

Project

- Group registration due today
 - No more than 5 people in a group.
- Proposal presentation
 - Dec. 14, 16
 - 5-10min per group (depending on the number of groups)

Reinforcement Learning

- A new class of problems: Reward-based
 - E.g. Autonomous driving



- What is my next move?
 - Reaching the destination with minimum cost

Reinforcement Learning

- Common theme: control problems where
 - Your actions beget rewards
 - Win the Go game
 - Make money by investing
 - Get home sooner
 - But not deterministically
 - A world out there that is not predictable
- From experience of **belated/delayed** rewards, you must learn to act rationally

RL Problem Setting

- A cartoon of the world



- Agent operates in an environment
 - Agent may be you..
 - Environment is the game, the market, the road..

RL Problem Setting

- A cartoon of the world



- Agent takes actions which affect the environment

RL Problem Setting

- A cartoon of the world



- Agent takes actions which affect the environment
- Which changes in a somewhat unpredictable way

RL Problem Setting

- A cartoon of the world



- Agent takes actions which affect the environment
- Which changes in a somewhat unpredictable way
- Which affects the agent's situation

RL Problem Setting

- A cartoon of the world



- The agent also receives rewards..
- Which may be apparent immediately
- Or not apparent for a very long time

RL Problem Setting

- Problem to solve



- How must the agent behave to maximize its rewards?

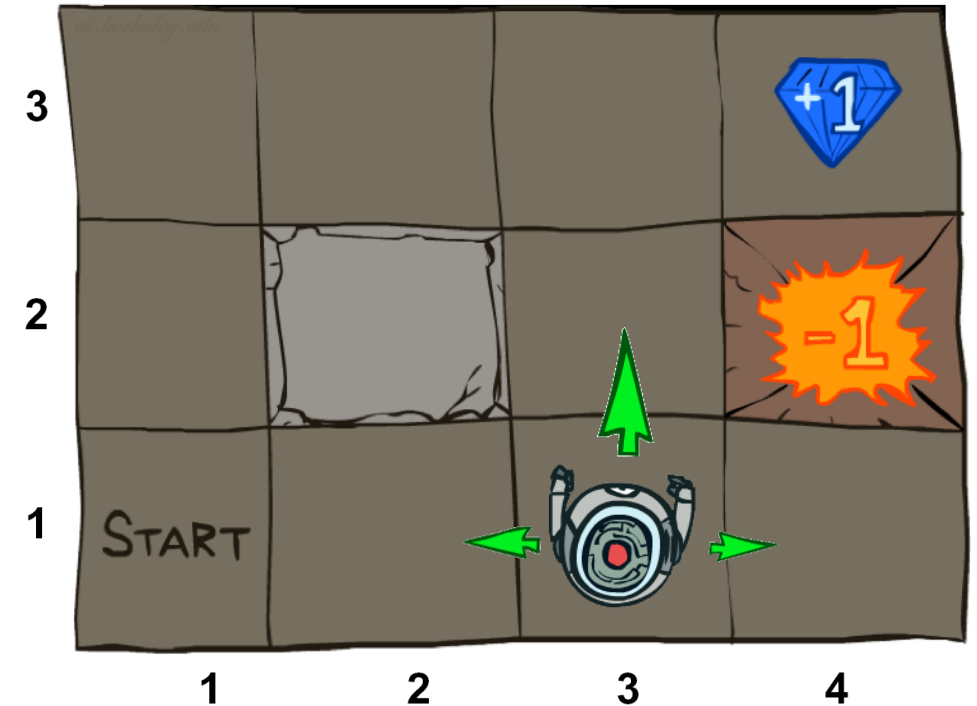
Markov Decision Processes



AIMA Chapter 17

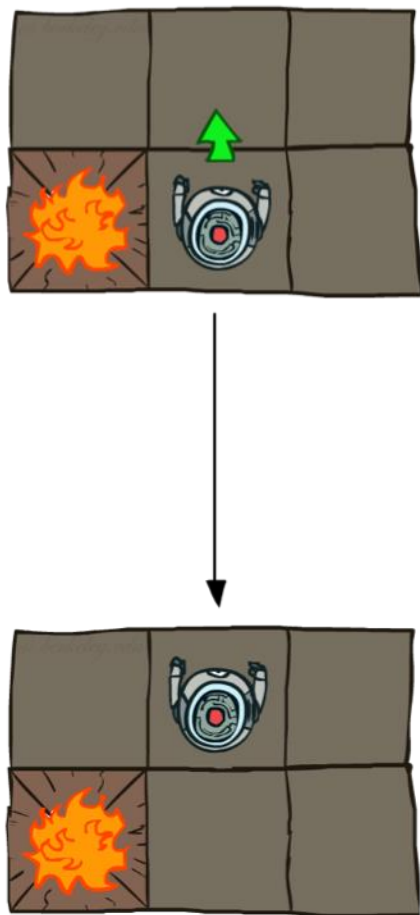
Example: Grid World

- 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

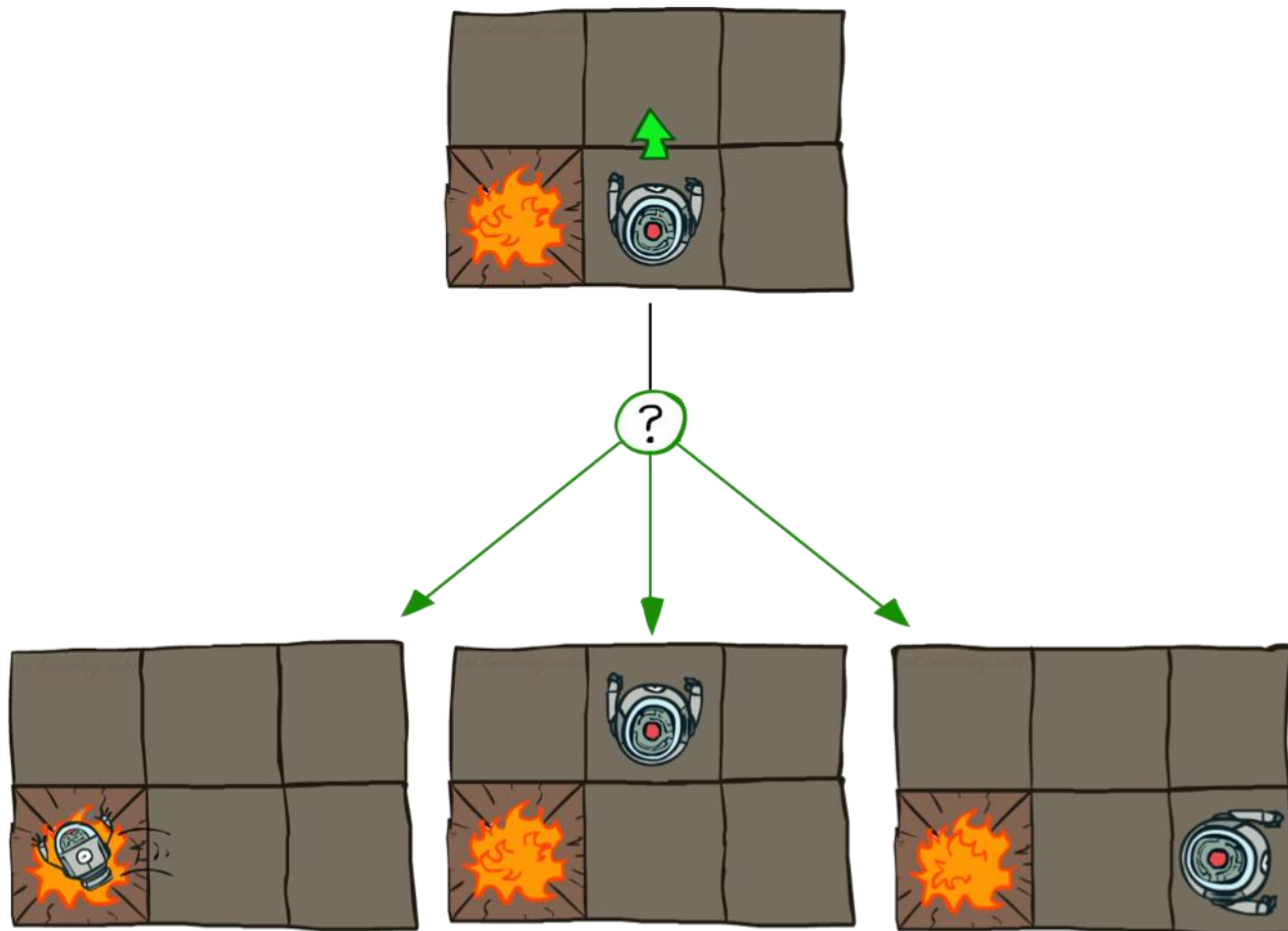


Grid World Actions

Deterministic Grid World

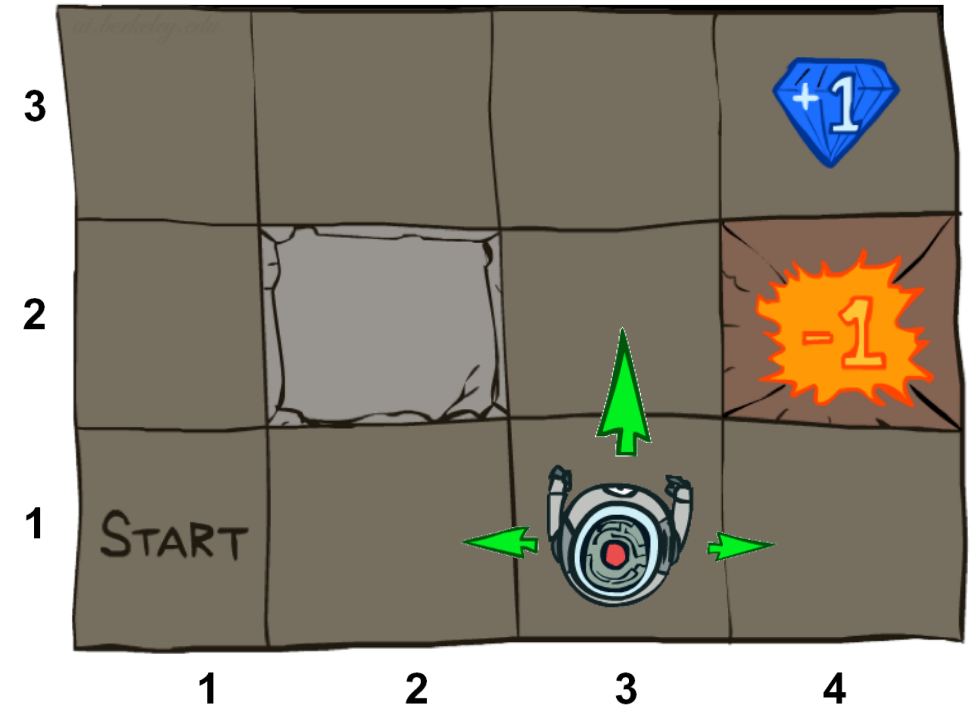


Stochastic Grid World



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

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

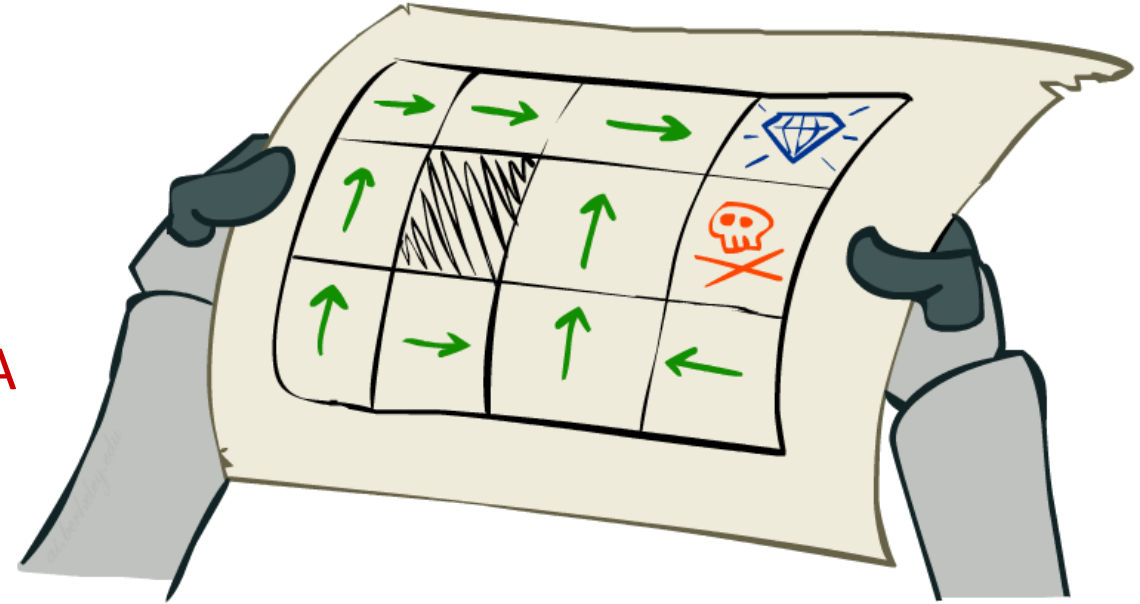
- This is just like search, where the successor function could only depend on the current state (not the history)



Andrey Markov
(1856-1922)

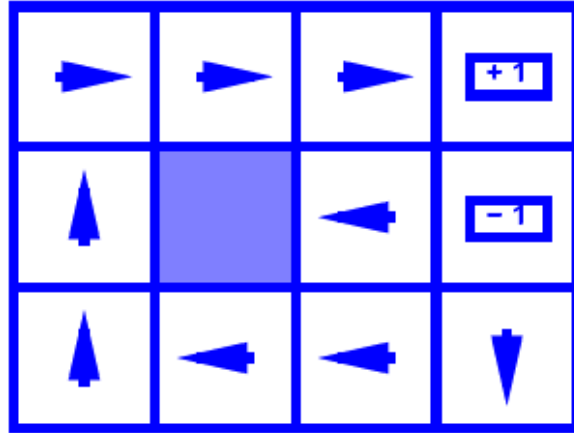
Policies

- In deterministic single-agent search problems, we wanted an optimal **plan**, or sequence of actions, from start to a goal
- For MDPs, we want an optimal **policy** $\pi^*: S \rightarrow A$
 - A policy 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

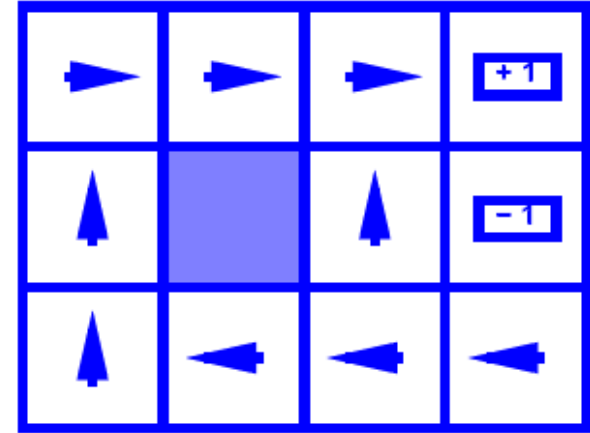


Optimal Policies

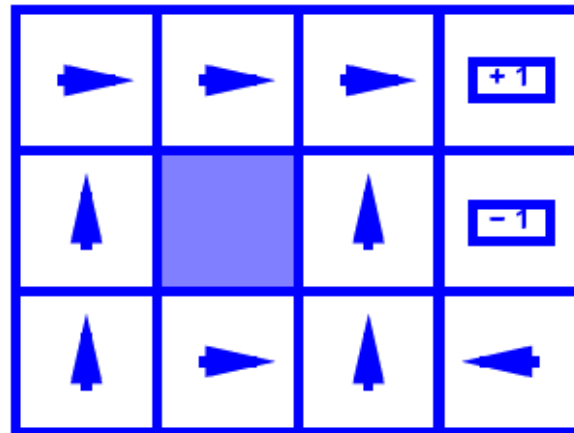
$R(s)$ = “living reward”



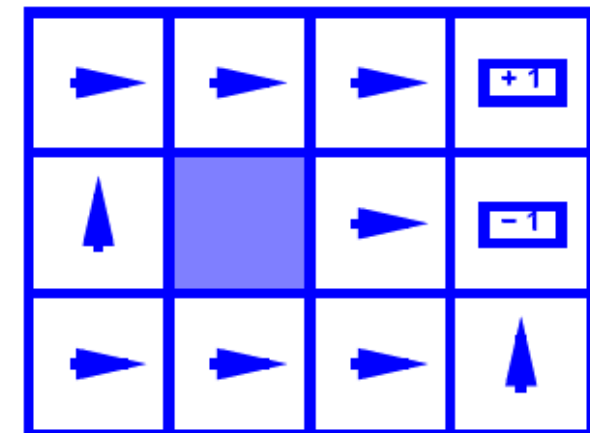
$R(s) = -0.01$



$R(s) = -0.03$



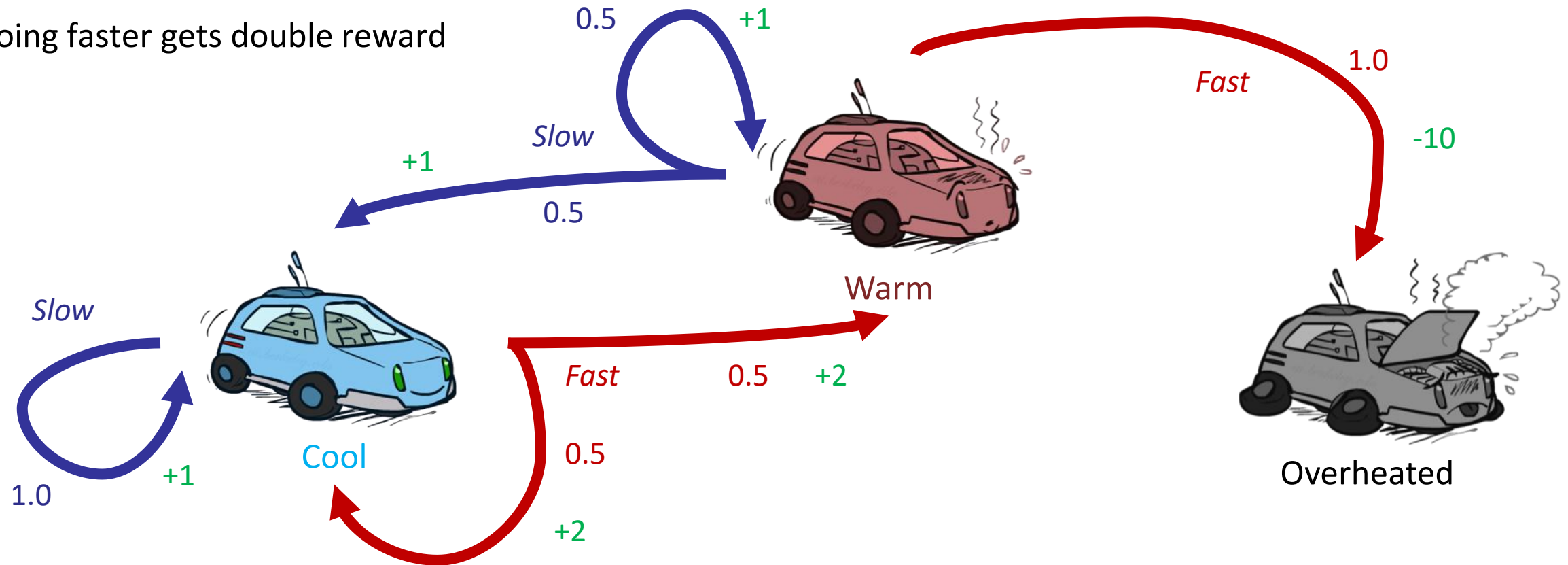
$R(s) = -0.4$

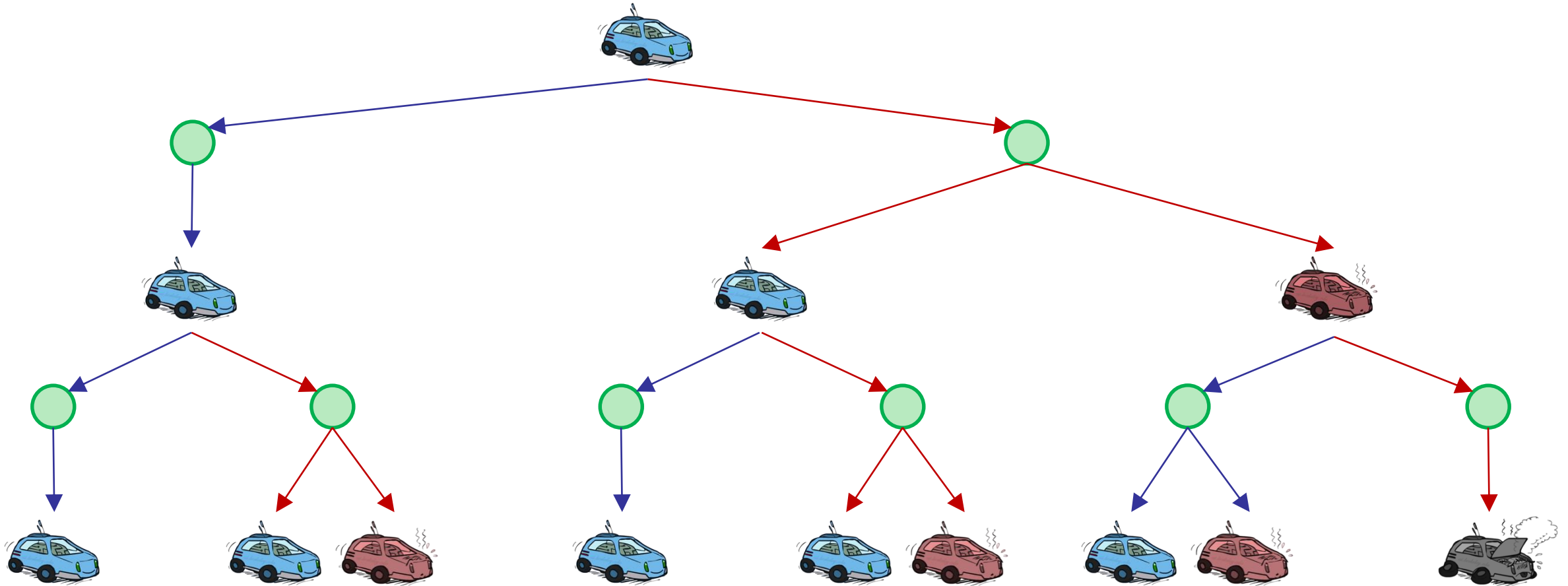


$R(s) = -2.0$

Example: Racing

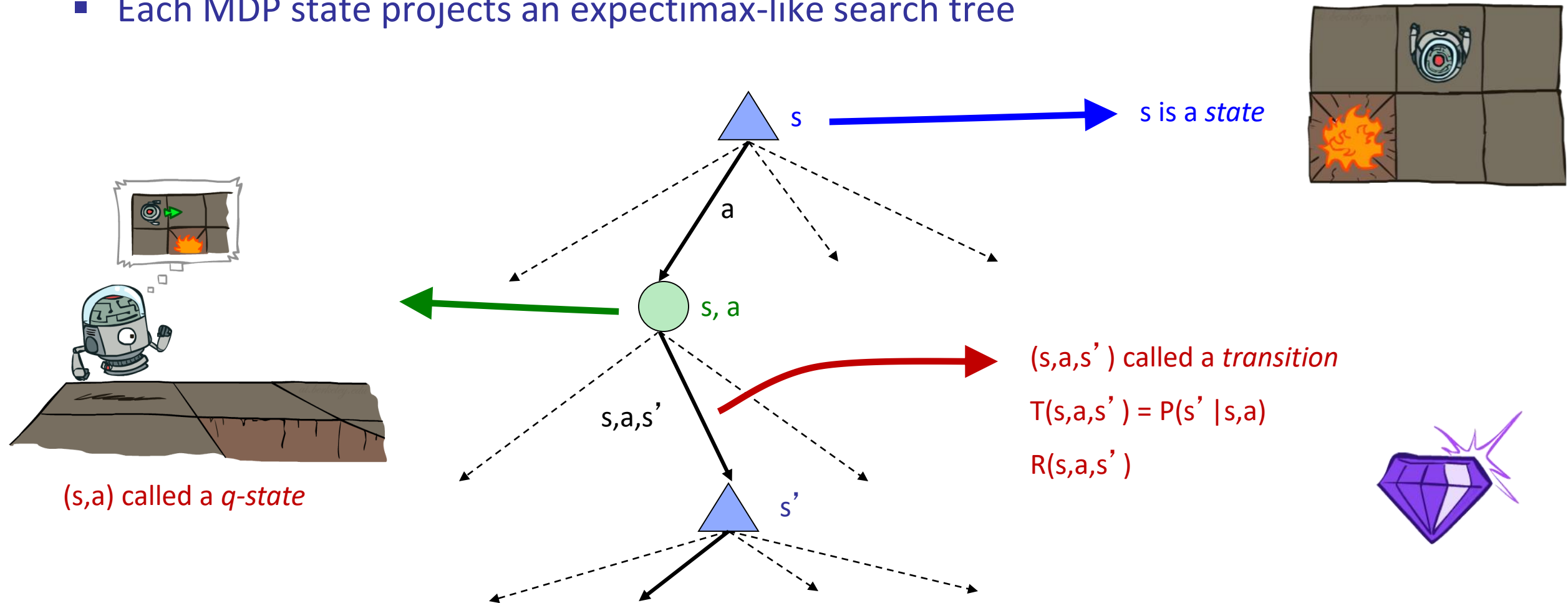
- A robot car wants to travel far, quickly
- Three states: **Cool**, **Warm**, Overheated
- Two actions: **Slow**, **Fast**
- Going faster gets double reward



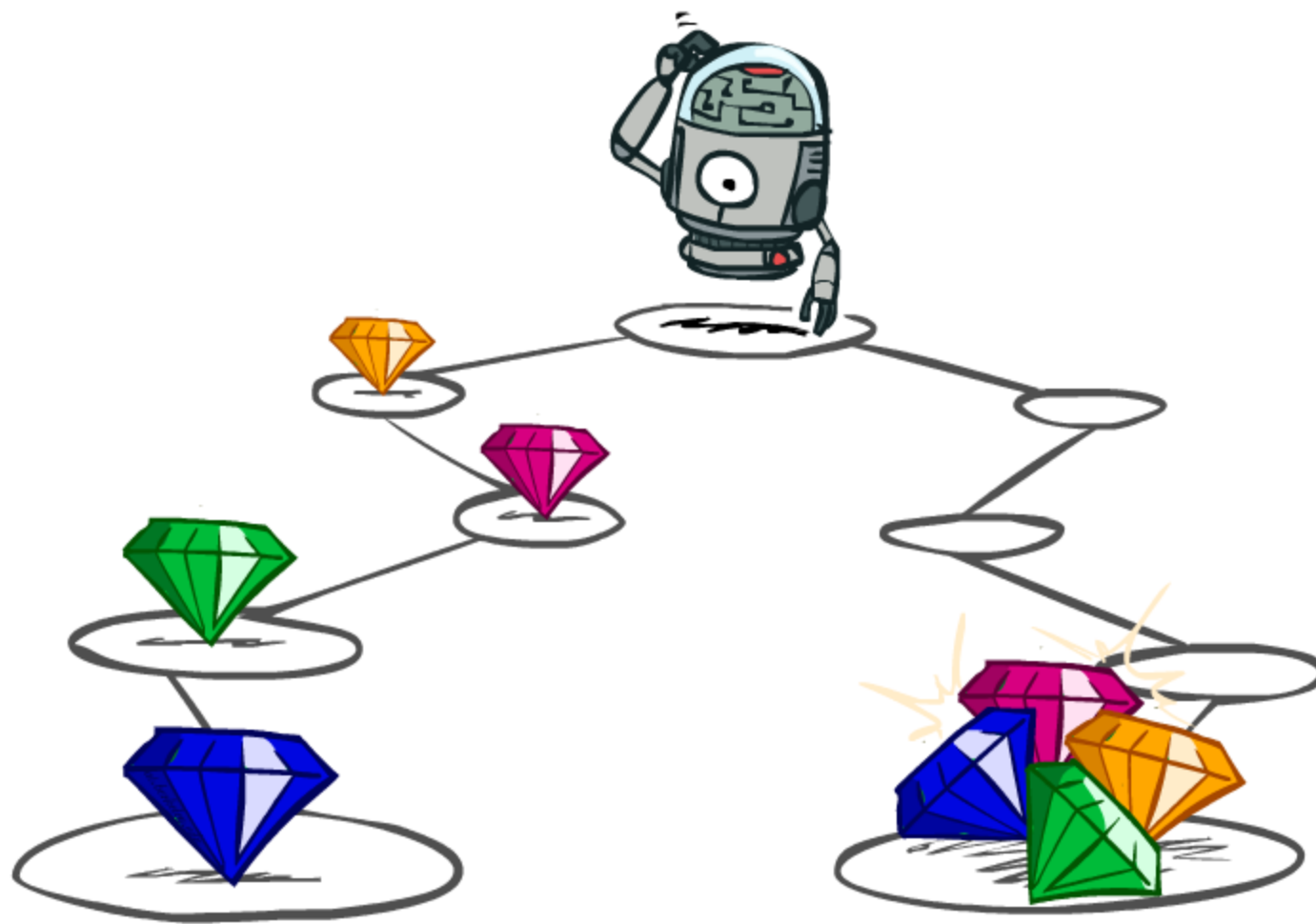


MDP Search Trees

- Each MDP state projects an expectimax-like search tree

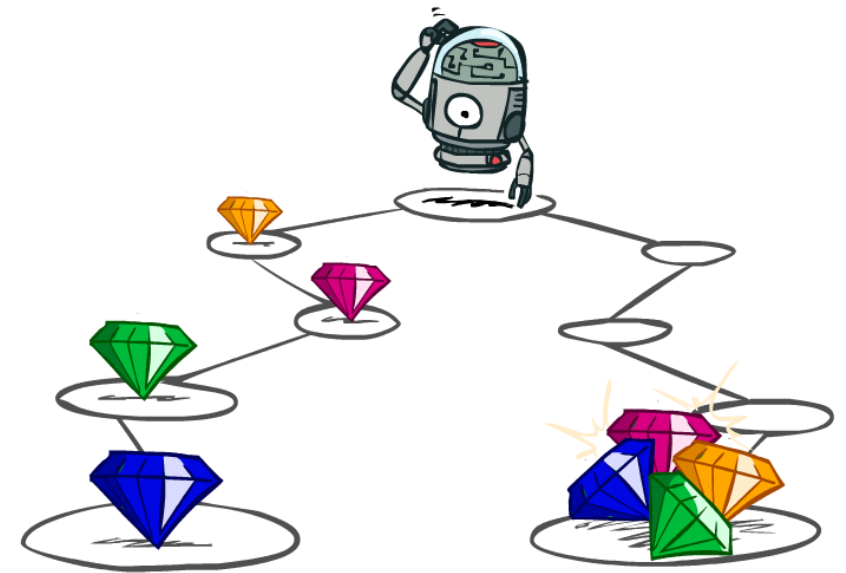


Utilities of Sequences



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



γ

Worth Next Step



γ^2

Worth In Two Steps

Discounting

- How to discount?

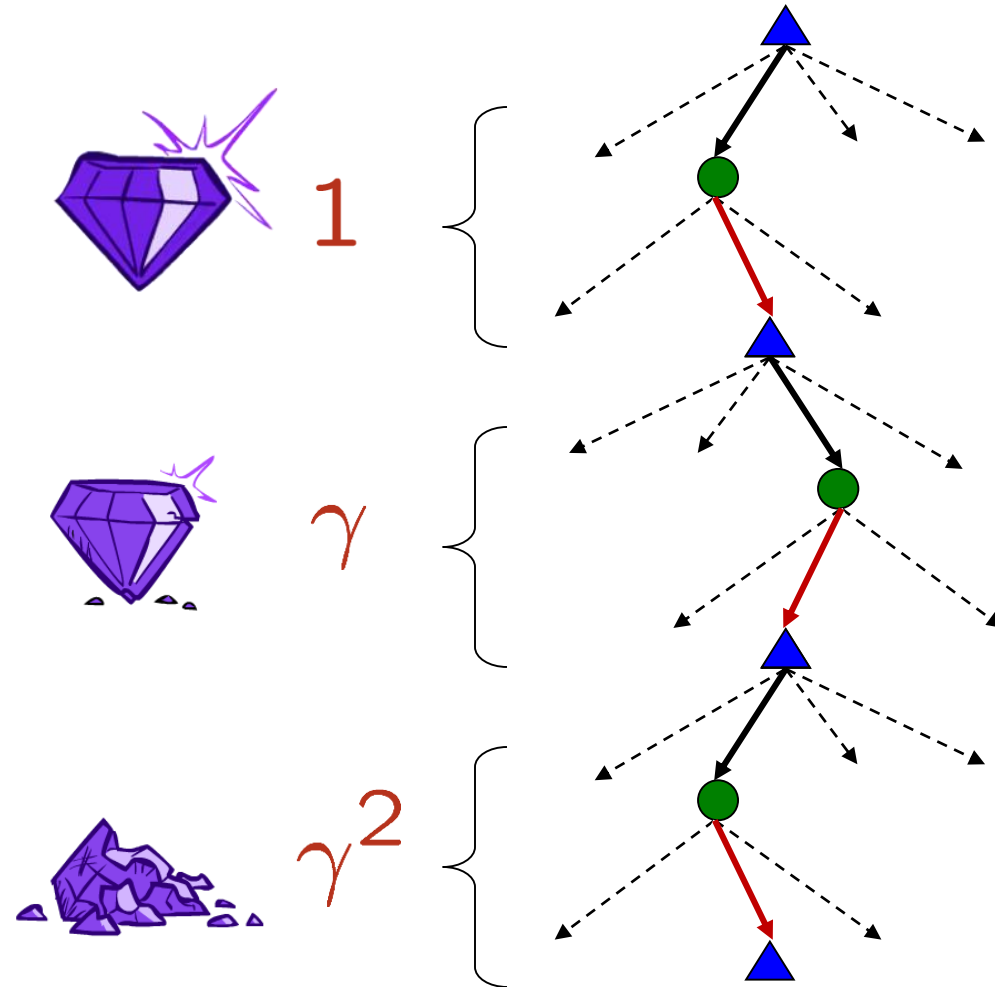
- Each time we descend a level, we multiply in the discount once

- Why discount?

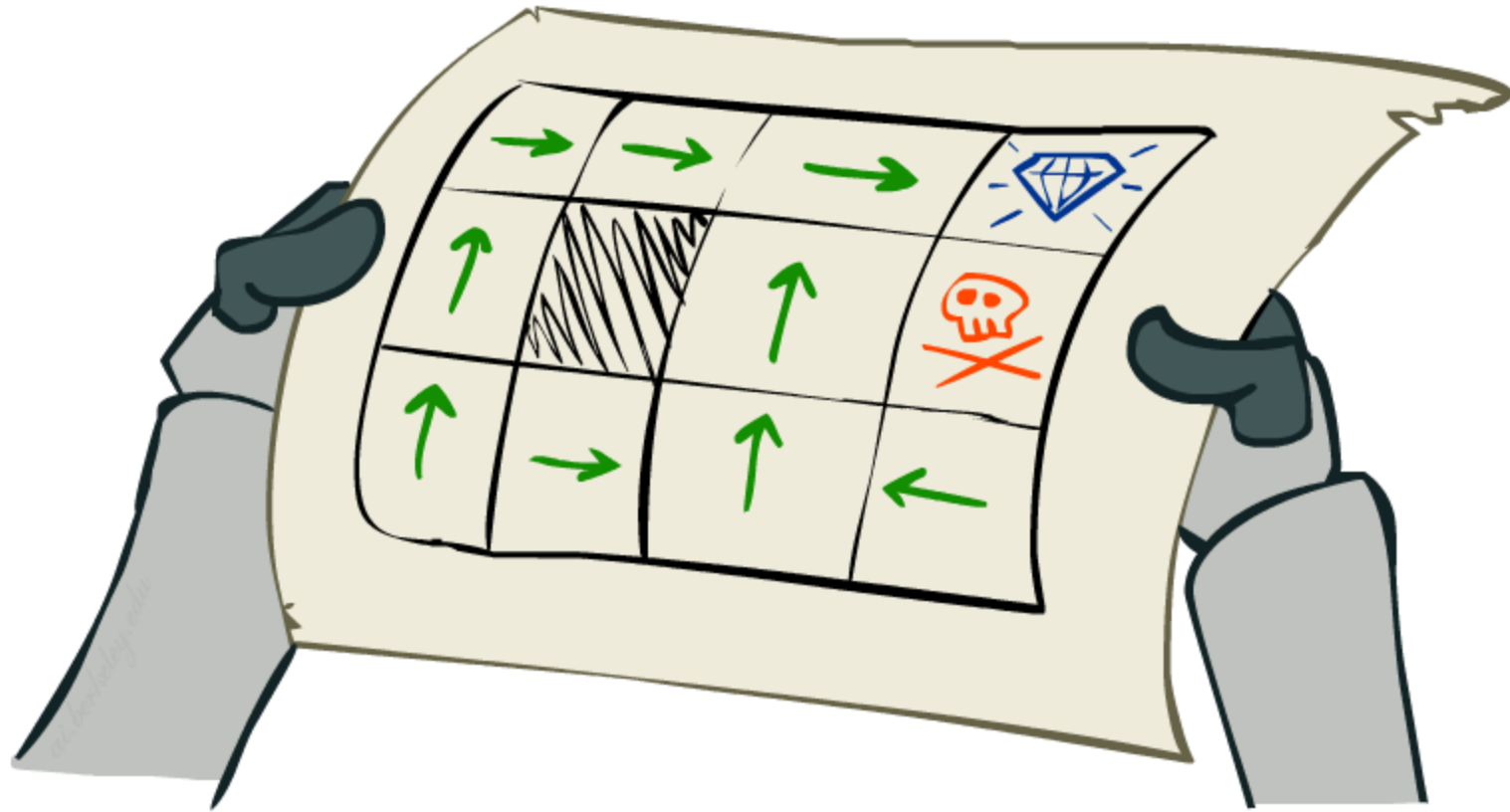
- Sooner rewards probably do have higher utility than later rewards
- Also helps our algorithms converge

- Example: discount of 0.5

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

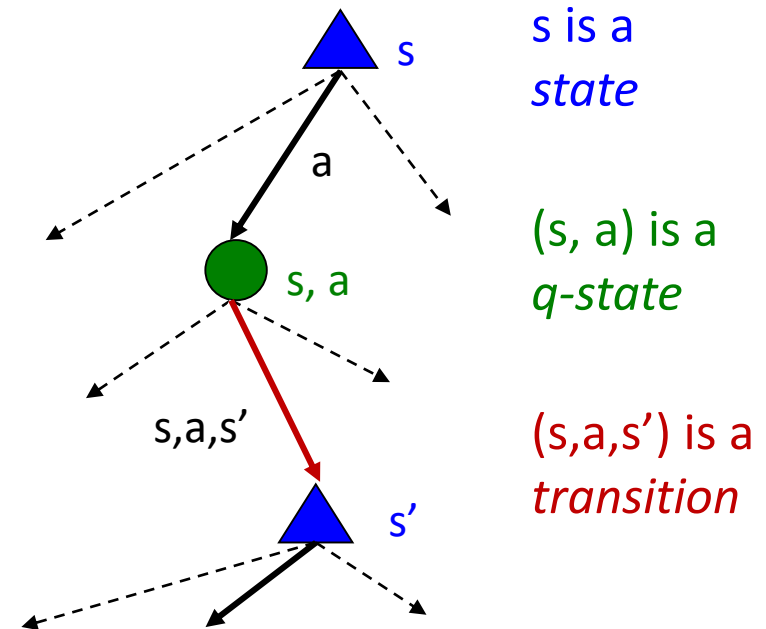


Solving MDPs

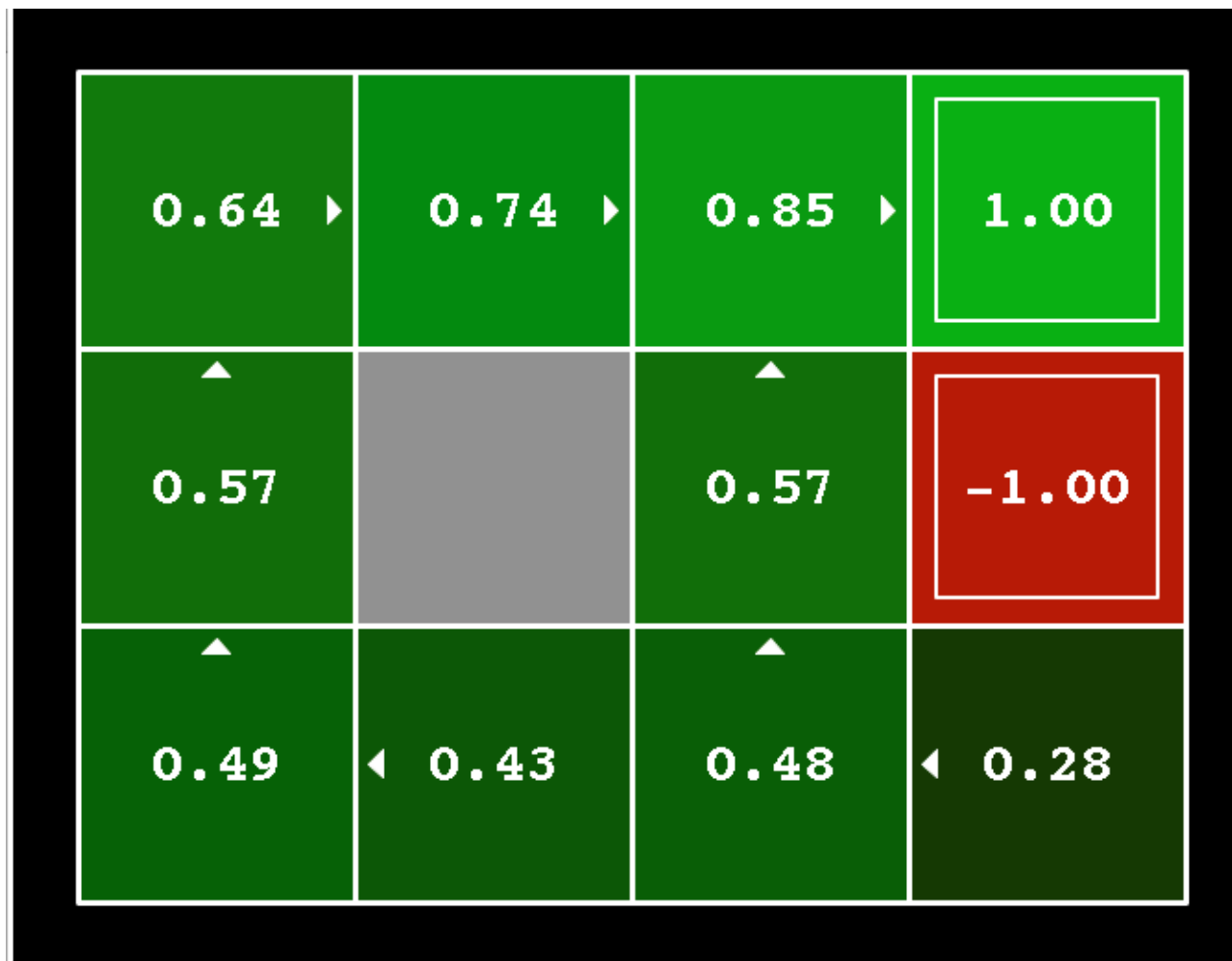


Optimal Quantities

- 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

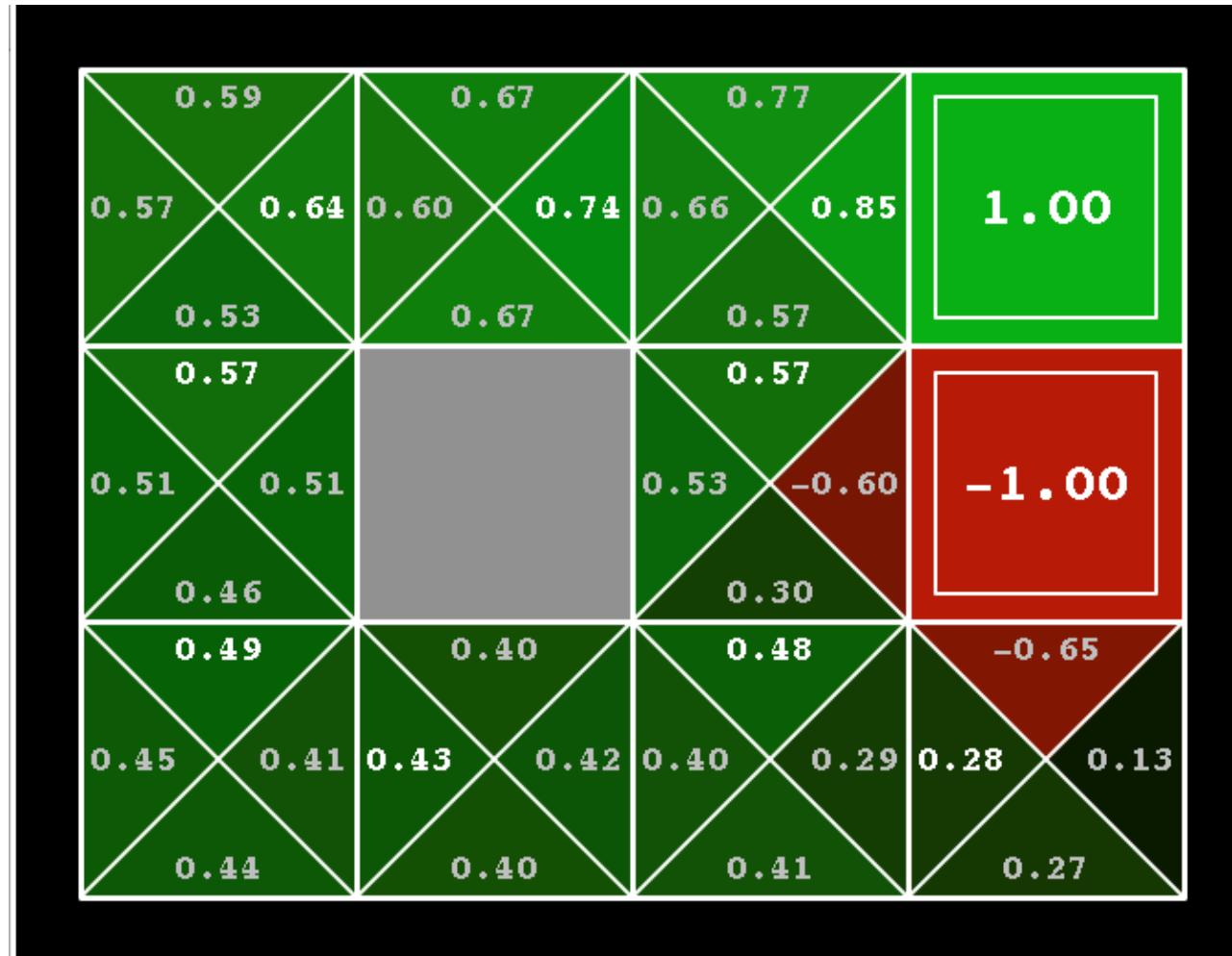


Gridworld V Values



Noise = 0.2
Discount = 0.9
Living reward = 0

Gridworld Q Values



Noise = 0.2
Discount = 0.9
Living reward = 0

Values of States

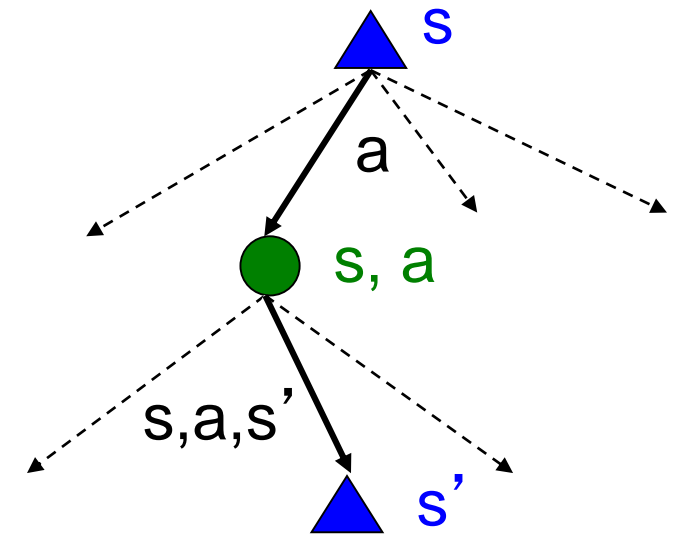
- How to compute the value of a state
 - Expected utility under optimal action
 - This is just what expectimax computed!

- Recursive definition of value:

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

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

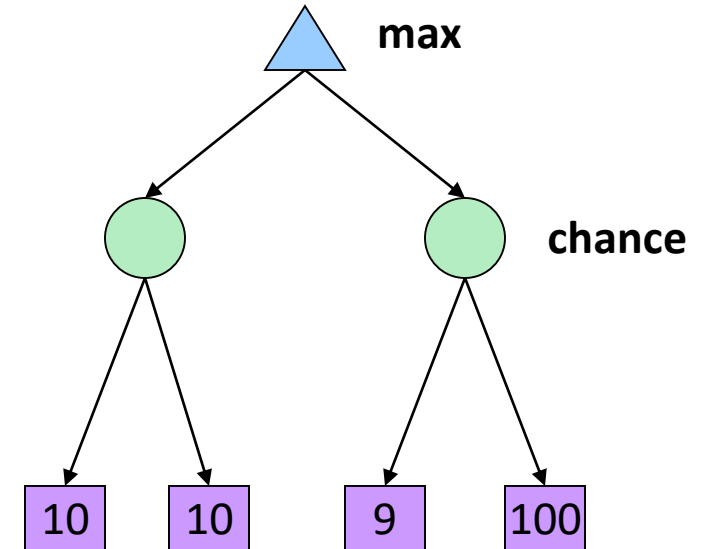
$$V^*(s) = \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')]$$



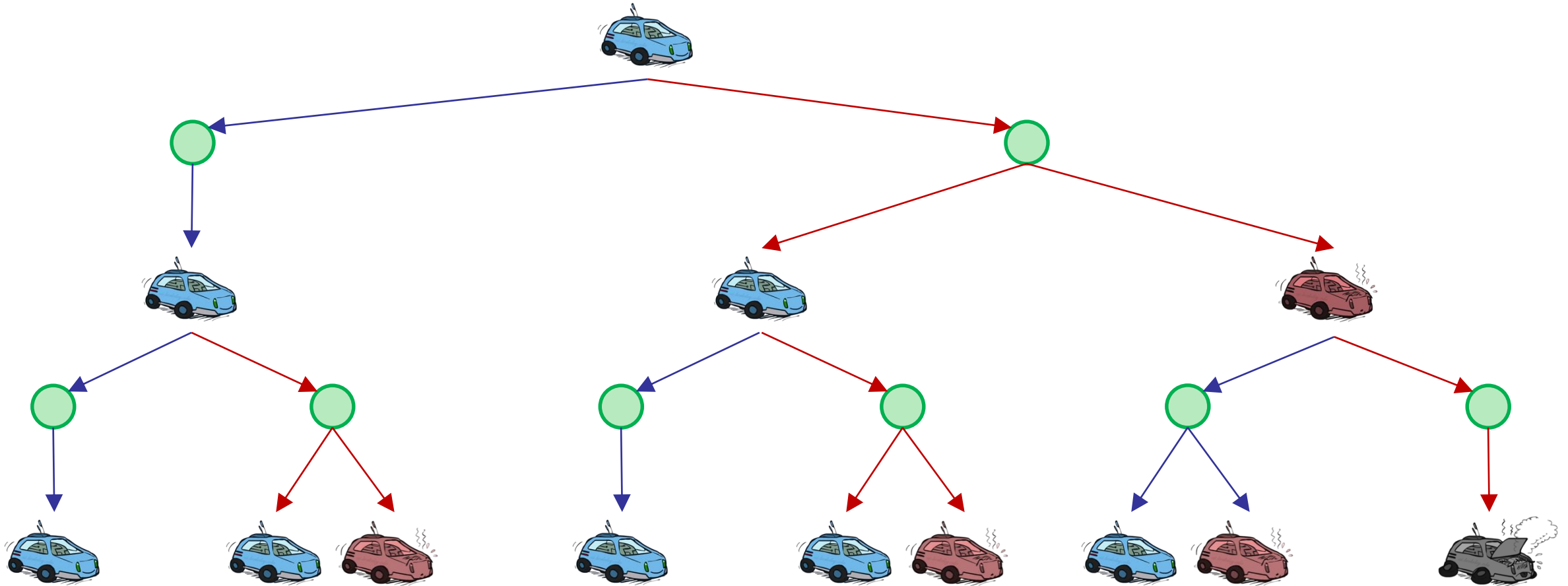
The Bellman Equation

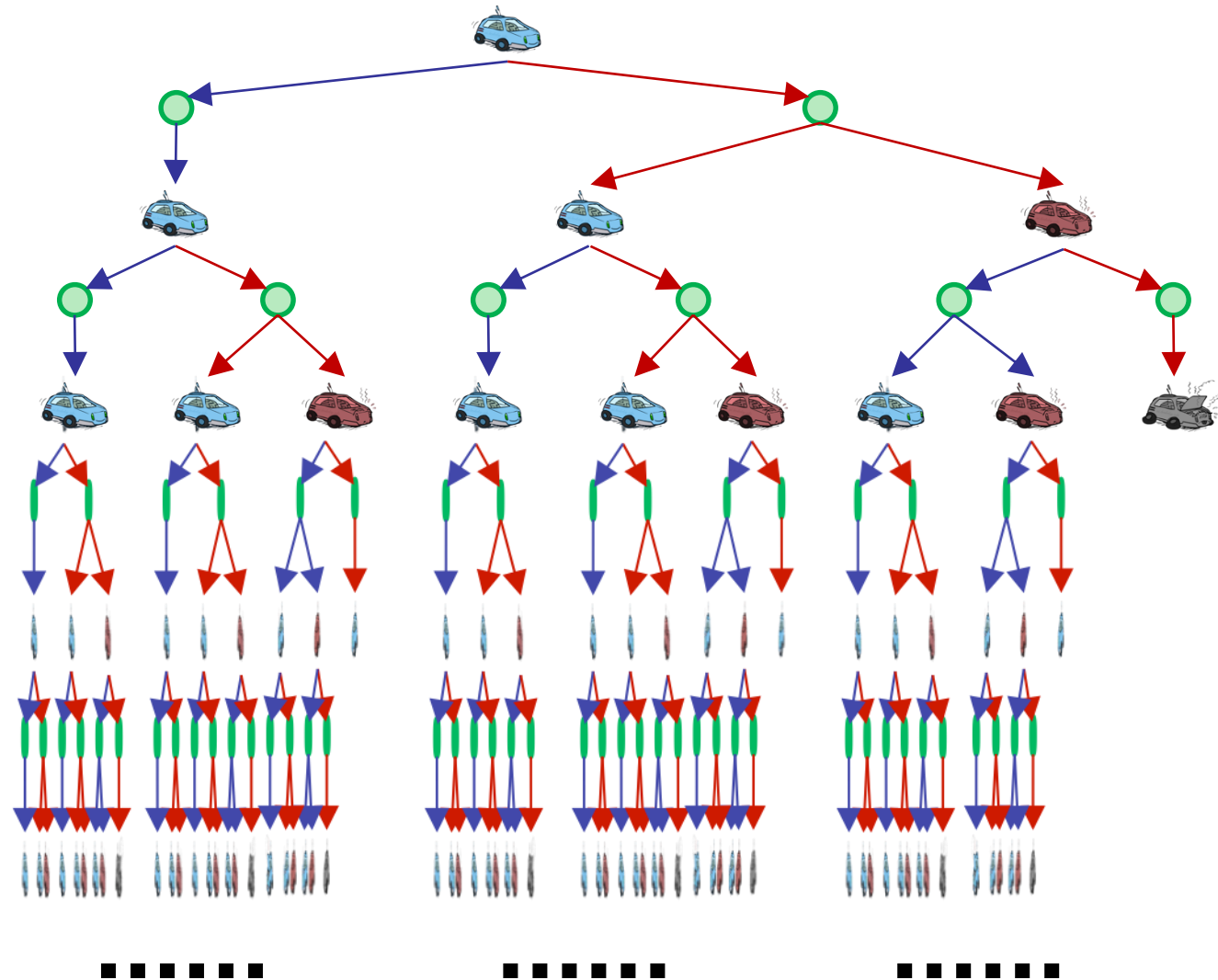
Recall: Expectimax Search

- **Expectimax search:** compute the average score under optimal play
 - Max nodes as in minimax search
 - Chance nodes are like min nodes but the outcome is uncertain
 - Calculate their **expected utilities**, i.e. taking weighted average (expectation) of children
- Later, we'll learn how to formalize the underlying uncertain-result problems as **Markov Decision Processes**



Racing Search Tree





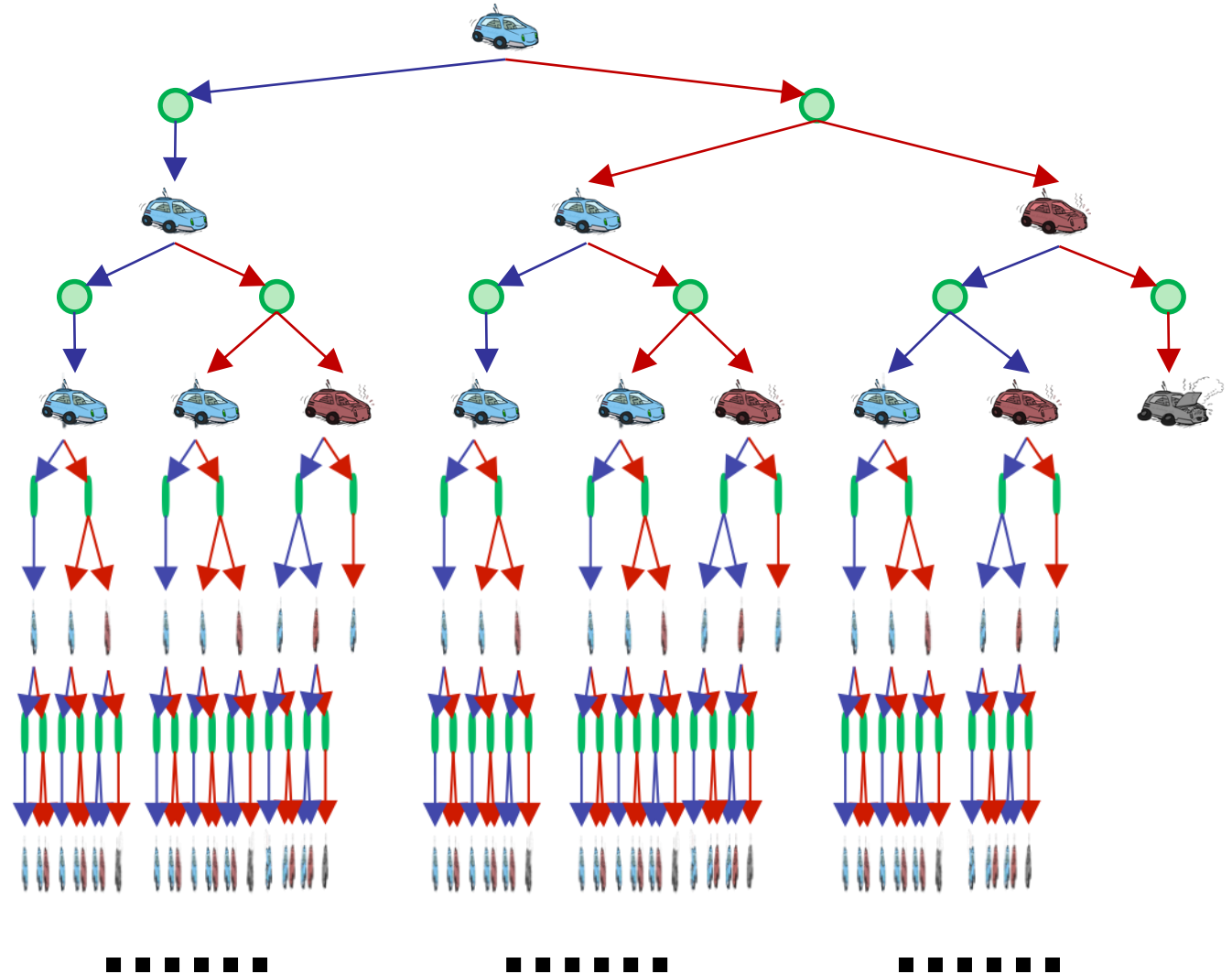
Problems with Expectimax

- Problem 1: States are repeated

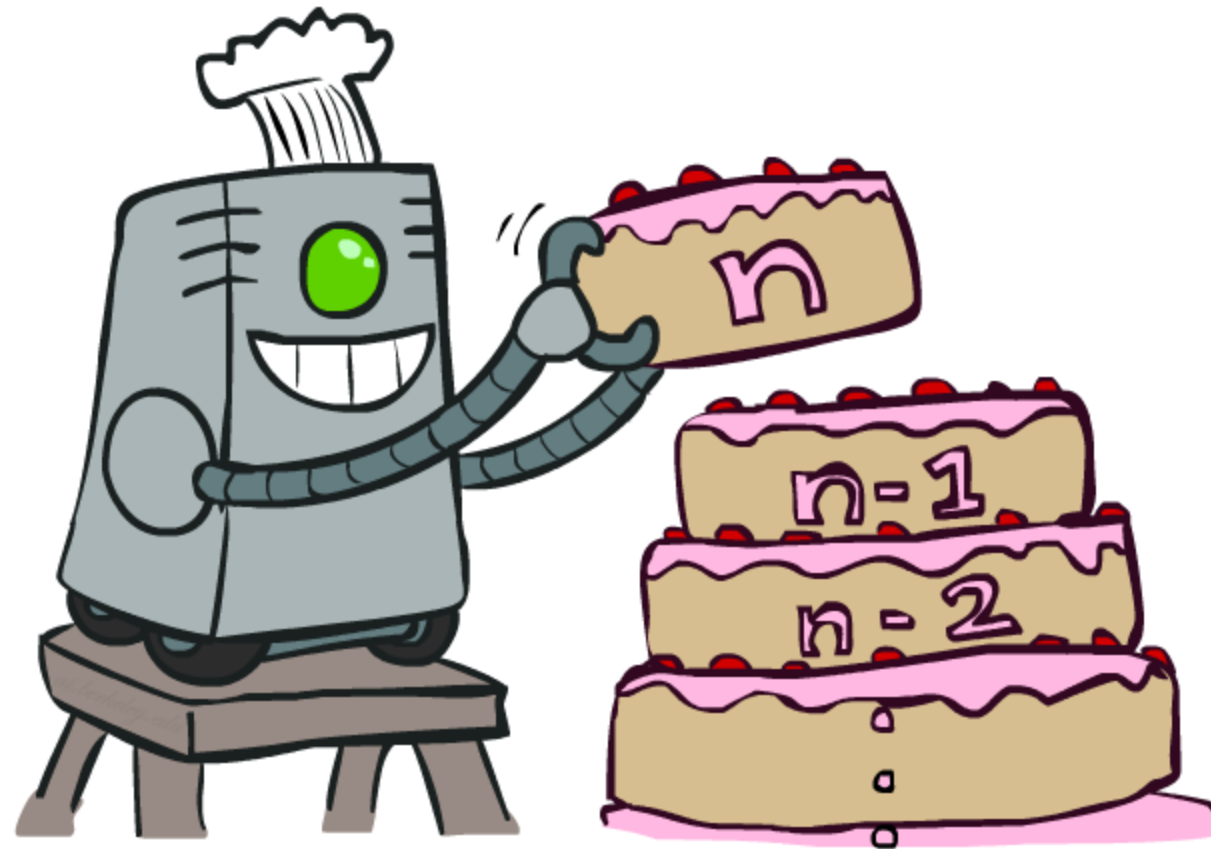
- Idea: Only compute needed quantities once

- Problem 2: 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 $\gamma < 1$

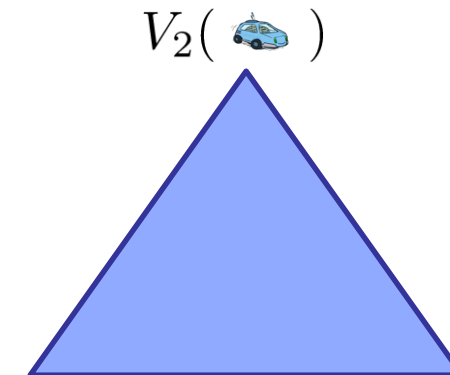
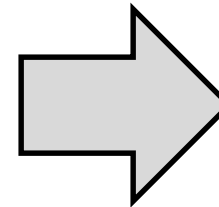
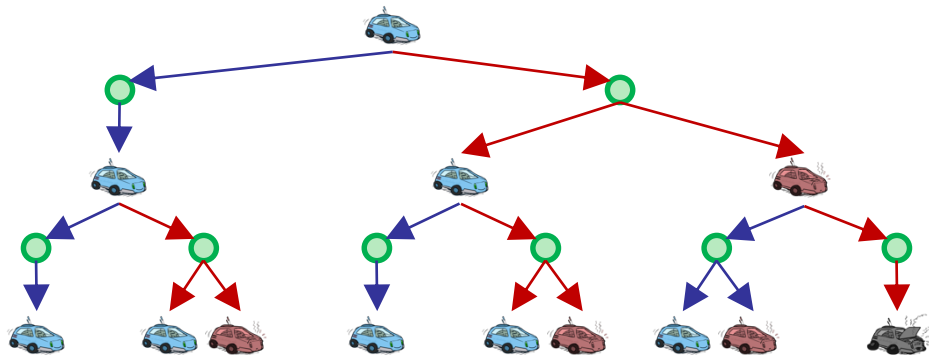
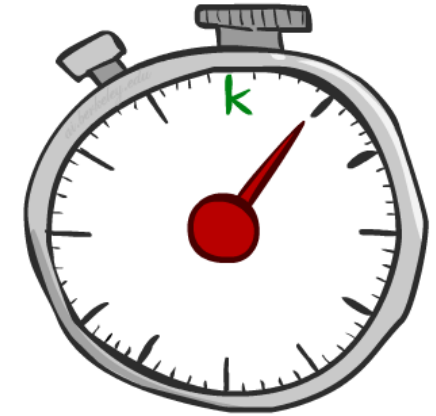


Value Iteration



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

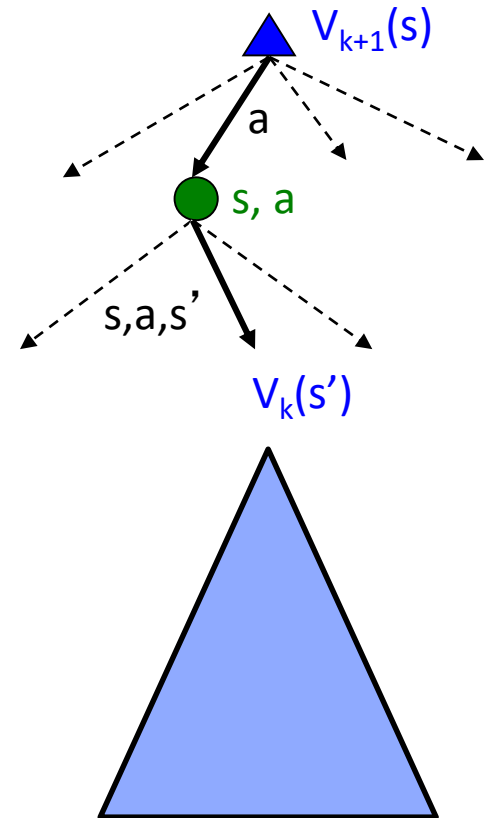


Value Iteration

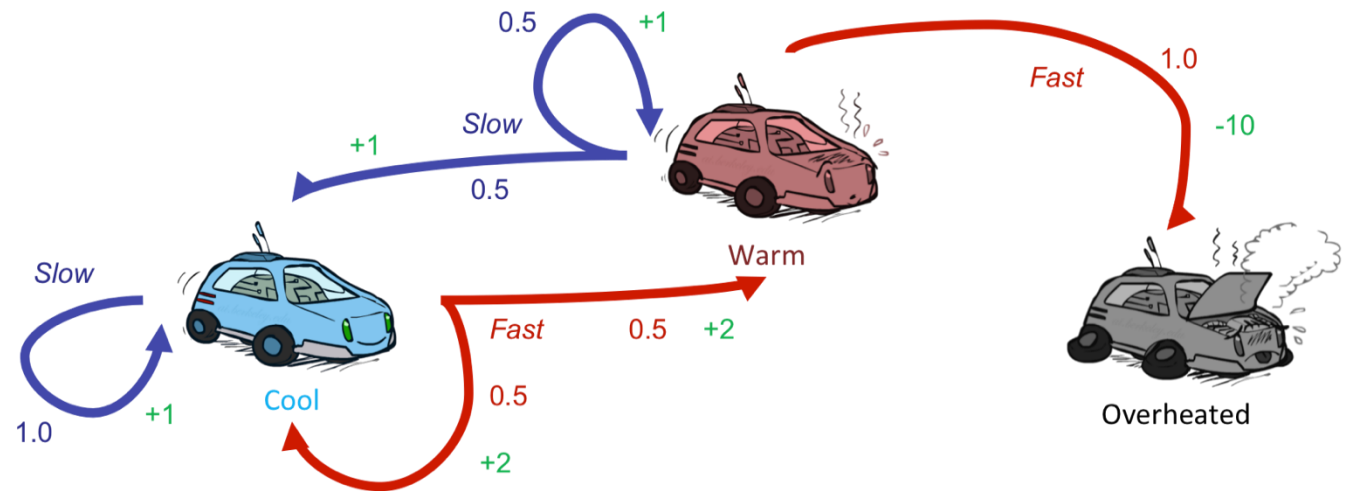
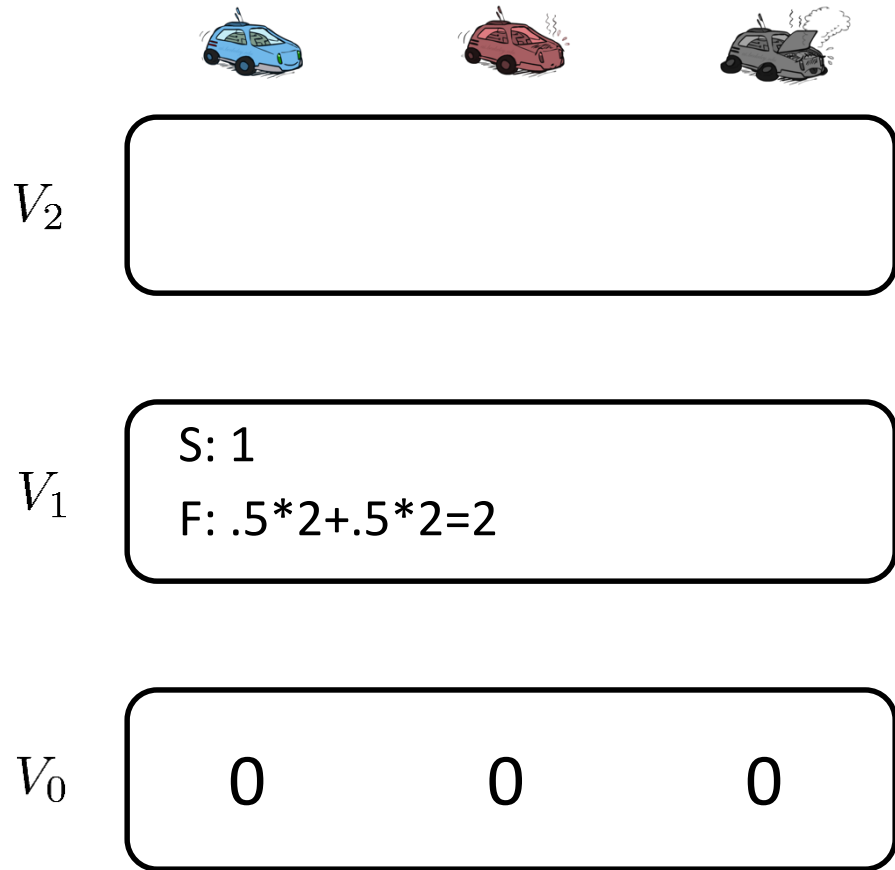
- Start with $V_0(s) = 0$: no time steps left means an expected reward sum of zero
- Given vector of $V_k(s)$ values, do one ply of expectimax from each state:

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

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



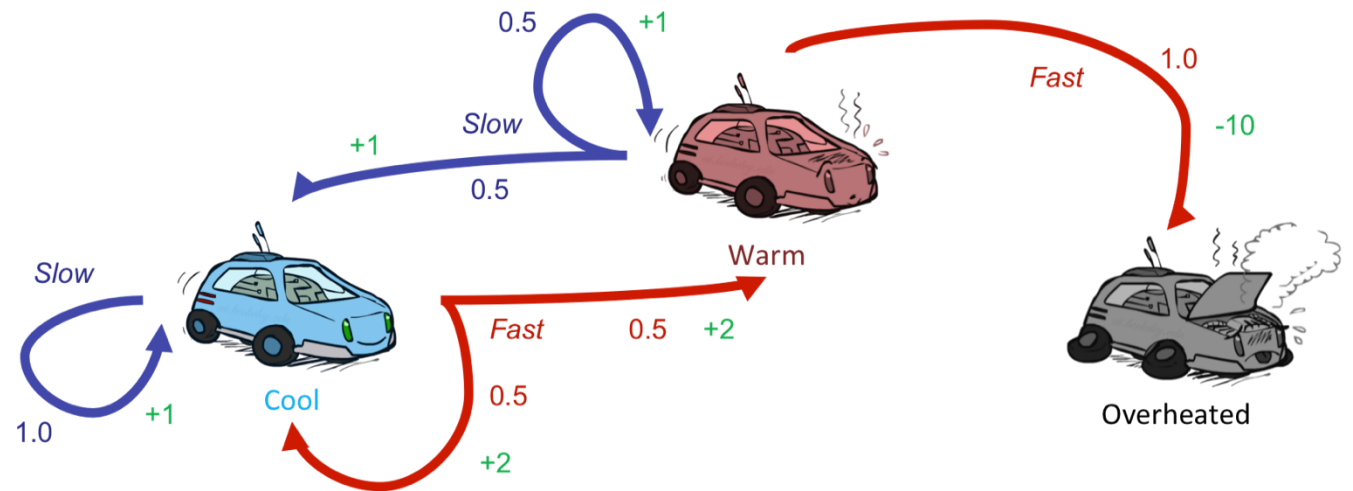
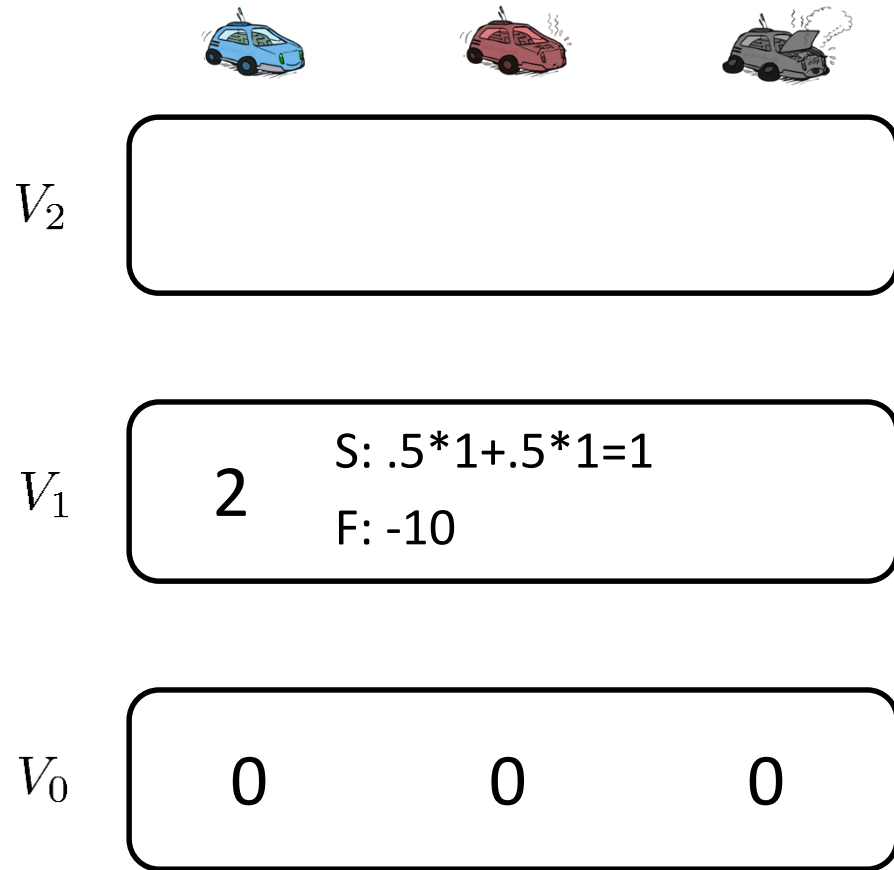
Example: Value Iteration



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

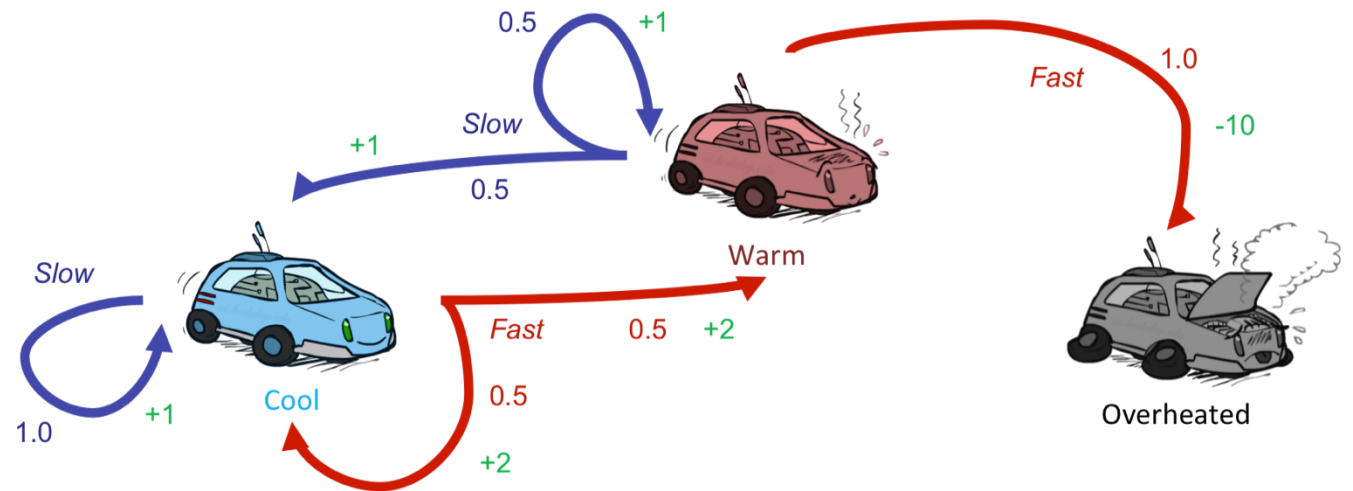


Assume no discount!

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

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

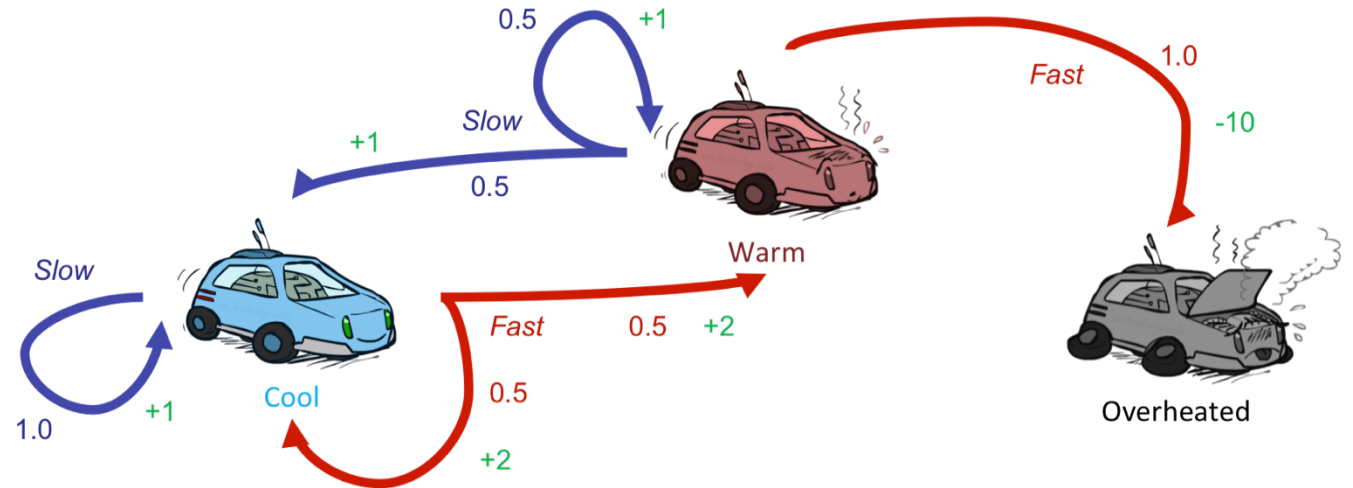
0

V_0

0

0




0

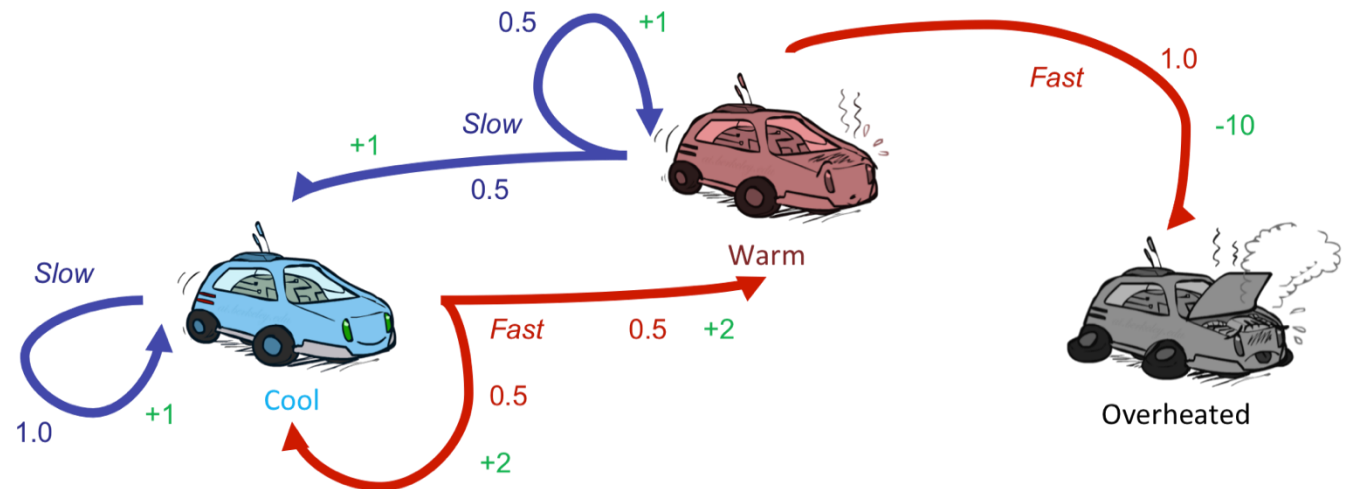


Assume no discount!

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

Example: Value Iteration




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

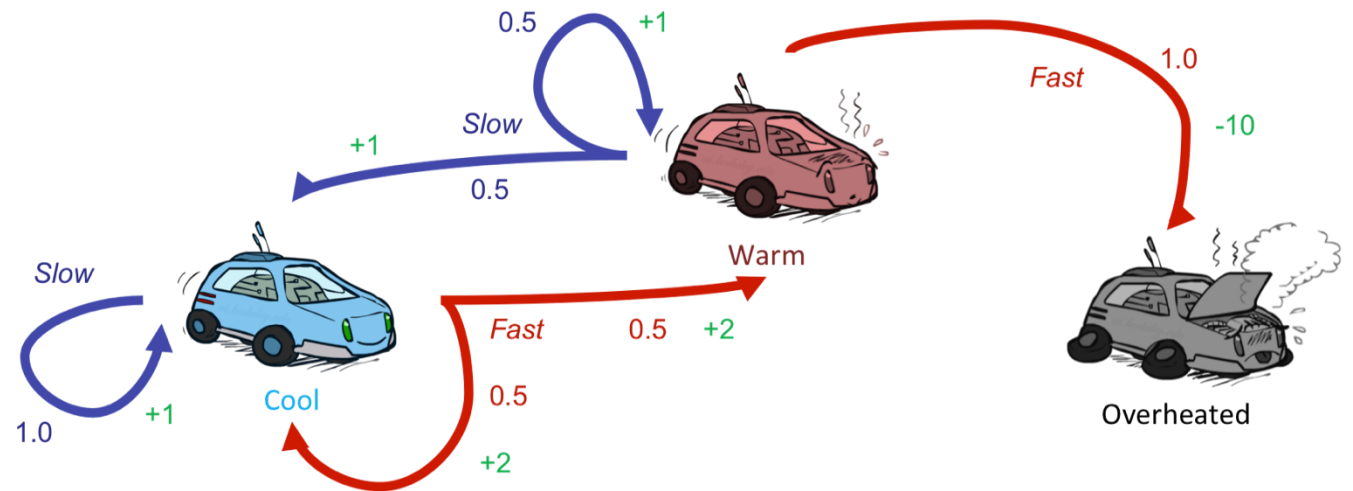


Assume no discount!

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

Example: Value Iteration

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

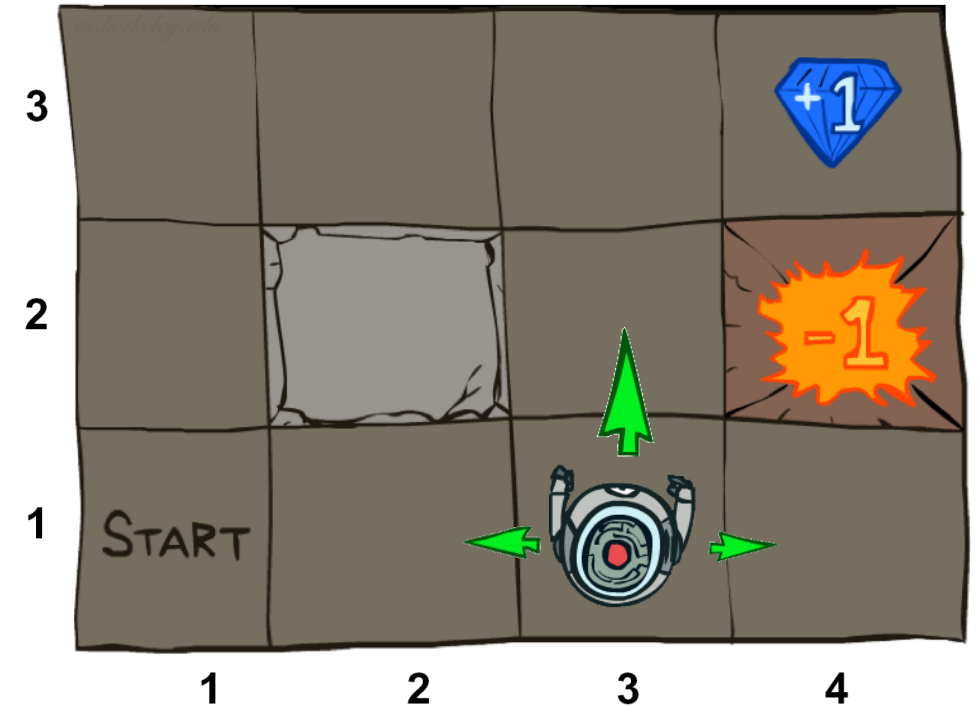


Assume no discount!

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

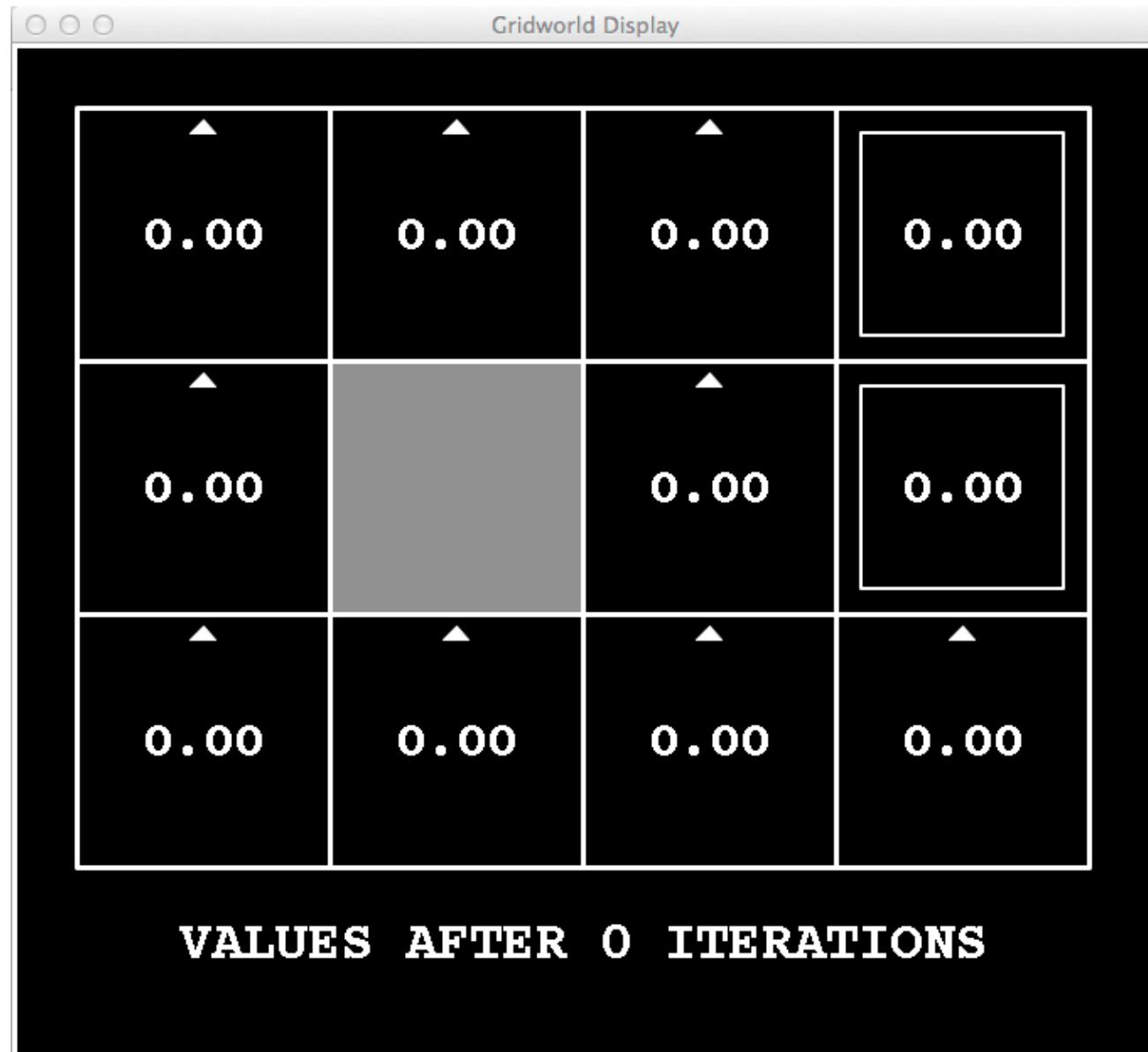
Example: Grid World

- 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)



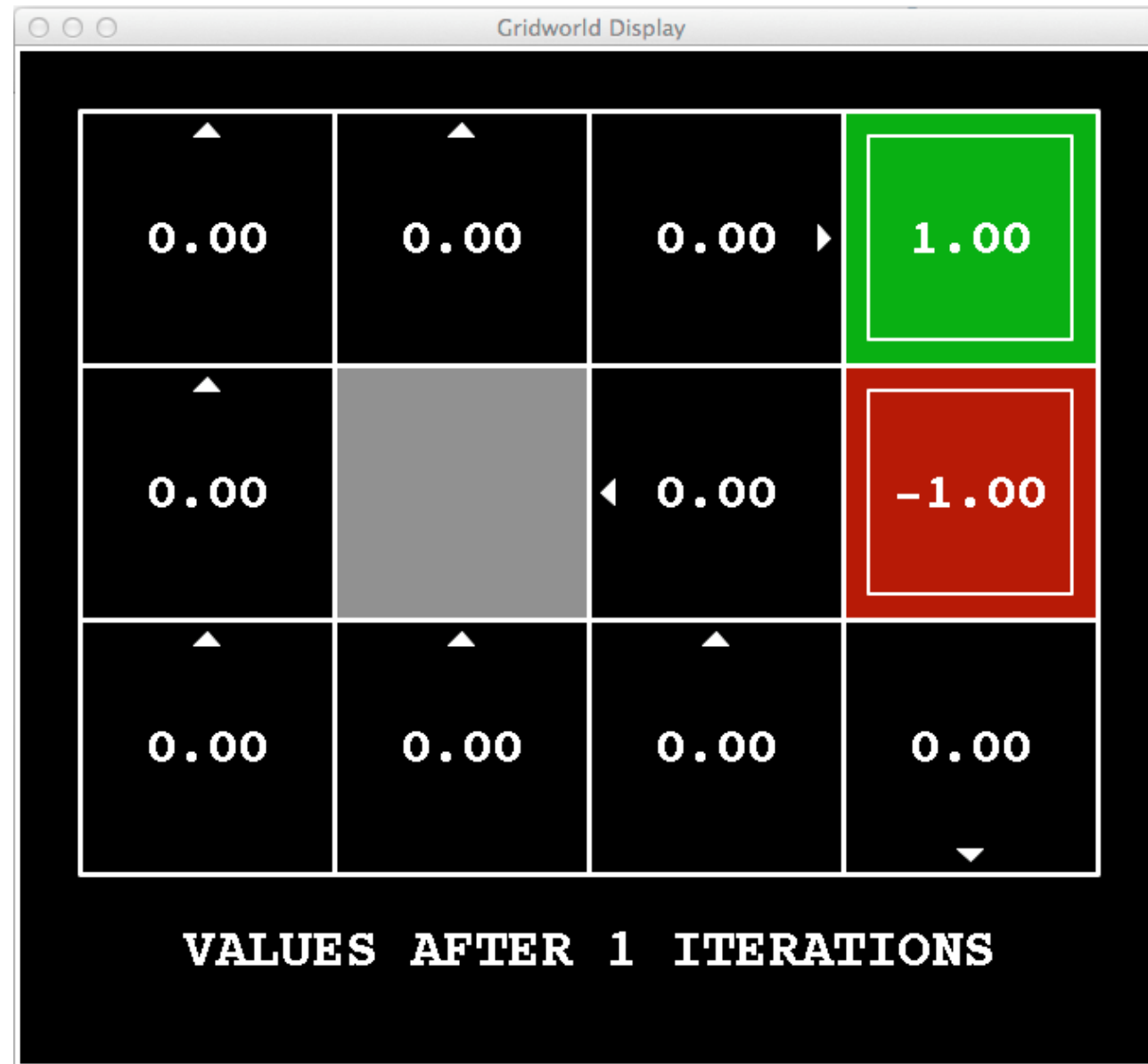
Suppose we get this reward by taking an "exit" action at a goal state

$k=0$



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



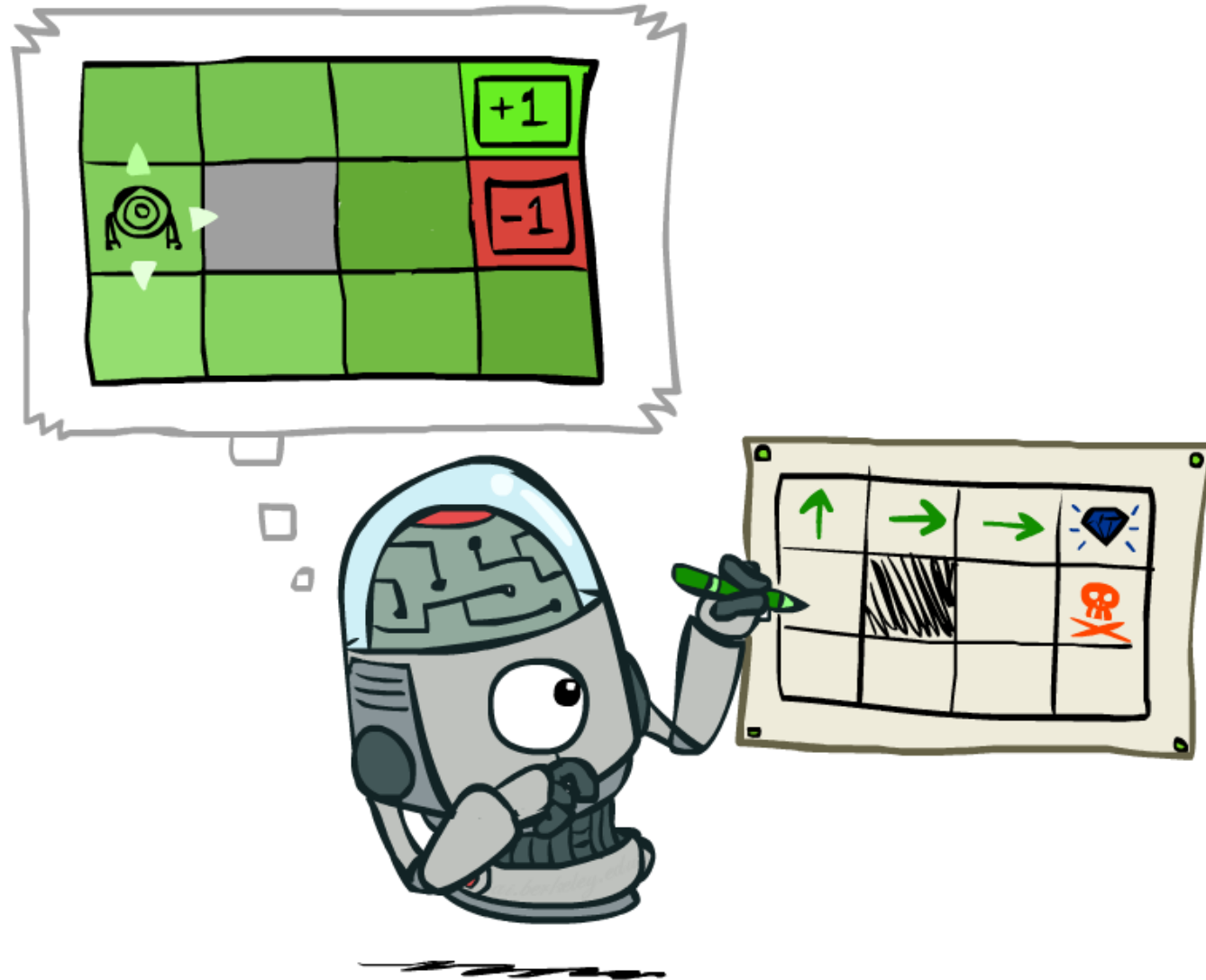
Noise = 0.2
Discount = 0.9
Living reward = 0

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)



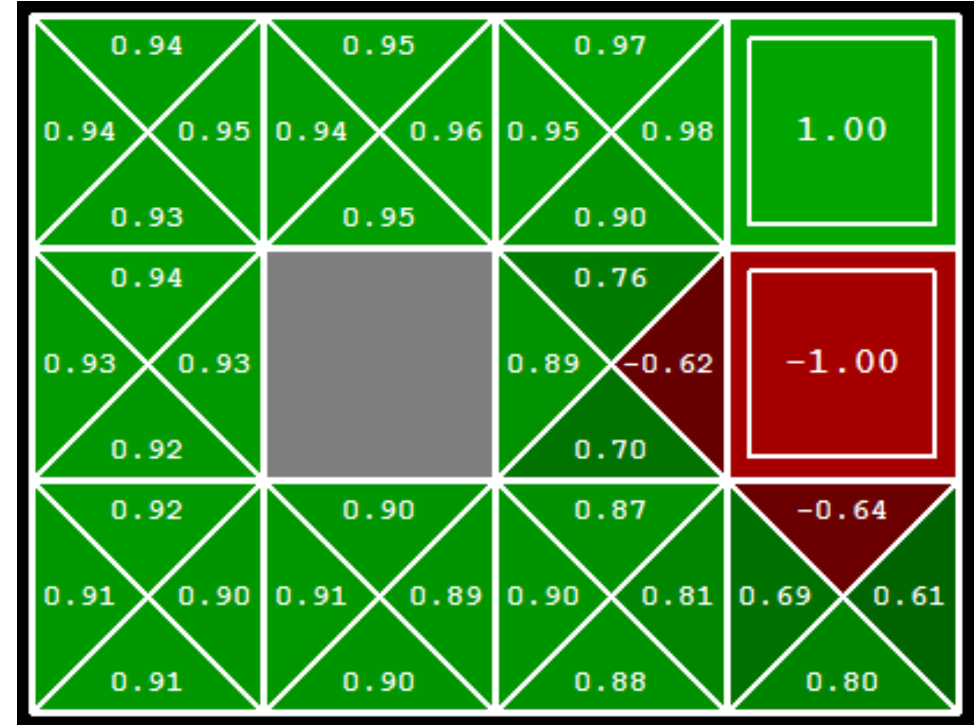
$$\pi^*(s) = \arg \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')]$$

- 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

Q-Value Iteration

- Value iteration: find successive (depth-limited) values

- Start with $V_0(s) = 0$
- Given V_k , calculate the depth $k+1$ values for all states:

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

- But Q-values are more useful, so compute them instead

- Start with $Q_0(s,a) = 0$
- Given Q_k , calculate the depth $k+1$ q-values for all q-states:

$$Q_{k+1}(s, a) \leftarrow \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma \max_{a'} Q_k(s', a')]$$