

# Project

---

- Group registration

- Register your group members by May 9
- By default, no more than 3 people in each group. Talk to me if you wish to form a larger group

- Proposal presentation

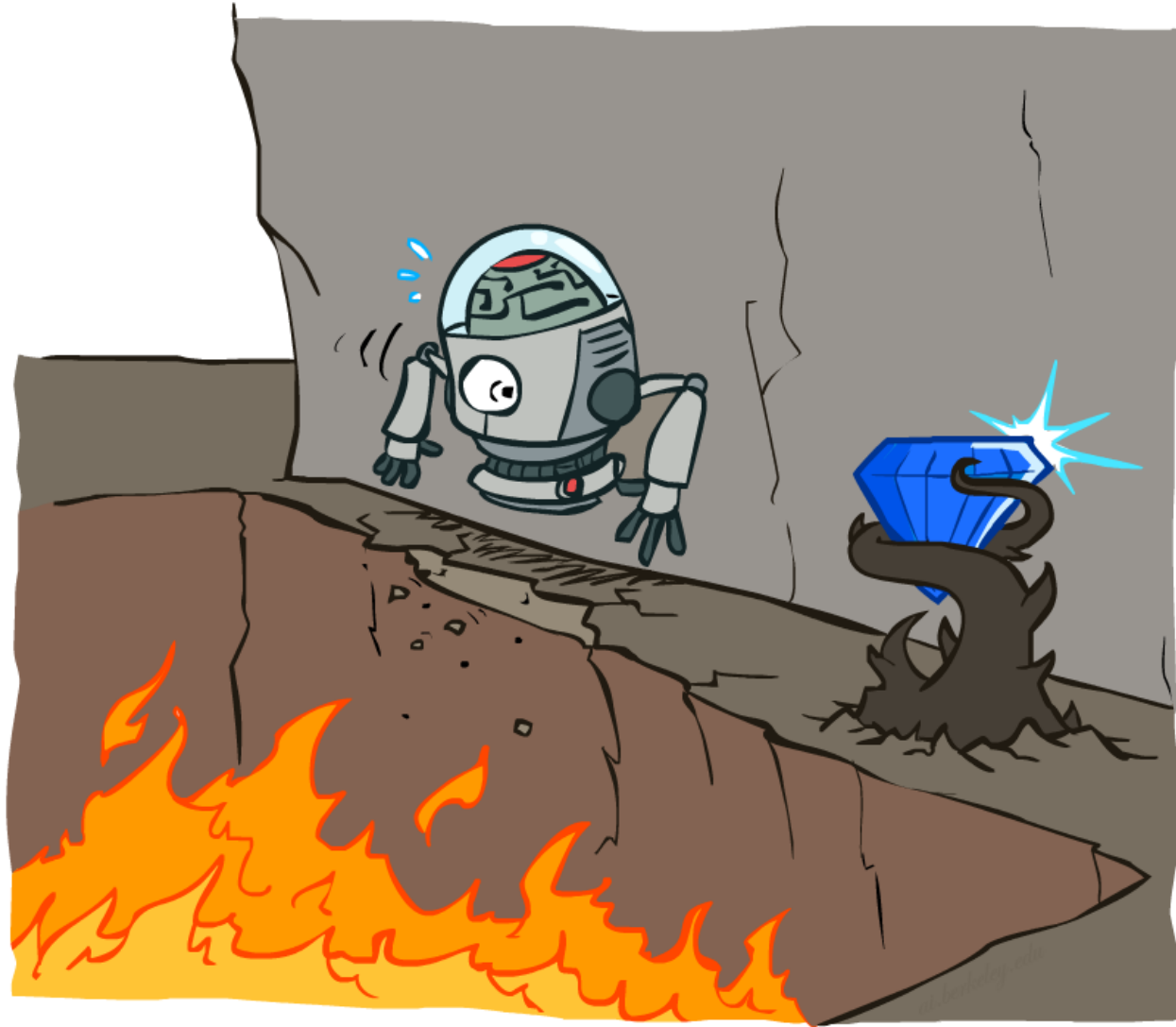
- May 17, 22
- 5-10min per group (depending on the number of groups, TBD)

# Markov Decision Processes



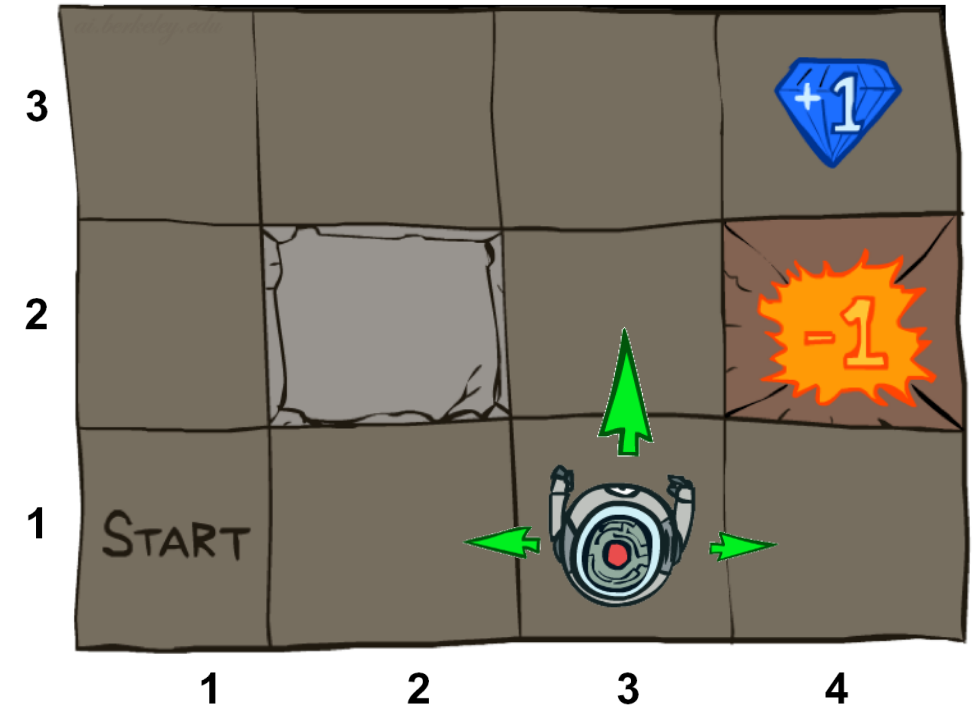
AIMA Chapter 17

# Non-Deterministic Search



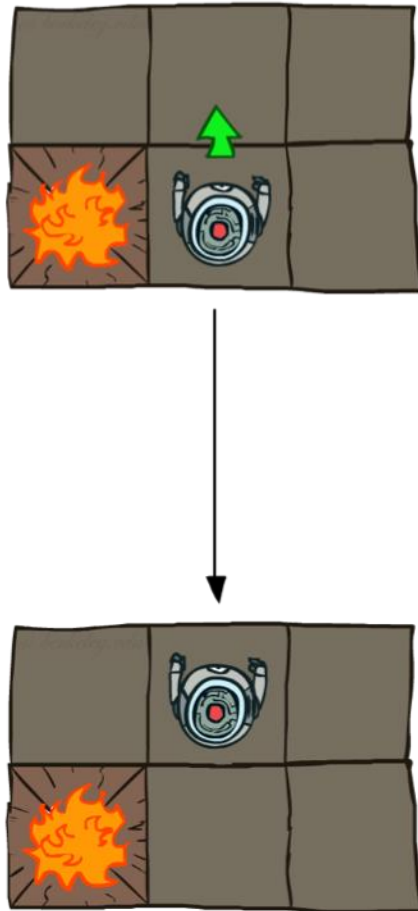
# Example: Grid World

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

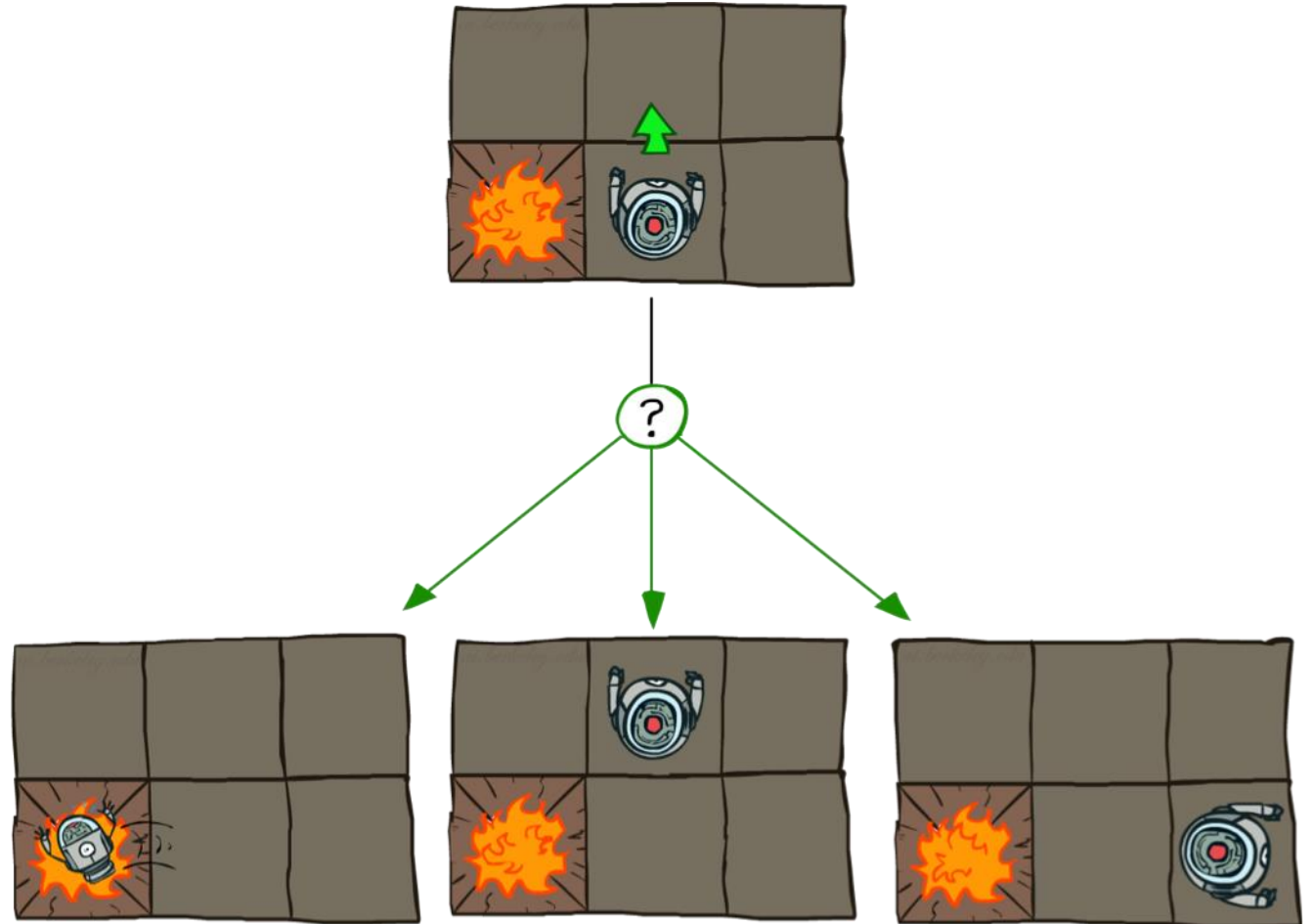


# Grid World Actions

Deterministic Grid World

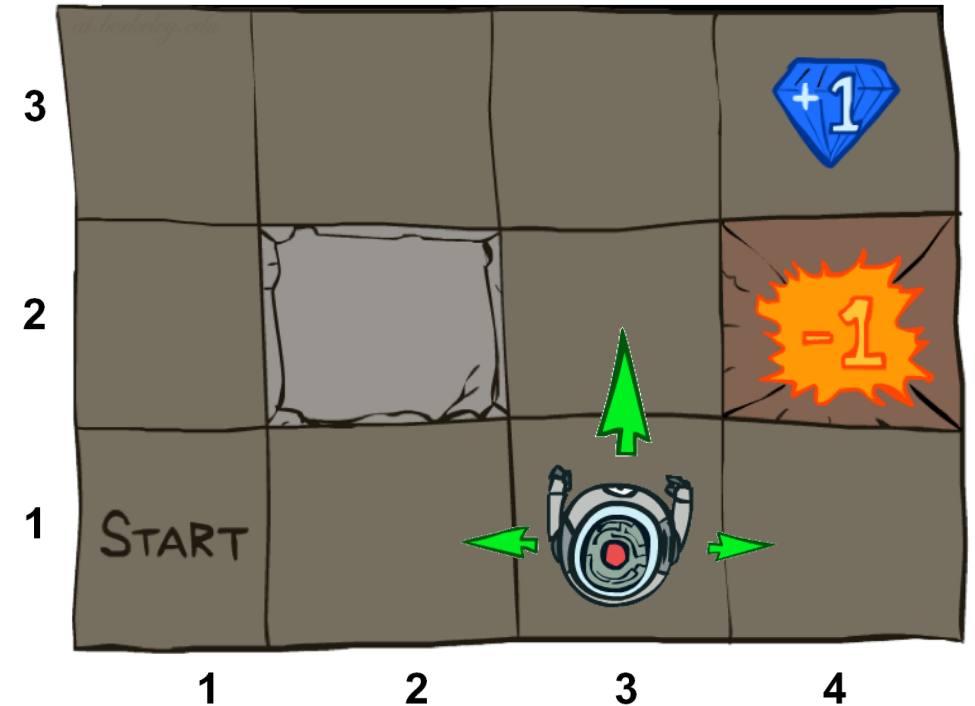


Stochastic Grid World



# Markov Decision Processes

- An MDP is defined by:
  - A **set of states**  $s \in S$
  - A **set of actions**  $a \in A$
  - A **transition function**  $T(s, a, s')$ 
    - Probability that  $a$  from  $s$  leads to  $s'$ , i.e.,  $P(s' | s, a)$
    - Also called the model or the dynamics
  - A **reward function**  $R(s, a, s')$ 
    - Sometimes just  $R(s)$  or  $R(s')$
  - A **start state**
  - Maybe a **terminal state**
- MDPs are non-deterministic search problems
  - One way to solve them is with expectimax search
  - We'll have a new tool soon



# What is Markov about MDPs?

- “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

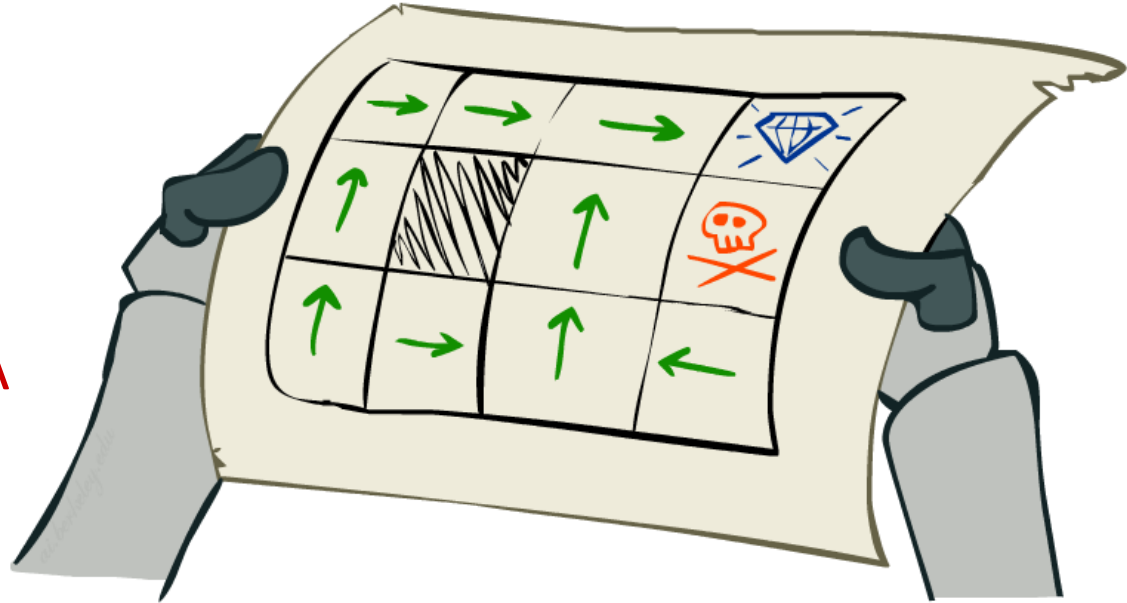
$$\begin{aligned} &P(S_{t+1} = s' | S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1}, \dots, S_0 = s_0) \\ &= \\ &P(S_{t+1} = s' | S_t = s_t, A_t = a_t) \end{aligned}$$

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



Andrey Markov  
(1856-1922)

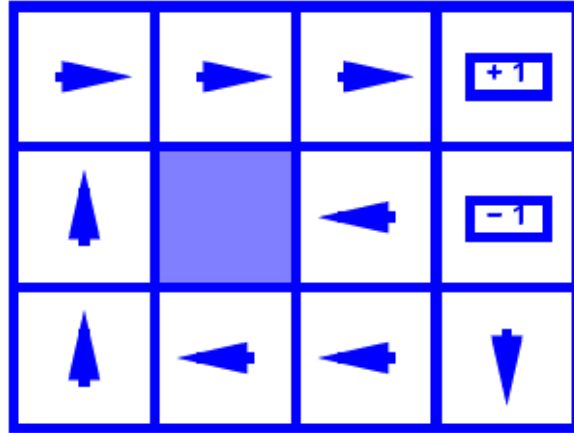
- 
- A hand-drawn illustration of a treasure map held by two hands. The map is a 4x4 grid with various symbols: green arrows indicating a path, a blue diamond in the top-right, a red skull and crossbones, and a hatched area representing a hazard.



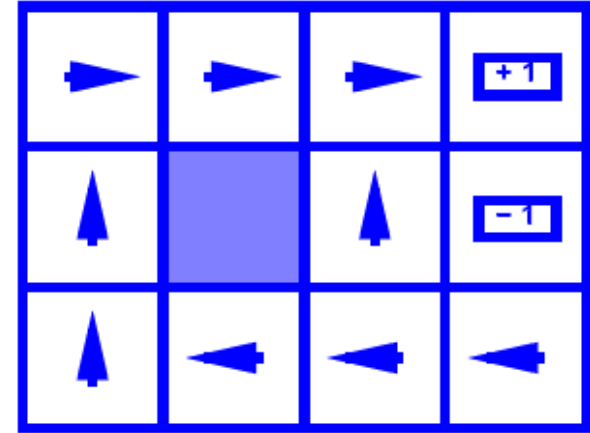


# Optimal Policies

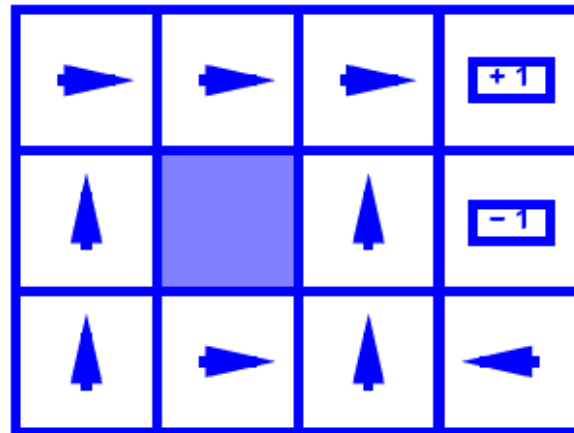
$R(s)$  = “living reward”



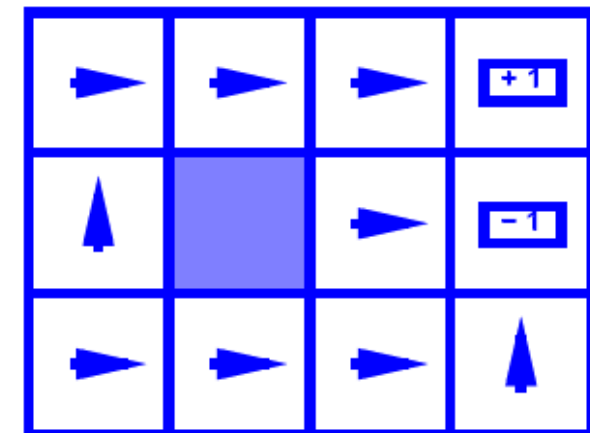
$R(s) = -0.01$



$R(s) = -0.03$



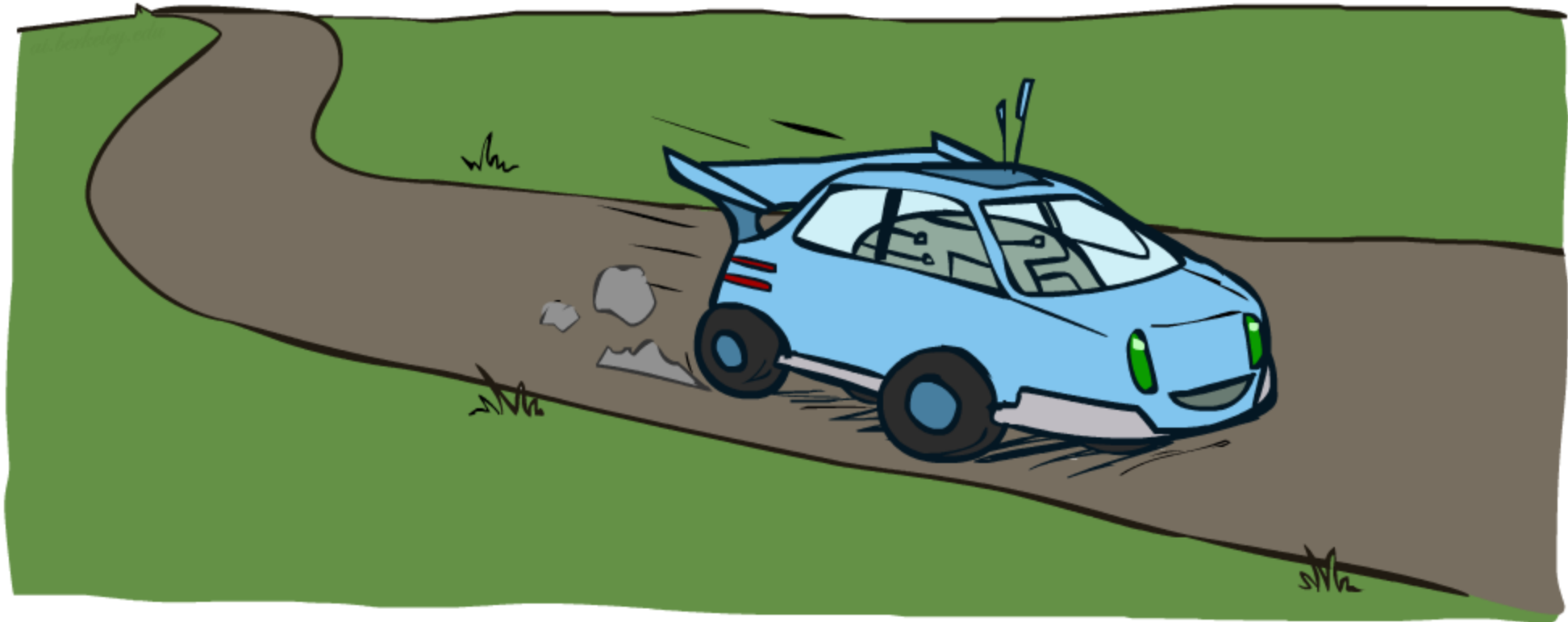
$R(s) = -0.4$



$R(s) = -2.0$

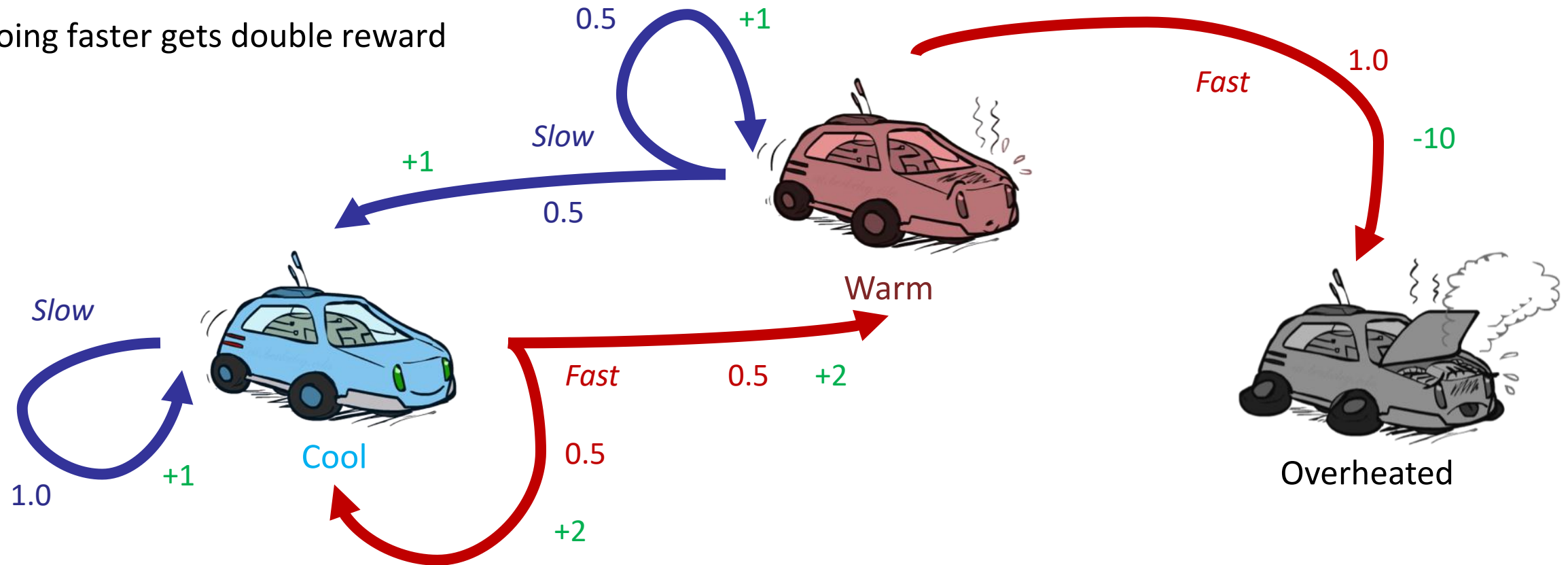
# Example: Racing

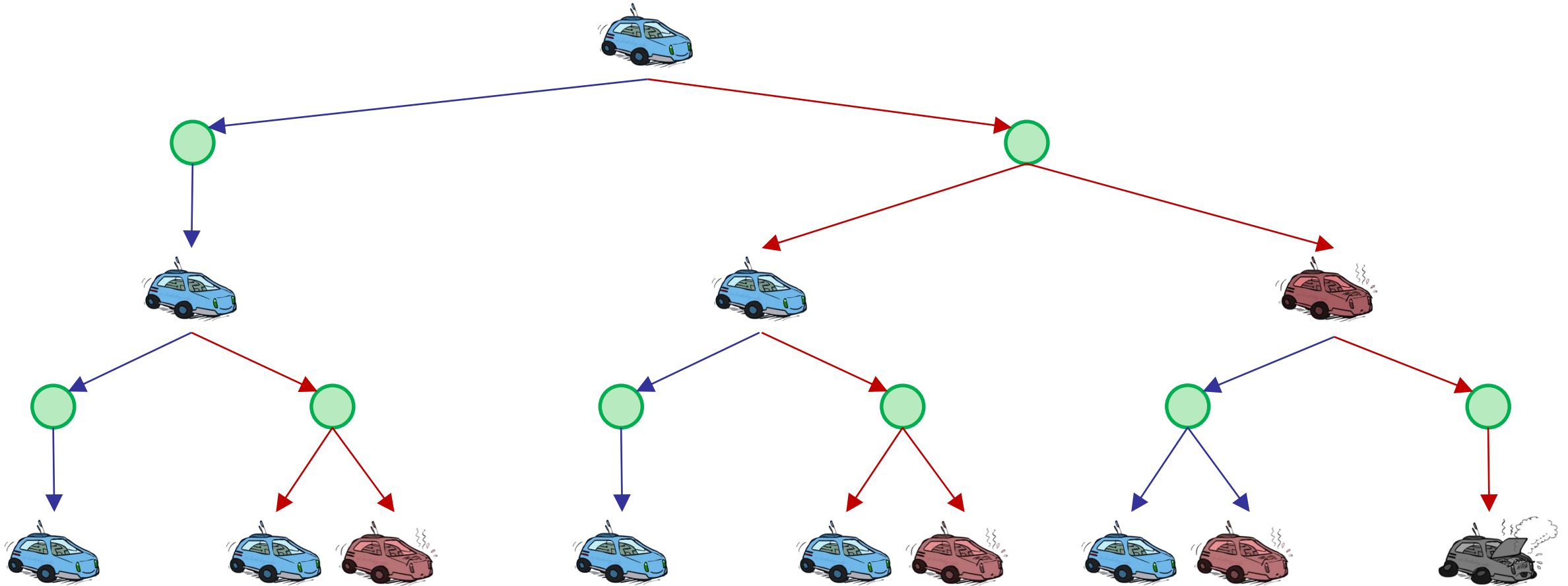
---



# Example: Racing

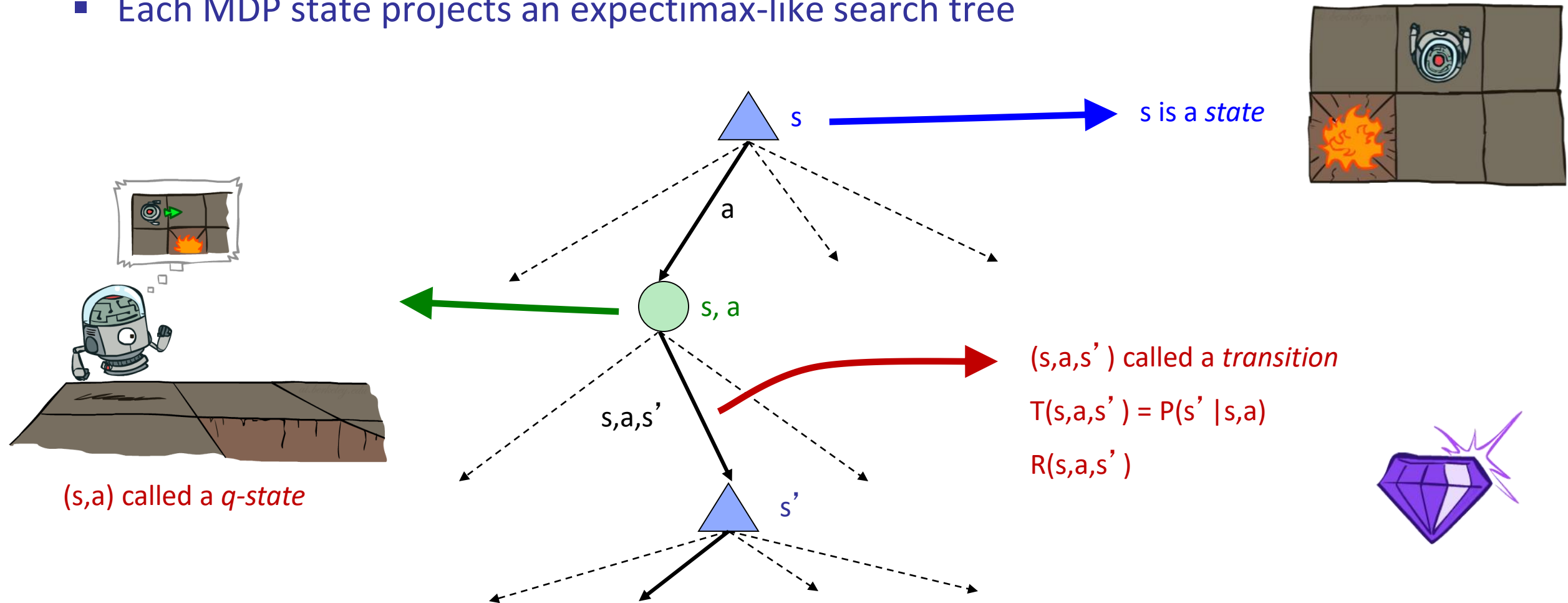
- A robot car wants to travel far, quickly
- Three states: **Cool**, **Warm**, Overheated
- Two actions: **Slow**, **Fast**
- Going faster gets double reward



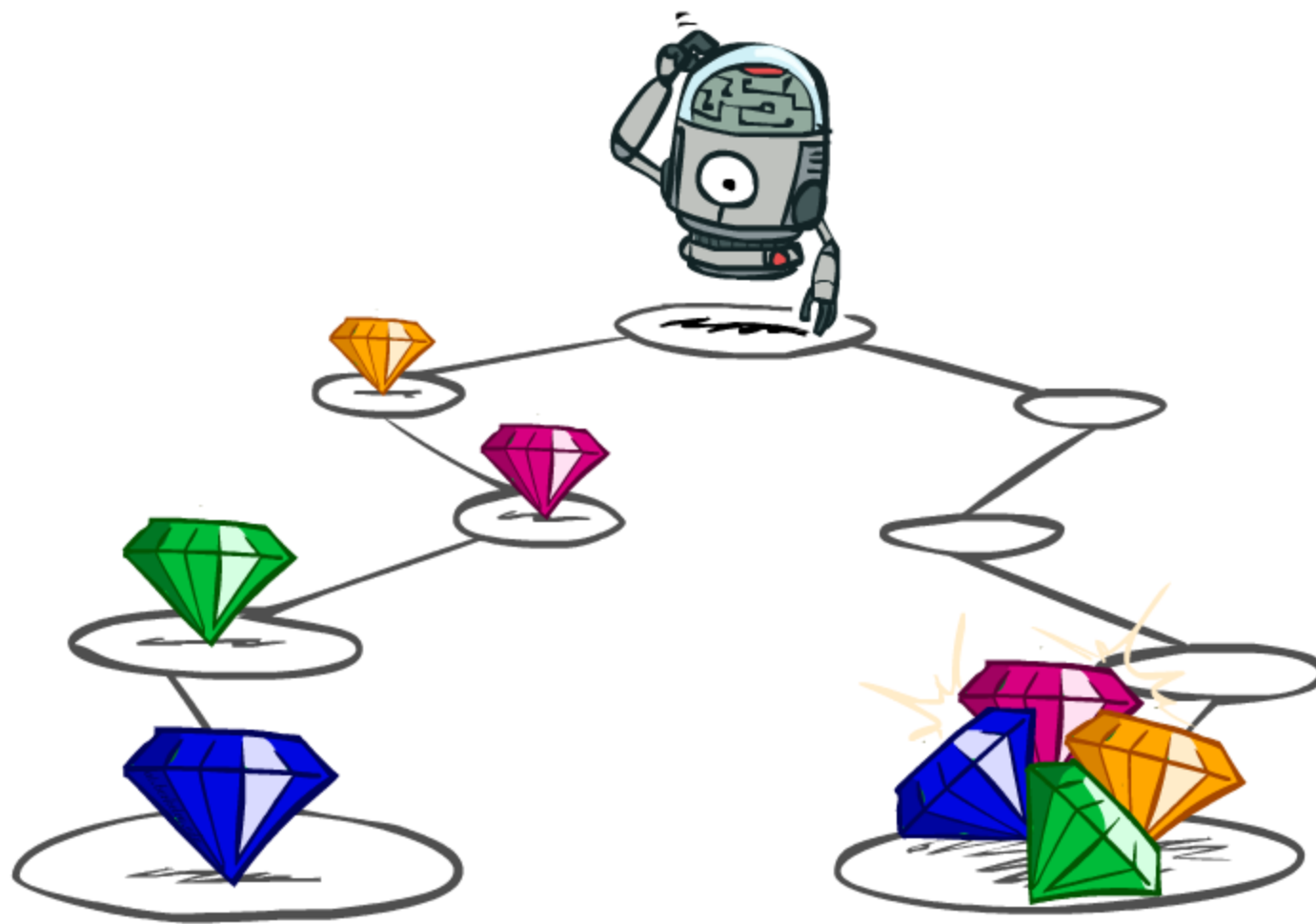


# MDP Search Trees

- Each MDP state projects an expectimax-like search tree

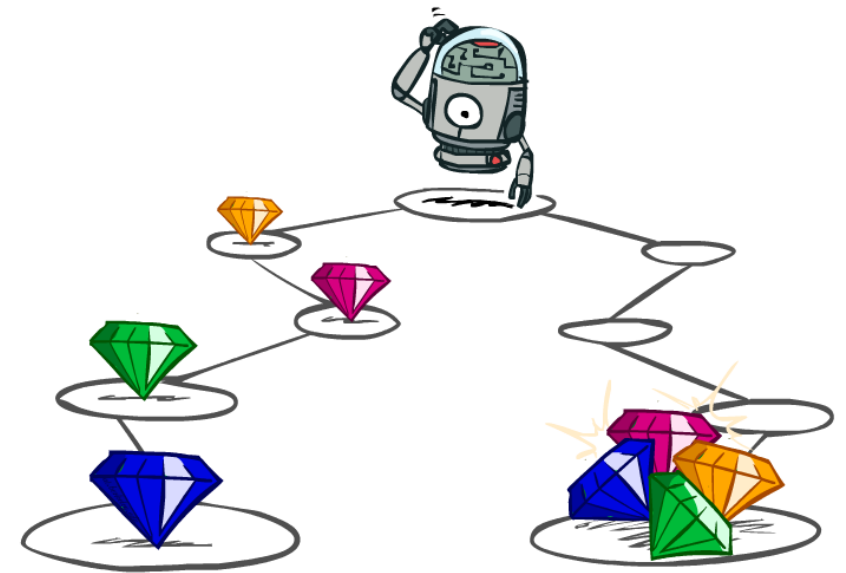


# Utilities of Sequences



# Utilities of Sequences

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



# Discounting

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



1

Worth Now



$\gamma$

Worth Next Step



$\gamma^2$

Worth In Two Steps



# Discounting

- How to discount?

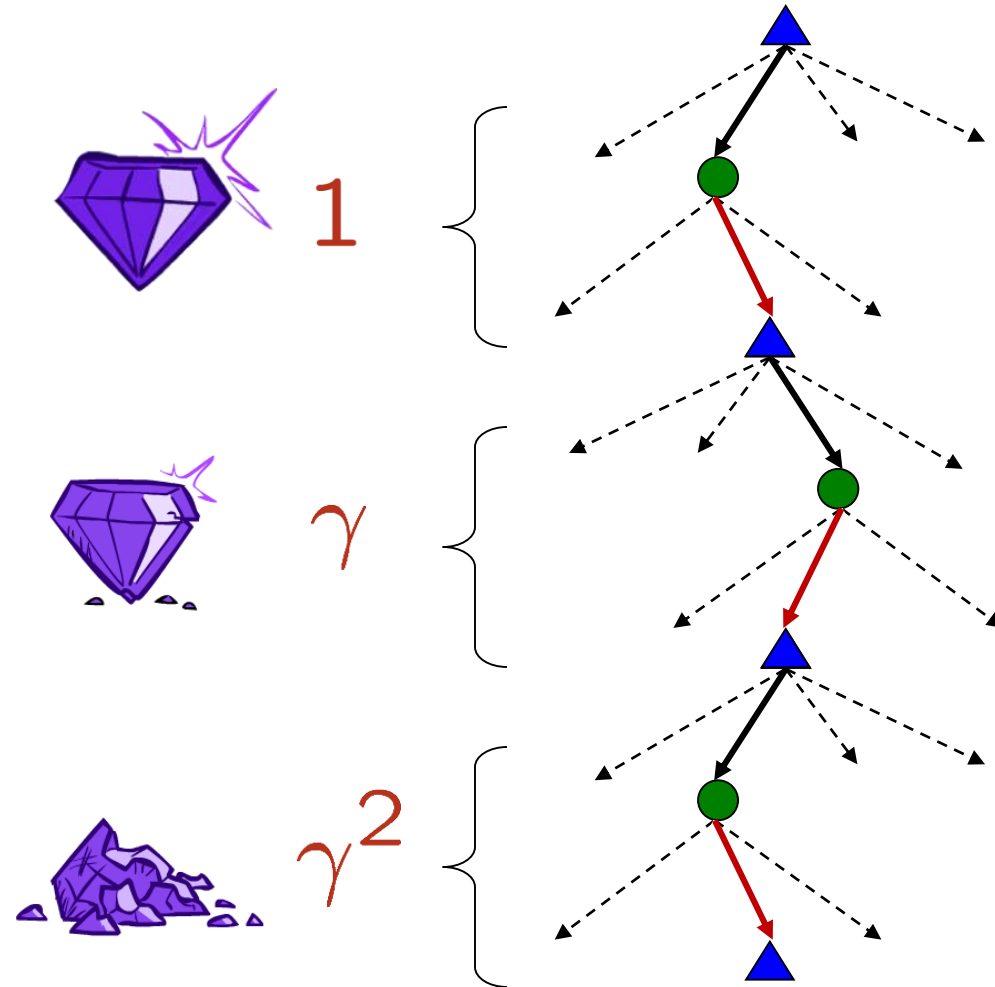
- Each time we descend a level, we multiply in the discount once

- Why discount?

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

- Example: discount of 0.5

- $U([1,2,3]) = 1*1 + 0.5*2 + 0.25*3$
- $U([1,2,3]) < U([3,2,1])$



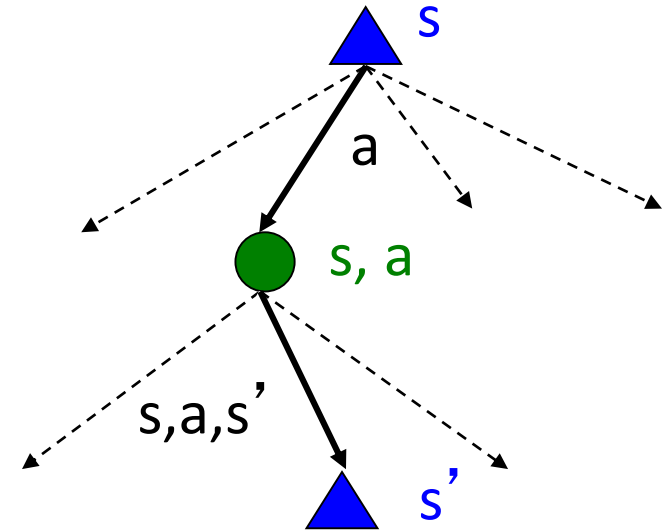
# Recap: Defining MDPs

- Markov decision processes:

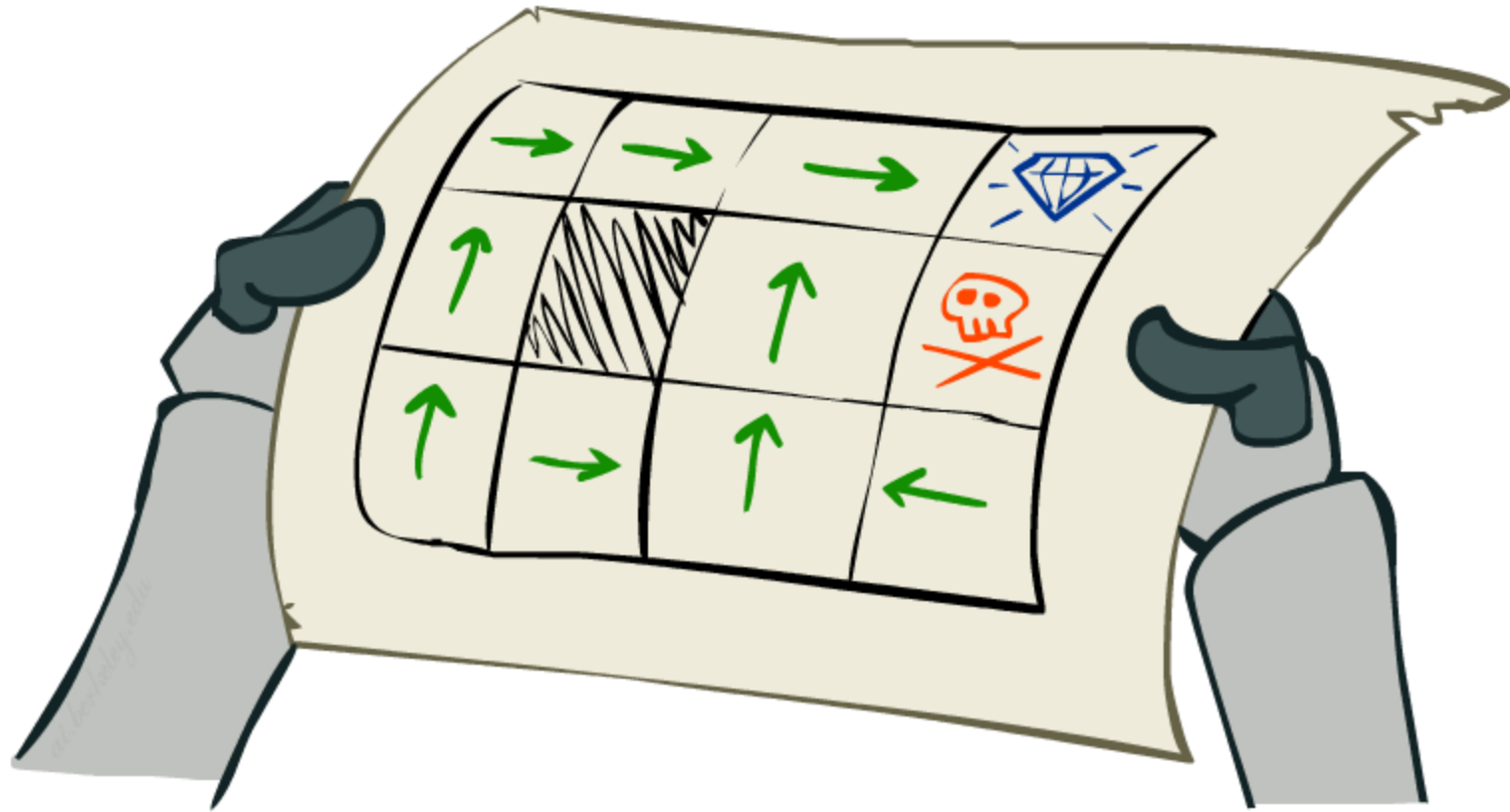
- Set of states  $S$
- Start state  $s_0$
- Set of actions  $A$
- Transitions  $P(s' | s, a)$  (or  $T(s, a, s')$ )
- Rewards  $R(s, a, s')$  (and discount  $\gamma$ )

- MDP quantities so far:

- Policy = Choice of action for each state
- Utility = sum of (discounted) rewards

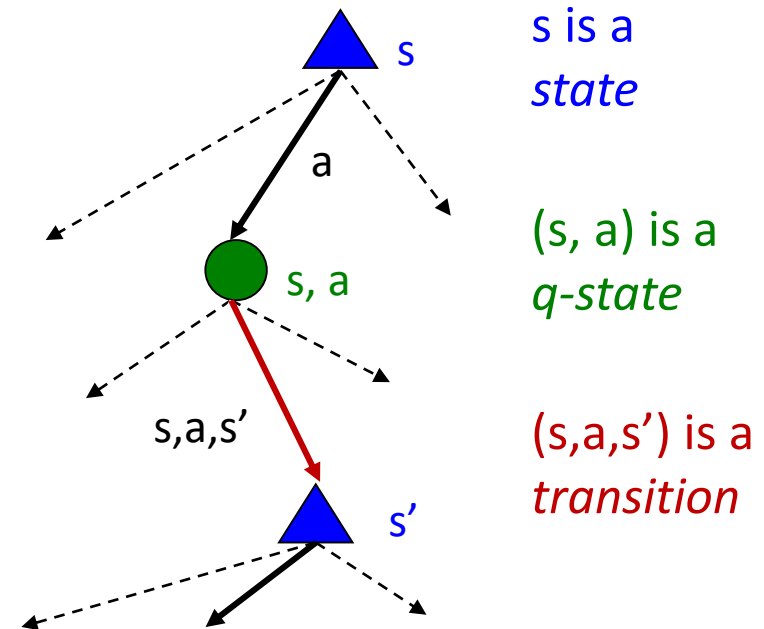


# Solving MDPs

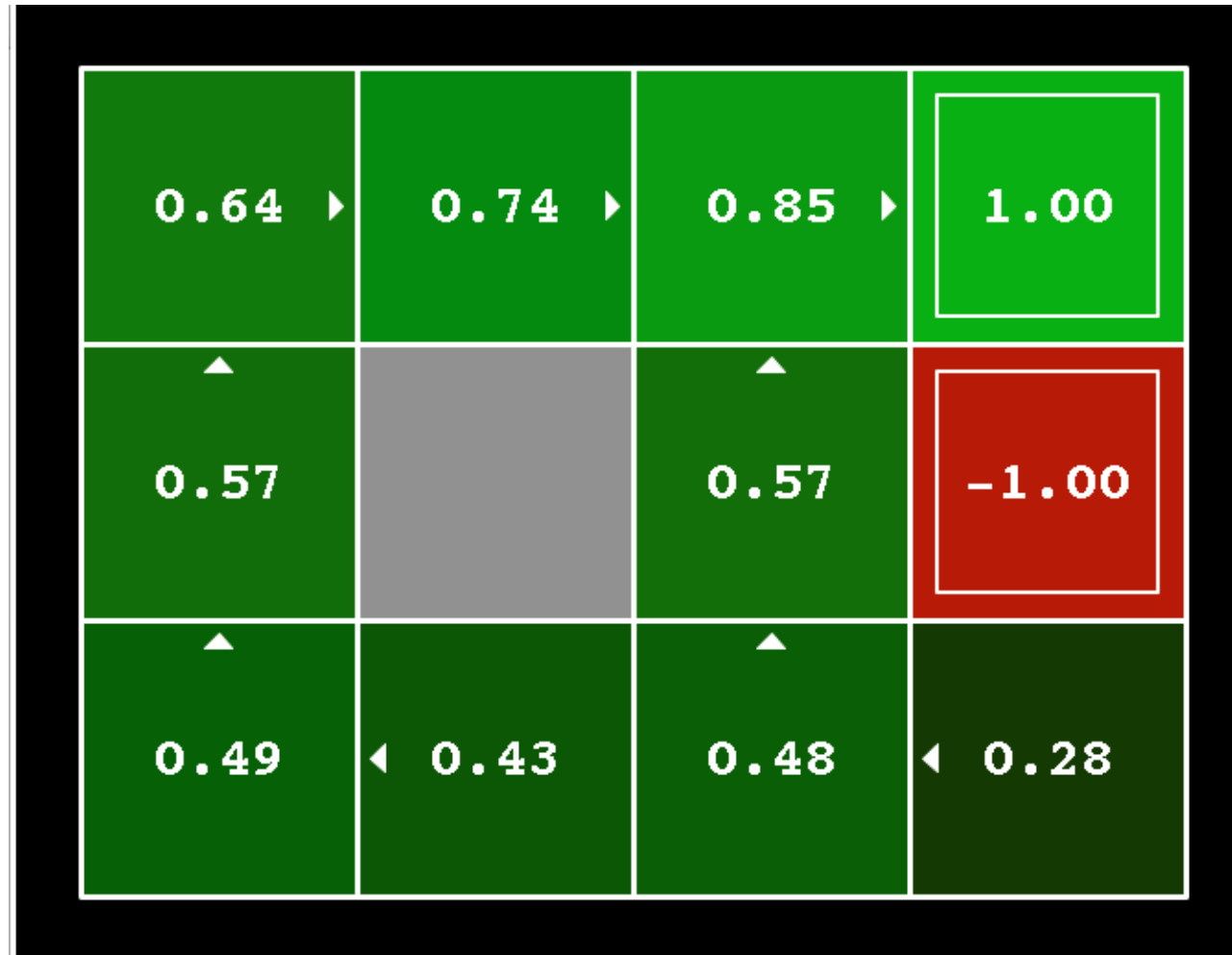


# Optimal Quantities

- 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  $s$

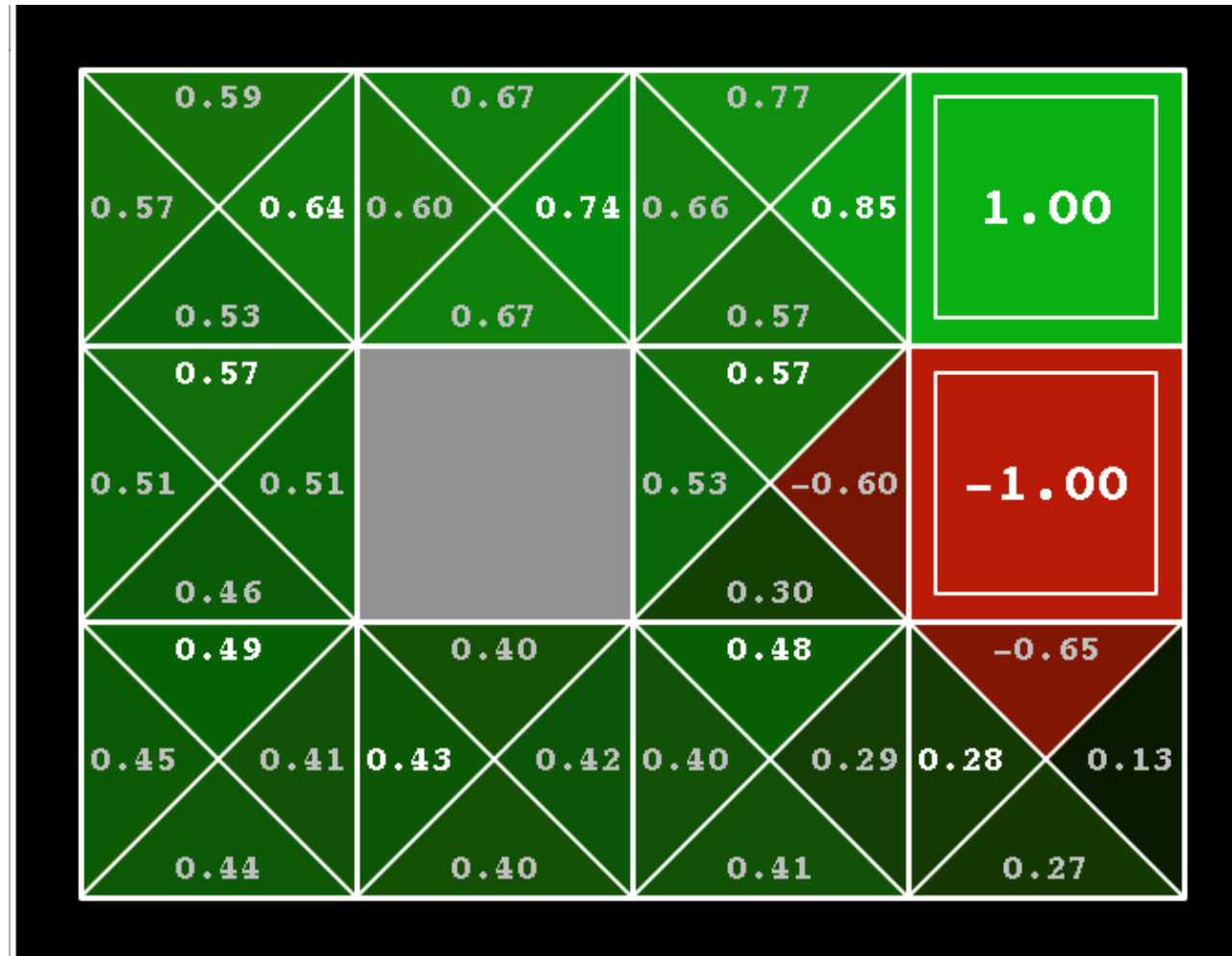


# Gridworld V Values



Noise = 0.2  
Discount = 0.9  
Living reward = 0

# Gridworld Q Values



Noise = 0.2  
Discount = 0.9  
Living reward = 0

# Values of States

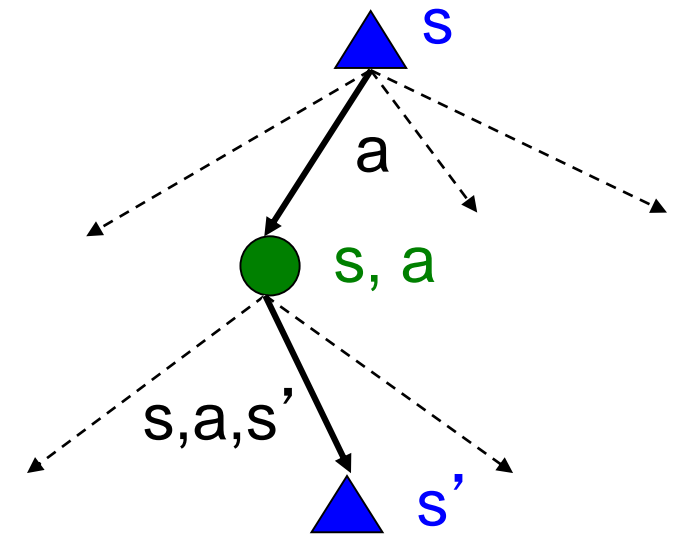
- How to compute the value of a state
  - Expected utility under optimal action
  - This is just what expectimax computed!

- Recursive definition of value:

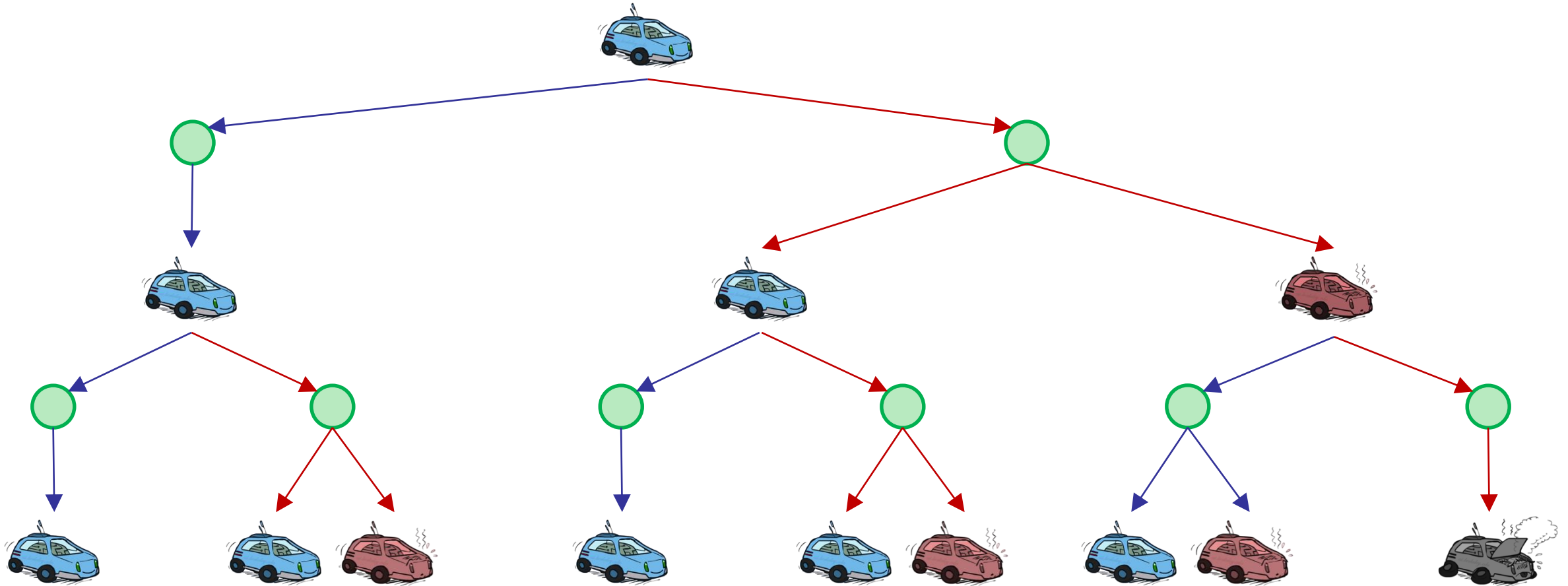
$$V^*(s) = \max_a Q^*(s, a)$$

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

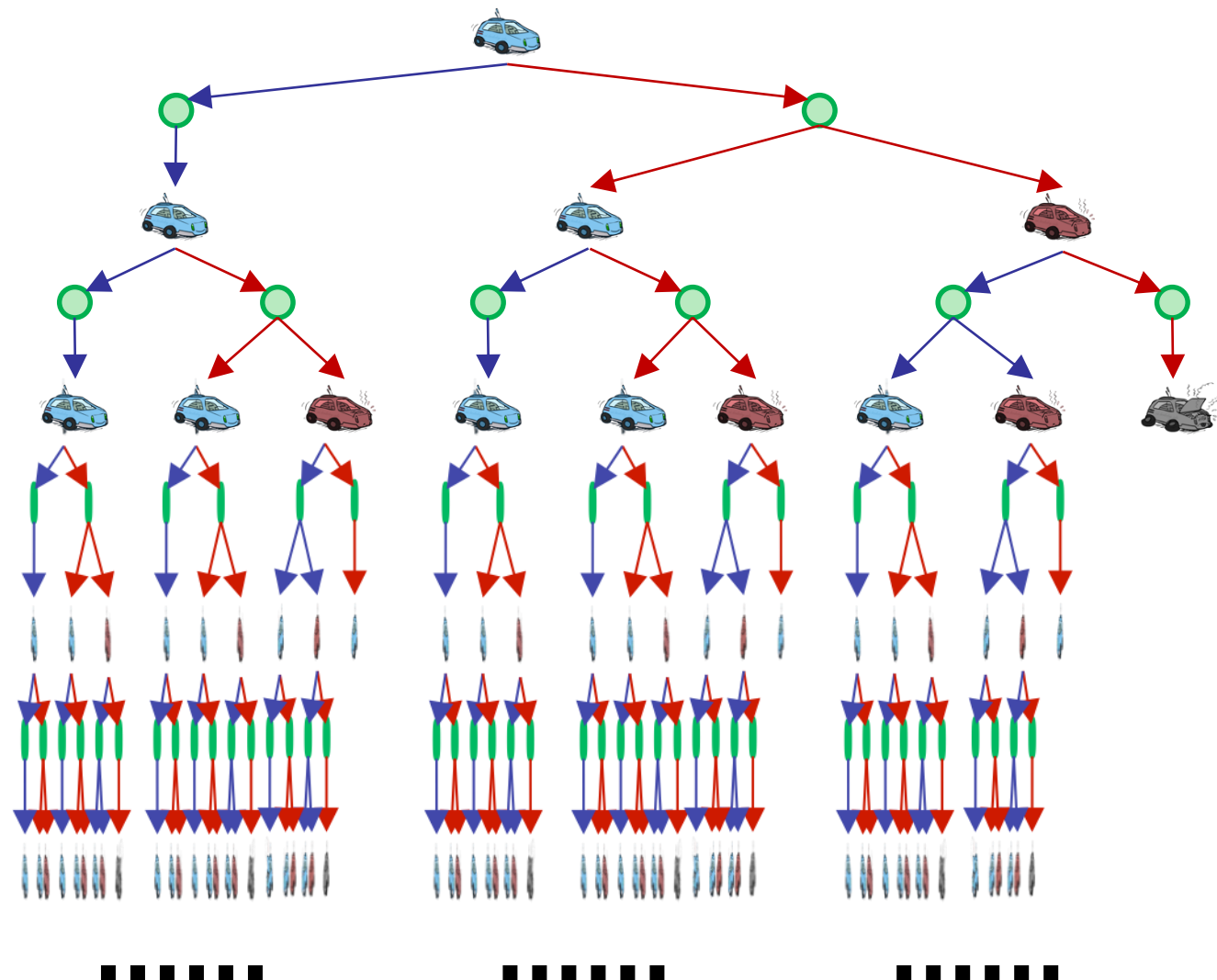
$$V^*(s) = \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')]$$



The Bellman Equation



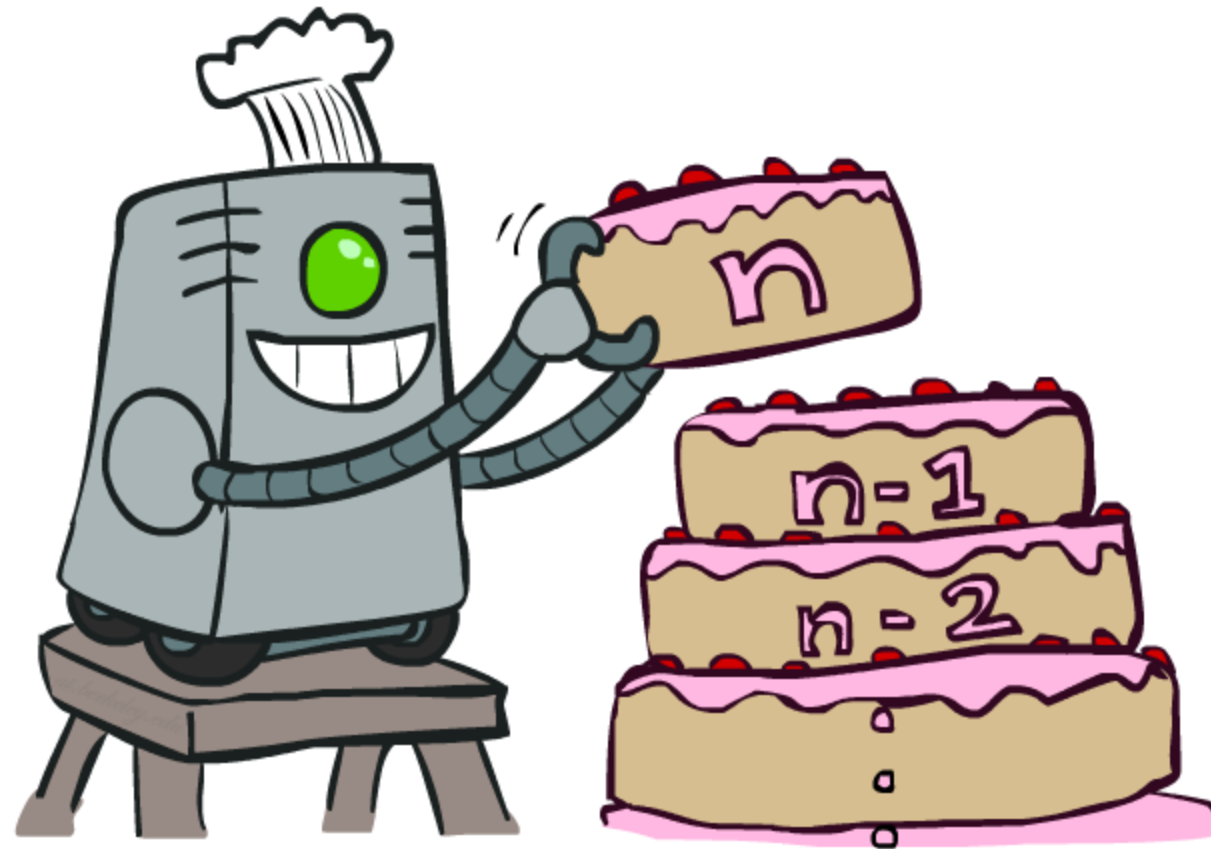






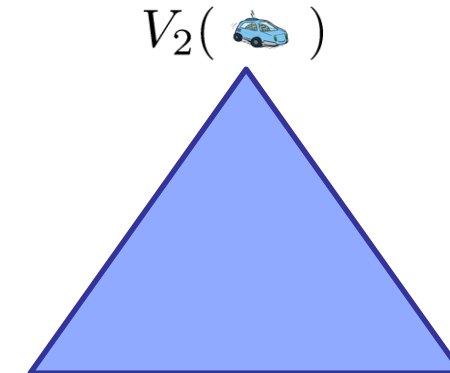
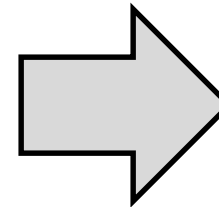
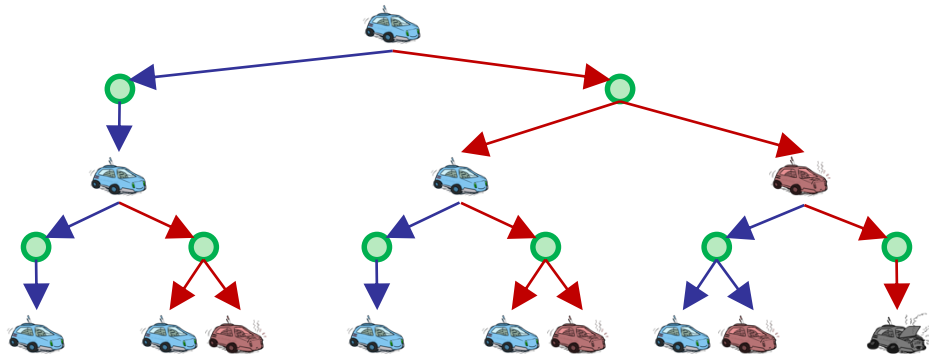
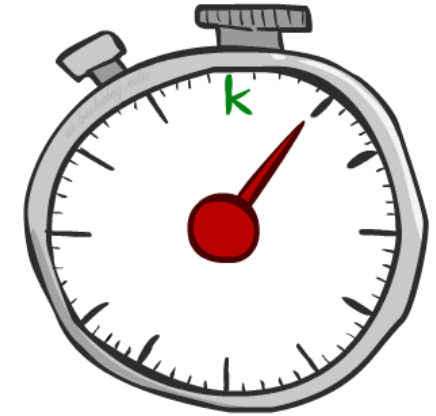
# Value Iteration

---



# Time-Limited Values

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

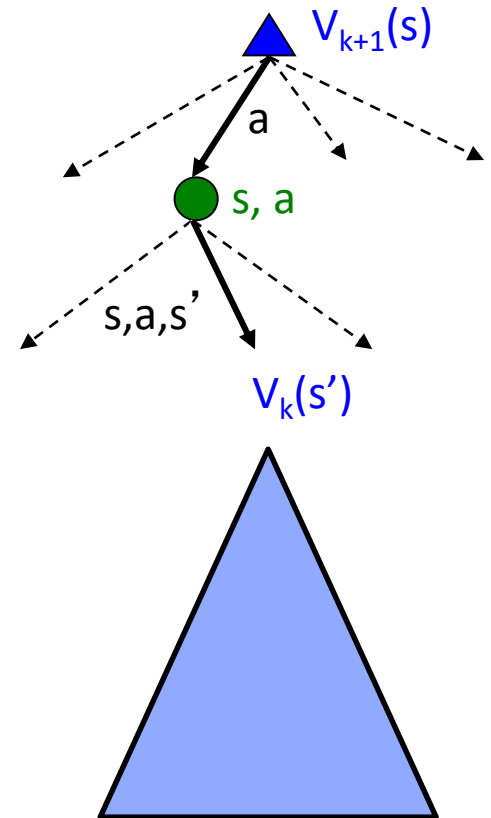


# Value Iteration

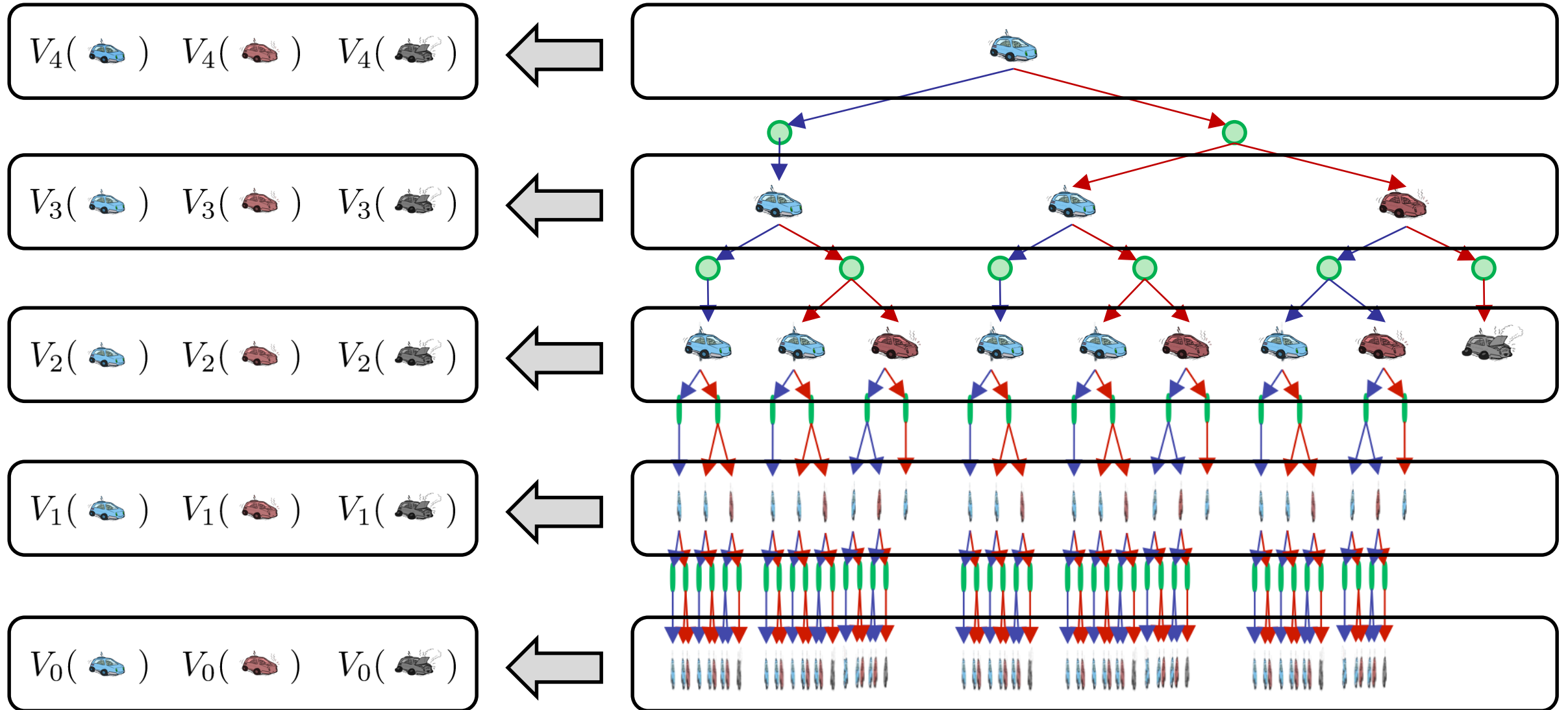
- Start with  $V_0(s) = 0$ : no time steps left means an expected reward sum of zero
- Given vector of  $V_k(s)$  values, do one ply of expectimax from each state:

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

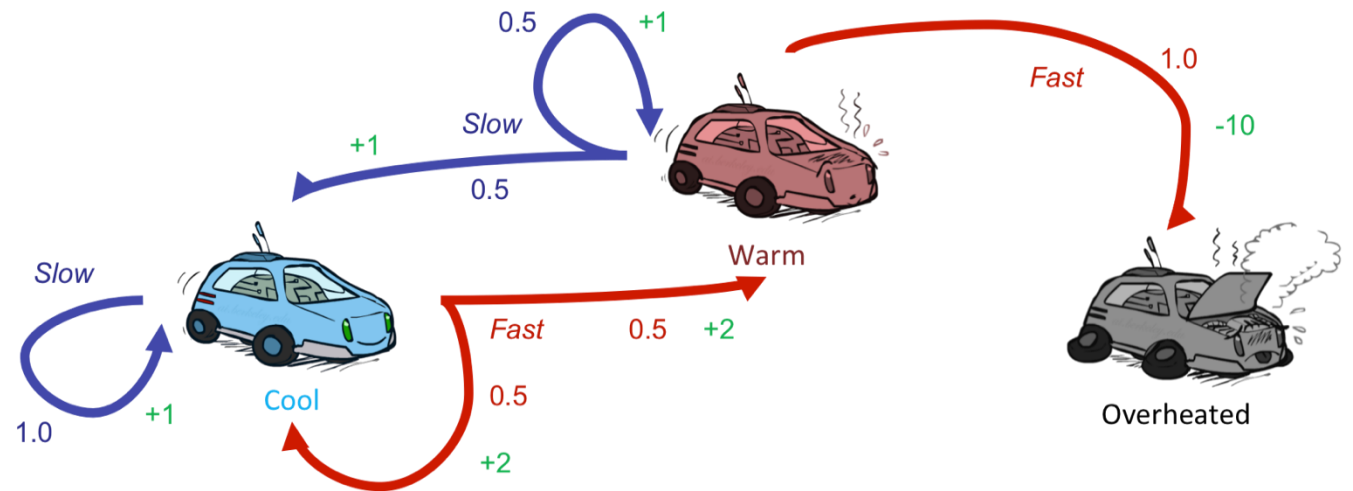
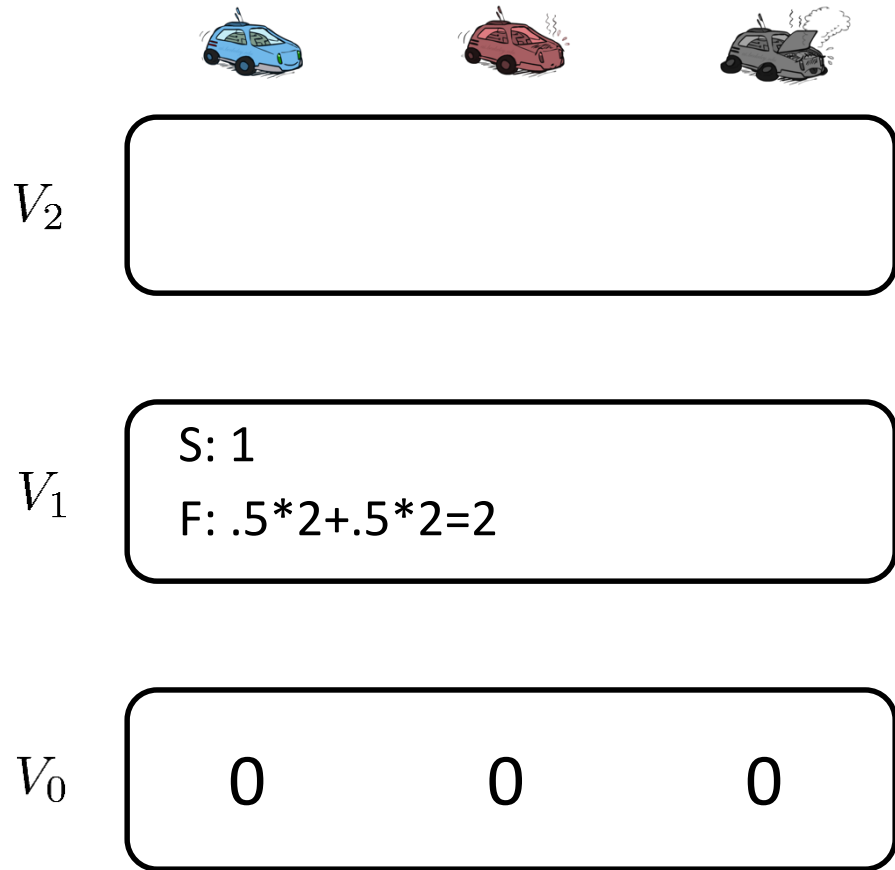
- Repeat until convergence
- Complexity of each iteration:  $O(S^2A)$
- Theorem: will converge to unique optimal values



# Computing Time-Limited Values






# Example: Value Iteration

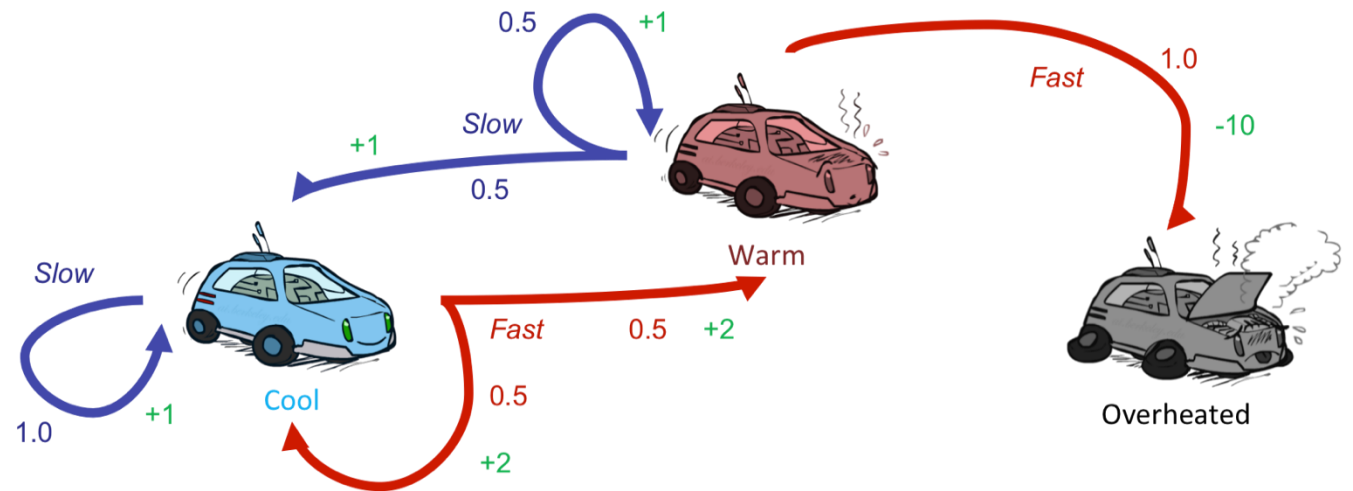


Assume no discount!

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

# Example: Value Iteration

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






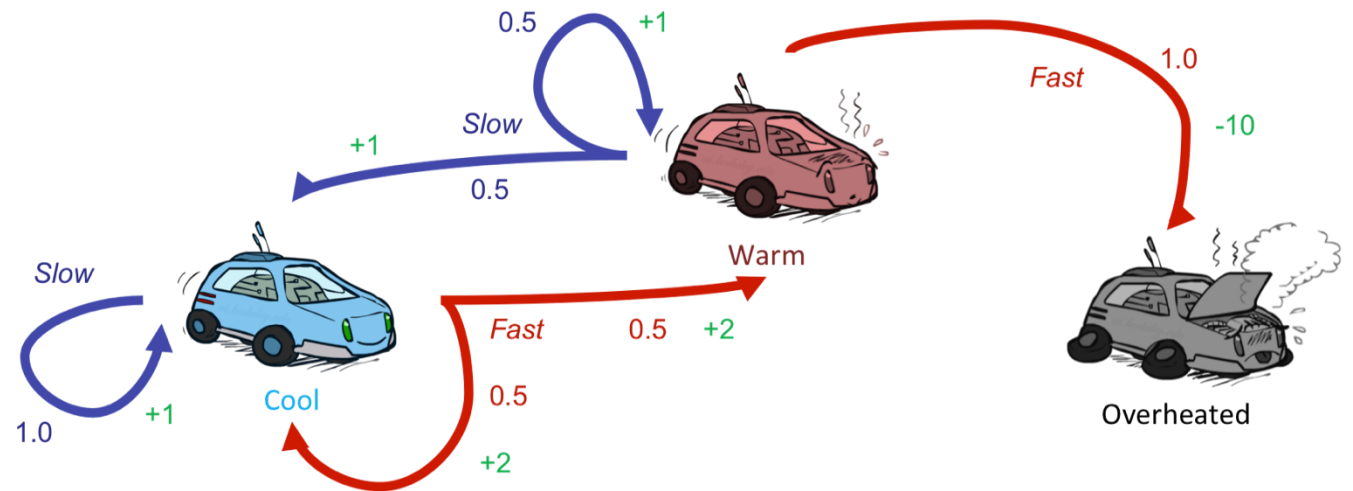
Assume no discount!

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



# Example: Value Iteration

			
$V_2$			
$V_1$	2	1	0
$V_0$	0	0	0



Assume no discount!

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

# Example: Value Iteration



$V_2$

S:  $1+2=3$

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

$V_1$

2

1

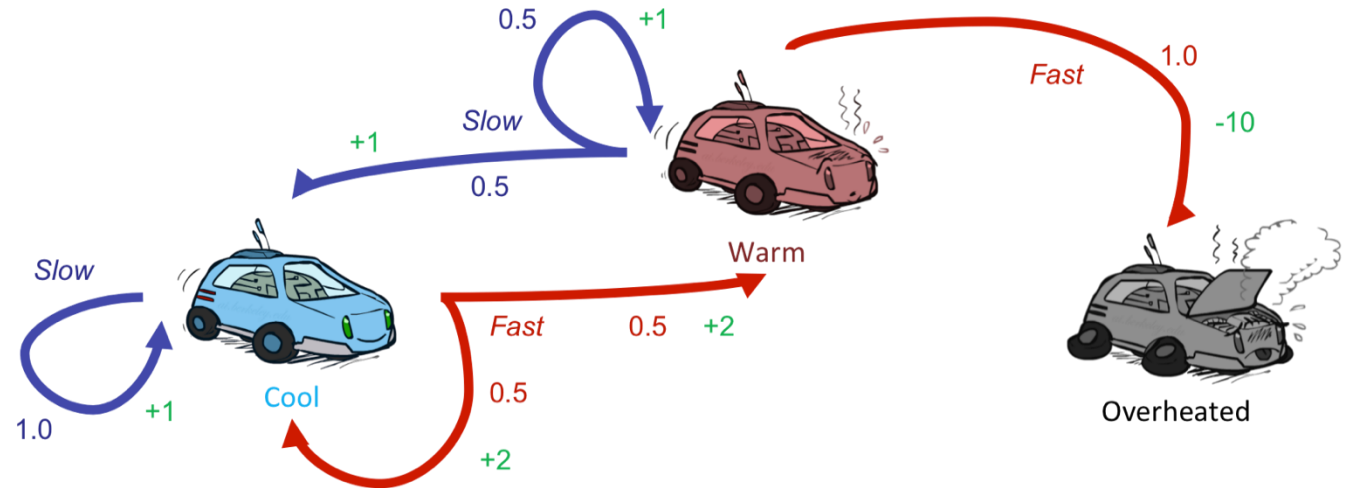
0

$V_0$

0

0




0

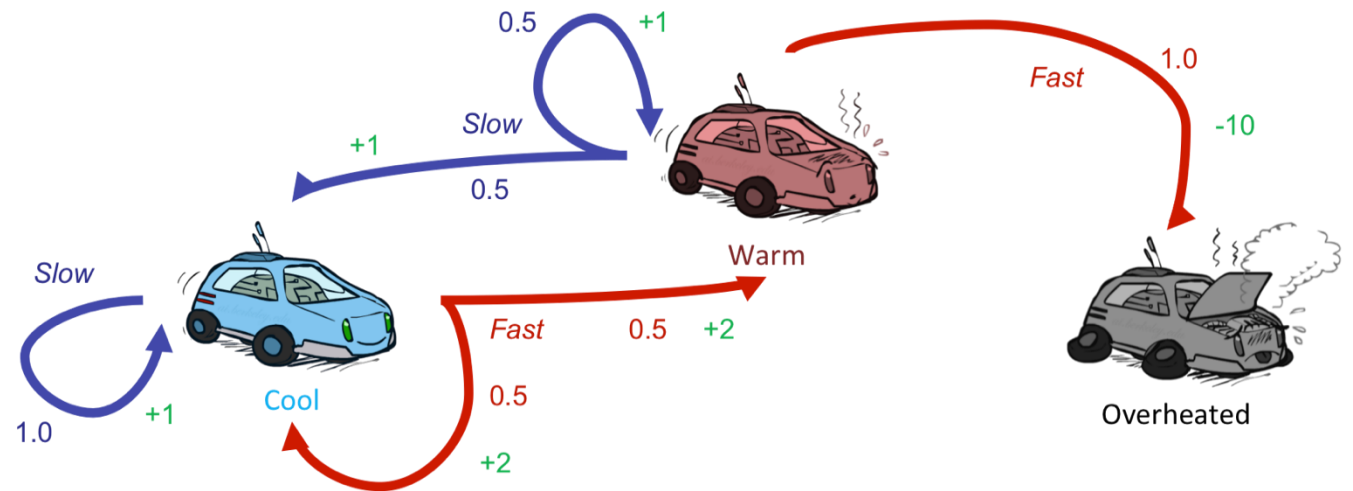


Assume no discount!

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

# Example: Value Iteration




			
$V_2$	<div> <div>3.5</div> <div>           S: <math>.5*(2+1)+.5*(1+1)=2.5</math>            F: -10         </div> </div>		
$V_1$	2	1	0
$V_0$	0	0	0

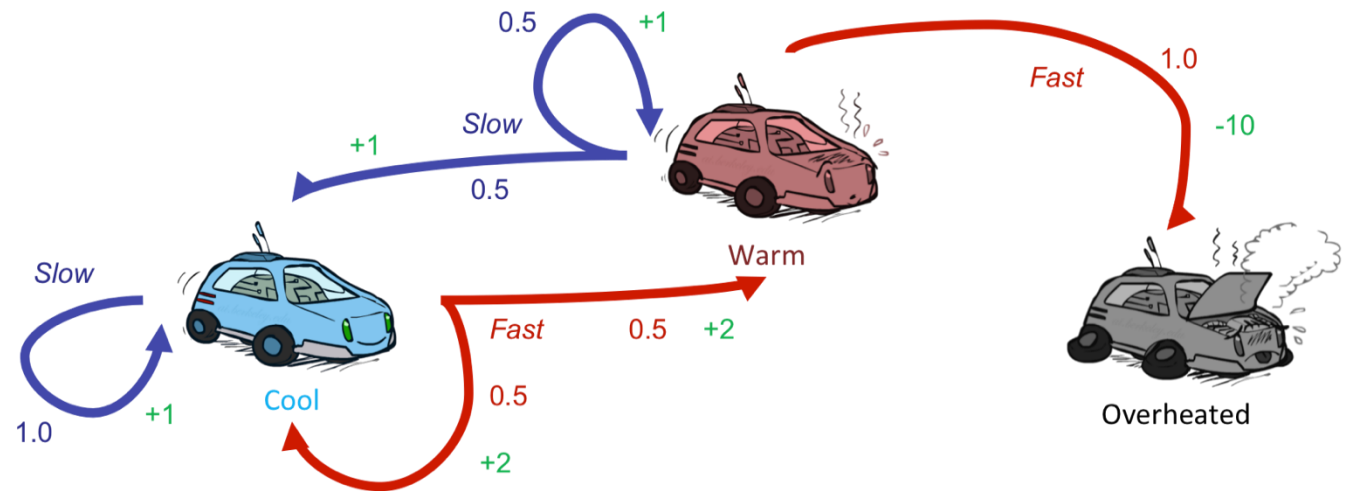


Assume no discount!

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

# Example: Value Iteration

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

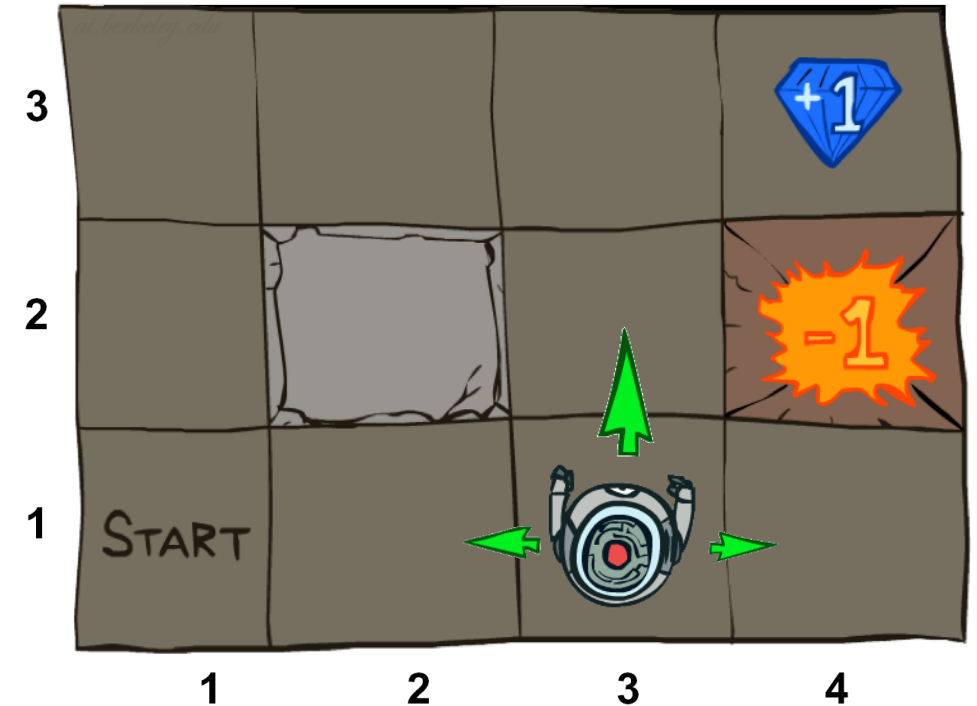


Assume no discount!

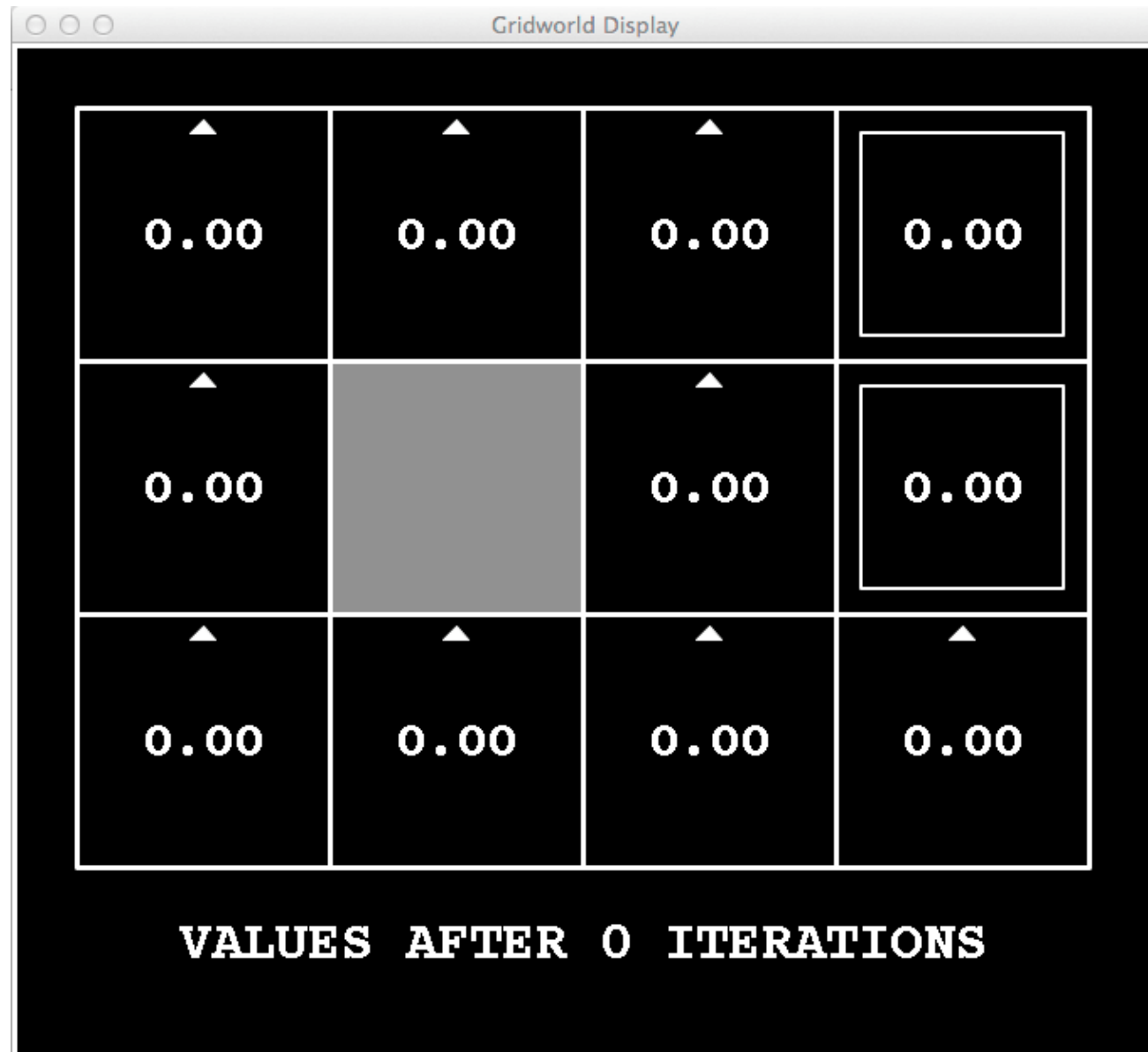
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_k(s')]$$

# Example: Grid World

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

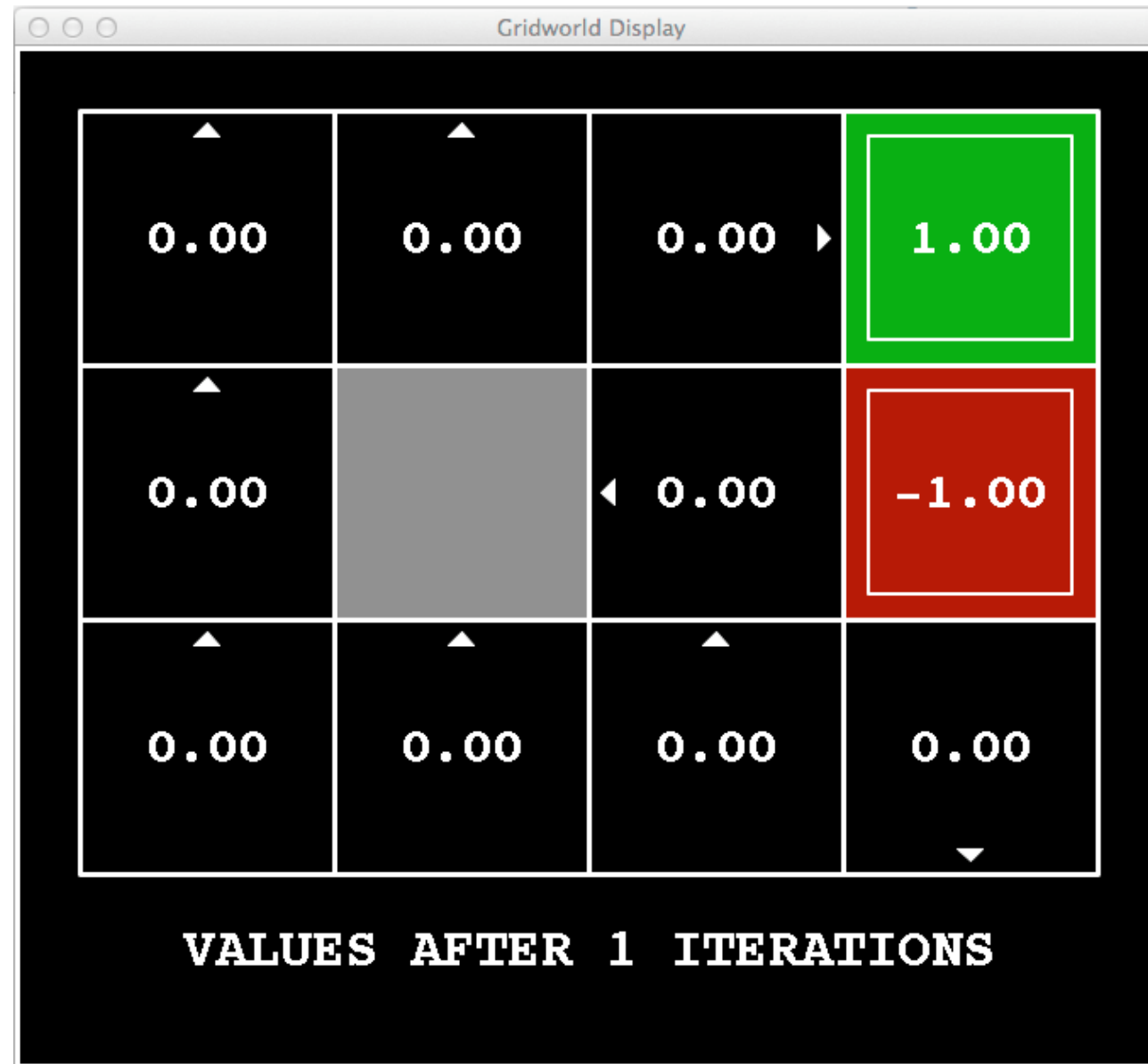


$k=0$



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



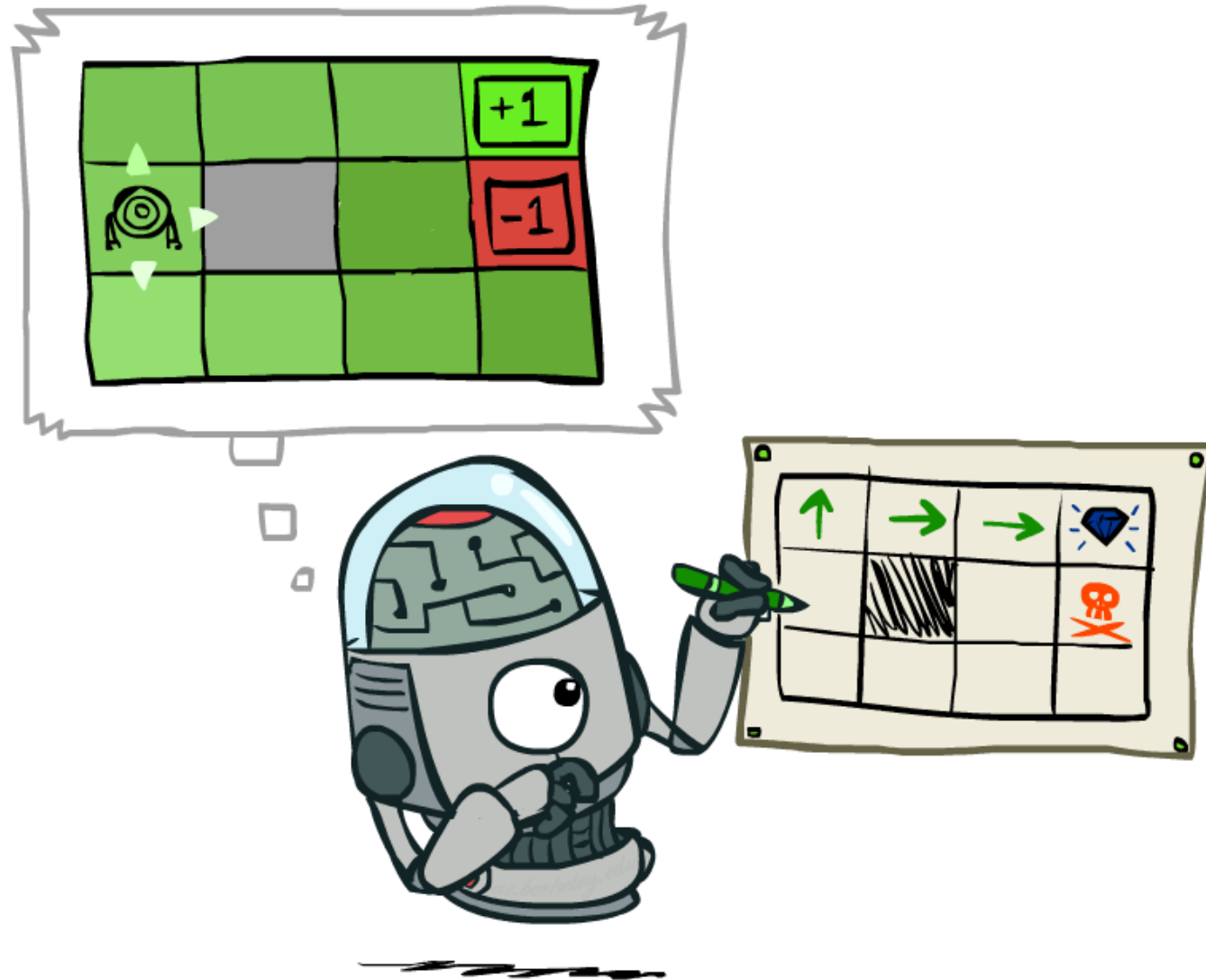
Noise = 0.2  
Discount = 0.9  
Living reward = 0

# k=100



Noise = 0.2  
Discount = 0.9  
Living reward = 0

# Policy 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)



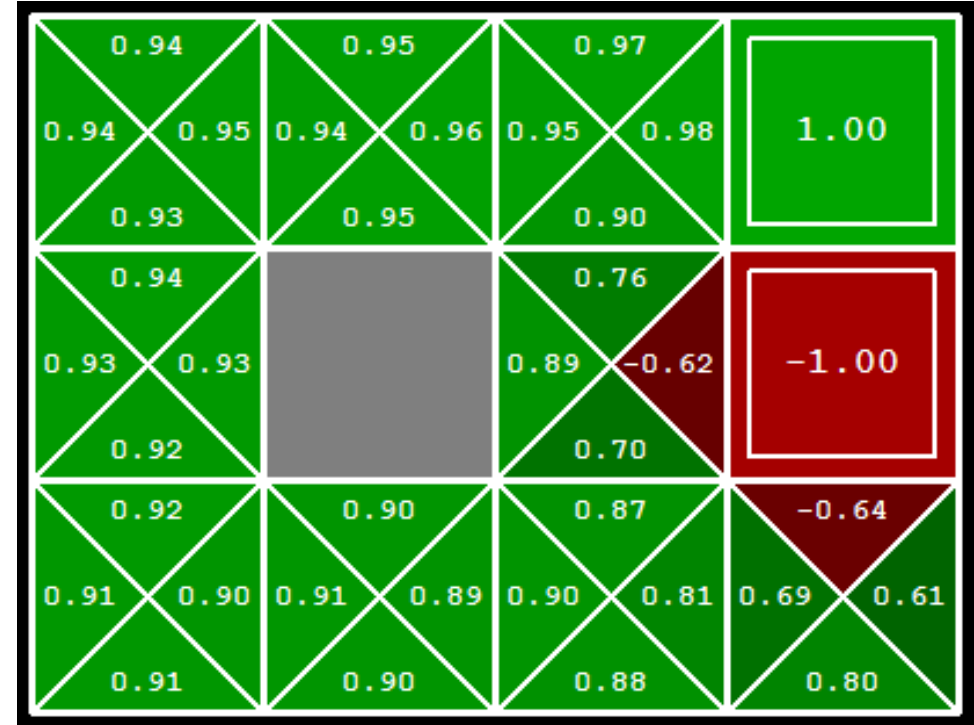
$$\pi^*(s) = \arg \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')]$$

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

# Computing Actions from Q-Values

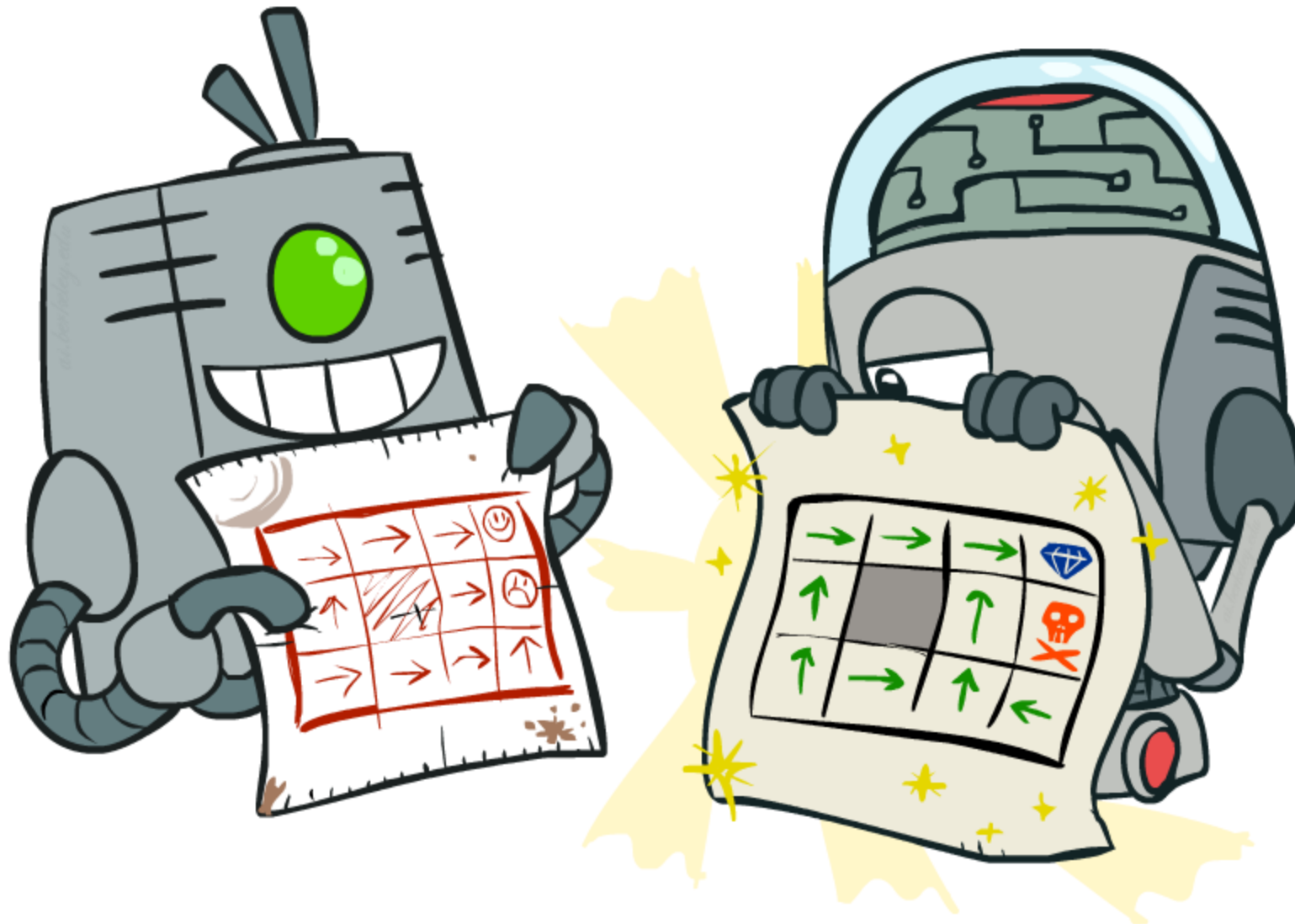
- Let's imagine we have the optimal q-values:
- How should we act?
  - Completely trivial to decide!

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



- Important: actions are easier to select from q-values than values!
- Q-values can also be computed in value iteration

# Policy Methods

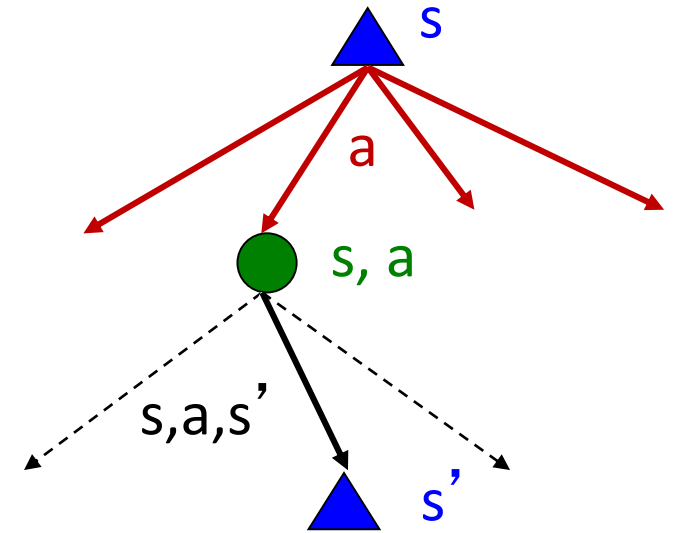


# Problems with Value Iteration

- Value iteration repeats the Bellman updates:

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

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





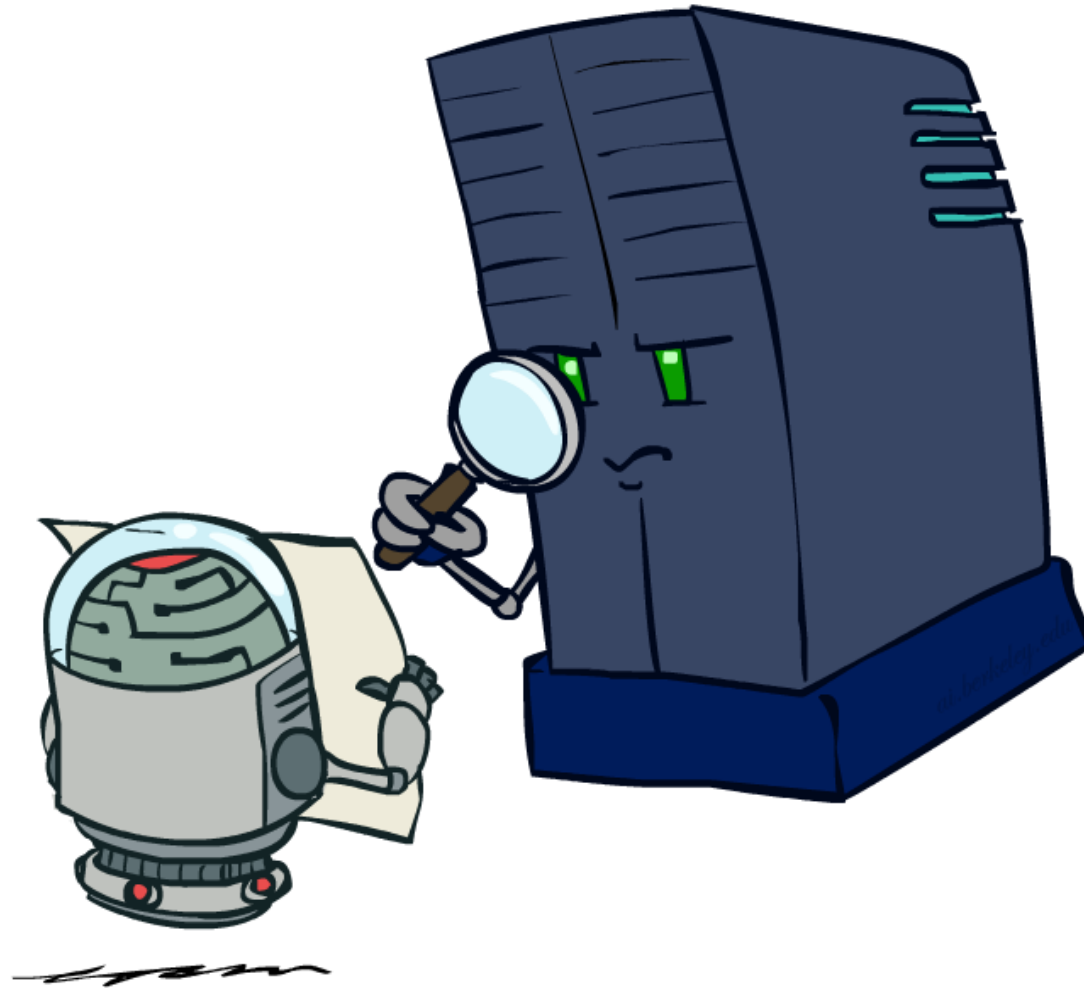
# Policy Iteration

---

- Alternative approach for optimal values:
  - **Step 1: Policy evaluation:** calculate utilities for some fixed (not optimal) policy
  - **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

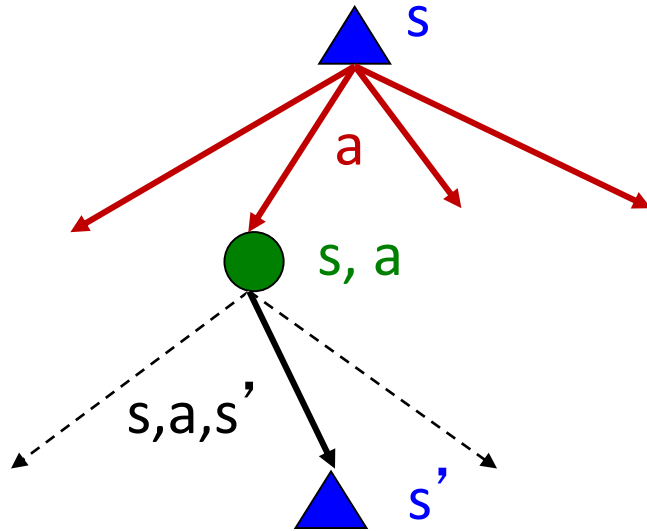
# Step 1: Policy Evaluation

---

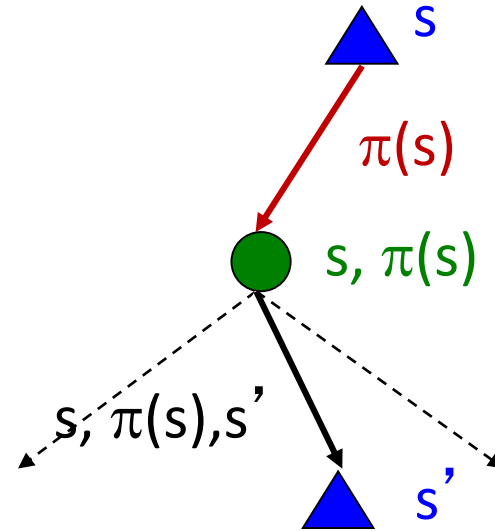


# Fixed Policies

Do the optimal action



Do what  $\pi$  says to do

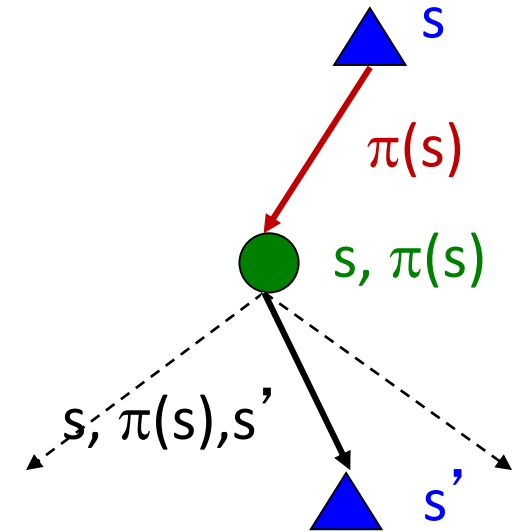


- Expectimax trees max over all actions to compute the optimal values
- If we fixed some policy  $\pi(s)$ , then the tree would be simpler – only one action per state

# Utilities for a Fixed Policy

- 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  $\pi$ :  
 $V^\pi(s)$  = expected total discounted rewards starting in  $s$  and following  $\pi$
- Recursive relation (one-step look-ahead):

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



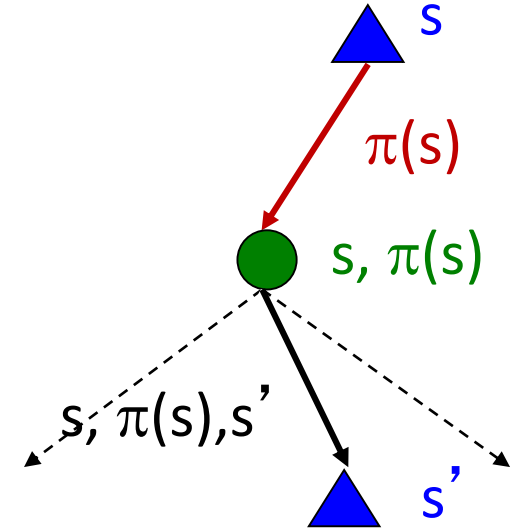
# Policy Evaluation

- How do we calculate the  $V$ 's for a fixed policy  $\pi$ ?

- Idea 1: Iterative updates (like value iteration)

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

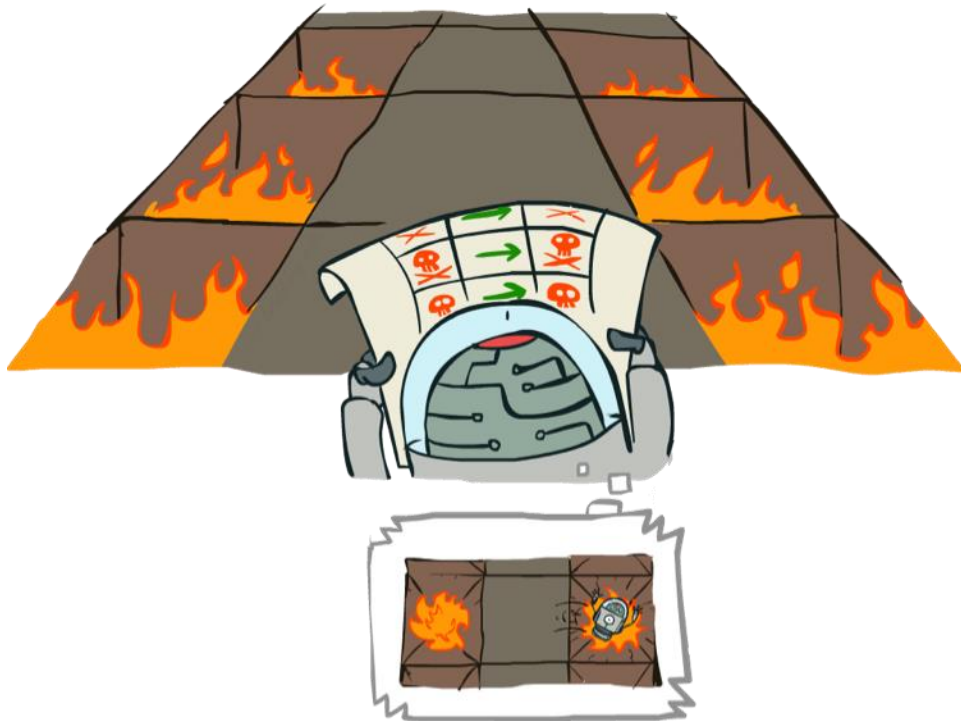
$$V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V_k^\pi(s')]$$



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

# Example: Policy Evaluation

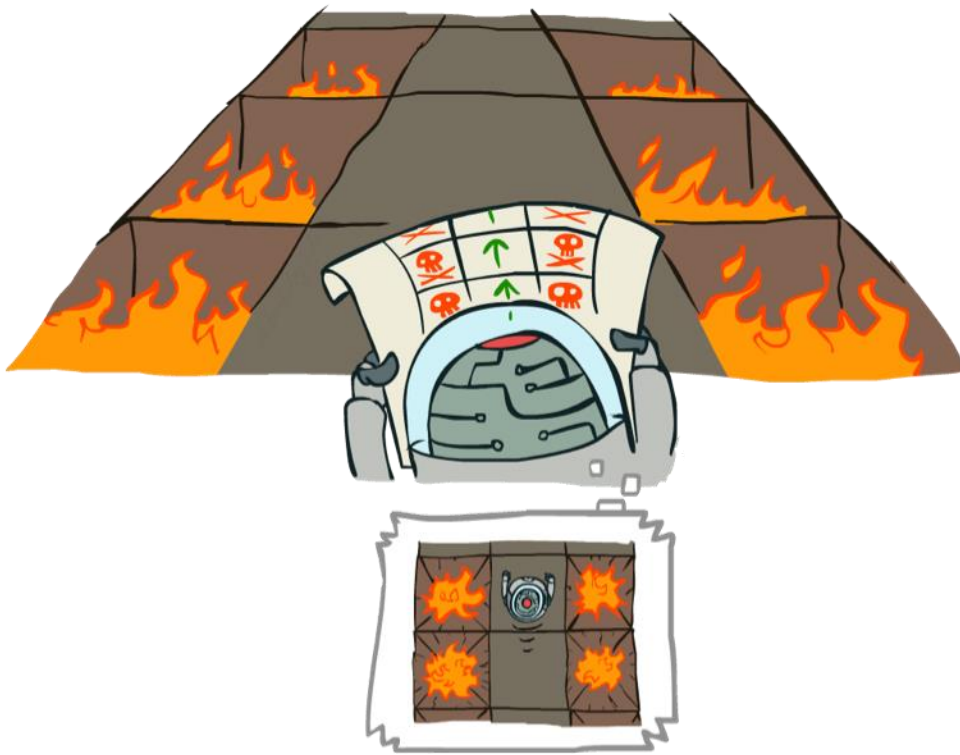
Always Go Right



-10.00	100.00	-10.00
-10.00	1.09 ▶	-10.00
-10.00	-7.88 ▶	-10.00
-10.00	-8.69 ▶	-10.00

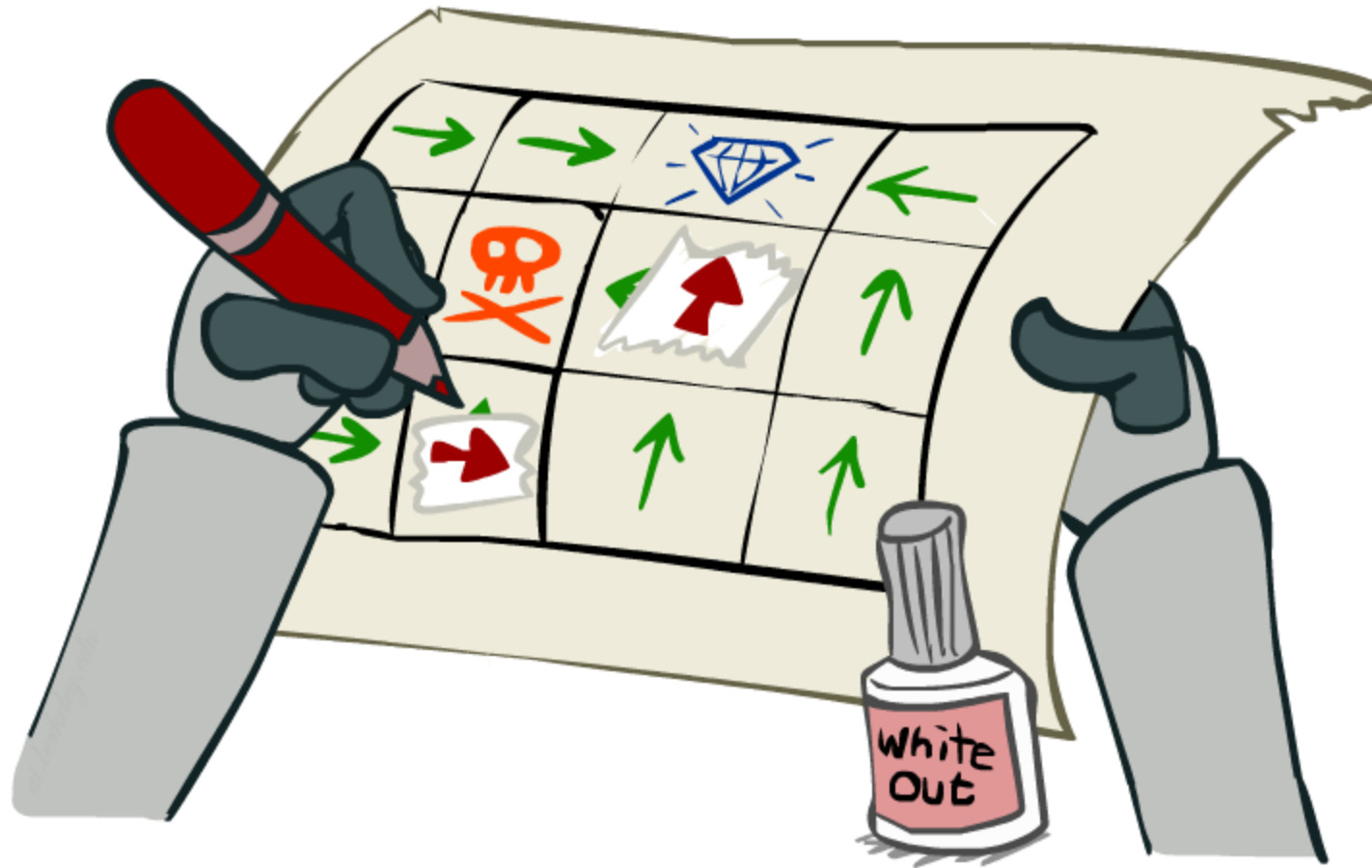
# Example: Policy Evaluation

Always Go Forward



-10.00	100.00	-10.00
-10.00	70.20	-10.00
-10.00	48.74	-10.00
-10.00	33.30	-10.00

# Step 2: Policy Improvement





# Policy Improvement

---

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

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

- Policy Iteration: repeat the two steps until policy converges

# Value Iteration vs. Policy Iteration

---

- 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

- Markov Decision Process
  - States  $S$ , Actions  $A$ , Transitions  $P(s' | s, a)$ , Rewards  $R(s, a, s')$
- Quantities:
  - Policy, Utility, Values, Q-Values
- Solve MDP
  - Value iteration
  - Policy iteration

