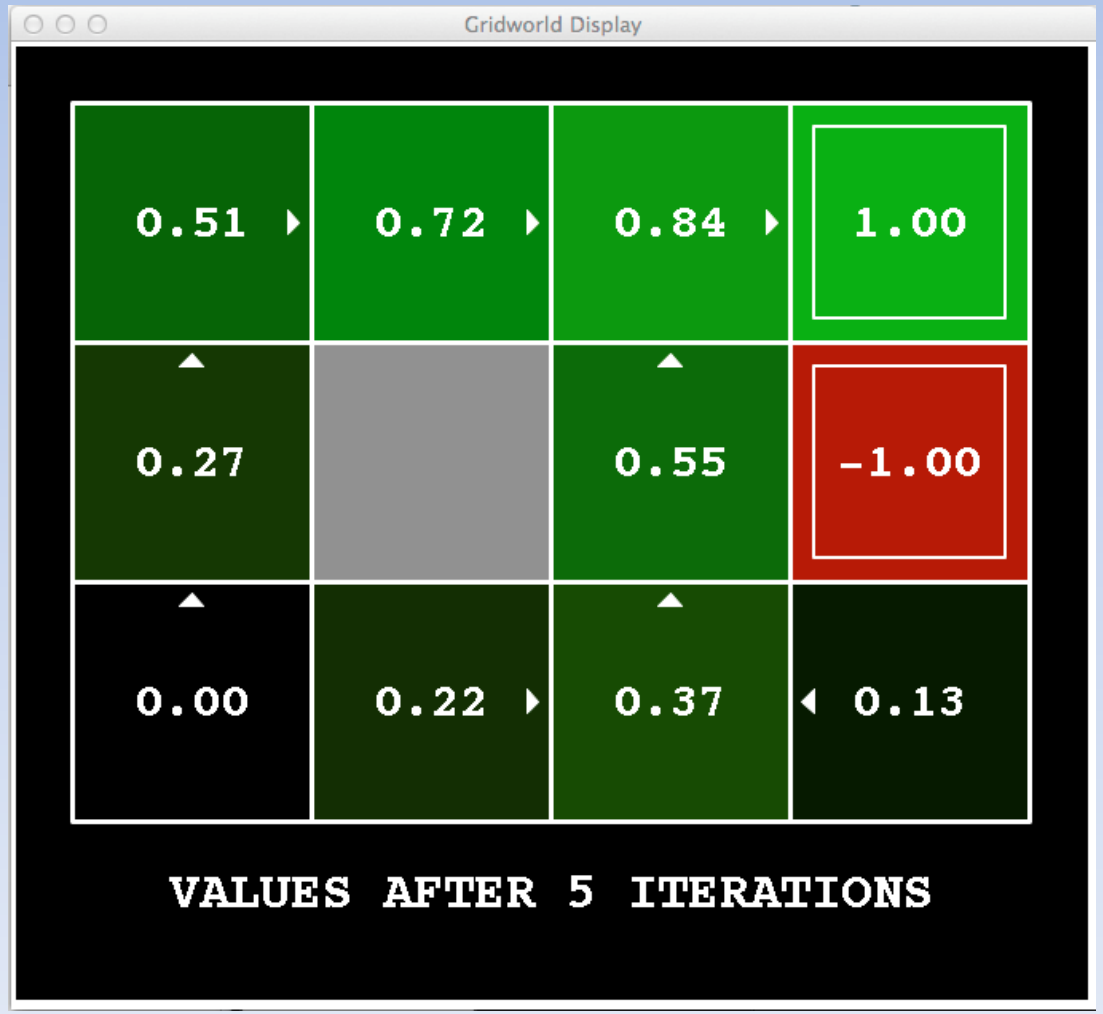
$$V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s') \right]$$



Discount = 0.9
Living reward = 0

K=5

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



Discount = 0.9
Living reward = 0

K=6

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

Discount = 0.9
Living reward = 0

K=7

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

Discount = 0.9
Living reward = 0

K=8

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

Discount = 0.9
Living reward = 0

K=9

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

Discount = 0.9
Living reward = 0

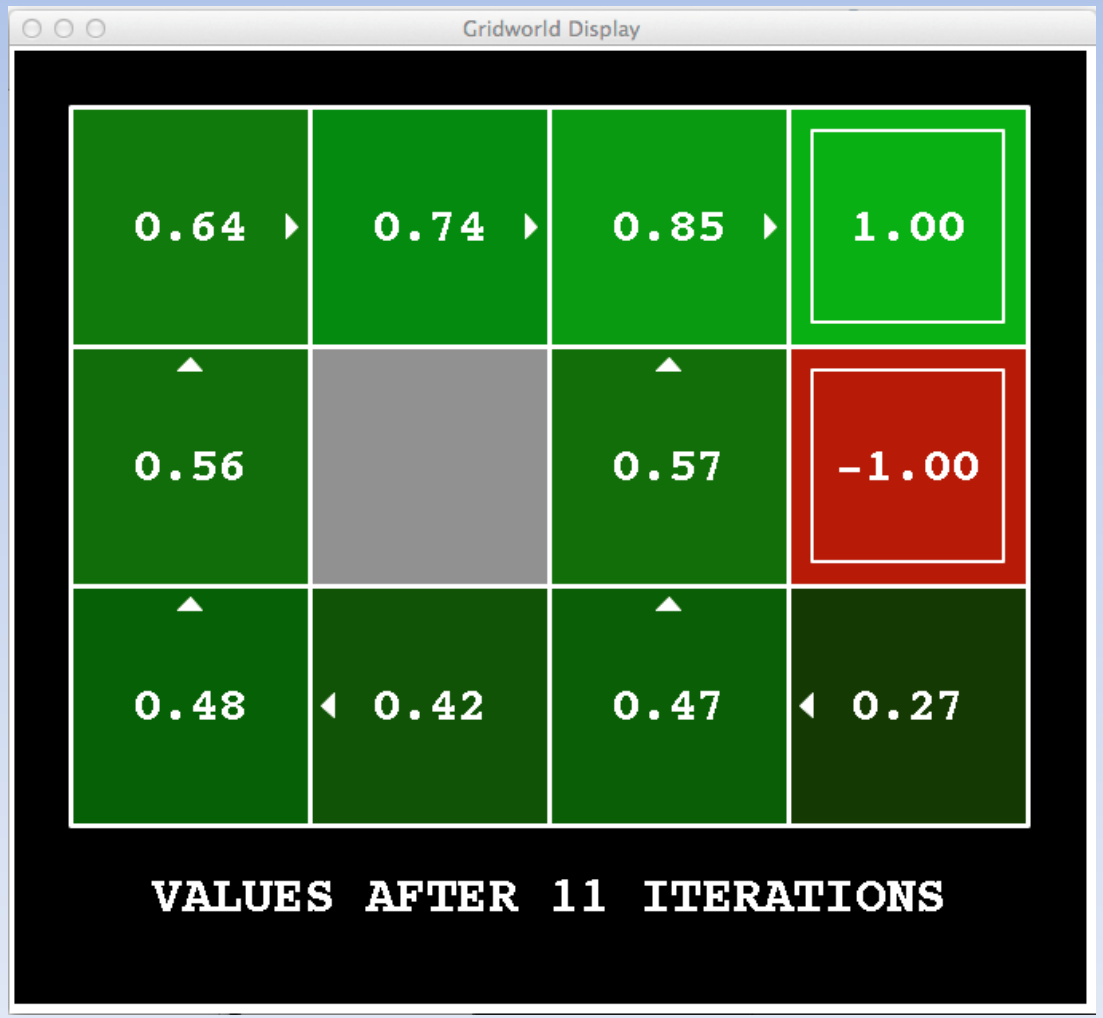
K=10

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

Discount = 0.9
Living reward = 0

K=11

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



Discount = 0.9
Living reward = 0

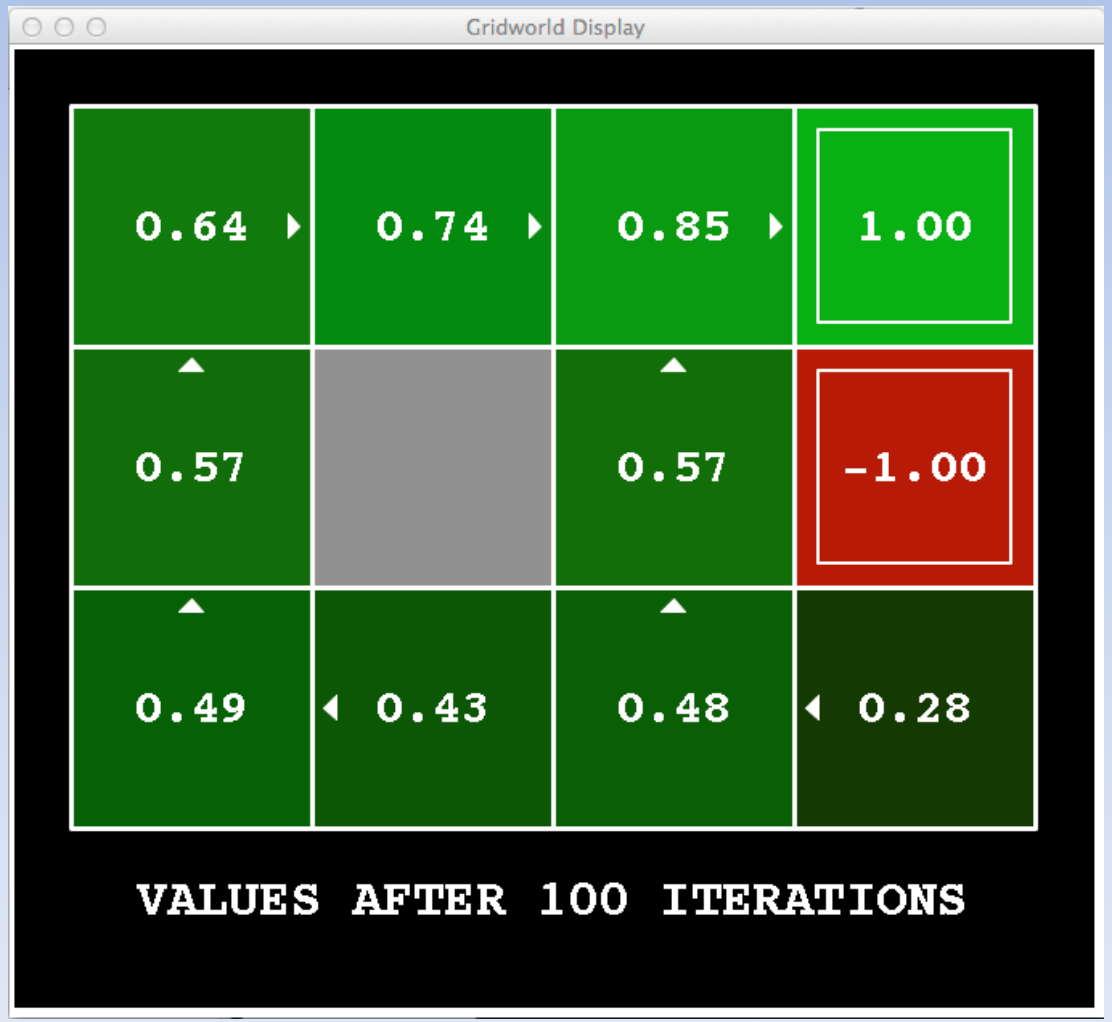
K=12

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

Discount = 0.9
Living reward = 0

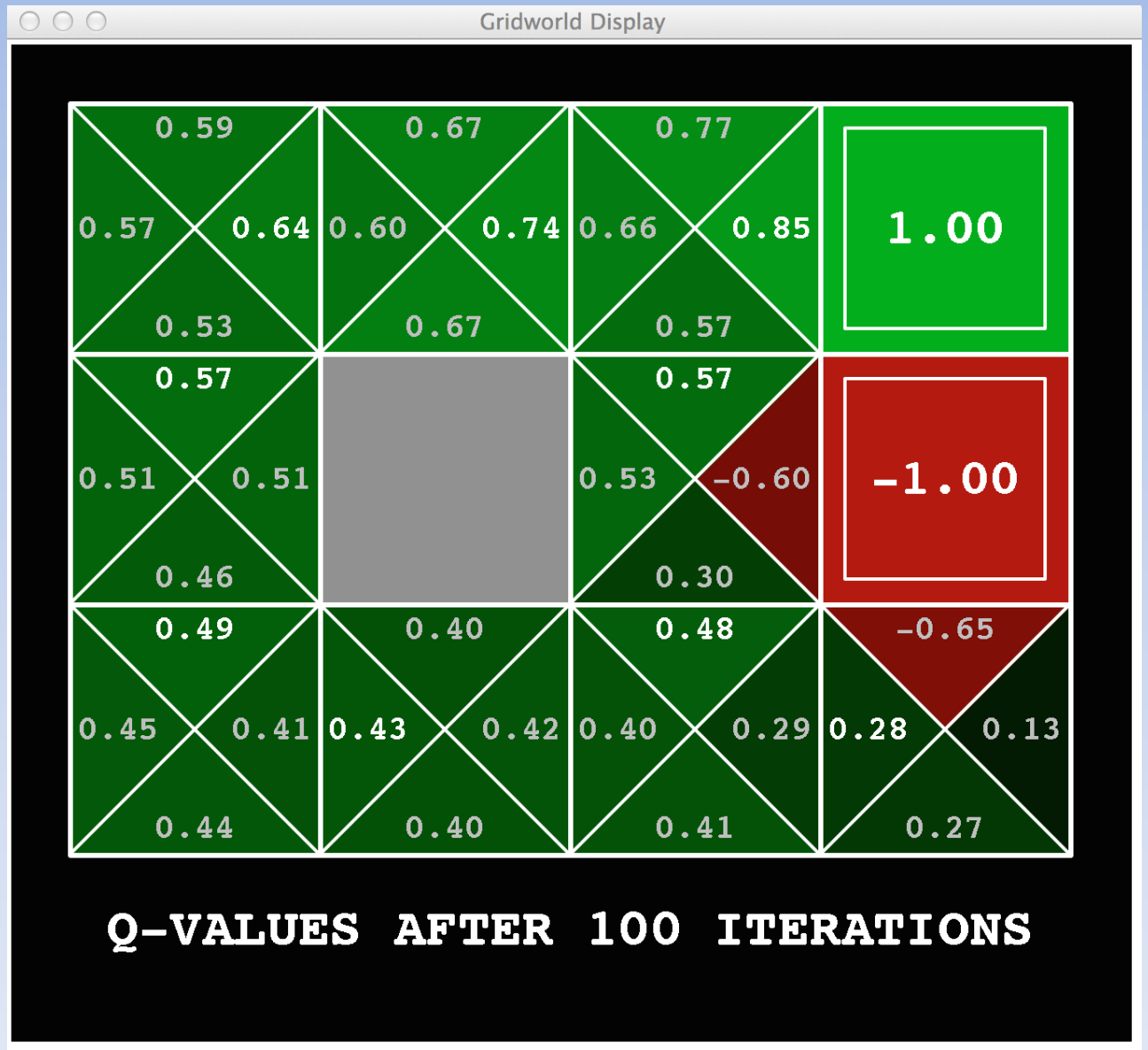
K=100

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

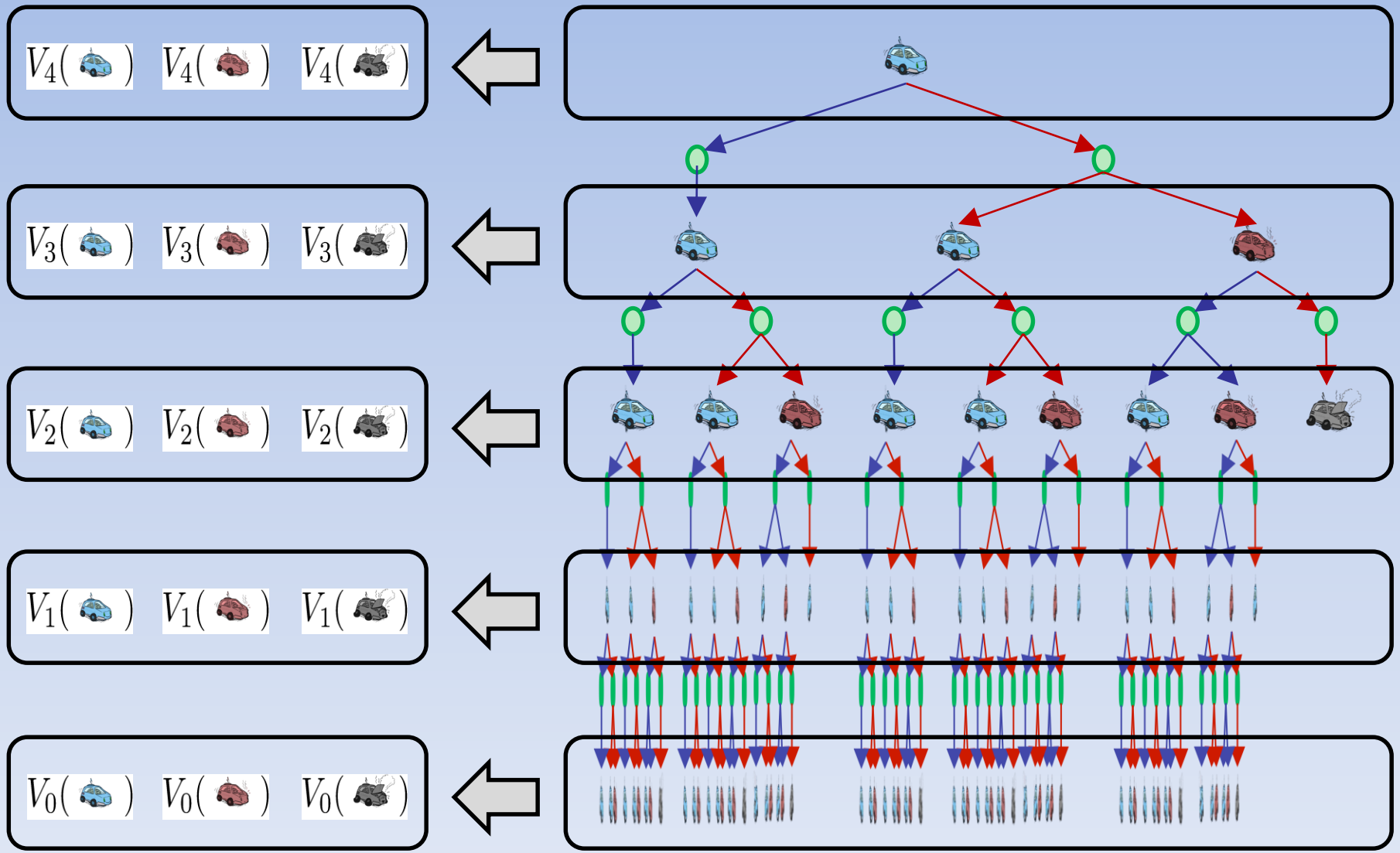


Discount = 0.9
Living reward = 0

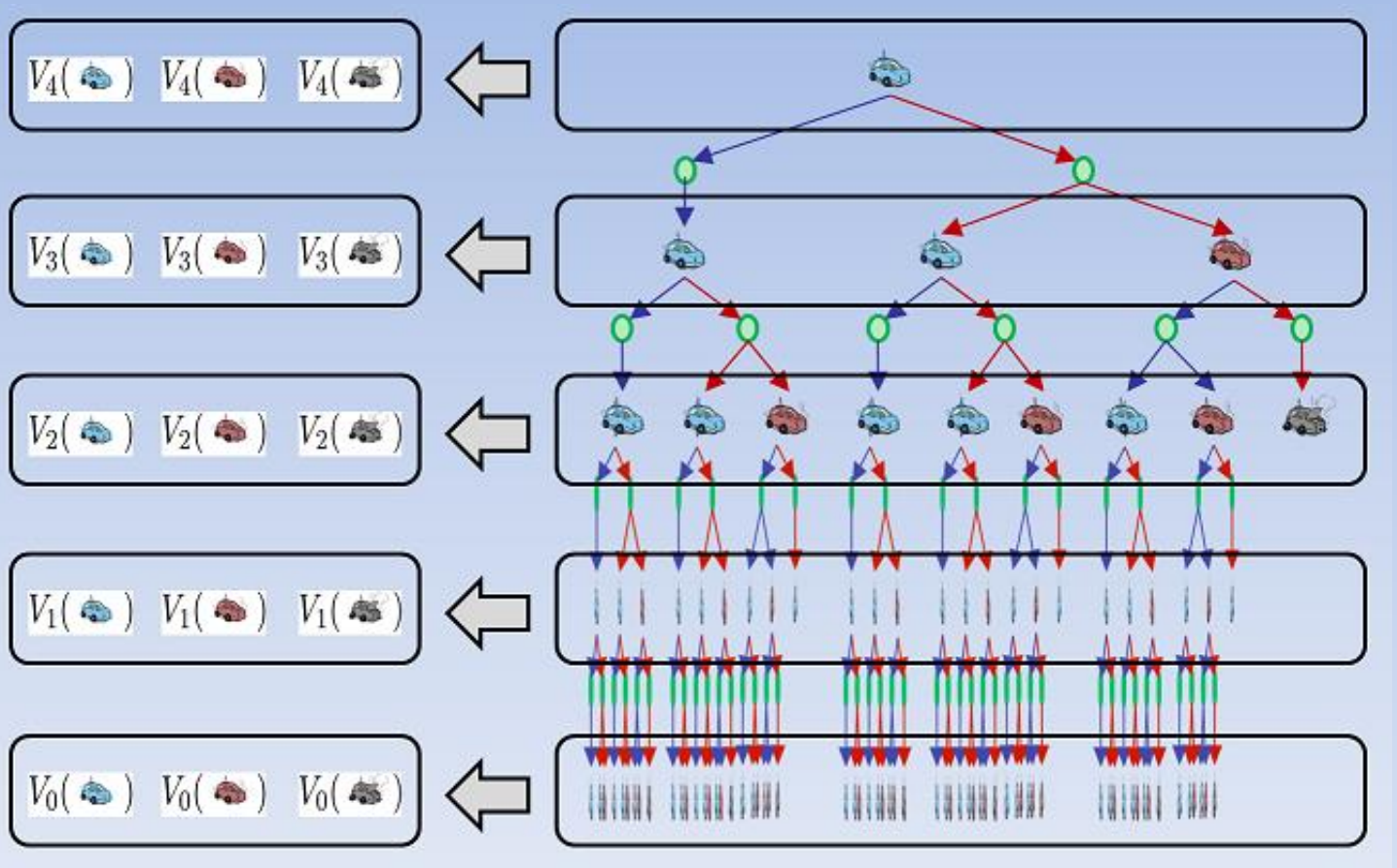
Q*-Values


Gridworld Display
0.59
0.57
0.64
0.53
0.67
0.60
0.74
0.67
0.77
0.66
0.85
0.57
1.00
0.57
0.51
0.51
0.46
0.57
0.53
-0.60
0.30
-1.00
0.49
0.45
0.41
0.44
0.40
0.43
0.42
0.40
0.48
0.40
0.29
0.41
-0.65
0.28
0.13
0.27
Q-VALUES AFTER 100 ITERATIONS

Computing Time-Limited Values

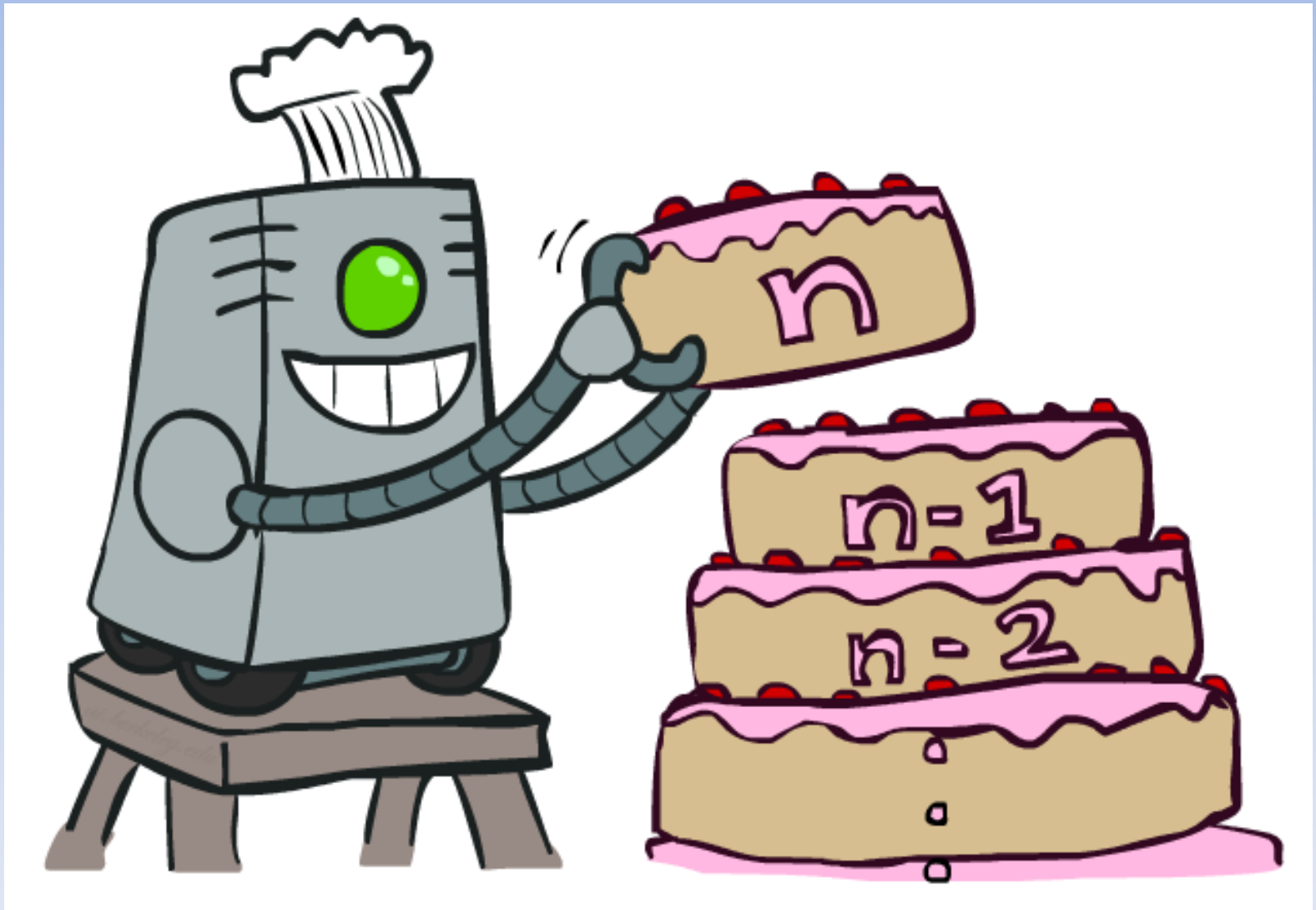


Dynamic Programming Idea



- Cache state values V_k
- Compute V_{k+1} using V_k
- Initialize with V_0

Value Iteration

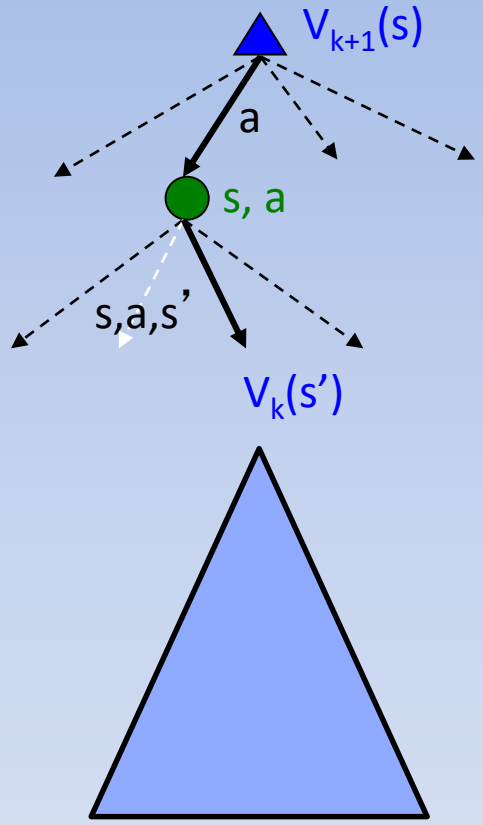


Value Iteration

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

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

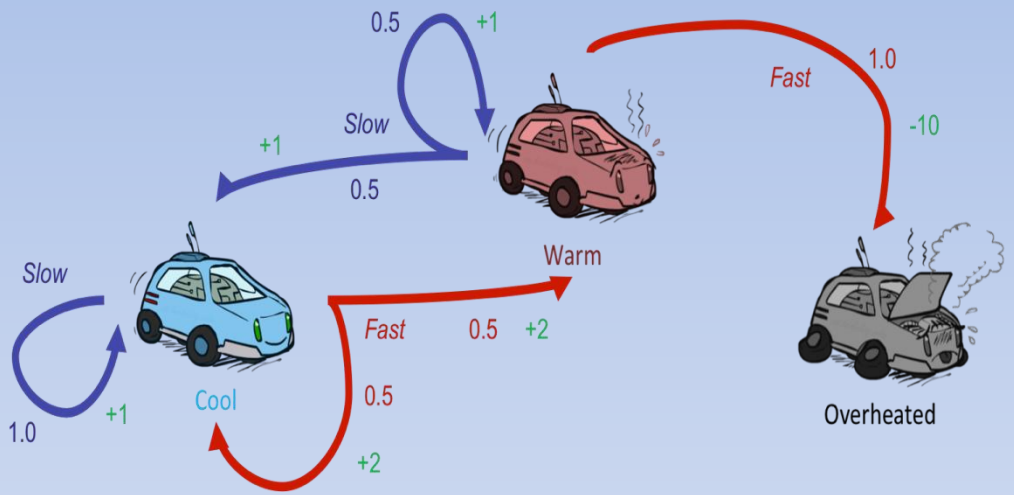
- Repeat until convergence



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

Example: Value Iteration

V_2	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 w/ Textbook R(s)

function VALUE-ITERATION(mdp, ϵ) **returns** a utility function
  inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$,
    rewards $R(s)$, discount γ
    ϵ, the maximum error allowed in the utility of any state
  local variables: U, U', vectors of utilities for states in S, initially zero
    δ, the maximum change in the utility of any state in an iteration

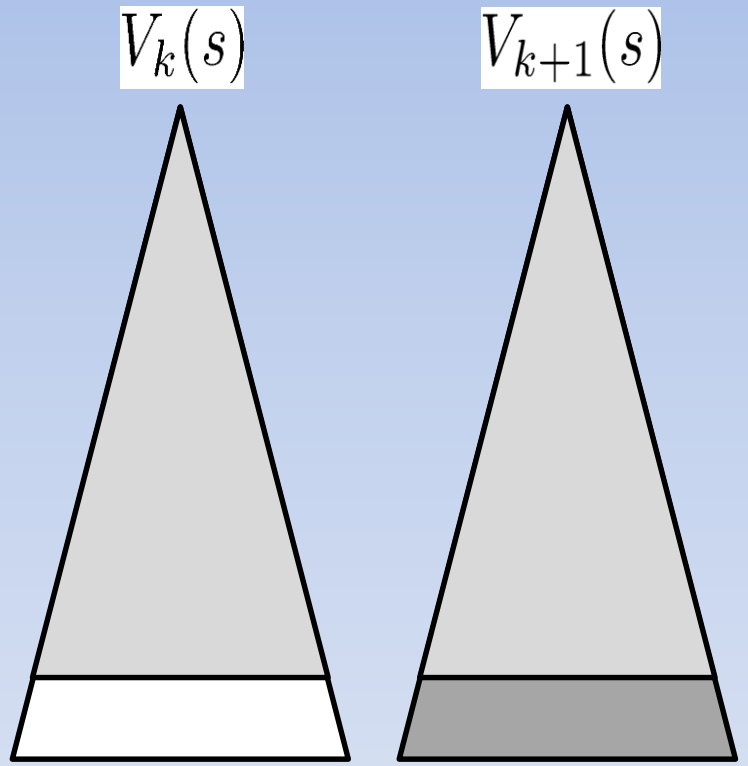
  repeat
    $U \leftarrow U'$; $\delta \leftarrow 0$
    for each state s **in** S **do**
      $U'[s] \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\, U[s']$
      if $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$
  until $\delta < \epsilon(1-\gamma)/\gamma$
  return U

Figure 17.4 The value iteration algorithm for calculating utilities of states. The termination condition is from Equation (17.8).

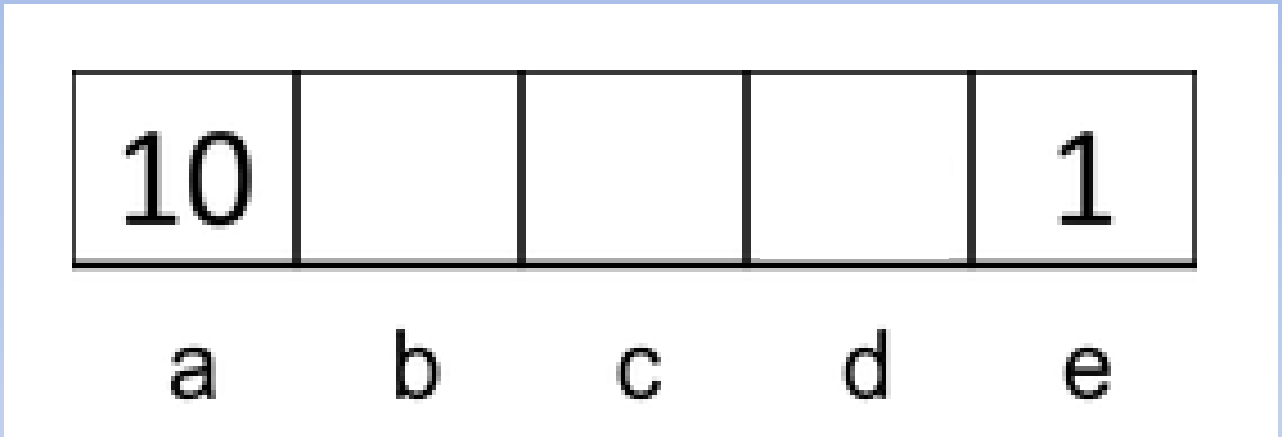
Convergence*

- 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

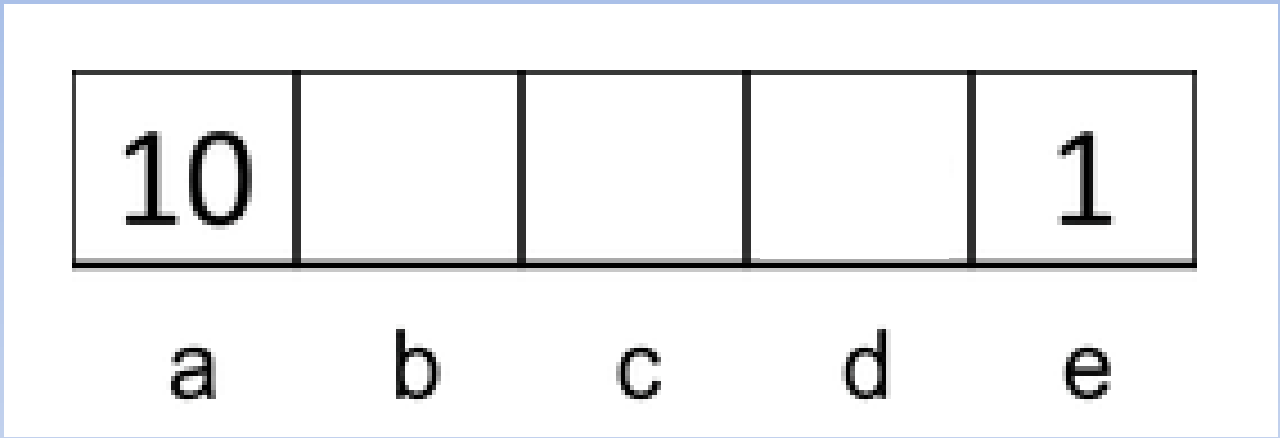


Discounting + Stochastic Element



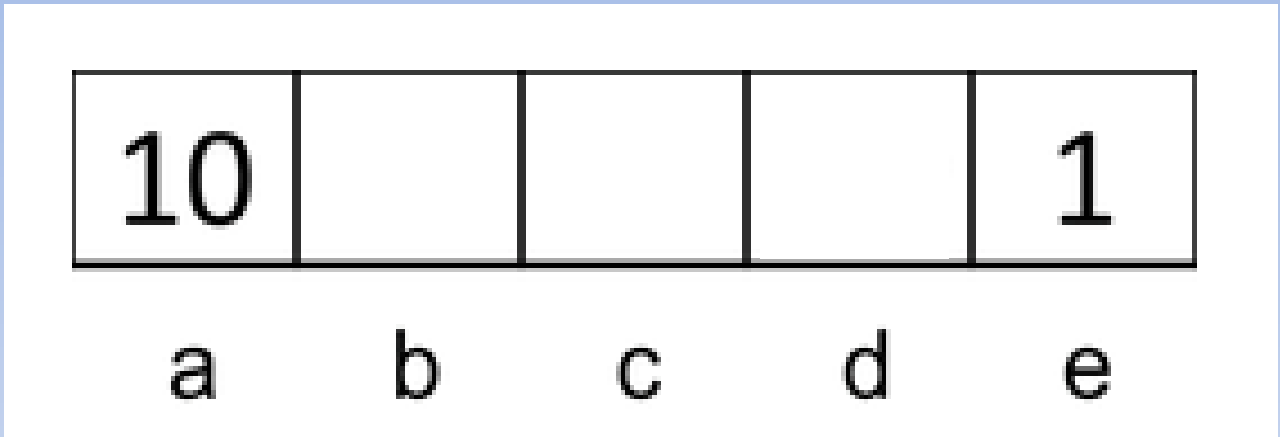
- Our World:
  - 5 State = {a, b, c, d, e}
  - 3 Actions = {Left, Right, Exit}
- **Transitions are no longer deterministic**
- Left and Right are successful 80% of the time.
  - When not success, the agent stays in place.
- Exit is successful 100% of the time
- State = {a, b, c, d, e}
- $\gamma = 0.2$
- Calculate V*(state) = V_∞(state)

Discounting
+ Stochastic Element



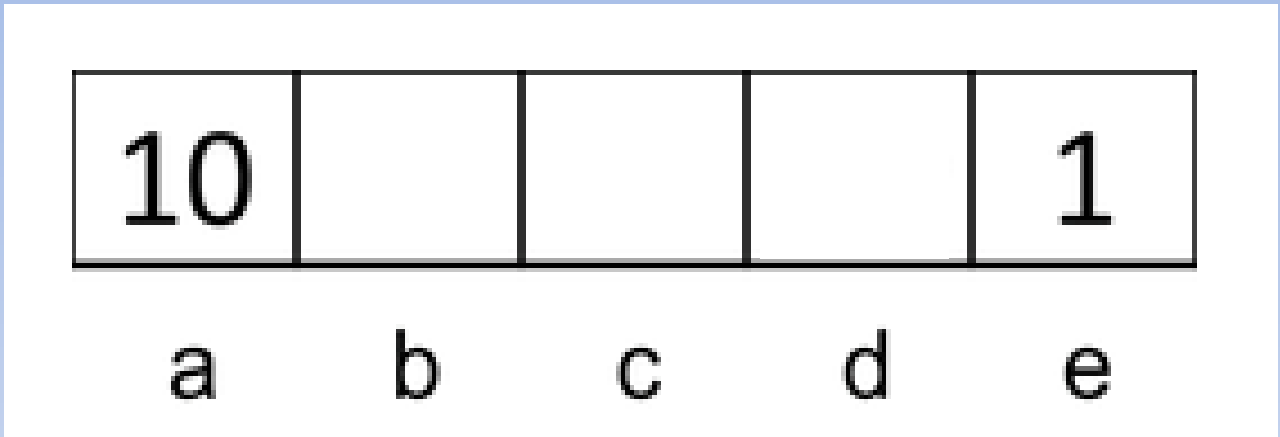
- V*(a) =

Discounting
+ Stochastic Element



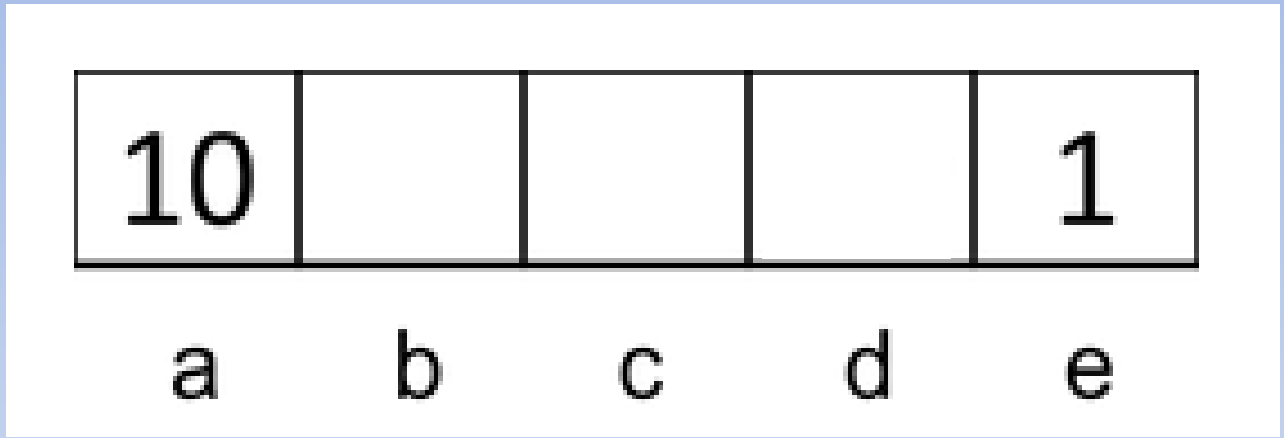
- $V_0(a) = 0$
- $V_0(b) = 0$
- $V_0(c) = 0$
- $V_0(d) = 0$
- $V_0(e) = 0$

Discounting
+ Stochastic Element



- $V_\infty(a) = 10$
- $V_\infty(e) = 1$

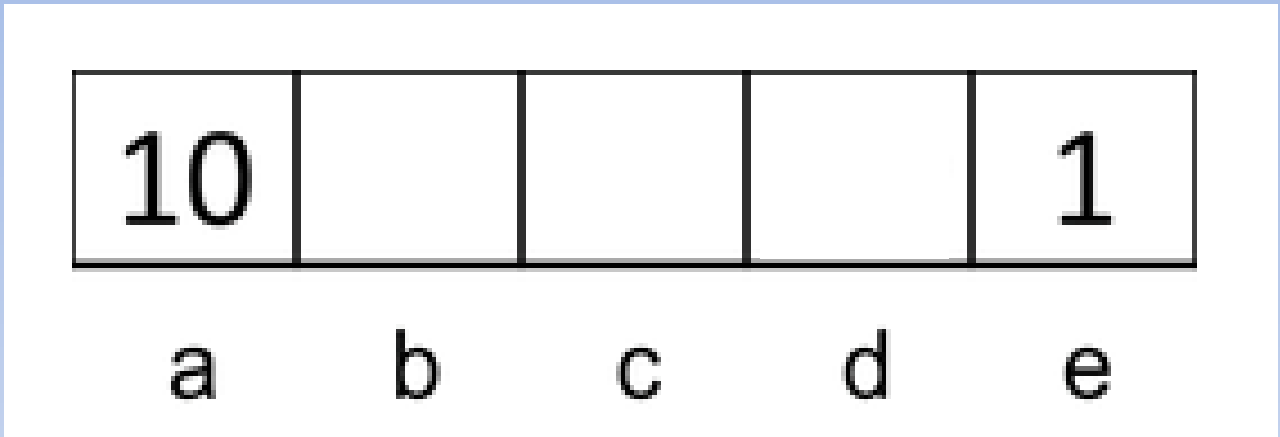
Discounting
+ Stochastic Element



$$U_{k+1}(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a) U_k(s')$$

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

Discounting
+ Stochastic Element



- $V_\infty(a) = 10$
- $V_\infty(b) = 1.67$
- $V_\infty(c) = 0.278$
- $V_\infty(d) = 0.167$
- $V_\infty(e) = 1$

- http://bit.ly/c166f19vi
- https://colab.research.google.com/drive/1gmxgDdSsdf3Eh3aWgcGMOMswLTQHQLEI

- The following MDP world consists of 5 states and 3 actions:

(1, 1) Actions: down, right	(1, 2) Action: Exit = -10
(2, 1) Actions: down, right	(2, 2) Action: Exit = -10
(3, 1) Action: Exit = 10	

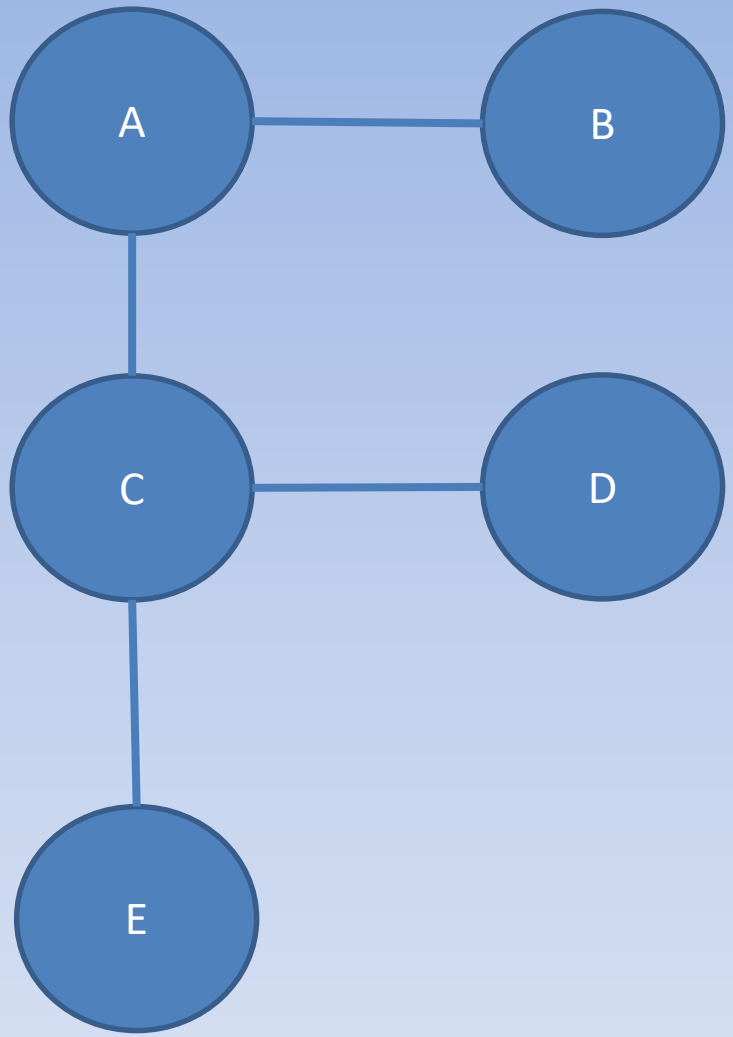
- When taking action down, it is successful with probability 0.6, otherwise you go right.
- When taking action right, it is successful with probability 0.6, otherwise you go down.
- When taking action Exit, it is successful with probability 1.0.
- The only reward is when taking action Exit, and there is no discounting.

- Calculate the value of states using Value Iteration algorithm showing work for time step K=0, 1, 2

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

$V_0(1, 1)=$ $V_1(1, 1)=$ $V_2(1, 1)=$	$V_0(1, 2)=$ $V_1(1, 2)=$ $V_2(1, 2)=$
$V_0(2, 1)=$ $V_1(2, 1)=$ $V_2(2, 1)=$	$V_0(2, 2)=$ $V_1(2, 2)=$ $V_2(2, 2)=$
$V_0(3, 1)=$ $V_1(3, 1)=$ $V_2(3, 1)=$	

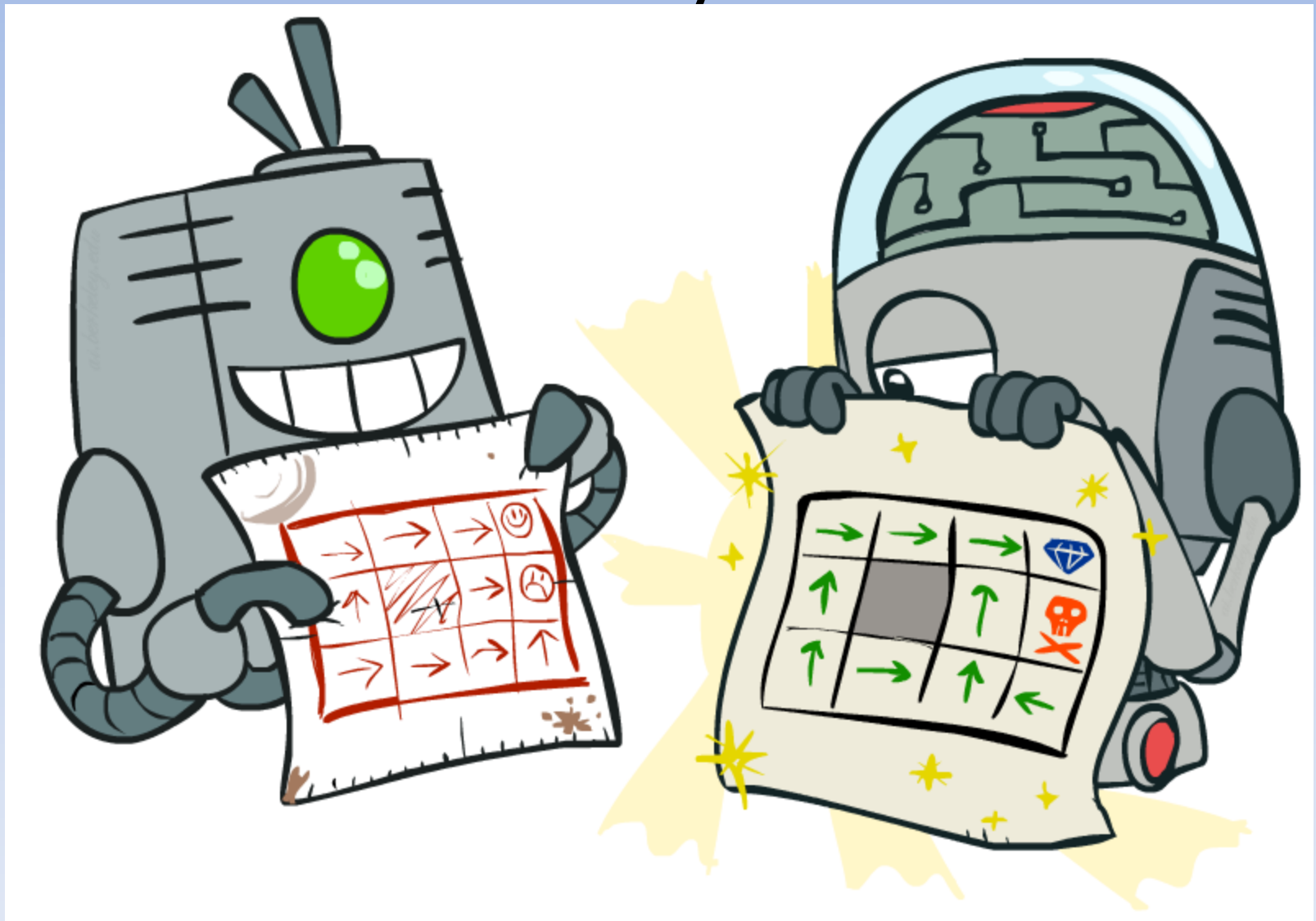
Graph



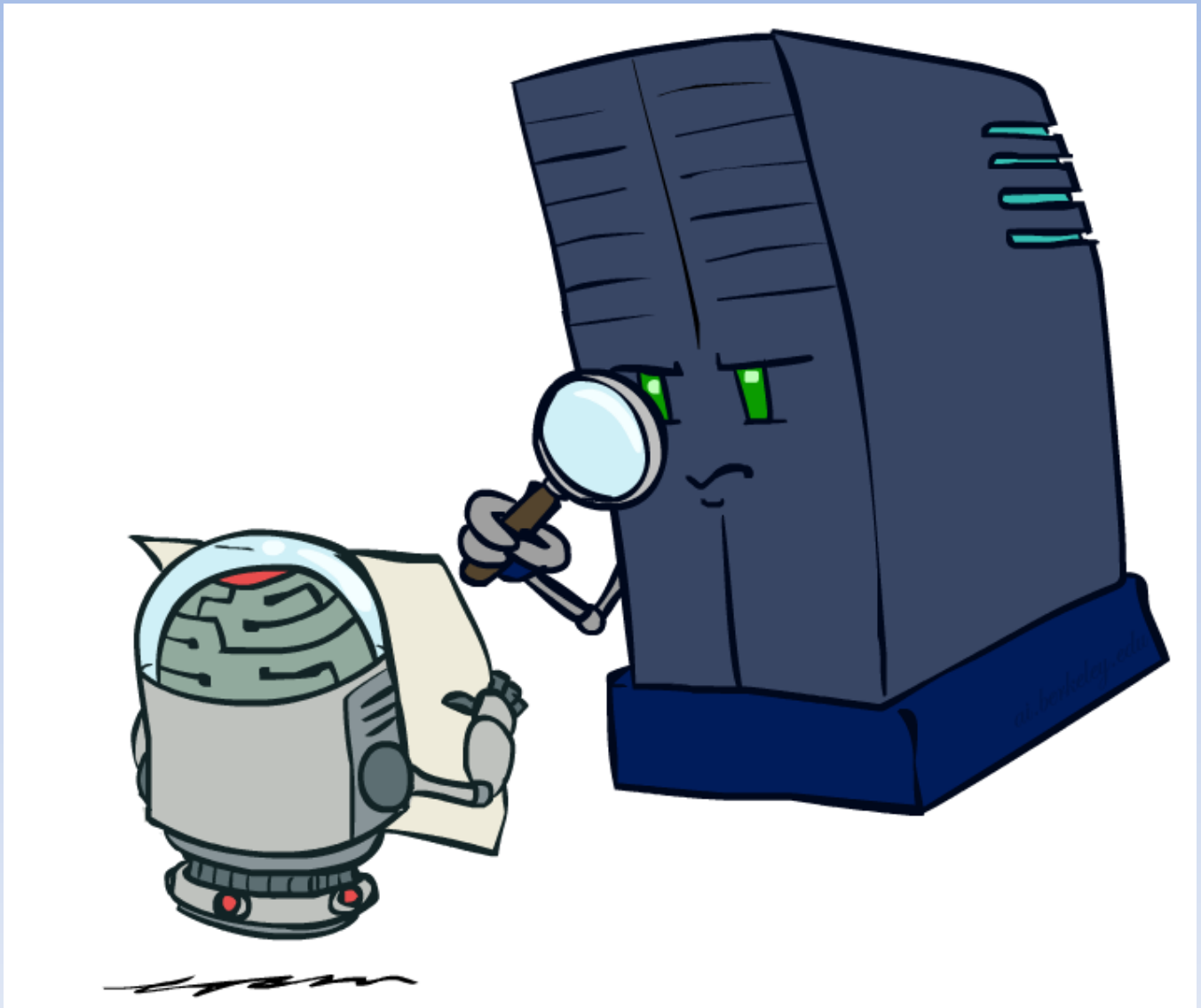
- R(B)=-1
- R(D)= -10
- R(E) = 100
- R(A) = 0
- R(C) = 0

- $U_{k+1}(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a) U_k(s')$

Policy Methods: Which Policy is Better?

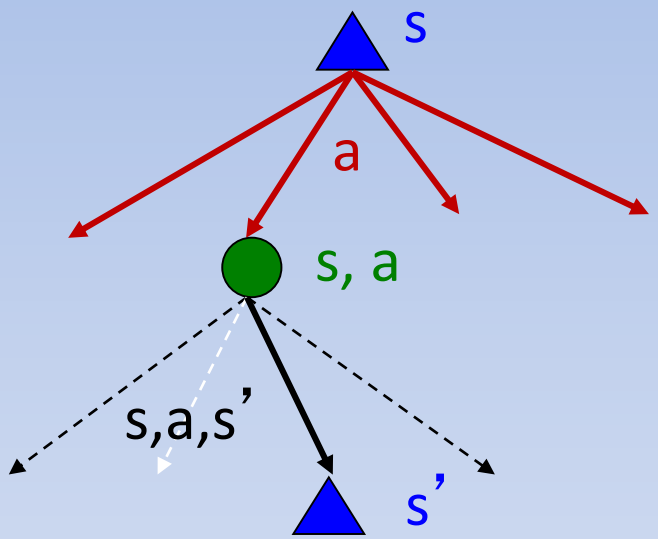


Policy Evaluation

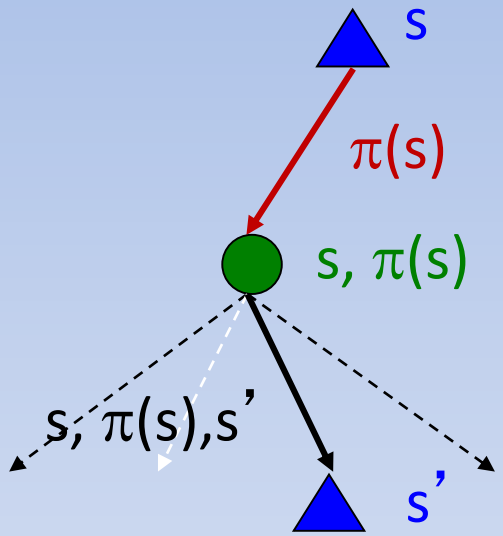


Fixed Policies

Do the optimal action



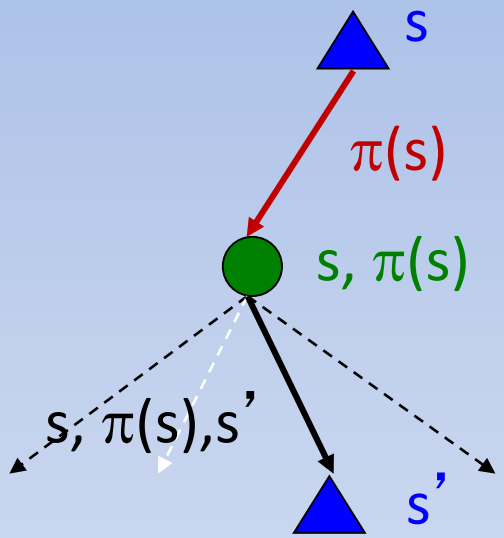
Do what π says to do



- Expectimax trees max over all actions to compute the optimal values
- If we fixed some policy $\pi(s)$, then the tree would be simpler – only one action per state
  - … though the tree's value would depend on which policy we fixed

Utilities for a Fixed Policy

- Another basic operation: compute the utility of a state s under a fixed (generally non-optimal) policy



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

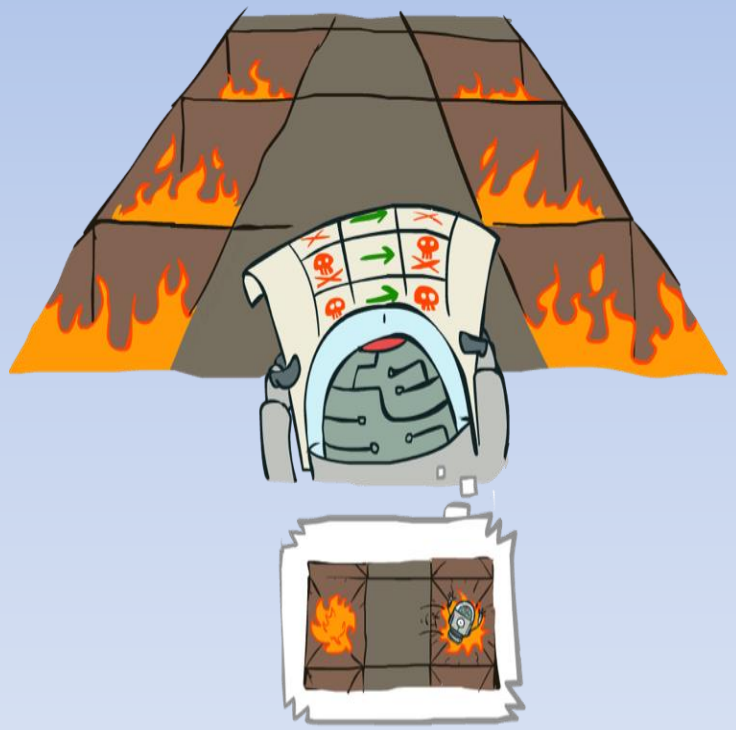
  $V^{\pi}(s)$ = expected total discounted rewards starting in s and following π

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

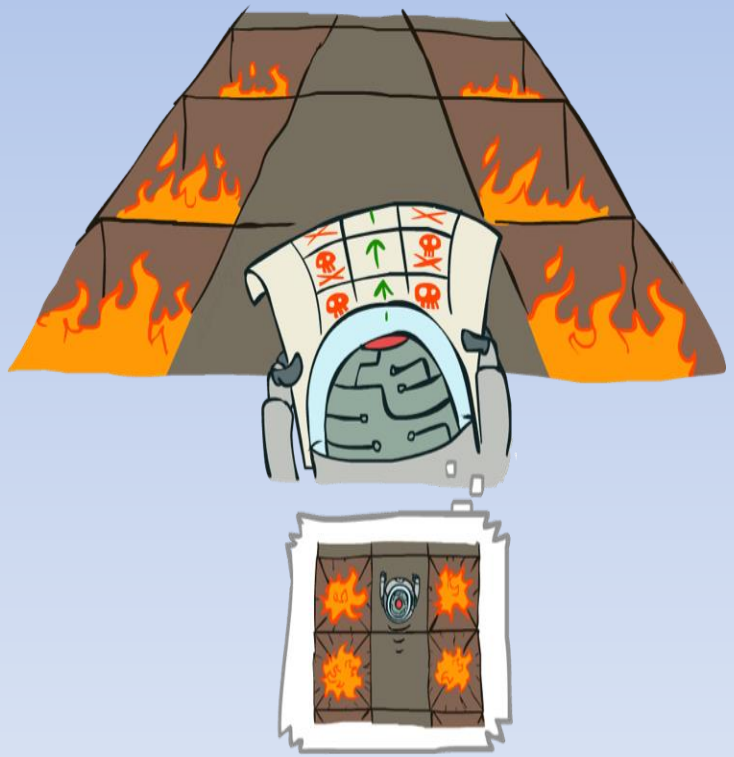
$$V^{\pi}(s) = \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^{\pi}(s')]$$

Example: Policy Evaluation

Always Go Right

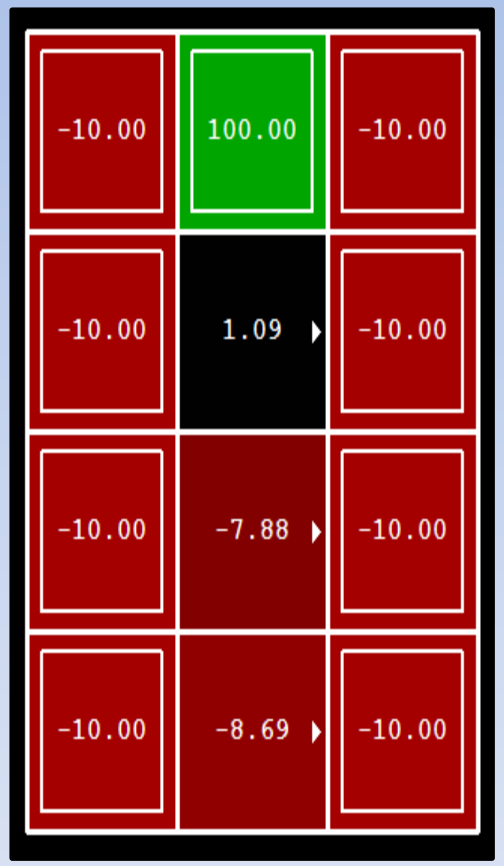


Always Go Forward

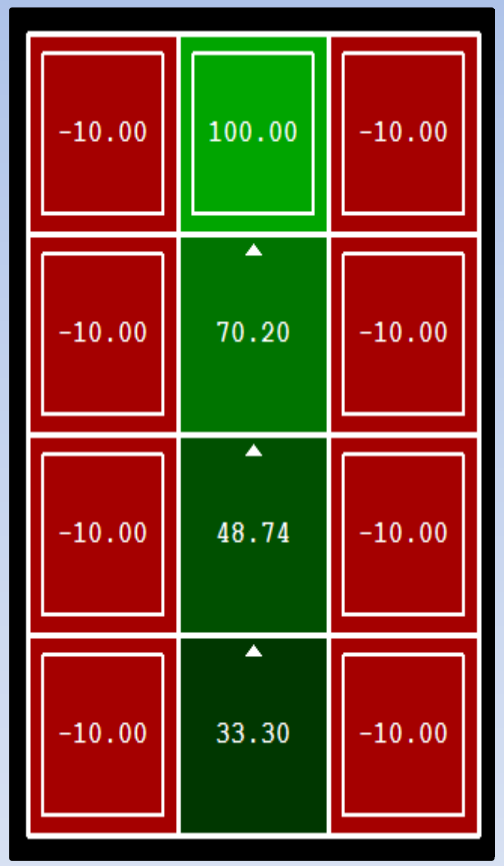


Example: Policy Evaluation

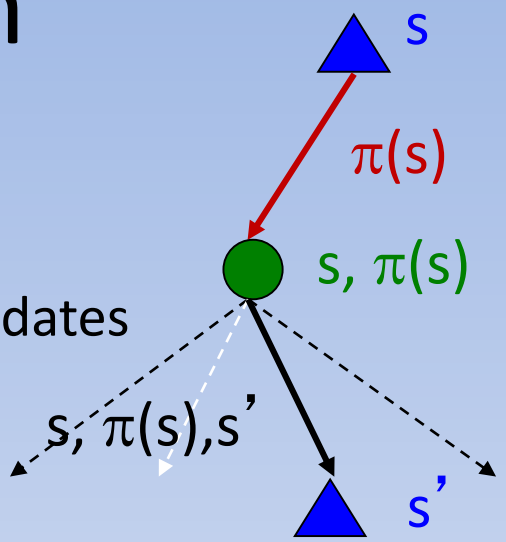
Always Go Right



Always Go Forward



Policy Evaluation



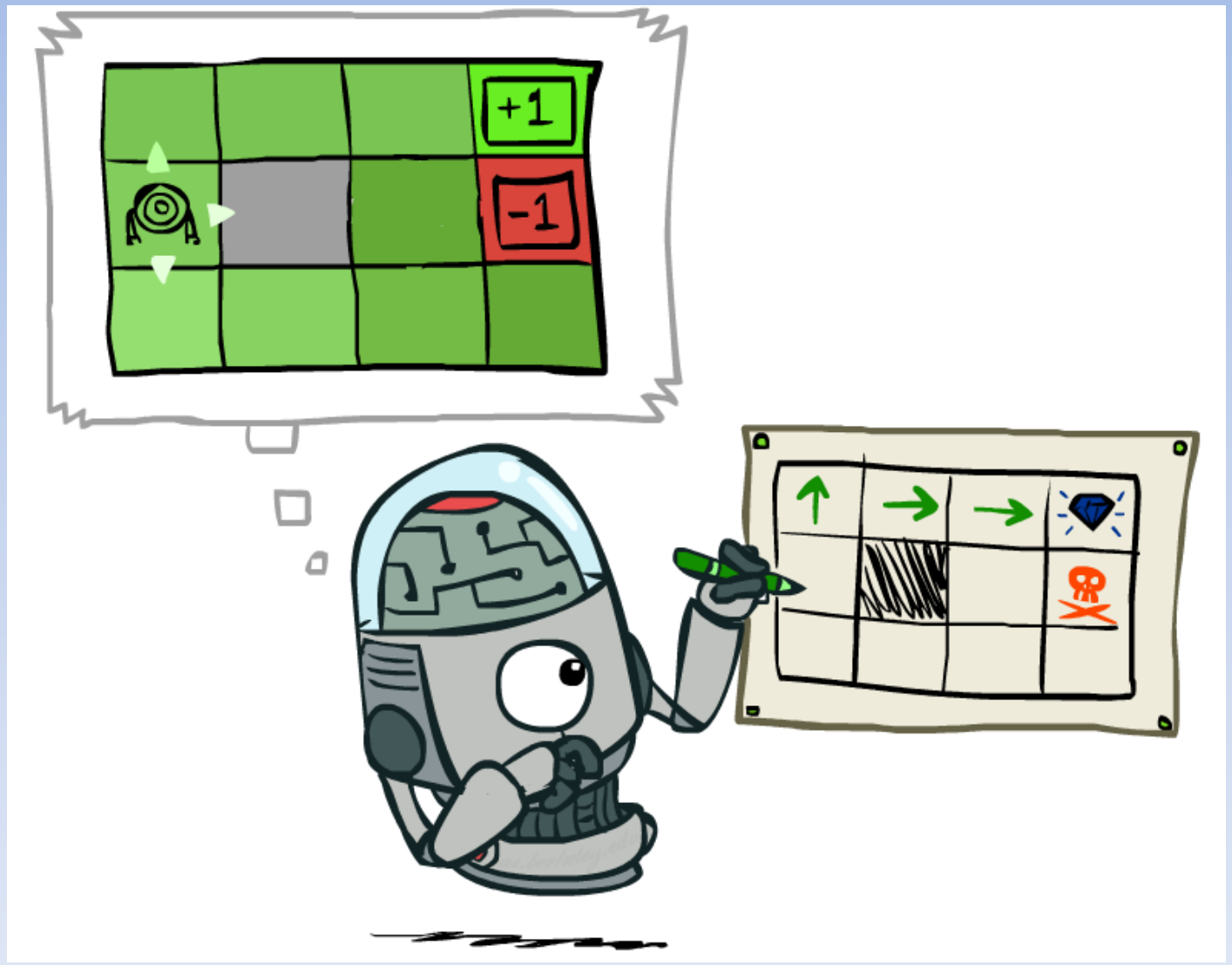
- How do we calculate the V's for a fixed policy π?
- Idea 1: Turn recursive Bellman equations into updates (like value iteration)

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

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

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

Policy Extraction



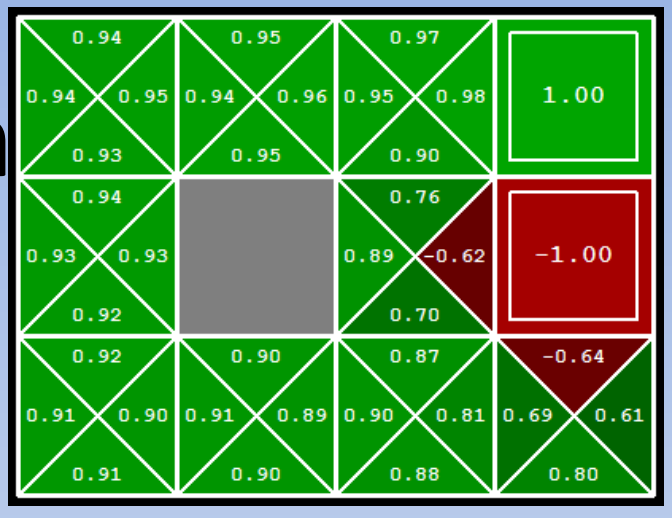
Computing Actions from Values

- 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

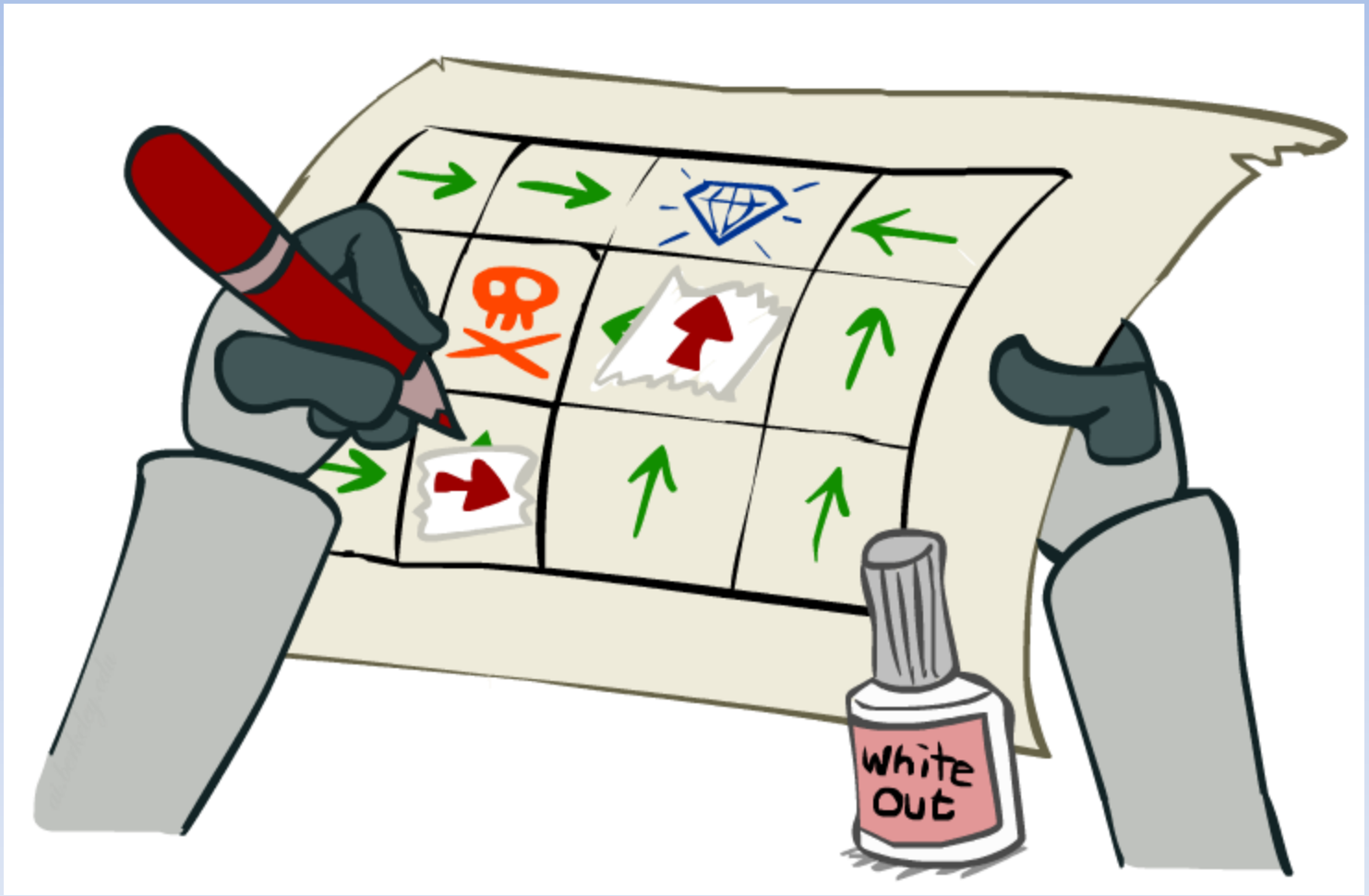


- Let's imagine we have the optimal q-values:
- How should we act?
  - Completely trivial to decide!

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

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

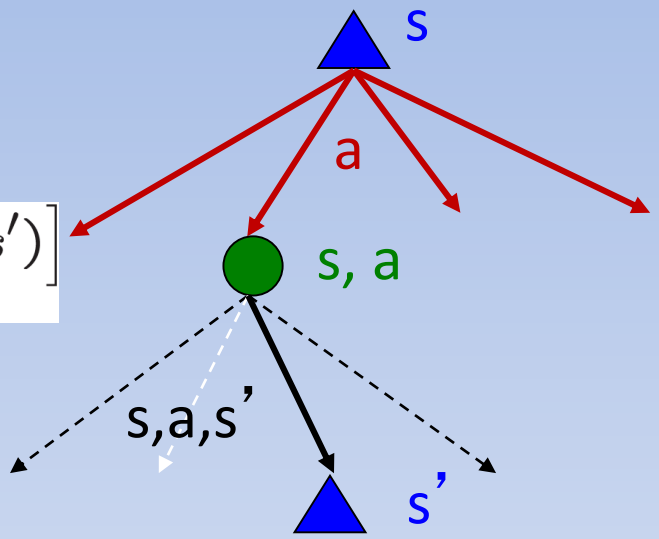
Policy Iteration



Problems with Value Iteration

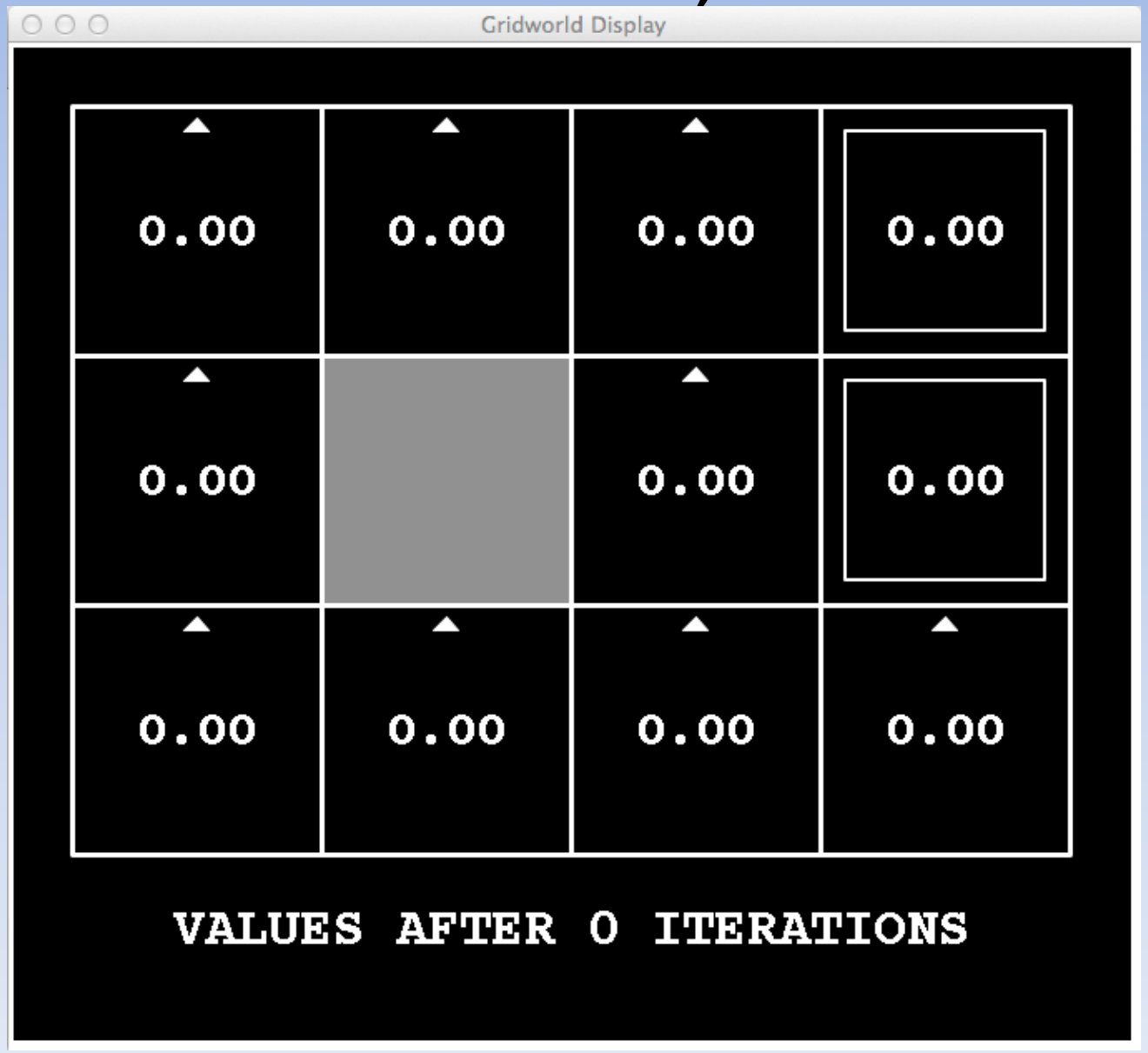
- Value iteration repeats the Bellman updates:

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



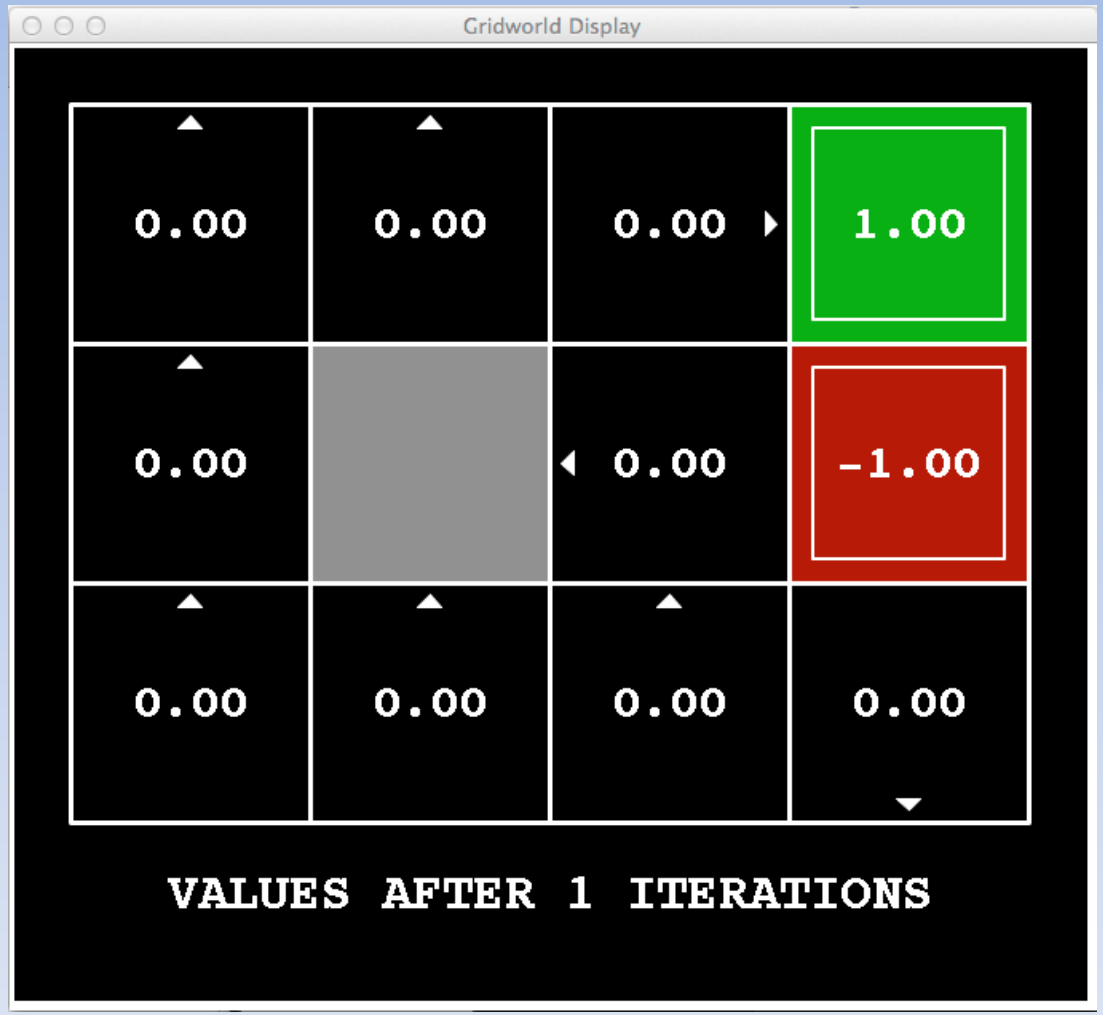
- Problem 1: It's slow – $O(S^2A)$ per iteration
- Problem 2: The "max" at each state rarely changes
- Problem 3: The policy often converges long before the values

K=0, 0 moves



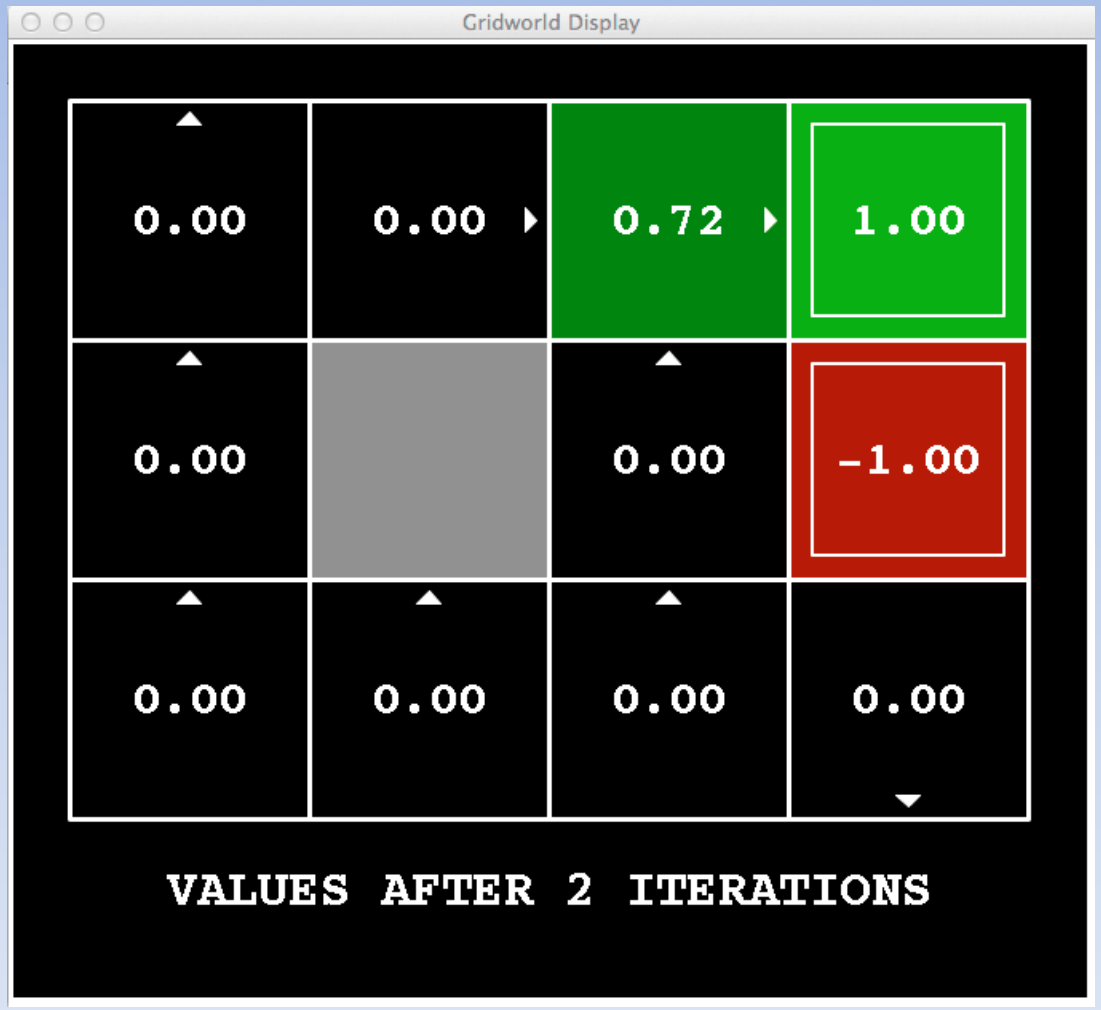
Discount = 0.9
Living reward = 0

K=1, 1 move



Discount = 0.9
Living reward = 0

K=2, 2 moves



Discount = 0.9
Living reward = 0

K=3

Discount = 0.9
Living reward = 0

K=4

Discount = 0.9
Living reward = 0

K=5

Discount = 0.9
Living reward = 0

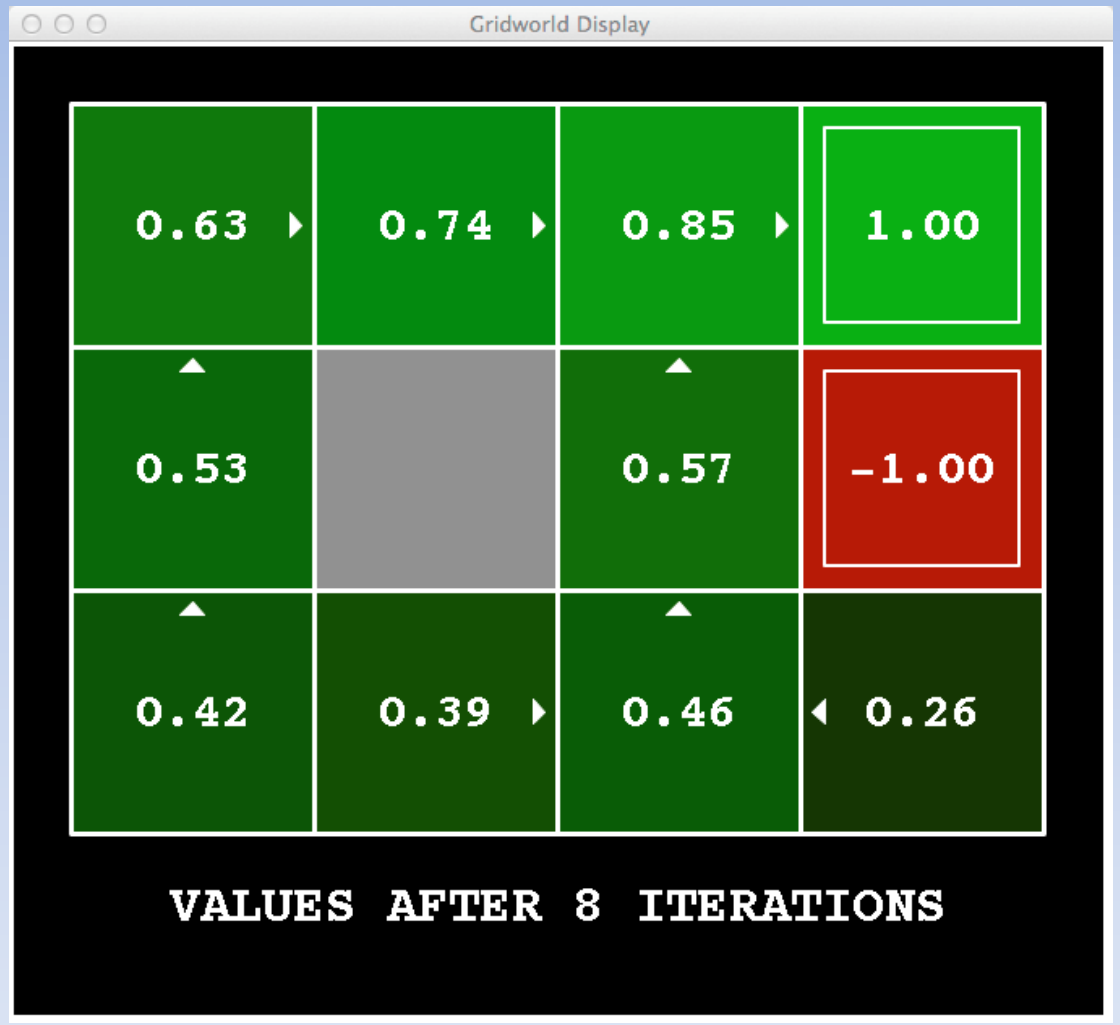
K=6

Discount = 0.9
Living reward = 0

K=7

Discount = 0.9
Living reward = 0

K=8

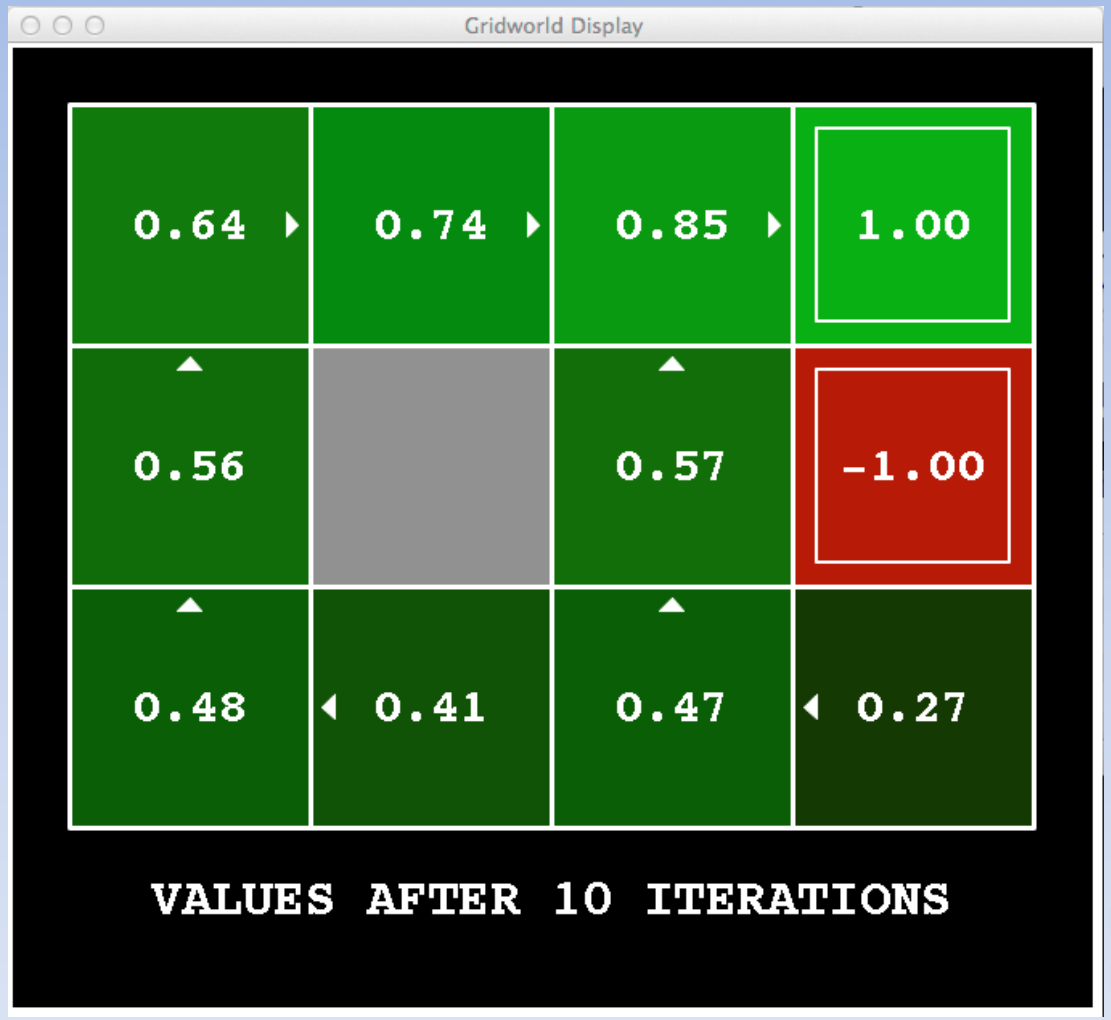


Discount = 0.9
Living reward = 0

K=9

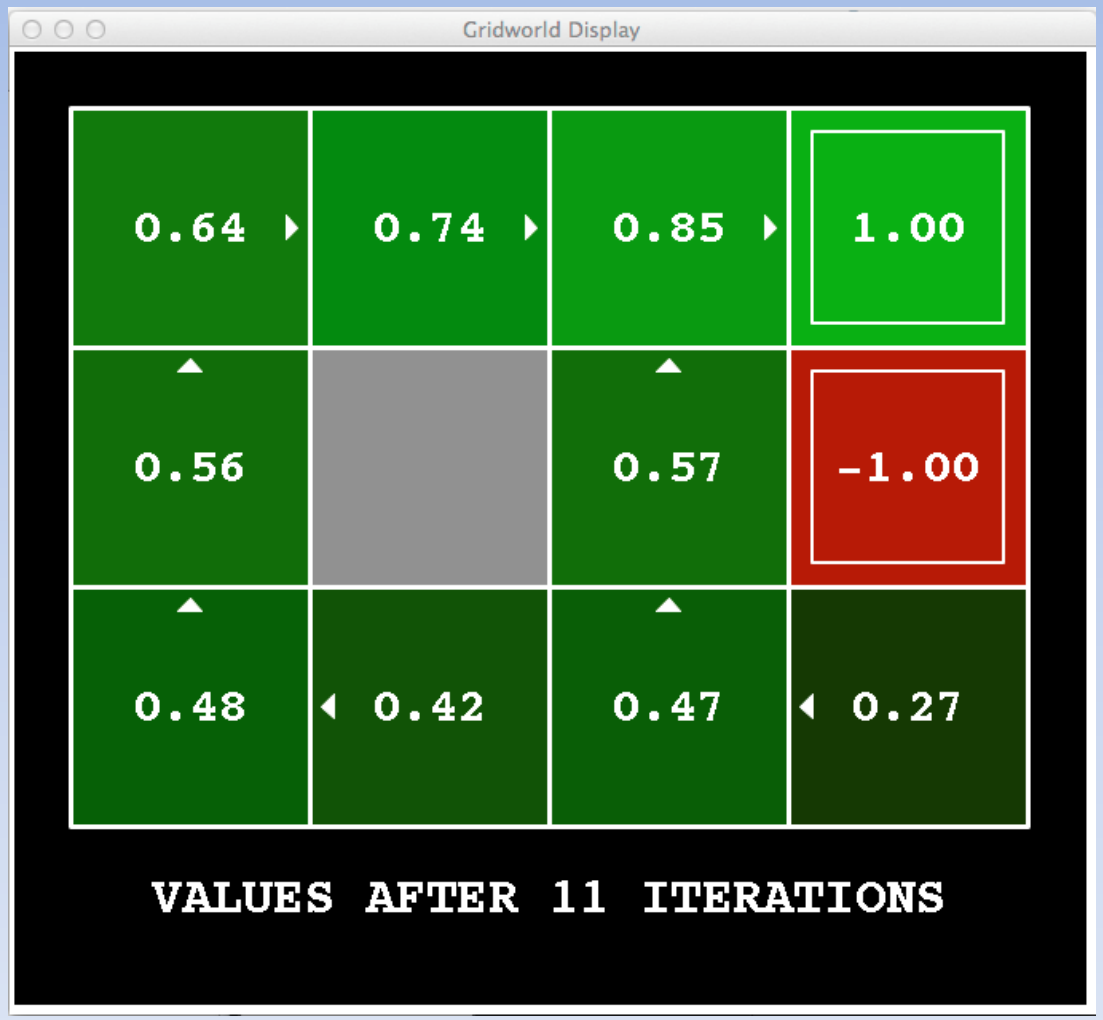
Discount = 0.9
Living reward = 0

K=10



Discount = 0.9
Living reward = 0

K=11

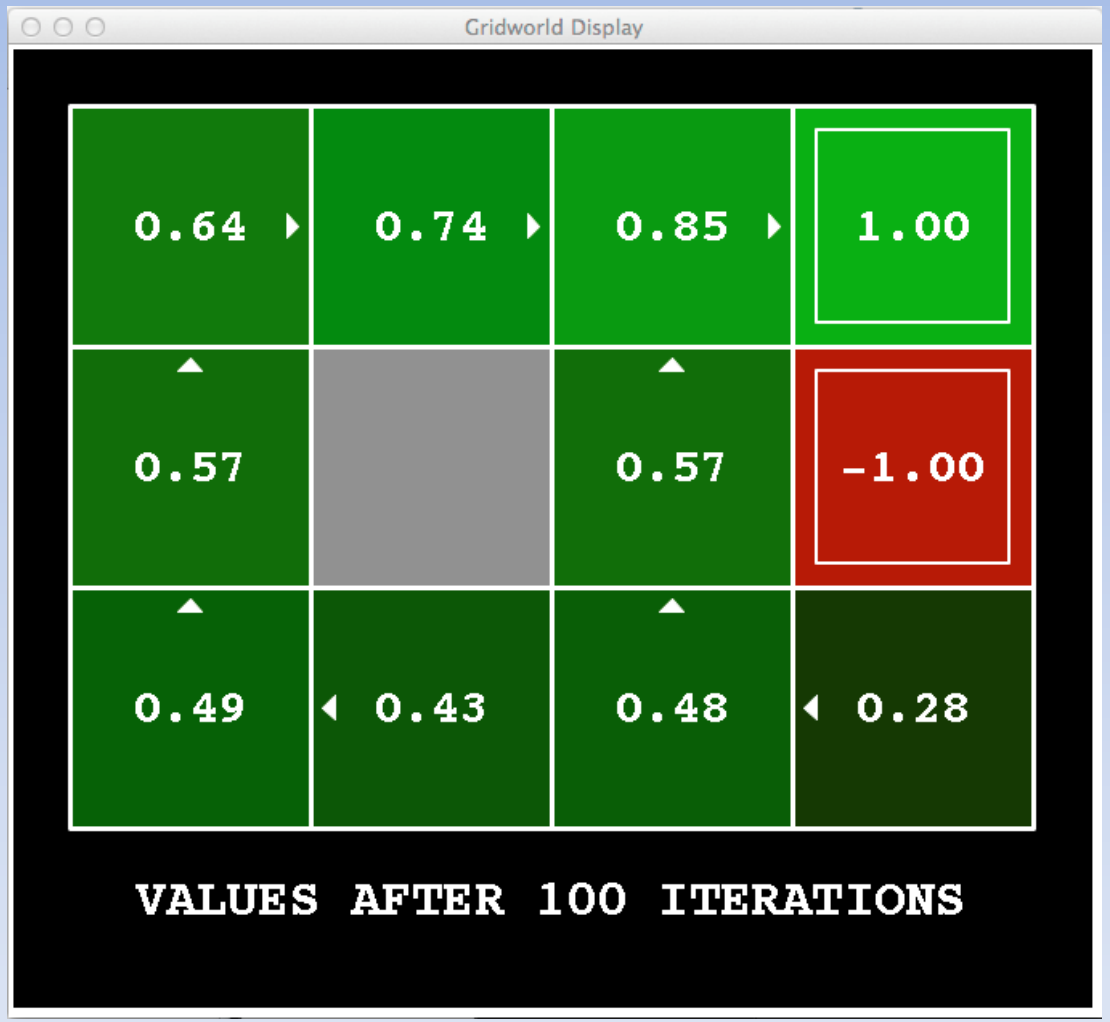


Discount = 0.9
Living reward = 0

K=12

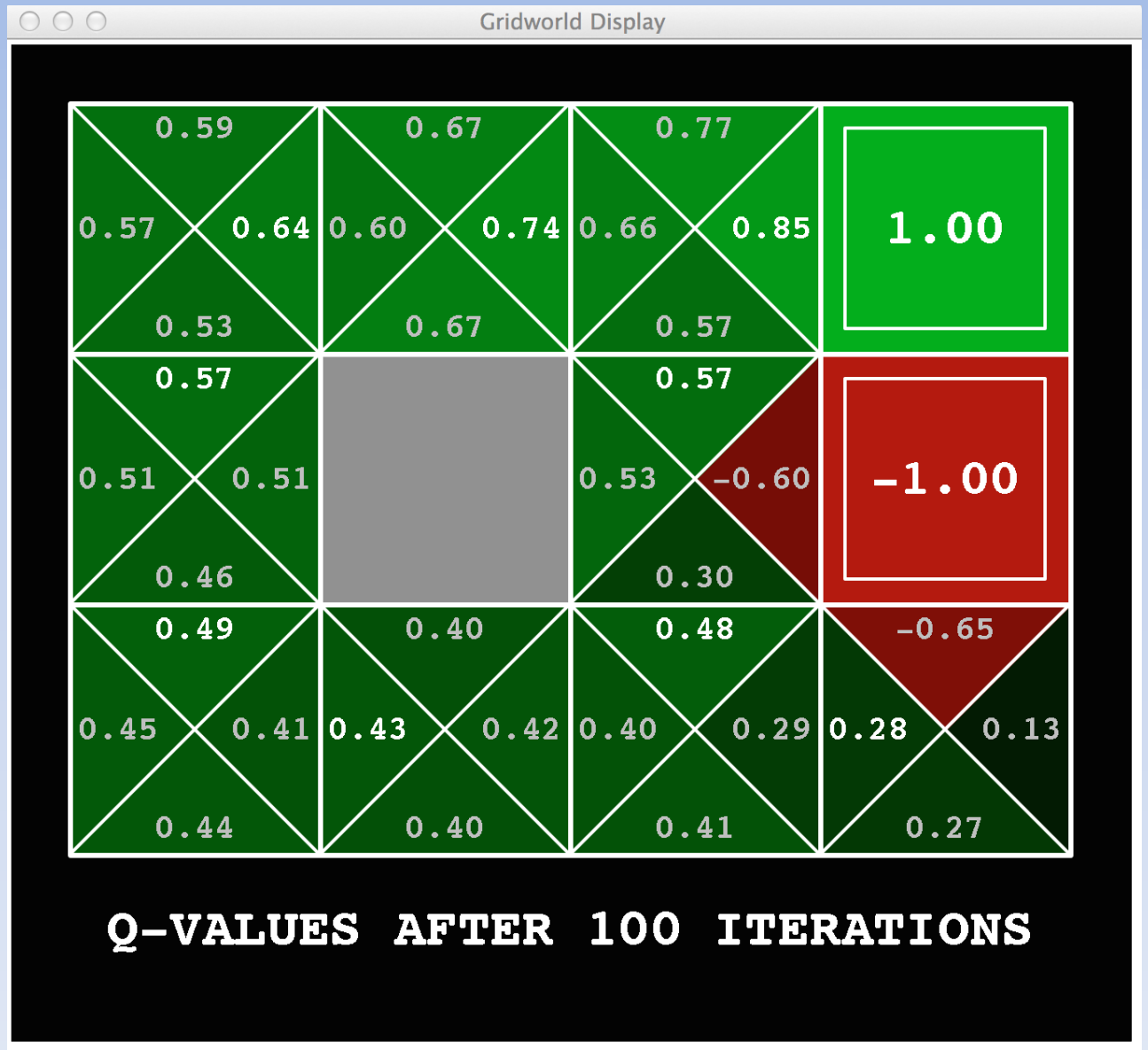
Discount = 0.9
Living reward = 0

K=100



Discount = 0.9
Living reward = 0

Q*-Values


Gridworld Display
0.59
0.57
0.64
0.53
0.67
0.60
0.74
0.67
0.77
0.66
0.85
0.57
1.00
0.57
0.51
0.51
0.46
0.57
0.53
-0.60
0.30
-1.00
0.49
0.45
0.41
0.44
0.40
0.43
0.42
0.40
0.48
0.40
0.29
0.41
-0.65
0.28
0.13
0.27
Q-VALUES AFTER 100 ITERATIONS

Policy Iteration

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

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

Policy Iteration

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

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

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

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

Policy Iteration w/ Textbook

function POLICY-ITERATION(mdp) **returns** a policy
  inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$
  local variables: U, a vector of utilities for states in S, initially zero
    π, a policy vector indexed by state, initially random

  repeat
    $U \leftarrow$ POLICY-EVALUATION(π, U, mdp)
    $unchanged? \leftarrow$ true
    for each state s **in** S **do**
      if $\max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s'] > \sum_{s'} P(s' \mid s, \pi[s])\ U[s']$ **then do**
        $\pi[s] \leftarrow \operatorname{argmax}_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s']$
        $unchanged? \leftarrow$ false
  until $unchanged?$
  return π

Figure 17.7 The policy iteration algorithm for calculating an optimal policy.

Comparison

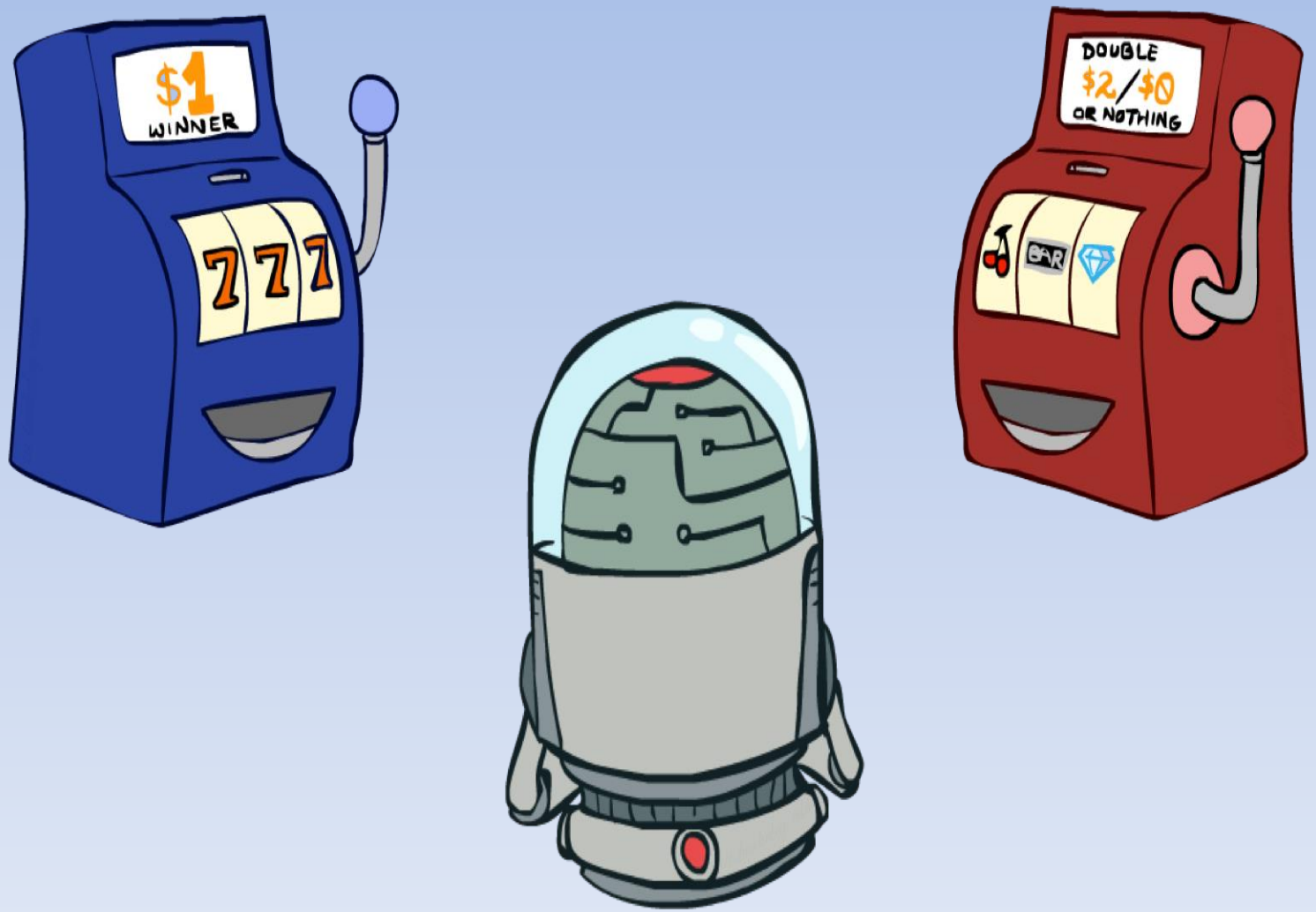
- Both value iteration and policy iteration compute the same thing (all optimal values)
- In value iteration:
  - Every iteration updates both the values and (implicitly) the policy
  - We don't track the policy, but taking the max over actions implicitly recomputes it
- In policy iteration:
  - We do 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)

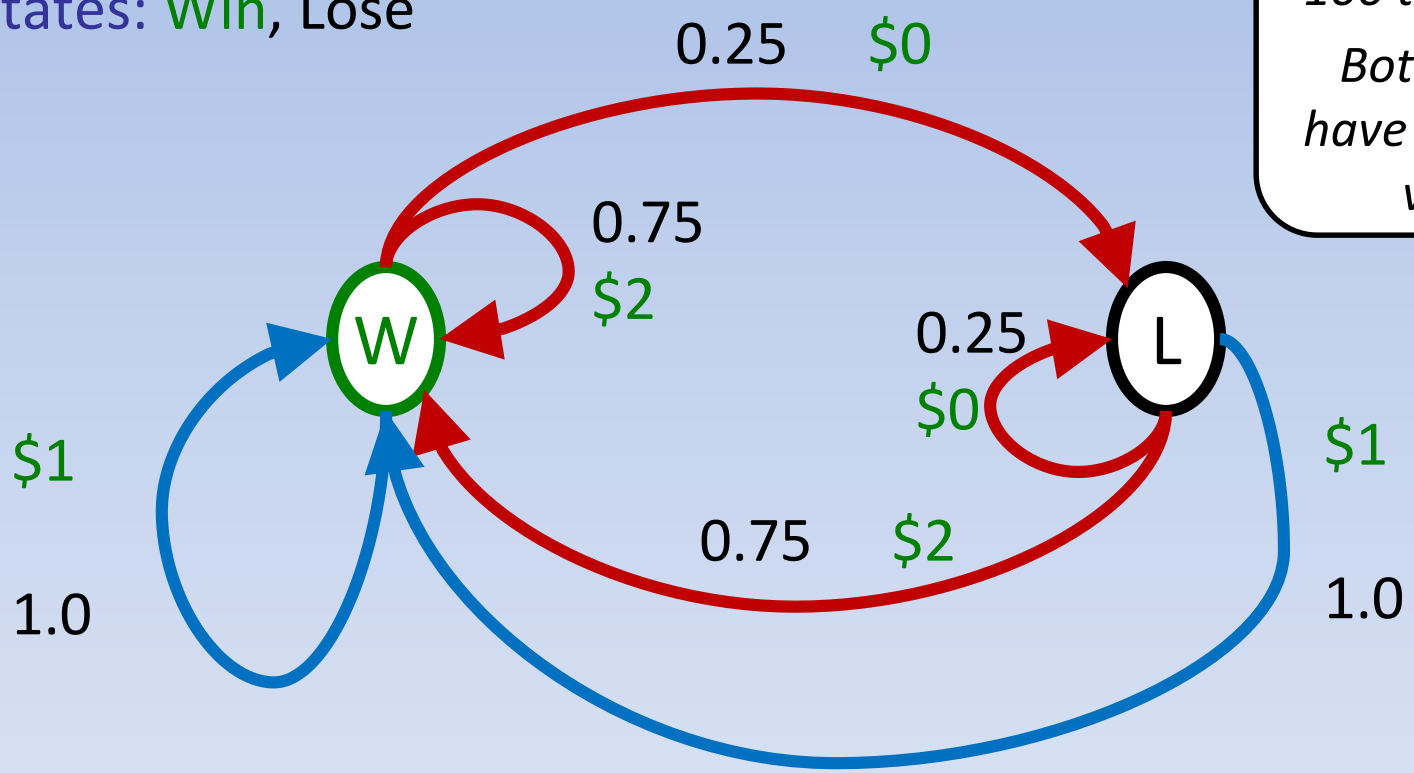
- These all look the same!
  - They basically are – they are all variations of Bellman updates
  - They all use one-step lookahead expectimax fragments
  - They differ only in whether we plug in a fixed policy or max over actions

Double Bandits



Double-Bandit MDP

- Actions: *Blue*, *Red*
- States: Win, Lose



No discount

100 time steps

Both states have the same value

Offline Planning

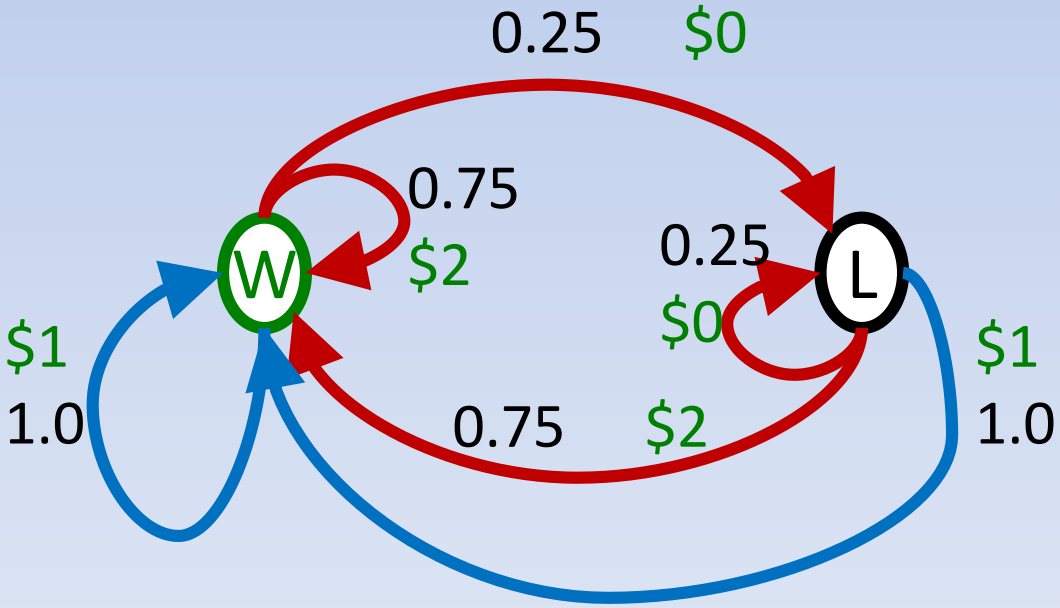
- Solving MDPs is offline planning
  - You determine all quantities through computation
  - You need to know the details of the MDP
  - You do not actually play the game!

No discount

100 time steps

Both states have the same value

	Value
Play Red	150
Play Blue	100



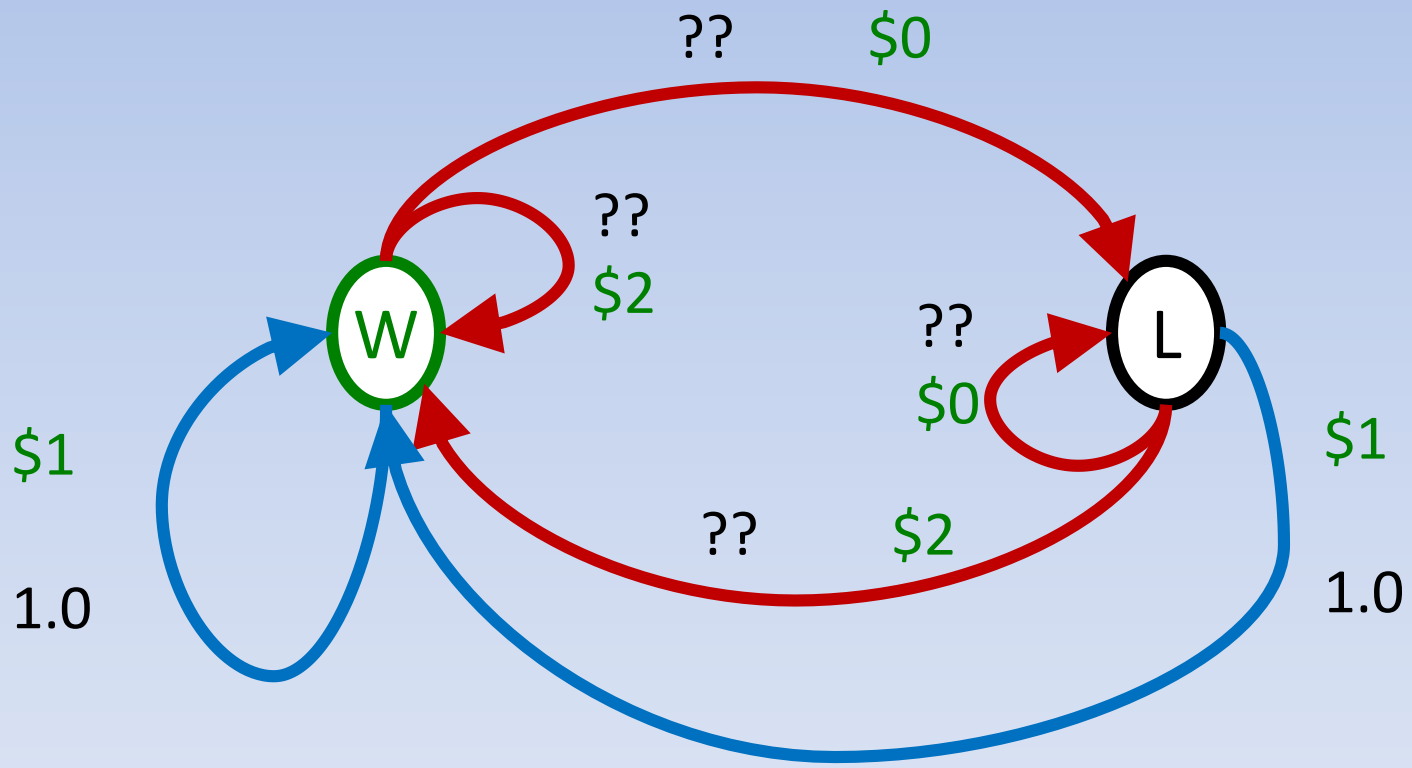
Let's Play!

$2 $2 $0 $2 $2

$2 $2 $0 $0 $0

Online Planning

- Rules changed! Red's win chance is different.



Let's Play!

$0 $0$0 $2$0

$2 $0$0 $0$0

What Just Happened?

- That wasn't planning, it was learning!
  - Specifically, reinforcement learning
  - There was an MDP, but you couldn't solve it with just computation
  - You needed to actually act to figure it out
- Important ideas in reinforcement learning that came up
  - Exploration: you have to try unknown actions to get information
  - Exploitation: eventually, you have to use what you know
  - Regret: even if you learn intelligently, you make mistakes
  - Sampling: because of chance, you have to try things repeatedly
  - Difficulty: learning can be much harder than solving a known MDP